Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Konzeption und Realisierung einer Steuerungssystem-HMI für Mobilgeräte

Erik-Felix Tinsel

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. rer. nat. Stefan Wagner
Betreuer/in:	Dr. rer. nat. Asim Abdulkhaleq, M. Sc. Matthias Strljic
Beginn am:	01. Dezember 2016
Beendet am:	01. Juni 2017
CR-Nummer:	H.5.2, J.7

Kurzfassung

Diese Arbeit umfasst die Konzeption und Realisierung einer Benutzerschnittstelle zur Steuerung eines automatisierten Cocktailmixers des Instituts für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart. Dazu wird insbesondere der Stand der Technik möglicher Architekturmuster und Kommunikationsparadigmen analysiert und in einer Softwarekonzeption, den zugrundeliegenden Anforderungen entsprechend, angewendet. Ebenso werden Mechanismen und Vorgehensweise zur Schaffung von Benutzerfreundlichkeit recherchiert und berücksichtigt. Die Arbeit präsentiert eine mögliche Implementierung zur Bedienung und Verwaltung des automatisierten Cocktailmixers und begründet die angewandten Softwareentscheidungen basierend auf deren Ergebnissen.

Inhaltsverzeichnis

Akronyme

APT Advanced Package Tool. 15

CRUD Create, Read, Update, Delete. 33

CSS Cascading Style Sheets. 66

EER Enhanced entitiy-relationship. 64

GPIO General-purpose input/output. 14

GUI Graphical User Interface. 49

HDMI High-Definition Multimedia Interface. 14

HMI Human Machine Interface. 85

HTML Hypertext Markup Language. 20

HTTP Hypertext Transfer Protocol. 33

IDE Integrated development environement. 74

ISW Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen. 9

JSON JavaScript Object Notation. 33

LED Light-emitting diode. 16

MIME Multipurpose Internet Mail Extensions. 33

MVC Model-View-Controller. 25

RAM Random-access memory. 14

REST Representational State Transfer. 33

SOA Serviceorientierte Architektur. 31

SPI Serial Peripheral Interface. 15

SSH Secure Shell. 15

UML Unified Modeling Language. 54

URI Uniform Resource Identifier. 33

URL Uniform Resource Locator. 38

USB Universal Serial Bus. 14

XML Extensible Markup Language. 33

1 Einleitung

Das Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW) entwickelte gemeinsam mit einigen Studenten einen automatischen Cocktailmixer. Dieser wird bei gegebenem Anlass zu Präsentationszwecken vor Ort oder auf Messen vorgeführt. Die bisherige Entwicklung der Studenten aus den Studiengängen Mechatronik und Maschinenbau stützte sich besonders auf die Zusammenstellung der Hardware und deren Funktionalität. Diese Arbeit legt den Fokus auf die Entwicklung der Software für den Einsatz des automatischen Cocktailmixers. Einleitend befindet sich dafür die konkrete Motivation dieser Arbeit, deren Zielsetzung, sowie der Aufbau dieses Dokuments.

1.1 Motivation

Der am Institut entwickelte, automatische Cocktailmixer kann sechs Saftbehälter und acht Cocktaildispenser über eine motorisierte Schiene ansteuern, auf welcher ein zu befüllendes Glas aufgestellt wird. Die Aufträge erhält der Cocktailmixer über eine Weboberfläche, welche von den Benutzern über eine drahtlose Verbindung aufrufbar ist. Eine solche, benutzerfreundliche Webanwendung, die sich sowohl für den Bediener als auch für zukünftige Programmierer übersichtlich gestaltet, stellt einen Kernpunkt in der Konzeption und anschließenden Nutzung des Cocktailmixers dar. Eine besondere Herausforderung ist das unterschiedliche Verhalten der Oberfläche unter Berücksichtigung verschiedener Benutzerrollen, sowie der Kompatibilität mit mobilen Endgeräten, welche eine angepasste Darstellung verlangen. Für eine möglichst variierbare Architektur der Softwarekomponenten ist zudem insbesondere die Modularisierung ein unabdingbares Paradigma.
Die bestehende Softwareumsetzung (vgl. Abbildung 1.1) des Cocktailmixers erfüllt diese und weitere Anforderungen an das System nicht ausreichend oder gar nicht. Deswegen soll mit dieser Arbeit eine neue Benutzerschnittstelle geschaffen werden, welche möglichst viele funktionale und nichtfunktionale Anforderungen an das bestehende System abdeckt.

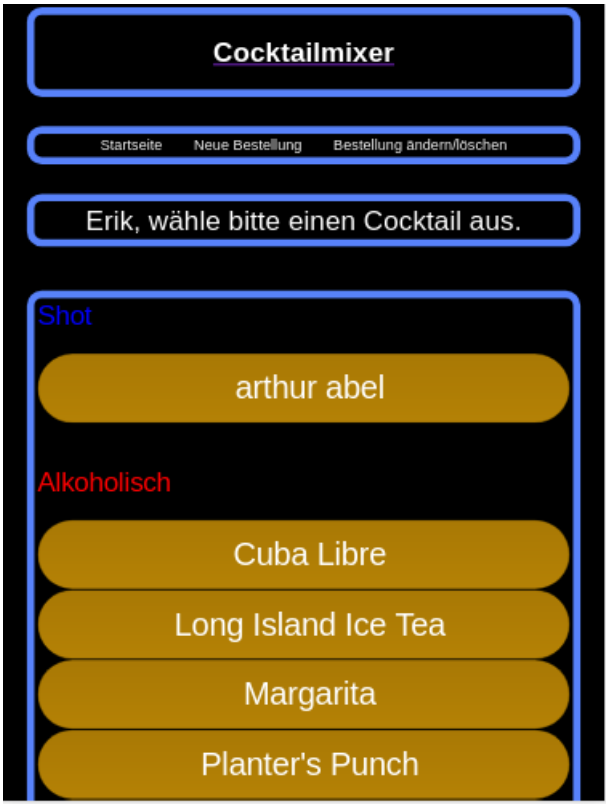

Abbildung 1.1: Ursprüngliche grafische Oberfläche des Cocktailmixers

1.2 Zielsetzung

Ziel dieser Arbeit ist es, die Benutzerschnittstelle des Cocktailmixers zu Analysieren und daraufhin neu zu Konzipieren und Entwickeln. Die nachfolgend definierten Phasen sollen diese Zielsetzung detaillieren:

1. **Recherche über den Stand der Technik**

   Für die Erfassung des Stands der Technik sollen aktuelle, frei verfügbare Mechanismen für die Softwareentwicklung untersucht und gegenübergestellt werden, um so eine begründete Softwarearchitektur zusammenzustellen.

2. **Analyse der Anforderungen an die Umsetzung**

   Die Analyse erfordert zum einen eine Einarbeitung, als auch eine Recherche über die notwendigen Anforderungen an das System.

   - Einarbeitung in das Ist-System

     Die bestehende Architektur soll hinreichend überblickt werden, um ein Verständnis für und Erkenntnisse zu der Einbettung dieser Arbeit zu gewinnen.

   - Bedürfnisanalyse

     Das bestehende System soll so analysiert werden, dass die Ziele dieser Arbeit ersichtlich werden. Dazu gehört es, die Bedürfnisse aller Akteure an das System zu erfassen. Je mehr Anforderungen dabei abgedeckt werden, umso vollständiger lässt sich schließlich ein Entwurf des Softwaresystems anfertigen, der auf diese Anforderungen in seiner Zielsetzung direkten Bezug nimmt.

3. **Ausarbeitung eines Softwareentwurfs**

   Ein Softwareentwurf soll erstellt und begründet werden.

   - Erarbeiten des Entwurfs

     Auf Grundlage der Bedürfnisanalyse und des Stands der Technik soll ein Entwurf angefertigt werden. Dieser beinhaltet wesentliche Strukturmerkmale der zu entwickelnden Software und dokumentiert die Architektur, um eine spätere Implementierung zu erleichtern und zukünftige Wartungs- oder Erweiterungsarbeiten zu vereinfachen.

   - Begründung der Designentscheidungen und der gewählten Entwicklungsplattform

     Die gewählten Entwicklungswerkzeuge und Architekturentscheidungen sollen nach einer Analyse der Alternativen begründet werden.

4. **Implementierung des Entwurfs**

   Das Softwaresystem soll auf Basis des erstellten Entwurfs erstellt werden.

5. **Validierung des Resultates gegen die Anforderungen**

   Die Ergebnisse der Implementierung sollen geprüft und gegebenenfalls korrigiert werden. Insbesondere die in der Bedürfnisanalyse festgelegten Anforderungen sollten sich in der Prüfung wiederfinden.

1.3 Aufbau der Arbeit

Diese Arbeit ist nach den Phasen der Zielsetzung aufgebaut. Zu Beginn befindet sich eine Einführung in die Grundlagen des Cocktailmixers und den für diese Arbeit benötigten Stand der Technik in Kapitel 2. Im Anschluss daran wird das zu entwickelnde System durch Anforderungen und Vergleiche, wie auch einer Abgrenzung in Kapitel 3 spezifiziert. Mit Hilfe dieser Erkenntnisse wurde der Entwurf des Systems und dessen Prototyp entwickelt. Er befindet sich in Kapitel 4. Kapitel 5 präsentiert die Ergebnisse der Entwicklung dieses Entwurfs. In diesem Kapitel werden zudem die Anforderungen anhand der Realisierung validiert. Zum Schluss wird in Kapitel 6 eine Zusammenfassung gegeben und die Möglichkeiten weiteren Vorgehens erläutert.

2 Grundlagen und Technik

Um ein Verständnis für die in dieser Arbeit behandelten Themen zu vermitteln, verschafft das Kapitel Grundlagen und Technik einen Überblick auf das bestehende System des automatisierten Cocktailmixers. Dabei wird vertiefend auf die eingesetzte Hardware, den Raspberry Pi, die Steuerungssysteme, als auch auf die bereits entwickelte Software eingegangen. Zusätzlich werden die Ergebnisse der Recherche des aktuellen Stands der Technik im Bereich der Softwarearchitektur, den Kommunikationsparadigmen und der Benutzerfreundlichkeit im Softwaredesign aufgeführt.
Das Grundgerüst des automatisierten Cocktailmixers besteht aus Bosch-Profilen. An diesen sind Dispenser für die alkoholischen Getränke, Saftbehälter und die Rückwand, welche die Hardware beinhaltet, verbaut. Am Boden des Cocktailmixers befinden sich zwei Führungsschienen auf denen ein Schlitten mit einer Druckplatte von einem Schrittmotor an die Position der Getränke bewegt wird. Ein ebenfalls am Schlitten angebrachter Hebelarm, der mit einem Getriebemotor betrieben wird, löst die Dispenser aus; die Saftbehälter werden über Magnetventile geöffnet und geschlossen. Auf der linken Seite ist ein Display an einem Profil angebracht. Dieses zeigt dem Benutzer die Weboberfläche des Cocktailmixers. (vgl. Abbildung 2.1)

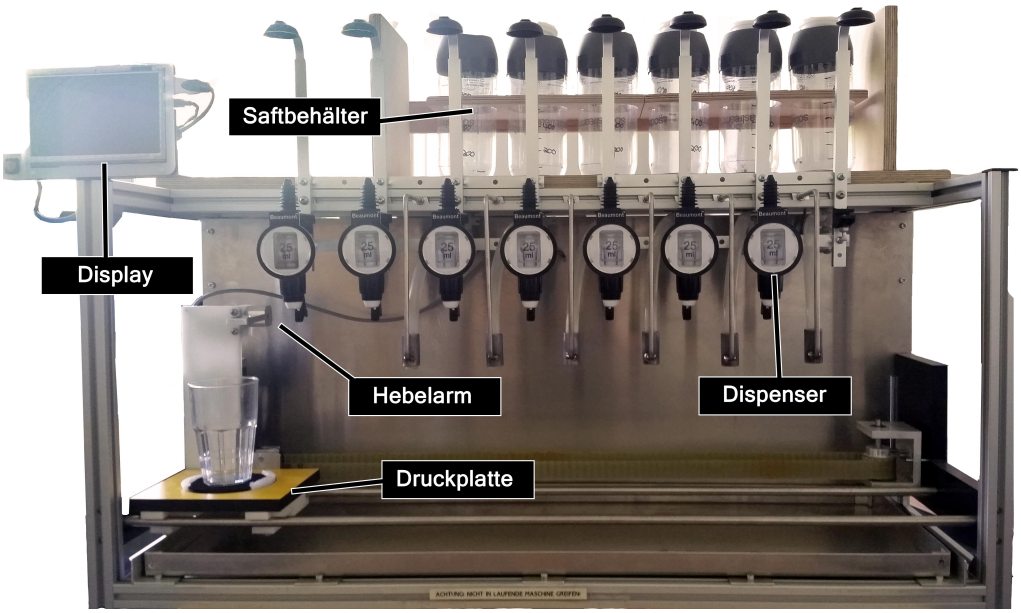

Abbildung 2.1: Vorderansicht des Cocktailmixers

2.1 Raspberry Pi

Um die Anzeige auf einem Touchscreen und die verarbeitende Logik bereitzustellen, kommt innerhalb des automatisierten Cocktailmixers ein *Raspberry Pi 3 Modell B* zum Einsatz. Dieser ist mit einer Taktfrequenz von 4x1200 Megahertz und 1024 Megabyte Random-access memory (RAM) eine leistungsstarke Grundlage zum Betreiben des Betriebssystems. Zu seinen für den Cocktailmixer und dessen Entwicklung relevanten Eigenschaften gehören 4 Universal Serial Bus (USB)-Ports (vgl. Abbildung 2.2, Nummber 1), 26 General-purpose input/output (GPIO)-Pins (vgl. Abbildung 2.2, Nummer 2), ein High-Definition Multimedia Interface (HDMI)-Port (vgl. Abbildung 2.2, Nummer 3), ein microSD-Kartenleser (vgl. Abbildung 2.2 auf der Rückseite) und ein 802.11n drahtloses Netzwerk (vgl. Abbildung 2.2, Nummer 4). Zudem lässt sich der *Raspberry Pi* mit einer Größe von 85,6mm × 56mm platzsparend an den Cocktailmixer anbringen. [Rasb]

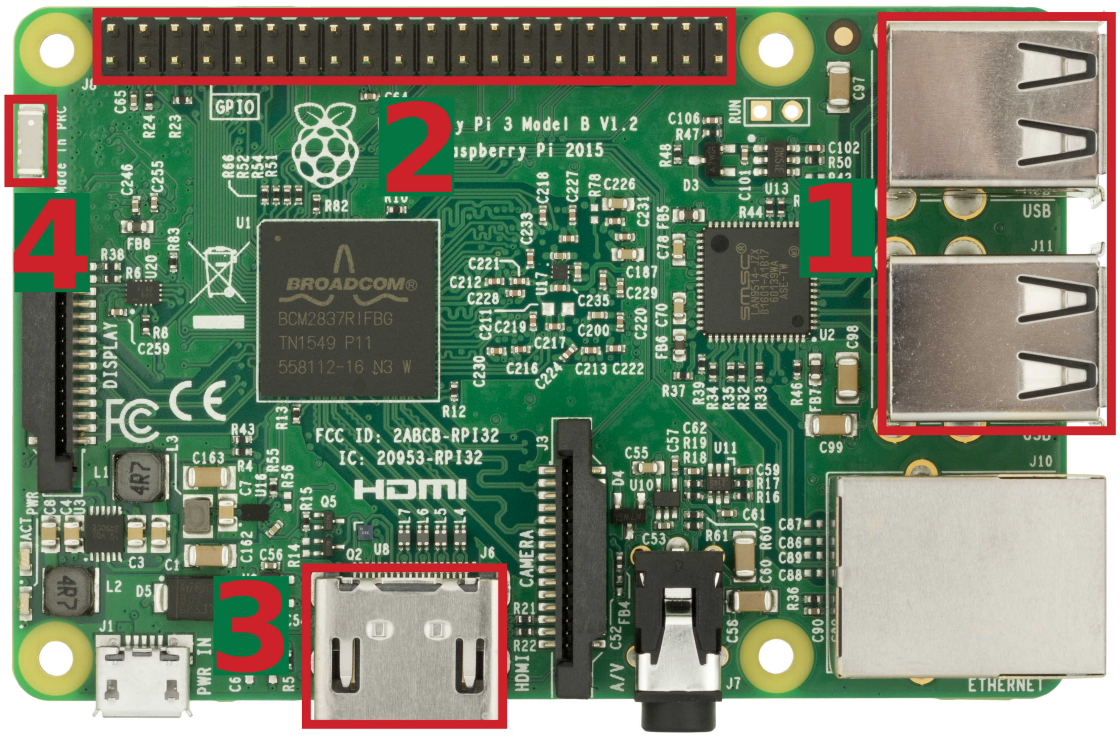

Abbildung 2.2: Draufsicht Raspberry Pi 3 Modell B

Eine Besonderheit des *Raspberry Pi* sind die GPIO-Pins. Diese Pins können programmiert werden, um mit der Umgebung zu interagieren. Dabei kann die Schnittstelle beispielsweise ein Signal eines Sensors oder Daten eines anderen Computers sowie eines elektrischen Geräts entgegennehmen. Da die maximal anliegende elektrische Leistung an den GPIO-Pins nicht zu hoch sein darf, kann an ihnen eine Erweiterung angeschlossen werden, mit der auch leistungsstarke Motoren zu Betreiben sind. [Rasa]
Auf der in den *Raspberry Pi* eingesetzten microSD-Karte ist das Open Source Betriebssystem *Raspbian* [Rasc] installiert. Dieses basiert auf *Debian* [Deb] und ermöglicht mit seiner integrierten Benutzeroberfläche ein erleichtertes Einrichten der benötigten Software, um den Cocktailmixer zu betreiben. Mit dem Paketmanager Advanced Package Tool (APT) können aus einer großen Paketliste benötigte Anwendungen anhand weniger Eingaben auf dem Betriebssystem installiert werden. Um die Arbeitsweise von APT zu verdeutlichen, zeigt Listing 2.1 exemplarisch drei Befehle, die eine aktuelle Paketliste anfordern (Zeile 1), vorhandene Anwendungen, welche von APT verwaltet werden und neue Versionen bereitstellen, erneuert (Zeile 2) und anschließend das Paket *OpenJDK 8* auf dem Betriebssystem installiert (Zeile 3).

Listing 2.1: APT Code-Beispiele

```
1 sudo apt-get update
2 sudo apt-get upgrade
3 sudo apt-get install openjdk-8-jdk
```

Über ein im System integriertes Konfigurationsmenü lässt sich der *Raspberry Pi* einrichten. Dazu gehört die Benutzerverwaltung, die (De)Aktivierung einzelner Schnittstellen wie der Secure Shell (SSH) oder dem Serial Peripheral Interface (SPI), Übertaktungsoptionen und der Konfiguration der Lokalisierung.
Der *Raspberry Pi* kann ein drahtloses Netzwerk aufbauen mit dem sich Benutzer bis in einige Meter Entfernung authentifizieren und anschließend verbinden können. Einmal verbunden lassen sich lokal bereitgestellte Webseiten aufrufen, über welche schließlich eine Interaktion mit der Maschine möglich wird.
An den HDMI-Port lässt sich ein Monitor anschließen, um auf das Betriebssystem zuzugreifen und dieses zu konfigurieren. Für den Arbeitsbetrieb mit dem Cocktailmixer ist ein Touchscreen angeschlossen. Über diesen kann, als Alternative oder Ergänzung zu einem drahtlosen Netzwerk, direkt mit dem System interagiert werden.

2.2 Steuerungssysteme

Außer dem *Raspberry Pi* kommen noch weitere Hardwarekomponenten innerhalb des Cocktailmixer-Systems zum Einsatz. Da diese Arbeit den Schwerpunkt auf die Software des Cocktailmixers legt, wird die eingesetzte Hardware an dieser Stelle nur kurz beschrieben. Der Fokus dieses Abschnitts liegt auf der Realisierung des Maschinencodes, der für die Steuerung der Hardware auf den *Raspberry Pi* programmiert wurde.
Wie bereits in Abschnitt 2.1 beschrieben, ist es möglich an die GPIO-Pins des *Raspberry Pi Modell 3 B* eine Erweiterung anzuschließen. Dies ist vor allem dann notwendig, wenn leistungsstarke Komponenten betrieben werden müssen. Da der automatisierte Cocktailmixer unter anderem zwei Motoren steuern soll, wurde eine solche Erweiterung installiert:
Das *Gertboard* besitzt unterschiedliche Funktionsblöcke, die miteinander verbunden werden können. Dabei handelt es sich um

- 12 gepufferte Ein- und Ausgänge für die Interaktion mit anderen Geräten
- 3 Druckschalter zum Auslösen von Eingangssignalen
- 6 Open-Kollektor-Treiber (50V, 0,5A) zum Betreiben von Lampen oder Relais mit hoher Leistung
- 18V,2A Motorsteuerung mit Geschwindigkeits- und Richtungskontrolle
- 28-Pin dual-in-line ATmega-Microcontroller zum Programmieren von Funktionalität
- 2-Kanal 8,10 oder 12 Bit digital- zu analog- Konverter zum Wandeln digitaler Signale in analoge
- 2-Kanal 10 Bit analog- zu digital- Konverter zum Wandeln analoger Signale in digitale

[LV12]

Mit Hilfe des *Gertboards* werden ein Getriebemotor, ein Schrittmotor, zwei Light-emitting diodes (LEDs) und die Magnetventile angesteuert. Der Schrittmotor bewegt den Schlitten an die Position des gewünschten Dispensers oder Saftbehälters. Falls ein Dispenser angesteuert wurde, wird dieser durch den Getriebemotor ausgelöst. Die LEDs sollen dem Benutzer signalisieren, dass der Cocktailmixer den Mischvorgang startet, sobald der Start-Taster betätigt wird und ihn über den Abschluss ebenjenes Vorgangs informieren. Die Magnetventile öffnen sich für eine im Maschinencode festgelegte Zeit und geben so die Flüssigkeit in das Glas auf dem Schlitten ab.

Der Maschinencode des automatisierten Cocktailmixers wurde mit der Programmiersprache *Python 2.7* realisiert. Für *Python* existieren unterschiedliche Softwarepakete wie *RPi.GPIO* [RPi] oder *WiringPi* [Wir], um die GPIO-Pins und somit das *Gertboard* über den *Raspberry Pi* anzusprechen [LV12]. Ersteres Paket kommt in der Programmierung des Cocktailmixers zum Einsatz.
Wie in Abbildung 2.3 zu sehen, gliedert sich die Programmierung in drei Module, die in den folgenden Unterabschnitten näher beschrieben werden.

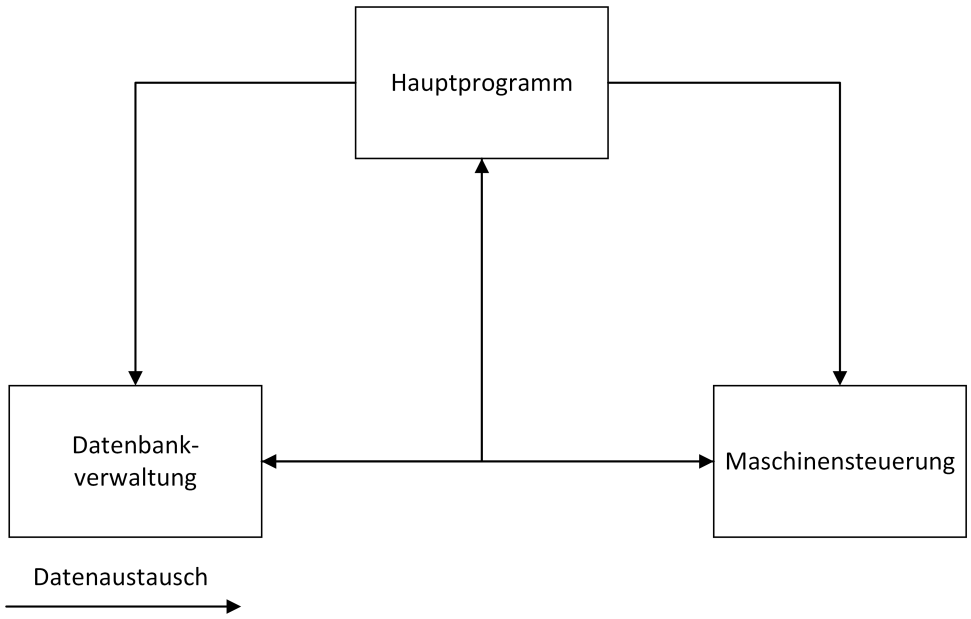

Abbildung 2.3: Struktur des Maschinencodes nach Hermann

2.2.1 Hauptprogramm

Das Hauptprogramm besteht aus drei Klassen: *Init*, *Main* und *Jobs*. *Init* ist die Klasse, die der Benutzer beim Starten der Maschinensteuerung über die Konsole aufruft. Innerhalb dieser werden die in der Datenbank hinterlegten Zustände initialisiert und die Hauptklasse *Main* aufgerufen. Von dort aus wird, wie in Abbildung 2.4 skizziert, eine Endlosschleife gestartet. In jedem Durchlauf wird dabei überprüft, ob in der Datenbank der Zustand zum Beenden des Cocktailmixers eingetragen ist. Falls dies der Fall ist, verlässt der Cocktailmixer die Schleife und trägt die Zustände entsprechend in der Datenbank ein. Findet sich kein Zustand zum Beenden in der Datenbank, weißt die Maschinensteuerung den ausgehenden GPIO-Pins ihre Spannung (LOW/HIGH) und dem eingehenden Pin einen pull-down-Widerstand zu.

Nun wird überprüft, ob ein Service-Zustand in der Datenbank hinterlegt ist und sich die Maschine derzeit im Leerlauf befindet. Sind beide Zustände erfüllt, wird der Service-Modus aktiviert - ansonsten prüft die Maschinensteuerung, ob derzeit eine Bestellung offen ist. Im Falle einer nicht bearbeiteten Bestellung in der Datenbank wird die Maschine in einen Arbeitszustand versetzt und die Klasse *Jobs* aufgerufen. Diese Klasse lädt die benötigten Schritte für die anliegende Bestellung aus der Datenbank und wartet, bis der Benutzer ein Glas auf die Druckplatte stellt und den Start-Schalter bedient. Für diesen Vorgang hat der bestellende Benutzer 30 Sekunden Zeit, danach wird die Bestellung gelöscht und die Schleife in der Klasse *Main* beginnt von vorn. Wurde der Schalter innerhalb der gegebenen Zeit ausgelöst und meldet der Druckplattensensor ein aufgestelltes Glas, werden die einzelnen Stationen abgefahren. Wurden alle Schritte ausgeführt und befindet sich der Schlitten wieder an seinem ursprünglichen Platz, wartet die Maschine, bis der Druckplattensensor eine Entnahme des Glases signalisiert. Der Cocktailmixer beginnt daraufhin die Schleife von neuem.

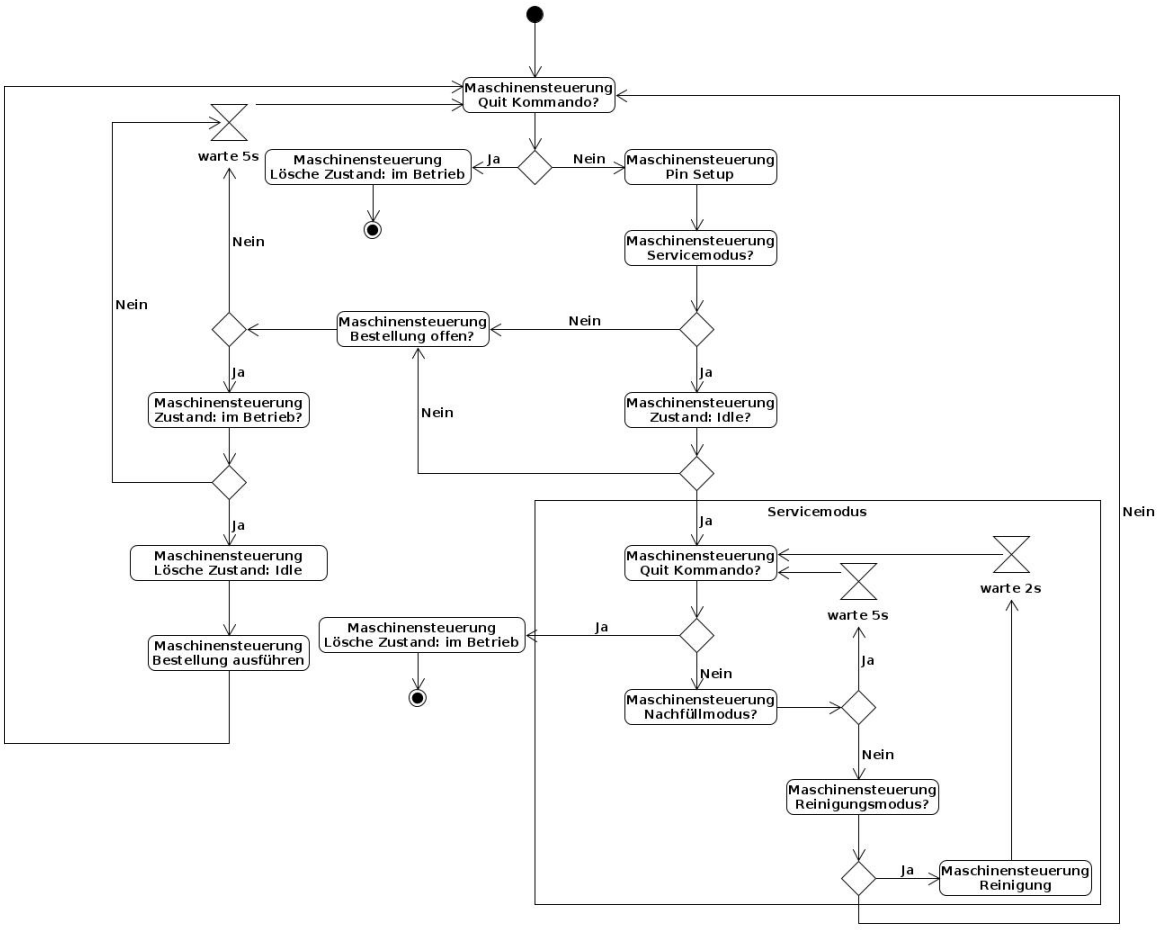

Abbildung 2.4: Ausschnitt des Aktivitätsdiagramms der Klasse *Main*

2.2.2 Datenbankverwaltung

Die Datenbankverwaltung besteht aus den Klassen *Drucksensor*, *Order*, *SQL*, *Tasks*, *Volume* und *Ventile*. Die Klasse *SQL* wird von den restlichen Klassen genutzt, um eine Verbindung mit einer lokal betriebenen *MySQL*-Datenbank herzustellen. So ruft die Klasse *Drucksensor* die Daten aus der Datenbank auf, die bestimmen, ab welchem Wert der Drucksensor ein aufgestelltes Glas signalisieren soll. Die Klasse *Order* prüft auf offene Bestellungen in der Datenbank, ruft das benötigte Rezept für eine Bestellung ab und legt abgeschlossene Bestellungen als erledigt ab. Die Klasse *Tasks* holt sämtliche Zustände aus der Datenbank und setzt diese entsprechend des Zustands in dem sich die Maschine befindet. Die Klasse *Volume* verarbeitet die aktuellen Füllstände der Saftbehälter und der alkoholischen Getränke. Die Klasse *Ventile* regelt das Öffnen und Schließen der Ventile im Reinigungsmodus.

2.2.3 Maschinensteuerung

Die Maschinensteuerung ist die Schnittstelle zwischen dem Maschinencode und der eingesetzten Hardware, wie dem Getriebemotor, dem Schrittmotor, der Druckplatte und den LEDs. Dabei wird vor allem das Softwarepaket *RPi.GPIO* eingesetzt, um über die GPIO-Pins des *Raspberry Pi* das *Gertboard* und so die einzelnen Komponenten anzusteuern. Listing 2.2 zeigt als Beispiel der Maschinensteuerung die Realisierung des Getriebemotors, der an einen Dispenser herangeführt werden soll. Die Funktion wird mit den Parametern *2000, 50* und dem Pin des Getriebemotors aufgerufen. In Zeile 7 wird die benötigte Pulsweite errechnet und in Zeile 8 die Zeitperiode, um auf den benötigten Hertz-Wert zu gelangen. Die errechneten Werte zeigten in vorherigen Experimenten mit 50% der Maximaldrehgeschwindigkeit ein sanftes Anlegen mit einem vernachlässigbar kleinen, zusätzlichen Zeitaufwand [Her16]. In der For-Schleife wird schließlich der Getriebemotor über den GPIO-Pin angesteuert.

Listing 2.2: Funktion vor_slow zur Steuerung des Getriebemotors

```
1  import RPi.GPIO as GPIO
2  import sys
3  from time import sleep
4  class Dispenser: #Aus Darstellungsgruenden leicht vereinfacht
5          Reps = 400
6          Hertz = 2000
7          Freq = (1 / float(Hertz))
8          def vor_slow(self, Reps,pulse_width_percent, port_num):
9                  pulse_width = pulse_width_percent/float(100)*self.Freq
10                 time_period = (self.Freq
11                 - (self.Freq*pulse_width_percent/float(100)))
12                 for i in range(0, Reps):
13                         GPIO.output(port_num, 0)
14                         time.sleep(pulse_width)
15                         GPIO.output(port_num, 1)
16                         time.sleep(time_period)
```

2.3 Benutzeroberfläche

Die Benutzeroberfläche des automatisierten Cocktailmixers wird über ein eigens aufgespanntes drahtloses Netzwerk bereitgestellt. Benutzer können sich an diesem authentifizieren und verbinden und daraufhin die Webseite des Cocktailmixers aufrufen. Auf dieser Webseite werden folgende Funktionen angeboten:

- Benutzername eingeben
- Getränke bestellen
- Bestellung löschen
- Bestellung ändern

Folgende Funktionen dienen der Administration der Benutzeroberfläche und benötigen somit einen restriktiven Zugang um ausgeführt zu werden:

- Ressource hinzufügen
- Ressource ändern
- Rezept hinzufügen
- Rezept ändern
- Rezept löschen
- Ressource nachfüllen
- Behälter leeren und reinigen
- Servicemodus starten und beenden

Die Programmierung der Benutzeroberfläche wurde in *PHP* realisiert. Die Hauptfunktionalität ist in der Datei *fun.inc.php* umgesetzt. Dort befinden sich alle Klassen, die für die Verwaltung der Benutzeroberfläche notwendig sind. Auch die Datenbankzugriffe werden größtenteils in diesen Klassen durchgeführt. Die restlichen Dateien beinhalten den Hypertext Markup Language (HTML)-Code der Webseite, der zum einen Teil mit Hilfe der Funktionen aus der Datei *fun.inc.php* und zum anderen Teil mit Datenbankaufrufen innerhalb der Datei selbst befüllt wird. Die beiden Benutzerrollen *Admin* und *Bediener* besitzen jeweils eigene, größtenteils redundante Codeabschnitte zur Umsetzung der Administrationsansicht.

2.4 Stand der Technik

Dieser Abschnitt schafft durch die Aufarbeitung des aktuellen Stands der Technik eine Rahmenumgebung der Entwurfsentscheidungen dieser Arbeit. Dabei werden unterschiedliche Erkenntnisse und/oder Standards aus den Bereichen der Architekturmuster, der Kommunikationsparadigmen und der Usability von Software vorgestellt.

2.4.1 Architekturmuster

Die Softwarearchitektur einer Anwendung oder eines Systems ist eine Struktur, welche ihre einzelnen Komponenten, deren extern sichtbaren Eigenschaften und die Relationen unter ihnen beinhaltet [BCK03]. Architekturmuster stellen einen Teil der Softwarearchitektur dar, indem sie den Stil der Gesamtarchitektur eines Systems beschreiben [Has06]. Im Folgenden werden sechs Architekturmuster aus dem Buch *Architektur-und Entwurfsmuster der Softwaretechnik* [GD13] von Joachim Gall und Manfred Dausmann in alphabetischer Reihenfolge beschrieben. Diese Muster werden laut den Autoren in der Praxis besonders häufig eingesetzt.

Broker

Das Architekturmuster *Broker* setzt ein System mit mehreren Clients und Servern zusammen. Dabei vermittelt eine zwischen Clients und Servern platzierte Instanz die Aufrufe an die korrekte Adresse und liefert deren Antwort zurück. Diese Instanz wird als *Broker* bezeichnet. Hintergrund dieses Architekturmusters ist der wachsende Bedarf an Rechenleistung und an Speicherplatz bei gleichzeitig wachsender Benutzerzahl und somit die Notwendigkeit eines verteilten Systems. Bei diesem Muster stellen die Server-Komponenten einen oder mehrere Dienste zur Verfügung, welche von einem Client benötigt werden. Zu erwähnen ist hierbei, dass ein Server auch in der Rolle des Clients agieren kann, um den Dienst eines anderen Servers zu benutzen - analoges gilt für den Client. Stellt ein Server seine Dienste über einen *Broker* zur Verfügung, kann ein Client, der diesen Dienst benötigt eine Anfrage an den *Broker* senden. Server und Client kennen sich somit nicht physisch, sondern rein logisch. Wenn sowohl Client als auch Server Klassen bereitstellen, die eine Serialisierung und Deserialisierung ihrer gesendeten Nachrichten übernehmen, können beide in unterschiedlichen Programmiersprachen realisiert werden. Soll das Architekturmuster auf mehrere Rechner verteilt werden, wird der *Broker* nicht auf einem eignen Rechner realisiert, wie man zuerst annehmen könnte; stattdessen wird er selbst verteilt. Jeder Rechner, der eine Client- oder Serverkomponente enthält erhält einen *Broker*. So kann ein lokaler *Broker* die Kommunikation innerhalb des Rechners übernehmen und sendet Daten an einen entfernten *Broker*, falls ein Dienst des entfernten Servers benötigt wird oder ein entfernter Client einen Zugriff verlangt. Ein Dienst muss durch dieses Prinzip auch nur an dem lokalen *Broker* angemeldet werden - ist diesem ein Dienst unbekannt kommuniziert er mit den übrigen *Brokern* und leitet die Anfrage an die korrekte Adresse weiter. [GD13]

Tabelle 2.1 zeigt drei Vor- und drei Nachteile, die sich aus der Anwendung des Broker-Architekturmusters ergeben.

Tabelle 2.1: Vor- und Nachteile des Broker-Architekturmusters nach Goll und Dausmann

Vorteile	**Nachteile**
+ Die Kommunikation zwischen Server und Client wird von der Funktionalität des Servers / Clients getrennt.	- Der Ausfall eines lokalen *Brokers* beeinträchtigt alle auf dem Rechner installierten Client- und Serverkomponenten.
+ Dienste können (auch dynamisch zur Laufzeit) verändert werden, ohne die Client-Seite anpassen zu müssen[1].	- Ein indirekter Aufruf über einen Broker resultiert in schlechterer Performance als eine direkte Kommunikation.
+ Client und Server können in unterschiedlichen Programmiersprachen realisiert werden.	- Der Broker kann ein Engpass bezüglich des Durchsatzes sein.

Gründe zum Einsatz des Broker-Architekturmusters:

- Eine Entkopplung von Client- und Serverkomponenten ist erwünscht.
- Komponenten sollen untereinander auf ihre Dienste zugreifen, ohne Kenntnis über deren physischen Adresse.
- Komponenten sollen zur Laufzeit veränderbar sein[1].
- Details der Implementierung von Client-Komponenten und Diensten sollen verborgen werden.

[1] Falls gerade kein Aufruf stattfindet und die Schnittstelle, welche die Aufrufe verwaltet, nicht verändert wird

Pipes and Filter

Das Architekturmuster Pipes and Filter erhielt seinen Namen aufgrund der Struktur welches dieses Muster prägt. Ein System wird dabei in seine Verarbeitungsschritte zerlegt. Ein einzelner Schritt wird als Filter bezeichnet. Je zwei Filter sind über eine Pipe miteinander verbunden. Ein Filter erhält eine Eingabe, verarbeitet sie sequentiell und wandelt sie in eine Ausgabe um. Diese Ausgabe kann schließlich über eine Pipe zu einem weiteren Filter fließen. Der Filter kann der Eingabe Teile entnehmen, hinzufügen oder Teile der Eingabe abändern. Die Pipes zwischen den Filtern puffern die Daten, die durch sie hindurch geschickt werden - die Filter werden somit asynchron entkoppelt. Eine Folge von Pipes and Filtern wird als *Pipeline* bezeichnet. Am Kopf einer solchen *Pipeline* befindet sich die Informationsquelle, an deren Ende die Informationsausgabe. Die beiden Komponenten bilden die einzige Schnittstelle nach außen. Abbildung 2.5 zeigt die Struktur des Architekturmusters Pipes and Filter; dabei sind die Schnittstellen grau, die Pipes schwarz und die Filter weiß hinterlegt. Die Gesamtheit der Pipes *eins* bis *vier* und der Filter *eins*, *zwei* und *drei* wird in ihrer Reihenfolge als eine Pipeline bezeichnet.

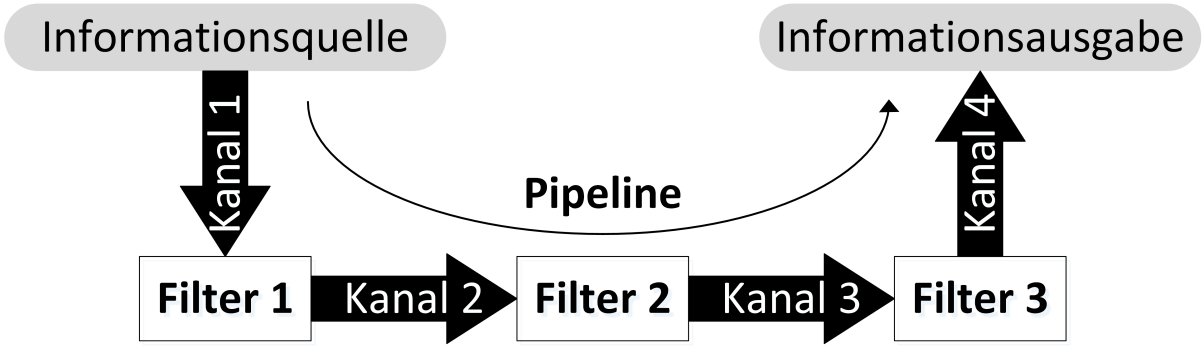

Abbildung 2.5: Struktur des Architekturmusters Pipes and Filter

Die Informationsquelle kann sich entweder aktiv oder passiv verhalten. Ist sie aktiv, schickt sie Daten an eine Pipe, während sie passiv wartet, bis der nächste Filter Daten von ihr anfordert. Ebenso verhält es sich mit der Informationsausgabe; diese fordert aktiv Daten aus einer Pipe an und wartet passiv, bis sie Daten erhält. Auch Filter können aktiv oder passiv im System agieren. Ein aktiver Filter sendet und empfängt an und von seinen anliegenden Pipes. Passive Filter erhalten ihre Daten von einer Pipe mit einem vorhergehenden, aktiven Filter. Ebenso werden die Daten des passiven Filters über eine nachfolgende Pipe von einem aktiven Filter abgeholt. Durch diese Festlegung sind einige Szenarien ableitbar, so zum Beispiel das *Pull-Prinzip*, bei dem nur die Informationsausgabe aktiv ist, oder das *Push-Prinzip*, bei dem die Informationsquelle allein aktiv ist. [GD13]

Tabelle 2.2 zeigt drei Vor- und drei Nachteile, die sich aus der Anwendung des Pipes and Filter-Architekturmusters ergeben.

Tabelle 2.2: Vor- und Nachteile des Pipes and Filter-Architekturmusters nach Goll und Dausmann

Vorteile	**Nachteile**
+ Innerhalb einer Pipeline sind Filter, Informationsquelle und/oder Informationsausgabe leicht aus- oder vertauschbar[1].	- Da im System kein gemeinsamer Zustand existiert, ist die Umsetzung einer Fehlerbehandlung erschwert.
+ Verarbeitungsschritte, die nicht nebeneinanderliegen, besitzen keinen direkten Datenaustausch und sind somit entkoppelt.	- Die Geschwindigkeit der Pipeline wird durch den langsamsten Filter bestimmt - keine vollständige Parallelisierung, da Filter aufeinander warten.
+ Filter sind in anderen Pipelines leicht wiederverwendbar.	- Bei einem schlecht gewählten Datenformat der Pipes kann ein erhöhter Aufwand bei der Datenkonvertierung in einzelnen Filtern entstehen.

Gründe zum Einsatz des Pipes and Filter-Architekturmusters:

- Die Verarbeitungsschritte werden sequentiell abgearbeitet.
- Pipelines und deren Komponenten könnten in zukünftigen Realisierungen wiederverwendet werden.

[1] Dies ist auch abhängig von der Festlegung des Datenformats innerhalb der Pipes

Model-View-Controller

Häufig muss in interaktiven Systemen mit unterschiedlichen *Darstellungen* auf verschiedene *Benutzereingaben* reagiert werden. Die angeforderten *Daten* müssen für diesen Vorgang gespeichert und verarbeitet werden. Das Architekturmuster Model-View-Controller (MVC) trennt die Komponenten Modell (*Model*), Ansicht (*View*) und Controller strikt voneinander ab. Dies bedeutet jedoch nicht, dass die Komponenten nicht miteinander kommunizieren können. Die Komponente Modell beinhaltet die Daten als auch deren Verarbeitung. Die Ansicht erhält die anzuzeigenden Daten von der Modell-Komponente und präsentiert sie den Benutzern. Der Controller verarbeitet die Benutzereingaben innerhalb des Systems. Eine Ansicht ist dabei einem Controller zugeordnet; ein Controller kann allerdings mehrere Ansichten steuern. Da die Controller- und Ansichtskomponente in einer wechselseitigen Abhängigkeit stehen (die Ansicht informiert den Controller über die vom Benutzer getätigten Eingaben und der Controller bestimmt den Zustand der Ansicht), werden diese in einer Abwandlung des Musters zu der Komponente Delegate (= Benutzerschnittstelle) zusammengefasst. Im Folgenden werden die drei Komponenten des MVC-Musters im Detail erläutert.
Das Modell wird von den übrigen beiden Komponenten unabhängig behandelt. Dies bedeutet, dass ein Konzept und eine Implementierung des Modells ohne Berücksichtigung der Ansicht oder des Controllers getätigt werden kann. Es besitzt die Funktionalität, gespeicherte Daten aufzurufen, zu verarbeiten und zu verändern. Um die Ansichtskomponente über die Änderung von Daten zu informieren, existieren zwei Vorgehensweisen. Beim passiven Modell teilt der Controller die Änderungen der Daten mit. Das aktive Modell informiert die Ansichten des Systems selbst über seine Zustandsänderungen. Hier können zwei Modi unterschieden werden: Im *Push-Betrieb* sendet das Modell seine Daten als Übergabeparameter an die Ansichtskomponente. Im *Pull-Betrieb* wird die Ansicht lediglich darüber informiert, dass neue Daten verarbeitet wurden - die Ansicht holt sich diese daraufhin selbst von der Modellkomponente ein.
Die Ansichtskomponente präsentiert die Daten des Modells dem Benutzer. Dabei sind unterschiedliche Ansichten, auch für identische Daten, vorgesehen. Dafür muss eine Ansicht genaue Informationen über ein Modell besitzen. Ändert sich der Aufbau eines Modells, hat das häufig auch eine Änderung der Ansicht zur Folge. Eine Ansicht enthält zudem Steuerelemente, wie zum Beispiel Eingabefelder, deren Benutzung ein Ereignis auslösen, das an den für die Ansicht zuständigen Controller weitergegeben wird.
Der Controller interpretiert die Anfrage(n) eines Benutzers und legt fest, welche Ereignisse zu welchen Funktionsaufrufen innerhalb des Modells führen. Ebenso gehört es zu seinen Aufgaben, die Ansicht über Änderungen zu informieren, die im Zusammenhang mit dem Funktionsaufruf stehen (zum Beispiel das Deaktivieren einer Schaltfläche). [GD13]

Tabelle 2.3 zeigt drei Vor- und zwei Nachteile, die sich aus der Anwendung des MVC-Architekturmusters ergeben.

Tabelle 2.3: Vor- und Nachteile des MVC-Architekturmusters nach Goll und Dausmann

Vorteile	**Nachteile**
+ Die Modell-Komponente kann unabhängig von der Benutzeroberfläche entworfen und die Benutzeroberfläche ohne Veränderungen an der Modell-Komponente angepasst werden. + Das Modell ist unabhängig von einer Oberfläche testbar. + Es können verschiedene Benutzeroberflächen für dieselbe Anwendung entworfen werden.	- Die Performance kann sinken, wenn eine Anzeige mehrere Aufrufe benötigt, um die zur Anzeige benötigten Daten von dem Modell zu erhalten. - Bei kleineren Anwendungen ist der erhöhte Implementierungsaufwand für die Erfüllung des Architekturmusters nicht gerechtfertigt.

Gründe zum Einsatz des MVC-Architekturmusters:

- Das System soll interaktiv sein.
- Die Benutzerschnittstelle soll vom Rest des Systems abgeschirmt werden.

Plugin

Der Begriff Plugin (auch Plug-in) stammt aus dem Englischen „*to plug sth. in*“ und bedeutet übersetzt „*etwas einstecken*“. Als Architekturmuster verstanden bedeutet dies ein lauffähiges System, das ständig erweiterbar ist. Der Hintergrund dieses Musters ist der Wunsch, Software, die relativ verlässlich in Betrieb ist, nicht mehr zu verändern, sondern flexibel um neue Funktionalitäten zu erweitern. Um dies zu bewerkstelligen, benötigt es Schnittstellen, die eine solche Erweiterung zulassen. Die Anwendung soll dabei auch ohne ein Hinzufügen neuer Komponenten funktionieren können. Eine Plugin-Komponente ist nicht als ein Endpunkt zu sehen - viel mehr kann das Plugin ebenfalls eine Schnittstelle implementieren, an der neue Plugins kaskadiert werden können. Ein Plugin-Manager realisiert die Schnittstelle zu einzelnen Plugins einer Anwendung. Möchte der Benutzer beispielsweise einen, der Applikation unbekannten, Dateityp öffnen, fragt der Plugin-Manager die existenten Plugins ab und prüft, ob diese Funktionen bereitstellen, welche diesen Dateityp behandeln können. Ist dies für ein Plugin der Fall, wird der Aufruf an das gefundene Plugin weitergeleitet. Wird kein Plugin gefunden führt die Anwendung allerdings weiter aus und startet eine Fehlerbehandlung. Abbildung 2.6 zeigt eine Realisierung des Plugin-Architekturmusters mit drei Plugins. Die Anwendung interagiert über einen Plugin-Manager mit ihnen. [GD13]

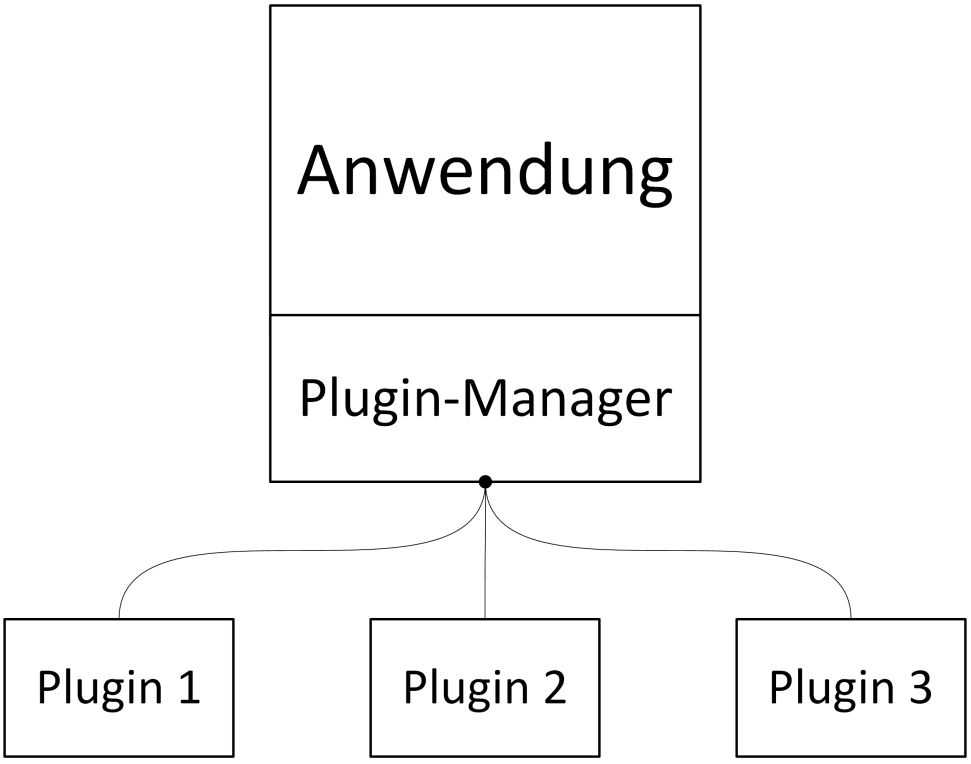

Abbildung 2.6: Struktur des Plugin-Architekturmusters

Tabelle 2.4 zeigt drei Vor- und drei Nachteile, die sich aus der Anwendung des Plugin-Architekturmusters ergeben.

Tabelle 2.4: Vor- und Nachteile des Plugin-Architekturmusters nach Goll und Dausmann

Vorteile	**Nachteile**
+ Jedes Plugin besitzt seine eigene Zuständigkeit. + Plugins erlauben die Erweiterung eines bestehenden Systems ohne Kenntnisse über dessen Programmcode. + Die Entwicklung komplexer Software ist bequem aufteilbar.	- Die Entwicklungszeit der initialen Anwendung ist höher, da in diese die für das Muster benötigten Schnittstellen implementieren werden müssen. - Der Verwaltungsaufwand während der Ausführung einer speziellen Anwendung steigt. - Es muss für alle Erweiterungen eine gemeinsame Schnittstelle gefunden werden.

Gründe zum Einsatz des Plugin-Architekturmusters:

- Ein großer Kreis von Benutzern hat unterschiedliche Anforderungen an die Anwendung.
- Einzellösungen zu entwickeln ist nicht kosteneffizient.
- Die Benutzer sollen die Anwendung selbst erweitern können.

Schichtenmodell

Das Schichtenmodell ist ein Architekturmuster, dass seine Komponenten als Teilsysteme in horizontale Schichten einteilt. Die Komponenten sind dabei Unteraufgaben wie die Datenhaltung oder die Kommunikation. Tiefere Schichten stellen den höheren Funktionalität in Form von Diensten bereit; unter den Schichten gilt das Client-Server-Prinzip. Eine Schicht hängt dabei nie von den ihr übergeordneten Schichten ab und jede Schicht bietet der nächst höheren ihre Dienste an. Ein Zugriff findet dabei über Schnittstellen statt, welche von der direkt darunterliegenden Schicht bereitstellt werden. Eine weniger strikte Variation des Musters erlaubt auch Zugriffe auf alle Schichten unter der aufrufenden Schicht. Die einzelnen Schichten des Schichtenmodells werden abstrakt betrachtet. So kann man ein Betriebssystem als eine Schicht ansehen welche sich über der Hardware-Schicht befindet. Geht man von einem verteilten System mit Client und Server aus, lässt sich das Schichtenmodell auf zwei Arten implementieren. Bei der *Thin-Client*-Architektur ist die oberste Schicht des Clients die Schicht der Benutzerschnittstelle (Eingabe/Ausgabe). Darunter folgt, da es sich um ein verteiltes System handelt, eine Kommunikationsschicht, um mit dem entfernten Server zu kommunizieren. Der Server stellt in seiner obersten Schicht die Kommunikationsschicht bereit. Darunter befindet sich eine Verarbeitungsschicht. Diese greift wiederum auf eine unter ihr liegende, persistente Datenhaltungsschicht zum Speichern oder Laden von Daten zu. Wendet man die *Fat-Client*-Architektur an, befindet sich die Verarbeitungsschicht statt auf dem Server direkt über der Kommunikationsschicht des Clients. Abbildung 2.7 veranschaulicht die beiden Varianten des Architekturmusters. [GD13]

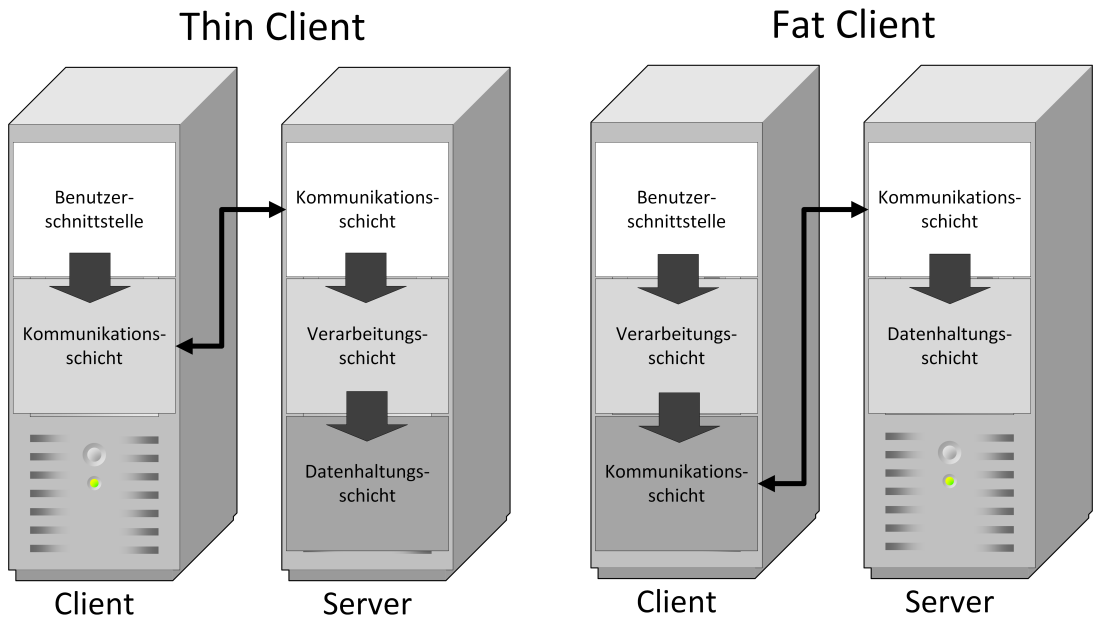

Abbildung 2.7: Architektur Thin/Fat-Client als Schichtenmodell

Tabelle 2.5 zeigt drei Vor- und drei Nachteile, die sich aus der Anwendung des Schichtenmodells ergeben.

Tabelle 2.5: Vor- und Nachteile des Schichtenmodells nach Goll und Dausmann

Vorteile	**Nachteile**
+ Jede Schicht stellt eine Abstraktion einer Funktionalität dar und erleichtert die Verständlichkeit des Ganzen.	- Eine Anfrage, die mehrere Schichten durchlaufen muss, ist langsamer als der direkte Zugriff.
+ Sind die Schnittstellen festgelegt, können die einzelnen Schichten parallel entwickelt werden.	- Änderungen können sich über eine Schicht hinaus erstrecken was Mehraufwand bedeutet.
+ Schichten können gegebenenfalls wiederverwendet werden.	- Es kann sich als schwer herausstellen, die *richtige* Anzahl an Schichten festzulegen.

Gründe zum Einsatz des Schichtenmodells:

- Da das Schichtenmodell abstrahiert, kann es besonders gut als ein übergeordnetes Architekturmuster verwendet werden, in dessen Schichten weitere Architekturmuster eingebettet werden.

Serviceorientierte Architektur

Eine Serviceorientierte Architektur (SOA) fasst Teile eines Geschäftsprozesses als Dienste in Form von Komponenten in einem verteilten System auf. Damit ein solcher Dienst den Anforderungen der Geschäftsprozesse gerecht wird, muss er eine Vielzahl von Eigenschaften besitzen. So sollte er zuallererst in einem Netzwerk zur Verfügung stehen. Von dort aus soll er unabhängig aufrufbar sein. Sein Zustand darf sich nach einer Benutzung für darauffolgende Aufrufe nicht verändern - er muss zustandslos sein. Die Dienste sollen untereinander entkoppelt sein und dynamisch zur Laufzeit aufgerufen werden können. Ein Dienst sollte zudem stets austauschbar sein. Dafür benötigt das System standardisierte Schnittstellen. Der Dienst soll plattform- und ortsunabhängig sein. Zur Nutzung des Dienstes sollen seine Schnittstellen genügen; die Implementierung bleibt verborgen. Die Dienste sollen schließlich in einem Verzeichnis registriert sein.
Ein Dienst kann aus mehreren elementaren Diensten zusammengesetzt sein, die genau eine einfache Funktion der Anwendung in sich kapseln. Ruft ein Dienst weitere Dienste oder elementare Dienste zur Erbringung seiner Leistung auf, nennt man ihn zusammengesetzten Dienst. Die Beziehungsstruktur zwischen dem Verzeichnis, dem Benutzer und dem Dienstanbieter wird als SOA-Dreieck dargestellt. Dieses ist in Abbildung 2.8 abgebildet. [GD13]

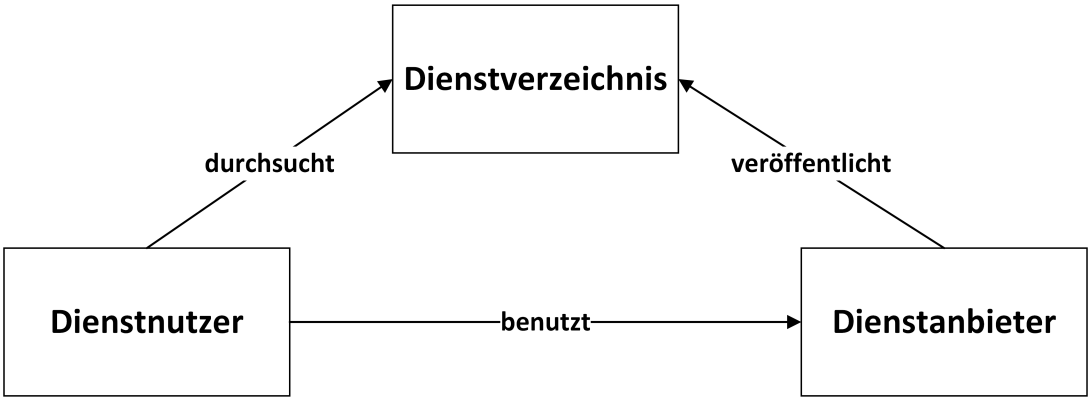

Abbildung 2.8: SOA-Dreieck

Tabelle 2.6 zeigt drei Vor- und drei Nachteile, die sich aus der Anwendung der serviceorientierten Architekur ergeben.

Tabelle 2.6: Vor- und Nachteile einer serviceorientierten Architektur nach Goll und Dausmann

Vorteile	**Nachteile**
+ Die Komplexität verteilter Systeme ist durch die Aufteilung in Komponenten von Diensten und elementaren Diensten reduziert. + (elementare) Dienste sind wiederverwendbar. + Bei gleichbleibender Schnittstelle kann die Implementierung eines Dienstes stets ausgetauscht werden.	- Eine feingranulare Dienstaufteilung erzeugt hohe Komplexität. - Es entsteht ein Mehraufwand durch eine Kommunikation über mehrere Schichten hinweg. - Die SOA ist nur bei klar definierten und dokumentierten Geschäftsprozessen möglich.

Gründe zum Einsatz einer serviceorientierten Architekur:

- Die Geschäftsprozesse sind gut dokumentiert und wohldefiniert.
- Das eingesetzte System ist komplex und soll vereinfacht werden.
- Kunden bzw. Lieferanten sollen in die Geschäftsprozesse eingebunden werden.

2.4.2 Kommunikationsparadigmen

Unter Einhaltung eines modularen Architekturansatzes, bietet es sich an, die Kommunikation zwischen den Hauptkomponenten möglichst unabhängig zu gestalten. Das bedeutet, dass Server und Client, unabhängig von ihrer konkreten Implementierung, Daten über die Kommunikationsschnittstelle austauschen können. Für dieses Ziel existiert ein Architekturansatz, der schon im Jahr 2000 entwickelt wurde und heute in vielen großen Frameworks wiederzufinden ist. Es handelt sich hierbei um das Programmierparadigma Representational State Transfer (REST). Das Paradigma basiert auf folgenden Prinzipien [Rod08]:

- Hypertext Transfer Protocol (HTTP)-Methoden werden explizit so verwendet, wie man sie konventionell vorgesehen hat [FGM+99]. Insbesondere die grundlegenden Create, Read, Update, Delete (CRUD)-Funktionen können so auf die HTTP-Methoden *POST*, *GET*, *PUT* und *DELETE* übertragen werden.
- Jeder Zugriff auf den angefragten Server erfolgt unter REST zustandslos. Diese Eigenschaft ist gerade durch die Anforderung an Verfügbarkeit und schneller Antwortzeit auch bei einer hohen Belastung unerlässlich. Jeder Client wird somit vom Server, seiner Anfrage bezüglich, gleichbehandelt; lediglich der Inhalt der übermittelten Nachricht und deren Antwort verändern die Zustände beider Seiten.
- Uniform Resource Identifiers (URIs) sind einzigartig, eingängig und sinnvoll gewählt. Hierbei wird zum Beispiel die Hierarchie, welche durch die Trennung der URI mit Schrägstrichen erfolgt, als ein Baum aufgefasst und so die Adresse logisch unterteilt.
- Das genutzte Format zum Datenaustausch ist simpel gehalten und für den Menschen leicht zu lesen und interpretieren. Dafür werden gängige Multipurpose Internet Mail Extensionss (MIMEs), wie Extensible Markup Language (XML) und JavaScript Object Notation (JSON), genutzt.

Eine Alternative des REST-Architekturansatzes ist die Verwendung von nachrichtenorientierter Middleware. Hierbei übernehmen Broker die Kommunikation zwischen den verteilten Systemen und ermöglichen so eine Entkopplung der Komponenten. Während bei REST ein synchroner Datenaustausch stattfindet, können die Broker Nachrichten auch asynchron untereinander austauschen. Ein synchroner Datenaustausch benötigt eine Fehlerbehandlung auf der Seite des Senders, wenn gesendete Nachrichten keine Antwort erhalten. Ein asynchroner Datenaustausch erfordert die Fehlerbehandlung auf Seiten des Brokers, der für die Nachrichtenübermittlung zuständig ist. Eine Schlussfolgerung dieser Eigenschaft ist, dass ein Ausfall oder Fehlverhalten des Brokers den gesamten Nachrichtenaustausch zum Stillstand bringt. [Cur04]

2.4.3 Benutzerfreundlichkeit

Die Benutzerfreundlichkeit (engl. *usability*) wird nach Nielsen wie folgt definiert:

„*Usability is a quality attribute that assesses how easy user interfaces are to use. The word 'usability' also refers to methods for improving ease-of-use during the design process.*“ [Nie03]

Nielsen unterteilt die Benutzerfreundlichkeit zudem in fünf Komponenten:

- **Lernbarkeit:** Wie leicht können Benutzer eine gegebene Aufgabe bei einer Erstbenutzung des Designs lösen.
- **Effizienz:** Wie schnell können Aufgaben gelöst werden, wenn der Benutzer einmal mit dem Design vertraut ist.
- **Informationsaufnahme:** Wie leicht kann erlerntes Können bezüglich des Designs nach einer gewissen Zeit wieder aufgerufen werden.
- **Fehler:** Wie viele Fehler machen Benutzer? Wie schwerwiegend fallen diese aus? Wie leicht ist die Fehlerbehandlung?
- **Bedürfnisbefriedigung:** Ist das Design angenehm bedienbar?

Neben der Benutzerfreundlichkeit existiert die Nützlichkeit als Qualitätsmerkmal, welches besagt, zu welchem Grad Software die von ihr geforderten Funktionen bereitstellt. Beide Merkmale legen fest, wie verwendbar eine Softwareanwendung ist. Ein hoher Grad an Benutzerfreundlichkeit ist vor allem im Internet von großer Relevanz. Ist diese nicht gegeben, verlassen die Benutzer eine Webseite. Im Intranet erhöht sie die Produktivität der Angestellten, falls sie hoch angesetzt ist. [Nie03]

Ein etabliertes Maß um ein Softwaredesign auf Benutzerfreundlichkeit hin zu prüfen, ist der Abgleich mit den acht goldenen Regeln von Shneiderman:

- **Konsistenz:** Verwandte Funktionen (z. B. Löschen, Weiter oder Zurück) sollten über das gesamte Design sowohl vorhanden und gleich benannt sein als auch gleich funktionieren.
- **Universelle Benutzbarkeit:** Sowohl neuen Benutzern als auch erfahrenen Anwendern des Designs soll eine praktische Benutzung angeboten werden. Dazu muss für Neulinge eine einfache, intuitive Bedienung angeboten werden, während erfahrenen Benutzern durch Abkürzungen ein schnellerer Arbeitsablauf ermöglicht wird.
- **Informative Rückmeldungen anbieten:** Der Benutzer muss über die durchlaufenen Aktionen unverzüglich nach deren Abschluss informiert werden. Die Rückmeldungen können bei häufig auftretenden Aktionen geringer und bei weniger auftretenden Aktionen höher ausfallen.
- **Abgeschlossenheit:** Bei Aktionen, die mehrere Sequenzen benötigen, soll der Benutzer zu jedem Zeitpunkt wissen, an welcher Stelle er sich befindet. Ebenso soll die Sequenz einen erkennbaren Anfang und ein erkennbares Ende haben.
- **Fehlervermeidung:** Dem Benutzer sollen illegale oder inkorrekte Interaktionen mit dem System erkennbar gemacht werden. Ebenso sollen dem Benutzer mögliche Lösungsvorschläge angeboten werden.
- **Umkehrbarkeit:** Es sollen so viele Aktionen wie möglich umkehrbar sein, damit sich der Benutzer aus einem unerwünschten Zustand entfernen kann.
- **Benutzerkontrolle:** Dem Benutzer soll die Kontrolle über das System gewährt werden. Er selbst soll durch seine Interaktionen Prozesse in Gang setzen, diese unterbrechen oder stoppen können.
- **Kurzzeitgedächtnis entlasten:** Der Benutzer soll mit der bereitgestellten Information nicht überfordert werden. Als eine Faustregel gilt, dass ein Mensch im Durchschnitt sieben plus/minus zwei Informationseinheiten im Kurzzeitgedächtnis aufnehmen kann [Mil56].

Um die Benutzerfreundlichkeit eines Designs zu steigern, muss die visuelle Wahrnehmung angezeigter Elemente bekannt sein. Einen ersten Zusammenhang stellte Wertheimer mit einer Sammlung wesentlicher Faktoren (später als Gestaltgesetze bekannt) her:

- **Gesetz der Nähe:** Gleiche oder sich ähnelnde Elemente, die nah beieinander liegen, werden auch als zusammengehörig wahrgenommen.
- **Gesetz der Ähnlichkeit:** Gleiche oder sich ähnelnde Elemente werden als zusammengehörig wahrgenommen.
- **Gesetz der Geschlossenheit:** Elemente, die einfache Formen wie ein Dreieck, einen Kreis oder ein Quadrat bilden, werden als zusammengehörig wahrgenommen.
- **Prägnanzgesetz:** Einzelne Elemente, die sich von allen anderen unterscheiden, werden vorrangig wahrgenommen.
- **Gesetz der einfachen Gestalt:** Unterbrochene oder mehrdeutige Formen werden als einfache Formen (z. B. Dreieck, Kreis oder Quadrat) wahrgenommen.
- **Gesetz der einfachen Fortsetzung:** Linien, die sich schneiden, werden so wahrgenommen, als ob sie in ihre ursprüngliche Richtung weiter verlaufen.

3 Analyse

In diesem Kapitel sind sämtliche Ergebnisse der Recherche festgehalten. Diese beginnt mit einer Prüfung des bisherigen Systems. Anschließend sind die Anforderungen an das System dokumentiert. Diese unterteilen sich in die funktionalen, die nichtfunktionalen und die sonstigen Anforderungen. Schließlich werden zwei ähnliche Arbeiten präsentiert und die Notwendigkeit dieser Arbeit durch eine Abgrenzung von diesen und dem bestehenden System untermauert. Die Analyse der genannten Abschnitte ist ein notwendiger Schritt, welcher der Konzeption vorausgeht. Die Erkenntnisse dieses Kapitels, besonders die Anforderungen an das System, legen den Grundstein für die Erstellung des Systementwurfs.

3.1 Ausgangssituation

Dieser Abschnitt soll erläutern, inwiefern eine Neuentwicklung des bestehenden Systems aus Sicht des aktuellen Stands der Technik notwendig ist. Dabei werden Probleme benannt und mögliche Lösungsansätze[1] skizziert, welche später im Systementwurf vertieft behandelt werden. Die Betrachtung beschränkt sich dabei lediglich auf die Benutzeroberfläche des Cocktailmixers (siehe auch Abschnitt 3.4). Der Abschnitt nimmt dabei Bezug auf den Stand der Benutzeroberfläche von Dezember 2016.
Die Analyse der Ausgangssituation teilt sich in die Benutzerfreundlichkeit und dem Softwaredesign auf.

[1] Die Lösungsansätze werden an dieser Stelle nur exemplarisch genannt, um zu verdeutlichen, dass eine Lösung der Probleme überhaupt möglich ist.

3.1.1 Benutzerfreundlichkeit

Die Benutzerfreundlichkeit der bestehenden Benutzerschnittstelle weißt an vier Stellen erkennbare Mängel auf, die im Folgenden genauer beschrieben werden.

Notwendiges Vorwissen

Um die bestehende Webanwendung zu benutzen, müssen Bediener und Administratoren die Uniform Resource Locators (URLs) für die Anmeldeseite wissen und manuell aufrufen. Ein Zugriffspunkt, der nach erfolgreichem Einloggen in die allgemeine Bedienoberfläche eingebettet ist, würde das Vorwissen auf die Zugangsdaten beschränken.

Verletzung der goldenen Regeln von Shneiderman

Die Benutzeroberfläche liefert an einigen Stellen eine unzureichende oder keine Rückmeldung auf getätigte Benutzereingaben. Diese können durch das Hinzufügen von Textmeldungen ergänzt werden. Ebenso wird die Regel der universellen Benutzbarkeit für neue Bediener und Administratoren verletzt - die Benutzeroberfläche ist wegen ihrer verschachtelten Menüführung und einiger uneinheitlicher Seiten zu kompliziert und muss vereinfacht werden.

Keine Internationalisierung

Das bestehende System ist nur in deutscher Sprache verfügbar. Durch die Implementierung einer Internationalisierung können weitere Sprachen eingepflegt werden. Dafür müssen jedes Vorkommnis eines sprachlichen Konstrukts dynamisch der ausgewählten Sprache angepasst werden.

Responsives Design

Die bestehende Benutzeroberfläche stellt kein *responsives Design* für mobile Endgeräte bereit. Die Webseite wird bei einem Aufruf von einem solchen Gerät der Bildschirmgröße des Geräts angepasst. Dies ist insbesondere bei *Smartphones* nachteilig, da die Anzeige dort sehr klein ausfällt und ein manuelles Vergrößern notwendig ist. Die Benutzeroberfläche muss demnach für mobile Endgeräte angepasst werden.

3.1.2 Softwaredesign

Das Softwaredesign der bestehenden Benutzerschnittstelle beinhaltet zwei Mängel, die an dieser Stelle genannt werden.

Fehlende Dokumentation

Der Cocktailmixer wurde inkrementell über mehrere Entwickler hinweg programmiert. Eine Dokumentation des Programmcodes wurde dabei minimal gehalten. Diese muss in Form eines Systementwurfs und einer Dokumentation innerhalb des Programmcodes erstellt werden.

Keine Modularisierung

Die Architektur des bestehenden Systems folgt keinem klaren Architekturmuster (vgl. Unterabschnitt 2.4.1). Stattdessen wirkt es, als wäre an den jeweils zu bearbeitenden Stellen Programmcode hinzugefügt worden, obwohl von vorherigen Entwicklern funktionsidentische Klassen an anderer Stelle bereitgestellt wurden. Der Code für den Bediener und den Administrator ist größtenteils redundant und könnte mit wenigen Änderungen vereint werden. Eine Kapselung des Codes findet ausschließlich bei dem Code des Bedieners und des Administrators statt, die jeweils in einen eigenen Ordner platziert wurden. Eine große Zahl unterschiedlicher Funktionen wurde in eine einzelne Datei geschrieben (vgl. Abschnitt 2.3), die von den anderen Klassen aufgerufen wird - das macht eine Wartung schwierig und den Code unübersichtlich. Dass System muss unter Einhaltung eines oder mehrerer Architekturmuster entwickelt werden, sodass eine Übersichtlichkeit hergestellt wird.

3.2 Anforderungskatalog

Dieser Abschnitt enthält alle gesammelten Anforderungen, die an eine Benutzeroberfläche des Cocktailmixers gestellt werden. Die Anforderungen wurden zum einen aus vorherigen Arbeiten entnommen, zum anderen entstammen sie mehreren geführten Gesprächen mit dem Betreuer und den Verantwortlichen. In Unterabschnitt 3.2.1 befinden sich alle erkannten funktionalen Anforderungen an das System. In Unterabschnitt 3.2.2 werden die nichtfunktionalen Anforderungen präsentiert. Schließlich sind in Unterabschnitt 3.2.3 alle sonstigen Anforderungen aufgezählt. Daraufhin werden zwei der Anwendungsfälle vorgestellt, welche eine Interaktion mit dem System beschreiben. Die Benutzerrollen sind zum einen der Bediener und zum anderen ein Administrator für die Verwaltung der Benutzeroberfläche. Das System besteht aus den Rollen Client und Server. Abbildung 3.1 veranschaulicht ein Anwendungsfalldiagramm bestehend aus den Anwendungsfällen, die aus den funktionalen Anforderungen abgeleitet wurden. In diesem Diagramm wurden die Anwendungsfälle der Benutzerrollen Bediener und Administrator veranschaulicht.

3.2.1 Funktionale Anforderungen

In diesem Unterabschnitt sind die funktionalen Anforderungen, ihrer Funktionalität nach sortiert, aufgezählt. Tabelle 3.1 beinhaltet alle Anforderungen an die Rezepte, Tabelle 3.2 zeigt alle Anforderungen an eine Rezeptliste. In Tabelle 3.3 werden die Anforderungen an eine Warteliste aufgeführt. Die Anforderungen an eine Startseite sind in Tabelle 3.4 aufgezählt. Tabelle 3.5 beinhaltet die Benutzerprofil-Anforderungen. Schließlich sind in Tabelle 3.6 alle erkannten Anforderungen eingetragen, welche die Administrationsoberfläche betreffen.

Tabelle 3.1: Softwareanforderungen - Rezepte

Nummer	Beschreibung der Anforderung
R1	Einzelne Rezepte sollen aufrufbar sein.
R2	Die Zutaten der Rezepte sollen angezeigt werden.
R3	Dem Benutzer soll ein Bild (mehrere Bilder) von dem Rezept angezeigt werden.
R4	Dem Benutzer sollen weitere Informationen über das Rezept angezeigt werden.
R5	Der Benutzer soll eine oder mehrere Rezepte bestellen können.

Tabelle 3.2: Softwareanforderungen - Rezeptliste

Nummer	Beschreibung der Anforderung
RL1	Dem Benutzer soll eine Liste mit verfügbaren Rezepten angezeigt werden.
RL2	Existieren viele Rezepte sollen diese über mehrere Seiten hinweg angezeigt werden.
RL3	Die Zutaten der Rezepte in der Rezeptliste sollen auf Wunsch angezeigt oder ausgeblendet werden.
RL4	Dem Benutzer soll ein Bild (mehrere Bilder) von dem Rezept in der Rezeptliste angezeigt werden.
RL5	Der Benutzer soll auf Wunsch weitere Informationen zu dem Rezept über die Rezeptliste einholen können.
RL6	Der Benutzer soll eine oder mehrere Rezepte über die Rezeptliste bestellen können.

Tabelle 3.3: Softwareanforderungen - Warteliste

Nummer	Beschreibung der Anforderung
W1	Eine aktuelle Liste mit offenen Bestellungen soll auf der Einstiegsseite angezeigt werden.
W2	Die Liste soll, je nach Bedarf, ein- und ausblendbar sein.
W3	Falls mehrere offene Bestellungen in der Liste vorhanden sind, soll dem Benutzer die Reihenfolge erkennbar gemacht werden.
W4	Nach Ausführung einer Bestellung soll diese aus der Warteliste entfernt werden.
W5	Falls eine Bestellung aktiv ist und der Benutzer innerhalb von einer bestimmten Zeit das Glas nicht abstellt und/oder den Starttaster nicht drückt, soll der Vorgang abgebrochen werden, die Bestellung aus der Liste der offenen Bestellungen gestrichen werden und die nächste Bestellung nachrücken.

Tabelle 3.4: Softwareanforderungen - Startseite

Nummer	Beschreibung der Anforderung
S1	Der Benutzer soll auf einer Seite informiert werden, welcher Vorgang derzeit abgearbeitet wird.
S2	Der Benutzer soll auf einer Seite informiert werden, wann das Glas auf die Ablagefläche des Tisches gestellt werden darf.
S3	Der Benutzer soll auf einer Seite informiert werden, wann das Getränk fertig zur Entnahme ist.

Tabelle 3.5: Softwareanforderungen - Benutzerprofil

Nummer	Beschreibung der Anforderung
B1	Der Benutzer soll sich am System anmelden können und von diesem Zeitpunkt an mit seinem eingetragenen Namen interagieren können.
B2	Falls der Benutzer sich mit einem korrekten Administrator-Passwort am System anmeldet, soll er von diesem Zeitpunkt an als Administrator interagieren können.
B3	Ein angemeldeter Benutzer soll sich vom System abmelden können und dabei sämtliche Privilegien, die er nach seiner Anmeldung erhielt, wieder verlieren.

Tabelle 3.6: Softwareanforderungen - Administrationsoberfläche

Nummer	Beschreibung der Anforderung
A1	Ein Administrator soll über die Benutzeroberfläche auf einen Administrationsbereich zugreifen können.
A2	Innerhalb des Administrationsbereichs sollen alle aktuellen Zustände des Cocktailmixers angezeigt werden.
A3	Innerhalb des Administrationsbereichs sollen alle Zutaten angezeigt werden.
A4	Innerhalb des Administrationsbereichs sollen Zutaten hinzugefügt werden können. Dabei soll festlegbar sein, ob die Zutat aktiv ist. Zudem soll der Name, der Typ, die Viskosität, der Port, die Information und die Bilder gesetzt werden können.
A5	Innerhalb des Administrationsbereichs sollen Zutaten bearbeitet werden können. Dabei soll festlegbar sein, ob die Zutat aktiv ist. Zudem soll der Name, der Typ, die Viskosität, der Port, die Information und die Bilder gesetzt werden können.
A6	Zwei Zutaten sollen sich nicht denselben Port teilen dürfen.
A7	Innerhalb des Administrationsbereichs sollen Zutaten gelöscht werden können, falls sie in keinem Rezept mehr vorhanden sind.
A8	Innerhalb des Administrationsbereichs sollen Zutaten nachgefüllt werden. Eine Statusleiste soll dabei den aktuell berechneten Füllstand der Zutat anzeigen.
A9	Innerhalb des Administrationsbereichs sollen alle Rezepte angezeigt werden.
A10	Innerhalb des Administrationsbereichs sollen Rezepte hinzugefügt werden können. Dabei soll festlegbar sein, ob das Rezept aktiv ist. Zudem soll der Name, die Zutaten, die Information und die Bilder gesetzt werden können.
A11	Innerhalb des Administrationsbereichs sollen Rezepte bearbeitet werden können. Dabei soll festlegbar sein, ob das Rezept aktiv ist. Zudem soll der Name, die Zutaten, die Information und die Bilder gesetzt werden können.
A12	Innerhalb des Administrationsbereichs sollen Rezepte gelöscht werden können.
A13	Innerhalb des Administrationsbereichs sollen alle Bestellungen angezeigt werden.
A14	Innerhalb des Administrationsbereichs sollen einzelne Bestellungen gelöscht werden können.
A15	Innerhalb des Administrationsbereichs sollen alle Bestellungen auf einmal gelöscht werden können.
A16	Innerhalb des Administrationsbereichs soll der Servicemodus zum Befüllen und Reinigen aktiviert werden können.

3.2.2 Nichtfunktionale Anforderungen

In diesem Abschnitt befinden sich jene Anforderungen, die keiner einzelnen Funktion innerhalb der Anwendung zugeordnet sind. Stattdessen entstammen die hier aufgeführten Anforderungen den Funktionen oder betten diese ein. Die nichtfunktionalen Anforderungen sind in Tabelle 3.7 aufgelistet.

Tabelle 3.7: Nichtfunktonale Anforderungen an das Softwaresystem

Nummer	Beschreibung der Anforderung
N1	Das System soll von mehreren Benutzern gleichzeitig bedienbar sein. Dabei gilt eine Untergrenze von 10 Personen.
N2	Das System soll von Seiten der Software ohne bemerkbare Verzögerungen bedienbar sein.
N3	Das System soll stabil sein.
N4	Das System soll testbar sein.
N5	Das System soll intuitiv bedienbar sein.
N6	Das System soll nachträglich änderbar und erweiterbar sein.
N7	Die Komponenten des Systems sollten möglichst unabhängig funktionieren.
N8	Das System soll möglichst plattformunabhängig sein.
N9	Das System soll verständlich konstruiert und dokumentiert sein.
N10	Das System soll wartbar sein.
N11	Das System soll auf Geräten mit unterschiedlicher Displaygröße angepasst dargestellt werden.

3.2.3 Sonstige Anforderungen

Dieser Abschnitt bündelt alle Anforderungen, die sich weniger auf das System, sondern mehr auf den Arbeitsprozess beziehen. Die Ergebnisse sind in Tabelle 3.8 festgehalten.

Tabelle 3.8: Sonstige Anforderungen an das Softwaresystem

Nummer	Beschreibung der Anforderung
SO1	Während der Entwicklung ist mit dem Betreuer Rücksprache zu halten und ihn über den Fortgang zu unterrichten.
SO2	Ein Prototyp soll angefertigt werden.
SO3	Das entwickelte System soll nach Fertigstellung bei einer Abnahme vorgestellt werden.

Tabelle 3.9: Anwendungsfall Rezept bestellen

Ziel		Der Besteller möchte ein Rezept zur Warteliste hinzufügen.	
Akteure		Besteller, Client-System, Server-System	
Beschreibung		Der Besteller intendiert das automatische Zubereiten des ausgewählten Rezepts von dem Cocktailmixer.	
Ebene		Benutzerebene	
Priorität		Medium	
Hauptablauf			
Vorbedingung		Der Benutzer befindet sich innerhalb der Benutzerschnittstelle des Cocktailmixers.	
1	Besteller	Der Besteller ruft die Rezeptliste auf.	
2	Besteller	Der Besteller wählt ein Rezept aus der Rezeptliste aus.	
3	Besteller	Der Besteller betätigt die Schaltfläche Bestellen in der Taskleiste des Rezepts.	
4	Client-System	Das Client-System übermittelt die Bestellung.	
5	Server-System	Das Server-System nimmt die Bestellung entgegen.	
		Alternative: Die Bestellung enthält fehlerhafte Daten.	Alternativablauf 5a
6	Server-System	Das Server-System fügt die Bestellung der Warteliste hinzu.	
7	Server-System	Das Server-System sendet eine Nachricht an das Client-System, um es über die erfolgreiche Eintragung der Bestellung zu informieren.	
8	Client-System	Das Client-System zeigt eine Meldung über den Erfolg der Eintragung an.	
Nachbedingung		Das Rezept ist der Warteliste hinzugefügt.	
Alternativablauf 5a			
Vorbedingung		Die Bestellung enthält fehlerhafte Daten.	
5a1	Server-System	Das Server-System übermittelt eine Fehlernachricht an das Client-System.	
5a2	Client-System	Das Client-System zeigt eine Fehlermeldung an.	
Nachbedingung		Das Rezept ist der Warteliste nicht hinzugefügt.	

Tabelle 3.10: Anwendungsfall Servicemodus aktivieren

Ziel		Der Administrator möchte den Servicemodus aktivieren.	
Akteure		Administrator, Client-System, Server-System	
Beschreibung		Der Servicemodus dient der Reinigung und dem Nachfüllen der Saftbehälter.	
Ebene		Administrationsebene	
Priorität		Medium	
Hauptablauf			
Vorbedingung		Der Administrator ist am System authentifiziert. Der Administrator befindet sich innerhalb der Benutzerschnittstelle des Cocktailmixers.	
1	Administrator	Der Administrator öffnet die Administrationsoberfläche über den zugehörigen Link.	
2	Administrator	Der Administrator wählt den gewünschten Servicemodus durch Betätigen der Schaltfläche aus (Reinigungsmodus / Nachfüllmodus).	
3	Client-System	Das Client-System übermittelt eine Nachricht an das Server-System.	
4	Server-System	Das Server-System validiert, dass der Administrators authentifiziert ist und somit eine Interaktion erlaubt ist.	
		Alternative: Der Benutzer ist nicht authentifiziert.	Alternativablauf 4a
5	Server-System	Der Server nimmt die Serviceanfrage entgegen.	
		Alternative: Die Daten sind fehlerhaft.	Alternativablauf 5a
6	Server-System	Der Server aktiviert den Servicemodus durch verändern einer Zeile in der Datenbank.	
7	Server-System	Der Server übermittelt eine Nachricht an das Client-System und informiert über eine erfolgreiche Eintragung.	
8	Client-System	Das Client-System zeigt dem Administrator eine Meldung über den Erfolg der Eintragung an.	
Nachbedingung		Der Servicemodus wurde aktiviert.	

Alternativablauf 4a		
Vorbedingung		Der Benutzer ist nicht authentifiziert.
4a1	Server-System	Das Server-System übermittelt eine Fehlernachricht an das Client-System.
4a2	Client-System	Das Client-System zeigt eine Fehlermeldung an.
Nachbedingung		Der Servicemodus wurde nicht aktiviert.
Alternativablauf 5a		
Vorbedingung		Die Daten sind fehlerhaft.
5a1	Server-System	Das Server-System übermittelt eine Fehlernachricht an das Client-System.
5a2	Client-System	Das Client-System zeigt eine Fehlermeldung an.
Nachbedingung		Der Servicemodus wurde nicht aktiviert.

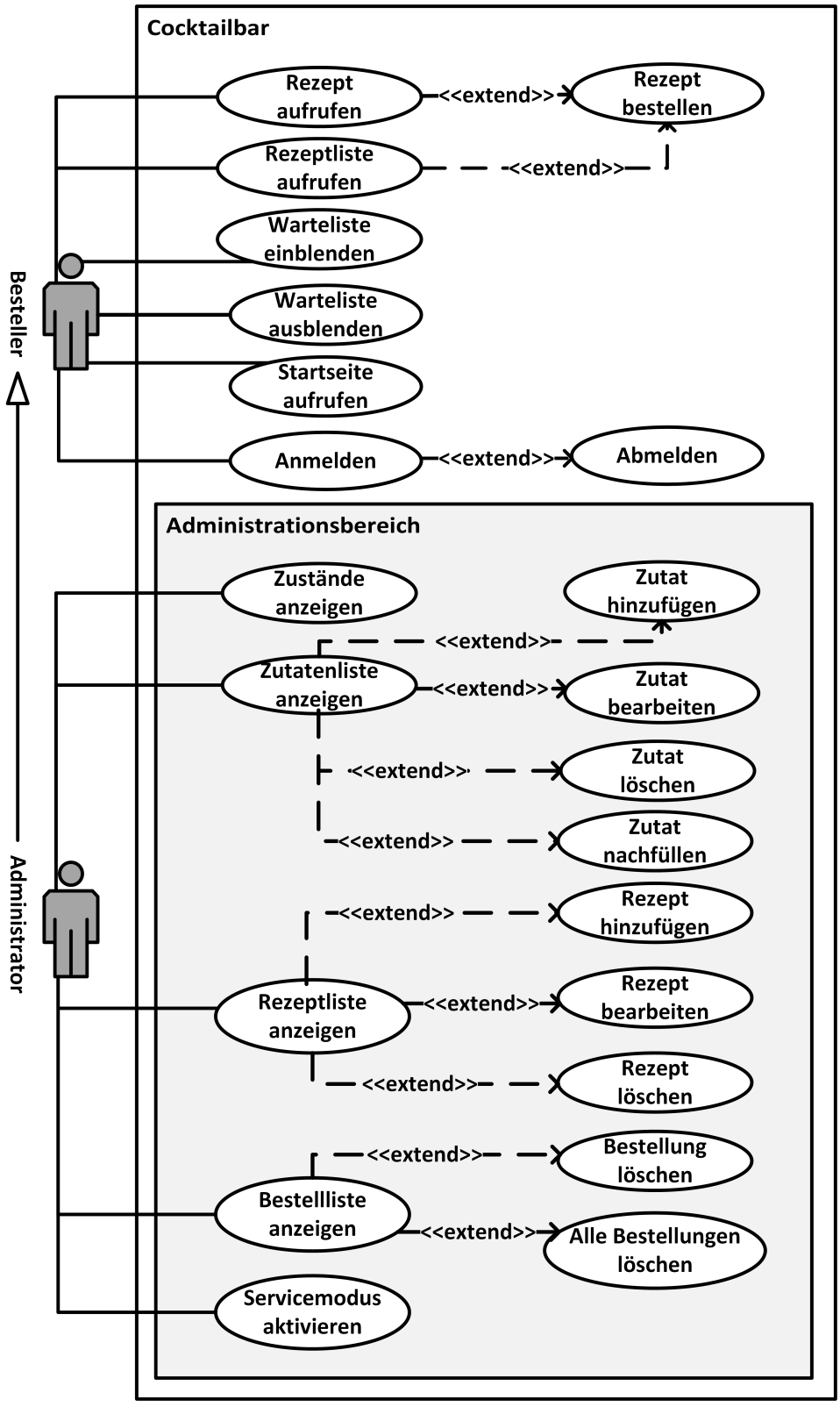

Abbildung 3.1: Anwendungsfalldiagramm

3.3 Verwandte Arbeiten

3.3.1 Bartendro™

Bartendro™ [Bara] ist ein auf *Kickstarter* [Kic] finanzierter, automatisierter Cocktailmixer. Die Schnittstelle zu dem Benutzer wird, ähnlich wie bei dem Cocktailmixer des ISW, über ein selbst erstelltes drahtloses Netzwerk hergestellt. Über einen Browser gelangt man auf die Hauptseite, auf welcher die zur Verfügung stehenden Cocktails ausgewählt werden können. Nach der Auswahl eines Cocktails können noch zusätzliche Einstellungen in Abhängigkeit der Zutaten vorgenommen werden. Beispielsweise kann einem Cocktail mehr oder weniger Alkohol zugewiesen werden. Erweiterte Einstellungen sind über eine Administrationsschicht ausführbar (z. B. das Einstellen neuer Getränke, das Zuschalten von Dispensern und das Reinigen dieser) Für die Programmierung wurde die Programmiersprache *Python* mit dem Framework *Flask* gewählt. Die entwickelte Benutzeroberfläche steht unter der *GNU General Public License* [Barc] zur Verfügung. [Par14]

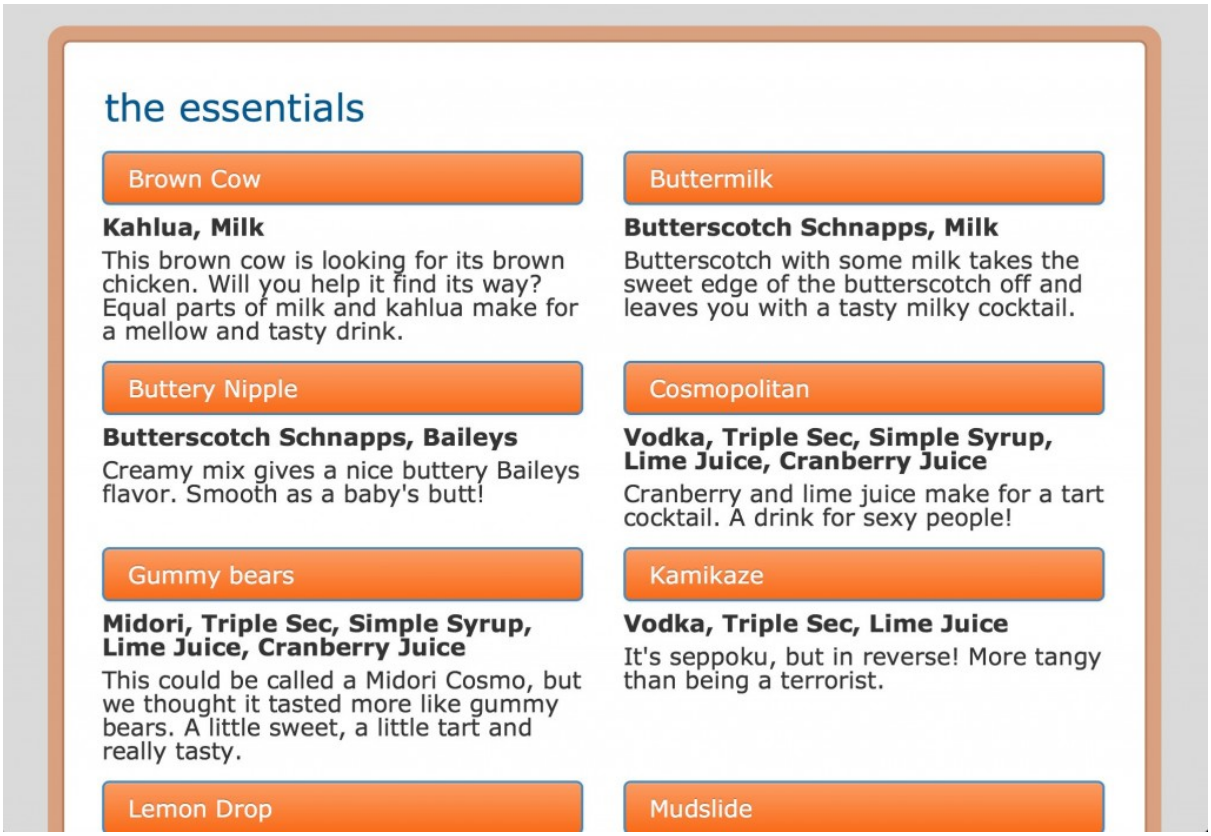

Abbildung 3.2: Cocktail-Auswahlbildschirm des *Bartendro™* Graphical User Interface (GUI) [Barb]

3.3.2 Barobot

Barobot [Bar14a] ist ein ebenfalls über *Kickstarter* [Kic] finanzierter, automatisierter Cocktailmixer. Zwar wurde das angestrebte Ziel an Einnahmen nie erreicht, dennoch entwickelten die Erfinder lange Zeit weiter an der Maschine. Mittlerweile existiert die Webseite des *Barobot* nicht mehr, der Programmcode ist allerdings noch frei zugänglich. [Bar13a] Anders als bei *Bartendro* kann *Barobot* mithilfe einer *Android App* bedient werden. Diese ermöglicht eine Auswahl aus über 1500 Cocktails, die bereits in der Datenbank vorinstalliert sind. Falls hierbei eine Entscheidung schwer fällt, können Cocktails auch zufällig bestellt werden. Der Benutzer kann auswählen, welche Flaschen an die Dispenser angebracht sind und das Programm filtert automatisch, sodass nur Cocktails angezeigt werden, welche mit der Auswahl an angeschlossenen Getränken mischbar sind. Auch neue Getränke können in die Datenbank eingegeben werden. [Bar13b]

Abbildung 3.3: Cocktail-Auswahlbildschirm der *Barobot* GUI [Bar14b]

3.4 Abgrenzung

Diese Arbeit beschäftigt sich mit der Entwicklung einer Benutzeroberfläche des bestehenden Cocktailmixers. Eine Anpassung des Maschinencodes findet nur seitens der Schnittstellen statt, die diesen mit der Benutzeroberfläche und der Datenbank verbinden. Des Weiteren wird das Betriebssystem so angepasst, dass die entwickelte Benutzeroberfläche auf diesem ausführbar ist. Eine Anpassung der Hardwarekomponenten findet nicht statt.
Zur Entwicklung der Benutzeroberfläche sollen insbesondere aktuelle Techniken (vgl. Abschnitt 2.4) zum Einsatz kommen. Die Architektur soll einen modularen Ansatz verfolgen, um eine schwache bis lose Kupplung einzelner Komponenten zu gewährleisten. Damit wird die Notwendigkeit einer weiteren Neuentwicklung der gesamten Oberfläche verringert.
Anders als bei dem entwickelten Cocktailmixer *Barobot* (vgl. Unterabschnitt 3.3.2), soll die Benutzeroberfläche von dem automatisierten Cocktailmixer des ISW als reine Webplattform realisiert werden. Dies ermöglicht eine hohe Plattformunabhängigkeit und erspart so einen Mehraufwand bei der Entwicklung. Die Oberfläche soll, anders als bei den verwandten Arbeiten, den Benutzern mitteilen, welche Interaktion nötig ist und ich welchem Zustand sich die Maschine befindet.

4 Konzeption

Abbildung 4.1: Logo der Weboberfläche *Cocktailbar*

Dieses Kapitel beinhaltet den Systementwurf der entwickelten Benutzeroberfläche des Cocktailmixers. Der Name der Arbeit wurde passend auf die reale Welt übertragen und lautet *Cocktailbar*. Das Logo (vgl. Abbildung 4.1), bestehend aus drei Gläsern, welche man häufig in einer Cocktailbar wiederfindet, soll die Oberfläche mit dem Anwendungsgebiet assoziieren. Das Kapitel gliedert sich in zwei Abschnitte. Im ersten wird der Entwurf, aufgeteilt in seine beiden Hauptkomponenten, vorgestellt. Der zweite Abschnitt zeigt die Ergebnisse eines Prototypen der Webanwendung *Cocktailbar*.
Hinweis: In den vorherigen Kapiteln wurde das Rezept als Begriff für eine Sammlung von Zutaten verwendet. Dieser Begriff ist zur Verständlichkeit gewählt worden, da eine Sammlung von Zutaten nicht zwangsläufig ein Cocktail ist. In der internen Realisierung der Softwareanwendung wurde statt Rezept jedoch der Begriff Cocktail benutzt.

4.1 Systementwurf

Cocktailbar ist in zwei Hauptkomponenten unterteilt: Dem *Server* und dem *Client*. Die Komponenten sind gemäß dem Schichtenmodell (vgl. Abschnitt 2.4.1) aufgebaut. Die Verarbeitungskomponente befindet sich dabei auf der Serverseite, was dem *Thin Client*-Prinzip entspricht. Die verarbeitende Komponente wird im folgenden Abschnitt über die serverseitige Architektur (vgl. Abschnitt 4.1.2) vertieft behandelt. Die Kommunikationsschnittstelle der beiden Komponenten folgt dem REST-Paradigma. Eine Kapselung mit Hilfe des Schichtenmodells und die Verfolgung des REST-Paradigmas ermöglichen es jederzeit, eine der beiden Komponenten auszutauschen oder sie ohne die jeweils andere Komponente zu betreiben.Dadurch ist die nichtfunktionale Anforderung N7 erfüllt. Der Client ist für die Anzeige, Eingabebehandlung und Aktualisierung der Bedienelemente zuständig, während der Server Informationen über den Cocktailmixer aufruft, abspeichert und an den Client weitergibt.
Das Schichtenmodell unter Einhaltung des *Thin Client*-Prinzips eignet sich besonders gut für das zu entwickelte System: Wie in den folgenden Unterabschnitten beschrieben, besitzen die Komponenten eine Hierarchie, wie sie auch bei diesem Architekturmuster Anwendung findet. Eine ähnliche Hierarchie ist auch bei dem Architekturmuster Pipes und Filter (vgl. Unterabschnitt 2.4.1) gegeben, allerdings wird der Gebrauch aktiver oder passiver Filter nicht benötigt. Die beidseitige Benutzung einer Kommunikationsschicht erlaubt es außerdem, das in Unterabschnitt 2.4.2 beschriebene REST-Paradigma einzusetzen, um eine Abkopplung der beiden Hauptkomponenten *Server* und *Client* zu bewirken (vgl. Anforderung N7). Der Beschluss die Verarbeitungsschicht serverseitig zu platzieren begründet sich durch das Vorhandensein restriktiver Zugänge. Eine clientseitige Verarbeitung wäre in diesem Fall ein potentielles Sicherheitsrisiko und wird deshalb vermieden. Betrachtet man die Kommunikationsschicht als einen Broker, kann ebenjenes Broker-Muster angewandt werden. Um dieses in einem verteilten System zu betreiben, müssten die Schnittstellen allerdings erst angepasst werden (Für die bisherigen Anforderungen ist ein verteiltes System nicht notwendig). Das MVC-Modell eignet sich für diese Architektur nicht. Würde man dieses übertragen, wäre der Server als *Model* anzusehen und der Client als *View* / *Controller* - Der Server hat allerdings mehr Funktionalität als die einer *Model*-Komponente und das Zusammenlegen von Server und Client verringert wiederum die Modularisierung. Der Einsatz einer SOA ist elegant, wäre allerdings für diese Arbeit zu komplex; Geschäftsprozesse werden für den automatischen Cocktailmixer nicht angewandt. Da das bisherige System keine Schnittstellen bereitstellt, ist eine Realisierung als Plugin-Architekturmuster ebenfalls nicht umsetzbar.

4.1.1 Systemarchitektur

Dieser Unterabschnitt behandelt den Server und den Client im Detail. Dafür werden die konzipierten Komponenten erläutert und zur Förderung der Verständlichkeit mit Hilfe von Unified Modeling Language (UML)-Diagrammen veranschaulicht. Weitere Informationen zu

UML können dem Werk *Unified Modeling Language Reference Manual, The (2Nd Edition)* von Rumbaugh, Jacobson und Booch entnommen werden.

4.1.2 Serverseitige Architektur

Der Server ist in sechs Pakete unterteilt. Das Paket *transport* ist der Kommunikationsschicht zuzuordnen. Es empfängt GET- und POST-Anfragen und sendet Antworten, die es von der Controller-Komponente erhält. Diese ist gemeinsam mit dem Paket *models*, *exceptions*, und *machine* Teil der Verarbeitungsschicht. Der Controller prüft dabei, ob es sich bei den Anfragen um valide Operationen handelt und sendet gegebenenfalls einen Fehler, der im Paket *exceptions* definiert ist. Die für die zu verarbeitenden Objekte benötigten Klassen befinden sich in dem Paket *models*. Die Komponente *machine* ist von den übrigen Paketen völlig losgelöst. In ihr befindet sich der ursprüngliche Maschinencode, der die Schnittstelle zu der Hardware bildet (vgl. Unterabschnitt 2.2.3). Da der Code jedoch über eigene Schnittstellen Code aus der Datenbank ausliest, wurde er in die Schicht über der Datenhaltung eingebettet. Die Datenhaltungsschicht bildet die unterste Schicht des Servers und stellt Funktionalitäten zum Lesen und Schreiben in einer *MySQL*-Datenbank bereit. Die geschilderte Struktur kann in Abbildung 4.2 nachvollzogen worden. Die Starter-Klasse *CocktailBackendApplication* vervollständigt das Diagramm. Diese Klasse befindet sich in keinem Paket und beinhaltet einige innere Verwaltungsklassen und eine Startfunktion.

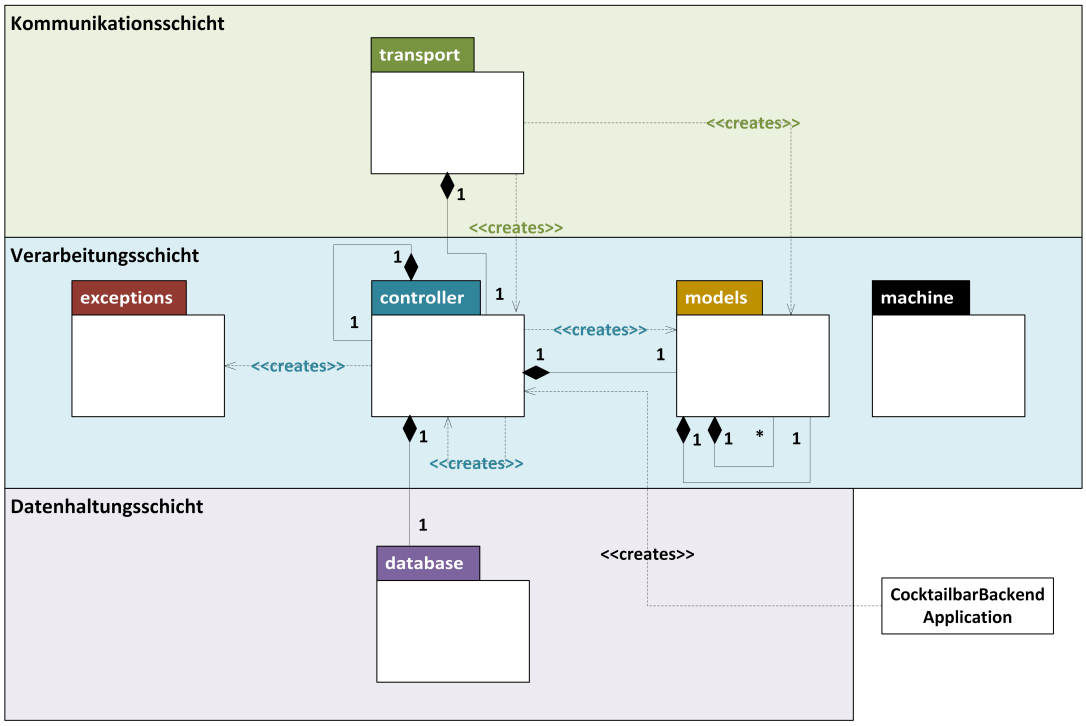

Abbildung 4.2: UML-Paketdiagramm des Servers

Paket transport

Das Paket *transport* bildet die Kommunikationsschicht und damit die oberste Schicht des Servers. Hier befindet sich die Schnittstelle zum Frontend. Die Funktionen, die hier bereitgestellt werden, können vom Frontend beispielsweise über GET- und POST-Anfragen angesprochen werden und so einen Datenaustausch bewirken. Die Objekte werden an dieser Stelle auch an Klassen aus dem Paket *models* gebunden. Die in diesem Paket bestehenden Klassen beinhalten alle Funktionen des Pakets *controller*, welche eine Interaktion über die Kommunikationsschicht erfordern. Eine Zugriffskontrolle findet ebenfalls auf dieser Ebene statt; unerlaubte Zugriffe auf restriktive Funktionen werden direkt abgewiesen. Folgende Klassen sind hier realisiert:

- **AdministrationReceiver:** Verwaltet das Setzen von Servicemodi und das Nachfüllen von Zutaten.
- **AuthenticationReceiver:** Verwaltet Authentifizierungsanfragen und antwortet mit einem Token, der den Benutzer für zukünftige Anfragen authentifiziert, oder einer Fehlermeldung.
- **CocktailReceiver:** Beinhaltet sämtliche Funktionen zum Transport von Informationen über die Rezepte des Systems. Dazu gehört beispielsweise das schrittweise Ausgeben aller vorhandenen Rezepte oder das Hinzufügen/Entfernen eines Rezeptes.
- **IngredientReceiver:** Beinhaltet sämtliche Funktionen zum Transport von Informationen über die Zutaten des Systems. Dazu gehört beispielsweise das schrittweise Ausgeben aller vorhandenen Zutaten oder das Hinzufügen/Entfernen einer Zutat.
- **OrderReceiver:** Stellt die Schnittstelle rund um den Bestellvorgang zur Verfügung. Hier wird die Kommunikation zum Hinzufügen, Entfernen oder Einholen einer oder mehrerer Bestellungen realisiert.
- **PortReceiver:** Gibt eine Liste aller hinterlegten Anschlüsse des Cocktailmixers zurück, die schließlich von Zutaten belegt werden können.
- **UserReceiver:** Regelt das Einholen von Informationen über einen Benutzer oder dessen Ausloggen aus dem System.

Paket exceptions

Das Exception-Paket des Servers enthält alle Fehlertypen, die von *Cocktailbar* benötigt werden. Folgende Fehlertypen können vom Paket *controller* geworfen werden:

- **CocktailStepsCocktailNullException:** Beim Hinzufügen eines einzelnen Schritts für die Zubereitung eines Rezepts wurde kein Rezept angegeben.
- **CocktailStepsExistException:** Das gegebene Rezept enthält bereits Zubereitungsschritte.
- **CocktailStepsIngredientNullException:** Beim Hinzufügen eines einzelnen Schritts für die Zubereitung eines Rezepts wurde keine Zutat angegeben.
- **CocktailStepsNotConsecutiveException:** Die übermittelten Schrittnummern sind nicht durchgängig.
- **CocktailStepsStepnumberExistsException:** Die angegebene Schrittnummer existiert bereits. Dies kann beispielsweise auftreten, wenn eine Schrittnummer innerhalb eines Rezepts doppelt an den Server gesendet wurde.
- **IngredientInUseException:** Die zu löschende Zutat ist noch Teil eines Rezepts und kann nicht entfernt werden.
- **OrderCocktailNullException:** Eine gegebene Bestellung steht nicht im Kontext zu einem Rezept.
- **OrdererUsernameExistsException:** Der gewünschte Benutzername ist bereits belegt.
- **OrdererUsernameNotAllowed:** Der gewünschte Benutzername ist nicht erlaubt (z. B. *admin*).
- **OrdererUsernameNullException:** Kein Benutzername wurde angegeben.
- **PortIdentifierNotFoundException:** Der angegebene Anschlussbezeichner ist nicht im System hinterlegt.
- **PortIdentifierNullException:** Es wurde kein Anschlussbezeichner angegeben.
- **PortOccupiedException:** Der gegebene Anschluss ist bereits belegt.

Paket controller

Das Controller-Paket des Servers enthält die Grundfunktionen der Webanwendung Cocktailbar und befindet sich in der Verarbeitungsschicht des Servers. Die Verarbeitungsklassen der Benutzeranfragen sind als Singleton realisiert, die, mit Hilfe von Modellobjekten, Anfragen verarbeiten. Außer den Klassen, welche die Benutzeranfragen verarbeiten, befinden sich in diesem Paket Klassen für den korrekten Betrieb des Servers. Dazu gehören vor allem Authentifizierungslogiken und erweiterte Zugriffsregeln, sowie Sicherheitskomponenten. Im Folgenden werden die Verarbeitungsklassen der Benutzeranfragen genauer erläutert.

- **AdministrationProcessor:** Die Administrationsverwaltungsklasse beinhaltet alle Methoden, die einen administrativen Zugang erfordern. Die Klassen werden von den zugriffsbeschränkten Methoden aus dem Paket *transport* aufgerufen. Die Klasse erlaubt Funktionalitäten wie das Hinzufügen oder Entfernen von Rezepten und Zutaten. Sie sendet nach einer erfolgreichen Verarbeitung die Daten entweder an die Datenhaltungsschicht weiter oder wirft einen Fehler aus dem Paket *exceptions*. Um die Verarbeitung besser zu verstehen, ist in Abbildung 4.3 ein exemplarischer Ablauf zum Hinzufügen einer Zutat modelliert. Zur Vereinfachung ist in diesem Diagramm die Verarbeitung von Zutatenbildern nicht berücksichtigt. Ebenso wurde, zur Bewahrung der Übersicht, der gesamte *Client* zusammengefasst. Wie in der Abbildung zu sehen ist, ruft die Klasse *AdministrationProcessor*, nachdem sie von der korrespondierenden Transportklasse aufgerufen wurde, einige Methoden der Datenhaltungsschicht auf, um zu prüfen, ob die Anfrage valide ist. Daraufhin wird ein Zutatenmodell befüllt und in der Datenhaltung abgespeichert.
- **OrderProcessor:** Die Bestellverarbeitung behandelt den Umgang mit Bestellungen. Dazu gehört das schrittweise Ausgeben der Cocktails, das Prüfen von eingehenden Bestellungen, das Ausgeben einer Bestellliste und die Ausgabe des Maschinenzustands.
- **UserProcessor:** Die Klasse *UserProcessor* erlaubt die Verwaltung der Benutzer. Dazu gehört das Hinzufügen, Ausloggen oder Ausgeben eines Benutzers und seines Profils.

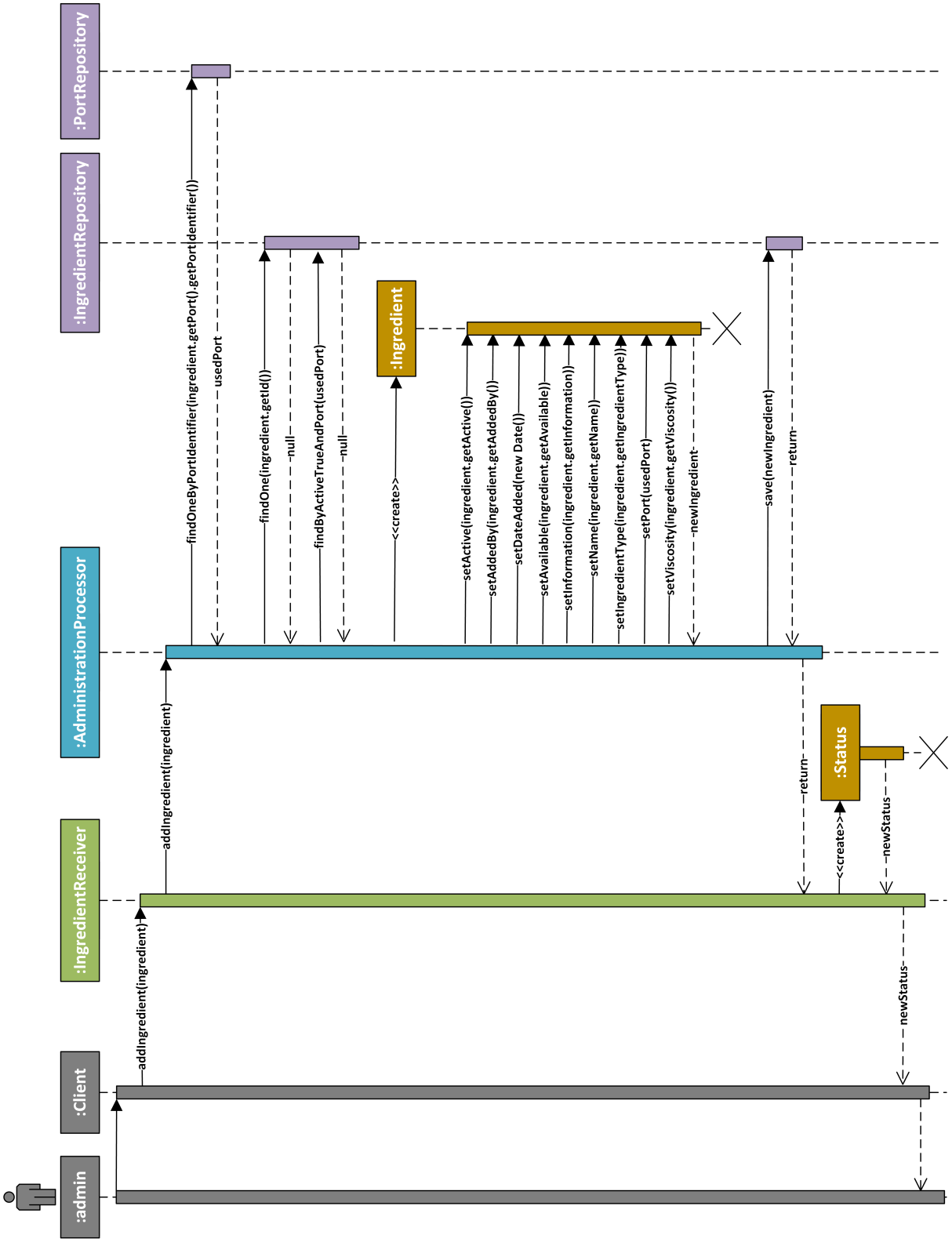

Abbildung 4.3: UML-Sequenzdiagramm - Zutat hinzufügen

Paket models

Das Models-Paket beinhaltet alle Klassen, die notwendig sind, um *Cocktailbar* in einer objektorientierten Umgebung zu betreiben. Hier werden Klassen für Objekte wie Cocktails oder Besteller definiert. Das Erzeugen neuer Objekte wird im Paket *controller* oder *transport* vorgenommen. Dort werden die von der REST-Schnittstelle empfangenen Daten in ein Modell überführt. Die Modelle kommen in fast identischer Form auch clientseitig zum Einsatz. Ebenso werden sie nahezu identisch in der Datenbank abgelegt. Die folgende Klassen sind auf dem System im Einsatz, ihre Abhängigkeiten sind in Abbildung 4.4 anhand eines Klassendiagramms dargestellt.

- **Cocktail:** Das Modell *Cocktail* beschreibt die Rezepte innerhalb des Systems. Rezepte können aktiv sein oder nicht. Sie besitzen eine eindeutige Identifizierungssnummer, einen Namen, ein Datum der Erstellung sowie einen Ersteller, ein oder mehrere Bilder (vgl. *ImageUrl*), einen Informationstext, die Anzahl bisheriger Bestellungen, die benötigte Zeit für die Zubereitung, keinen, einen oder mehrere Schritte (vgl. *Step*) und einen Typen.
- **Drucksensor:** Das Modell *Drucksensor* repräsentiert den Druckschalter im System und beschreibt, ab welchem Gewicht ein Glas erkannt werden soll. Das Modell wird innerhalb des Systems nicht eingesetzt, muss aber in der Datenbank abgelegt werden, damit der Maschinencode diese Informationen auslesen kann.
- **ImageUrl:** Das Modell *ImageUrl* speichert den Pfad der Bilddateien für Rezepte und Zutaten. Ein Pfad besitzt eine eindeutige Identifizierungsnummer, die Pfadadresse und einen Cocktail oder eine Zutat.
- **Ingredient:** Das Modell *Ingredient* beschreibt die Zutaten innerhalb des Systems. Zutaten können aktiv sein oder nicht. Sie besitzen eine eindeutige Identifizierungsnummer, einen Namen, ein Datum der Erstellung sowie einen Ersteller, ein oder mehrere Bilder (vgl. *ImageUrl*), einen Informationstext, einen Port, die Anzahl bisheriger Bestellungen, die benötigte Zeit für die Zubereitung, eine Viskosität und einen Typen. Das Modell Zutaten ist exemplarisch in Abbildung 4.5 samt seiner Abhängigkeiten veranschaulicht.
- **Order:** Die Klasse *Order* beinhaltet ein Modell für Bestellungen, das vom System für die Verarbeitung ebenjener benötigt wird. Dabei hat eine Bestellung immer eine eindeutige Identifizierungsnummer, ein bestelltes Rezept, ein Datum der Erstellung (wichtig für die Reihenfolge der Abarbeitung), einen Besteller und einen Status.
- **Orderer:** Das Modell *Orderer* repräsentiert den Benutzer innerhalb des Systems. Es besitzt eine eindeutige Identifizierungsnummer, einen Benutzernamen, ein Password (für Administratoren), ein Datum der Erstellung, und eine oder mehrere Rollen.
- **OrdersAndTasks:** Das Modell *OrdersAndTasks* wird benötigt, um dem Client in regelmäßigen Abständen Informationen über den Zustand des Systems zu übermitteln. In ihm befindet sich eine Liste der Aufgaben und ihres Zustands (vgl. *Task*) und eine Liste der obersten Bestellungen (vgl. *Order*) für das Anzeigen der Warteliste.

- **Port:** Das Modell *Port* spiegelt einen Anschluss an dem Cocktailmixer wider. Jeder Anschluss hat einen eindeutigen Namen. Ebenso hat ein Anschluss eine X-Koordinate, die bestimmt, wie weit der Schrittmotor zu fahren hat, bis er an der korrekten Position angelangt ist. Der Typ des Anschlusses (Dispenser oder Ventil) ist ebenso hinterlegt.
- **Role:** Eine Rolle wird von der Klasse *Role* im System repräsentiert. Jedem Benutzer kann eine Rolle zugewiesen werden, damit er bestimmte Aufgaben innerhalb des Systems ausführen kann.
- **Status:** Die Klasse *Status* ist für die Kommunikationsschicht bestimmt. Ein Status besteht lediglich aus einem Titel und einer Nachricht, die über die Kommunikationsschicht versendet wird. Der Status beinhaltet dabei Fehler oder Informationen, die der Server an den Client übermitteln muss.
- **Step:** Das Modell *Step* dient der Datenhaltung einzelner Schritte zur Zubereitung eines Cocktails. Einem Schritt ist dementsprechend auch immer ein Rezept und eine Zutat zugewiesen. Hinzu kommt die Anzahl der Abgaben, die der jeweilige Anschluss seinem Getränk abziehen soll (z. B. bedeutet die Zahl eins an einem Dispenser 25ml). Schließlich hat jeder Schritt, der einem Rezept zugeordnet ist eine Position, an der er ausgeführt wird.
- **Task:** Die Klasse *Task* stellt den Zustand der Maschine im System dar. Alle Zustände, die der Cocktailmixer einnehmen kann sind hier hinterlegt. Ebenso ist zu jedem Zustand festgehalten, ob dieser aktiv ist oder nicht. Der Cocktailmixer kann mehrere der folgenden Zustände einnehmen:
  - **IDLE:** Der Cocktailmixer wartet auf einen Befehl.
  - **JOBDONE:** Das Rezept wurde abgearbeitet, das Glas kann jetzt entnommen werden.
  - **DOINGJOB:** Der Cocktailmixer arbeitet momentan ein Rezept ab.
  - **QUIT:** Der Zustand SHUTDOWN wurde aktiv. Die Maschine soll bei der nächsten Möglichkeit heruntergefahren werden und keine weiteren Aufträge abarbeiten.
  - **REBOOT:** Der Cocktailmixer soll bei der nächsten Möglichkeit neu starten.
  - **RUNNING:** Der Cocktailmixer ist in Betrieb.
  - **SERVICE_CLEANING:** Der Reinigungsservice ist in Betrieb.
  - **SERVICE_REFILL:** Der Nachfüllservice ist in Betrieb.
  - **SHUTDOWN:** Der Cocktailmixer soll heruntergefahren werden.
  - **START:** Ist dieser Zustand aktiv, soll der Cocktailmixer die Startroutinen durchführen.
  - **STOP:** Zustand zum Aktivieren eines Sicherheitsstopps.

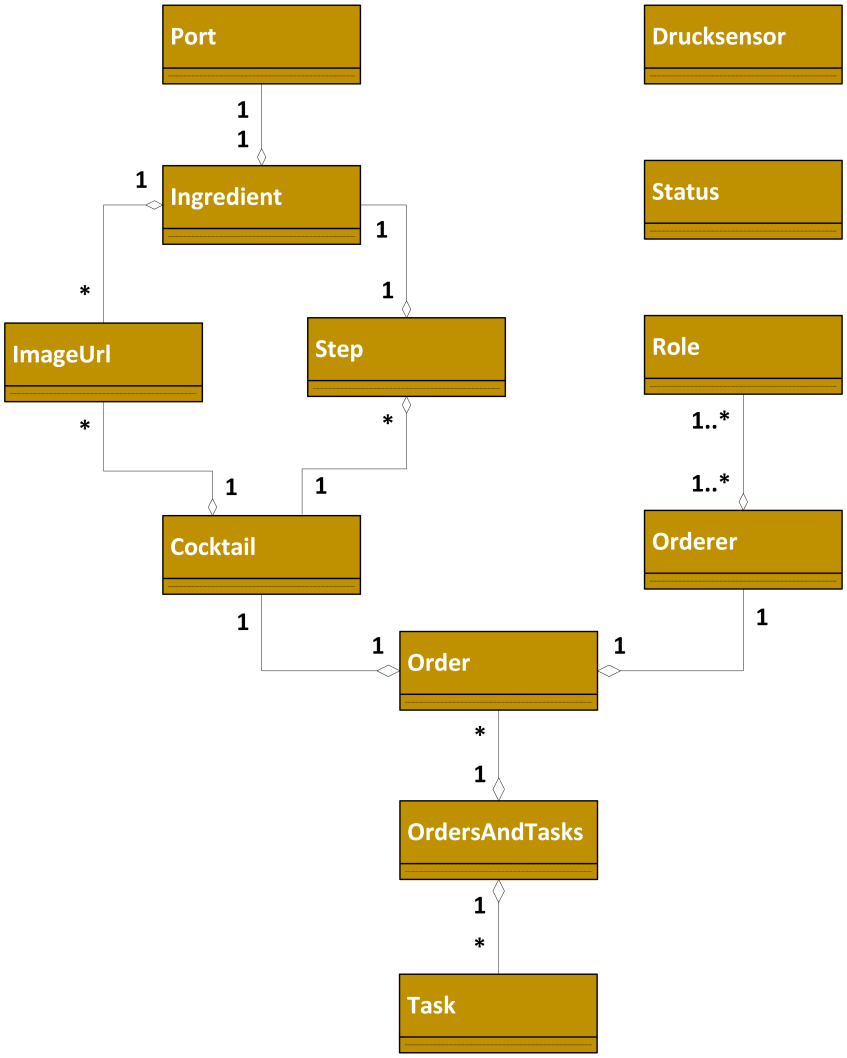

Abbildung 4.4: Klassen und deren Abhängigkeiten des Paket *models*

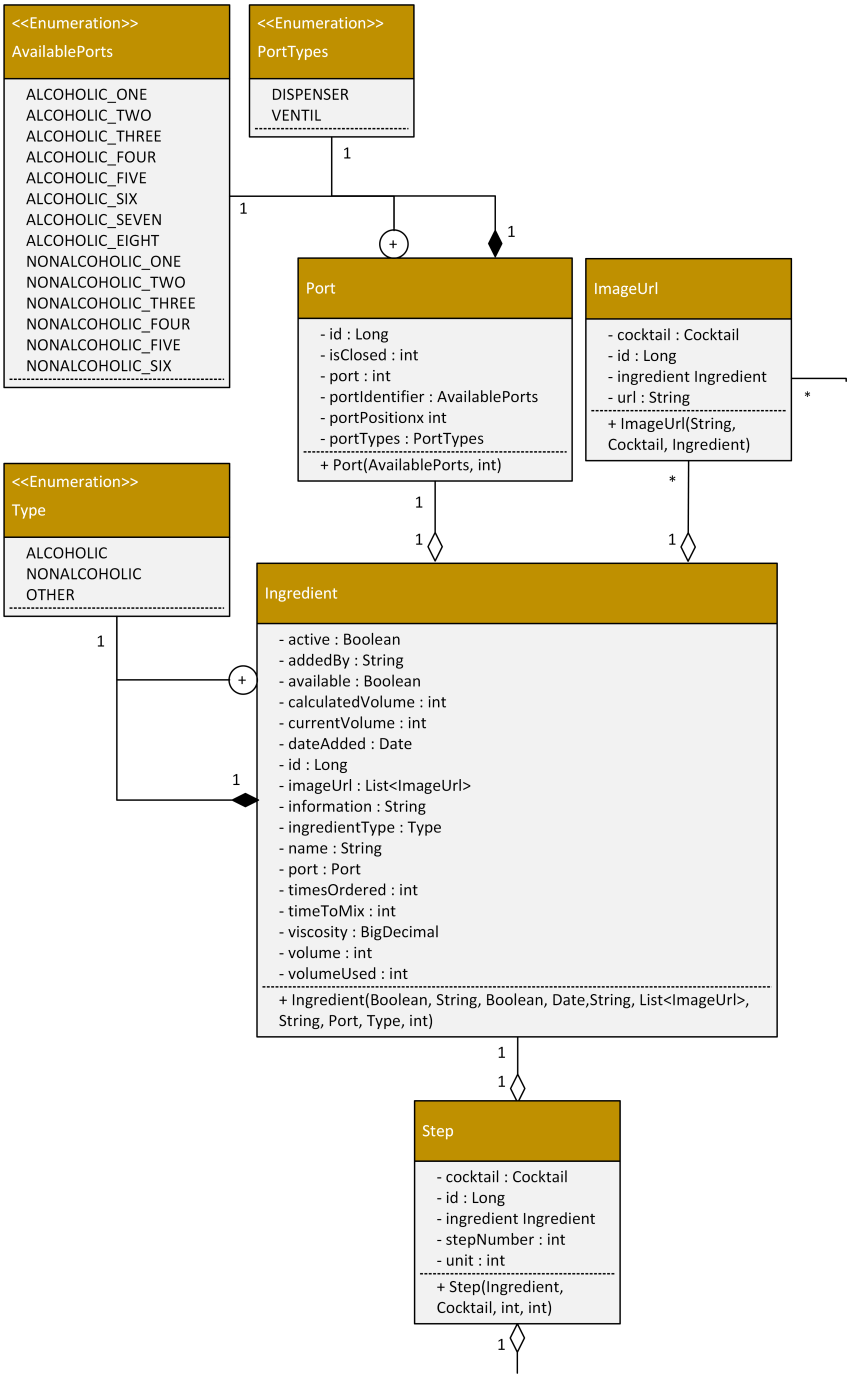

Abbildung 4.5: Ingredient-Klasse samt Abhängigkeiten

Paket machine

Das Paket *machine* enthält den in Unterabschnitt 2.2.3 beschriebenen Maschinencode. Das Paket hat keine Schnittstellen zu den restlichen Komponenten und stellt auch keine Schnittstellen zur Verfügung. Die Kommunikation zwischen dem Maschinencode und dem restlichen System findet ausschließlich über das Ablegen und Auslesen von Datenbankinhalten statt.

Paket database

Das Paket *database* beinhaltet Klassen zur Interaktion mit einer Datenbank. Da die Klassen im Umgang mit der Datenbank generisch agieren, können in den meisten Fällen unterschiedliche *SQL*-Datenbanken eingesetzt werden ohne ihren Code anzupassen [1]. Die Klassen des Pakets basieren auf der *Java Persistence* Programmschnittstelle, was das Ablegen und Aufrufen von Daten aus der Datenbank in einer objektorientierten Umgebung stark vereinfacht, da der Entwickler nur noch in seltenen Fällen in *SQL* selbst schreiben muss. Das verbessert die Lernkurve beim Einarbeiten in das System und erfüllt die nichtfunktionalen Anforderungen N9 und N10. Jedes in Unterabschnitt 4.1.2 beschriebene Modell (außer das zusammengesetzte Modell *OrdersAndTasks* und das rein transportbezogene Modell *Status*) besitzt eine eigene Klasse für die Interaktion mit der Datenbank. Abbildung 4.6 zeigt ein Enhanced entitiy-relationship (EER)-Diagramm [Elm08] der Datenbank mit allen Tabellen samt Spalten und deren Abhängigkeiten untereinander.

[1] Spezifische Muster der gewählten SQL-Datenbank müssen im Code dennoch berücksichtigt werden

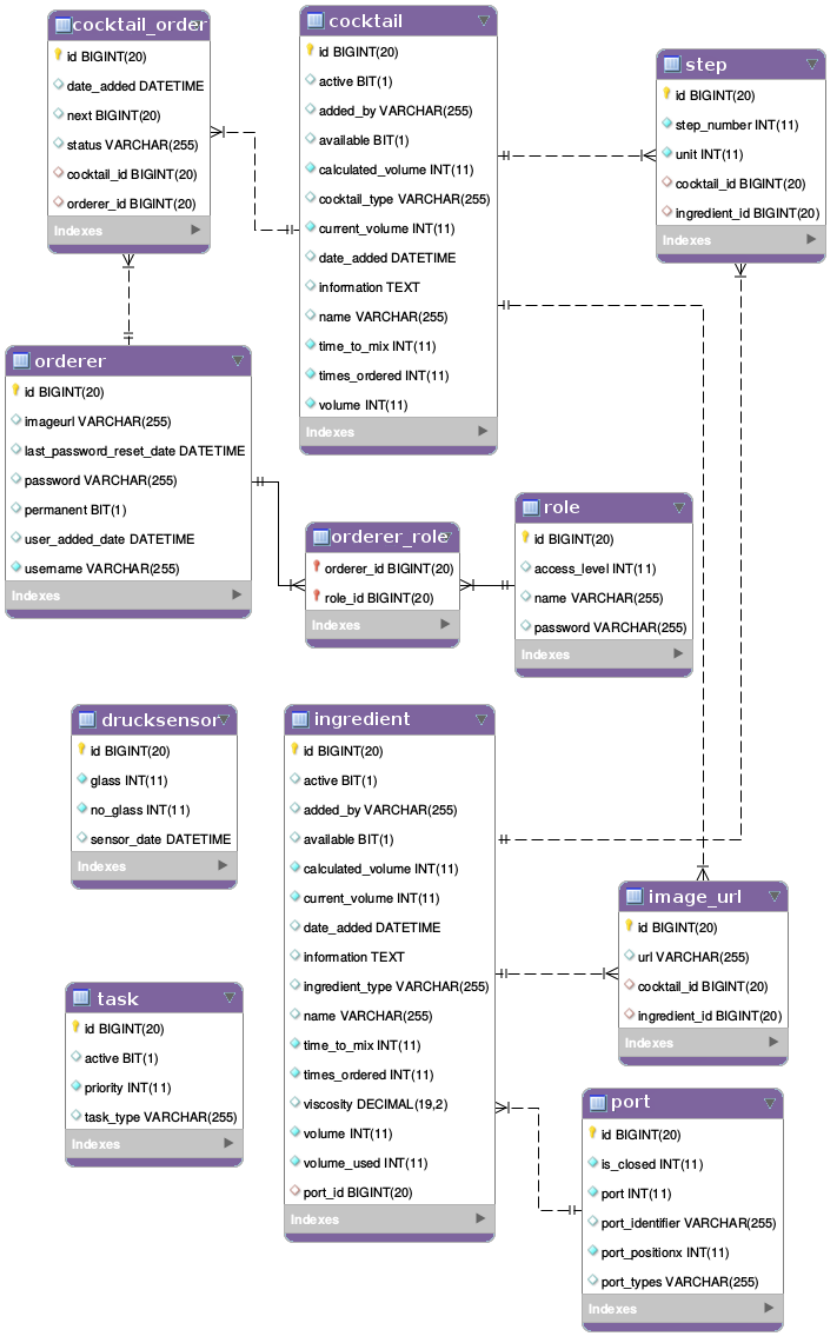

Abbildung 4.6: EER-Diagramm der Datenbank

4.1.3 Clientseitige Architektur

Der Client ruft Funktionen und Ressourcen des Servers über die Kommunikationsschnittstelle auf und bildet die Softwareanwendung damit auf dem Endgerät ab. Für die Anzeige der Software und die Erfüllung der Anforderungen müssen eine Vielzahl an Elementen bereitgestellt werden. Bekannte Vertreter sind hierbei Text, Schaltflächen, Bilder oder Eingabemasken. Darüber hinaus ist ein wesentlicher Faktor der Anzeige dessen Interaktivität und die Fähigkeit, angezeigte Informationen nahtlos aktuell zu halten. Eine große Herausforderung stellt die Bereitstellung der Anwendung auf unterschiedlichsten Geräten dar. Bekannte Vertreter sind hierbei der Computer, das Tablet und das Smartphone. Jedes dieser Geräte benutzt schließlich einen eigenen Webbrowser, der gegebenenfalls eine gesonderte Behandlung benötigt. Sowohl für den Einsatz unterschiedlichster Elemente als auch der Notwendigkeit einer Interaktivität und Aktualität von Informationen existieren Frameworks, die dem Entwickler helfen, sich auf die wesentliche Entwicklung zu konzentrieren, statt jedes Konzept von Grund auf selbst zu programmieren. Frameworks für die Anzeige von Elementen basieren hierbei fast immer auf den Websprachen HTML und Cascading Style Sheets (CSS). Der Client ist in sechs Pakete unterteilt. Das Paket *actions* fungiert als einziges Paket der Kommunikationsschicht und ermöglicht den Informationsaustausch mit dem Server. Die Ergebnisse dieser Aufrufe wird an das Paket *reducers* weitergeleitet, welches diese dann in die Pakete *components* und *routes* übergibt. Das Paket *components* beinhaltet einzelne Anzeigekomponenten. Das Paket *routes* enthält alle notwendigen Seiten für die Anzeige der Komponenten. Komponenten und Seiten bedienen sich des Pakets *messages*, in welchem Übersetzungen für unterschiedliche Sprachen enthalten sind.

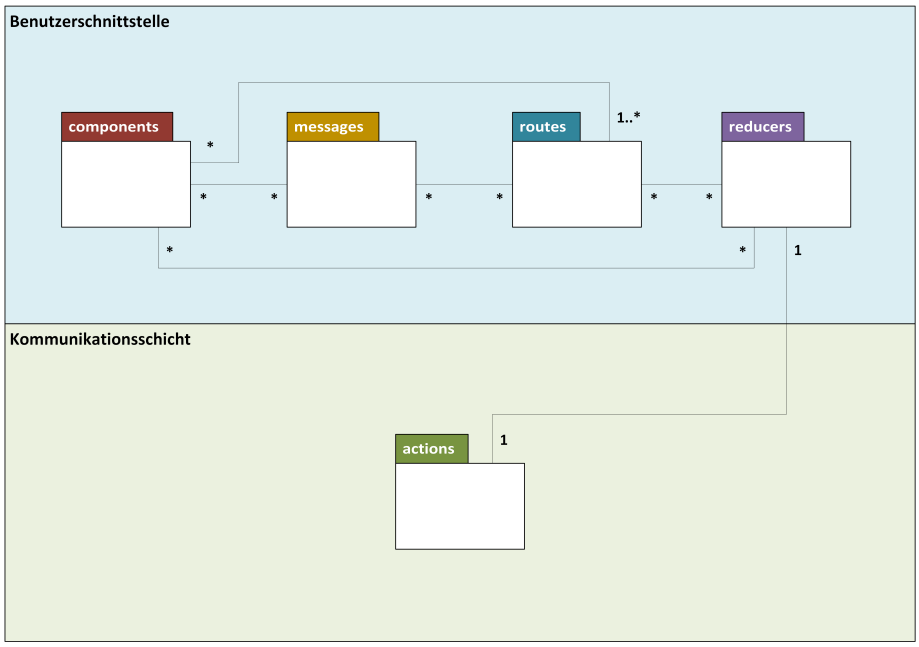

Abbildung 4.7: UML-Paketdiagramm des Clients

Paket components

Das Paket *components* beinhaltet die für die Anzeige benötigten Komponenten. Dabei handelt es sich nicht um primitive Anzeigeelemente, wie Textboxen oder Schaltflächen, sondern viel mehr um Elemente, welche über diese primitiven Elemente hinaus zur Anzeige benötigt werden oder eine Zusammensetzung aus primitiven Anzeigeelementen darstellen. Die hier angelegten Elemente werden schließlich auf den Seiten des Pakets *routes* aufgerufen und eingebunden. Folgende Anzeigekomponenten wurden für die Anzeige erstellt:

- **AdminCocktails:** Die Komponente *AdminCocktails* ist für die Anzeige der Rezeptliste im Administrationsbereich zuständig. Schaltflächen für das Speichern, die Bearbeitung und das Löschen werden hier definiert. Ebenso existiert für jedes Feld des Modells *Cocktail* eine Eingabemöglichkeit, um dieses abzuändern. Die Anzeige der Rezepte ist auf mehrere Seiten verteilt, die über ein horizontal ausgerichtetes Zahlenfeld angesteuert werden können. Auf einer Seite werden bis zu fünf Rezepte abgebildet.
- **AdminIngredients:** Die Komponente *AdminIngredients* ist für die Anzeige der Zutatenliste im Administrationsbereich zuständig. Schaltflächen für das Speichern, die Bearbeitung und das Löschen werden hier definiert. Ebenso existiert für jedes Feld des Modells *Ingredient* eine Eingabemöglichkeit, um dieses abzuändern. Die Anzeige der Zutaten ist auf mehrere Seiten verteilt, die über ein horizontal ausgerichtetes Zahlenfeld angesteuert werden können. Auf einer Seite werden bis zu fünf Zutaten abgebildet.
- **AdminOrders:** Die Komponente *AdminOrders* ist für die Anzeige der Bestellliste im Administrationsbereich zuständig. Schaltflächen für das Löschen einzelner oder aller Bestellungen sind hier definiert. Die Anzeige der Bestellungen ist auf mehrere Seiten verteilt, die über ein horizontal ausgerichtetes Zahlenfeld angesteuert werden können. Auf einer Seite werden bis zu zwanzig Bestellungen abgebildet.
- **Cocktail:** Die Komponente *Cocktail* ist für die Anzeige eines einzelnen Rezepts zuständig. Dieses unterscheidet sich in der Anzeige von der Rezeptliste insofern, dass zusätzlich zu den Zutaten der Informationstext angezeigt wird. Ebenso hat der Benutzer über ein Zurück-Bedienfeld die Möglichkeit, zu seiner vorherig besuchten Seite zurückzukehren.
- **CocktailList:** Die Komponente *CocktailList* ist für die Anzeige der Rezeptliste zuständig. Jedes Rezept wird dabei samt seinen Zutaten angezeigt. Eine Schaltfläche ermöglicht es, auf Geräten mit kleinem Display die Zutaten auszublenden, um die Liste so schneller zu durchlaufen. Zusätzlich wird am unteren Rand des Rezepts eine Schaltfläche zum Bestellen angezeigt. Die Anzeige der Rezepte ist auf mehrere Seiten verteilt, die über ein horizontal ausgerichtetes Zahlenfeld angesteuert werden können. Auf einer Seite werden bis zu fünf Rezepte abgebildet.
- **Footer:** Die Fußleiste der Webseite beinhaltet einen Link zum ISW.

- **Header:** Die Kopfleiste der Webseite beinhaltet die Links zum Navigieren auf der Webseite und die Komponente *LanguageSwitcher*. Ebenso wird das Logo der Anwendung Cocktailbar angezeigt.
- **LanguageSwitcher:** Die Komponente *LanguageSwitcher* ermöglicht das Umschalten der Sprache mit Hilfe von Flaggensymbolen im Kopfbereich der Webseite.
- **Link:** Eine Komponente zum Navigieren auf der Webseite.
- **Navigation:** Eine Sammlung von Links, die in der Kopfleiste angezeigt wird.
- **Orders:** Die Komponente *Orders* realisiert die Warteliste, die zusammen mit der Kopf- und Fußleiste auf jeder Seite der Webanwendung angezeigt wird.
- **ServiceModePanel:** Die Komponente *ServiceModePanel* zeigt zwei Schaltflächen zum Aktivieren der vorhandenen Servicemodi.
- **Statuspanel:** Die Komponente *Statuspanel* dient der Anzeige des Status der Hardware im Administrationsbereich. Mit Hilfe dieser Anzeige ist der Administrator mit einem Blick in der Lage den Zustand der Maschine nachzuvollziehen.

Paket messages

Das Paket *messages* beinhaltet die Übersetzungen der Texte auf der Webseite Cocktailbar. Jeder Text erhält eine eindeutige Identifizierung und wird umschrieben, sodass ein Übersetzer die Datei ohne Vorwissen bearbeiten kann und so eine neue Sprache in das System eingliedern kann. Bisher wurden die Sprachen Deutsch und Englisch im System hinterlegt.

Paket routes

Das Paket *routes* beinhaltet die einzelnen Seiten der Webseite Cocktailbar. Auf diesen werden die Komponenten des Pakets *components* angezeigt. Ebenso können diese Seiten über die Adressleiste des Webbrowsers direkt angesteuert werden. Bei mehrdeutigen Seiten, wie denen eines einzelnen Rezepts, hilft eine Identifizierungsnummer bei der Ansteuerung. Die folgenden Seiten sind für den Client notwendig:

- **admin:** Diese Seite beinhaltet die Liste der Rezepte, Zutaten und Bestellungen als Administrationsansicht. Ebenso wird der Maschinenzustand hier abgebildet.
- **cocktail:** Diese Seite stellt ein einzelnes Rezept dar. Der Benutzer kann jedes Rezept über eine Identifizierungsnummmer in der Adressleiste des Webbrowsers ansteuern. Die Rezepte sind gleichzeitig auch über die Rezeptliste zu erreichen.
- **display:** Die Seite *display* zeigt dem Benutzer Informationen über den aktuellen Verarbeitungsschritt der Maschine an. Dazu gehört beispielsweise die Aufforderung an den obersten Benutzer der Warteliste, ein Glas auf den Schlitten der Maschine zu stellen.

- **error:** Diese Seite wird dem Benutzer angezeigt, falls ein Fehler im System auftrat.
- **home:** Die Seite *home* zeigt dem Benutzer die Rezeptliste und dient gleichzeitig als Startseite der Webanwendung Cocktailbar.
- **login:** Die Login-Seite ermöglicht das Anmelden am System, entweder nur mit einem Benutzernamen, oder mit einem Passwort, um den Administrationsbereich aufzurufen.
- **notFound:** Diese Seite wird angezeigt, falls der Benutzer einen Abschnitt der Webseite ansteuert, der nicht definiert ist oder auf den er mit seinen aktuellen Rechten nicht zugreifen darf.

Paket actions

Das Paket *actions* bildet die Kommunikationsschicht des Clients und damit die Schnittstelle zum Server ab. Die Funktionen, senden über GET- und POST-Anfragen Befehle an den Server und verarbeiten dessen Antwort. Unterabschnitt 4.1.2 beschreibt bereits die Schnittstellen der Kommunikationsschicht des Servers. Auf der Clientseite ist ein gegenläufig äquivalenter Aufbau realisiert.

Paket reducers

Das Paket *reducers* verarbeitet die Anfragen aus dem Paket *actions* und teilt diese mit den Komponenten und Seiten der Webanwendung Cocktailbar. In diesen Klassen kann somit auf die Antworten des Servers reagiert werden, um dem Benutzer eine angemessene Darstellung auf seine Anfrage zu ermöglichen. Falls ein Fehler bei der Anfrage auftrat kann der Zustand zum Beispiel so manipuliert werden, dass eine Textbox erscheint, die dem Benutzer den Fehlertext präsentiert.

4.2 Prototyp

Dieser Abschnitt beinhaltet die Ergebnisse der Erstellung eines Prototypen für die Webanwendung *Cocktailbar*. Der Prototyp wurde für eine Darstellung auf einem Computer und einem Smartphone bzw. Tablet optimiert. Dies erfüllt Anforderung N11. Der Prototyp erleichtert die darauffolgende Entwicklung, da unterschiedliche Darstellungsformen bereits im Voraus geplant werden können. Im oberen Teil der mobilen Ansicht (vgl. Abbildung 4.8) ist die Warteliste zu erkennen. Daran anschließend befindet sich eine Liste zur Sammlung der eigenen Bestellungen. Dies ist nicht Bestandteil der Anforderung gewesen, könnte allerdings nachträglich in die Oberfläche eingepflegt werden. Die Rezeptliste listet die Rezepte mit Bild und den einzelnen Zutaten auf. Durch betätigen der Details-Schaltfläche gelangt man zu einer detaillierteren Ansicht des Rezepts. Ebenso kann man, durch Betätigung der Bestellen-Schaltfläche ein Rezept bestellen, damit es in die Warteliste aufgenommen und schließlich von dem Cocktailmixer zubereitet wird. Auf der rechten Seite der Kopfleiste erkennt man, dass aktuell ein Benutzer am System angemeldet ist. Durch antippen der Grafik neben dem Namen gelangt man auf das Benutzerprofil und kann sich vom System abmelden. In Abbildung 4.9 ist eine Desktopansicht der Cocktailauswahlliste dargestellt. Im Vergleich zu der mobilen Ansicht fällt auf, dass die Warteliste nicht eingerückt mit den restlichen Elementen dargestellt wird, sondern einen eigenen Platz erhält. Dadurch wir der vorhandene Platz effektiver ausgenutzt.

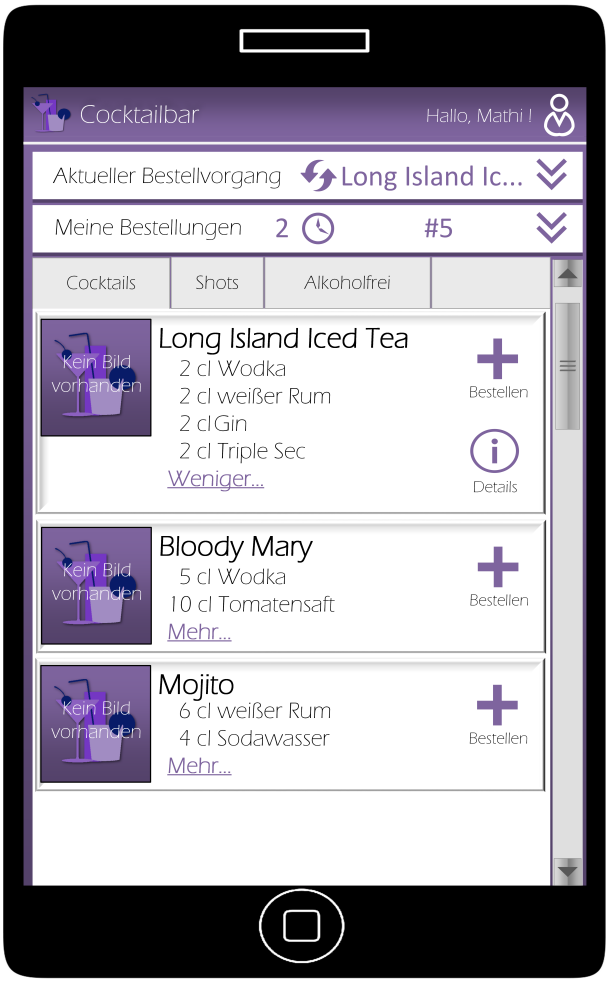

Abbildung 4.8: Prototyp der Cocktailauswahlliste (mobile Ansicht)

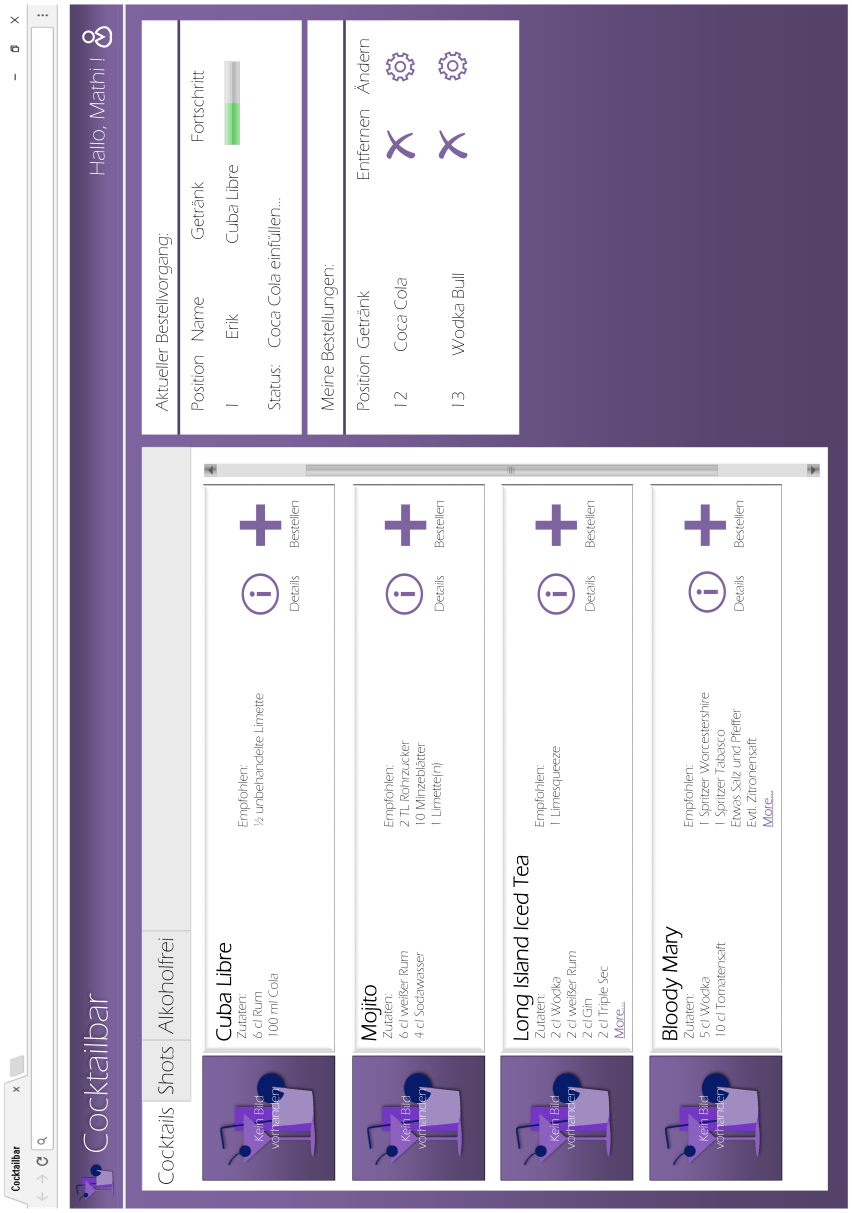

Abbildung 4.9: Prototyp der Cocktailauswahlliste (Desktopansicht)

5 Implementierung

Das Kapitel Implementierung enthält das Vorgehen bei der Entwicklung der Weboberfläche *Cocktailbar*. Der Abschnitt 5.1 erläutert die eingesetzten Werkzeuge und die einzuhaltenden Richtlinien. Um zukünftigen Administratoren und Entwicklern einen Einstieg zu erleichtern, befindet sich in Abschnitt 5.2 ein Handbuch zur Nutzung von *Cocktailbar*. Schließlich befinden sich in Abschnitt 5.3 die Ergebnisse der Implementierung.

5.1 Werkzeuge und Richtlinien

Um die Implementierung von *Cocktailbar* möglichst effizient und geordnet durchzuführen, benötigt es einiger wohl gewählter Werkzeuge und Richtlinien bei der Programmierung. Es ist dabei wichtig festzuhalten, dass kein *richtiges* Werkzeug für einen Anwendungsfall existiert. Viel mehr ist die Wahl des Werkzeugs abhängig von der Gewohnheit und den persönlichen Präferenzen des Entwicklers. Als ein Beispiel könnte man das Schreiben von Code betrachten: ein Entwickler kann den Code in einem auf seinem System vorinstallierten Texteditor schreiben - er spart sich damit den Download von Drittanbietersoftware und hat somit keinen Mehraufwand. Ein anderer Entwickler würde allerdings eine Software für das Schreiben von Code bevorzugen, um so deren Eigenschaften ausnutzen zu können. Folgende Werkzeuge zur Erstellung von Code sind an dieser Stelle nennenswert:

- **Code-Editor** Der Code-Editor stellt zahlreiche Werkzeuge zur Erleichterung der Programmierung und der Verständlichkeit bereit. Dazu gehören Mechanismen wie die Kolorierung von Codesegmenten, deren Formatierung oder die Autovervollständigung von intendiertem Code.
- **Compiler und/oder Interpreter** Compiler übersetzen den gesamten geschriebenen Code in eine gewünschte Zielsprache. Ein wichtiger Teil dieses Vorgangs ist die Syntaxanalyse. Sie zerlegt den Code in einzelne Bestandteile und kann so durch hinterlegte Regeln prüfen, ob es sich um korrekt zusammengesetzte Befehle handelt. Durch diesen Vorgang wird dem Entwickler die Fehlersuche stark vereinfacht. Ein Interpreter führt ebenfalls eine Syntaxanalyse durch, mit dem Unterschied, dass der Code von Ausdruck zu Ausdruck eingelesen wird, statt anfangs den gesamten Code zu verarbeiten. Ob ein Interpreter oder ein Compiler zum Einsatz kommen hängt oft stark von der zu entwickelnden Programmiersprache ab.

- **Build-Automatisierung** Die Build-Automatisierung geht einen Schritt über das Kompilieren oder Interpretieren von Code hinaus und beinhaltet zusätzliche Funktionalität für den Entwicklungsprozess von Software. Einige Mechanismen sind dabei das Verpacken von kompiliertem Code in komprimierte Formate, die Erstellung von Ausführungsdateien für gezielte Betriebssysteme oder die Versionsverwaltung.
- **Debugger** Ein Debugger ist ein Werkzeug zum Testen von Programmcode und der Fehlersuche in diesem. Für den Programmcode können dabei Testfälle geschrieben werden, die dem Entwickler bestenfalls Fehler in der Programmierung erkennbar machen. Debugger erlauben häufig auch das Schrittweise ausführen von Programmcode, womit die Fehlersuche enorm erleichtert werden kann. Mit Hilfe eines Debuggers ist es somit möglich, die eigene Programmierung zu validieren und die Interpretation des Systems nachzuvollziehen.

Für den Einsatz von mehreren Werkzeugen existieren diverse Entwicklungsumgebungen (Integrated development environement (IDE)). Diese IDEs stellen häufig große Mengen an Werkzeugen bereit. Nach dem Plugin-Architekturmuster (vgl. Unterabschnitt 2.4.1) können Entwickler oft eigene Werkzeuge in diese IDEs einfügen.
Eine IDE kommt auch bei der Entwicklung von *Cocktailbar* zum Einsatz. Diese unterstützt die oben genannten Funktionen. Darüber hinaus können beispielsweise Diagramme für ein *Reverse Engineering* erstellt werden, Code aufgeräumt werden oder automatisch Dokumentationen erzeugt werden.

Die Programmierung wird darüber hinaus unter Einhaltung gewisser Richtlinien entwickelt. Diese Richtlinien dienen in erster Linie dazu, die spätere Wartung zu erleichtern und während der Entwicklung Fehler zu vermeiden. Richtlinien, die den Code betreffen, fordern den Programmierer häufig dazu auf, diesen möglichst übersichtlich und standardisiert zu programmieren. Eine für den in Java geschriebenen Server eingesetzte Richtlinie stammt von dem Unternehmen *Google* und nennt sich *Google Java Style Guide* [Goo]. Diese Richtlinie bündelt eine Vielzahl von Regeln, die bei der Entwicklung eingehalten werden sollten. Die Regeln beginnen bereits bei der Erstellung der Datei; der Dateiname ist gleich dem Namen der Klasse, die sich in dieser Datei befinden soll plus der Dateiendung *„.java“*. Eine weitere Regel ist beispielsweise, dass der Programmierer bei *if*-, *else*-, *for*-, *do*- und *while*-Ausdrücken immer eine öffnende und eine schließende Klammer anfügt. Des Weiteren soll eine Zeile nicht mehr als hundert Zeichen enthalten. Auch die Deklaration von Variablen wird reglementiert: Eine lokale Variable wird erst an der Stelle deklariert, an der sie auch einsetzt wird. [Goo]

5.2 Handbuch

Der folgende Abschnitt beschreibt die Einrichtung der Webanwendung *Cocktailbar*.

Um *Cocktailbar* zu betreiben muss sich der Cocktailmixer in Betrieb befinden. Es wird davon ausgegangen, dass auf dem integrierten *Raspberry Pi 3 Modell B* das Betriebssystem *Raspbian* installiert ist. Folgende Pakete sind unter Raspbian zu installieren[1]: java8, python2.7, python-mysql.connector, python-spidev, python-rpi.gpio, mysql-server, npm, nodeJS, yarn.

Nach der Installation lässt sich der Source Code des Servers mit den Befehlen in Listing 5.1 (Zeile 1 und 2) in der Kommandozeile zu einer ausführbaren *JAR*-Datei komprimieren. Nach Ausführung dieses Befehls befindet sich die *JAR*-Datei im Ordner *target* des Basisverzeichnisses. Der Server lässt sich nach Ausführung dieser Schritte für jeden weiteren Start mit dem Befehl in Listing 5.1 (Zeile 3) ausführen:

Listing 5.1: Erstellen einer JAR-Datei und Starten des Servers in der Kommandozeile

```
1 cd /path/to/server
2 mvn package # mvnw package on windows host
3 java -jar serverjarfile.jar
```

Zum Starten des Clients muss zuerst in den Ordner des Source Codes navigiert werden (vgl. Listing 5.2 Zeile 1). Anschließend lädt der Paketmanager mit einem Befehl die benötigten Pakete herunter (vgl. Listing 5.2 Zeile 2). Der folgende Befehl wird nach der Installation ausgeführt kompiliert den Source Code (vgl. Listing 5.2 Zeile 3). Um den Client zu starten wird schließlich der letzte Befehl ausgeführt (vgl. Listing 5.2 Zeile 4).

Listing 5.2: Installieren und Starten des Clients in der Kommandozeile

```
1 cd /path/to/client
2 npm install
3 yarn build -- --release
4 PORT=3001 node build/server.js
```

[1] für das Installieren von Paketen vgl. Abschnitt 2.1

Falls ein Debugging des Maschinencodes erwünscht ist, kann dieser gesondert vom Server ausgeführt werden. Dazu müssen zuerst folgende Zeilen aus der *Main*-Methode des Servers auskommentiert werden:

Listing 5.3: Integrierter Maschinencode des Servers

```
1 try {
2  Process p = Runtime.getRuntime().exec("python machine/start-init.py");
3 } catch (IOException e) {
4  e.printStackTrace();
5 }
```

Schließlich kann der Maschinencode manuell mit folgendem Befehl gestartet werden:

Listing 5.4: Manuelles Starten des Maschinencodes

```
1 cd /path/to/server/src/main/java/de/isw/unistuttgart/machine
2 python start-init.py &
```

5.3 Resultate

Dieser Abschnitt präsentiert die Resultate der Implementierung der Webanwendung *Cocktailbar*. Hierfür wird die Webanwendung mit ihren Anforderungen (vgl. Abschnitt 3.2) abgeglichen und Anhand relevanter Abbildungen beschrieben.

Abbildung 5.1: Cocktailbar - Rezept

Abbildung 5.1 zeigt die Anzeige eines einzelnen Rezepts. Jedes im System vorhandene Rezept lässt sich auf diese Weise über eine eigene URL ansteuern und anzeigen (vgl. Anforderung R1). Des Weiteren werden sämtliche Zutaten, die in dem Rezept enthalten sind dem Benutzer angezeigt (vgl. Anforderung R2). Ebenso enthält die Rezeptansicht ein Bild des Rezepts und weitere Informationen über das Rezept (vgl. Anforderung R3 und R4). Über die Taskleiste unterhalb des Rezepts kann der Benutzer das angezeigte Rezept bestellen (vgl. Anforderung R5). Die Taskleiste und die Zurück-Schaltfläche sind über die gesamte Anwendung gleich gehalten. Dies erfüllt die goldene Regel der Konsistenz von Shneiderman (vgl. Unterabschnitt 2.4.3. Ebenso wird die Regel der informativen Rückmeldung erfüllt, wenn man eine Bestellung tätigt: nach der Interaktion mit der Schaltfläche öffnet sich ein Dialog, der den Benutzer informiert, ob die Bestellung erfolgreich im System eingegangen ist.

Abbildung 5.2: Cocktailbar - Rezeptliste

Abbildung 5.2 zeigt die Rezeptliste der Webanwendung *Cocktailbar* (vgl. Anforderung RL1). Da mehr als fünf Rezepte im System hinterlegt sind, wird im oberen (und im unteren) Bereich der Liste eine Auswahl der Seite angeboten, sodass die Rezepte über mehrere Seiten hinweg verteilt werden können (vgl. Anforderung RL2). Dies erfüllt die goldene Regel Shneidermans über die Entlastung des Kurzzeitgedächtnis (vgl. Unterabschnitt 2.4.3). Ebenfalls befindet sich am oberen Rand der Liste ein Kontrollkästchen, mit dem die Zutaten der Rezepte ein- und ausgeblendet werden können (vgl. Anforderung RL3). Wie auch in der einzelnen Rezeptansicht wird ein Bild für jedes Rezept der Liste angezeigt (vgl. Anforderung RL4). Mit Hilfe der Taskleiste unter jedem Rezept kann der Benutzer in die Einzelansicht des Rezepts gelangen oder das Rezept bestellen (vgl. Anforderung RL5 und RL6). Die Bündelung sämtlicher Interaktionsmöglichkeiten und somit deren kombinierter Wahrnehmung in der Taskleiste beruht auf dem Gesetz der Nähe (vgl. Unterabschnitt 2.4.3).

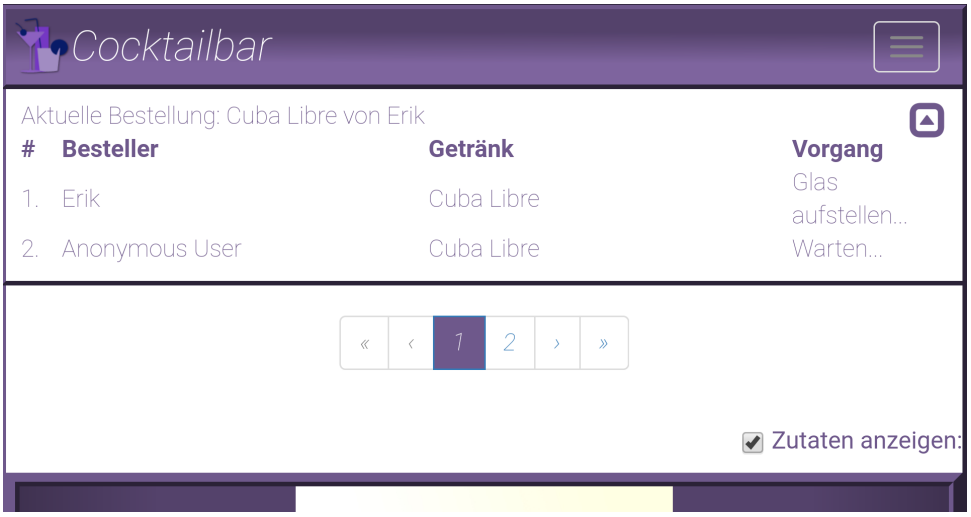

Abbildung 5.3: Cocktailbar - Warteliste

Die vorherigen Abbildungen zeigen bereits die realisierte Warteliste in ausgeblendetem Zustand. Abbildung 5.3 zeigt die Liste, wenn der Benutzer sie durch die Interaktion mit der Schaltfläche eingeblendet hat (vgl. Anforderung W1 und W2). Die fünf obersten Bestellungen werden mit ihrem Rezept und Besteller nach der Reihenfolge ihrer Abarbeitung aufgezählt (vgl. Anforderung W3). Die oberste Bestellung ist dabei auch im ausgeblendeten Zustand ersichtlich. Nach Ausführung einer Bestellung wird diese aus der Warteliste entfernt werden. Dafür wird der aktuelle Stand der Warteliste in kurzem Abstand über den Server abgefragt (vgl. Anforderung W4 und W5). Da die Warteliste, so wie auch die Rezepte und Zutaten, jeweils innerhalb eines Quadrats abgebildet sind, werden deren Inhalte auch als zusammengehörig wahrgenommen (vgl. Unterabschnitt 2.4.3, Gesetz der Geschlossenheit).

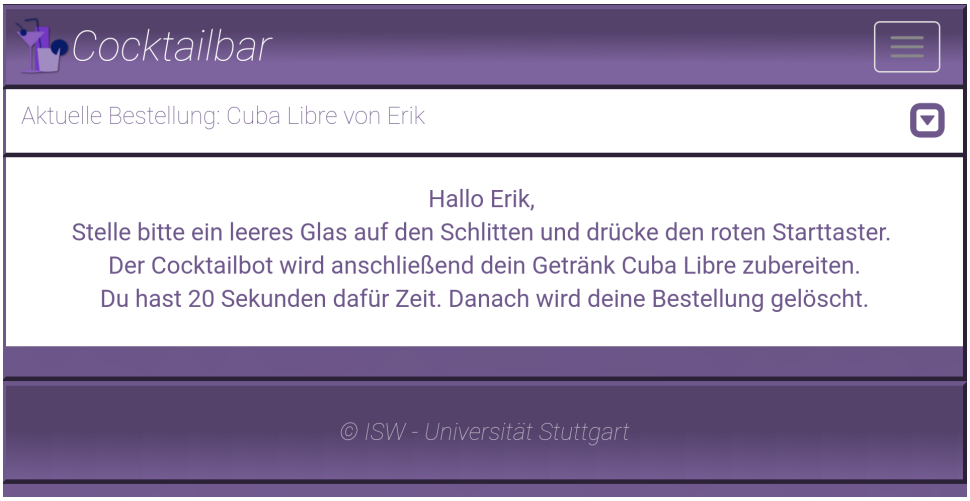

Abbildung 5.4: Cocktailbar - Startseite

Abbildung 5.4 zeigt die Startseite, welche Informationen über die aktuellen Vorgänge des automatisierten Cocktailmixers enthält. Die Startseite ist dabei nicht die Eingangsseite, auf die man gelangt, wenn *Cocktailbar* zum ersten Mal aufgerufen wird (dies ist die Rezeptliste). Das Display des automatisierten Cocktailmixers soll dagegen diese Startseite ständig anzeigen[1], um dem Benutzer, der an der Reihe ist mitzuteilen, welche Schritte er zu befolgen hat (vgl. Anforderung S1, S2 und S3).

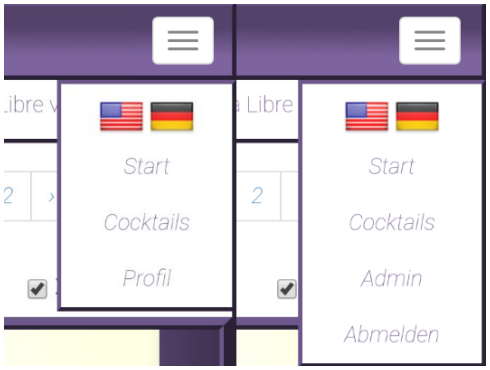

Abbildung 5.5: Cocktailbar - Kopfleisten

[1]Die übrige Webseite ist über das Display dennoch aufrufbar.

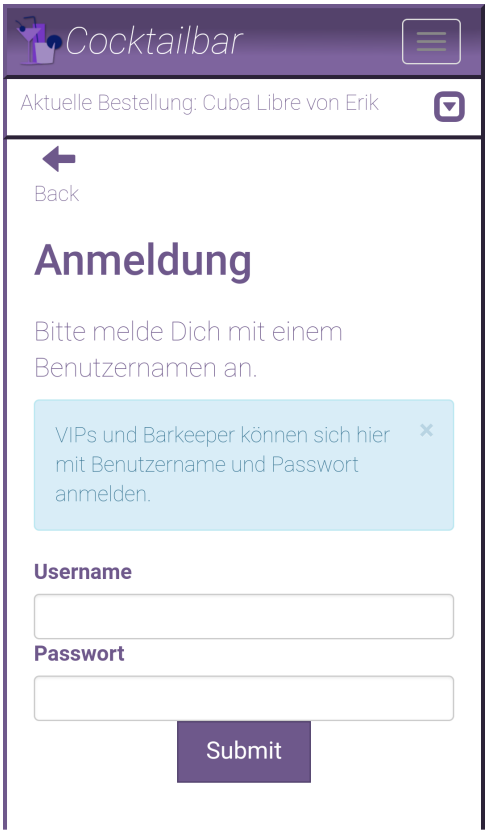

Abbildung 5.6: Cocktailbar - Login

Auf Abbildung 5.6 ist der Login-Bereich der Webanwendung *Cocktailbar* zu sehen. Unter Eingabe eines Benutzernamens und Bestätigung mit Hilfe der Schaltfläche interagiert der Benutzer mit dem eingegebenen Pseudonym (Ist kein Benutzername angegeben wird das Pseudonym „*Anonymous User*“ benutzt). Falls der Benutzer, zusätzlich zu seinem Benutzernamen, noch ein Passwort angibt, prüft das System, ob mit dieser Kombination aus Benutzername und Passwort ein administrativer Account hinterlegt ist. Ist dies der Fall, kann der Benutzer fortan als Administrator interagieren. (vgl. Anforderung B1 und B2)
Abbildung 5.5 zeigt eine unterschiedliche Ansicht der Kopfleiste; einmal wenn ein Benutzer am System angemeldet ist und einmal wenn ein Benutzer sich am System anmelden kann. Ist ein Benutzer angemeldet kann er über den Link „Ausloggen“ sämtliche erhaltenen Privilegien ablegen und fortan wieder unter dem Pseudonym „*Anonymous User*“ interagieren (vgl. Anforderung B3). Die Bündelung der Links in einer Menüleiste entspricht dem Gesetz der Nähe (vgl. Unterabschnitt 2.4.3) und vereinfacht dem Benutzer, durch die Webseite zu navigieren.

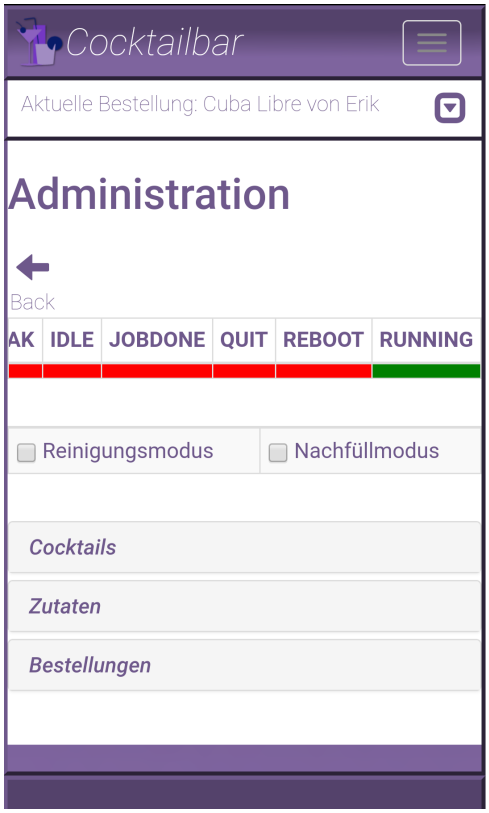

Abbildung 5.7: Cocktailbar - Administrationsbereich

Hat sich ein Benutzer mit einem administrativen Account am System angemeldet, erreicht er über die Kopfleiste (vgl. Abbildung fig:cockkopf) den Administrationsbereich (vgl. Anforderung A1). Auf dieser Seite kann der Benutzer im oberen Bereich durch alle aktuellen Zustände des Systems hindurchscrollen und so, auch ohne Sicht auf die Maschine, feststellen, ob diese beispielsweise in Betrieb ist, gerade ein Rezept zubereitet oder auf die Entnahme des Glases wartet (vgl. Anforderung A2). Unter der Statusleiste befindet sich die Serviceleiste. Mit dieser lässt sich der Reinigungsmodus und der Nachfüllmodus aktivieren. Wird der Nachfüllmodus aktiviert, pausiert das System die Abarbeitung der Bestellungen, bis dieser wieder deaktiviert wird. Bei aktivem Reinigungsmodus können Zutaten, die sich an Ventilen befinden über die Taskleiste gereinigt werden - Die Ventile öffnen sich nach Betätigung der Schaltfläche für eine gewisse Zeit (vgl. Anforderung A16). Unterhalb der Statusanzeige können drei Reiter ausgeklappt werden, in denen die administrative Ansicht der Rezept-, Zutaten- und Bestellliste angezeigt wird (vgl. Anforderung A3, A8 und A13). Mit Hilfe der Taskleiste unter den Rezepten und Zutaten lassen sich hier Rezepte und Zutaten löschen und bearbeiten (vgl. Anforderung A5,A7, A11 und A12). Dabei können nur Zutaten gelöscht werden, die sich in keinem Rezept mehr befinden. Ebenfalls kann sich eine Zutat keinen Anschluss mit einer weiteren Zutat teilen (vgl. Anforderung A6). Eine Zutat kann darüber hinaus über die Taskleiste nachgefüllt werden; ihr Füllstand wird im unteren Bereich des Rezepts mit Hilfe eines Fortschrittsbalkens

angezeigt (vgl. Anforderung A7). Im linken, oberen Bereich der Rezepte und der Zutaten kann der Benutzer über eine Schaltfläche neue Rezepte (bzw. neue Zutaten) zu dem momentanen Bestand hinzufügen (vgl. Anforderung A4 und A10). In der administrativen Bestellliste können entweder einzelne oder alle Bestellungen aus dem System gelöscht werden (vgl. Anforderung A14 und A15).

6 Schlussbetrachtung

In diesem Kapitel schließt die Arbeit über die Konzeption und Realisierung einer Steuerungssystem-Human Machine Interface (HMI) für Mobilgeräte mit einer Zusammenfassung. In dieser werden insbesondere die gesetzten Ziele mit dem Inhalt dieser Arbeit verglichen. Ebenso befindet sich in diesem Kapitel ein Ausblick, der die mögliche Weiterentwicklung der Webanwendung *Cocktailbar* umschreibt. Hierfür werden auch Ideen und Konzepte vorgeschlagen, die in dieser Arbeit nicht realisiert wurden.

6.1 Zusammenfassung

Diese Arbeit begann mit der Recherche über den Stand der Technik. Dabei wurden sechs Architekturmuster vorgestellt sowie deren Vor- und Nachteile aufgezählt. Ebenso wurde über das Kommunikationsparadigma REST recherchiert. Schließlich wurden etablierte Gesetze und Richtlinien der Benutzerfreundlichkeit ergründet. Im Anschluss daran folgte die Analyse des bestehenden Systems. Dazu wurde dieses mit den Ergebnissen der Recherche abgeglichen und auf Mängel hin überprüft. Daran anschließend wurden funktionale, nichtfunktionale und sonstige Anforderungen an das System in einem Anforderungskatalog gesammelt und die Arbeit von existierenden Arbeiten abgegrenzt. Der Analyse als Grundlage folgte die Konzeption der Webanwendung *Cocktailbar* in Form eines Systementwurfs und dessen Prototypen. Innerhalb des Systementwurfs konnten die Ergebnisse des Stands der Technik wiederverwendet werden. Ebenso wurden die Anforderungen aus dem Analyseteil berücksichtigt. Als Ergebnis des Entwurfs wurden seine Teilkomponenten ausführlich aufgezählt und in ihrer Funktion beschrieben. Der Prototyp diente einer ersten Impression des fertig entworfenen Systems. Durch ihn wurden die Entwurfsentscheidungen überprüft und gegebenenfalls ergänzt oder beschnitten. Der abschließende Teil dieser Arbeit schilderte die Implementierung des Softwareentwurfs. Dabei wurden Rahmenbedingungen wie eingesetzte Werkzeuge und Richtlinien erläutert. Ebenso wurde ein Handbuch angefügt, dass bei der weiteren Entwicklung und Benutzung der Webanwendung *Cocktailbar* unterstützen soll. Im letzten Teil des Implementierungsabschnitts wurden schließlich die praxisorientierten Ergebnisse der Arbeit präsentiert und mit den Anforderungen des Analyseteils abgeglichen.

6.2 Ausblick

Die Webanwendung *Cocktailbar* stellt ein Fundament für die Weiterentwicklung der Softwareseite des automatischen Cocktailmixers da. Dabei beschränkt sich *Cocktailbar* auf die Benutzerschnittstelle und lässt den Maschinencode weitgehend unberührt. Ein erster Schritt könnte die Angliederung des Maschinencodes an den Server darstellen. Da die Komponente in der Verarbeitungsschicht des Servers einzuordnen ist und ihre Funktionalität der des *controller*-Pakets des Servers gleicht, ist eine lose Kopplung hier nicht sinnvoll. Beide Komponenten sind darüber hinaus in unterschiedlichen Programmiersprachen entwickelt, was die Wartung erschwert. Die Webanwendung *Cocktailbar* selbst könnte um einige Funktionen erweitert werden, welche die Interaktion mit dem System interessanter gestalten könnten. Dazu gehört beispielsweise die Möglichkeit, Rezepte zufällig zu bestellen (vgl. Funktionalität von Barobot in Unterabschnitt 3.3.2). Ebenso könnte man Rezepte von den Benutzern selbst zusammenstellen lassen und so nach internen oder benutzerdefinierten Rezepten filtern. Die Statusmeldungen, die der Maschinencode bisher ausgibt, könnten darüber hinaus erweitert werden. Man könnte dazu die aktuellen Schritte des Cocktailmixers genauer anzeigen lassen (z. B. „Fülle Orangensaft ein...“).
Das System *Cocktailbar* wurde so entwickelt, dass zukünftig noch weitere Programmierer an ihm weiterarbeiten können - Die Entwicklung des automatisierten Cocktailmixers hat mit dieser Arbeit einen neuen Baustein hinzugewonnen.

Literaturverzeichnis

[Bara] Bartendro™. *Automatisierter Cocktailmixer*. URL: http://partyrobotics.com/ (zitiert auf S. 49).

[Barb] Bartendro™. *Cocktail-Auswahlbildschirm der GUI*. URL: http://cdn.shopify.com/s/files/1/0455/5013/files/blog_UI-1024x703.jpg?259 (zitiert auf S. 49).

[Barc] Bartendro™. *GNU General Public License*. URL: https://github.com/partyrobotics/bartendro/blob/master/COPYING (zitiert auf S. 49).

[Bar13a] Barobot. *Source Code*. 2013. URL: https://code.google.com/archive/p/barobot/source/default/source (zitiert auf S. 50).

[Bar13b] Barobot. *Technische Details*. 2013. URL: https://code.google.com/archive/p/barobot/wikis/TechnicalDetails.wiki (zitiert auf S. 50).

[Bar14a] Barobot. *Automatisierter Cocktailmixer*. 2014. URL: https://www.kickstarter.com/projects/barobot/barobot-a-cocktail-mixing-robot (zitiert auf S. 50).

[Bar14b] Barobot. *Cocktail-Auswahlbildschirm der GUI*. 2014. URL: http://cdn.firstwefeast.com/assets/2014/05/barobotscreen.jpg (zitiert auf S. 50).

[BCK03] L. Bass, P. Clements, R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 2003 (zitiert auf S. 21).

[Cur04] E. Curry. „Message-oriented middleware“. In: *Middleware for communications* (2004), S. 1–28 (zitiert auf S. 33).

[Deb] Debian. *Offizielle Webseite*. URL: https://www.debian.org/ (zitiert auf S. 15).

[Elm08] R. Elmasri. *Fundamentals of database systems*. Pearson Education India, 2008 (zitiert auf S. 64).

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. United States, 1999 (zitiert auf S. 33).

[GD13] J. Goll, M. Dausmann. *Architektur-und entwurfsmuster der softwaretechnik*. Bd. 1. Springer, 2013 (zitiert auf S. 21–32).

[Goo] Google. *Google Java Style Guide*. URL: https://google.github.io/styleguide/javaguide.html (zitiert auf S. 74).

[Has06] W. Hasselbring. „Software-Architektur“. In: *Informatik-Spektrum* 29.1 (2006), S. 48–52. ISSN: 1432-122X. DOI: 10.1007/s00287-005-0049-5. URL: http://dx.doi.org/10.1007/s00287-005-0049-5 (zitiert auf S. 21).

[Her16] V. Hermann. „Konzeption und Entwicklung eines web- und datenbankbasierten Steuerungssystems für einen automatisierten Cocktailmischer“. Bachelorarbeit. Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen, Universität Stuttgart, 2016 (zitiert auf S. 17, 19).

[Kic] Kickstarter. *Projektfinanzierung über Crowdfunding*. URL: https://www.kickstarter.com (zitiert auf S. 49, 50).

[LV12] G. van Loo, M. VanInwegen. *Gertboard Benutzerhandbuch*. 2012 (zitiert auf S. 16, 17).

[Mil56] G. A. Miller. „The magical number seven, plus or minus two: some limits on our capacity for processing information.“ In: *Psychological review* 63.2 (1956), S. 81 (zitiert auf S. 35).

[Nie03] J. Nielsen. *Usability 101: Introduction to usability*. 2003 (zitiert auf S. 34).

[Par14] Party Robotics. *Getting Started Guide*. 2014 (zitiert auf S. 49).

[Rasa] Raspberry Pi. *Dokumentation GPIO*. URL: https : / / www . raspberrypi . org / documentation/usage/gpio/ (zitiert auf S. 15).

[Rasb] Raspberry Pi. *Frequently Asked Questions*. URL: https://www.raspberrypi.org/help/faqs/ (zitiert auf S. 14).

[Rasc] Raspbian OS. *Offizielle Webseite*. URL: https://www.raspberrypi.org/downloads/raspbian/ (zitiert auf S. 15).

[RJB04] J. Rumbaugh, I. Jacobson, G. Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004. ISBN: 0321245628 (zitiert auf S. 55).

[Rod08] A. Rodriguez. „Restful web services: The basics“. In: *IBM developerWorks* (2008) (zitiert auf S. 33).

[RPi] RPi.GPIO. *Package Webseite*. URL: https://pypi.python.org/pypi/RPi.GPIO (zitiert auf S. 17).

[Shn10] B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India, 2010 (zitiert auf S. 35, 38).

[Wer23] M. Wertheimer. „Untersuchungen zur Lehre von der Gestalt. II“. In: *Psychologische forschung* 4.1 (1923), S. 301–350 (zitiert auf S. 36).

[Wir] WiringPi. *Package Webseite*. URL: https://pypi.python.org/pypi/wiringpi/2.44.0 (zitiert auf S. 17).

Alle URLs wurden zuletzt am 28. 05. 2017 geprüft.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings