Ausarbeitung

Entwurf einer Datenbank für ein Verwaltungssystem von Lebensmitteln in einem Kühlschrank

Private Hochschule für Wirtschaft und Technik Vechta/Diepholz
vorgelegt von: Claas Meints
Semester: 6

Inhaltsverzeichnis

1 Beschreibung der Anforderungen

1.1 Das Kühlschrank-Verwaltungssystem

Die im Rahmen dieser Ausarbeitung zu entwerfende Datenbank ist Teil eines Projektes zur Entwicklung eines Verwaltungssystems für Lebensmittel, die in Kühl- oder Gefrierschränken gelagert werden. Ziel des Projektes ist es, eine günstige Alternative zu smarten Kühlschränken zu entwickeln, die es ermöglicht die Lagerung von Lebensmitteln zu überwachen, ohne einen neuen Kühlschrank anschaffen zu müssen. Mit Hilfe der Lösung soll der Verschwendung von Lebensmitteln entgegengewirkt werden.
In vielen Haushalten werden aufgrund mangelnder Planung bzw. mangelndem Überblick über die Situation im Kühlschrank teilweise große Mengen an Lebensmitteln entsorgt. Zum Einen geraten Lebensmittel die im Kühlschrank weiter hinten stehen in Vergessenheit und werden nicht rechtzeitig verwendet. Zum Anderen werden Lebensmittel gekauft, ohne dass klar ist, wann und in welcher Form dieses Benötigt werden. Die zu entwickelnde Lösung soll beiden Problemen entgegenwirken. Durch die Überwachung der Lebensmittellagerung wird die Planung von Einkauf und Verwertung dieser erleichtert.
Aus Benutzer:innen Sicht stellt eine Smartphone-App das zentrale Element der Lösung dar. Über diese können Kühl- und Gefrierschränke, oder andere Lagerstätten für Lebensmittel eingebunden und verwaltet werden. Auf einen Blick sollen Benutzer:innen alle Lebensmittel in all ihren Lagerstätten aufgelistet bekommen. Zusätzlich soll der Füllstand bzw. die Menge der jeweiligen Lebensmittel und die Haltbarkeit angezeigt werden. Benutzer:innen sollen zusätzlich die Möglichkeit haben detaillierte Informationen wie Zutatenlisten einzelner Produkte ab zu rufen. Mit Hilfe der App können Benutzer:innen darüber hinaus ihren Einkauf planen und nicht automatisierbare Ein- bzw. Auslagerungsprozesse manuell einpflegen.
Damit Benutzer:innen die Lösung tatsächlich verwenden muss die Bedienung so komfortabel wie möglich sein. Kritisch ist besonders die Anzahl der manuell zu pflegenden Prozesse. Lebensmittel sollen mit Hilfe eines IoT-Gerätes automatisch Ein- bzw. Ausgelagert werden können. Der Großteil an Produkten wird mit Hilfe eines Barcode-Scanners

erfasst. Produkte ohne Barcode sollen mittels bildgestützter Verfahren erkannt werden. Um die IoT-Geräte optimal einbinden und verwalten zu können sollte die Lösung eine IoT-Plattform integrieren.
Bei der Entwicklung der Lösung werden spätere potentielle Erweiterungen mit bedacht. Unter dem Aspekt der Lebensmittelverschwendung wäre die Integration einer Rezepteverwaltung bzw. einer Ernährungsplanung nützlich. Die Analyse von Nutzungsdaten kann potentiell dazu Verwendet werden die Zahl der nötigen manuellen Eingriffe von Benutzer:innen zu reduzieren. Abseits der Anwendungsfälle des Lebensmittelverwaltungssystems, kann eine doppelte Nutzung der in Kühlschränken installierten IoT-Geräte sinnvoll sein. So könnten zum Beispiel Temperaturdaten von Kühl- und Gefrierschränken interessant für Projekte sein, die den Energieverbrauch von Haushalten optimieren.
Die im Rahmen dieser Ausarbeitung zu entwickelnde Lösung stellt das MVP (Minimum Viable Product), das minimal funktionsfähige Produkt des gesamten Verwaltungssystems dar. Der Fokus liegt dabei auf dem Entwurf der Datenbank, sowie den zum Test der Datenbank benötigten Komponenten. Es wird Prototyp für das Backend in Node.js entwickelt, dass an die Datenbank angebunden ist. Dieses enthält die nötige Verwaltungslogik des Systems und bietet eine REST-API für die Anbindung der Smartphone-App. Über die App wird es in der MVP-Phase nur möglich sein die Liste der Aktuell eingelagerten Lebensmittel an zu zeigen.

1.2 Anforderungen an die Datenbank

Aus dem beschriebenen Projektziel ergeben sich die Anforderungen an die zu entwerfende Datenbank. Da die Daten unterschiedlicher Benutzer:innen durch ein Zentrales Backend verarbeitet werden wird eine Benutzer:innenverwaltung benötigt. Die Datenbank muss also die Möglichkeit bieten Benutzer:innen inklusive ihrer Anmeldedaten zu speichern. Darüber hinaus müssen allen Benutzer:innen ihre Geräte (also Kühl- oder Gefrierschränke, etc.) sowie ihre Einkaufslisten zugeordnet werden. Alle nicht einzelnen Benutzer:innen zuordbaren Entitätsklassen sollen möglichst übergreifend genutzt werden.
Neben den Benutzer:innen müssen auch die Geräte der Benutzer:innen verwaltet und entsprechend in der Datenbank gespeichert werden. Jedes der in der Datenbank gespeicherten Geräte repräsentiert dabei ein real existierendes IoT-Gerät, welches an einer Lagerstätte für Lebensmittel installiert ist. Das Gerät stellt damit auch die Schnittstelle für ein an zu knüpfende IoT-Plattform dar.

Damit Benutzer:innen in der Lage sind ihren Lebensmittelhaushalt zu überwachen, müssen die Lebensmittel, die eine Lagerstätte enthält gespeichert werden. Benutzer:innen sollen zugriff auf eine Reihe von Informationen über die Lebensmittel in ihren Kühlschränken haben. Es müssen also die Menge, das Öffnungs- bzw. Kaufdatum und die EAN (European Article Number) aller Produkte gespeichert werden. Damit Produkte aus dem Kühlschrank herausgenommen und später wieder hineingelegt werden können, ist es außerdem sinnvoll eine Produkthistorie zu speichern. Zur besseren Übersicht der Benutzer:innen sollen Produkte außerdem Kategorisiert werden können. Auch diese Eigenschaft der Produkte muss durch das Datenbankenmodell abbildbar sein. Die Einordnung von Produkten in Kategorien ist darüber hinaus für weitere optionale Funktionen, wie die Einbindung von Rezepten hilfreich.
Damit für alle Benutzer:innen das Erscheinungsbild von Anwendungen die auf die Datenbank zurückgreifen gleichartig ist, sollen auch Anzeigeelemente, die spezifischen Entitäten zugeordnet sind in der Datenbank gespeichert werden. So lässt sich sicherstellen, dass zum Beispiel das Icon einer Produktkategorie überall identisch ist und nicht in unterschiedlichen Versionen oder Anwendungen variiert. Auch wird so verhindert, dass bei der Einführung neuer Kategorien die Anwendungsprogramme angepasst werden müssen.
Neben der Speicherung des Ist-Zustandes soll aus diesem außerdem automatisiert eine Einkaufsliste erstellt werden können. Produkte, welche immer vorhanden sein sollen, müssen automatisch auf eine Einkaufsliste gesetzt werden, welche durch den Benutzer bearbeitet werden kann. Perspektivisch ist auch die automatisierte Verarbeitung von Rezepten zu einer Einkaufsliste denkbar und sollte durch das Datenbankmodell abbildbar sein.
Da bereits bekannt ist, dass es sich bei der zu entwickelnden Lösung um ein MVP handelt, ist die Datenbank wann immer möglich so zu entwerfen, dass die bereits geplanten späteren Funktionen auf einfache weise integriert werden können. So soll verhindert werden, dass die Daten aus der Datenbank zu einem späteren Zeitpunkt aufwändig migriert werden müssen.

1.3 Abgrenzung der Anforderungen

Wie bereits beschrieben, soll im Rahmen dieser Ausarbeitung lediglich ein MVP des gesamten Projekts entwickelt werden. Es ist also nicht gefordert bereits alle Anforderungen an die fertige Lösung zu erfüllen. Viele Funktionen werden in dieser ersten Entwicklungsiteration zunächst nur simuliert oder vollständig ausgelassen. Die zu entwickelnde Smartphone-App dient vor allem zum anschaulichen Test der zu entwickelnden API und

ist noch nicht als fertiges Anwendungsprogramm zu sehen. Ein großteil der späteren Funktionalität wird noch nicht implementiert und auch das Benutzer:inneninterface hat noch nicht den Anspruch intuitiv oder ansprechend zu sein. Für Tests der Gesamten Lösung ist eine prototypische Implementierung aber durchaus geeignet.
Die Anbindung von Geräten wird in der ersten Iteration vollständig simuliert. Da noch keine Geräte entwickelt werden, ist auch die Anbindung einer IoT-Plattform noch wenig sinnvoll. Die in dieser Phase bereits zu entwickelnde Schnittstelle für die Gerätekommunikation im Backend wird durch ein Python-Skript angesprochen. Mit diesem Skript soll das Herausnehmen und das Hineinlegen von Produkten mit einer bestimmten simuliert werden. Das Anlegen neuer Geräte kann ohne die Verwendung einer IoT-Plattform noch nicht realitätsgetreu durchgeführt werden. Auch diese Funktion wird durch ein Python-Skript und eine zu entwickelnde API simuliert.
Eine spätere hoch skalierbare Lösung soll bei einem Cloud-Dienstleister betrieben werden. Da sich die Lösung noch in der Prototypphase befindet und noch von keinen Benutzer:innen verwendet wird, sollen die Datenbank und das Backend zunächst lokal auf dem Rechner des Entwicklers betrieben werden. Die Lösung soll vollständig in Entwicklungscontainern ausführbar sein, um komfortabel Tests durchführen zu können.
Auf verschiedene Sicherheitsmaßnahmen kann in dieser Phase verzichtet werden, da noch keine Daten echter Benutzer:innen verarbeitet werden und die ein unerlaubter Zugriff auf die API oder die Datenbank in diesem Stadium noch nicht kritisch ist. Die sowohl das Backend, als auch die Datenbank werden vom Entwickler lokal betrieben und sind somit lediglich potentiellen Angreifern im eigenen Netzwerk ausgeliefert. Da der Wert der Daten in der Datenbank zu diesem Zeitpunkt gering ist das Risiko vertretbar. Benutzer:innen oder Anwendungsprogramme sollen sich für die Benutzung der API weder Autorisieren noch Authentifizieren müssen.

2 Modellierung der Datenbank

2.1 Konzeptionelles Schema

Aus den in Kapitel 1.2 beschriebenen Anforderungen lassen sich bereits verschiedene Entitätsklassen und deren Attribute ableiten. Das Resultierende Konzeptionelle Schema ist in Abbildung 2.1 dargestellt. Für die Verwaltung von Benutzer:innen wird die Entitätsklasse *fridge_user* verwendet. An diese wird zunächst nur die Anforderung gestellt, dass sich Benutzer:innen anmelden können. Um das zu gewährleisten werden der *login* von Benutzer:innen sowie ein Hash des Passwortes gespeichert.
Alle Benutzer:innen sollen die Möglichkeit haben mehrere Geräte zu überwachen. Gleichzeitig kann zum Beispiel in einer Wohngemeinschaft ein Gerät auch mehreren Benutzer:innen gehören bzw. von diesen Überwacht werden. Es wird also eine Entitätsklasse *device* benötigt, die eine M zu M Beziehung zu *fridge_user* hat. Ein Gerät soll neben einer eindeutigen ID zur Identifikation des Gerätes auch einen Namen haben, den Benutzer:innen vergeben können, um ihre Geräte auseinander halten zu können.
Damit Benutzer:innen auch Produkte in ihren Kühlschränken lagern können, wird eine weitere Entitätsklasse *product* benötigt. Zwischen *device* und *product* gibt es eine M zu M Beziehung, da unterschiedliche Kühlschränke jeweils die gleichen Produkte enthalten können, ein Gerät aber auch mehrer verschiedene Produkte enthalten kann. Produkte repräsentieren dabei sowohl Lebensmittel mit Barcode, als auch solche ohne. Bei Produkten mit Barcode können die wesentlichen Informationen zu dem jeweiligen Produkt mit der von einer externen API abgerufen werden. Diese Informationen sind in Abbildung 2.1 als Attribute mit gestrichelter Linie dargestellt. Neben den produktspezifischen Informationen sind für das Lebensmittelverwaltungssystem zusätzliche Informationen interessant. So muss zum Beispiel bekannt sein, wie lange ein Produkt bereits gelagert wird bzw. wie lange es bereits geöffnet ist. Auch der Füllstand eines Produktes muss gespeichert werden. Um der Anforderung einer Produkthistorie gerecht zu werden, muss auch der Zeitpunkt der Entnahme eines Produktes gespeichert werden.

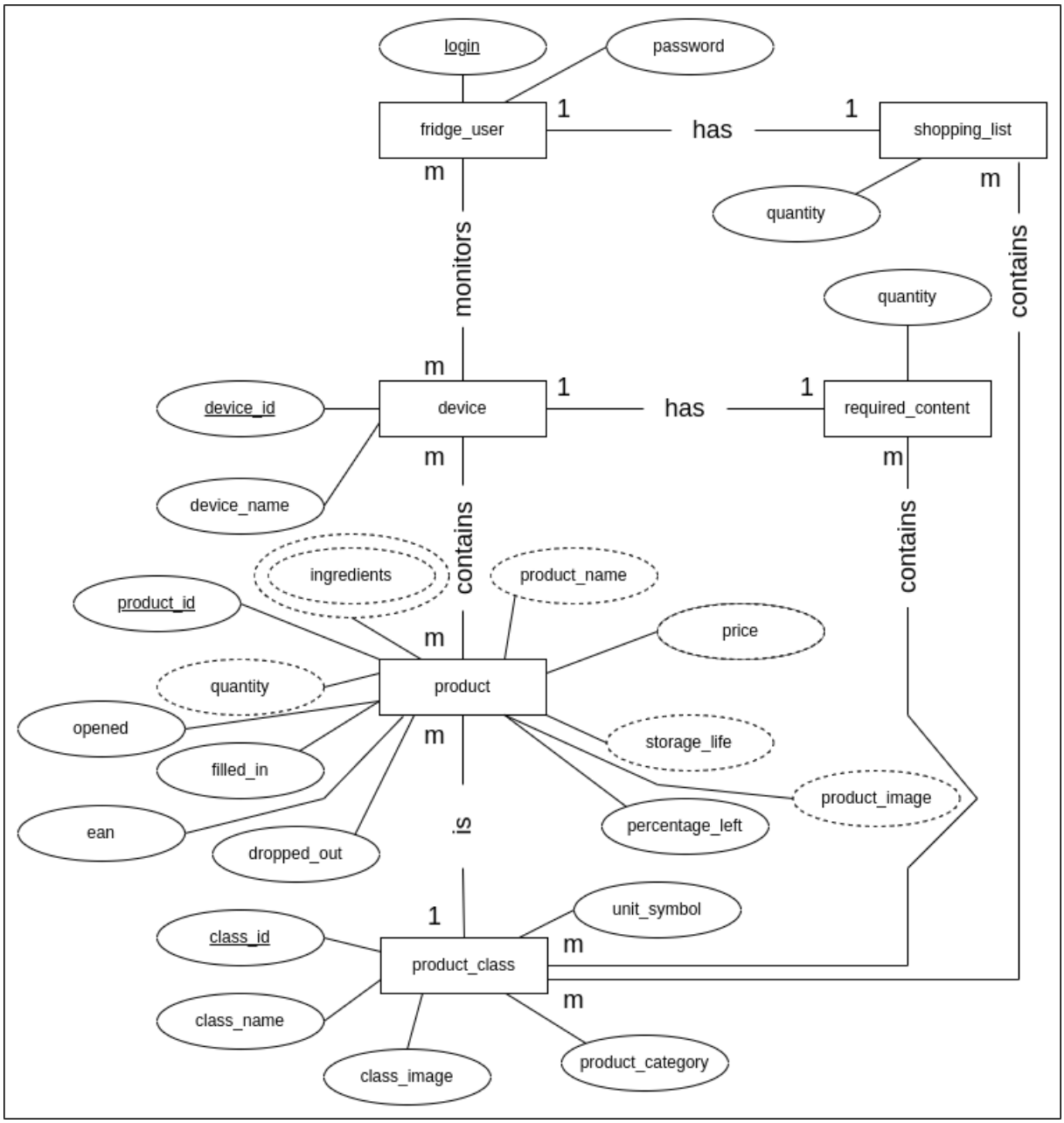

Abbildung 2.1
Konzeptionelles Schema

Für Produkte ohne Barcode müssen auch die produktspezifischen Informationen in der Datenbank abbildbar sein. Dazu wird die Entitätsklasse *product_class* verwendet. Jedes Produkt ist bis zu einer Produktklasse zu zu ordnen, es gibt als eine 1 zu M Beziehung. Auch Produkte mit Barcode können einer Produktklasse zugeordnet werden, so kann zum Beispiel das Produkt mit der *4004980806405* der Klasse *Erdnüsse* zugeordnet werden. Produktklassen haben neben einer eindeutigen ID die Attribute *class_name*, *class_image* und *unit_symbol*. Darüber hinaus lassen sich Produktklassen einer Produktkategorie zuordnen. Erdnüsse würden zum Beispiel der Kategorie Nüsse angehören.
Damit Benutzer:innen Einkaufslisten erstellen können, gibt es zusätzliche die Tabelle *shopping_list*. Diese kann von Benutzer:innen manuell bearbeitet werden. Jedem Gerät ist zusätzlich eine Tabelle *required_content* zugeordnet. Fragen Benutzer:innen ihre Einkaufsliste ab, werden alle Produkte von *shopping_list*, sowie alle von *required_content*, welche nicht im Kühlschrank vorrätig sind gelistet.

2.2 Physisches Schema

Im nächsten Schritt wird aus dem Konzeptionellen Schema das Physische Schema erstellt. Dieses ist in Abbildung 2.2 dargestellt. Während die Tabelle *fridge_user* unverändert übernommen werden kann, muss die M zu M Beziehung zum Gerät aufgelöst werden. Es wird die zusätzliche Tabelle *fridge_user_device_relation* eingeführt, welche die Geräte den Benutzer:innen zuordnet. Gemäß dem Standardvorgehen für die Auflösung von M zu M Beziehungen werden die Primärschlüssel von *fridge_user* und *device* als Fremdschlüssel der zusammengesetzte Primärschlüssel von *fridge_user_device_relation*. Wie auch *fridge_user* kann die Tabelle *device* unverändert übernommen werden.
Bei der Auflösung der M zu M Beziehung zwischen *device* und *product* wechseln einige Attribute von *product* zu der neu geschaffenen Relationstabelle *device_content*. Während Produkte in diesem Kontext abstrakte Beschreibungen eines Produkttyps repräsentieren, sind die Entitäten der Tabelle *device_content* als tatsächliche Lebensmittel in einem Kühlschrank zu verstehen. Entsprechend sind Informationen, wie Einlagerungs-, Öffnungs- und Auslagerungsdatum, sowie der Füllstand in *device_content* geführt. Der Primärschlüssel wird aus dem Fremdschlüssel des Gerätes und dem Einlagerungsdatum gebildet, da jedes Gerät zeitgleich nur ein Produkt erfassen kann.

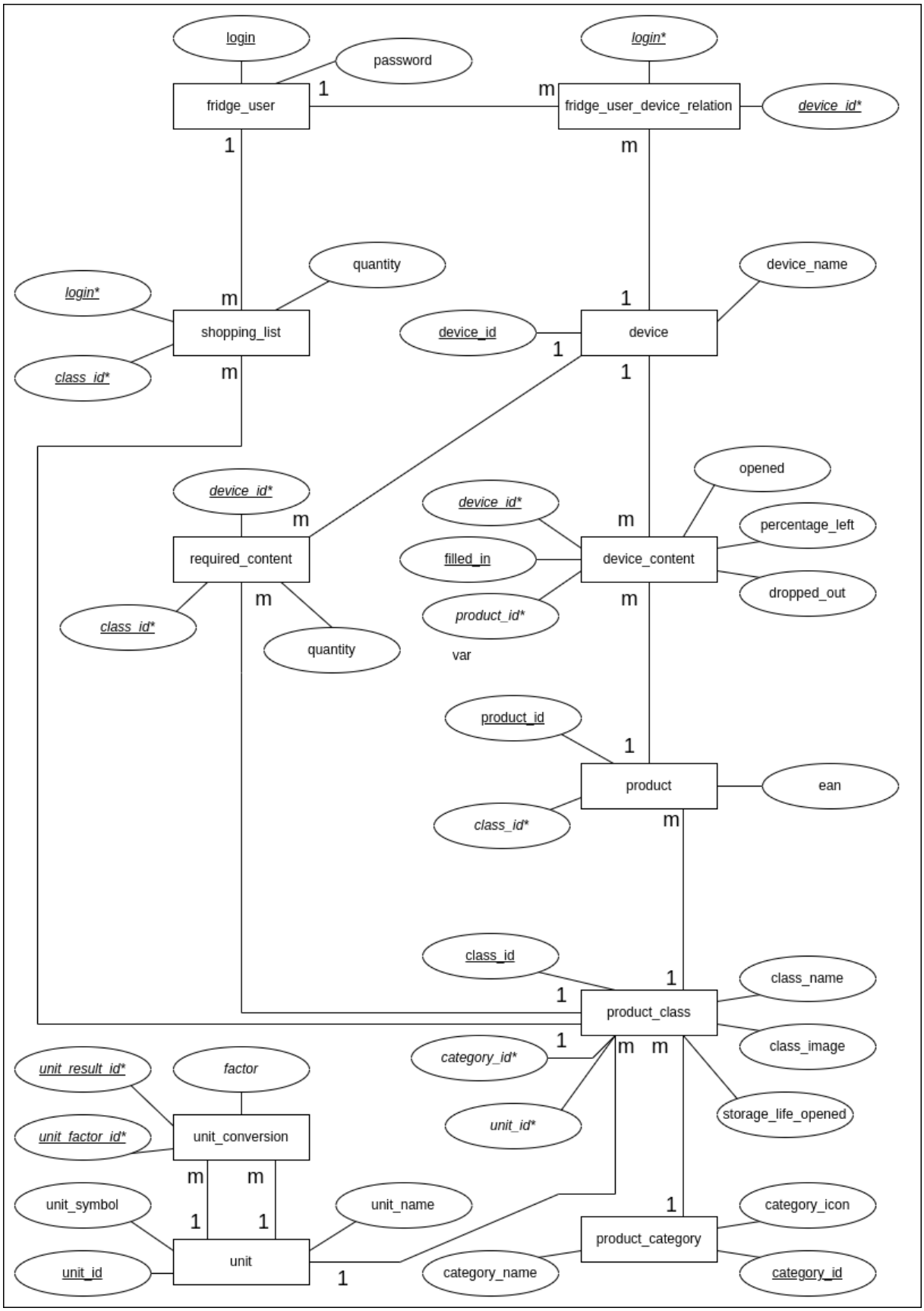

Abbildung 2.2
Physisches Schema

Die 1 zu M Beziehung zwischen *product* und *product_class* kann unverändert bestehen bleiben. Der Tabelle *product_class* wird ein zusätzliches Attribut *storage_life_opened* hinzugefügt. Dies ist notwendig, da diese Information sonst bei Produkten ohne Barcode nicht verfügbar wäre. Darüber hinaus werden die beiden Attribute *product_category* und *unit(_symbol)* in eingene Tabellen umgewandelt. So wird sichergestellt, dass nur zuvor festgelegte Werte angenommen werden können. Außerdem wird sichergestellt, dass Produkte nach Einheit und Kategorie sortiert werden können. Auch zusätzliche Informationen, wie der Name einer Einheit oder ein Icon für die Darstellung einer Kategorie in Anwendungsprogrammen sind so abbildbar. Neben der Tabelle *unit* gibt es eine zusätzliche Einheit *unit_conversion*. Diese bildet Informationen über die Umrechnung von Einheiten ineinander ab. Möchten Benutzer:innen zum Beispiel stets 2Kg Erdnüsse vorrätig haben, hat aber eine 200g Dose Erdnüsse im Regal, kann mit Hilfe dieser Informationen dennoch die fehlende Menge an Erdnüssen bestimmt werden.
Bei der Überführung der Tabellen *required_content* und *shopping_list* ergeben sich weitere Veränderungen. Beide Tabellen werden zu Relationstabellen, welche jeweils Produktklassen entweder einem Gerät oder einem:einer Benutzer:in zuordnet. Die Tabellen enthalten jeweils die beiden Fremdschlüssel als zusammengesetzten Primärschlüssel sowie die Menge der jeweiligen Produktklasse, die benötigt wird bzw. eingekauft werden soll.

2.3 Nachweis der Normalformen

Um eine Reihe von häufig auftretenden Modellierungsfehlern wie Redundanzen zu vermeiden und Anomalien im Betrieb zu verhindern wird im nächsten Schritt das Datenbankmodell normalisiert. Im Zuge dieser Normalisierung wird das Modell auf die Einhaltung der ersten drei Normalformen hin überprüft und gegebenenfalls angepasst.
Um die Einhaltung der 1NF (erste Normalform) zu prüfen, müssen die Attribute aller Tabellen betrachtet werden. Es ist zu prüfen, ob diese atomar sind, also sich nicht weiter zerlegen lassen. In Tabelle 2.1 sind alles Attribute, mit Ausnahme von Fremdschlüsseln dargestellt. Für alle Attribute außer den beiden Bilder *class_image* und *category_icon* ist anhand der Beispiele ersichtlich, dass diese atomar sind. Bei den Bilddaten handelt es sich um listenwertige Attribute, da diese aber nicht separat voneinander verwendet werden sondern ein einzelnes Bild repräsentieren, liegt dennoch kein verstoß gegen die 1NF vor. Da für keines der Attribute des Schemas ein Verstoß gegen die 1NF vorliegt, erfüllen auch alle Tabellen die 1NF.
Für den Nachweis der 2NF (zweite Normalform) darf kein Attribut einer Tabelle bereits

Tabelle	Attributname	Beispielwert	atomar
fridge_user	login	”claas”	✓
fridge_user	password	”$2a$12$C16E0aHgerPbr...”	✓
device	device_id	1	✓
device	device_name	”Kühlschrank”	✓
device_content	filled_in	2022-09-23 12:22:03.142+00	✓
device_content	opened	2022-09-23 12:22:03.142+00	✓
device_content	dropped_out	2022-09-23 12:22:03.142+00	✓
device_content	percentage_left	100	✓
product	product_id	1	✓
product	ean	4004980806405	✓
product_class	class_id	1	✓
product_class	class_name	”Erdnüsse”	✓
product_class	class_image	”253, 80, 78, 10, 40, 43, ...”	✗
product_class	storage_life_opened	7	✓
product_category	category_id	1	✓
product_category	category_name	”Nüsse”	✓
product_category	category_icon	”253, 80, 78, 10, 40, 43, ...”	✗
unit	unit_id	1	✓
unit	unit_symbol	”g”	✓
unit	unit_name	”Gramm”	✓
unit_conversion	factor	1000	✓
required_content	quantity	1	✓
shopping_list	quantity	1	✓

Tabelle 2.1
Erste Normalform der Attribute

von einem Teil des zusammengesetzten Primärschlüssels abhängig sein. Die 2NF ist also für alle Tabellen mit einfachem Primärschlüssel trivialerweise erfüllt. Damit ist der Nachweis für *fridge_user_device_relation*, *device_content*, *unit_conversion*, *required_content* und *shopping_list* zu erbringen. Da *fridge_user_device_relation* außer dem zusammengesetzten Primärschlüssel keine weiteren Attribute enthält, ist auch hier die 2NF trivialerweise erfüllt. Die Tabelle *device_content* enthält neben dem Primärschlüssel Eigenschaften, die einem Konkreten Lebensmittel zugeordnet werden. Diese sind nicht von der *device_id* abhängig, da ein Gerät unterschiedliche Lebensmittel enthalten kann. Auch von *filled_in*-Datum sind diese Attribute unabhängig, da in unterschiedliche Geräten unterschiedliche Lebensmittel zum gleichen Zeitpunkt (gleiche Millisekunde) hineingelegt werden können. Auch für *unit_conversion* ist die 2NF erfüllt, da Umrechnungsfaktor zwischen zwei Einheiten nicht nur von einer der Einheiten abhängig sein kann. Für *required_content* und *shopping_list* ist jeweils die Menge von der Produktklasse unabhängig, da unterschiedli-

che Geräte bzw. Benutzer:innen unterschiedlich viel einer Produktklasse voraussetzen bzw. einkaufen können. Auch umgekehrt gilt, dass Benutzer:innen bzw. Geräte von unterschiedlichen Produktklassen unterschiedliche Mengen einkaufen bzw. voraussetzen können, die *quantity* als voll vom zusammengesetzten Primärschlüssel abhängt. Die 2NF ist also für alle Tabellen erfüllt.
Damit auch die 3NF (dritte Normalform) erfüllt ist, müssen zusätzlich zur 2NF alle Determinanten Schlüsselkandidaten sein. Für alle Tabellen in denen es keine Abhängigkeiten von Attributen gibt, die nicht Teil des Primärschlüssels sind, ist die 3NF trivialerweise erfüllt.
Im Fall der Tabelle *product* sind viel der im Konzeptionellen Schema enthaltenen Attribut von der abhängig. Da diese kein Primärschlüsselkandidat ist, wurden die Attribute aus dem Physischen Schema entfernt. Die Daten können allerdings durch eine externe API abgerufen werden und müssen deshalb nicht in einer separaten Tabelle vorgehalten werden.
Bei den Tabellen *product_class*, *product_category* und *unit* gäbe es jeweils eine Abhängigkeit anderer Attribute vom Namen, falls dieser eindeutig ist. Da in jedem Fall mehrere Entitäten mit dem gleichen Namen ausdrücklich erlaubt sind, ist dies nicht der Fall.
In der Tabelle *unit_conversion* ist jede Kombination aus zwei Attributen Primärschlüsselkandidat, es kann also keine Determinanten geben, die keine Primärschlüsselkandidaten sind. Die 3NF ist also für alle Tabellen erfüllt und die Normalisierung des Datenbankenmodells ist vollständig.

2.4 Domänen und Standardwerte

Im Letzten Schritt der Model der Datenbank werden die Domänen und weitere Eigenschaften der Attribute festgelegt. In Tabelle 2.2 sind für alle Attribute ausgenommen der Fremdschlüssel die Domänen dargestellt. Außerdem ist angegeben, ob diese den Wert *Null* haben dürfen und ob diese automatisch Inkrementiert werden. Einige der Attribute werden zusätzlich mit einem Standardwert versehen, welcher immer verwendet wird, wenn kein Wert für das Attribut angegeben ist. Für alle Attribut mit Standardwert sind diese in Tabelle 2.3 dargestellt. Da es zwei Attribute mit dem Namen *quantity* gibt sind diese jeweils mit einer Nummer versehen. Bei (1) handelt es sich um das Attribut aus der Tabelle *required_content*, bei (2) um das aus *shopping_list*.

Attribut	Domäne	Nullbar	Auto Inkrement
login	CHARACTER VARYING(255)	✗	✗
password	CHARACTER VARYING(255)	✗	✗
device_id	INTEGER	✗	✓
device_name	CHARACTER VARYING(255)	✗	✗
filled_in	TIMESTAMP WITH TIMEZONE	✗	✗
opened	TIMESTAMP WITH TIMEZONE	✓	✗
dropped_out	TIMESTAMP WITH TIMEZONE	✓	✗
percentage_left	INTEGER	✗	✗
product_id	INTEGER	✗	✓
ean	CHARACTER VARYING(255)	✓	✗
class_id	INTEGER	✗	✓
class_name	CHARACTER VARYING(255)	✗	✗
class_image	BYTEA	✓	✗
storage_life_opened	INTEGER	✓	✗
category_id	INTEGER	✗	✓
category_name	CHARACTER VARYING(255)	✗	✗
category_icon	BYTEA	✓	✗
unit_id	INTEGER	✗	✓
unit_symbol	CHARACTER VARYING(255)	✗	✗
unit_name	CHARACTER VARYING(255)	✓	✗
factor	NUMERIC	✗	✗
quantity(1)	NUMERIC	✗	✗
quantity(2)	NUMERIC	✗	✗

Tabelle 2.2
Domänen der Attribute

Attribut	Standardwert
device_name	Ünbenanntes Gerät”
filled_in	CURRENT_TIMESTAMP
percentage_left	100
factor	1
quantity(1)	1
quantity(2)	1

Tabelle 2.3
Standardwerte der Attribute

3 Entwicklung des Backends

3.1 Entwurf der Architektur

Damit die zu entwickelnde Smartphone-App die Daten aus der Datenbank in aggregierter Form abrufen kann, wird eine zusätzliche Komponente benötigt. Der Backend-Server bildet wie in Abbildung 3.1 dargestellt das Bindeglied zwischen Anwendungsprogrammen wie der Smartphone-App und der Datenbank. Darüber hinaus enthält das Backend die Verarbeitungslogik des Systems und reichert die Daten aus der Datenbank mit Daten aus externen Quellen an. Für die Entwicklung des Backends wird das Node.js-Framework NestJS verwendet.

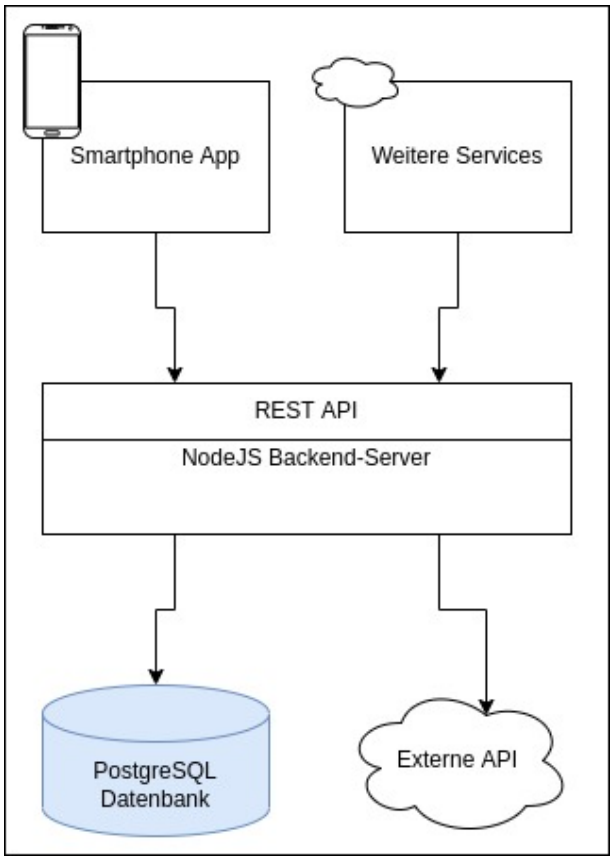

Abbildung 3.1
Architektur des MVP

Der Backend-Server bietet eine Schnittstelle in Form einer REST-API, an die Anwendungen wie die Smartphone-App anknüpfen können. Der Backend-Server ist wie bei NestJS üblich in Module gegliedert die jeweils einer Tabelle im Datenbankmodell zugeordnet sind. Jedes Modul besteht dabei unter anderem aus einem Service, der die der jeweiligen Entitätsklasse zugehörige Verarbeitungslogik ausführt. Einigen Modulen sind zusätzlich mit Controllern ausgestattet. Ein Controller ist einer Route der API zugeordnet und bietet die Schnittstellenmethoden bzw. -operationen für diese an.

3.2 Implementierung des Datenbankmodells

Um die Kommunikation mit der Datenbank zu ermöglichen, hat jedes Modul Zugriff auf den Provider des Datenbank-Moduls. Darüber kann eine Verbindung zur Datenbank aufgebaut werden. Auch das Modell der jeweiligen Entitätsklasse ist zusätzlich jeweils in Form einer Klasse in der *.entity.ts*-Datei des jeweiligen Moduls gespeichert. Die Klassen werden jeweils mit dem *Table*-Dekorator des ORM (Object-Relational Mapper) Sequelize annotiert. Weitere Dekoratoren für die Definition von Domänen werden an den Attributen der Klassen angebracht. Beispielhaft ist die gekürzte Variante der Implementierung der Tabelle *fridge_user* im Codebeispiel 3.1 zu sehen.

```
1  ...
2  @Table({ tableName: 'fridge_user' })
3  export class fridge_user extends Model<fridge_user> {
4    @PrimaryKey
5    @Column
6    login: string;
7
8    @Column
9    password: string;
10
11   @BeforeCreate
12   @BeforeUpdate
13   static async hashPassword(user: fridge_user) {
14     user.password = await hash(user.password, 10);
15   }
16
17   @HasMany(() => fridge_user_device_relation)
18   fridge_user_device_relation: fridge_user_device_relation[];
19
20   @HasMany(() => shopping_list)
21   shopping_list: shopping_list[];
22 }
```

Code 3.1

Sequelize-Entitätsklasse (gekürzte Darstellung)

4 Entwicklung und Simulation von Anwendungsprogrammen

4.1 Entwurf der Smartphone-App

Im Rahmen dieser Arbeit wird bereits ein erster Prototyp der späteren Smartphone-App entwickelt. Für die Entwicklung wird das React-Native-Framework verwendet, da dieses die Möglichkeit bietet mit einer Codebasis sowohl für Android als auch für iOS zu entwickeln. Die Entwicklung der App erfolgt wie auch im Backend mit der Programmiersprache TypeScript.
Da es sich bei der App lediglich um einen Prototyp handelt, wird auf eine vollständige Implementierung aller späteren Funktionen verzichtet. Die primäre Aufgabe der App ist es, die Funktion des Backends zu testen und die Ergebnisse zu präsentieren. In Abbildung 4.1 das aktuelle Layout der App dargestellt. Es wird nur die dargestellte *Fridge Content*-Ansicht verwendet. Aus dem Backend werden die Daten aller Produkte die sich in den Geräten der Benutzerin bzw. des Benutzers befinden geladen und angezigt.

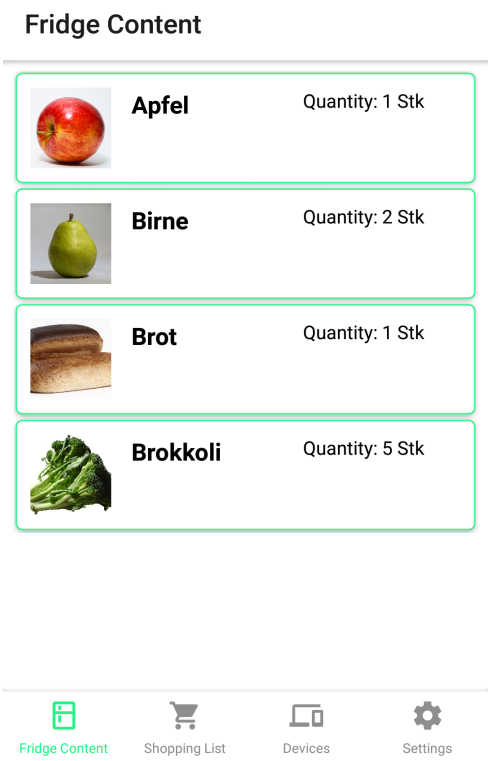

Abbildung 4.1
Layout der Smartphone-App

4.2 Simulation von Drittsystemen

Alle nicht nicht in der App implementierten Funktionen, die für den Betrieb des Prototypen benötigt werden, werden mit Hilfe eines in Python geschriebenen CLI (Command Line Interface) simuliert. Mit diesem ist es Möglich sowohl administrative Aufgaben, wie das Anlegen neuer Produktklassen oder Einheiten zu erledigen, als auch Aktionen die durch Benutzer:innen ausgeführt werden zu simulieren. Alle diese Funktionen werden durch die Verwendung der REST-API des Backend-Servers ausgeführt. In zukünftigen Versionen können die Schnittstellen durch die Smartphone-App oder durch ein Admin-Panel angesprochen werden.
Auch der Betrieb der Geräte wird durch das CLI simuliert. Auch hier wird die Bereits für spätere Anwendungen entwickelte REST-API verwendet. Erkennt ein Gerät ein Produkt anhand eines Barcodes oder anhand eines klassifizierten Bildes wird die Information an den Server gesendet. Während die Übermittlung der EAN bereits originalgetreu implementiert ist, muss die Schnittstelle für die Produkterkennung auf Basis von Bilddaten noch auf die konkrete Lösung angepasst werden.

5 Ergebnisse

Die im Rahmen dieser Arbeit entwickelte Datenbank stellt die Basis für die weitere Arbeit an dem Verwaltungssystem für Lebensmittel. Die Tests mit dem Entwickelten Backend-Server sowie der Smartphone-App und dem CLIhaben gezeigt, dass das Datenbankmodell auf kleiner Skala grundsätzlich für die Verwaltung der Lebensmittel von mehreren Benutzer:innen mit mehreren Geräten geeignet ist. Falls die Architektur des Backends nicht grundsätzlich verändert wird kann die Datenbank mit kleineren Modifikationen weiter verwendet werden.
Auch der Entwickelte Backend-Server ist grundsätzlich für die weiterentwicklung im Rahmen des weiteren Projektverlaufs geeignet. Es ist allerdings zu evaluieren, ob mit der Einführung von weiteren Funktionen die monolithische Struktur des Backends weiter bestehen soll. Es ist denkbar, dass sowohl das Benutzer:innenmanagement als auch das Gerätemanagement durch separate Services übernommen werden. Besonders für die Anbindung echter IoT-Geräte ist Möglicherweise die Verwendung einer IoT-Plattform erforderlich. In diesem Fall bietet es sich an auch die Geräteverwaltung auf dieser Plattform zu implementieren.
Der Entwickelte App-Prototyp kann zwar für Testzwecke weiter verwendet werden, eignet sich allerdings auf Grund der getroffenen Designentscheidungen schlecht als Ausgangspunkt für die Entwicklung der finalen App. Es wurde aufgrund der begrenzten Funktionalität des App-Prototypen auf einen sorgfältigen Entwurf der Architektur verzichtet. Wichtige Komponenten, wie eine Cache-Datenbank oder ein State-Management-System wurden nicht implementiert. Es empfiehlt sich die Entwicklung einer neuen App auf Basis eines geeigneten Templates mit einer ausgereiften Architektur.
Insgesamt wurden bereits wichtige Vorarbeiten für die Entwicklung des Verwaltungssystems für Lebensmittel geleistet. Auch wenn sich nicht alle Komponenten der MVP-Lösung für die Weiterentwicklung eignen, wurden dennoch wichtige Erkenntnisse gewonnen, die beim sorgfältigen Entwurf weiterer Produktiterationen berücksichtigt werden sollten. Besonders an Auswahl von NestJS als Backend-Framework und an der PostgreSQL-Datenbank sollte festgehalten werden.

Abkürzungsverzeichnis

1NF erste Normalform. 9

2NF zweite Normalform. 9–11

3NF dritte Normalform. 11

API Application Programming Interface. 3–5, 11

CLI Command Line Interface. 16, 17

EAN European Article Number. 3, 16

MVP Minimum Viable Product. 2, 3, 13, 17

ORM Object-Relational Mapper. 14