POSEIDON

SOFTWAREARCHITEKTUR ZU “REFILL”

Hochschule Mannheim

Fakultät für Informatik

Informatik Master

Sommersemester 2024
02.07.2024

Interne Projektbetreuer: Prof. Dr. Peter Knauber, Prof. Kirstin Kohler

Externe Betreuer: Steffen Heß, Balthasar Weitzel

Versionshistorie

Version	Datum	Änderungen	Autoren
1.0	14.06.24	Veröffentlichung erste Version	Alle
1.1	01.07.24	Überarbeitet Fassung nach Review	Alle

Autoren

Vorname	Nachname	Kontaktmöglichkeiten
Jonas	Blum	3011057@stud.hs-mannheim.de
Heiko	Michel	1813589@stud.hs-mannheim.de
David	Miller	2011708@stud.hs-mannheim.de
Yusuf Can	Özdemirkan	1813055@stud.hs-mannheim.de
Alejandro	Restrepo Klinge	3009557@stud.hs-mannheim.de

1 Inhaltsverzeichnis

2 Einleitung

Dieses Architekturdokument dient als umfassende Darstellung der Architektur für die Anwendungen, welche im Rahmen der Vorlesung PSE2 des Masterstudienganges „Informatik – Software-Engineering“ an der Hochschule Mannheim in Kooperation mit dem Fraunhofer IESE konzipiert wurde. Die Vorlesung PSE1 hat sich mit der Problematik beschäftigt, für eine von der Fraunhofer IESE gestellte Challenge ein Konzept zu entwerfen, testen, verfeinern und als Pflichtenheft zusammenzufassen.

Dieses Architekturdokument befasst sich mit der Umsetzung des zuvor entwickelten Konzepts, was im Rahmen der Lehrveranstaltung PSE1 stattgefunden hat. Das Hauptziel dieses Dokuments ist es, einen Einblick in die strukturellen Grundlagen und technologischen Richtlinien zu bieten, die die Grundlage der Anwendung bilden.

Das Dokument gliedert sich so: Anfangs werden der Rahmen und Ursprung des Projekts sowie das Konzept der im Zusammenhang damit erarbeiteten Anwendung präsentiert. Danach werden die Designentscheidungen als Architekturtreiber dargestellt. Anschließend wird die Architektur aus verschiedenen Sichten präsentiert und erläutert.

Selbstverständlich werden stets Personen jeglicher Geschlechtsidentität angesprochen. Zuletzt wird sich für die Struktur des Dokuments an eine modifizierte Version des Architekturtemplate der Vorlesung SWA gehalten. Die Vorlesung wurde hauptverantwortlich gehalten von Jens Knodel, wissenschaftlicher Mitarbeiter des Fraunhofer IESE.

2.1 Projektziele

Im Jahre 2012 wurde von den Vereinten Nationen die Post-2015-Entwicklungsagenda eingeleitet. Ziel dieses Programmes ist die weltweite nachhaltige Entwicklung, welche ausgeschrieben als “Sustainable Development Goals” abgekürzt als SDGs bezeichnet werden. Insgesamt wurden 17 solcher SDGs thematisch definiert. Das übergeordnete Ziel der einzelnen SDGs ist es, auf globaler Ebene eine nachhaltige Lebensweise zu schaffen. Die SDGs beziehen sich dabei sowohl auf ökologische als auch soziale nachhaltige Ziele.

Die hier vorgestellte Arbeit basiert auf einem neu entwickelten Konzept, welches im Wintersemester 23/24 in der Vorlesung PSE1 an der Hochschule Mannheim von einer anderen Arbeitsgruppe entwickelt wurde. Die Challenge, für welches das Konzept entwickelt worden ist, lautete wie folgt:

„Wie kann man den Bürgerinnen und Bürgern den Beitrag der Stadt Kaiserslautern zu den Sustainable Development Goals auf spielerische, ansprechende Weise verständlich machen? Dabei gehört es auch zu Ihren Aufgaben, sich zu überlegen, wie sich aus den gegebenen Daten der Beitrag zu den SDGs ableiten lässt.“ - Steffen Heß

Das Projektziel dieser Arbeit ist es, die Anforderungen und Spezifikationen des Pflichtenheftes, welches im vorherigen Semester entworfen wurde, auf eine Umsetzbarkeit zu überprüfen. Hierbei sollen gegebenenfalls auch kritische Punkte identifiziert und verbessert werden. Als Endergebnis soll das dort entwickelte Konzept in der Praxis umgesetzt sein und dem Kunden, die Fraunhofer IESE, in Form eines Demonstrators übergeben werden können. So soll als übergeordnetes Ziel die Verbreitung und das Bewusstsein für die SDGs weiterverbreitet werden.

2.2 Abgrenzung

Das Projekt und Produkt werden, wie im vorherigen Absatz erwähnt, für einen Demonstrator entwickelt. Die Architektur, welche hierfür entwickelt wird, ist für eine skalierbaren Rollout entwickelt worden. Hierbei wurde das System für eine Last von 1000 gleichzeitigen Nutzern ausgelegt. Ein Feldversuch mit einer Vielzahl an realen Nutzern und mehreren Hardwaregeräten wurde im Rahmen des Projektes nicht durchgeführt.

Des Weiteren wird in Abstimmung mit dem Kunden dieses Projektes in diesem Konzept nur Ausblicke für ein Accountmanagement gegeben und dieses nicht aktiv implementiert. Des Weiteren findet keine Betrachtung der API Sicherheit statt. Im Kapitel Risiken / Offene Punkte wird hierzu detaillierter eingegangen.

2.3 Stakeholders

Die Stakeholder sind weiterhin die bereits im Anforderungsdokumentation zum Projekt REFILL-KL von Team Gaia genannten Stakeholder, welche basierend hierauf erneut genannt werden.

Bei den Stakeholdern wird unterschieden zwischen der Hauptzielgruppe des Produktes (“Core Target Group”), der direkten Stakeholdern des Produkts und den indirekten Stakeholdern.

Zur **Core Target Group** zählen die jungen Bürger der Stadt Kaiserslautern, welche repräsentativ für alle jungen Menschen dieser Altersgruppe in anderen Städten wie Kaiserslautern gelten. Sie sollen die Hauptnutzer des Produkts sein.

Als **direkte Stakeholder** gilt zum einen die Stadtverwaltung Kaiserslautern, welche bei einem weiteren Rollout des Konzepts als Verwaltung angedacht ist. Außerdem das Fraunhofer IESE, welches als Auftraggeber fungiert. Zudem die Wirtschaft vor Ort, da im Falle der gewünschten veränderten Nachhaltigkeitsbewusstsein der Bürger ein verändertes Konsumverhalten eingehen könnte.

Als **indirekte Stakeholder** sind besonders die Landespolitik und der Staat anzusehen, da diese gemeinsam 30 % des Auftraggebers, die Fraunhofer-Gesellschaft, finanzieren. Zusätzlich haben diese Stakeholder eine politische Motivation, in der Erfüllung der SDGs, zu welchem die Mitgliedstaaten der UN verpflichtet sind.

3 Systemübersicht

Die Refill-App Kaiserslautern, welche als Ergebnis der Challenge konzipiert wurde, ermöglicht das Finden von Wasserstationen und die Ausgabe von Wasser an diesen Wasserstationen. Diese Wasserstationen werden im Rahmen dieses Dokuments Refill-Stationen genannt. Der Name leitet sich von der Idee ab, eine leere Flasche wieder kostenlos nachzufüllen, was im Englischen mit “refill” übersetzt wird. Das Produkt, dessen Architektur in diesem Dokument vorgestellt wird, hat folgenden Mehrwert und Funktion:

Der App-Nutzer ist in der Lage, alle Refill-Stationen in seiner Nähe auf einer Karte zu sehen und alle Details zu den jeweiligen Stationen aufzurufen. So kann der Nutzer, bevor er sich zu einer Station bewegt, beispielsweise den Zustand und die Öffnungszeiten überprüfen.

Befindet sich der App-Nutzer an einer Refill-Station, kann er in seiner App seine Wasserwünsche, bestehend aus Wassermenge und Wasserart (Still / Sprudel) angeben. Die gewählten Präferenzen können anschließend kontaktlos mithilfe eines am Smartphone angebrachten NFC-Chips (Near Field Communication) an eine Refill-Station übergeben werden. Ein NFC-Chip ist ein kleiner Funkchip, der es zwei Geräten ermöglicht, kabellos Daten auszutauschen, wenn sie sich in unmittelbarer Nähe befinden. Die Wasserausgabe startet anschließend. Kommt es hierbei zu einem Fehler, kann der Nutzer eine Rückmeldung beziehungsweise einen Fehlerbericht senden.

Generell können die Refill-Stationen bewertet werden, sodass die Nutzer pro Refill-Station eine gesamte Bewertung der Community einsehen können.

Dank der smarten Funktionen der App und der Refill-Station, sieht der Nutzer jederzeit, wie viel Wasser er bereits entnommen hat und wie seine Auswirkungen auf die Umwelt im Vergleich zu einem Neukauf einer Flasche ist. Diese einzelnen Benutzerstatistiken können auch gebündelt als Community Auswirkung in der App eingesehen werden.

Jeder App-Nutzer kann außerdem beliebig viele sogenannte smarte Flaschen in seinem Account anlegen. Hierfür benötigt er einen beschreibbaren NFC-Chip. Mithilfe der App kann er diesen mit seinem Account verknüpfen und anschließend an eine reale Flasche kleben. In der App hat er dann die Möglichkeit für diese smarte Flasche genau zu hinterlegen, welche Wasser Präferenzen gewünscht sind. Einmal hinterlegt, kann mithilfe des NFC-Chips, die Flasche ohne Smartphone an allen Refill-Stationen verwendet werden, und die hinterlege gewünschte Wasserpräferenzen wird automatisch ausgeben. Durch die Verbindung mit der App kann der App-Nutzer jederzeit seine Präferenzen für jede smarte Flasche ändern.

Zur Bereitstellung der oben genannten Dienste besitzt das System eine Verwaltung-Web-App auch Dashboard genannt. Dort können die Refill-Stationen in das System eingepflegt und hinterlegte Daten aktualisiert oder gelöscht werden. Auch können die von dem Nutzer gemeldeten Probleme und Hinweise zu den jeweiligen Refill-Stationen eingesehen werden. Als dritte Funktion kann in der Verwaltung-Web-App Statistiken zu den jeweiligen Wasserstationen eingesehen werden. Alternativ zu den vordefinierten Statistiken können die Daten im Dashboard exportiert werden und mit einem externen Large-Language Modell analysiert werden. Die Verwaltung-Web-App steht im Gegensatz zur Refill-App nur einem kleinen begrenzten Personenkreis zur Verfügung.

3.1 Design Prinzipien

Bei der Entwicklung der Refill-App ist es entscheidend, grundlegende Designprinzipien zu beachten, um eine konsistente, benutzerfreundliche und erweiterbare Software zu gewährleisten.

Die Refill-App legt großen Wert auf Konsistenz sowohl in der Benutzeroberfläche als auch in der Code-Struktur. Ein einheitliches Layout und Design der Benutzeroberfläche, einschließlich der Verwendung konsistenter Farben, Schriftarten und Schaltflächenstile, trägt zur intuitiven Bedienbarkeit bei. Ebenso soll der Code einheitlich strukturiert sein, um die Lesbarkeit und Wartbarkeit zu verbessern, indem Namenskonventionen eingehalten und der Code in logisch zusammenhängende Module unterteilt wird. Material 3[1] wird als Standardvorlage für die Design Guidelines verwendet.

Ein weiteres zentrales Prinzip ist das benutzerzentrierte Design. Die Benutzeroberfläche der Refill-App soll intuitiv und einfach zu bedienen sein, wobei regelmäßig Benutzerfeedback eingeholt und in die Weiterentwicklung einbezogen wird. Diese Rückmeldungen helfen dabei, die Benutzerfreundlichkeit kontinuierlich zu verbessern.

Modularität und Wiederverwendbarkeit sind ebenfalls essenziell für die Architektur der Refill-App. Die App ist in klar abgegrenzte Module unterteilt, die unabhängig voneinander entwickelt und getestet werden können. Dies fördert die Wiederverwendbarkeit von Code-Komponenten in verschiedenen Teilen der App und erleichtert zukünftige Erweiterungen und Anpassungen.

Schließlich ist die Performanz ein entscheidender Faktor. Der Code der Refill-App soll effizient geschrieben sein, um die Systemressourcen optimal zu nutzen und eine hohe Leistung sicherzustellen. Dies umfasst die Optimierung der Ausführungsgeschwindigkeit und die Minimierung des Speicherverbrauchs, um eine reibungslose und schnelle Benutzererfahrung zu gewährleisten.

Durch die Beachtung dieser Designprinzipien und die Anwendung der Clean Code-Praktiken soll sichergestellt werden, eine robuste und benutzerfreundliche Refill-App zu entwickeln, die den Anforderungen des Projekts gerecht wird und eine nachhaltige Nutzung fördert.

3.2 Coding Guidelines

Die Einhaltung von klaren und einheitlichen Richtlinien ist entscheidend für die Lesbarkeit, Verständlichkeit und Wartbarkeit des Codes der Refill-App. Diese Richtlinien basieren auf bewährten Praktiken und sind speziell auf die Bedürfnisse der Refill-App zugeschnitten.

Kommentare sind unerlässlich. Jede Funktion und Methode sollte mit Kommentaren versehen sein, die erklären, was der Code tut und warum. So können auch neue Entwickler schnell nachvollziehen, wie die Refill-App funktioniert und Änderungen leichter umsetzen.

Namenskonventionen spielen eine wichtige Rolle. Variablen, Konstanten, Funktionen und Methoden sollten in englischer Sprache und nach gängigen Konventionen wie CamelCase oder snake_case benannt werden. Dies sorgt für Konsistenz und erleichtert das Verständnis des Codes.

[1] https://m3.material.io/

Die objektorientierte Programmierung (OsOP) sollte, wo sinnvoll, eingesetzt werden. OOP hilft, den Code in wiederverwendbare und modular aufgebaute Einheiten zu strukturieren. Dadurch bleibt die Refill-App flexibel und erweiterbar.

Für die in der Refill-App verwendeten Programmiersprachen gelten spezifische Richtlinien:

- Flutter: Es gelten die Coding-Guidelines des Flutter-Repos (siehe Flutter Style Guide).
- Golang: Es gelten die Coding-Guidelines des Golang-Repos (siehe Golang Style Guide).
- Python: Es gelten die Coding-Guidelines der Programmiersprache (Siehe Python Style Guide).

Ein weiterer wichtiger Punkt ist die Nutzung von Git als Versionskontrollsystem. Regelmäßige Commits mit aussagekräftigen Nachrichten helfen, Änderungen nachzuvollziehen und effektiv im Team zusammenzuarbeiten. Diese Praxis unterstützt die kontinuierliche Integration und erleichtert das Management von Codeänderungen.

Durch die Beachtung dieser Coding-Guidelines wird sichergestellt, dass der Code der Refill-App sauber und wartungsfreundlich bleibt. So kann die App kontinuierlich verbessert und weiterentwickelt werden, um den Anforderungen der Nutzer gerecht zu werden.

4 Architekturtreiber

In den nachfolgenden Unterkapiteln werden die Hauptmerkmale sowie die zentralen Qualitätsmerkmale erläutert. Hierbei liegt der Fokus darauf, die wichtigen Eigenschaften und Funktionen des Systems eingehend zu beschreiben.

4.1 Hauptmerkmale

ID	Name	Beschreibung
1	Schnelle und transparente Wasserentnahme mittels App	Die Wasserentnahme kann intuitiv mit wenigen Klicks mit einer App gestartet werden
2	Wasserentnahme mit smarter Wasserflasche möglich	Die Wasserentnahme kann mit einer smarten Flasche gestartet werden
3	Wasserentnahme ohne Smartphone möglich	Die Wasserentnahme an einer Refill-Station kann vor Ort manuell ohne Smartphone oder smarter Flasche gestartet werden
4	Refill-Station finden	Der Nutzer kann Refill-Stationen auf einer Karte finden.
5	Echtzeitdaten der Wasserentnahmen und Zustand der Refill-Stationen	Die Statistik der Wasserentnahmen steht immer in Echtzeit zur Verfügung, ebenso der Zustand und die Informationen der Stationen
6	Einfache Verwaltung der Wasserstationen	Die Verwaltung kann Daten der Refill-Stationen einsehen und bearbeiten

Tabelle 1: Architekturtreiber - Hauptmerkmale

4.1.1 Schnelle und transparente Wasserentnahme mittels App

Treibername	Schnelle und transparente Wasserentnahme mittels App	
Treiber-ID	1	
Ablauf	**Beschreibung**	**Quantifizierung**
Umgebung	Der Nutzer hat die Refill-App bereits auf seinem Smartphone installiert. Der Nutzer befindet sich vor einer smarten Refill-Station. Der Nutzer hat einen NFC-Chip an seinem Smartphone angebracht und in der App hinterlegt	
Stimulus	Der Nutzer wählt seine Wasserpräferenzen aus und hält sein Smartphone mit dem angebrachten NFC-Chip an das NFC-Lesegerät der Wasserstation	● Bisherige Starts der App >= 1 ● Smartphone unterstützt NFC ● Smartphone besitzt aktive Netzwerkverbindung
Antwort	Refill-Station gibt Wasser aus, Eintrag wird in den Statistiken des Benutzers hinterlegt.	
Schritte	1. Nutzer wählt seine Wasserpräferenz in der Refill-App aus 2. App speichert hinterlegte Wasserpräferenz mit dem am Smartphone angebrachten NFC ID im Backend 3. Nutzer hält Smartphone an NFC-Lesegerät der Refill-Station 4. Das Smartphone überträgt per NFC die ID an die Refill-Station 5. Die Refill-Station sendet eine Anfrage an das Backend, um die Wasserpräferenzen zum erhaltenen ID zu erfahren. 6. Refill-Station erhält vom Backend Wasserpräferenzen für die ID ab. 7. Refill-Station startet die Wasserausgabe entsprechend den Wasserpräferenzen und sendet den Start der Wasserentnahme per Internet an den MQTT-Broker, welcher von der App abgefragt wird, sodass die App über den Start informiert wird. 8. Nutzer sieht eine sich füllende Wasserflasche auf dem Smartphone und erkennt, wie lange die Füllung dauern wird. 9. Wasserstation beendet Wasserausgabe und sendet das Ende der Wasserentnahme per Internet an den MQTT-Broker. Die App erhält so die Information über den Abschluss der Wasserausgabe. 10. Nutzer sieht eine Erfolgsmeldung in der App.	

<table>
<tr><td rowspan="2">Design

Entscheidungen</td><td>Akzeptiert
DD1.1: Wasserentnahme Start in App mittels NFC-Chip und Internet
DD1.2: Wasserausgabe Fortschritt wird per Internet an App übertragen</td></tr>
<tr><td>Verworfen
Übertragung der Wasserpräferenzen mittels WLAN
Übertragung der Wasserpräferenzen mittels Bluetooth
Übertragung Wasserpräferenzen via Auslesen von NFC-Chip Speicher</td></tr>
<tr><td>Vorteile und Chancen</td><td>Nachteile und Risiken</td></tr>
<tr><td>Verbindungsvorgang und Datenübertragung zwischen Station und NFC-Chip sehr schnell (<= 1 Sekunde)
NFC besitzt eine Übertragungsrate von bis zu 424 kbit/s
Station muss nicht manuell ausgewählt werden
Nutzer wird positiv beeinflusst durch Erfolgsmeldung</td><td>NFC-Chip muss einmalig initial am Smartphone angebracht werden und in der App hinterlegt werden
Refill-Station benötigt eine permanente Internetverbindung
Smartphone- und Refill-Station-Internetverbindung dürfen nicht unterbrochen werden</td></tr>
<tr><td>Annahmen und Schätzungen</td><td>Abwägungen und Kompromisse</td></tr>
<tr><td>-</td><td>Smartphones ohne NFC werden nicht unterstützt
Zur Nutzung der smarten Funktionen muss die Refill-App lokal installiert werden</td></tr>
</table>

Tabelle 2: Architekturtreiber - Schnelle und transparenten Wasserentnahme mittel App

4.1.2 Wasserentnahme mit smarter Wasserflasche möglich

Treibername	Wasserentnahme mit smarter Wasserflasche	
Treiber-ID	2	
Ablauf	**Beschreibung**	**Quantifizierung**
Umgebung	Der Nutzer hat die Refill-App bereits auf seinem Smartphone installiert. Dem Nutzer steht ein NFC-Chip zur freien Verfügung	• Bisherige Starts der App >= 1 • Smartphone unterstützt NFC • Aktive Netzwerkverbindung
Stimulus	Nutzer hält eine smarte Wasserflasche an Refill-Station	
Antwort	Refill-Station gibt gewünschtes Wasser aus, Eintrag wird in den Statistiken des Benutzers hinterlegt	
Schritte	1. Nutzer hinterlegt eine Wasserart und Wassermenge für neue smarte Wasserflasche in der App 2. Nutzer hält NFC-Chip an Smartphone und diese liest die NFC ID aus 3. App speichert hinterlegte Wasserpräferenz mit zugehöriger NFC ID im Backend 4. Nutzer befestigt NFC-Chip an gewünschter Flasche 5. Nutzer hält Flasche mit NFC-Chip an Lesegerät einer beliebigen Refill-Station 6. Hinterlegte Wasserpräferenzen werden von Refill-Station ausgegeben a. Steht die hinterlegte Wasserart an einer Refill-Station nicht zur Verfügung, wird die dort unterstützte Wasserart ausgegeben 7. Refill-Station sendet Wasserentnahme an Datenbank im Backend für den für diese NFC ID hinterlegten Nutzer	
Design Entscheidungen	Akzeptiert • DD2.1: Wasserpräferenzen der smarten Wasserflasche werden in Datenbank gespeichert	
	Verworfen • Wasser Präferenzen werden lokal auf dem NFC-Chip gespeichert	

Vorteile und Chancen	Nachteile und Risiken
• Anschaffungskosten für Nutzer zum Einrichten einer smarten Wasserflasche ist sehr gering (< 1 €/Stück) • Eigene Flasche kann verwendet werden, was gut aus ökologischer Sicht ist • Smartphone wird bei der Wasserentnahme nicht benötigt • Wasser Präferenzen pro smarte Wasserflasche können in der App geändert werden, ohne dass die Flasche physisch vor Ort sein muss	• Refill-Station benötigen eine permanente Internetverbindung
Annahmen und Schätzungen	**Abwägungen und Kompromisse**
-	• Durch den Eigenbau jeder smarten Wasserflasche, gibt es kein einheitliches Design, was vermarktet werden oder zu einer Wiedererkennung führen kann

Tabelle 3: Architekturtreiber - Wasserentnahme mit smarter Wasserflasche

4.1.3 Manuelle Wasserentnahme möglich

<table>
<tr><td>Treibername</td><td colspan="2">Manuelle Wasserentnahme möglich</td></tr>
<tr><td>Treiber-ID</td><td colspan="2">3</td></tr>
<tr><td>Ablauf</td><td>Beschreibung</td><td>Quantifizierung</td></tr>
<tr><td>Umgebung</td><td>Nutzer befindet sich vor einer aktiven Wasserstation</td><td></td></tr>
<tr><td>Stimulus</td><td>Nutzer verwendet mechanische Knöpfe an Station</td><td></td></tr>
<tr><td>Antwort</td><td>Wasser wird ausgegeben</td><td></td></tr>
<tr><td>Schritte</td><td colspan="2">1. Nutzer verwendet Knöpfe zur Wasserausgabe
a. Je nach Wasser Präferenz verwendet der Nutzer unterschiedliche Knöpfe
2. Wasser wird ausgegeben
3. Anhand von der Farbe der LED kann erkannt werden, ob gerade stilles oder Sprudelwasser ausgegeben wird
a. Grün = Still
b. Blau = Sprudel
4. Refill-Station meldet, wie viel Wasser von welchem Typ an dem Backend ausgegeben wurde.</td></tr>
<tr><td rowspan="2">Design Entscheidungen</td><td colspan="2">Akzeptiert
• DD3.1: Jede Wasserstation besitzt manuelle Knöpfe zur Wasserentnahme</td></tr>
<tr><td colspan="2">Verworfen
• LCD-Display zur Anzeige des Status der Station</td></tr>
<tr><td colspan="2">Vorteile und Chancen</td><td>Nachteile und Risiken</td></tr>
<tr><td colspan="2">• Wasserentnahme auch möglich, wenn Internetverbindung ausgefallen ist.
• Simple Bedienung, welche bekannt von normalen Wasserstationen ist.</td><td>• Wasserentnahme fließt nicht auf persönliche Statistik ein
• Kosten einer Refill-Station in der Anschaffung leicht erhöht (ungefähr 1,50 € pro Station)</td></tr>
<tr><td colspan="2">Annahmen und Schätzungen</td><td>Abwägungen und Kompromisse</td></tr>
<tr><td colspan="2">-</td><td>-</td></tr>
</table>

Tabelle 4: Architekturtreiber - Manuelle Wasserentnahme möglich

4.1.4 Refill-Station finden

<table>
<tr><td>Treibername</td><td colspan="2">Refill-Station finden</td></tr>
<tr><td>Treiber-ID</td><td colspan="2">4</td></tr>
<tr><td>Ablauf</td><td>Beschreibung</td><td>Quantifizierung</td></tr>
<tr><td>Umgebung</td><td></td><td>Der Nutzer ist bereits angemeldet</td></tr>
<tr><td>Stimulus</td><td>Der Nutzer öffnet die Refill-App</td><td></td></tr>
<tr><td>Antwort</td><td>Kartenansicht mit allen Refill-Stationen und eigener Standort</td><td>Standortermittlung aktiviert</td></tr>
<tr><td>Schritte</td><td colspan="2">• Nutzer öffnet die App
• Kartenansicht wird angezeigt, Refill-Station sind mit Kartenmarkierungen eingezeichnet</td></tr>
<tr><td rowspan="2">Design Entscheidungen</td><td colspan="2">Akzeptiert
• DD4.1: Startbildschirm der App zeigt Kartenübersicht
• DD4.1: OpenStreetMap als Kartengrundlage</td></tr>
<tr><td colspan="2">Verworfen
• Verwendung von Google Maps aufgrund von Lizenzkosten der API</td></tr>
<tr><td colspan="2">Vorteile und Chancen</td><td>Nachteile und Risiken</td></tr>
<tr><td colspan="2">• Nach Öffnen der Refill-App keine weiteren Klicks notwendig
• Verständlich und intuitiv aufgrund von ähnlichem Design zu bekannten Navigation-Apps</td><td>-</td></tr>
<tr><td colspan="2">Annahmen und Schätzungen</td><td>Abwägungen und Kompromisse</td></tr>
<tr><td colspan="2">• Der Nutzer verfügt über ein internetfähiges Mobiltelefon</td><td>• Finden anhand von Stationsname im Vergleich zu Listenansicht schwieriger</td></tr>
</table>

Tabelle 5: Architekturtreiber - Refill-Station finden

4.1.5 Echtzeitdaten der Wasserentnahmen und Zustand der Refill-Stationen

<table>
<tr><td>Treibername</td><td colspan="2">Echtzeitdaten der Wasserentnahmen und Zustand der Refill-Stationen</td></tr>
<tr><td>Treiber-ID</td><td colspan="2">5</td></tr>
<tr><td>Ablauf</td><td>Beschreibung</td><td>Quantifizierung</td></tr>
<tr><td>Umgebung</td><td>Refill-Station ist mit dem Internet verbunden</td><td></td></tr>
<tr><td>Stimulus</td><td>Wasserentnahme wurde abgeschlossen</td><td>Es wurden mehr als 50 ml entnommen</td></tr>
<tr><td>Antwort</td><td>Wasserentnahme wird in Datenbank hinterlegt</td><td></td></tr>
<tr><td>Schritte</td><td colspan="2">1. Wasser wird von Refill-Station ausgegeben (unabhängig davon, ob per Smartphone, smarter Wasserflasche oder Knopf)
2. Die Daten der Wasserentnahme werden an das Backend übermittelt und dort in die Datenbank eingepflegt.
3. Optional: Bei einer Fehlfunktion der Refill-Station wird diese direkt in der DB hinterlegt</td></tr>
<tr><td rowspan="2">Design Entscheidungen</td><td colspan="2">Akzeptiert
• DD5.1: Übertragung der Wasserentnahmen direkt von Refill-Station zu Backend</td></tr>
<tr><td colspan="2">Verworfen
• Übertragung der Statistik mittels NFC und Internetverbindung eines Smartphones</td></tr>
<tr><td colspan="2">Vorteile und Chancen</td><td>Nachteile und Risiken</td></tr>
<tr><td colspan="2">• Statistik der Wasserentnahmen werden in Echtzeit übertragen
• Störungen an Wasserstation wird in Echtzeit angezeigt
• Refill-Stationen können per Fernwartung aktiviert und deaktiviert werden</td><td>• Refill-Station benötigen eine permanente Internetverbindung</td></tr>
<tr><td colspan="2">Annahmen und Schätzungen</td><td>Abwägungen und Kompromisse</td></tr>
<tr><td colspan="2">-</td><td>• Jede Refill-Station hat durch die permanent Internetverbindung Fixkosten pro Monat</td></tr>
</table>

Tabelle 6: Architekturtreiber - Echtzeitdaten der Wasserentnahmen der Refill-Stationen

4.1.6 Einfache Verwaltung der Wasser-Stationen

<table>
<tr><td>Treibername</td><td colspan="2">Einfache Verwaltung der Wasser-Stationen</td></tr>
<tr><td>Treiber-ID</td><td colspan="2">6</td></tr>
<tr><td>Ablauf</td><td>Beschreibung</td><td>Quantifizierung</td></tr>
<tr><td>Umgebung</td><td>Verwaltungsnutzer ist an einem Computer angemeldet und hat das Verwaltungsportal im Browser geöffnet</td><td>Computer hat eine Internetverbindung</td></tr>
<tr><td>Stimulus</td><td>Verwaltungsnutzer aktualisiert die Daten einer Refill-Station</td><td></td></tr>
<tr><td>Antwort</td><td>Aktualisierte Daten werden allen Nutzer in ihrer Refill-App angezeigt</td><td></td></tr>
<tr><td>Schritte</td><td colspan="2">1. Verwaltungsnutzer öffnet Verwaltungsportal
2. Verwaltungsnutzer sieht eine tabellarische Sicht der Refill-Stationen
a. Nutzer klickt auf Eintrag und sieht in einer Eingabemaske alle Daten und kann diese aktualisieren
b. Nutzer klickt auf Button (“Neue Refill-Station”) und sieht eine Eingabemaske, in welche er alle Daten einer neuen Refill-Station hinterlegen kann
3. Aktualisierte Daten werden im Backend gespeichert und stehen den App-Nutzern zur Verfügung</td></tr>
<tr><td rowspan="2">Design Entscheidungen</td><td colspan="2">Akzeptiert
• DD6.1: Daten werden in einer relationalen Datenbank gespeichert
• DD6.2: Verwaltungsprogramm wird als Web-App zur Verfügung gestellt
• DD6.3: Web-App wird mit dem Tool PowerApps umgesetzt</td></tr>
<tr><td colspan="2">Verworfen
• Verwaltung der Station in lokaler App möglich
• Verwendung einer dokumentenbasierten Datenbank
• Native Entwicklung einer Web-App</td></tr>
</table>

Vorteile und Chancen	Nachteile und Risiken
• Web-App kann von Verwaltungsnutzer ohne Installation verwendet werden • Relationale Datenbank erleichtert die Arbeit mit PowerApps • Einfache Anpassung in der Web-App können dank Low-Code in PowerApps von externen Personen mit wenig Informatikhintergrund durchgeführt werden • Relationale Datenbank erleichtert Datenexporte für weitere Analysen • Weite verbreitete Nutzung in der Industrie	• Auf Daten kann nur mit aktiver Internetverbindung zugegriffen werden
Annahmen und Schätzungen	**Abwägungen und Kompromisse**
-	• Abhängigkeit von Microsoft durch die Nutzung von PowerApps

Tabelle 7: Architekturtreiber - Einfache Verwaltung der Wasser-Stationen

4.2 Qualitätsmerkmale

Im Kontext der Softwarearchitektur definieren Qualitätsmerkmale die Gesamtheit der Eigenschaften, die die Güte einer Architektur bestimmen. Sie sind maßgeblich dafür verantwortlich, dass die Architektur die spezifischen Anforderungen und Ziele eines Softwareprojekts erfüllt.

ID	Name	Beschreibung
Q1	Plattformübergreifende gleiche mobile User Experiences	Die Refill-App soll auf iOS und Android mit der gleichen UX zur Verfügung stehen.
Q2	Hohe Verfügbarkeit	Das Backend soll eine hohe Verfügbarkeit besitzen. Da MQTT-Server extern von System ist und keine kritische Funktionalität beinhaltet, ist deren Verfügbarkeit nicht Risikobehaftet.

Tabelle 8: Qualitätsmerkmale

4.2.1 Plattformübergreifende gleiche mobile User Experiences

<table>
<tr><td>Treibername</td><td colspan="2">Plattformübergreifende gleiche mobile User Experiences</td></tr>
<tr><td>Treiber-ID</td><td colspan="2">Q1</td></tr>
<tr><td rowspan="2">Design Entscheidungen</td><td colspan="2">Akzeptiert
Q-DD1: Mobilen App wird mit Flutter umgesetzt</td></tr>
<tr><td colspan="2">Verworfen
Entwicklung mit LowCode Tool “Flutterflow”
Native Entwicklung</td></tr>
<tr><td colspan="2">Vorteile und Chancen</td><td>Nachteile und Risiken</td></tr>
<tr><td colspan="2">Native Performance: Hohe Leistung und Benutzerfreundlichkeit durch native Kompilierung
Große Entwickler-Community, breites Angebot an Ressourcen, Bibliotheken und Support.
Geringe Abhängigkeit von externen Diensten und Unternehmen</td><td>Anpassungen nur mir Entwicklerkenntnisse möglich</td></tr>
<tr><td colspan="2">Annahmen und Schätzungen</td><td>Abwägungen und Kompromisse</td></tr>
<tr><td colspan="2">-</td><td>Abwägung zwischen plattformübergreifender Entwicklung und optimaler Performance
Erhöhter Entwicklungsaufwand für eine flexiblere Lösung im Gegensatz zu einer No-Code-Plattform</td></tr>
</table>

Tabelle 9: Qualitätsmerkmale - Plattformübergreifende gleiche mobile User Experiences

4.2.2 Hohe Verfügbarkeit

<table>
<tr><td>Treibername</td><td colspan="2">Hohe Verfügbarkeit</td></tr>
<tr><td>Treiber-ID</td><td colspan="2">Q2</td></tr>
<tr><td rowspan="2">Design Entscheidungen</td><td colspan="2">Akzeptiert
Q-DD2: Redundanten Maschinen für Backend</td></tr>
<tr><td colspan="2">Verworfen
-</td></tr>
<tr><td colspan="2">Vorteile und Chancen</td><td>Nachteile und Risiken</td></tr>
<tr><td colspan="2">Risiko eines Ausfalls des Systems wird reduziert
Verbesserte Leistung
Erhöhte Skalierbarkeit
Flexibilität</td><td>Zusätzliche Wartungsarbeit
Konsistenz der Daten muss sichergestellt werden
Komplexität
Erhöhte Anforderungen an die Datensicherheit</td></tr>
<tr><td colspan="2">Annahmen und Schätzungen</td><td>Abwägungen und Kompromisse</td></tr>
<tr><td colspan="2">-</td><td>Durch redundante Systeme werden die Betriebskosten steigen</td></tr>
</table>

Tabelle 10: Qualitätsmerkmale - Hohe Verfügbarkeit

4.3 Zusammenfassung der Designentscheidungen

ID	Designentscheidung	Beschreibung
DD1.1	Wasserentnahme Start in App mittels NFC-Chip und Internet	Zum Start einer Wasserausgabe mittels eines Smartphones wird die Refill-App benötigt. Die Refill-Station erhält alle notwendigen Informationen mittels einem am Smartphone angebrachten NFC-Chips und einer Internetabfrage im Backend.
DD1.2	Wasserausgabe Fortschritt wird per Internet an App übertragen	Der Nutzer sieht in seiner Refill-App, wie weit die Wasserausgabe bereits fortgeschritten ist und wird über den Abschluss informiert. Hierfür wird ein MQTT-Broker verwendet.
DD2.1	Wasserpräferenzen der smarten Wasserflasche werden in Datenbank gespeichert	Die hinterlegten Wasserpräferenzen pro smarte Wasserflasche werden nicht lokal auf dem Chip der smarten Flasche gespeichert, sondern in der Datenbank, sodass eine Änderung der Präferenzen mittels der App jederzeit möglich ist
DD3.1	Jede Wasserstation besitzt manuelle Knöpfe zur Wasserentnahme	An einer Wasserstation kann Wasser jederzeit durch Betätigen von manuellen Knöpfen ausgegeben werden.
DD4.1	Startbildschirm der App zeigt Kartenübersicht	Beim Öffnen der App wird dem Nutzer eine Kartenübersicht mit allen Refill-Stationen angezeigt.
DD4.2	OpenStreetMap als Kartengrundlage	Die Darstellung der Kartenübersicht wird mit OpenStreetMap umgesetzt
DD5.1	Übertragung der Wasserentnahmen direkt von Refill-Station zu Backend	Nach jeder Wasserausgabe wird diese direkt im Backend gespeichert. Dies gelingt durch eine direkte Verbindung zwischen Refill-Station und Backend.
DD6.1	Daten werden in einer relationalen Datenbank gespeichert	Alle Daten werden in einer relationalen Datenbank (PostgreSQL) gespeichert.
DD6.2	Verwaltungsprogramm wird als Web-App zur Verfügung gestellt	Die App zur Verwaltung der Station und Einsicht von Feedback und Statusinformationen wird als Webapp zur Verfügung gestellt.
DD6.3	Web-App wird mit dem Tool PowerApps umgesetzt	Die Web-App wird mit PowerApps umgesetzt.

Q-DD1	Mobilen App wird mit Flutter umgesetzt	Die Refill-App wird mit Flutter für Android und iOS entwickelt.
Q-DD2	Redundanten Maschinen für Backend	Für den Service im Backend und der DB werden redundanten Maschinen zur Verfügung gestellt.

Tabelle 11: Zusammenfassung der Designentscheidungen

5 Architektursichten

In den nachfolgenden Kapiteln wird die Architektur aus verschiedenen Sichten dargestellt und erläutert, um die Komplexität der architektonischen Darstellung zu verringern und den Fokus auf verschiedene Anliegen legen zu können.

5.1 Funktionen zur Laufzeit

In der Funktionen zur Laufzeit Sicht werden die Funktionalitäten des Systems und den einzelnen Modulen gezeigt. Hier liegt der Fokus auf dem Aufbau des Systems und Modulen. Für die Moduldiagramme werden 3-Layer Diagramm verwendet. In einer 3-Schichten-Architektur wird die Software in Präsentation (Benutzeroberfläche), Geschäftslogik und Datenhaltung aufgeteilt. Das sorgt für mehr Unabhängigkeit, bessere Wartung und einfachere Skalierung[2].

5.1.1 Modulübersicht

Um das System besser aufzuteilen, wurden insgesamt fünf getrennten Modulen definiert, welche voneinander unabhängig entwickelt, deployt und gewartet werden können. In der nachfolgenden Abbildung 1 sind die Module und deren Verbindung dargestellt, wichtig dabei ist, dass den gelb markierten Modulen systemexterne Module sind, die im Rahmen der Entwicklung nicht betrachtet werden müssen. In dem folgenden Kapitel wird im Detail auf diese eingegangen.

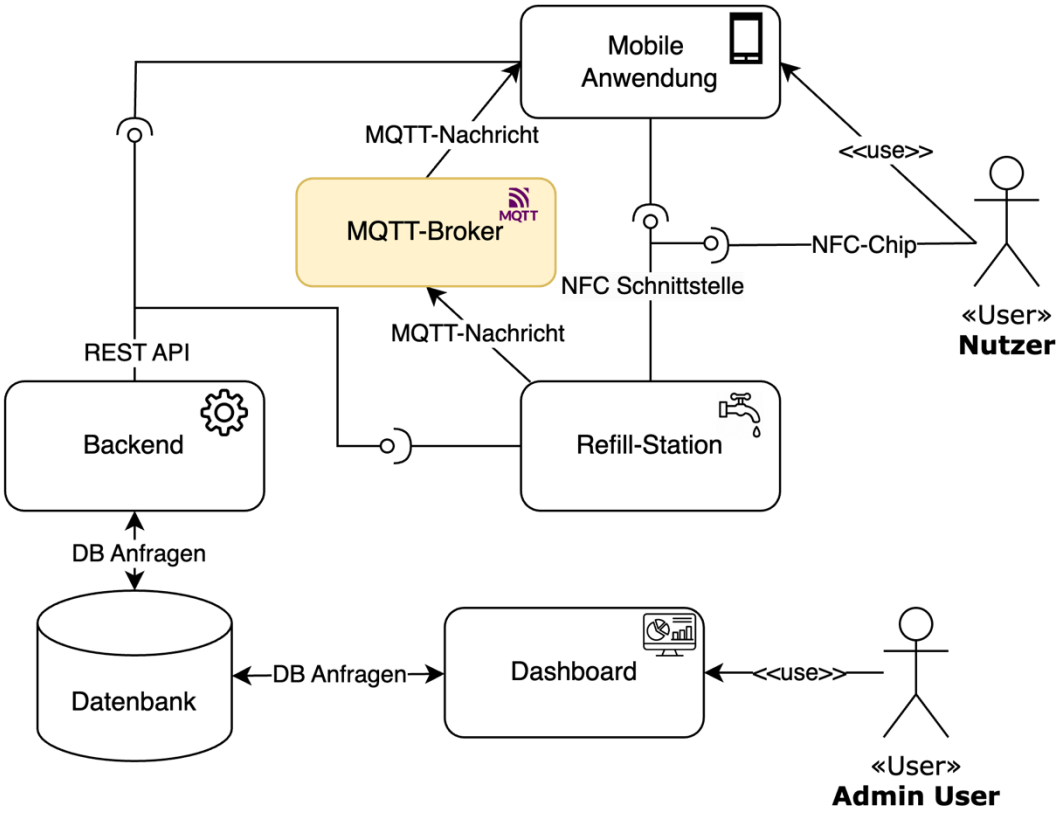

Abbildung 1: System Modulübersicht

[2] https://www.ibm.com/de-de/topics/three-tier-architecture

- Mobile Anwendung:
    - Dient dafür, dass der Nutzer mit dem System interagieren kann. Soll für Smartphones mit verschiedenen Betriebssystemen verfügbar sein. (iOS und Android)
- Refill-Station:
    - Hardwarekomponente, welche zuständig ist, dem Nutzer Wasser nach bestimmten Präferenzen (Still, Sprudel, Wassermenge) zu geben.
- MQTT-Broker:
    - Externes System, welches Nachrichten zwischen den Refill-Stationen und den mobilen Anwendungen verteilt.
- Dashboard:
    - Grafische Schnittstelle für Verwaltungsmitarbeiter.
- Backend:
    - Webservice, welcher eine API für die Datenverwaltung bereitstellt.
- Datenbank:
    - Zentrale Ablage für die im System zu findenden Daten. Es wird eine relationale Datenbank verwendet.

5.1.2 Mobile Anwendung

Die in der folgenden gezeigten Abbildung 2 Refill-App ist für den Endnutzer konzipiert. Die App bietet ein App-User-Interface, mit dem der Nutzer interagiert. Dieses Interface greift auf alle Dienste des Service Layer zu.

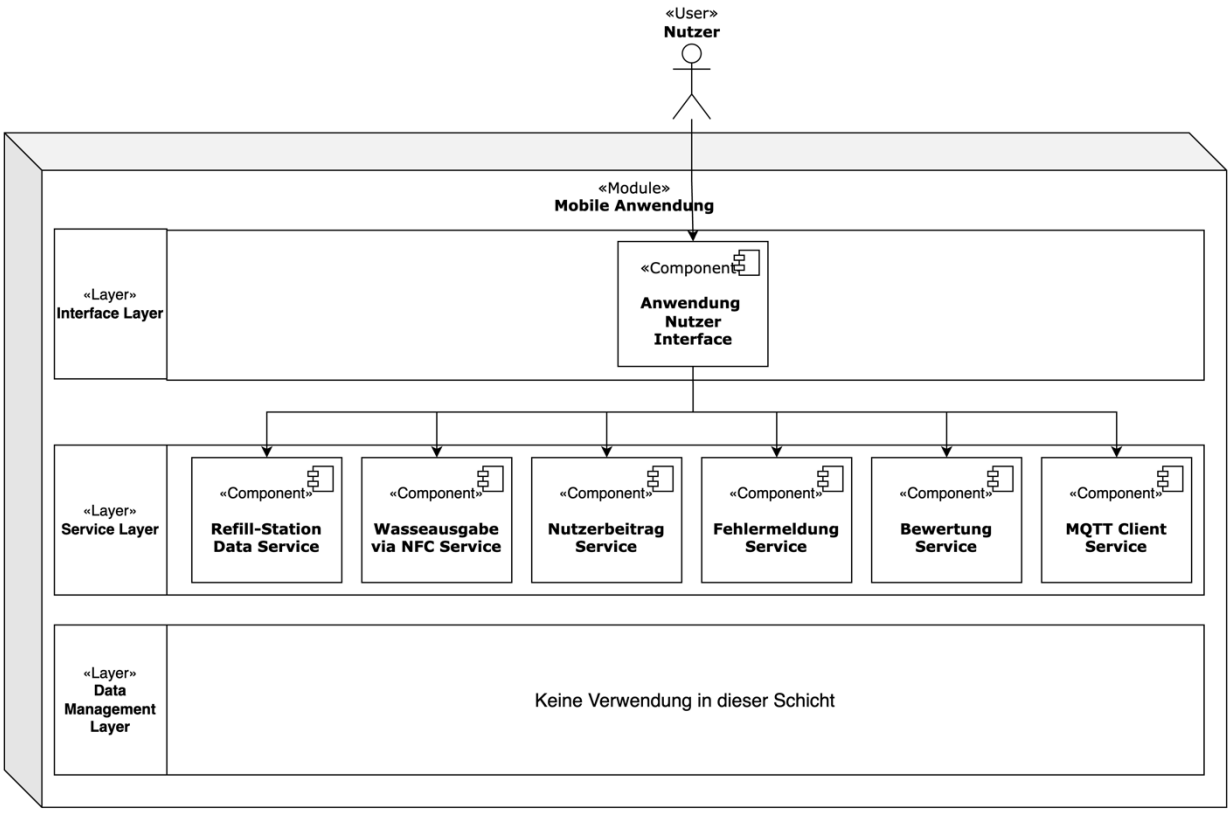

Abbildung 2: funktionales Moduldiagramm: Mobile Anwendung

Insgesamt umfasst die mobile Anwendung folgende Dienste:

- Refill-Station Data Service
    - Über diesen Service werden alle Daten bezüglich der Refill-Station abgefragt. Hierzu zählen alle Eigenschaften der Refill-Stationen, wie beispielsweise deren Position, Öffnungszeiten, Wasserarten etc.
- Wasserausgabe via NFC-Service
    - Dieser Service ist für die Kommunikation mit den Refill-Stationen zuständig. Er speichert die gewählten Wasserpräferenz in Verbindung zum am Smartphone angebrachten NFC-Chip, was die Datenübertragung mit der gewünschten Refill-Station ermöglicht. Außerdem wartet er auf Rückmeldungen vom Server zum Status der Wasserentnahme.
- Nutzerbeitrag Service
    - Sowohl Statistiken über eigenen Beitrag (z. B. entnommene Wasser, vermiedene Abfall) als auch über dem Beitrag der Stadt (z. B. Anzahl Refill-Stationen) und der Community (z. B. gesamt entnommene Wasser) können vom Nutzer über diesen Service aufgerufen werden.
- MQTT Client Service
    - In dem Anmeldprozess meldet sich die Anwendung als Subscriber für die MQTT-Topic, die dem Benutzer zugeordnet ist. Somit bekommt die Anwendung alle relevanten Nachrichten aus der Refill-Station.
- Fehlermeldung Service
    - Bemerkt der Nutzer eine Fehlfunktion an einer Refill-Station, kann er diesen mithilfe dieses Service melden.
- Bewertung Service
    - Der Nutzer kann mit dem Rating Service eine Refill-Station bewerten in den Kategorien Sauberkeit, Zugänglichkeit und Wasserqualität in einer Skala von 1 bis 5 bewertet. Anderen Nutzer wird so die Durchschnittsbewertung der Refill-Station in den einzelnen Kategorien angezeigt.

Keiner der Service greift auf den lokalen Datenspeicher des mobilen Endgeräts zu. Alle Daten werden direkt an das Backend gesendet und direkt von diesem bezogen.

5.1.3 Refill-Station

Die Refill-Station wird ebenfalls vom Endnutzer verwendet. Sie besitzt ein User Interface, bestehend aus Knöpfen, mit denen Wasser manuell entnommen werden kann, Kontrollleuchten und einem NFC-Lesegerät. Der Aufbau der Software einer Refill-Station in der Abbildung 3 dargestellt.

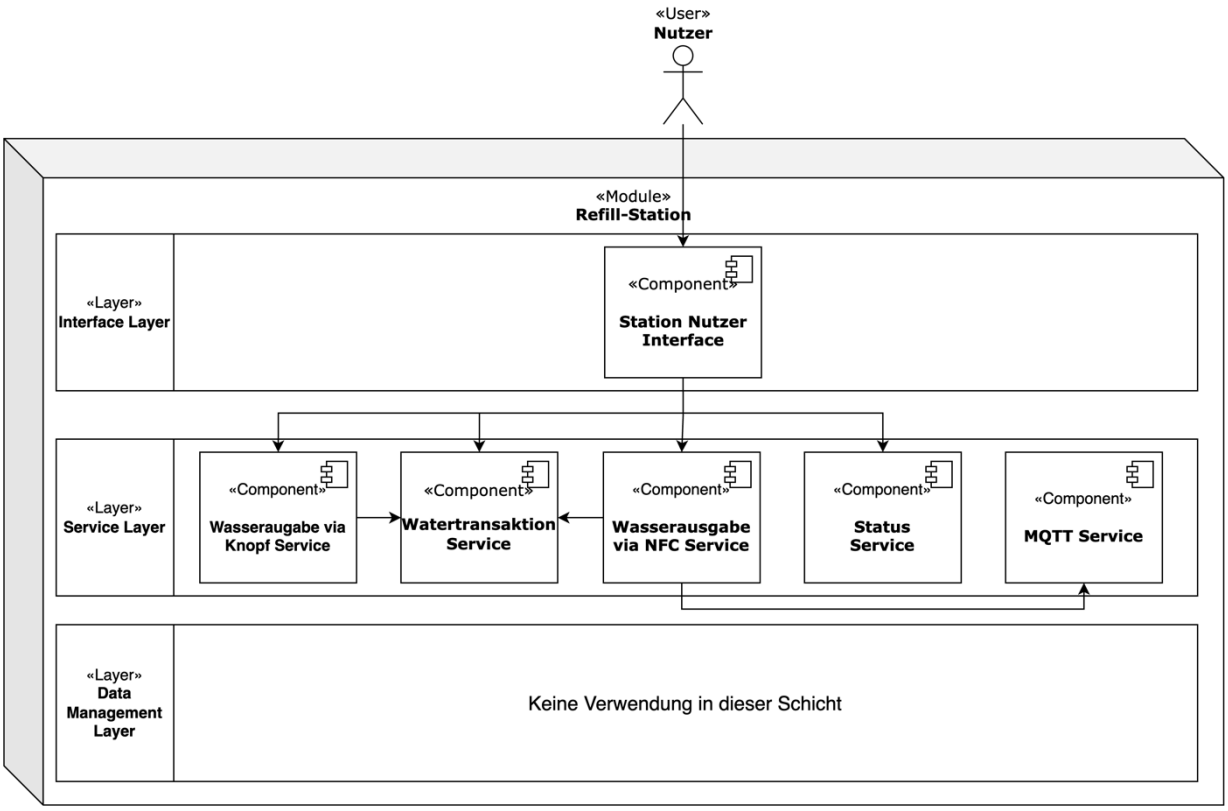

Abbildung 3: funktionales Moduldiagramm: Refill-Station

Jede Refill-Station besitzt folgende Services:

- Wasserausgabe via Knopf Service
    - Mit diesem Service kann ohne die Refill-App jederzeit von einer Refill-Station Wasser entnommen werden, durch Verwendung der verbauten Hardware-Knöpfe.
- Wasserausgabe via NFC-Service
    - Dieser Service dient dazu, eine Wasserentnahme mittels des Auslesens eines NFC-Chips zu starten. Sobald ein NFC-Chip ausgelesen wird, werden die hinterlegten Tag-Präferenzen (Wasserart, Volumen) im Backend aufgerufen und die gewünschte Wasserausgabe gestartet.
- Wassertransaktion Service
    - Sobald eine Wasserentnahme erfolgreich abgeschlossen wurde, wird diese Wasserentnahme an das Backend gesendet, sodass diese in den Nutzer und Community Statistiken aufgenommen wird.
- Status Service
    - Erkennt die Refill-Station einen Fehler, wird dieser durch den Status Service an das Backend gesendet, sodass eine Fehlfunktion den unterschiedlichen Nutzer in der Refill-App und dem Dashboard angezeigt wird.

- MQTT Service
    - Zuständig dafür, dass der MQTT-Broker darüber informiert, wann eine Wasserausgabe via NFC gestartet und beendet wird.

Die Refill-Station besitzt keinen Zwischenspeicher. Alle Daten werden direkt vom Backend abgefragt oder dort gespeichert.

5.1.4 Backend

Die Darstellung in der Abbildung 4 des Backends unterscheidet sich zu den anderen Diagrammen so, dass an oberster Stelle ein Gateway Layer anstelle eines Interfaces verwendet wird. Auf das Backend wird direkt von den vorgestellten Modulen “mobile Anwendung” und “Refill-Station” zugegriffen.

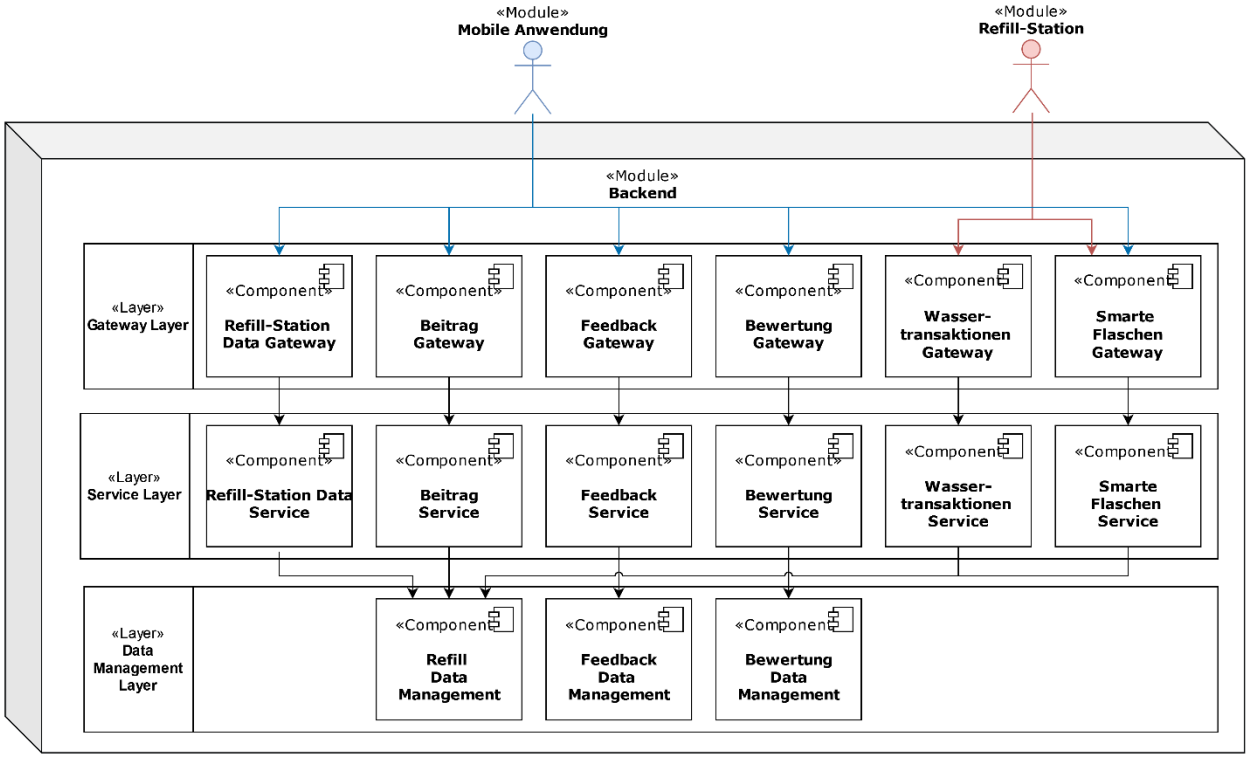

Abbildung 4: funktionales Moduldiagramm: Backend

Der Gateway-Layer koordiniert alle API-Aufrufe der mobilen App und der Refill-Stationen an den Service-Layer. Dieser Service-Layer bietet verschiedene Services, welche direkt auf den darunterliegenden Data Management Layer zugreifen. Im Data Management Layer werden abschließend alle Daten, die im System verwendet werden, gespeichert bzw. abgerufen.

Die Beziehung zwischen dem Gateway- und Servicelayer sind 1:1, sie werden daher im folgenden als “Bund” behandelt und erläutert:

- Refill-Station Data:
    - Zuständig für die Informationen zu den einzelnen Refill-Stationen, einschließlich Standort, Öffnungszeiten, Wasserquelle, Status usw.
- Beitrag:
    - Dient der Zusammenfassung der nutzerspezifischen und nutzerübergreifenden Verwendung des Systems und deren Auswirkung. Die vom einzelnen Nutzer

entnommene Wassermenge und die von allen Nutzern gemeinsam entnommene Wassermenge steht dabei im Vordergrund.

- Feedback:
  - Sorgt für die Übertragung und Protokollierung von Nutzerfeedback zu den Refill-Stationen.
- Bewertung:
  - Ermöglicht das Bewertungssystem für verschiedene Aspekte einer Refill-Station, wie Sauberkeit, Zugänglichkeit und Wasserqualität.
- Wassertransaktionen:
  - Zuständig für die Speicherung und Zuordnung von aufgegebenem Wasser zu einem Nutzer oder zur Community.
- Smarte Flaschen:
  - Verantwortlich für die Definition, Verwaltung und Verwendung von smarten Flaschen mit festgelegten Präferenzen.

5.1.5 Dashboard

Die Admin Web-App ist für die Verwaltung vorgesehen. Der Aufbau ist in Abbildung 5 zu sehen. Es bietet ein Admin User Interface, in welchem Daten zu den Refill-Stationen eingesehen und bearbeitet werden können. Außerdem werden Störungen angezeigt und in Statistiken können Kennzahlen über verschieden Aspekten des Systems angezeigt werden.

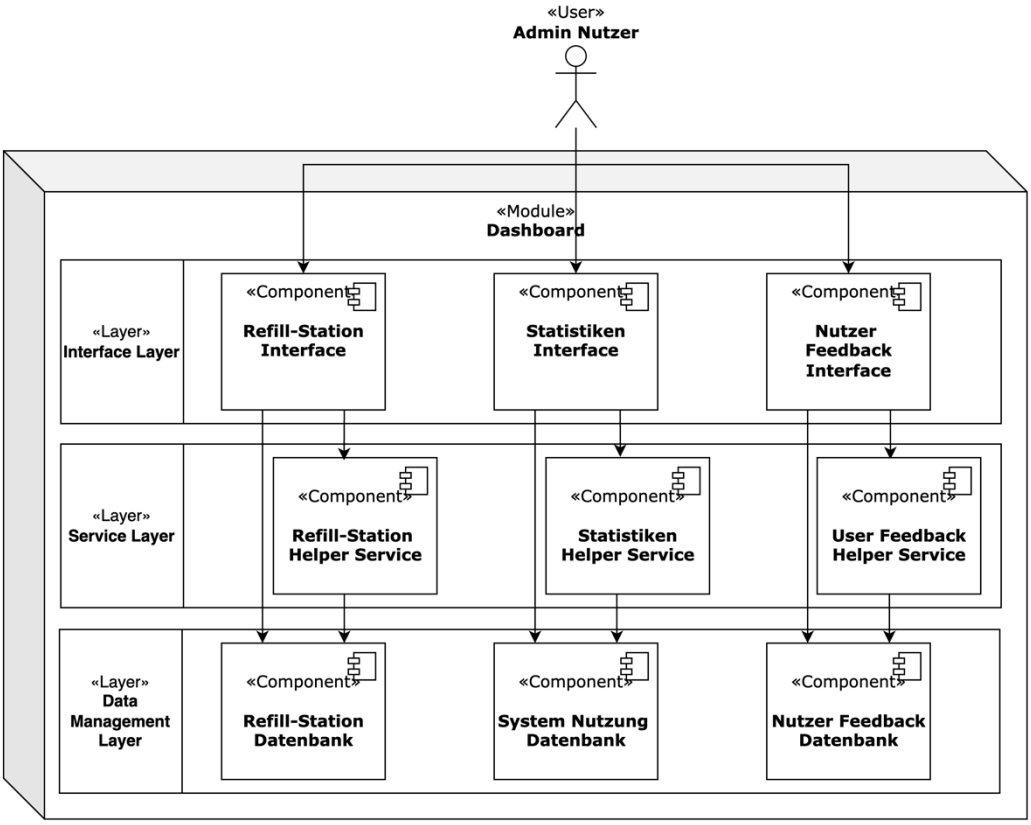

Abbildung 5: funktionales Moduldiagramm: Admin Dashboard

Im Gegensatz zur Refill-Station und der Client App besteht in der Entwicklung ein direkter Zugriff von dem Interface auf die Daten. Es besteht keine Verbindung zum Backend Server. Teilweise werden in PowerApps geschrieben Zusatzskripte (in Abbildung 5 sogenannte Helper Service) verwendet. Zur besseren Darstellung der Funktionen werden diese als getrennte Interface Layer dargestellt. Auf sie wird in derselben Web-App zugegriffen.

- Refill-Station Interface
    - Hier können alle Refill-Stationen eingesehen werden. Es können die einzelnen Attribute eingesehen und aktualisiert werden. Außerdem können neue Refill-Stationen eingepflegt werden.
- Statistiken Interface
    - Hier können Statistiken eingesehen werden, die für die Verwaltung relevant sind.
- Nutzer Feedback Interface
    - Hier können von Nutzern gemeldete Probleme angesehen werden und deren Status bearbeitet werden.

5.1.6 Datenbank

Die Datenbank für das Gesamtsystem wurde entworfen, um die verschiedenen Entitäten und deren Beziehungen zueinander zu verwalten. Das ER-Diagramm stellt die Struktur und die Verbindungen der Datenbanktabellen visuell dar. In der Abbildung 6 sind die Hauptkomponenten und ihre Beziehungen dargestellt. In der Tabelle 12 wird die Bedeutung der einzelnen Module erläutert werden.

Bezeichnung	**Funktion**
USER	Repräsentiert die Nutzer der Anwendung.
BOTTLE	Repräsentiert die Wasserflaschen der Nutzer, welcher ein NFC-Chip zugeordnet werden kann.
REFILLSTATION	Repräsentiert die Refill-Stationen, an denen Wasser nachgefüllt werden kann.
REFILLSTATIONREVIEW	Repräsentiert die Bewertungen der Nutzer für die Refill-Stationen.
REFILLSTATIONPROBLEM	Repräsentiert die Probleme, die an den Refill-Stationen gemeldet wurden.
WATERTRANSACTION	Repräsentiert die Wassertransaktionen an den Refill-Stationen.
LIKE	Repräsentiert die Likes, die Nutzer den Refill-Stationen gegeben haben.

Tabelle 12: Übersicht der Tabellen in der Datenbank

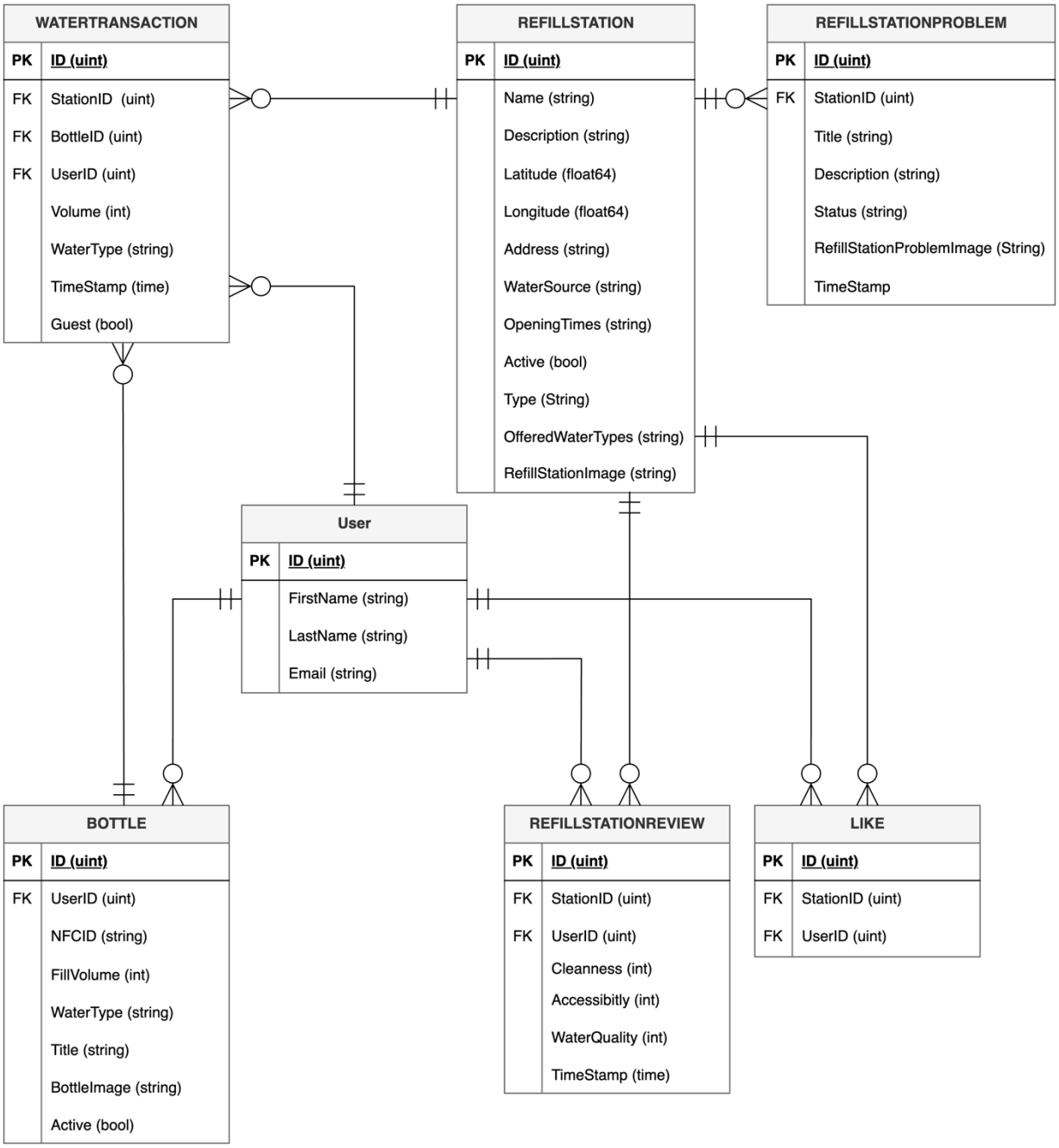

Abbildung 6: Refill Datenbankmodell

5.2 Daten zur Laufzeit

Die Wasserentnahme kann auf drei unterschiedliche Arten gestartet werden. Wie in Abbildung 7 dargestellt wird:

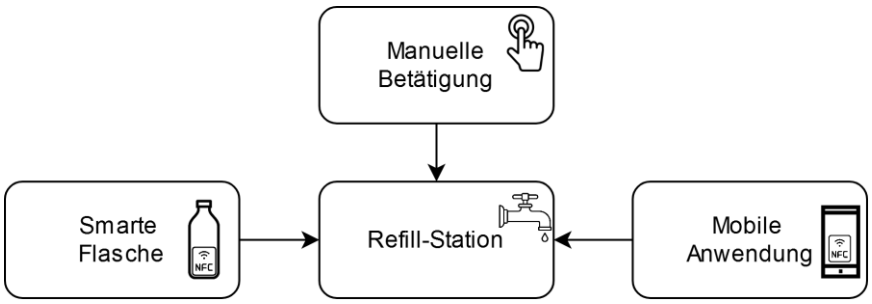

Abbildung 7: Arten der Wasserentnahme

Zum einen kann die Wasserausgabe durch das Verwenden einer smarten Flasche mit einem an der Flasche befestigten NFC-Chip gestartet werden, zum anderen kann mit einem Smartphone und der Refill-App mittels des eingebauten Hardware NFC-Chips die Wasserausgabe gestartet werden. Des weiteren ist eine Wasserausgabe mittels manuellen Knöpfen an der Wasserstation möglich. In allen folgenden vorgestellten Arten wird nach einer abgeschlossenen Wasserausgabe diese von der Wasserstation in die Datenbank geschrieben. Dieser Datenfluss wird in den Abbildungen nicht dargestellt.

5.2.1 Datenfluss Wasserstation – Manuelle Bestätigung

Für eine manuelle Wasserausgabe kann direkt an der Wasserstation ein Knopf verwendet werden, welcher die Wasserausgabe startet. Hierfür wird kein Smartphone verwendet und auch wenn die Internetverbindung der Wasserstation ausfällt, steht diese Funktion zur Verfügung. Die abgegebene Wassermenge kann in diesem Fall vom Nutzer nach der Wasserausgabe in der App eingetragen werden.

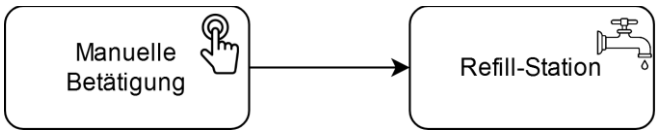

Abbildung 8: Wasserausgabe durch manuelle Betätigung

5.2.2 Datenfluss Wasserstation – Smarte Flasche

Zum Erstellen einer smarten Flasche wird zur Einrichtung ein Smartphone benötigt. Ein NFC-Chip wird an einer Flasche befestigt. Anschließend werden in der App die Wasserpräferenzen für die zukünftige smarte Flasche angelegt. Der NFC-Chip, welcher an der Flasche befestigt wurde, wird nun vom Handy ausgelesen und alle Daten an das Backend gesendet. So ist ein eindeutiges Mapping zwischen dem NFC-Chip und dem hinterlegten Nutzer und den Wasserpräferenzen angelegt. Dieser Vorgang ist in Abbildung 9 visualisiert.

Abbildung 9: Anlegen einer smarten Flasche

Wird die soeben erstellte Wasserflasche an das NFC-Lesegerät einer Refill-Station gehalten, überprüft die Refill-Station, ob für diesen NFC-Chip im Backend Daten hinterlegt sind und ruft die hinterlegten Präferenzen ab, wie in Abbildung 10 zu dargestellt ist. Nach dieser Abfrage erfolgt die Wasserausgabe und anschließend die Zuschreibung der Wasserentnahme an den für den NFC-Chip hinterlegten Nutzer.

Abbildung 10: Verwenden einer smarten Flasche

Der App Nutzer sieht seine Wasserentnahme auf seinem Smartphone, ist nun in der Lage, ohne direkten Kontakt zu seiner smarten Flasche zu haben, die Wasser Präferenzen einfach in der App anzupassen. Eine erneute Verbindung per NFC zwischen Smartphone und smarter Flasche ist nicht mehr notwendig

5.2.3 Datenfluss Wasserstation –Mobile Anwendung

Als zweite Möglichkeit kann eine Wasserentnahme auch mit dem Smartphone erfolgen. Hier hinterlegt der App-Nutzer die gewünschte Wasserausgabe. Dafür steht ihm auch die Möglichkeit zur Verfügung, von ihm vordefinierte Angaben auszuwählen. Anschließend hält er, dass an seinem Smartphone angebrachte NFC-Chip an die Wasserstation.

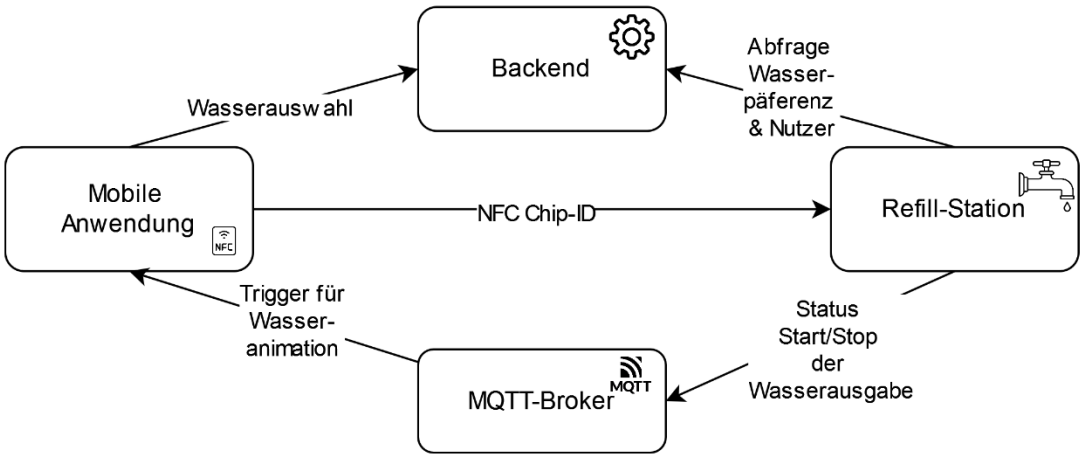

Abbildung 11: Datenfluss mit mobiler Anwendung

Die gewählte Wasserausgabe wird dabei im Backend gespeichert. Per NFC-Chip wird nur eine eindeutige ID an die Wasserstation übertragen. Die Refill-Station fragt dann im Backend die gewünschte Wasserausgabe für die übermittelte ID ab und startet die Wasserausgabe. Mit dem Start der Wasserausgabe wird eine Animation in der App angezeigt, die dem Nutzer verdeutlicht, wie weit die Wasserausgabe bereits fortgeschritten ist. Hierzu wird ein MQTT-Broker verwendet.

Die mobile Anwendung registriert sich als Subscriber beim MQTT-Topic, das dem aktuell eingeloggten Nutzer zugewiesen ist. Wenn die Wasserausgabe startet, sendet die Refill-Station eine MQTT-Nachricht an dieses Topic. Dadurch wird die mobile Anwendung benachrichtigt, dass die Wasserausgabe begonnen hat und kann die entsprechende Animation starten. Nach Beendigung der Wasserausgabe sendet die Refill-Station erneut eine Nachricht an den MQTT-Broker, woraufhin die App dem Nutzer eine Erfolgsmeldung anzeigt.

5.3 Bereitstellung zur Laufzeit

In diesem Abschnitt liegt der Fokus auf der Verteilung und Platzierung der einzelnen Softwaremodule in der Produktionsumgebung. Ziel ist es, ein klares Bild davon zu vermitteln, wie die Anwendung nach der Bereitstellung auf den unterschiedlichen Rechenknoten läuft, welche in der Abbildung 12 gezeigt werden.

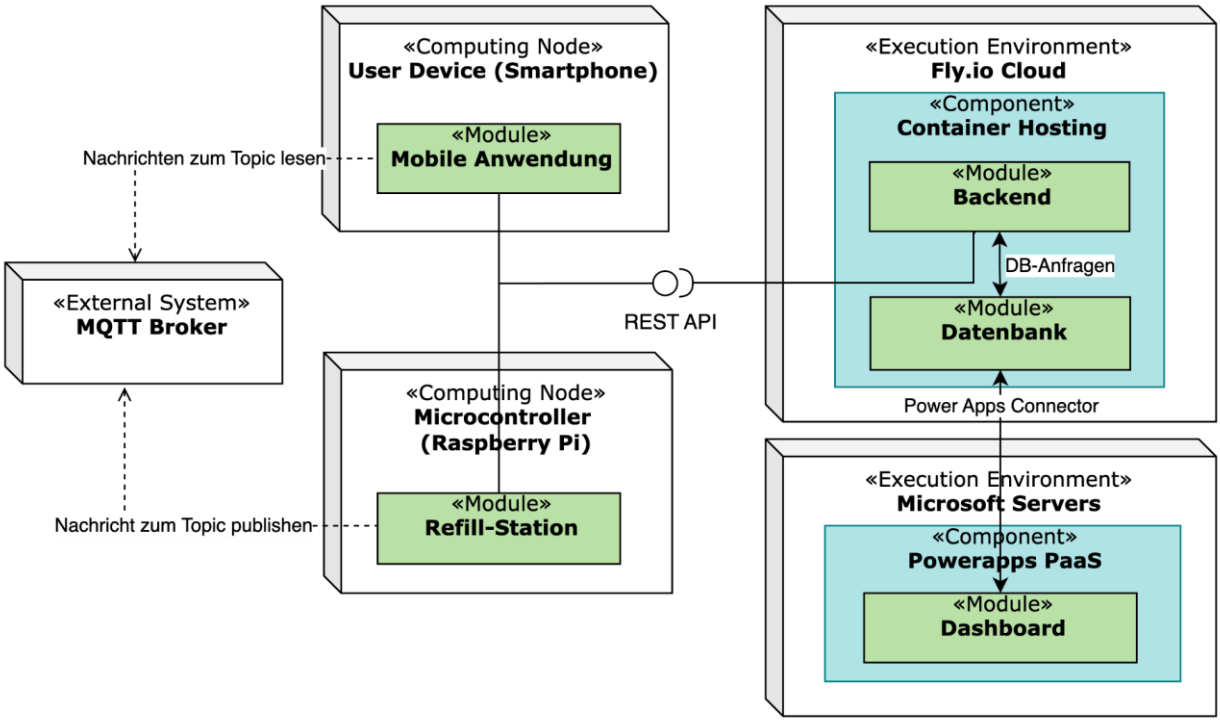

Abbildung 12: Bereitstellung von System Laufzeit

Das System wird auf verschiedenen Rechenknoten unabhängig voneinander deployt, dadurch ist es möglich, dass die Entwicklung, Testung, Skalierung und Wartung der einzelnen Module gekapselt wird. In der Laufzeit des Systems sind insgesamt fünf unterschiedliche Rechenknoten involviert.

- MQTT-Broker
  - Externes System, welches sich darum kümmert, Nachrichten an die mobile Anwendung zu vermitteln, die von der Refill-Station kommen. Wichtig dabei ist, dass zu jedem Nutzer ein MQTT-Topic angelegt wird. Ein externer MQTT-Broker bietet eine skalierbare, wartungsfreie und zuverlässige Lösung, die hohe Verfügbarkeit und Skalierbarkeit ermöglicht. Somit wird den Entwicklern den Aufwand für Infrastrukturverwaltung erspart.
- User Device (Smartphone)
  - Smartphone, auf dem die Refill-App installiert wird.
- Microcontroller (Raspberry Pi)
  - Einplatinencomputer mit der Besonderheit GPIO-Ports anzubieten, zu welchem Hardwarekomponenten angeschlossen werden können. Siehe Kapitel Hardware Refill-Station.

- Fly.io Cloud:
    - Cloud-Computing-Plattform, die sich speziell an Entwickler richtet, die ihre Webanwendungen weltweit bereitstellen möchten. Ihr Fokus liegt dabei auf Einfachheit, Skalierbarkeit und globaler Reichweite [3].
- Microsoft Servers:
    - Microsoft Power Apps ist ein Platform-as-a-Service (PaaS)-Angebot, das die zugrunde liegende Serverinfrastruktur vollständig abstrahiert [4].

5.4 Technologien

In diesem Kapitel werden die Technologien vorgestellt, die zur Entwicklung und Implementierung des Projekts eingesetzt werden. Jede Technologie wurde sorgfältig ausgewählt, um sicherzustellen, dass die Anwendung effizient, skalierbar und benutzerfreundlich ist. Die Auswahl basiert auf verschiedenen Faktoren wie Leistung, Entwicklerfreundlichkeit und Integrationsmöglichkeiten.

5.4.1 Frontend: Flutter

Für die Entwicklung der mobilen Anwendung wird Flutter verwendet. Flutter ermöglicht die plattformübergreifende Entwicklung von Anwendungen für iOS, Android, Web und Desktop aus einer einzigen Codebasis.

Flutter Version 3.22.0-35.0

5.4.2 Dashboard: Microsoft Power Apps

Bei Microsoft Power Apps handelt es sich um ein Low-Code Dienst zur Erstellung von Web-Apps. Es wird genutzt, um benutzerdefinierte Geschäftsanwendungen zu erstellen, die eine Integration und Automatisierung von Prozessen ermöglichen.

5.4.3 Backend: Golang

Für das Backend, welches in Golang Version: 1.22.2 entwickelt wird, ist eine umfangreiche Auswahl an Modulen und Frameworks im Einsatz, die zur Effizienz und Leistungsfähigkeit der Anwendung beitragen. Ein wesentlicher Bestandteil ist das HTTP Web Framework Gin-Gonic/Gin, das für seine Effizienz und seinen geringen Overhead bekannt ist und häufig in Produktionsumgebungen zum Einsatz kommt.

Die Dokumentation und Validierung der API erfolgen durch Tools wie Swaggo/Swag, das automatisch Swagger-Dokumentationen für REST-APIs generiert, und die Go-OpenAPI Suite, die bei der Arbeit mit OpenAPI Spezifikationen unterstützt. Bei der Datenverwaltung setzt man auf das ORM-Tool Gorm für eine effiziente Datenbankinteraktion und Jackc/pgx, einen optimierten PostgreSQL-Datenbanktreiber.

[3] https://fly.io/
[4] https://powerapps.microsoft.com/de-de/low-code-platform/

5.4.4 Datenbank: PostgreSQL

Für die Datenbanklösung wird objektrelationales Datenbanksystem PostgreSQL verwendet.

5.4.5 Cloud Plattform: Fly.io

Fly.io ist eine Cloud-Computing-Plattform, die sich speziell an Entwickler richtet, die ihre Webanwendungen weltweit bereitstellen möchten. Ihr Fokus liegt dabei auf Einfachheit, Skalierbarkeit und globaler Reichweite.

5.4.6 NFC

Zur Übertragung der Nutzer-ID und Wasserpräferenz wird die NFC-Technologie verwendet. NFC ermöglicht den kontaktlosen Austausch von Daten über kurze Distanzen und ist eine Erweiterung der RFID-Technologie.

5.4.7 MQTT

Zur Kommunikation zwischen den Refill-Stationen und den mobilen Anwendungen wird das MQTT-Protokoll verwendet. MQTT (Message Queueing Telemetry Transport) ist ein leichtgewichtiges Messaging-Protokoll, das für die Kommunikation in Netzwerken mit niedriger Bandbreite und hoher Latenz entwickelt wurde.

5.4.8 Python

Für die Steuerung der Wasserstation wird Python in der Version 3.9 verwendet, ergänzt durch spezifische Bibliotheken für eine effiziente Hardware-Ansteuerung und Kommunikation. Die Bibliothek pn532 ermöglicht die Interaktion mit dem NFC/RFID-Controller PN532, was für Zugangskontrollen und Nutzeridentifikation wichtig ist. Mit paho.mqtt.client nutzt die Station das MQTT-Protokoll, um Nachrichten zu senden und zu empfangen, was ideal für IoT-Anwendungen ist. Die gpiozero-Bibliothek erlaubt es, die GPIO-Pins auf einem Raspberry Pi zu steuern, was für das Öffnen und Schließen von Ventilen oder das Aktivieren von Pumpen entscheidend ist.

5.5 Code-Verwaltung

In diesem Projekt wurde GitHub als primärer Dienst für die Verwaltung des Codes genutzt. Der Code ist auf verschiedene Repositories aufgeteilt, dies ermöglicht es, spezifische Teile des Projekts getrennt zu bearbeiten und zu aktualisieren, ohne unbeabsichtigte Änderungen an anderen Teilen vorzunehmen.

- code_app: Enthält den Quellcode der plattformübergreifenden mobilen Anwendung.
- code_backend: Ist das Backend, das in Go implementiert wurde.
- refillstation: Die Refill-Station wurde mit Python umgesetzt.

Durch die Verwendung von GitHub und die strukturierte Aufteilung des Codes kann sichergestellt werden, dass der Entwicklungsprozess effizient, transparent und gut organisiert ist.

6 Hardware Refill-Station

Dieses Kapitel fokussiert sich auf die Funktionsweise einer Refill-Station, wobei insbesondere das Zusammenspiel von Software und Hardware und der Aufbau einer Refill-Station detailliert beleuchtet werden.

6.1 Refill-Station Aufbau

Eine Refill-Station funktioniert so, dass alle die Hardwarekomponenten an den GPIO-Ports eines Raspberry Pi 4 B angeschlossen sind. Die Hardwarekomponenten sind:

- PN532: Zuständig für das Auslesen der Inhalte eines NFC-Chips.
- Wasserpumpe 5V (x2): Bei Stromversorgung wird ein elektrischer Motor gestartet, der Wasser aus Wasserbehälter ausliefert. Diese werden jeweils an ein Relais angeschlossen.
- Wasserbehälter (x2): Pro Wassertyp ein Wasserbehälter notwendig (Still / Sprudel).
- Relais 5V (x2): Zuständig dafür, den Stromkreis zu den einzelnen Wasserpumpen zu schließen, wenn ein Signal aus dem Raspberry Pi kommt. In der Refill-Station kann ein Relais so verstanden werden, dass es die Funktionalität eines Switches darstellt.
- Knöpfe (3x): Jeweils einen pro Wassertyp. Dritter Knopf ist ein Notausschalter.
- LED (3x): Farbkodierte Anzeige von Funktionen
    - **Weiß**: Leuchtet, während die Wasserstation aktiv und bereit ist. Blinkt dreimal, wenn ein NFC-Chip bzw. Handy erfolgreich ausgelesen wurde.
    - **Grün**: Leuchtet, während die Wasserpumpe für stilles Wasser angeschlossen ist.
    - **Blau**: Leuchtet, während die Wasserpumpe für Sprudelwasser angeschlossen ist.

In der nachfolgenden Abbildung 13 kann der Schaltplan zu der Refill-Station eingesehen werden. Dabei ist wichtig zu erwähnen, dass der Softwareanteil abhängig von dem Hardwareaufbau ist, demzufolge bei Änderungen an der Hardware, muss die Software angepasst werden.

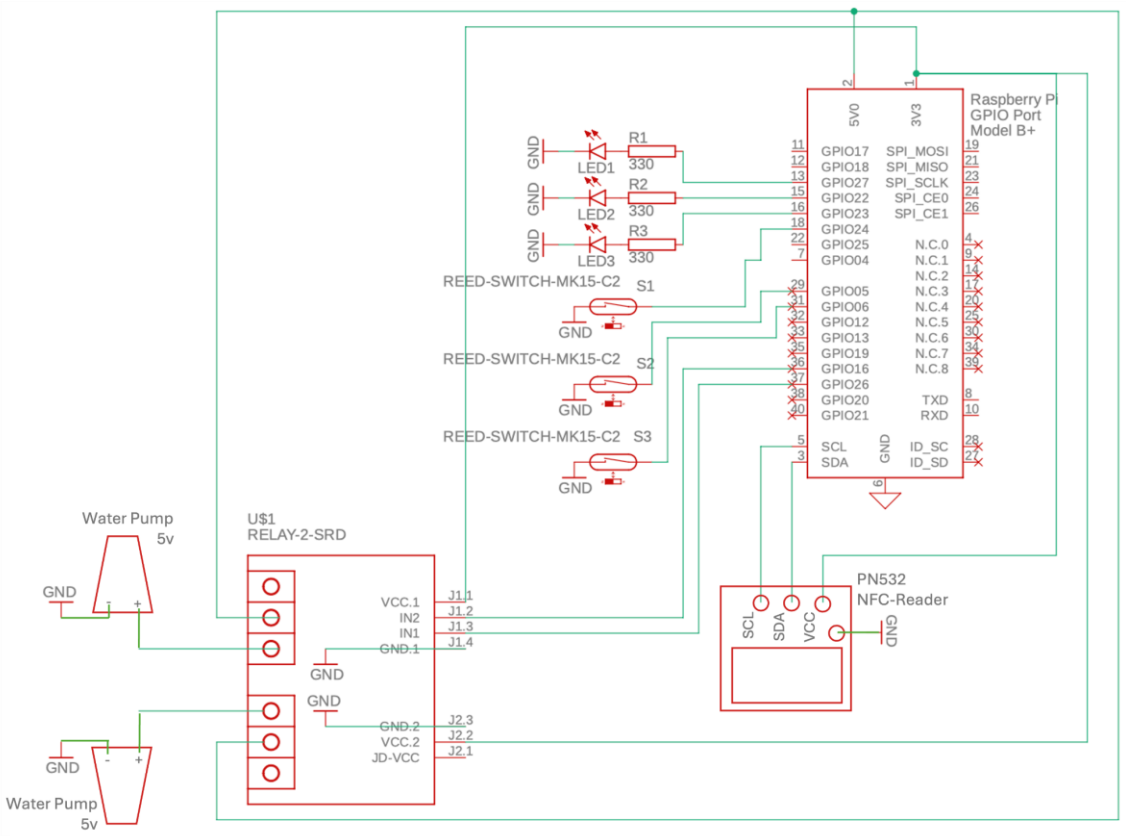

Abbildung 13: Schaltplan Refill-Station

6.2 Zusammenspiel Hardware-Software

Die Refill-Station besteht aus Software und Hardware. Der Softwareanteil steuert die unterschiedlichen Abläufe und somit die angeschlossenen Hardwarekomponenten. In der Refill-Station sind zwei Hauptprozesse zu finden. Im folgenden Unterkapiteln werden diese abgebildet und erläutert.

6.2.1 Manuelle Wasserausgabe

In der Abbildung 14 kann der Prozess einer manuellen Wasserausgabe grafisch dargestellt werden. In diesem Fall ist die Interaktion eines Nutzers mit den Hardware-Knöpfe der Refill-Station dargestellt. In der Abbildung ist die X-Achse die Zeit. Blau zeigt an, wann eine Hardwarekomponente mit Strom versorgt wird, rot hingegen, wann dies nicht der Fall ist.

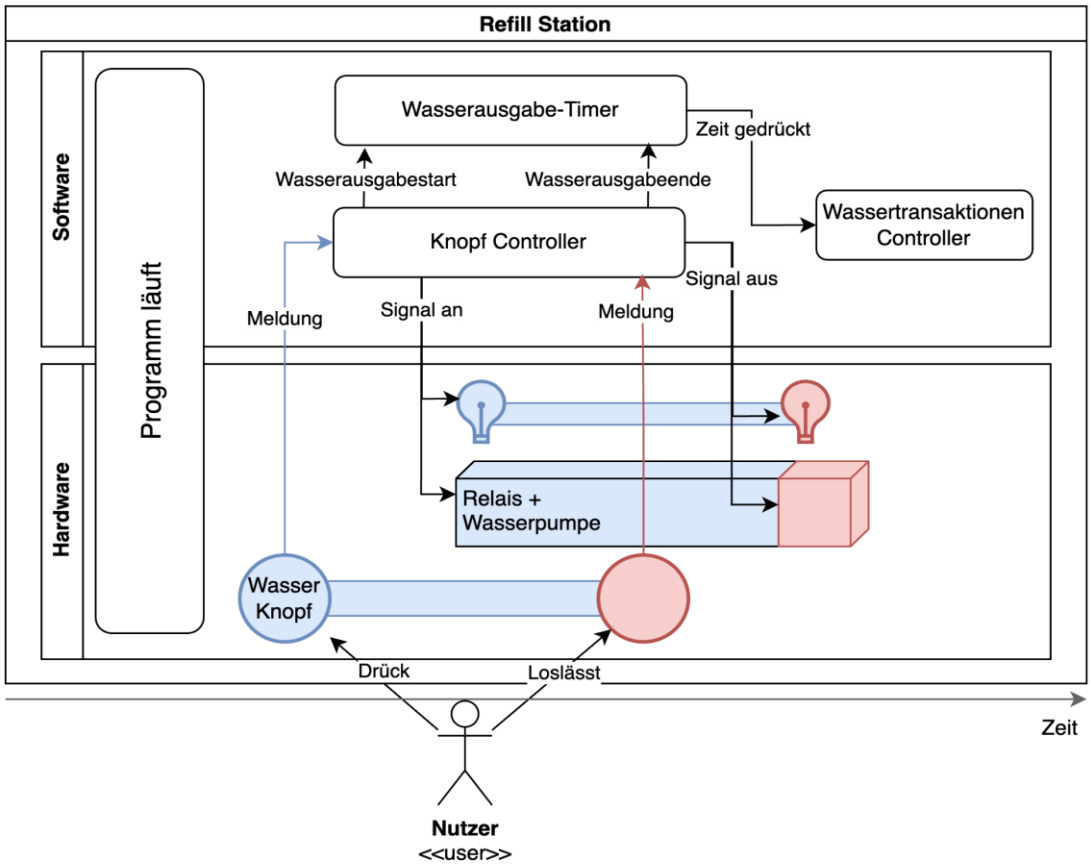

Abbildung 14. Kommunikationsdiagramm: manuelle Wasserausgabe

Im Folgenden wird der Ablauf für die in der Abbildung 14 gezeigte Funktionen erläutert:

1. Der Nutzer betätigt den gewünschten Knopf an der Refill-Station (Sprudel oder Still).
2. Der aktivierte Knopf beginnt durchgehend ein Signal an den Knopf-Controller zu senden.
3. Der Knopf-Controller reagiert auf das Signal und aktiviert die LED und das Relais. Das Relais wiederum schaltet die Wasserpumpe ein.
4. Der Knopf-Controller informiert den Wasserausgabe-Timer und dieser beginnt die Dauer der Wasserausgabe zu erfassen.
5. Der Nutzer beendet die Betätigung des Knopfes.
6. Der Knopf beendet das Senden seines Signals an den Knopf-Controller.
7. Der Knopf-Controller reagiert auf das Ende des Signal, deaktiviert die LED und des Relais, wodurch die Wasserpumpe abgeschaltet wird.
8. Der Wasserausgabe-Timer beendet die Zeiterfassung und übermittelt die erfasste Zeit an den Wassertransaktionen Controller.
9. Der Wassertransaktionen Controller sendet an das Backend-System alle relevanten Daten zur Wasserausgabe, wie Station-ID, Wassertyp und ausgegebenes Volumen.

6.2.2 Automatische Wasserausgabe

In der Abbildung 15 hingegen kann der Prozess einer automatischen Wasserausgabe betrachtet werden. In diesem Fall ist die Interaktion eines Nutzers mit der Hardware durch Kommunikation via NFC gezeigt. Wie in der Abbildung zuvor ist die X-Achse die Zeit. Blau zeigt an, wann eine Hardwarekomponente mit Strom versorgt wird, rot hingegen, wann dies nicht der Fall ist. Wichtig dabei ist, dass die PN532 ständig mit Strom versorgt ist.

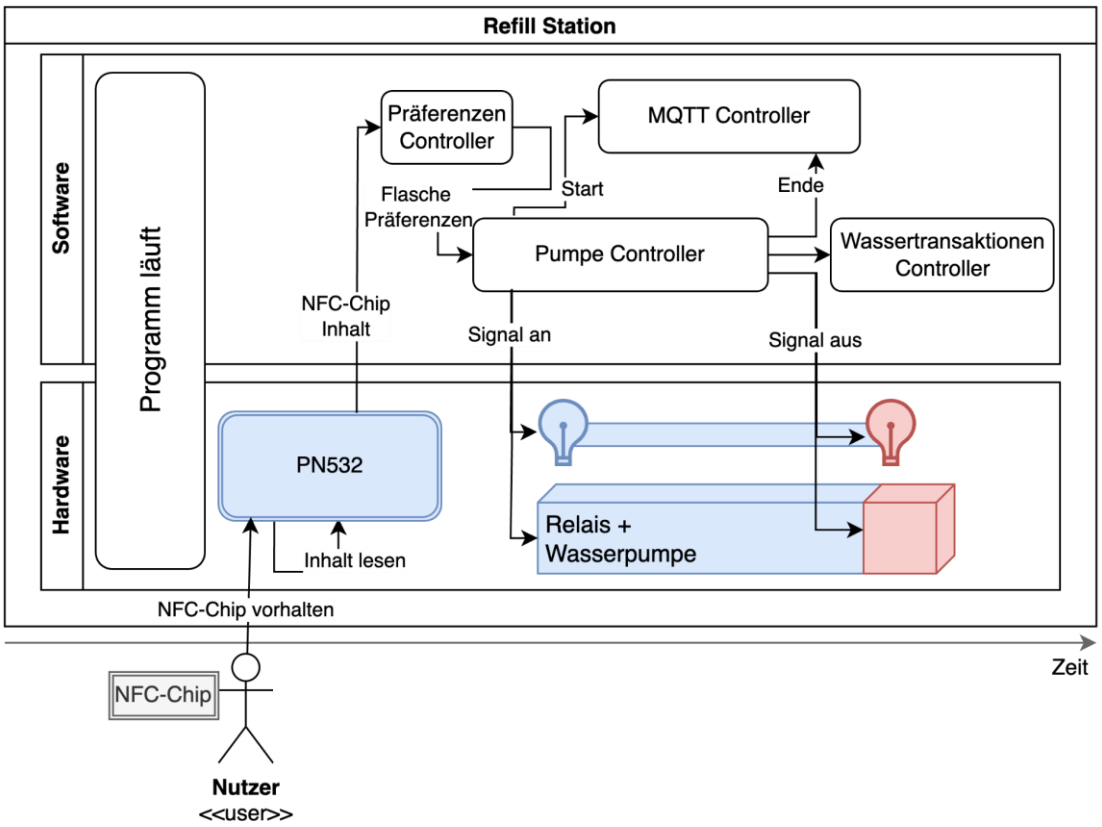

Abbildung 15: Kommunikationsdiagramm: automatische Wasserausgabe

Initiierung der automatischen Wasserausgabe:

1. Der Nutzer hält seinen NFC-Chip an den NFC-Reader (PN532) der Wasserstation.
2. Der NFC-Reader liest die Daten des NFC-Chips aus.
3. Der in der PN532 ausgelesene ID wird an den Präferenzen Controller weitergegeben.
4. Der Präferenzen Controller frägt im Backend die gespeicherten Präferenzen der gefundenen ID ab.
5. Das Backend übermittelt die relevanten Präferenzen der abgefragten ID an den Präferenzen Controller.
6. Der Präferenzen Controller leitet die empfangenen Präferenzen an den Pumpe Controller weiter.
7. Der Pumpe Controller aktiviert das Relais und die LED-Anzeige. Das Relais schaltet die Wasserpumpe ein.
8. Der MQTT-Controller sendet eine Startnachricht an den MQTT-Broker, um den Beginn der Wasserausgabe zu signalisieren.
9. Der Pumpe Controller überwacht die ausgegebene Wassermenge anhand der erhaltenen Wasserpräferenzen und der Pumpengeschwindigkeit.
10. Sobald die voreingestellte Wassermenge erreicht ist, schaltet der Pumpe Controller das Relais und die LED-Anzeige wieder aus.
11. Der MQTT-Controller sendet eine Abschlussmeldung an den MQTT-Broker, um das Ende der Wasserausgabe zu signalisieren.
12. Der Wassertransaktionen Controller sendet ein HTTP-Request an das Backend-System. Die Anfrage enthält relevante Daten zur Wasserausgabe, wie ausgegebenes Volumen, Wassertyp, Station-ID und User-ID.

7 Datenanalyse mittels Large Language Models

Die Daten, welche vom Gesamtsystem gespeichert werden, können mithilfe des externen Dienstes ChatCSV genauer analysiert werden. Hierbei handelt es sich um eine online Anwendung, die auf dem Chat GPT Model 3.5 basiert. Die Anwendung ist darauf spezialisiert auf die Auswertung von Tabellen im csv Format. Der Dienst unterstützt auch eine API, allerdings konnte hier kein zusammenhängender Chat erreicht werden, die Nachfragen zu dem gegebenen Output des Models ermöglicht hätte. Es erfolgte keine Analyse der Datensicherheit oder eines möglichen Datenabflusses, da die Daten keine personalisierten Informationen erhalten. Das Modell wurde insbesondere in Bezug auf die Analyse der Wasserentnahmen getestet.

Zur Durchführung der Analyse müssen die Daten exportiert und lokal gespeichert werden. Hierfür steht dem Nutzer im Dashboard eine entsprechende Funktion zur Verfügung. Die heruntergeladene Datei muss anschließend in dem Tool ChatCSV hochgeladen werden. Anschließend kann eine Analyse der Daten mittels natürlicher Sprache durchgeführt werden. Hierbei ist das Modell auch in der Lage, die Daten visuell in gewünschten Grafiken anzeigen zu lassen. Der Dienst ChatCSV steht kostenfrei zur Verfügung. Der Ablauf ist in Abbildung 16 zu sehen.

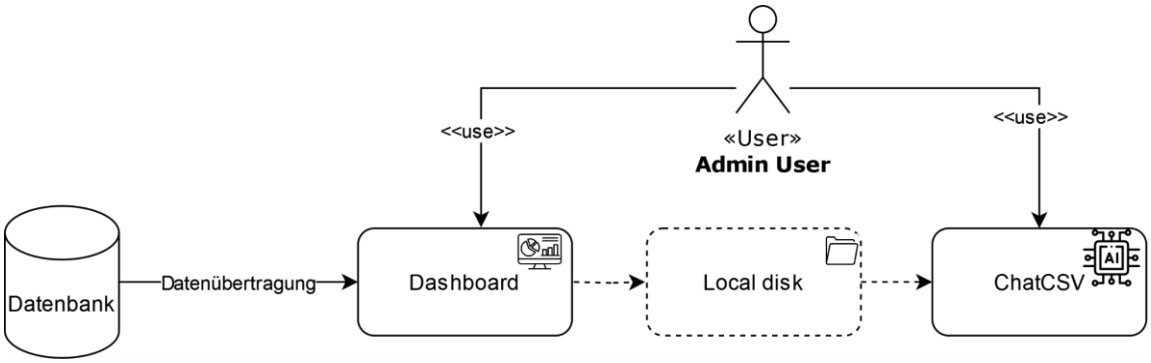

Abbildung 16: Datenanalyse Large Language Model

8 Evolution

Unter Evolution wird die fortlaufende Weiterentwicklung und Anpassung von Systemen an neue Anforderungen und veränderte Umgebungsbedingungen verstanden. Dieser Vorgang hat das Ziel, Systemlösungen ständig auf dem neuesten Stand zu halten und ihre Effizienz sowie Sicherheit zu gewährleisten. Dadurch können sowohl technische Innovationen als auch die Bedürfnisse der Nutzer zeitgerecht berücksichtigt werden. Evolution stellt daher einen wesentlichen Faktor für die langfristige Bedeutung und den Erfolg eines Projekts dar.

8.1 Risiken / Offene Punkte

Aufgrund der prototypischen Umsetzung der Lösung wurden gewisse Aspekte vernachlässigt. Diese Eingrenzung wurde mit dem Kunden vereinbart und auf dessen Vorschlag hin vorgenommen. In diesem Abschnitt werden diese Punkte nochmals aufgegriffen.

8.1.1 API-Sicherheit

Ein zentraler Punkt ist die Implementierung eines umfassenden Sicherheitskonzepts für die API. In der aktuellen Version wurde lediglich ein grundlegender API-Key-Mechanismus implementiert, um den Zugang zu den Schnittstellen zu kontrollieren. Dabei werden jedoch weitergehende Sicherheitsmaßnahmen, wie beispielsweise die Verschlüsselung der API-Keys, das regelmäßige Rotieren der Schlüssel sowie eine detaillierte Überwachung und Protokollierung der API-Nutzung, nicht berücksichtigt. Diese Maßnahmen sind entscheidend, um unbefugten Zugriff zu verhindern und die Integrität der Daten zu gewährleisten. In zukünftigen Versionen sollten diese Aspekte prioritär behandelt werden, um die Sicherheit der API umfassend zu gewährleisten.

8.1.2 Benutzerverwaltung

Ein weiterer Punkt betrifft die Benutzerverwaltung. Im aktuellen Prototyp wurde bewusst auf die Implementierung eines Passwort-Management-Systems verzichtet. Stattdessen wurde ein alternatives Authentifizierungsverfahren gewählt, das keine Passwörter erfordert. Diese Entscheidung wurde getroffen, um den Entwicklungsprozess zu beschleunigen und den Fokus auf die Kernfunktionalitäten der Lösung zu legen. Es ist darauf hinzuweisen, dass ein sicheres und benutzerfreundliches Passwort-Management-System ein wesentlicher Bestandteil einer umfassenden Benutzerverwaltung ist. In zukünftigen Iterationen sollte daher ein Augenmerk auf die Integration sicherer Passwortpraktiken gelegt, um die Sicherheit und das Vertrauen der Benutzer weiter zu stärken.

8.2 Entwicklungsherausforderungen

Die Entwicklung der Refill-App steht vor verschiedenen Herausforderungen, die es zu bewältigen gilt. Diese Herausforderungen betreffen unterschiedliche Bereiche der Anwendung und erfordern spezifische Lösungen und Ansätze. Im Folgenden werden die wesentlichen Entwicklungsherausforderungen detailliert beschrieben, die während der Umsetzung der Refill-App aufgetreten sind. Die Schwerpunkte liegen auf der mobilen Anwendung, dem Backend, dem Dashboard, dem Deployment und der Refill-Station. Jeder dieser Bereiche stellt eigene Anforderungen und Schwierigkeiten, die in diesem Abschnitt näher beleuchtet werden.

8.2.1 Mobile Anwendung

Bei der Entwicklung der mobilen Anwendung traten mehrere spezifische Probleme auf. Ein großes Hindernis war die direkte Testbarkeit von NFC-Funktionen. Da ein Teil der Entwicklergruppe nur über Emulatoren und nicht über physische Geräte arbeitete, konnte NFC nicht direkt getestet werden. Dies führte zu Verzögerungen und zusätzlichen Iterationen bei der Implementierung und dem Testen der NFC-Integration.

Ein weiteres bedeutendes Problem war der anfängliche Ausfall des Backends. Zu Beginn der Entwicklung konnten keine echten Daten vom Backend abgerufen werden, weshalb mit Mock-Daten gearbeitet werden musste. Dies beeinträchtigte die Entwicklung und führte zu Verzögerungen, da die Integration und die Funktionalität der App erst später mit echten Daten validiert werden konnten.

Zusätzlich gab es Herausforderungen aufgrund der Entscheidung, Bilder in die Anwendung einzubinden. Die Formatierung und Darstellung von Bildern wurde in Flutter nicht wie erwartet unterstützt, was zu Problemen führte. Unterschiedliche Ansätze mussten ausprobiert werden, um eine zufriedenstellende Lösung zu finden. Diese Komplikationen erforderten kreative Lösungswege und führten zu einer verlängerten Entwicklungszeit.

8.2.2 Backend (Go/DB)

Die Implementierung des Datenbank-Modells in Go verlief zunächst zügig und reibungslos. Auch die Definition der richtigen Endpunkte für die API konnte schnell abgeschlossen werden. Jedoch traten in der Folge einige Herausforderungen auf, die den Entwicklungsprozess verzögerten.

Eine wesentliche Komplikation ergab sich beim Testen der Datenbank. Ohne ein vollständiges Deployment war es nicht möglich, die Datenbank zu testen und sicherzustellen, dass sie korrekt initialisiert wurde. Dies führte dazu, dass es keine Möglichkeit gab, zu überprüfen, ob die Datenbank überhaupt funktionierte oder aktuell war. Erst nach der Integration des Deployments konnte über PostgreSQL nachvollzogen werden, ob die Datenbank richtig implementiert wurde. Eine weitere Herausforderung bestand in der ständigen Anpassung der Datenbankstruktur und der Endpunkte. Bei jeder Änderung an der Datenbank oder der Hinzufügung neuer Endpunkte musste der gesamte Prozess wiederholt werden: die Änderungen implementieren, neu deployen und testen. Dieser iterative Prozess nahm oft mehr Zeit in Anspruch als ursprünglich geplant. Darüber hinaus dauerte es eine Weile, bis alle API-Calls und Endpunkte definiert und funktionsfähig waren. Die genaue Definition und Implementierung aller notwendigen Endpunkte erforderte Anpassungen, was den Entwicklungsprozess weiter verzögerte.

8.2.3 Dashboard

Die Entwicklung des Dashboards mittels des Tools Microsoft Power Apps konnte zügig beginnen, da bereits Lizenzen der Hochschule Mannheim für die Studenten freigeschaltet waren. Des Weiteren war keine Einrichtung einer Entwicklerumgebung notwendig, da sich diese im Browser ausführen ließ. Die Lernkurve während der Entwicklung war sehr stark, und es zeichnete sich ab, dass man das Grundsystem dahinter durchschauen muss, was von der Komplexität allerdings für Personen mit Programmiererfahrung überschaubar scheint.

Die Dokumentation, welche im Internet vorgefunden werden konnte, war zum Teil veraltet, da sich der Sprachsyntax in den letzten Jahren leicht geändert hat, was zum Teil zu kleinen,

störenden Anpassungen gegenüber den Tutorials geführt hat. Allerdings gibt es eine sehr große Community, sodass für die meisten Probleme sehr schnell eine Lösung gefunden werden konnte. Eigenschaften oder Aktionen, welche nicht mit den Standardoptionen abgedeckt werden können, wurden mit der Programmsprache PowerFx gelöst. In der folgenden Abbildung 17 ist der Programmiercode zu sehen, der notwendig ist, um ein Bild, welches per Schaltfläche hochgeladen wird, in das Base64 Dateiformat umzuwandeln. Außerdem wird überprüft, ob die Bilddatei nicht 2 MB Dateigröße überschreitet.

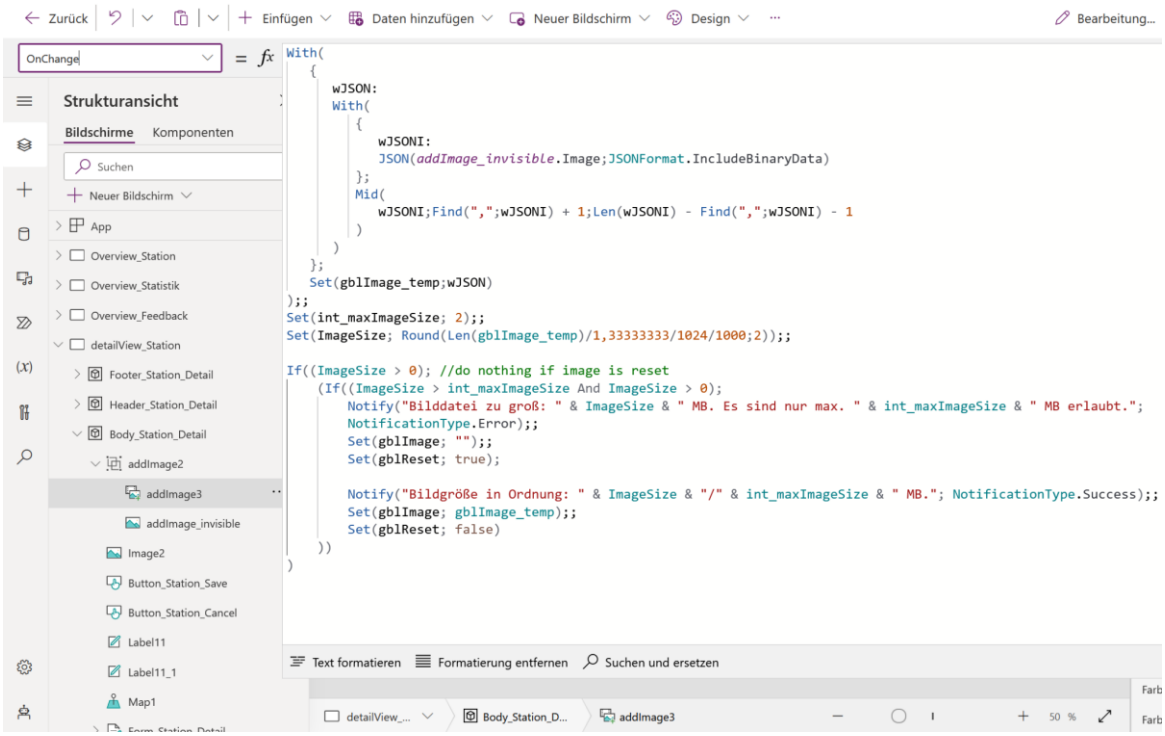

Abbildung 17: Beispielcode zum Upload von Bildern

Zusätzlich stehen in der Entwicklung sogenannte PowerFlows zur Verfügung. Diese können von gewissen Ereignissen getriggert werden, z. B. dem Einfügen eines neuen Eintrags. In den PowerFlows können komplexere Prozesse abgebildet werden, mit diversen Konnektoren zu anderen Diensten und von der Programmierung gewohnten Schleifen und Fallunterscheidungen. So können zum Beispiel durch das Einfügen eines neuen Eintrags das Senden einer Mail und das Aufrufen eines API-Calls gestartet werden. Diese Flows werden zum Export von csv Dateien verwendet. In Abbildung 18 wird allerdings ein vereinfachter Flow dargestellt.

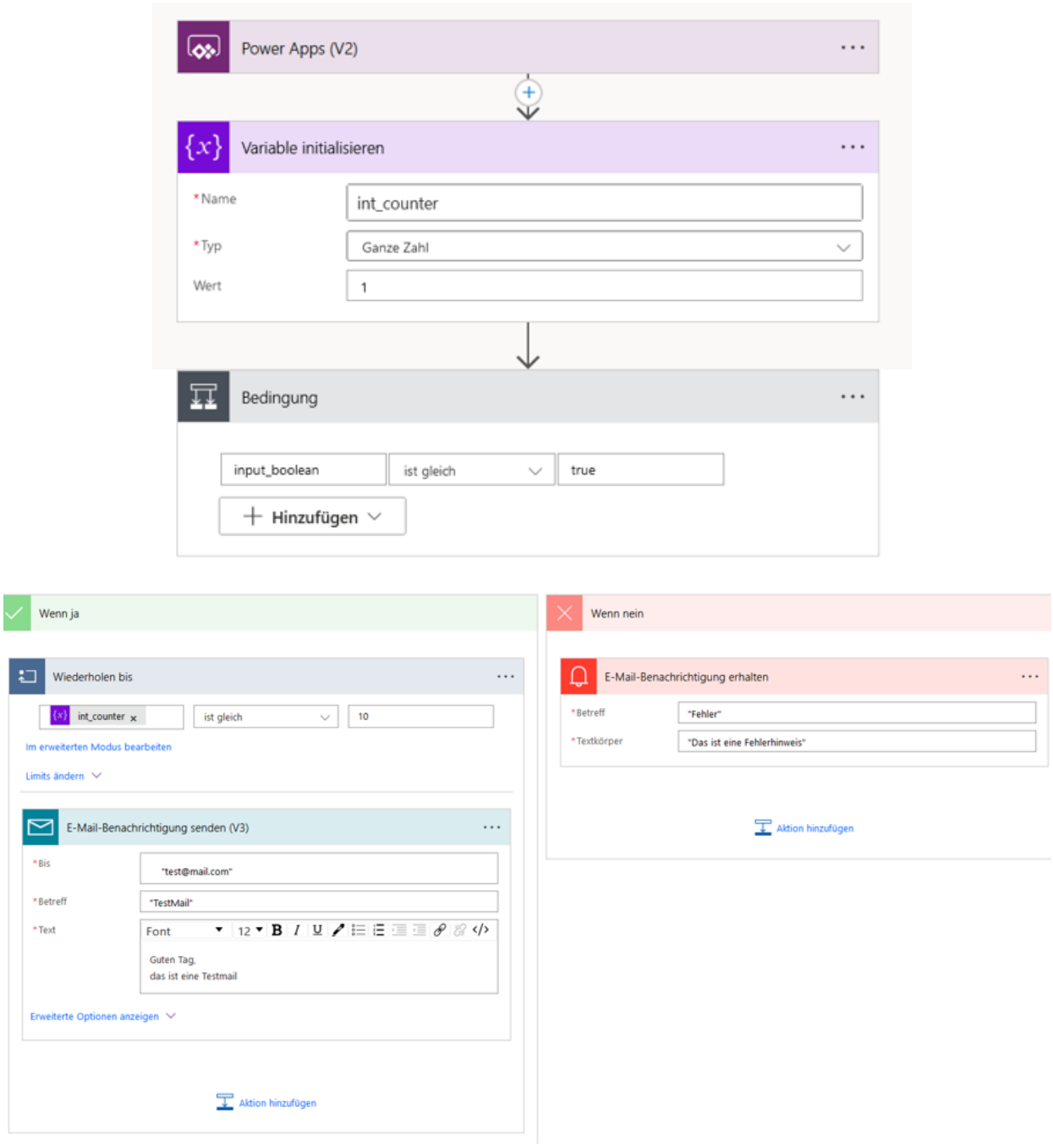

Abbildung 18: Beispiel eines PowerFlows

Das Dashboard wurde nicht gleichzeitig von zwei Teammitgliedern gleichzeitig entwickelt, sodass keine Erfahrung in der parallelen Bearbeitung gesammelt werden konnte. Allerdings zeigt eine Internetrecherche, dass eine Anbindung an Git möglich ist, und so ein paralleles Arbeiten an derselben App möglich sein soll.

PowerApps hat in der Entwickleroberflächen die Herausforderungen von Low-Code Anwendungen, auf welche die Teammitglieder schon bei anderen Lösungen gestoßen sind. Je komplexer der Anwendungsfall, desto mehr Anpassung mittels PowerFx wurde notwendig. Der Code musste dabei in unterschiedlichen Bereichen der App aufgeteilt werden, zwischen welchen mit vielen manuellen Klicks gewechselt werden muss. So konnte es vorkommen, dass man sich erst wieder erinnern musste, in welchen Teilbereichen welcher Code hinterlegt wurde. Der Wechsel der Bereiche war dabei in der Summe zeitintensiv und erschwerte das Verständnis eines Vorgangs auf einen Blick.

Anstatt eine Tabelle direkt in PowerApps zum Speichern der Daten zu verwenden, wurde in diesem Projekt eine Postgres Datenbank mit einem Konnektor an PowerApps angebunden. So ist es möglich, dass das Dashboard und das Backend der mobilen Anwendung immer auf demselben Stand sind. Der Konnektor funktionierte, allerdings zeigten sich immer wieder Performanceschwächen während der Verwendung der entwickelten App. Auch mussten Workarounds verwendet werden, um zum Beispiel das Laden eines Bildes in eine für Postgres unterstützten Datentyp umzuwandeln.

Das Verständnis und der Aufbau des Designs sollten auch für andere Personen einfach sein und Anpassungen möglich sein. Allerdings sind allgemeine Kenntnisse weiterhin notwendig, da für eine gute Darstellung Elemente gruppiert werden müssen und bei beispielsweise Breite und Höhe auch auf Elternelemente verwiesen werden sollte.

Eine Anpassung der Businesslogik ist speziell in diesem Projekt schwieriger, da die Tabellenstruktur in der externen Postgres definiert ist, welche von der PowerApps Entwicklerumgebung aus nicht angepasst werden kann.

Die gesetzten Hoffnungen auf eine schnellere Entwicklung mittels PowerApps im Gegensatz zur nativen Entwicklung einer Web-App haben sich im Allgemeinen bestätigt. Der Aufwand wurde etwas zu niedrig angesetzt, die Fortschritte waren allerdings insbesondere zum Ende der Entwicklung sehr schnell, nachdem eine schnelle und steile Lernkurve durchlaufen worden ist.

8.2.4 Deployment

Herausforderung

Die Bereitstellung eines Backends und einer Datenbank war bei dem Projekt sehr wichtig, um die Entwicklung andere Komponenten zu ermöglichen. Demzufolge war die Wahl der geeigneten Plattform relevant. Ziel davon war, eine sichere, kostengünstige, skalierbare und simple Möglichkeit zu finden, welche zu den Projektanforderungen passt.

Ursprünglich geplante Lösung: Google Cloud Platform (GCP)

Zunächst wurde die Implementierung auf Google Cloud Platform (**GCP**) unter Verwendung von GitHub Actions in Betracht gezogen. Dieser Ansatz bot eine hohe Skalierbarkeit, Automatisierung und Robustheit, erforderte jedoch eine komplexe Konfiguration von Benutzern, Zugriffsschlüsseln und Pipelines. Nach mehreren Versuchen gelang es, eine Pipeline zu erstellen, der einen Docker-Container für das Go-Programm generierte und diesen in einem privaten Registry auf **GCP** registrierte. Aus diesem Container wurde dann ein öffentlich zugänglicher Dienst im Rahmen der Pipeline erstellt.

Herausforderungen bei GCP

Der nächste Schritt war die Definition des Datenbankschemas mithilfe der Bibliothek Gorm in Go. Dazu wurde eine zweite Pipeline erstellt, um einen Container für die PostgreSQL-Datenbank und einen Dienst zu generieren, mit dem sich das Go-Backend verbinden konnte. Ziel war das Datenbankschema zu erstellen und so eine vollständige Datenbank und ein online verfügbares Go-Backend zu erhalten.

Es stellte sich jedoch heraus, dass GCP-Dienste nicht für das Hosting von Datenbanken geeignet sind. Die GCP-Lösung Cloud SQL erwies sich als komplex einzurichten und erlaubte keinen direkten Zugriff auf die Datenbank, weder vom Go-Backend noch von externen Verbindungen, was für PowerApps erforderlich war.

Alternative Lösung: Fly.io

Angesichts der Einschränkungen von GCP wurde Fly.io als alternative Plattform gewählt. Die Konfiguration ist einfach und automatisiert die Erstellung von Docker-Images und Diensten. Die Erstellung einer Datenbank erfordert nur wenige Klicks.

Der einzige Nachteil war, dass die Datenbank nicht von außerhalb von Fly.io aus Standardweise zugänglich war. Um dies zu beheben, wurde eine statische IPv4-Adresse zugewiesen (mit zusätzlichen monatlichen von 2 Dollar Kosten verbunden). Mit dieser Konfiguration funktioniert die Verbindung problemlos und sowohl die Datenbank als auch das Go-Backend sind öffentlich zugänglich.

Fazit

Die Implementierung eines Backends und einer Datenbank in der Cloud für ein Go-Programm ist mit verschiedenen Herausforderungen verbunden. Die Wahl der geeigneten Plattform hängt von den spezifischen Anforderungen des Projekts ab. In diesem Fall erweist sich Fly.io als eine praktikable und einfach zu verwendende Lösung, insbesondere für Projekte mit Anforderungen an die öffentliche Zugänglichkeit.

8.2.5 Refill-Station

Thema	Herausforderung	Lösung
Verbindung und Steuerung vom NFC-Lesegerät	Für die reibungslose Kommunikation mit dem gewählten NFC-Modul (PN532) spielte die Auswahl geeigneter Bibliotheken und Treiber eine wichtige Rolle.	Durch gründliche Recherche und Auswahl geeigneter Bibliotheken wurde die Kommunikation mit dem NFC-Lesegerät optimiert.
Steuerung von Peripherien	Die Zuordnung der GPIO-Pins des Raspberry Pi 4B zu den jeweiligen Hardwarekomponenten und die Implementierung der Steuerungslogik waren erforderlich.	Mithilfe von diversen in Internet zu findenden Tutorials, geeignete Bibliotheken und der Dokumentation des Herstellers [5] wurde die Implementierung möglich. In diesem Zusammenhang erwies sich die Bibliothek gpiozero als besonders empfehlenswert für die einfache und intuitive Steuerung der Raspberry Pi-Peripheriegeräte mit Python.

[5] https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

Verbindung mit dem Backend und MQTT-Broker	Die Anbindung an ein separates Backend-System zur Abfrage von Nutzerpräferenzen und zur Meldung von Wassertransaktionen war notwendig. Dazu war die Anbindung zum externen MQTT-Broker auch nötig.	Die Implementierung von jeweils einer separaten Schnittstelle (Backend und MQTT) ermöglichte die flexible Verwaltung von Nutzerpräferenzen, Wassertransaktionen und die zuverlässige Datenübertragung mit der mobilen Anwendung des Users und dem MQTT-Broker.

Tabelle 13: Entwicklungsherausforderungen Refill-Station

Die Wahl des Raspberry Pi 4 als Hardwareplattform erwies sich als ideal aufgrund seiner Leistungsfähigkeit, Flexibilität und des breiten Angebots an Bibliotheken und Tools. Python bot sich als Programmiersprache aufgrund seiner Benutzerfreundlichkeit, großen Community und umfangreichen Bibliotheken für die Implementierung der Anwendung an. Die Kombination von diesen und die Bibliothek gpiozero hat sich als ideale Wahl für die Entwicklung der Refill-Station erwiesen und zeigt das Potenzial dieser Plattform, Programmiersprache und Bibliothek für die Realisierung innovativer und intelligenter Lösungen in verschiedenen IoT Projekten.

9 Tabellenverzeichnis

10 Abbildungsverzeichnis

11 Anhang

11.1 Abkürzungen

- **GCP**: Google Cloud Platform
- **Fraunhofer IESE:** Fraunhofer-Institute for Experimental Software-Engineering
- **PSE1:** Projekt Software-Engineering 1
- **PSE2:** Projekt Software-Engineering 2
- **REFILL-KL:** Refill Kaiserslautern
- **SDGs:** Sustainable Development Goals
- **SWA**: Software Architecture
- **UN:** United Nations

11.2 Glossar

- **Docker-Container:** standardisierte, leichtgewichtige und portable Umgebung für die Ausführung von Anwendungen [6].
- **GitHub Actions:** Plattform für Continuous Integration und Continuous Delivery (CI/CD), mit der man Build-, Test- und Bereitstellungspipeline automatisieren kann [7].
- **GORM:** leistungsstarkes ORM (Object-Relational Mapping) für Go. Es bietet eine Abstraktionsschicht für die Interaktion zwischen Go-Anwendungen und relationalen Datenbanken [8].
- **GPIO-Ports:** (engl. General-purpose Input/Outputs) vielseitige Pins auf Mikrocontrollern, die als Ein- oder Ausgänge programmierbar sind [9].
- **Mifare Classic:** Mifare Classic ist eine Art der kontaktlosen Chipkartentechnologie, die im Nahbereich mit 13,56 MHz Frequenz funktioniert. Sie zählt zu den weltweit am meisten verbreiteten Technologien für kontaktlose Smartcards (über 10 Milliarden Karten im Umlauf). Mithilfe von Nahfeldkommunikation (NFC) ermöglicht Mifare Classic das Speichern und Übertragen von Daten [10].
- **MQTT:** leichtes Messaging-Protokoll für Sensornetzwerke und das Internet der Dinge (IoT). Es zeichnet sich durch Einfachheit, Zuverlässigkeit und Effizienz aus [11].
- **MQTT-Topic:** Zeichenkette, die Nachrichten in einem MQTT-Netzwerk kategorisiert. Publisher senden Nachrichten an Topics, und Subscriber abonnieren Topics, um diese Nachrichten zu empfangen [12].
- **Platform-as-a-Service:** Cloud-Computing-Modell, das Entwicklern eine Plattform zur Verfügung stellt, auf der sie Anwendungen erstellen, betreiben und verwalten können. Die Plattform beinhaltet dabei die gesamte Infrastruktur, wie Server, Betriebssysteme, Datenbanken und Middleware, sodass sich Entwickler auf die eigentliche Anwendungsentwicklung konzentrieren können [13].

---

[6] https://www.opc-router.de/was-ist-docker/
[7] https://docs.github.com/de/actions/learn-github-actions/understanding-github-actions
[8] https://gorm.io/
[9] https://de.wikipedia.org/wiki/GPIO
[10] https://de.wikipedia.org/wiki/Mifare
[11] https://www.iot-mesh.de/mqtt/
[12] https://www.iot-mesh.de/mqtt/
[13] https://developer.ibm.com/technologies/paas/

- **PN532:** Ein PN532 Lesegerät ist ein kompaktes Gerät, das mithilfe des PN532 Chips Daten von RFID-Tags und NFC-Geräten auslesen und beschreiben kann [14].
- **Raspberry Pi 4 B:** vielseitiger und leistungsstarker Einplatinencomputer, der sich für eine Vielzahl von Anwendungen eignet. Seine kompakte Bauweise, sein geringer Stromverbrauch, seine vielfältigen Schnittstellen, seine Erweiterbarkeit und seine Programmierbarkeit machen ihn zu einer beliebten Wahl für IoT Projekten [15].
- **Rechenknoten:** Im Rahmen verteilter Systeme definiert man einen Rechenknoten als eigenständiges System, welches in einem Netzwerkverbund mit anderen Knoten agiert, um gemeinsam Aufgaben zu erledigen. Diese Knoten können dabei physisch realisierte Computer, virtuelle Maschinen oder sogar spezialisierte Hardware-Einheiten sein [16].
- **Team Gaia:** Studentische Arbeitsgruppe aus der Vorlesung PSE1.

[14] https://botland.de/rfid-module-und-tags/8240-rfid-nfc-pn532-1356-mhz-i2c-spi-modul-karte-und-schlusselanhanger-5904422375775.html
[15] https://fastercapital.com/de/inhalt/Entdecken-Sie-die-Welt-der-Einplatinencomputer-mit-Raspberry-Pi.html
[16] https://www.atlassian.com/de/microservices/microservices-architecture/distributed-architecture