Bachelorarbeit im Fach Informatik

Vorgelegt von:
Jens Hinkelmann
Kaiserslautern, Deutschland
Matrikelnummer 09-923-160

Modellierung und Verwaltung von Nahrungsmitteldaten

Angefertigt am Institut für Informatik der Universität Zürich

Betreuender Professor:
Prof. Dr. Lorenz M. Hilty

Abgabe der Arbeit: 1. Oktober 2016

Zusammenfassung

Diese Bachelorarbeit präsentiert eine verbesserte Methode zur Integration und Verwaltung von Lebensmitteldaten für die Organisation Eaternity AG. Zur Erfüllung der Anforderungen, welche während eines Workshops mit den Mitarbeitenden von Eaternity erhoben wurden, wurde die bisherige Datenstruktur explizit modelliert und in Übereinstimmung mit allgemeinen Kriterien für gute Datenmodellierung ein verbessertes Modell erstellt. Um den Wert und die Funktionalität des neuen Datenmodells zu demonstrieren wurde ein Prototyp für die Datenerfassung und -verwaltung entwickelt. Die für den Prototypen entworfene Architektur ermöglicht die Wiederverwendbarkeit der Hauptkomponenten sowohl in einer Desktop-Applikation als auch in einem Web-Service. Der entstandene Prototyp kann als Vorlage für die Entwicklung eines produktiven Systems verwendet werden und die erhobenen Anforderungen geben Anstoss für weitere Projekte. Die Arbeit beschreibt den Innovationsprozess von der Untersuchung des konkreten Problems über die Abstraktion des Problems und die Definition der Ziele bis hin zur konkreten Umsetzung.

Abstract

This bachelor thesis proposes an improved method for the integration and administration of food related data at the organisation Eaternity AG. To accommodate a set of requirements that have been raised during a workshop with employees at Eaternity, the current data structure was modelled explicitly and in line with a set of key criteria for good data modelling an improved model was crafted. To show the value and functionality of the new data model, a prototype for data capture and administration was built on top of a relational database which implements the model. The prototype implements an architecture which allow for a high reusability of the single components for either a web based or desktop based release. The resulting prototype serves as a template to develop a productive system and the raised requirements introduce ideas for additional projects. The thesis describes the different phases of the innovation process: The observation of the concrete problem, the abstraction of the problem, the definition of goals and the concrete implementation

Danksagung

Ich danke **Prof. Dr. Lorenz M. Hilty** für das entgegenbrachte Vertrauen und die Freiheit, die Aufgabenstellung für diese Arbeit mitzubestimmen.

Ich danke **Aurelian Jaggi**, dem CTO von Eaternity, sowie **Manuel Klarmann**, dem CEO von Eaternity, für ihre Unterstützung und die Einblicke in ihr einzigartiges Unternehmen.

Ich danke **meinem Vater** für seine Unterstützung und seinen Zuspruch.

Ich danke **Holger Wache** fürs Gegenlesen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Eaternity bietet Privat- und Geschäftskunden die Möglichkeit, nachhaltige Entscheidungen bei der Auswahl der täglichen Nahrung zu treffen. Dazu hat das Unternehmen eine Software entwickelt, die auf einer jährlich überprüften CO_2-Datenbank basiert. Sie ist die derzeit grösste und umfangreichste Datenbank zu CO_2-Berechnungen für Menüs[1]. Die Datenbank enthält Informationen über Nahrungsmittel, ihre CO_2-Werte und Nährwerte, Allergene, entstehende Abfälle, sowie Einfluss von verschiedenen Arten der Verarbeitung auf ihren CO_2-Fussabdruck und ihre Nährwerte. Die Inhalte der Datenbank entstammen verschiedenen externen Quellen und werden bei Eaternity von Experten zusammengetragen und aufbereitet. Dazu wird eine Kombination von Online-Services genutzt.

Das Kerngeschäft und Ursprung der Datenbank war die Erfassung von CO_2-Werten. Die anderen Werte wurden nach und nach hinzugefügt. Die zu Grunde liegende Datenstruktur wurde dabei inkrementell erweitert, wo immer möglich ohne Anpassung der bisherigen Struktur. Mittlerweile hat diese eine Komplexität erreicht, für welche die Arbeit mit bisherigen Mitteln und Methoden nicht ausgelegt ist. Um dieses Problem zu lösen, soll in dieser Arbeit im Sinne eines Refactoring das Datenmodell restrukturiert werden und dazu passend ein System vorgeschlagen werden, das die Mitarbeitenden bei der systematischen Extraktion und Transformation der Daten aus externen Quellen unterstützt. Die Umsetzbarkeit des Vorschlags soll anhand eines lauffähigen Prototypen gezeigt werden, welcher das Datenmodell enthält und die wesentlichen Funktionalitäten realisiert.

Um konkret auf die Anforderungen der Mitarbeitenden eingehen zu können, wurde ein problembasiertes und nutzerorientiertes Vorgehen gewählt. Als übergeordnetes Verfahren wurde ein generischer Innovationsprozess [Beckman and Barry, 2007] gewählt und für die Problemanalyse Methoden aus dem Design Sprint von Google [Knapp et al., 2016] verwendet.

[1] www.eaternity.org/app

Kapitel 2

Methodik

Die Vorgehensweise dieser Arbeit orientiert sich am generischen Innovationsprozess von Beckman and Barry [2007], welcher auf die Entwicklung von Artefakten, sowohl Hardware als auch Software, zugeschnitten ist. Der Innovationsprozess unterscheidet zwischen konkreter und abstrakter Welt und verwendet in einer zweiten Dimension Analyse und Synthese um neue Produkte, Services, Geschäftsmodelle oder andere Artefakte zu erzeugen (siehe Abbildung 2.1). Der Prozess erfordert, dass die Beteiligten das konkrete Problem analysieren (Phase *Observation*), von dem konkreten Problem zu abstrahieren, um es besser zu verstehen und neue Sichtweisen in Betracht zu ziehen (Phase *Frameworks*), den Nutzen, Gestaltungsleitsätze und Anforderungen erarbeitet (Phase *Imperatives*) um anschliessend eine oder mehrere Lösungen zu realisieren (Phase *Solutions*). Im Folgenden wird beschrieben, wie diese Arbeit gegliedert und dem generischen Innovationsprozess nach Beckman and Barry [2007] zuzuordnen ist.

2.1 Observations

Die erste Phase, *Observations* (vgl. Abbildung 2.1), beschreibt den Teil eines Innovationsprozesses, während dessen das konkrete, vorliegende Problem erkannt und formuliert wird. Diese Phase bildet den Anstoss für den Innovationsprozess und charakterisiert diesen als problemorientiert.

Es wird in Kapitel 3 zunächst auf die Problemdomäne eingegangen. Es wird die Geschäftstätigkeit von Eaternity und das bisherige Vorgehen zur Datenverwaltung beschrieben. Anschliessend wird beschrieben, wie unter anderem während der Durchführung eines Workshops eine ausführliche Liste von Anforderungen generiert wurde. Diese Anforderungen werden in Abschnitt 3.4 im Detail aufgeführt und erläutert.

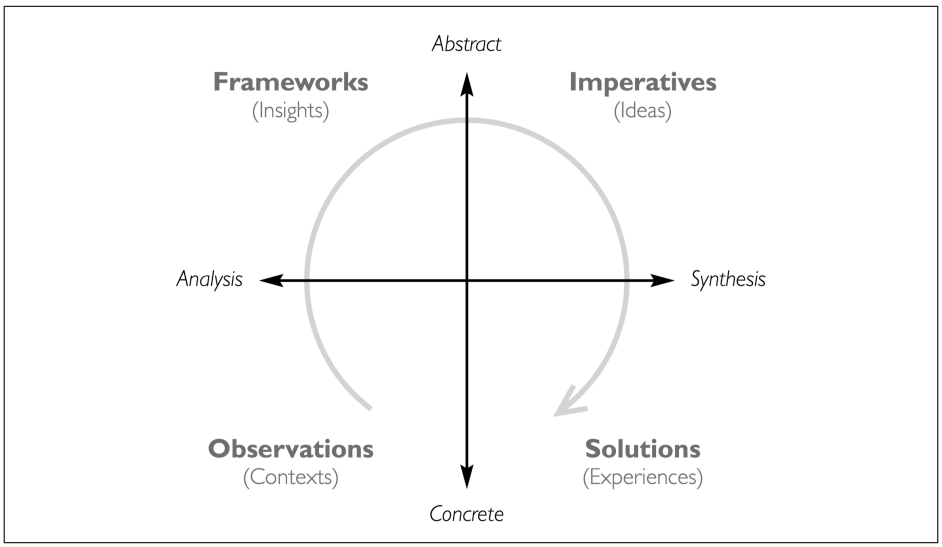


Abbildung 2.1: Generischer Innovationsprozess aus Beckman and Barry [2007]

2.2 Frameworks

Wurde der Anstoss für die Innovation, das konkrete Problem, erkannt, wird das Problem abstrahiert um übergeordnete Muster und Strukturen zu erkennen. Im Kapitel 4 wird das Problem der Datenverwaltung bei Eaternity in einen grösseren Kontext eingebettet und auf Literatur zu Datenintegration und Datenmodellierung näher eingegangen.

2.3 Imperatives

In der Phase *Imperatives* wird festgelegt, welchen Nutzen das zu entwickelnde Produkt erfüllen wird. Wie von Beckman and Barry [2007, S. 41]beschrieben, wird hier anhand von konkreten und abstrakten Erkenntnissen aus den vorangehenden Phasen eine angepasste Liste von (immer noch abstrakten) Anweisungen für das zu entwickelnde Produkt zusammengestellt. Diese finden sich im Kapitel 5 wieder.

2.4 Solutions

In der Phase *Solutions* des generischen Innovationsprozesses sollen verschiedene Lösungsansätze ausgearbeitet, anhand ihrer Eigenschaften evaluiert und schlussendlich konkret umgesetzt werden. Im Folgenden wird auf die verschiedenen Aspekte des Problems, zu welchen Vorschläge erarbeitet wurden, eingegangen.

- *Architektur:* in Abschnitt 6.1 werden verschiedene Vorschläge zur technischen Architektur für die zu entwickelnden Lösung beschrieben und die

ausgearbeitete und realisierte Architektur vorgestellt.

- *Technologien:* In Abschnitt 6.2 werden die in Erwägung gezogenen Frameworks und Toolsets vorgestellt. Es werden dabei die Komponenten der in Abschnitt 6.1 beschriebenen Architektur abgedeckt. Im Anschluss wird erklärt, welche Kombination von Technologien im Prototypen verwendet wird.

- *Datenmodell:* In Abschnitt 6.3 werden anhand des Entity Relationship Modells Defizite an der bisherigen Datenstruktur aufgezeigt, Lösungsvorschläge vorgestellt und die finale Modellierung anhand eines Entity Relationship Modells erklärt.

- *Implementation:* In Abschnitt 6.4 wird beschrieben, wie der Prototyp gemäss der entworfenen Architektur und dem erstellten Datenmodell implementiert wurde.

Kapitel 3

Beobachtung

Dieses Kapitel widmet sich der Beschreibung der konkreten Problemstellung, wie sie im Falle der Datenverwaltung bei Eaternity vorliegt. Im Abschnitt 3.1 werden die Informationen beschrieben, welche Eaternity im zu verbessernden System verwaltet. Im Abschnitt 3.2 wird auf die bisherige Vorgehensweise zur Datenerfassung und -verwaltung sowie die Art, in welcher Daten bisher vorliegen, eingegangen. Im Abschnitt 3.3 wird beschrieben, wie anhand eines Workshops Anforderungen erhoben wurden. Im Abschnitt 3.4 werden die erhobenen Anforderungen im Detail erläutert.

3.1 Die Dienstleistungen von Eaternity

Das Unternehmen Eaternity bietet sowohl für Restaurants als auch für Privatleute CO_2-Statistiken für Lebensmittel und Menüs. Zusätzlich hat Eaternity das Angebot auf Lebensmittelerkärungen für Allergene und Nährwerte erweitert. In Zukunft sollen noch weitere Informationen zu Lebensmitteln bereitgestellt werden, um für Geschäftskunden die Attraktivität des Produkts zu verbessern und die Arbeit in der Küche weiter zu erleichtern. Für Geschäftskunden bietet Eaternity eine RESTful Schnittstelle, um die Dienstleistungen von Eaternity in die eigenen Informationssysteme einzubinden[1].

LCIA und CO_2-Äquivalent: Eaternity bezieht sich bei den CO_2-Fussabdrücken, wie sie an die Endkunden geliefert werden, auf LCIAs (Life Cycle Impact Assessments), also Wirkungsabschätzungen, wie sie von wissenschaftlichen und wirtschaftlichen Institutionen durchgeführt und frei oder kommerziell zur Verfügung gestellt werden. Diese LCIA-Endwerte fassen den gesamten Einfluss eines Produktes während eines definierten Teils des Lebenszyklus auf die Umwelt als Äquivalent von ausgestossenem CO_2 zusammen.

Nährwerte: Eaternity erfasst Nährwertdaten zu Nahrungsmitteln um die Erstellung von "Lebensmittel-Erklärungen, konform mit der EU-Lebensmittel-

[1] Die API wird beschrieben unter http://docs.eaternity.apiary.io/

kennzeichnungs-Verordnung für Allergene und Nährwerte" anbieten zu können. Diese Daten werden meist von nationalen Datenbanken wie der Schweizerischen Nährwertdatenbank des FSVO [2] übernommen und länderspezifisch erfasst.

Fussabdruckänderungen durch Prozesse: Die von Eaternity erfassten CO_2-Fussabdrücke berücksichtigen Prozesse bis zum Verlassen der Produktionsstätte. Spätere Änderungen des CO_2-Fussabdrucks werden von Eaternity bei Kundenanfragen dynamisch berechnet. Dazu müssen unter anderem die Einflussgrössen von Präservations-, Produktions-, Verpackungs-, Verarbeitungs- und Konservierungsprozessen erfasst werden.

Food Waste Wird am Ende des Lebenszyklus nur ein Teil des Produktes tatsächlich verbraucht, muss der CO_2-Fussabdruck des gesamten Produkts auf den verbrauchten Anteil abgewälzt werden. Ausserdem entstehen durch die Entsorgung sowie die Verrottung von Lebensmittelabfällen weitere Emissionen, welche den CO_2-Fussabdruck beeinflussen [Hall et al., 2009]. Die Relevanz von Lebensmittelabfällen in Schweizer Haushalten für die Umwelt wird in [Beretta et al., 2013] untersucht. Seit Mai 2015 erfasst Eaternity Daten zu Nahrungsmittelabfällen. Dabei werden durchschnittliche anfallende Lebensmittelabfälle zu verschiedenen Zeitpunkten im Lebenszyklus eines Produktes pro produzierte Menge erfasst und nach vermeidbarem und nicht vermeidbarem Anteil getrennt.

Nährwertänderungen durch Prozesse: Werden Lebensmittel weiterverarbeitet, verändern sich deren Nährwerte. In Zukunft möchte Eaternity die Berechnung von Nährwerten in Menüs anbieten. Die Erfassung der Nährwertänderungen wird parallel zu dieser Arbeit von Eaternity eingeführt und soll in der Modellierung berücksichtigt werden.

3.2 Bisheriges System

Eaternity arbeitet bereits seit 2009 an der Katalogisierung von CO_2-Fussabdrücken für Menüs. Das System, welches heute für die Verwaltung des Datenbestandes von Eaternity verwendet wird, ist seit Anfang 2015 im Einsatz. Es besteht aus 3 Hauptkomponenten (vgl. Abbildung 3.2):

- Ein privates Git-Repository wird auf Github gehostet und beinhaltet unter anderem JSON-Dateien, welche den gesamten Datensatz von Eaternity umfassen.
- Ein Plugin für den Webseiten-Generator Jekyll, genannt Jekyll-DB, wird verwendet um eine Übersicht über alle Dateien zu generieren. Jekyll-DB läuft dazu auf einer von Eaternity betriebenen Webseite.

[2] naehrwertdatenbank.ch

- Eine modifizierte Version von Prose, einem Content Management System für Github, läuft ebenfalls auf der von Eaternity betriebenen Webseite und wird verwendet, um den Inhalt der Dateien auf Github zu bearbeiten.

3.2.1 Datenbestand

Die JSON-Dateien, welche den Datenbestand von Eaternity ausmachen, unterteilen sich in 4 Typen:

1. *nutrs* (Nutrients, Nährstoffe) beinhalten Datensätze zu den gesamten Inhaltsstoffen eines Produktes.
2. *procs* (Processes, Prozesse) beinhalten Beschreibungen zu einem Prozess und dem Einfluss auf den CO_2-Fussabdruck eines Produktes, sollte der Prozess angewendet werden.
3. *prods* (Products, Produkte) beinhalten Informationen zu einem einzelnen Produkt sowie Verknüpfungen zu Nutrient-Dateien, Process-Dateien und Waste-Dateien, deren Inhalte auf ein Produkt zutreffen.
4. *waste* (Food Waste, Nahrungsmittelabfälle) beinhalten gesamte Datensätze zu den verschiedenen Kategorien von Food Waste, welche Lebensmitteln zugeordnet werden können.

Um eine bessere Übersicht über den Datenbestand zu erhalten, wurden die Daten in einem Entity Relationship Modell (Abbildung 3.1) dargestellt.

3.2.2 Arbeitsweise

Im Folgenden wird gezeigt, welche Schritte notwendig sind, um mit dem bisherigen System Daten in die Datenbank einzuspeisen und zu bearbeiten.

Authentifizierung

Für die Bearbeitung der Dateien muss der Benutzer mehrere Schritte zur Authentifizierung und Authorisierung vornehmen:

- Einmalig wird der Benutzer bzw. die Benutzende dazu aufgefordert, die von Eaternity betriebene Webseite dazu zu autorisieren, die Rechte des eigenen Github-Accounts zur Bearbeitung des privaten Repository zu überprüfen.
- Ist beim Laden der Webseite kein entsprechendes Browser-Cookie vorhanden, muss sich der Benutzer bzw. die Benutzende mit seinem Github-Account authentifizieren (Abbildung 3.3a).
- Startet der Benutzer bzw. die Benutzende erstmalig in einer Session das Content Management System Prose, um eine Datei zu bearbeiten, muss er die Applikation autorisieren, das Github-Repository in dessen Namen zu verändern (Abbildung 3.3b).

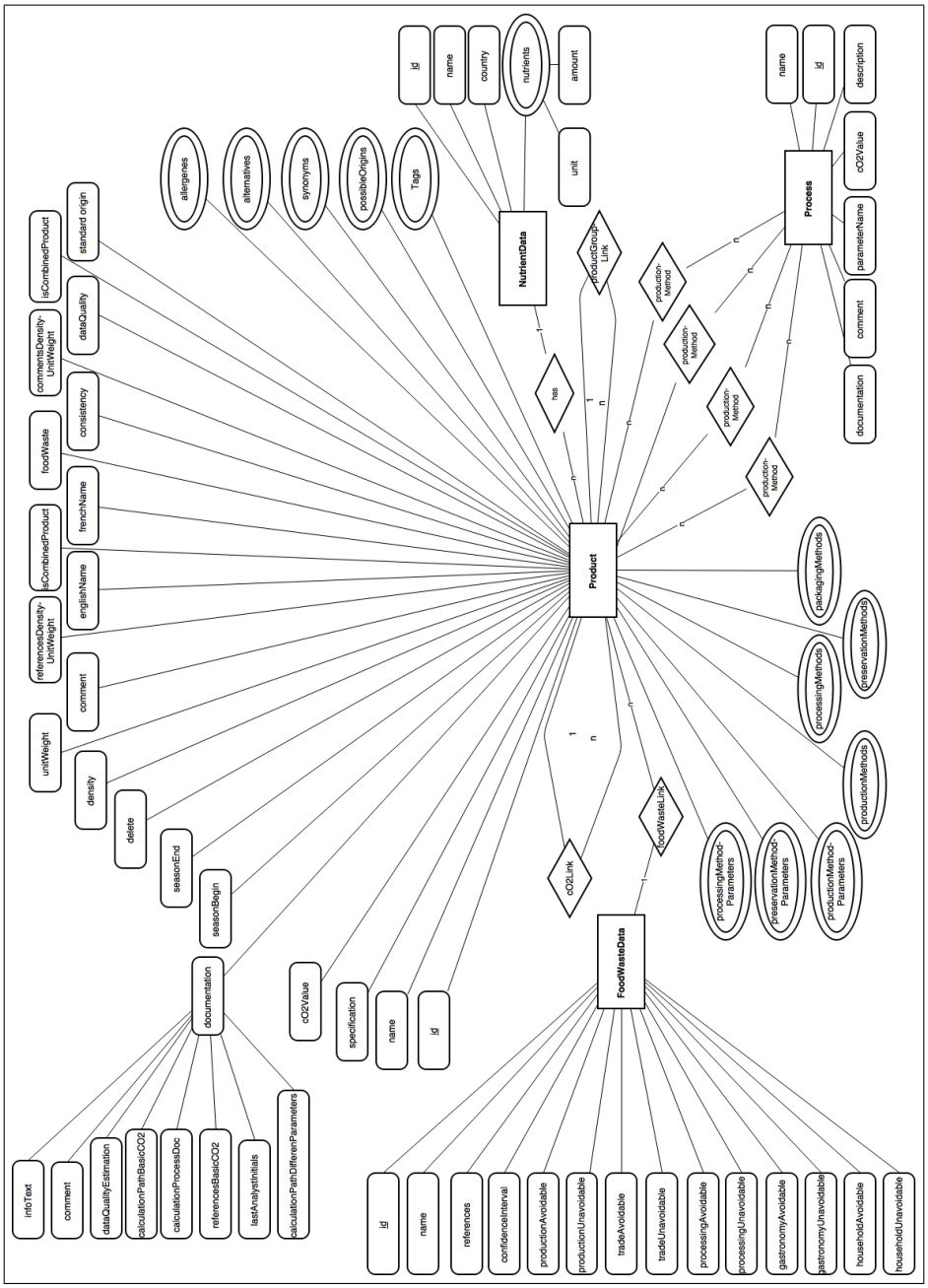


Abbildung 3.1: Das ursprüngliche Datenmodell als ERM

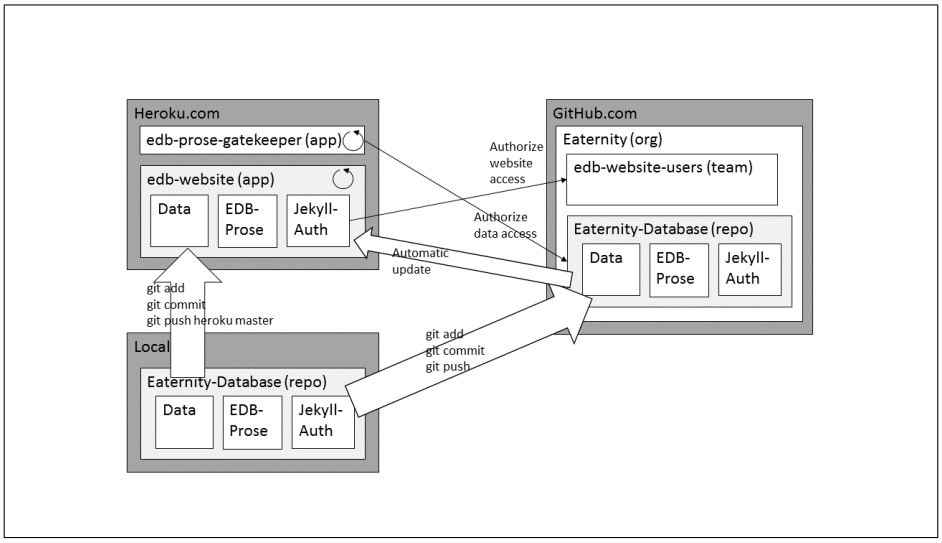
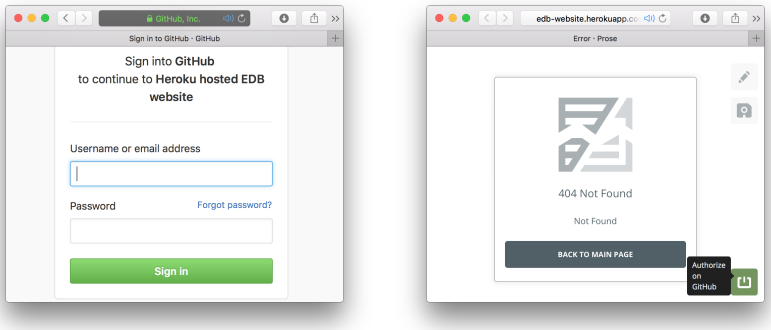
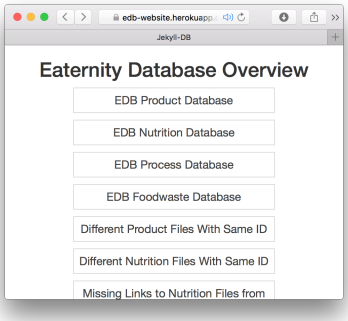


Abbildung 3.2: Bisheriges System. Gemäss [Fkostadinov, 2015].

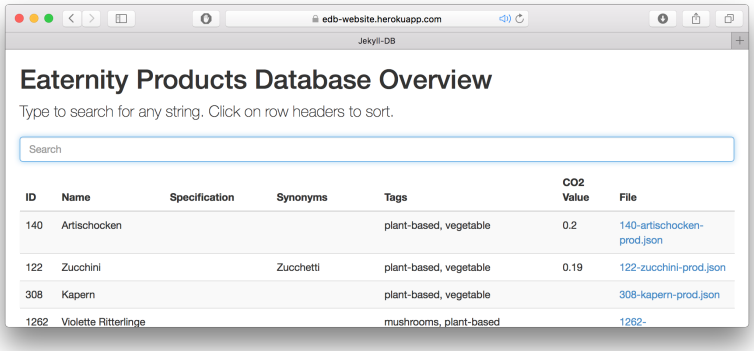


(a) Authentifizierung für Webseite (b) Authentifizierung für Prose

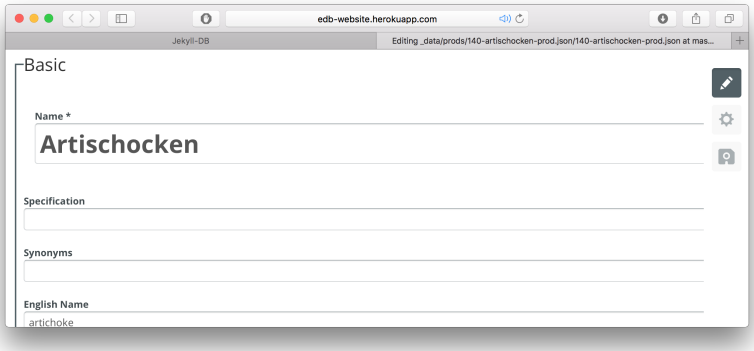
Abbildung 3.3: Zwei Authentifizierungen sind nötig für die Bearbeitung der Daten



(a) Startseite



(b) Produktübersicht mit Jekyll-DB



(c) Produktdatei bearbeiten mit Prose

Abbildung 3.4: Übersicht über die bisherige Benutzeroberfläche.

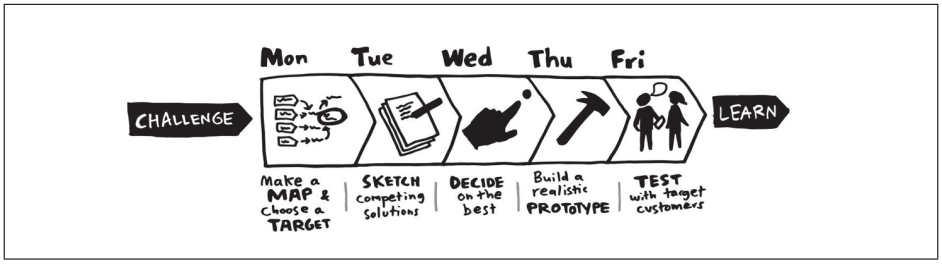


Abbildung 3.5: Der Design Sprint [Knapp et al., 2016]

Ansicht des Datenbestandes

Den Benutzern stehen mehrere Übersichten über den Datenbestand zur Verfügung. Man kann sich die Gesamtheit der Product-, Nutrition-, Process-, oder Foodwaste-Dateien tabellarisch anzeigen lassen. Ausserdem kann man sich Product-Dateien und Nutrition-Dateien anzeigen lassen, welche fälschlicherweise eine mehrfach vergebene ID enthalten (Vgl. Abbildung 3.4a) und nicht verwendete Nutrition-Dateien sowie Produkte ohne Link auf eine Nutrition-Datei.

Datei bearbeiten

Wählt die Benutzerin bzw. der Benutzer in der tabellarischen Übersicht eine Datei zur Bearbeitung aus, wird diese in Prose geöffnet und als Formular angezeigt. Ist die gewünsche Änderung durchgeführt, kann man die Datei von Prose zusammen mit einer Beschreibung der Änderungen ins Github-Repository abspeichern. Wurde die Datei währenddessen von einer anderen Person bearbeitet, meldet Prose einen Konflikt. Die Datei muss neu geladen und die Änderung erneut vorgenommen werden.

3.2.3 Weiterverwendung

Die Applikation für Endkunden hat keinen direkten Zugriff auf die fortlaufend aktualisierte Datenbank sondern verwendet eine eigene Datenbank, welche Eaternity Cloud genannt wird. Die Daten in der Eaternity Cloud werden einmal im Jahr oder bei einschneidenden Änderungen im Datenbestand aktualisiert. Dabei werden sie vom Datenformat für die laufend aktualisierte Datenbank in ein für die Anwendung optimiertes Datenformat konvertiert, welches einen Grossteil der erfassten Daten nicht berücksichtigt. Insbesondere Referenzen werden ausschliesslich zu Zwecken der Nachvollziehbarkeit durch Mitarbeitende erfasst und daher nicht in die Eaternity Cloud abgespeichert.

3.3 Workshop

Während der Problemdefinition sollte auf das Vorwissen und die Erfahrung der Mitarbeiter bei Eaternity, welche mit der Datenbank zu arbeiten haben würden,

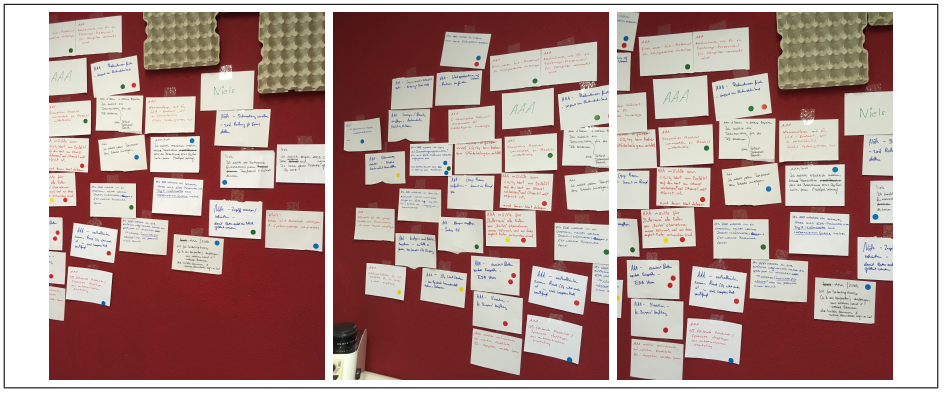


Abbildung 3.6: Am Workshop generierte User Stories

eingegangen werden. Ich bereitete zusammen mit Eaternitys CTO Aurelian Jaggi einen Workshop vor, an welchem sowohl diese Benutzer als auch andere Mitarbeiter von Eaternity teilnahmen. Der Workshop sollte dazu dienen, sich auf ein langfristige Ziel für die Datenbanklösung zu einigen. Der Workshop wurde anhand der Methoden zu Problemverständnis und Problemdefinition in [Direkova, 2015] strukturiert, welche sich am Design Sprint von Google [Knapp et al., 2016] orientiert. Der Design Sprint ist ein auf fünf Tage ausgelegter Innovationsworkshop (siehe Abbildung 3.5), wobei sich der erste Tag mit der Problemanalyse beschäftigt. Somit erlaubt die Methodik des Design Sprints ein effizientes und fokussiertes Vorgehen zum Verständnis des Problems.

3.3.1 Ablauf

Im Folgenden ist der Ablauf des Workshops bei Eaternity beschrieben:

1. *Vision von Eaternity:* Zur Einleitung des Workshops wurden die Teilnehmenden auf einen gemeinsamen Stand bezüglich der unternehmerischen Ziele und Zukunftspläne von Eaternity gebracht.

2. *Existierender Markt:* Um ein Verständnis des Wettbewerbsumfelds zu erhalten, wurden einige Unternehmen vorgestellt, welche wie Eaternity in der Katalogisierung von CO_2-Daten tätig sind. Es wurden auch die Beziehungen zu Datenlieferanten und der wissenschaftlichen Community näher beleuchtet.

3. *Business Opportunity:* Die bekannten Alleinstellungsmerkmale von Eaternity wurden den Teilnehmenden in einem kurzen Vortrag vorgestellt. Danach wurden weitere Geschäftsgelegenheiten für Eaternity diskutiert.

4. *Key Problems:* Die Teilnehmenden erhielten die Gelegenheit, bekannte Schlüsselprobleme zu notieren und gemeinsam mit den anderen Teilnehmenden zu verifizieren.

Abbildung 3.7: Am Workshop identifizierte Schlüsselprobleme

5. *Key-Stakeholder:* Gemeinsam identifizierten die Teilnehmenden die Key Stakeholder von Eaternity.

6. *Generierung von User Stories:* Für wichtigsten Stakeholder generierten die Teilnehmenden User Stories. Die User Stories wurden den übrigen Teilnehmenden im Plenum erklärt und die Validität kurz diskutiert. Im Anschluss bestimmten die Teilnehmer in einem Abstimmungsverfahren, welche User Stories sie als wichtig erachteten.

3.3.2 Resultate

Während des Workshop generierten die Teilnehmenden zwei entscheidende Arten von Artefakten:

- *Schlüsselprobleme:* Bedürfnisse, welche heute bereits bestehen und vom bestehenden System nicht erfüllt werden (Vgl. Abbildung 3.7).

- *User Stories:* Eine Menge von Anforderungen der Key-Stakeholder an das System (Vgl. Abbildung 3.6).

Die während des Workshops erhobenen Schlüsselprobleme und User Stories bildeten die Grundlage für die Ausarbeitung der Anforderungen.

3.4 Anforderungen

Im Folgenden ist eine umfassende und detaillierte Beschreibung der Anforderungen der Mitarbeitenden an das Datenbanksystem ersichtlich. Die Anforderungen entstanden aus den während dem Workshop erhobenen Artefakten sowie Erkenntnissen aus der Analyse der bisher verwendeten Software. Sie wurden mit den Mitarbeitenden von Eaternity abgeglichen.

3.4.1 Grundlegende Datenbankoperationen

Folgende grundlegenden Datenbankoperationen werden von der Datenbanklösung gefordert:

- *Neues Produkt anlegen:* Wird von einem Kunden ein Produkt in der Datenbank vermisst oder entscheidet ein Mitarbeiter, dass ein weiteres Produkt in die Datenbank aufgenommen werden sollte, so muss es möglich sein, zu diesem einen Eintrag in der Datenbank zu hinterlegen.
- *CO_2-Werte für ein Produkt hinterlegen:* Das Kernprodukt von Eaternity sind Daten zum CO_2-Fussabdruck von Nahrungsmitteln. Diese müssen in der Datenbank gespeichert und den korrekten Produkten zugewiesen werden können.
- *Wasserverbrauch für ein Produkt hinterlegen:*Neben dem CO_2-Fussabdruck können auch auch andere Einflüsse der Nahrungsmittelproduktion auf die Umwelt erfasst werden. Einer dieser Einflüsse ist der Wasserverbrauch. Bisher werden bei Eaternity keine Daten zum Wasserverbrauch gesammelt. Es wurde jedoch der Wunsch geäussert, dies in naher Zukunft tun zu können.
- *Energieverbrauch für ein Produkt hinterlegen:* Ähnlich dem Wasserverbrauch werden auch zum Energieverbrauch bisher von Eaternity keine Daten gesammelt. Auch das soll sich in naher Zukunft ändern.
- *Referenzen für CO_2-Werte hinterlegen:* Wird ein CO_2-Wert erfasst, so muss es möglich sein, die Herkunft dieses Wertes zu hinterlegen. Werden Daten von Datenlieferanten verwendet, welche eine Deklaration der Datenherkunft fordern, ist die Erfassung der Datenherkunft notwendig. In erster Linie soll die Referenzierung der Nachvollziehbarkeit innerhalb von Eaternity dienen.
- *Food Waste-Daten für ein Produkt hinterlegen:* Zu Food Waste, also Lebensmittelabfällen, werden bei Eaternity bereits Daten gesammelt. Die Daten machen ersichtlich, welcher Anteil eines Lebensmittels entsorgt wird, anstatt gegessen zu werden. Anteile von Lebensmitteln werden zu unterschiedlichen Zeitpunkten während des Lebenszyklus entsorgt und die gesamten Lebensmittelabfälle werden entsprechend aufgeteilt. Ausserdem werden Abfälle getrennt nach zwingenden unvermeidbaren Anteilen.

- *Allergene für ein Produkt hinterlegen:* Eaternity sammelt Daten zu Allergenen, die sich in den Lebensmitteln befinden. Es muss möglich sein, diese Informationen zu hinterlegen und mit einem Produkt zu verknüpfen.
- *Nährwerte für ein Produkt hinterlegen:* Eaternity sammelt Daten zu Nährstoffen und den Mengen, in welchen sich diese in den einzelnen Lebensmitteln befinden. Diese Informationen müssen hinterlegt und einem Produkt zugeordnet werden können.
- *CO_2-Prozesswerte für ein Produkt hinterlegen:* Werden Lebensmittel weiterverarbeitet ändert sich ihr CO_2-Fussabdruck. Den Wert dieser Änderungen erfasst Eaternity und er muss in der Datenbank hinterlegt werden können.
- *Nährwert-Prozesswerte für ein Produkt hinterlegen:* Ähnlich wie der CO_2-Fussabdruck ändern sich auch die Nährwerte, wenn ein Lebensmittel weiterverarbeitet wird. Auch zu diesen Änderungen sammelt und erfasst Eaternity Daten.
- *Referenzen hinterlegen:* Für die Nährwertänderung, die unterschiedlichen Anteilen der Lebensmittelabfälle pro Produkt bei der Weiterverarbeitung, die Daten bezüglich Allergenen, die Nährwertdaten und die Werte zur Änderung des CO_2-Fussabdruckes muss die Datenherkunft deklariert werden können.
- *Synonyme für ein Produkt hinterlegen:* Produkte müssen anhand verschiedener Namen in der Datenbank gefunden werden können. Dazu müssen die verschiedenen Namen als Synonyme hinterlegt werden.
- *Tags für ein Produkt hinterlegen:* Produkte müssen mit Tags versehen werden können um eine Stichwortsuche zu ermöglichen.
- *FAO-Produktkategorien:* Die Food and Agriculture Organisation (FAO) unterteilt Produkte in Kategorien. Diese Kategorisierung ist weit verbreitet und es sollen in Zukunft die Produktkategorien der FAO mit den Produkten verknüpft werden.
- *Einzelne Werte bearbeiten und löschen:* Alle hinterlegten Werte müssen angesehen, bearbeitet, und gelöscht werden können.

Die meisten grundlegenden Datenbankoperationen werden bereits vom bisherigen System unterstützt. Dies erfolgt jedoch oft nicht auf benutzerfreundliche Art und Weise. Die Vergabe von Tags erfolgt beispielsweise, ohne dass den Benutzenden Vorschläge zu bereits verwendeten Tags gemacht werden. Bei der Erfassung von Referenzen verhält es sich gleich, allerdings ist dies noch ausschlaggebender für Eaternity, da durch unterschiedliche Schreibweisen die Gefahr besteht, dass nicht nachvollzogen werden kann, welche Werte von einer bestimmten externen Quelle stammen.

3.4.2 Erweiterte Datenbankoperationen

Die folgenden Datenbankoperationen, welche das System unterstützen sollen, erfordern komplexere Eingabemethoden:

- *Hinterlegten CO_2-Wert für weiteres Produkt übernehmen:* Ist für ein Lebensmittel keine Quelle vorhanden, welche den CO_2-Fussabdruck liefert, soll zwischenzeitlich der Wert von einem anderen Produkt übernommen werden können.
- *Produkte kategorisieren:* Produkte müssen Kategorien zugewiesen werden können.
- *CO_2-Werte von Kategorie ableiten:* Ist für ein Lebensmittel kein CO_2-Wert hinterlegt und auch keine Verlinkung auf den Wert eines anderen Produktes vorhanden, soll der Wert von einer zugewiesenen Kategorie übernommen werden können.
- *Synonym umwandeln in Produkt:* Es soll möglich sein, einen Begriff, nachdem man ihn als Synonym für den Namen eines erfassten Produktes verwendet hat, stattdesse als Namen für ein neues Produkt zu verwenden.
- *Nach Land verschiedene Nährwerte pro Produkt hinterlegen:* Nährwertdaten können sich von Land zu Land unterscheiden. Dies steht im Zusammenhang mit unterschiedlichen Vorschriften zu Messmethoden. Es soll möglich sein, Nährwertdaten nur für gewisse Länder als gültig zu erklären. Dies ist momentan noch nicht möglich, einem Produkt kann nur ein einziger Nährwertdatensatz hinterlegt werden.
- *Nach Land verschiedene CO_2-Werte pro Produkt hinterlegen:* Es soll auch für CO_2-Fussabdrücke möglich sein, sie nur für gewisse Länder als gültig zu erklären. Dies ist momentan noch nicht möglich, einem Produkt kann nur ein einziger CO_2-Wert hinterlegt werden.
- *Übersetzungen:* Um die Verwendung der Produkte von Eaternity in verschiedenen Ländern zu ermöglichen, soll es möglich sein, Übersetzungen zu den Texten zu hinterlegen, welche für den Endkunden sichtbar sind.

Diese Anforderungen werden vom vom bisherigen System nur begrenzt oder gar nicht unterstützt und eine einfache Erweiterung des Systems ist nicht ohne erhebliche Beeinträchtigung der Nutzbarkeit denkbar. Ohne massgebliche Veränderung der Struktur würde eine Erweiterung zur Erfassung von Werten für verschiedene Länder voraussetzen, dass Attributen für jedes erdenkliche Land angelegt oder andere ähnlich wenig elegante Erweiterungen vorgenommen würden.

3.4.3 Übersicht

Die folgenden Anforderungen bestehen an die Ansicht der Daten und der Übersicht über die Daten:

- *Produktsuche:* Produkte sollen anhand ihrer Schlüsselmerkmale, zum Beispiel Namen, Schlüssel oder Tags, auffindbar sein.
- *Suche erweitert:* Produkte sollen anhand komplexer Anfragen auffindbar sein.
- *Sortieren:* Produkte sollen sortiert angezeigt werden können
- *Visualisierung Datenbestand:* Es soll eine visuelle Übersicht über die erfassten Daten existieren, um einen Überblick über Häufungen und Lücken im Datenbestand zu erhalten.

Dabei sind Suchen und Sortieren von deutlich höherer Priorität als eine erweiterte Suche und die Visualisierung des Datenbestandes, da ohne diese Funktionalität die Arbeit mit einer grossen Menge von Daten kaum vorstellbar ist.

3.4.4 Zugriff und Sicherheit

Folgende Anforderungen in Bezug auf Zugriff und Sicherheit bestehen an das neue System:

- *Kollaboration:* Mehrere Mitarbeiter arbeiten bei Eaternity gleichzeitig am Datenbestand. Die neue Lösung muss das ermöglichen.
- *Alte Daten migrieren:* Die bereits im bestehenden System erfassten Daten müssen mit möglichst geringem manuellem Aufwand in die neue Lösung übertragen werden.
- *Überarbeitungsverlauf:* Es soll ersichtlich sein, welche Änderungen am Datensatz vorgenommen wurden.
- *Änderungen rückgängig machen:* Einzelne Änderungen am Datensatz sollen sichtbar sein.
- *Backup:* Die gesamten Daten müssen gesichert werden können.
- *Export für Eaternity Cloud:* Die Daten müssen aus der neuen Lösung geladen werden können um sie anschliessend in die Eaternity Cloud einzubinden
- *Import von Nährwertdatensätzen:* Nährwertdaten sollen aus den Formaten, wie sie bei grossen Lieferanten verwendet werden, in die Lösung geladen werden können

Bis auf den Import der Datensätzen von Nährwertdatenbanken ist die Erfüllung dieser Anforderungen für den produktiven Einsatz zwingend.

3.4.5 Datenvalidierung

Viele Validierungen können direkt bei der Eingabe vorgenommen werden (Formularvalidierung). Zum Teil ist es jedoch notwendig, unvollständige Daten zu speichern. Dann ist es notwendig, dass nachträglich eine Validierung des Datensatzes vorgenommen werden kann und den Mitarbeitenden mitgeteilt wird, welche Daten unvollständig sind. Die geforderten Validierungen sind:

- *Hinweis bei Verlinkungen:* Es soll auf Produkte aufmerksam gemacht werden, welche einen Wert eines anderen Produktes als Eintrag haben.
- *Hinweis bei fehlenden Werten:* Es soll auf Produkte aufmerksam gemacht werden, für welche wichtige Werte oder Daten nicht vorhanden sind.
- *Minimalstandard für Nährwerte:* Es soll auf Produkte aufmerksam gemacht werden, für welche nicht alle als notwendig eingestuften Nährwerte erfasst wurden.

Die Hinweise können dabei entweder ständig oder nur auf Wunsch des Benutzers bzw. der Benutzerin angezeigt werden. Die Motivation für diese Anforderungen ist, dass die Mitarbeitenden anhand dieser Hinweise den Datensatz gezielt vervollständigen können.

Kapitel 4

Abstraktion

Das zu entwickelnde Informationssystem stellt Daten aus verschiedenen Quellen und Formaten an zentraler Stelle und in einer einheitlichen Form zur Verfügung und bewältigt damit typische Aufgaben der Datenintegration. Die Methoden zur Datenintegration und deren Vor- und Nachteile werden im Abschnitt 4.1 erläutert.

Nach dem Referenzmodell von Kampffmeyer [2007] besteht ein Informationssystem aus 5 Komponenten (siehe Abbildung 4.1).

- Das Capturing beinhaltet das Erfassen und Bereitstellen von Informationen und kann verschiedenste Aufgaben umfassen vom Scannen und Abtippen von Papierdokumenten bis zur automatisierten Texterkennung.
- Das *Managment* umfasst beispielsweise die Verwaltung von Inhalten oder die Organisation von Kollaboration.
- Die Komponente *Deliver* umfasst alle Arten der Präsentation der Daten von der Einspeisung in andere Systeme bis zur Darstellung für Benutzer.
- *Store* beinhaltet Funktionen zur Speicherung von Daten. Dazu gehören Datenbanken, Dateisysteme, Versionierung und zugrunde liegende Technologien, wie Speichermedien.
- *Preserve* beinhaltet Funktionen der Archivierung. Die Archivierung, also langfristige und unveränderbare Speicherung von Daten, ist vor allem für geschäftsrelevante Dokumente erforderlich [Kampffmeyer and Rogalla, 1997].

Datenintegration und Datenmodellierung, welche die Basis des zu entwickelnden Systems bilden, sind Funktionen der Manage-Komponente. Die Capture-Komponente findet sich in der Erfassung der Daten wieder. Das zu entwickelnde System stellt dazu eine Benutzeroberfläche zur Verfügung. Ein Teil der Benutzeroberfläche, welcher das Durchsuchen und Anzeigen der erfassten Daten

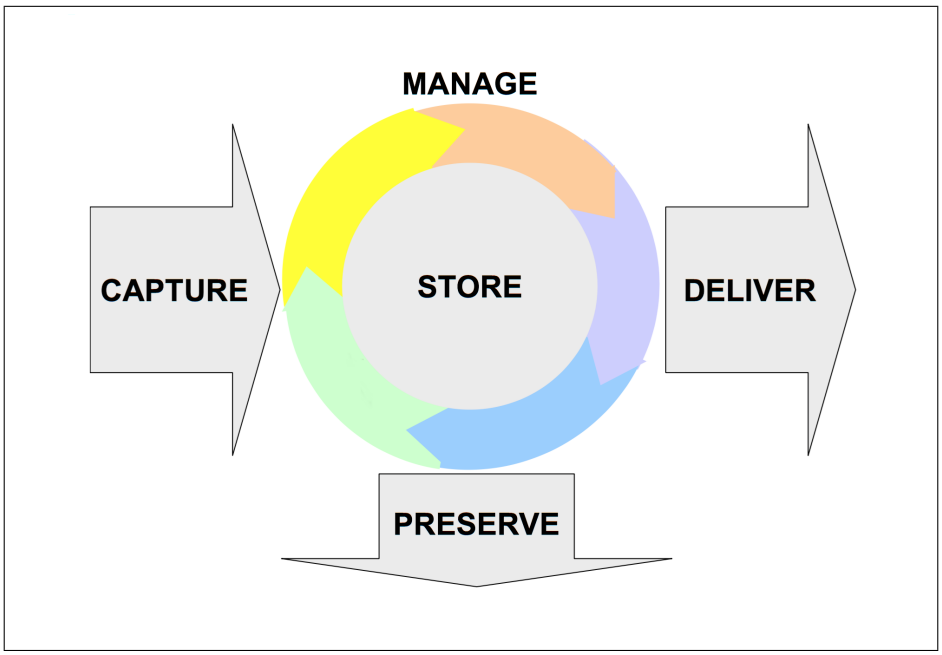


Abbildung 4.1: Informationssystem nach Kampffmeyer [2007]

ermöglicht, sowie der Datenexport für Eaternity Cloud, gehören zur Deliver-Komponente. Die Speicherung der erfassten Daten in Form einer Datenbank sowie die Versionierung und Absicherung der Daten stellt die Store-Komponente des zu entwickelnden Systems dar.

4.1 Datenintegration

In der Literatur wird Integration von Information als Erzeugung einer redundanzarmen Repräsentation von Information, die sich zunächst in einer Menge sich überlappender Systeme befinden, verstanden (vergleiche [Leser and Naumann, 2007]).

Integration kann anhand des Ortes, an welchem die integrierten Daten gespeichert werden, klassifiziert werden. So unterscheidet man zwischen virtueller und materialisierter Integration(Leser and Naumann [2007], Kapitel 4.1). Von virtueller Integration wird gesprochen, wenn die zu integrierenden Daten in den ursprünglichen Systemen belassen und erst zum Zeitpunkt der Verwendung ins integrierte System geladen und dem Benutzer bereitgestellt, danach jedoch wieder gelöscht werden. Im Gegensatz dazu werden bei der materialisierten Integration die Daten in das System selbst kopiert, also an zentraler Stelle gespeichert, beispielsweise mittels einer Datenbank. Virtuelle Integration bietet gegenüber der materialisierten Integration den Vorteil, dass die Aktualität der Daten ständig gewährleistet ist, während bei der materialisierten Integration

	Materialisiert	Virtuell
Aktualität	niedrig: Updatefrequenz	hoch: immer aktuell
Antwortzeit	niedrig: lokale Bearbeitung	hoch: Netzwerkverkehr
Komplexität	niedrig: wie DBMS	hoch: Anfrageplanung
Autonomie	Beantwortung von Anfragen	Bereitstellung von Load-Dateien
Anfragemöglichkeiten	unbeschränkt: volles SQL	beschränkt (bei beschränkten Quellen)
Read/Write	beides möglich	nur lesend
Speicherbedarf	hoch: gesamter Datenbestand	niedrig: nur Metadaten
Belastung der Quellen	hoch, aber planbar	eher niedrig, schlecht planbar
Datenreinigung	möglich	nicht möglich

Tabelle 4.1: Vergleich materieller und virtueller Integration [Leser and Naumann, 2007]

die Aktualität davon abhängig ist, wieviel Zeit seit der letzten Aktualisierung der zentralen Datenbank vergangen ist. Zudem ist der Speicheraufwand bei virtueller Integration gering. Virtuelle Datenintegration setzt jedoch im Gegensatz zur materialisierten die ständige Verfügbarkeit der zu integrierenden Systeme voraus. Handelt es sich bei diesen um vollständig unabhängige Systeme ist es unter Umständen nicht möglich, Änderungen an den Daten vorzunehmen.

Die Daten, die für diese Arbeit relevant sind, weisen Eigenschaften auf, welche sie für die virtuelle Integration ungeeignet machen:

- Die Daten sind teilweise nicht in ständig verfügbarer Form abgelegt, in Form von Volltext oder in externen Datenbanken, welche für eine programmatische Abfrage nicht ausgelegt sind.
- Die Experten bei Eaternity müssen die Daten bei der Erfassung modifizieren können. Die Bearbeitung der Daten direkt an der Quelle ist jedoch nicht möglich.

Eine prominente Architektur, die auf materialisierter Integration aufbaut, ist die von Data Warehouses. Data Warehouses sind Systeme, die Daten aus verschiedenen Quellen sammeln und in einem Basisschema vereinen. Die Integration wird hier in dem ETL-Prozess realisiert, der dem Data Capture entspricht (siehe Abbildung 4.1). Der ETL-Prozess besteht aus Extraktion, Transformation und Laden [Leser and Naumann, 2007, S. 383]. Diese finden sich, wie im Folgenden gezeigt auch für im Falle des für Eaternity zu entwickelnden Systems wieder.

- Während der Extraktion werden die Daten aus den Quellen entnommen. Die Extraktion erfolgt, wenn Eaternity-Mitarbeitende auf neue Informationen zu Lebensmitteln stossen und in das neue System einspeisen. Dabei

handelt es sich um manuelle im Vergleich zu automatisierter Extraktion. Dies ist notwendig, da die Quellen, welche die Daten bereitstellen einerseits nicht im Voraus bekannt sind also nicht direkt an das System angebunden werden können und andererseits oft in unstrukturierter Form vorliegen.

- Die Transformation umfasst die Schritte zur Anpassung an das zentrale Datenmodell. Diese kann ausserhalb oder innerhalb des eigentlichen System stattfinden. Sie erfolgt im Anwendungsfall mit der manuelle Eingabe sowie der Verarbeitung der eingegebenen Daten und der Eingliederung der verarbeiteten Daten in das interne Datenmodell des Systems.

- Das Laden ist der eigentliche Import der angepassten Daten in das Basisschema des Systems. Der Ladevorgang erfolgt im bisherigen System durch Abspeicherung in JSON-Dateien im Git-Repository. Im neuen System wird hier eine Datenbank verwendet (siehe Abschnitt 6.4.1).

Data Warehouses dienen der betriebsinternen Entscheidungsunterstützung. In diesem Punkt unterscheiden sie sich vom zu entwickelnden System.

4.2 Datenmodellierung

Datenmodellierung kann als einer der kritischsten Schritte für die Entwicklung eines Informationssystems betrachtet werden. Hauptgrund für die Wichtigkeit der Datenmodellierung ist der Einfluss des Datenmodell auf die restlichen Komponenten. Ein gutes Datenmodell kann das Programmieren deutlich vereinfachen und kleine Änderungen am Datenmodell können den gesamten Programmieraufwand deutlich reduzieren oder vergrössern. Ein Datenmodell hat Einfluss auf die Datenqualität. Zum einen, weil es die falsche Erfassung von Daten verhindern kann (über Konsistenzregeln), zum anderen, weil durch klare Bezeichnungen der Elemente des Datenmodells die Chance verringert wird, dass Daten falsch erfasst werden[Simsion and Witt, 2004].

Für ein Datenmodell sind folgende, teilweise im Widerspruch zueinander stehende Eigenschaften als vorteilhaft zu betrachten(vgl. [Simsion and Witt, 2004, Abschnitt 1.6]):

- Das Datenmodell soll vollständig sein. d. h. alle notwendigen Daten sollen im Modell erfasst werden.

- Ein gutes Datenmodell sollte möglichst redundanzfrei sein, also wo möglich verhindern, dass die gleiche Information durch mehrere unterschiedliche Elemente des Datenmodells repräsentiert wird.

- Das Datenmodell sollte Geschäftsregeln durchsetzen. D. h. Regeln, welche für den modellierten Teil der Realität gelten, sollten auch im Modell wiederzufinden sein.

- Das Datenmodell sollte die Wiederverwendbarkeit der Daten ausserhalb der geplanten Geschäftstätigkeit ermöglichen.

- Um zukünftige Erweiterungen zu ermöglichen sollte das Datenmodell flexibel oder stabil sein. Flexibel ist ein Datenmodell, wenn es auf die zukünftige Erweiterung im Falle sich ändernder Anforderungen vorbereitet ist. Stabil ist es, wenn selbst bei sich ändernden Anforderungen keine Erweiterungen des Datenmodell benötigt werden.
- Es sollte Eleganz im Gegensatz zu Komplexität angestrebt werden. Um die langfristigen Kosten in der Benutzung eines Datenmodells zu reduzieren, sollte eine elegante Darstellung der Daten gewählt werden.
- Das Datenmodell sollte so entworfen werden, dass Benutzer die modellierten Konzepte verstehen. Dadurch wird die Kommunikation über modellierte Daten zwischen verschiedenen Stakeholdern vereinfacht.
- Das Datenmodell sollte in seine Umwelt integrierbar sein. Kompatibilität mit anderen Datenmodellen ist für die schlussendliche Verwendung des Datenmodells entscheidend.

Alle diese Eigenschaften sind für die Datenmodellierung anzustreben. Die wesentliche Herausforderung für das zu entwickelnde Datenmodell bei Eaternity ist die Gewährleistung von Flexibilität und Stabilität. Ständige Erweiterung der Geschäftstätigkeit bringen ständig neue Anforderungen an das Datenmodell und es ist von Vorteil, wenn diese Anforderungen keine Änderungen am Datenmodell erfordern oder diese Änderungen leicht durchzuführen sind und keine grossen Einflüsse auf den bestehenden Teil des Datenmodells haben.

Kapitel 5

Problemdefinition

Im Rahmen der Abstraktion wurden mit Hilfe der Theorie von Informationssystemen die Schwerpunkte der Entwicklung festgelegt (Kapitel 4). Im nächsten Schritt wurden die Anforderungen aus Abschnitt 3.4 priorisiert und die relevanten Anforderungen ausgewählt. Abschnitt 5.1 zeigt, welche Anforderungen die Datenmodellierung betreffen und welche Ziele für die Restrukturierung des Datenmodells gesetzt werden. In Abschnitt 5.2 wird festgelegt, welche Anforderung der Prototyp erfüllen muss, um die Realisierbarkeit der wesentlichen Funktionalitäten für die Ablösung des bisherigen Systems zu demonstrieren.

5.1 Ziele Datenmodell

Das Datenmodell soll alle Daten, welche jetzt oder in naher Zukunft von Eaternity erfasst und verwaltet werden, beinhalten und die erforderten Datenbankoperationen unterstützen. Wie in Kapitel 4 hergeleitet, sind dabei die Gewährleistung von Flexibilität und Stabilität wesentlich für das zu entwickelnde Datenmodell. Dazu soll das Datenmodell so gestaltet sein, dass Erweiterungen der Geschäftstätigkeit keine Änderungen am Datenmodell erfordern oder diese zumindest keine grossen Einflüsse auf den bestehenden Teil des Datenmodells haben. Zudem soll das bisherige Modell auf Schwächen in Bezug auf die in Abschnitt 4.2 beschriebenen Eigenschaften untersucht werden und diese wo möglich für das neue Modell beseitigt werden. Daraus ergeben sich für das Datenmodell die in Tabelle 5.1 aufgelisteten Anforderungen. Die in der letzten Spalte mit *Nein* gekennzeichneten Anforderungen betreffen zukünftige Erweiterungen, welche zum jetzigen Zeitpunkt noch nicht Teil der Tätigkeit von Eaternity sind und in der Modellierung zwar antizipiert aber nicht umgesetzt werden sollen.

Nr.	**Anforderung**	**Datenmodell**
1	Neues Produkt anlegen	Ja
2	CO_2-Werte für ein Produkt hinterlegen	Ja

Nr.	Anforderung	Datenmodell
3	Wasserverbrauch für ein Produkt hinterlegen	Nein
4	Energieverbrauch für ein Produkt hinterlegen	Nein
5	Referenzen für CO_2-Werte hinterlegen	Ja
6	Food Waste-Daten für ein Produkt hinterlegen	Ja
7	Referenzen für Food Waste-Daten hinterlegen	Ja
8	Allergene für ein Produkt hinterlegen	Ja
9	Referenzen für Allergene hinterlegen	Ja
10	Nährwerte für ein Produkt hinterlegen	Ja
11	Referenzen für Nährwerte hinterlegen	Ja
12	CO_2-Prozesswerte für ein Produkt hinterlegen	Ja
13	Referenzen für CO_2-Prozesswerte	Ja
14	Nährwert-Prozesswerte für ein Produkt hinterlegen	Ja
15	Referenzen für Nährwert-Prozesswerte hinterlegen	Ja
16	Synonyme für ein Produkt hinterlegen	Ja
17	Tags für ein Produkt hinterlegen	Ja
18	FAO-Produktkategorien	Nein
19	Einzelne Werte bearbeiten und löschen	Ja
20	CO_2-Wert für weiteres Produkt übernehmen	Ja
21	Produkte kategorisieren	Ja
22	Nährwert-Prozesswerte von Kategorie ableiten	Ja
23	Synonym umwandeln in Produkt	Ja
24	Nach Land verschiedene Nährwerte pro Produkt	Ja
25	Nach Land verschiedene CO_2-Werte pro Produkt	Ja
26	Übersetzungen	Nein

Tabelle 5.1: Für Projekt bereinigte Anforderungsliste

5.2 Ziele Prototyp

Im Prototypen sollen die in Kapitel 4 beschriebenen Funktionen eines Informationssystems vertreten sein. Konkret wurden folgende Kombinationen von Anforderungen als passend bestimmt, um die einzelnen Funktionen zu vertreten:

- *Capture:* Es soll möglich sein, neue Produkte zu erfassen. Den Produkten sollen durch die Benutzenden sowohl neue CO_2-Werte als auch neue

Nährwerte hinzugefügt werden können. Den Werten soll eine Referenz hinterlegt werden können. Es sollen nach Land unterschiedliche Nährwerte pro Produkt hinterlegt werden können. Es soll ausserdem möglich sein, diese Daten im Format, wie sie bisher vorlagen, in den Prototypen einzuspeisen.

- *Manage:* Die gespeicherten Werte sollen durchsucht, bearbeitet und gelöscht werden können. Es soll ersichtlich sein, wenn Werte auf andere Werte verlinkt sind.
- *Store:* Die Daten sollen für die gleichzeitige Verwendung und Bearbeitung durch mehrere Personen gespeichert werden und es soll eine Form der Versionierung möglich sein.
- *Deliver:* Die Daten sollen aus dem System exportiert werden können, sodass sie in ein anderes System wie beispielsweise Eaternity Cloud geladen werden können.

Mit diesen Anforderungen kann die bisherige Kernaufgabe von Eaternity bereits abgedeckt werden und anhand deren Erfüllung ist ersichtlich, dass die Datenerfassung, Datenverwaltung und Datenbereitstellung unter Verwendung der entwickelten Architektur, der gewählten Technologien und des restrukturierten Datenmodells umgesetzt werden kann.

Kapitel 6

Umsetzung

In diesem Kapitel wird auf die Anpassung des Datenmodells und die Implementation des Prototypen eingegangen. Zunächst werden im Abschnitt 6.1 Möglichkeiten zur technischen Architektur vorgestellt und die ausgearbeitete Architektur des Prototypen erklärt. Im Abschnittt 6.2 werden zu den Komponenten, aus welchen sich die verschiedenen Architekturvorschläge zusammensetzen, Technologien vorgestellt, welche für deren Umsetzung in Erwägung gezogen wurden. Die Restrukturierung des Datenmodells wird im Abschnitt 6.3 beschrieben. Die Implementation des Prototypen, welcher die Tauglichkeit des Datenmodells demonstriert, wird im Abschnitt 6.4 genauer beschrieben.

6.1 Technische Architektur

Die Architektur eines Programms oder Informationssystems setzt sich zusammen aus den einzelnen Softwareelementen, deren Zusammenspiel und deren äusserlich sichtbaren Eigenschaften [Bass et al., 2003]. In Abschnitt 6.1.1 wird auf die Vor- und Nachteile bei der Wahl einer zentralisierten Architektur eingegangen. In Abschnitt 6.1.2 wird diskutiert, welche Vor- und Nachteile eine Desktop-Applikation mit sich bringt und in Abschnitt 6.1.3 wird die Kompromisslösung vorgestellt, welche für den Prototypen ausgearbeitet wurde.

6.1.1 Zentraler Web-Service

Die Entwicklung eines zentralen Web-Services (siehe Abbildung 6.1) stellt den konservativsten der Vorschläge dar, da sie der bisherigen Lösung sehr ähnlich ist. Ein zentraler Web-Service übernimmt hier sowohl die Verarbeitung der Benutzereingaben, als auch die Abspeicherung, Versionierung und Bereitstellung der Daten. Die Bereitstellung der Daten erfolgt in Form einer Webseite und einer Web-API, sodass Eaternity Cloud und zukünftig auch andere Anwendungen auf die Daten von Eaternity zugreifen können. Für die Entwicklung eines Web-Services sprechen vor allem folgende Vorteile:

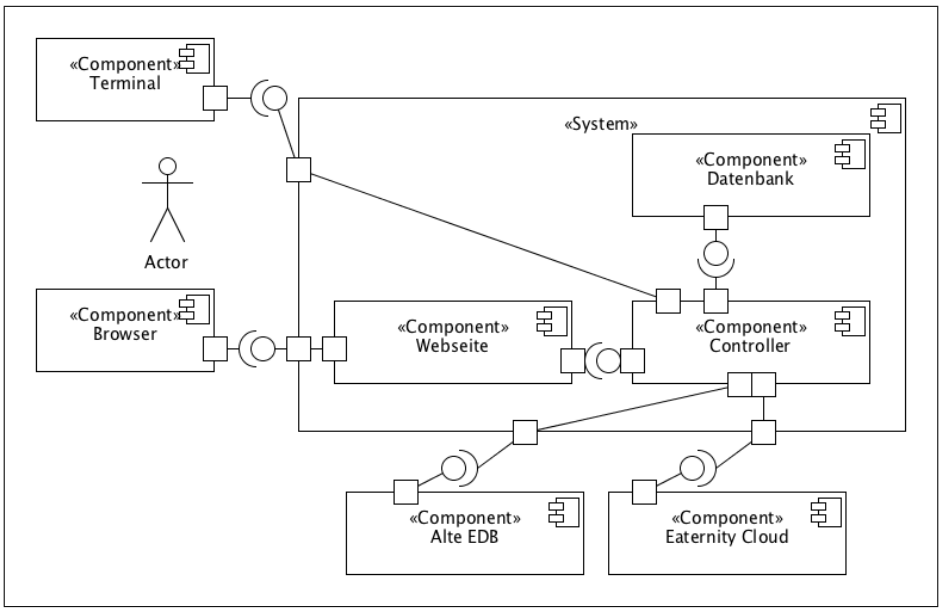


Abbildung 6.1: Architektur mit zentralem Server

- Die gesamte Anwendungslogik kann in einer Komponente realisiert werden. Dies erspart die Entwicklung von zusätzlichen internen Schnittstellen.
- Viele bewährte Frameworks, beispielsweise Django[1] für Python und Symfony[2] für PHP sind auf die Entwicklung aller Komponenten in dieser Architektur ausgelegt. Somit verkürzt sich die Zeit zur Inbetriebnahme.
- Die Bereitstellung einer Benutzerschnittstelle über den Browser vermeidet Kompatibilitätsprobleme bei der Verwendung verschiedener Betriebssysteme.
- Da nur eine zentrale Version des Datensatzes existiert, erübrigt sich die Herausforderung der Bereinigung von Konflikten aufgrund widersprüchlicher Veränderungen am Datensatz.

Es sprechen jedoch, wie am bisherigen System erkenntlich, auch einige Punkte gegen die Verwendung eines zentralen Web-Servers:

- Die Nutzung der Applikation ist nur mit einer Verbindung zum zentralen Web-Service möglich.
- Die Kommunikation muss gesichert werden. Eine Lösung, welche über das Internet kommuniziert, bedingt die Entwicklung einer sicheren Schnittstelle.

[1] https://www.djangoproject.com/
[2] http://symfony.com/

- Das zeitgleiche Bearbeiten von Dateien stellt eine Herausforderung dar. Im aktuellen System ist es beispielsweise nicht möglich Änderungen an einer Datei abzuspeichern, wenn seit dem Öffnen der Datei ein anderer Benutzer bzw. eine andere Benutzerin diese Datei verändert hat.
- Es muss von der Lösung selbst eine Möglichkeit zur Authentifizierung bereitgestellt werden, um nicht autorisierte Zugriffe und Änderungen zu verhindern.
- Der Unterhalt eines eigenen Web-Services bringt Wartungsaufwand mit sich.

Mögliche Varianten dieser Architektur umfassen:

- Die Speicherung der Daten kann wie bisher mit Github oder einem anderen Hosting-Provider erfolgen. Das bisherige System verwendet diese Lösung, um das Content Management System Prose für die Bearbeitung der Daten verwenden zu können.
- Der Zugriff auf den zentralen Web-Service kann anstatt über eine Webseite über eine Desktop-Applikation erfolgen.

Durch die Einbindung eines Hosting-Providers wird auch die Aufgabe der Datensicherung auf den Hosting-Provider abgewälzt. Wird eine Desktop-Applikation an den Web-Service angebunden, kann der Vorteil, die gesamte Logik in einer einzigen Komponente entwickeln zu können, zugunsten einer besseren Ausnutzung der Rechenleistung auf dem Benutzersystem aufgegeben werden.

6.1.2 Desktop-Applikation

Anstatt die Anwendungslogik ganz oder teilweise auf einem zentralen Server zu betreiben, kann diese vollständig auf den Systemen der Benutzer ablaufen (siehe Abbildung 6.2), sofern die Versionierung und Speicherung der Daten mit einem Hosting-Provider implementiert wird. Wird in der Desktop-Applikation ein permanenter lokaler Datenspeicher eingebunden, bietet sich zusätzlich die Möglichkeit, die Applikation ohne Anbindung an das Internet zu nutzen. Der wesentliche Nachteil im Vergleich zum zentralen Web-Service ist, dass auf die Kompatibilität mit unterschiedlichen Betriebssystem Rücksicht genommen werden muss. Zusätzlich ist eine Installation notwendig, was die Benutzerinnen und Benutzer davon abhält, die Applikation auf mehreren Geräten gleichzeitig zu verwenden.

6.1.3 Realisierte Lösung

Zum Zeitpunkt dieser Arbeit konnte keine abschliessende Entscheidung zwischen Web-Service und Desktop-Applikation getroffen werden. Um die Entscheidung nicht abschliessend zu treffen und für beide Entscheidungen einen Beitrag zu leisten, wurde eine Kompromisslösung ausgearbeitet, die für beide Fälle eine hohe Wiederverwendbarkeit der Komponenten aufweist (vgl. Abbildung 6.3):

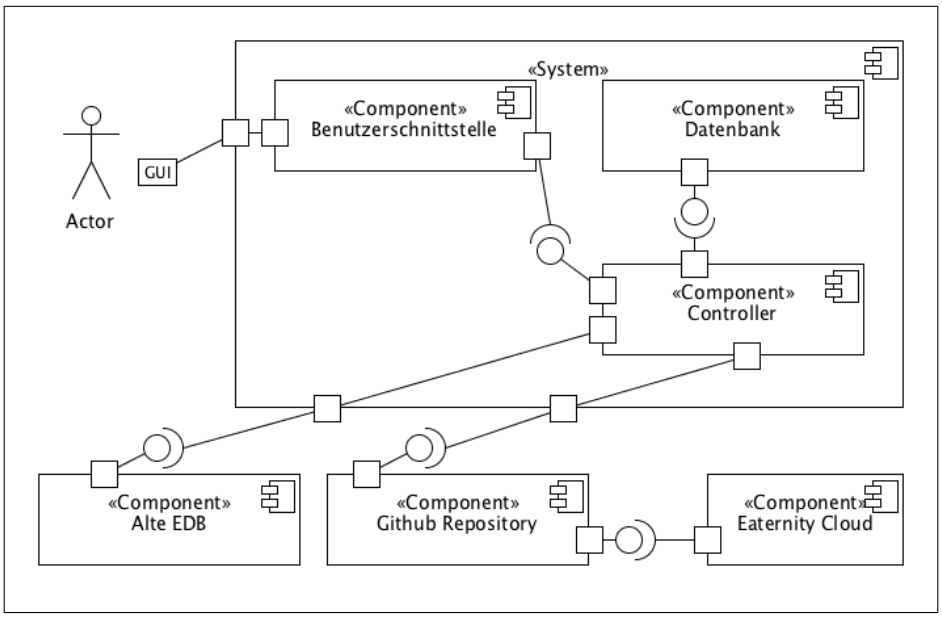


Abbildung 6.2: Desktop-Applikation mit lokaler Datenbank

- Es wird die gesamte Logik auf dem System des Benutzers ausgeführt, es erfolgt jedoch eine Trennung in Frontend und Backend.
- Frontend und Backend kommunizieren über Hyper Text Transfer Protocol (HTTP).
- Das Frontend verwendet Web-Technologien.
- Die Daten werden direkt von der Frontend-Komponente oder vom Benutzer selbst auf Github geladen und dort versioniert und zentral gespeichert.

Soll also in Zukunft die Applikation in einen Web-Service umgebaut werden, kann die Backend-Komponente auf einem Server weiterverwendet werden. Beim lokalen Betrieb erspart man sich die Sicherung der Schnittstelle und es ist nicht notwendig, eine Benutzerverwaltung zu entwickeln, da diese über die Github-Konten der Mitarbeiter übernommen wird. Da die Applikation die Daten im lokalen Backend zwischenspeichert, ist keine ständige Verbindung zum Internet notwendig. Durch gleichzeitige Bearbeitung der gleichen Datei entstandene Konflikte können über bestehende Git-Konfliktlösungs-Tools nachträglich beseitigt werden.

6.2 Technologien

Die Wahl der verwendeten Technologien erfolgte anhand der entworfenen Architektur. Für die einzelnen Komponenten wurden mögliche Frameworks zur

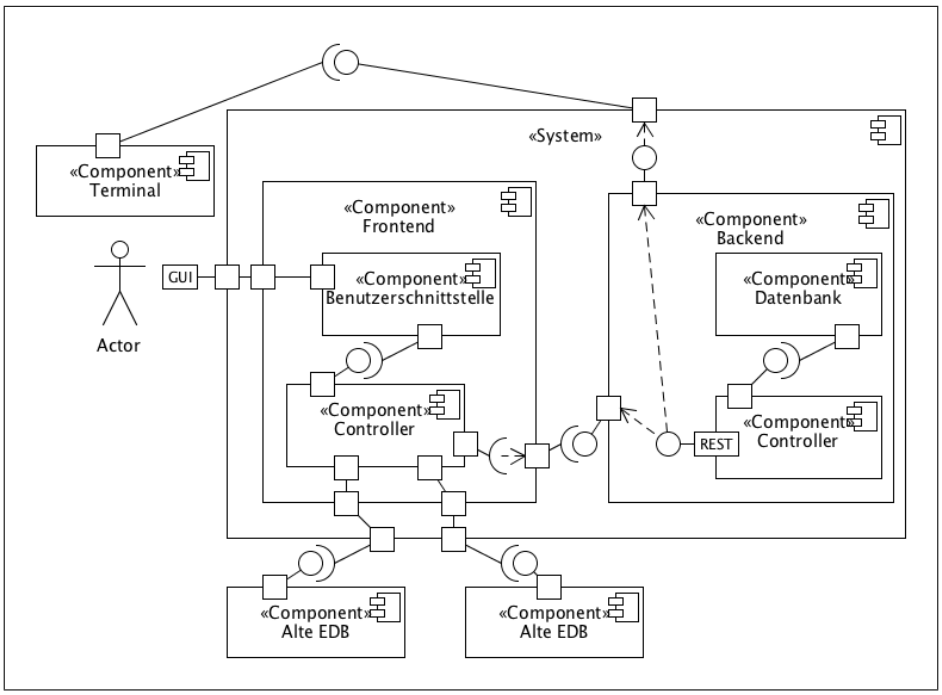


Abbildung 6.3: Lokale Web-Applikation

Implementation untersucht und deren Eignung für die Entwicklung eines Prototypen sowie die zukünftige Verwendung für Eaternity beurteilt.

6.2.1 Frontend

Für die Wahl der Technologien für das Frontend war entscheidend, dass es möglich sein sollte, auch dann noch Logik auf dem System des Benutzers ausführen zu können, falls die Backend-Komponente in Zukunft auf einem Web-Server betrieben würde. Daher konnte keine statische Webseite entwickelt werden. Die Entwicklung einer gewöhnlichen nativen Oberfläche wurde ebenfalls ausgeschlossen, um die Möglichkeit offen zu halten, das System in Zukunft ohne Installation als reine Browser-Applikation zu verwenden. Im Absprache mit Eaternity entschieden wir, dass Githubs Electron[3] für die Entwicklung des Frontends verwendet werden sollte. Dieses Framework erlaubt es, betriebssystemunabhängig Desktop-Applikationen mit JavaScript, HTML und CSS zu entwickeln. Es stellt die meisten Module von Node.js zur Verfügung und erlaubt es, Elemente der betriebssystemeigenen Benutzeroberfläche zu verwenden. Für die Entwicklung der eigentlichen Benutzeroberfläche wurden folgende JavaScript-Frameworks in Betracht gezogen:

[3] http://electron.atom.io

- *jQuery*[4] ist die weltweit am weitesten verbreitete JavaScript Bibliothek. Sie bietet Funktionen zur Bearbeitung der HTML-Struktur von Webseiten und für asynchrone Programmierung von Webanwendungen. Sie ist vor allem für die Anbindung simpler Logik an Webseiten-Elemente ausgelegt. Komplexere Logik kann mit jQuery zu unübersichtlichen Projektstrukturen führen.

- *AngularJS*[5] (Angular) ist ein Open-Source JavaScript-Framework welches auf die Entwicklung vollständiger MVC- oder MVVM-Architekturen in JavaScript für Webseiten ausgelegt ist. Angular wurde mit einer klaren Vorstellung zur Struktur einer Web-Applikation entwickelt und erzwingt die Einhaltung der Strukturen oder deutlichen Aufwand bei deren Umgehung. Angular wird von Google unterhalten.

- *React*[6] ist ein Open-Source JavaScript-Framework, welches die Aktualisierung der Darstellung bei der Veränderung eines zugrundeliegendes Datenmodells optimiert. Es wird von der Community, sowie von Facebook und Instagram unterhalten. Im Gegensatz zu Angular gibt es für React kaum klare Vorgaben für die Projektstruktur und der Benutzer ist bei deren Entwurf auf sich alleine gestellt. Dies lässt dem Benutzer mehr Freiheit, macht das Framework jedoch auch schwerer verständlich, da verschiedene Lernmaterialien zu React sich stark unterscheiden.

jQuery ist für die Entwicklung eines komplexen Frontends nur bedingt geeignet. Die Entscheidung zwischen AngularJS und React ist eine Frage der Philosophie der Entwickler. Sowohl React als auch Angular sind moderne JavaScript-Frameworks, welche die Entwicklung komplexer Web-Applikationen ermöglichen. Das Frontend des Prototypen verwendet Electron und AngularJS, da AngularJS' vorgegebene Projektstuktur das Nachvollziehen des Codes in Zukunft vereinfachen soll. Die Verwendung der Electron-Komponenten beschränkt sich auf die Verwendung mehrerer Anwendungsfenster, das automatische Starten des Backends und den Zugriff auf das Dateisystem zur lokalen Speicherung von Backups.

6.2.2 Backend

Die Entscheidung für ein geeignetes Backend-Framework wurde anhand folgender Kriterien getroffen:

- *Dokumentation:* Es sollten ausreichend Lernmaterial, also Tutorials, Beispielprojekte und Dokumentation vorhanden sein.

- *Support:* Das Framework sollte noch weiter entwickelt werden. Damit sollte sichergestellt werden, dass die während der Entwickung des Prototypen gemachten Erfahrungen auch langfristig Gültigkeit bewahren.

[4] https://jquery.com
[5] https://angularjs.org
[6] https://facebook.github.io/react/

- *Programmiersprache:* Die Kundenapplikation von Eaternity ist in Java geschrieben. In Zukunft ist die Verwendung und Einbindung von Brightway[7] geplant, ein LCA-Tool, welches in Python geschrieben wurde. Die Frontend-Applikation würde in JavaScript geschrieben werden. Um nicht noch mehr verschiedene Programmiersprachen im Unternehmen zu verwenden, sollte das Backend in Java, Python oder JavaScript geschrieben werden.

- *Entwicklerfreundlichkeit:* Die Entwicklung mit dem gewählten Framework sollte unkompliziert und intuitiv sein.

Im Folgenden werden die Frameworks, welche aufgrund dieser Kriterien in Betracht gezogen wurden, kurz vorgestellt:

- *Django* Django ist ein Web-Framework, welches bereits sehr viel Funktionalität mit sich bringt, z. B. Funktionen für die dynamische Generierung von Webseiten (templating engine), Benutzerverwaltung und eine Test-Umgebung. Es hat von allen Python Webframeworks die grösste Benutzergemeinschaft mit über 80'000 Fragen auf der Frage-Antwort-Webseite StackOverflow.

  Django ist zwar in Python geschrieben, jedoch ist für eine Erweiterung eine strikte Struktur notwendig, da Django-Applikationen über einen sogenannten Bootstrapper gestartet werden und dieser nur spezifisch für Django entwickelte Erweiterungen starten kann. Dies hat zur Folge, dass andere Python-Pakete nicht ohne Modifikationen eingebunden werden können, was den Sprachvorteil teilweise wieder zunichte macht. Django erscheint ausgerichtet für grosse Projekte mit vielen Standardkomponenten, welche bereits als Django-App verfügbar sind.

- *Flask* Flask ist im Vergleich zu Django sehr puristisch. Benutzerverwaltung, Testumgebung und auch Object-Relation-Mapper sind nicht in Flask integriert. Im Gegensatz zu Django ist Flask modularer, d. h. es ist leichter, existierende Python-Pakete in Flask zu integrieren, da Kompatibilität lediglich auf Ebene der Programmiersprache erforderlich ist. Flask benötigt kein Bootstrapping und Flask-Applikationen können als eigenständige Python-Applikationen gestartet werden. Es ist für Projekte geeignet, die nur die zentralste Funktionalität eines Web-Service benötigen und ist offen für beliebige Projektstrukturen.

- *Pyramid* Pyramid wird mit mehr integrierter Funktionalität als Flask geliefert, das Integrieren von eigenen, zusätzlichen Paketen ist aber einfacher als bei Django. Es bildet in Hinsicht auf Modularität und Umfang einen Kompromiss zwischen Flask und Django.

- *Tomcat* Tomcat ist ein Web-Container für die Java Enterprise Edition, welcher das Betreiben von Java-Servlets und Java Server Pages ermöglicht.

[7] https://brightwaylca.org

Java-Servlets sind Java-Applets mit Server-Funktionalität. Sie sind auf die Bearbeitung und Beantwortung von Webanfragen ausgelegt. Java Server Pages sind Webseiten mit integriertem Java-Code, welche beispielsweise von einem Java-Servlet generiert werden können. Einige Alternativen zu Tomcat, welche ebenfalls den Spezifikationen der Java Enterprise Edition genügen, sind Glassfish, JBoss oder Jetty.

- *Play* Play ist ein Web-Framework, welches in Scala geschrieben ist und eine Java-API anbietet. Im Gegensatz zu vielen anderen Java-basierten Lösungen ist Play nicht an die Java Enterprise Edition gebunden, sondern beinhaltet die Plattform-Funktionalität selbst. Es ist eine der wenigen Alternativen zur Java Enterprise Edition.

- *Node.js* Node.js ist ein Runtime Environment für JavaScript und beinhaltet Standardpakete zur einfachen und schnellen Erstellung von Web-Servern. Ähnlich wie Flask bietet Node.js daher die Möglichkeit, nur die zentralste Funktionalität eines Web-Services in eine Applikation zu integrieren ohne die Erweiterung der Server-seitigen Anwendungslogik zu behindern.

Die Wahl für das Backend fiel auf das Python-Framework Flask. Dafür sprach die Verfügbarkeit von Tutorials zur Kombination mit Electron. Zudem versprach es die einfachste Einbindung von Brightway, sollte diese Anforderung tatsächlich aufkommen. Die Modularität und Simpliziät des Frameworks machte Flask für die Entwicklung eines reinen Backends zudem attraktiver als die anderen Möglichkeiten.

6.2.3 Datenbank

Die Auswahl der Datenbanktechnologie fokussierte sich in erster Linie auf die Frage, ob weiterhin eine schemalose Datenbank zum Einsatz kommen sollte oder der Umstieg auf eine relationale Datenbank sich lohnen würde. Da schemalose Datenbanken dem Benutzer die Möglichkeit geben, sich in einzelnen Fällen nicht an ein vorgegebenes Schema zu halten und es mein Ziel war, ein verpflichtendes Datenmodell zu entwerfen, entschied ich mich für die Verwendung einer relationalen Datenbank. Es wurden die bekanntesten SQL-Implementationen, SQLite, MySQL und PostgreSQL in Erwägung gezogen wobei keine der Sprachen grundsätzliche Vorteile für die geplante Anwendung aufwies. Als konkrete Technologie wählte ich PostgreSQL, da so mit dem in Flask integrierten Object Relation Mapper die beste Funktionalität und vollständigste Dokumentation erreicht wurde.

6.3 Datenmodell

Während der Beobachtungsphase wurden, wie in Abbildung 3.1 ersichtlich, die bestehenden Daten in einem Entity Relationship Model (ERM) dargestellt. Im

Abschnitt 6.3.1 wird auf die Schwächen des alten Datenmodells eingegangen und in Abschnitt 6.3.2 beschrieben, welche Schritte unternommen wurden, um ein verbessertes Datenmodell zu erstellen.

6.3.1 Analyse des ursprünglichen Modells

Das ursprüngliche Modell, basierend auf den JSON-Dateien, anhand dessen die Daten bisher verwaltetet wurden, weisst Eigenschaften auf, welche für die Benutzung Herausforderungen darstellen:

- Das Datenmodell ist nicht normalisiert und weisst Redundanzen auf. Mit einer grossen Menge an mehrwertigen und mehrfachen Attributen erfüllt das Datenmodell beispielsweise in mehrfacher Instanz nicht die erste Normalform (vgl. [Elmasri and Navathe, 2011, Abschnitt 15.3.4]).

- Lebensmittelabfalldatensätze beinhalten ein Attribut zum Namen des Produkts, für welches der jeweilige Datensatz gültig ist. Dennoch können mehrere Produkte in Relation zum selben Datensatz stehen. Der Sachverhalt ist identisch für Nährwertdatensätze. Es ist somit für den Benutzer, bzw. die Benutzerin notwendig, das Attribut *name* des Datensatzes mit dem Namen des Produktes zu vergleichen, um nachzuvollziehen, ob der Datensatz gemäss Quelle, aus welcher der Datensatz entnommen wurde, diesem Produkt zuzuordnen ist.

- Ein Prozess kann gleichzeitig sowohl Produktions-Methoden (Relation *production-method*), als auch Verpackungs-, Preservations- und Verarbeitungsmethoden für mehrere Produkte darstellen. Anstatt die Art des Prozesses als dessen Eigenschaft zu modellieren, wird sie durch die Beziehung zum Produkt dargestellt.

- Referenzen auf die Datenherkunft werden in unterschiedlicher Art an verschiedenen Orten im Modell dargestellt. Die Herkunft von CO_2-Werten wird als Attribut von Produkten modelliert. *FoodWasteData* Entitäten haben ein Attribut welches die Referenz für alle enthaltenen Werte beinhaltet und für Nährwertdaten ist die Datenherkunft implizit im Attribut, *id* der *NutrientData* Entität erfasst. Dadurch ist es schwierig, den Beitrag einer einzelnen Referenz zum gesamten Datensatz nachzuvollziehen.

- Es können einem Produkt nicht die Werte für mehrere Länder zugeordnet werden, da die Datenstruktur nur einen Wert pro Produkt erfassen kann. Die Erweiterung der Datenbank auf die Erfassung internationaler Werte erfordert Anpassungen.

Das alte Modell weisst gemäss der in Abschnitt 4.2 aufgelisteten Kriterien deutliche Schwächen auf: Es ist nicht ausreichend, um die von Eaternity angestrebte Geschäftstätigkeit abzubilden und ist nicht auf die nötigen Erweiterung ausgelegt, bildet Geschäftsregeln unvollständig ab und erfasst gleichartige Werte an mehreren Stellen.

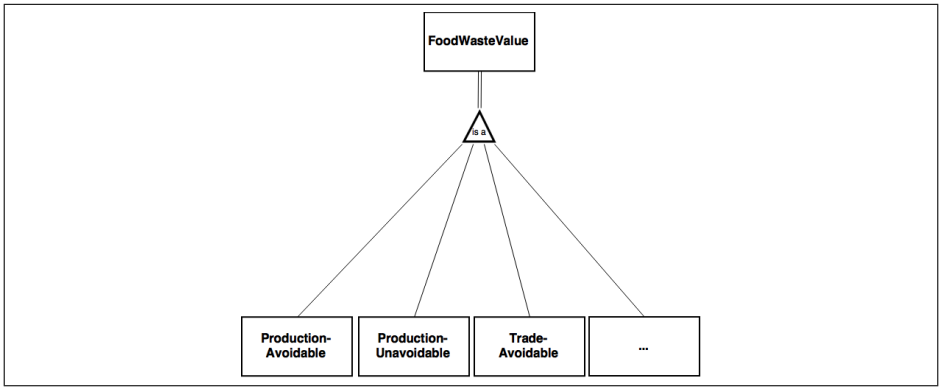


Abbildung 6.4: Alternative Modellierung der FoodWasteData Entität

6.3.2 Neues Datenmodell

Zur Verbesserung des Datenmodells wurden zunächst die nötigen Normalisierungen vorgenommen. Anschliessend wurde unter Anstrebung der in Abschnitt 4.2 beschriebenen Eigenschaften das Datenmodell weitgehend umstrukturiert und an die erhobenen Anforderungen angepasst. Folgende neue Entitäten wurden hinzugefügt:

- *Value mit abgeleiteten Entitäten:* Mit der Entität *Value* wurden mehrere Elemente der ursprünglichen Modellierung zusammengefasst, welche gemeinsam haben, dass sie aus externen Quellen zusammengetragen werden und einem Produkt einen Wert oder eine Aussage zuordnen. Wie aus der Abbildung hervorgeht, vereinfacht dies die Modellierung der *Product* Entität, da diese für die verschiedenen Arten von Values nun nur noch eine einzige Relation aufweist. Indem alle Werte, welche von einer definierbaren Quelle übernommen werden, in einer Entität repräsentiert werden, erhöht dies die Flexibilität des Datenmodells. Wird beispielsweise von Eaternity in Zukunft auch der Wasserverbrauch eines Produktes erfasst, kann eine neue Entität von *Value* abgeleitet werden, analog zu jener des CO_2-Wertes (siehe Abbildung 6.6).

- *Product-Process-Nutrient Association:* Diese aus der *Value* Entität abgeleitete Entität ging aus der bisher nicht modellierten Nährwertänderung durch Prozesse hervor. Sie stellt die Veränderung des Gehaltes eines spezifischen Nährstoffs in einem Produkt bei der Anwendung eines Prozesses dar. Beispielsweise würde eine solche Entität, verbunden mit dem Produkt Tomate, dem Prozess Kochen und dem Nährstoff Calcium, die Veränderung des Calciumgehaltes beim Kochen einer Tomate darstellen.

- *TemplateProduct und EDBProduct als von Product abgeleitete Entitäten:* Nicht alle Werte, welche Eaternity erfasst, sind einem konkreten Produkt zuzuordnen. Es kann vorkommen, dass die Mitarbeitenden Informationen

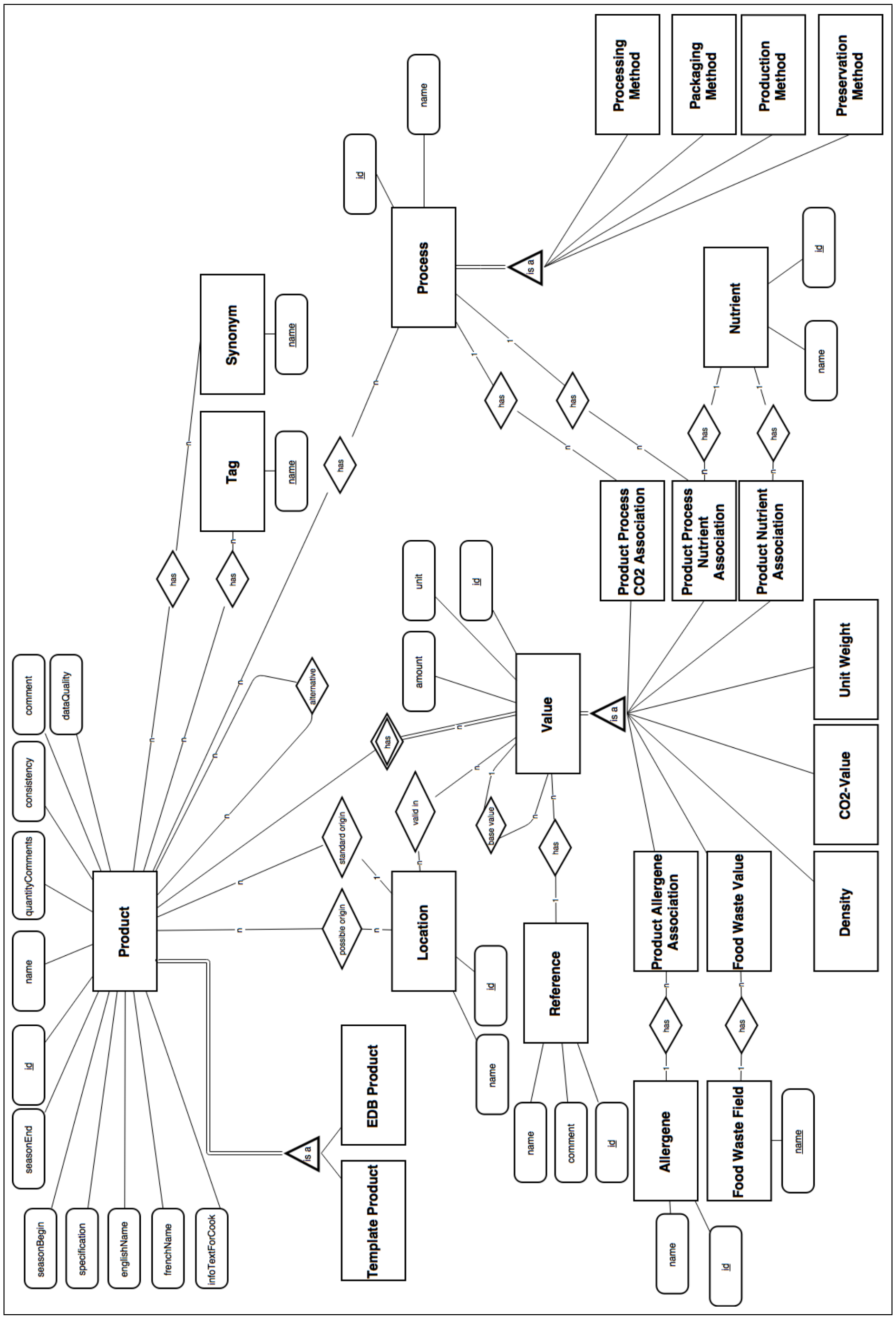


Abbildung 6.5: Das neue Datenmodell als ERM

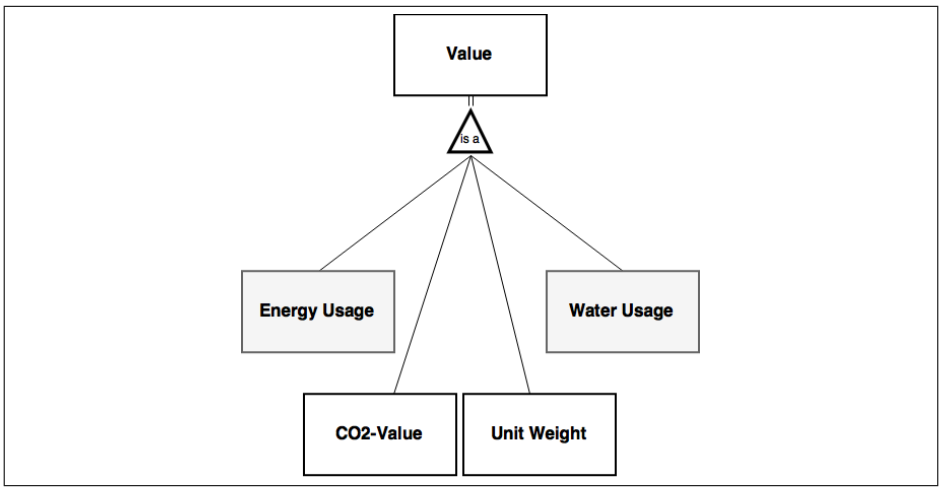


Abbildung 6.6: Es können leicht zusätzliche Werte modelliert werden

zu einer Menge von Produkten oder zu einem hypothetischen Produkt erfassen möchte. Im neuen Modell wird ein solches abstraktes Produkt als *TemplateProduct* modelliert. Somit bleibt die Zuordnung des Wertes zu den Produkten, zu denen sie gemäss Quelle zuzuordnen sind, erhalten.

- *Tags und Synonyme:* Diese Entitäten entstanden während der Normalisierung aus den gleichnamigen Attributen der Product Entität. Die Modellierung von Tags und Synonymen als eigene Entität vereinfacht die Abfrage aller vergebenen Tags und Synonyme was beispielsweise bei der Entwicklung einer Benutzeroberfläche von Vorteil ist, wenn dem Benutzer oder der Benutzerin während der Vergabe von Tags bereits verwendete Tags vorgeschlagen werden sollen. Da die Mehrfachvergabe von Synonymen dank dieser Modellierung einfacher nachzuvollziehen ist, kann ausserdem leichter festgestellt werden, wenn ein Produkt mehrfach erfasst wurde.

- *Process und abgeleitete Klassen* Anstatt Prozesse sowohl als Attribute der Product Entität zu modellieren, als auch als Entität, um die Änderung des Fussabdruckes zu modellieren, wurden diese Elemente vereint zu einer Prozess Entität. Anstatt den Typ der einzelnen Prozesse über deren Relation zu Produkten zu modellieren, wurden diese Typen mit abgeleiteten Klassen modelliert. So kann auch verhindert werden, dass ein Prozess sowohl als Verarbeitungs-, als auch als Produktionsprozess modelliert wird.

- *Allergene:* Diese Entität entstand während der Normalisierung aus dem mehrwertigen Attribut *allergenes* der Product Entität. Ähnlich wie bei den Tags ergibt sich auch hier der Vorteil, dass dem Benutzer während der Erfassung eine Liste der erfassten Allergene dargestellt werden kann.

- *Nutrient:* Diese Entität entstand während der Normalisierung aus dem

mehrwertigen Attribut nutrients der Klasse Nutrient-Data. Auch hier ergibt sich der Vorteil, dass bei der Erfassung von Nährwerten durch Benutzer oder Benutzerinnen leichter Vorschläge angezeigt werden können. Zudem kann diese Information für die Erfassung von Nährwertänderungen ebenfalls genutzt werden.

- *Reference:* Diese Entität fasst mehrere Attribute der Product Entität und das Attribut references der FoodWasteData Entität zusammen. Sie modelliert Dokumente, aus denen Werte entnommen wurden. Somit kann weiterhin nachvollzogen werden, woher die einzelnen Werte stammen. Es ist nun jedoch einfacher, alle Werte zu finden, welche aus einem Dokument stammen. In Zukunft kann das Datenmodell durch den Ausbau der *Referenz* Entität einfach erweitert werden, um Beziehungen zu Datenlieferanten zu verwalten.

- *FoodWasteField:* Diese Entität entstand aus der Aufteilung der ursprünglich in die FoodWasteData Entität zusammengefassten Werte in einzelne Werte. Es ist als Alternative zur starren Modellierung der verschiedenen Attribute der ursprünglichen FoodWasteData Entität als abgeleitete Klassen der neuen FoodWasteValue Entität zu betrachten (vgl. Abbildung 6.4). Dies verspricht Stabilität, da das Datenmodell bei der Erfassung von Abfällen in weiteren Phasen des Produkt-Lebenszyklus' nicht angepasst werden muss.

Neben dem Einfügen neuer Entitäten wurden weitere Änderungen am Datenmodell vorgenommen:

- Während bisher ein Produkt maximal einem CO_2-Wert, einem Nährwertdatensatz und einem Lebensmittelabfalldatensatz zugeordnet werden konnte, jedoch mehrere Produkte dem gleichen, können nun mehrere gleichartige Werte dem selben Produkt zugeordnet werden. Somit ist es möglich, die Werte von mehreren Quellen in der Datenbank zu erfassen und Produkten zuzuordnen. Ausserdem ist es den Mitarbeitenden nun möglich, einem Produkt Werte der gleichen Art für mehrere Länder zu hinterlegen.

- Im neuen Modell ist es nicht mehr möglich einen Wert anzulegen und ihn direkt mehreren Produkten zuzuweisen. Dabei ginge die Information, für welches Produkt der Wert gemäss Datenliefarant zutrifft, verloren (vgl. Abbildung. 6.7). Stattdessen wird im neuen Modell, falls für ein Produkt ein Wert nicht bekannt ist, ein temporärer Wert erstellt. Diesem kann dann über die Relation *base value* der Wert eines anderen Produktes zugrundegelegt werden (vgl. Abbildung. 6.8).

- Es wurden einige Attribute entfernt, welche im alten Datenmodell bereits nicht mehr relevant für Eaternity waren. Das Attribut *isCombinedProduct* der Entität *Product* war beispielsweise ein Relikt aus einem Versuch, Daten aus der Eaternity-Cloud in die Datenbank zurückzuführen.

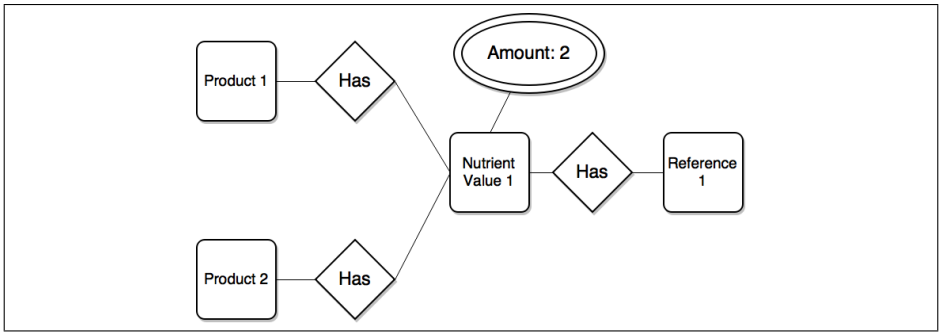


Abbildung 6.7: Bisher wurden Werte mehreren Produkten zugeordnet.

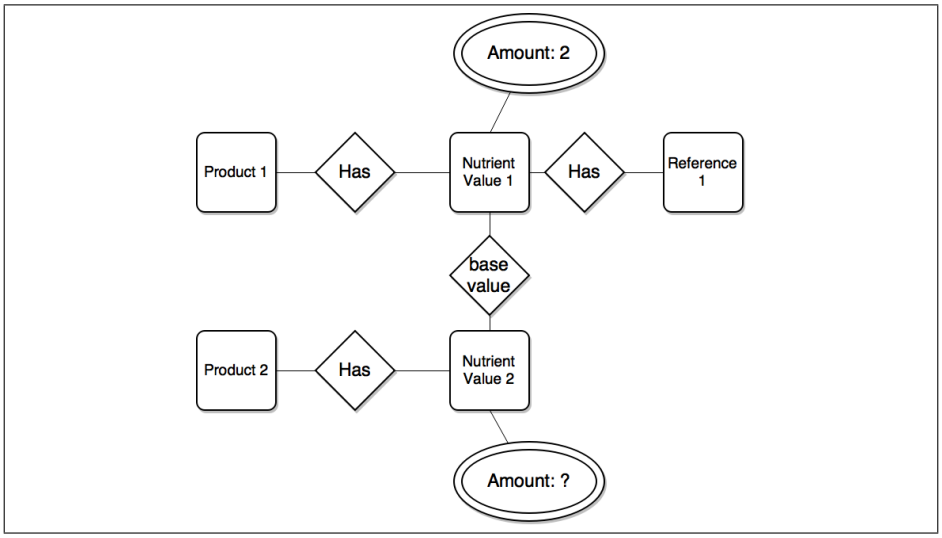


Abbildung 6.8: Neu gehört ein Wert zu genau einem Produkt.

Es wurden somit in Bezug auf die in Abschnitt 4.2 beschriebenen Eigenschaften deutliche Verbesserungen erreicht. Insbesondere bringen die Anpassungen Vorteile für die *Flexibilität* und die realitätsnahe Abbildung von *Geschäftsregeln*.

6.4 Prototyp

In diesem Abschnitt wird der entwickelte Prototyp beschrieben. In Abschnitt 6.4.1 beschreibe ich den Entwurf der Datenbank mit Hilfe des Objekt Relation Mappers SQLAlchemy. Die Web API sowie die JSON-Repräsentation der Daten wird im Abschnitt 6.4.2 erläutert. Die Benutzeroberfläche wird im Abschnitt 6.4.3 beschrieben und die Methoden zu Transformation der Daten aus dem alten Format, welche im Frontend ablaufen, wird im Abschnitt 6.4.4 im Detail besprochen.

6.4.1 Datenbank

Flask wird mit einer eigenen Version des Object Relation Mappers SQLAlchemy geliefert. Dieser erlaubt es, mit gewissen Restriktionen Python-Klassen zu definieren und automatisiert die Verwaltung in einer relationalen Datenbank. Für den Prototypen wurde eine PostgreSQL-Datenbank gewählt, da SQLAlchemy mit dieser die umfangreichste Funktionalität bietet. Mit SQLAlchemy wurde das in Abbildung 6.5 ersichtliche ERM in Python implementiert und durch SQLAlchemy in ein Datenbankschema übersetzt.

6.4.2 API

Die API welche vom Backend bereitgestellt wird, ermöglicht vollständige CRUD-Funktionalität (create, retrieve, update und delete) für die Datenbank. Somit kann das Backend auch mit eigenen Frontends, sowie mit Kommandozeilenprogrammen wie *curl* oder dem für das Testen von Web-APIs entwickelten Programm *Postman*[8](siehe Abbildung 6.9). Die API wurde im Sinne des Programmierparadigmas REST (Representational State Transfer) entworfen. Dabei wurden die Regeln aus [Masse, 2012], sofern als zwingend beschrieben, eingehalten. Insbesondere folgende Aspekte wurden berücksichtigt:

- *HTTP-Methoden:* Für die Erstellung, Ansicht, Veränderung und Löschung der zugänglichen Ressourcen werden ausschliesslich die Methoden GET, PUT, POST und DELETE verwendet, wie es gemäss den Regen in [Masse, 2012, S. 23 ff.] vorgegebenen ist.
- *Idempotenz:* Anfragen mit GET, PUT und DELETE sind idempotent, können also mehrfach hintereinander ausgeführt werden, ohne das Resultat der Anfrage zu verändern.

[8]https://www.getpostman.com

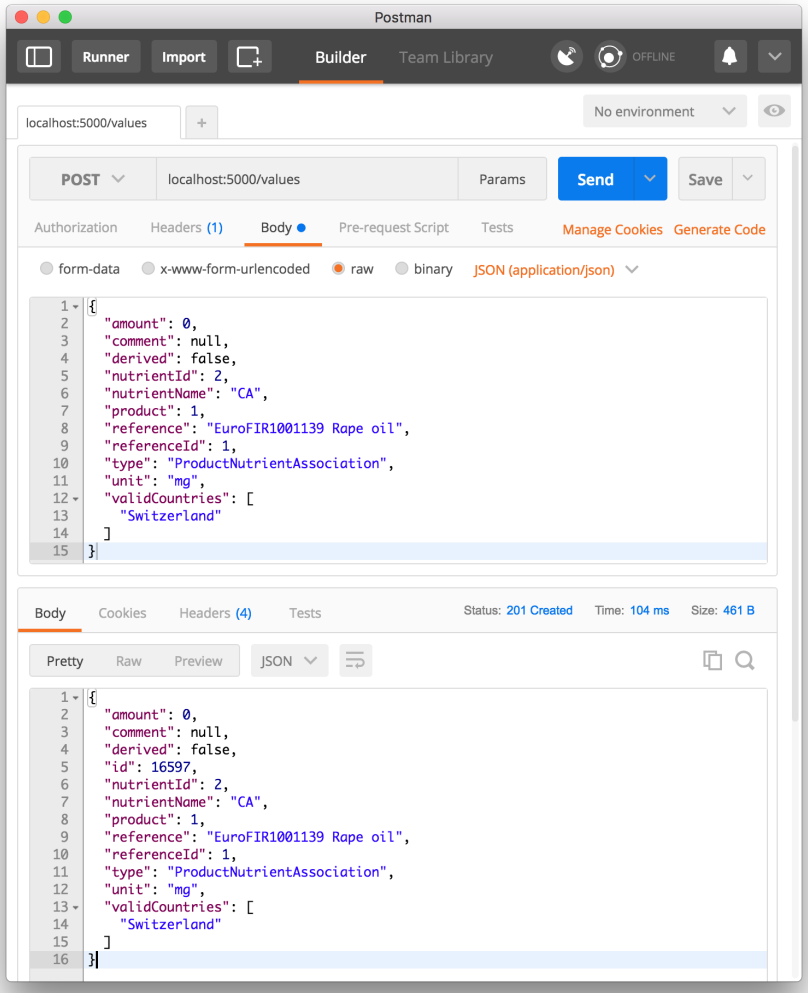


Abbildung 6.9: Anlegen eines Wertes über die API mit Postman

- *Status-Codes:* Bei der Beantwortung von Anfragen liefert die API Status-Codes gemäss [Masse, 2012, S. 28 ff.] wobei, wenn möglich, nicht generische sondern spezifische Codes vewendet werden.

- *Partielle Anfrage:* Um die Antwortzeit bei grossen Anfragen zu reduzieren, können auf Wunsch nur ausgewählte Felder von Ressourcen angefragt werden (vgl. [Masse, 2012, S. 74]).

Die Empfehlungen aus [Masse, 2012] wurden im Allgemeinen ebenfalls befolgt. Für nicht als zwingend bezeichnete Empfehlungen (siehe [Masse, 2012, S. 7] wurden zum Teil Ausnahmen gemacht. Insbesondere folgende Abweichungen sind zu bemerken:

- *Fehler-Repräsentation:* Wie von Masse [2012] gefordert, wird bei Antworten eine einheitliche Fehlerrepräsentation verwendet. Auf die ausführliche Modellierung, wie im Beispiel [Masse, 2012, S. 69] wird jedoch verzichtet.

- *Repräsentation:* Auf die Bereitstellung eines "format-agnostischen" ([Masse, 2012, S. 59]) Schemas wird verzichtet, da weder bisher noch für die nahe Zukunft die Anforderung besteht, die API selbsterklärend zu gestalten. Für den Einsatz ist eine Dokumentation der API auf Ebene der natürlichen Sprache ausreichend.

Die Schnittstelle akzeptiert JSON-Dateien und erfolgreiche Antworten auf gültige Anfragen enthalten ebenfalls JSON oder haben keinen Inhalt. Im Folgenden sind die verschiedenen Endpunkte umrissen. Für detaillierte Informationen kann die Dokumentation konsultiert werden.

- *http://localhost:5000/products:* Mit einer GET-Anfrage können über diesen Endpunkt sämtliche im Backend gespeicherten Produkte abgefragt werden. In der URL können über zusätzliche Parameter die Attribute der Produkte, welche in der Antwort enthalten sein sollen, eingeschränkt werden, um die Antwortzeit zu verkürzen. Mit einer POST-Anfrage kann die JSON-Repräsentation eines Produktes im Nachrichtenrumpf an den Server gesendet werden. Dieses wird dann als neues Produkt in der Datenbank abgelegt und die lokale Repräsentation wird in der Antwort zurückgesendet.

- *http://localhost:5000/products/<id>:* Mit einer GET-Anfrage kann über diesen Endpunkt das Produkt mit der spezifizierten ID abgefragt werden. Mit einer PUT-Anfrage kann die JSON-Repräsentation eines Produktes im Nachrichtenrumpf an den Server gesendet werden. Dieses wird dann, falls gültig, anstelle der vorherigen Repräsentation in der Datenbank gespeichert. Mit einer DELETE-Anfrage kann das Produkt gelöscht werden.

- *http://localhost:5000/values und http://localhost:5000//values/<id>:* Dieser Endpunkt funktioniert analog zu jenem für Produkte. Anstelle von Produkten können hier Werte ausgelesen, erstellt und geändert werden.

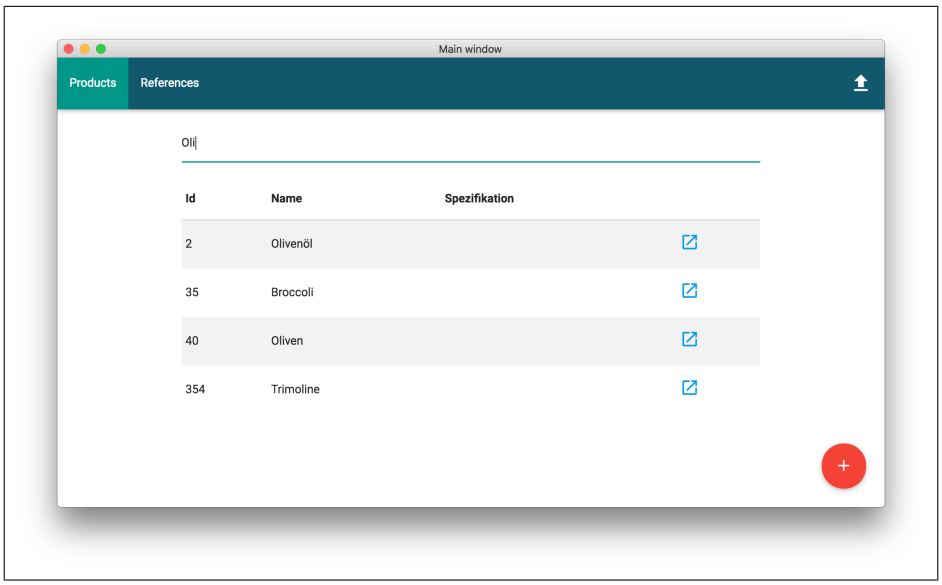


Abbildung 6.10: Das Hauptfenster der Applikation

- *http://localhost:5000/references/<id>*: Dieser Endpunkt funktioniert ebenfalls analog zu jenen für Produkte und Werte.
- *../allergenes, ../nutrients, ../processes*: Diese Endpunkte erlauben ausschliesslich das Auslesen der jeweiligen Entitäten mittels einer GET-Anfrage.

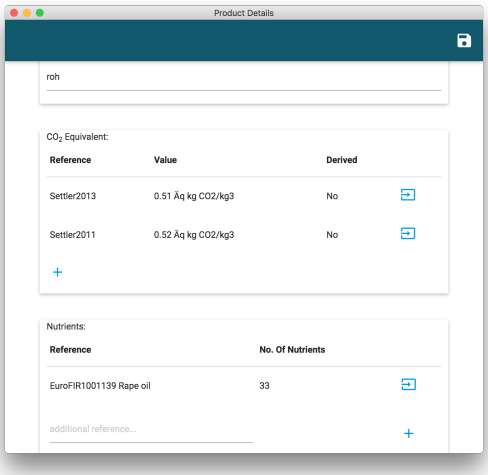
Die API wurde so entworfen, dass erfolgreiche Antworten auf GET-Requests auch als Rumpf für eine PUT- bzw POST-Anfrage verwendet werden können. Somit wird eine ausreichende Selbsterklärung des JSON-Formats der API gewährleistet, um das Konsultieren der Dokumentation für die Änderung der abgelegten Ressourcen zu vermeiden.

6.4.3 Benutzeroberfläche

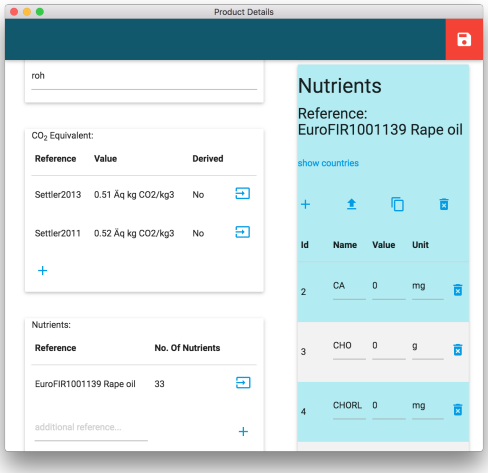
Die Benutzeroberfläche unterteilt sich in zwei Fenster:

- Das *Hauptfenster* (siehe Abbildung 6.10) erscheint beim Starten der Applikation. Dem Benutzer bzw. der Benutzerin wird eine Übersicht über den Datenbestand angezeigt. Es können sowohl die Produkte als auch die Referenzen betrachtet werden.

  Es besteht die Möglichkeit, die gesamten Daten aus der Datenbank als JSON-Repräsentation auf dem lokalen Datensystem zu speichern. Dies dient gleichzeitig zur Versionierung und Synchronisation, da die erstellten JSON-Daten im gehosteten Github-Repository abgelegt und abgeglichen werden können. Ausserdem ist es möglich, in eine leere Datenbank Daten

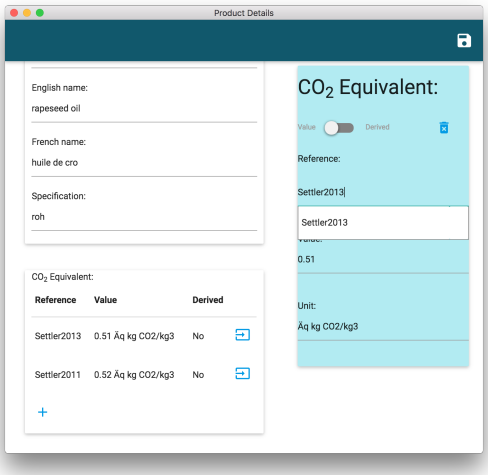


(a) Hauptansicht

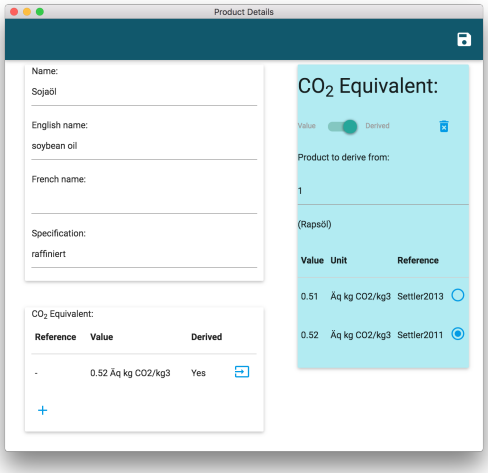


(b) Details Nährwerte

Abbildung 6.11: Produktfenster



(a) Referenz hinterlegen



(b) Grundwert auswählen

Abbildung 6.12: CO_2-Wert bearbeiten

hochzuladen, welche im Format der bisher verwendeten Datenbanklösung im lokalen Dateisystem gespeichert sind.

Die Übersicht über die Produkte ist als Tabelle gestaltet, in der ID, Name und Spezifikation aller Produkte sichtbar sind. Die Tabelle ist sortierbar anhand der verschiedenen Spalten und über ein Textfeld können die Zeilen in Echtzeit anhand einer Volltextsuche gefiltert werden. Es können auch neue Produkte hinterlegt werden.

Aus der Tabelle kann ausgewählt werden, welches der Produkte im *Produktfenster* angezeigt werden soll.

- Im *Produktfenster* (siehe Abbildung 6.11) lassen sich die Attribute eines Produktes sowie die zugehörigen Werte verändern. Die Werte können mit Referenzen versehen werden. Dazu wurde eine Auto-Vervollständigung implementiert, um dem Benutzer bzw. der Benutzerin während der Eingabe Vorschläge anzuzeigen die korrekte Schreibweise zu vereinfachen (siehe Abbildung 6.12a).

  Es können neue CO_2-Werte erfasst und der Wert sowie die Masseinheit geändert werden. Es ist möglich, CO_2-Werten den CO_2-Wert eines anderen Produktes als Basiswert zuzuweisen (siehe Abbildung 6.12b). Dazu kann zuerst das Produkt ausgesucht werden, von welchem der Wert übernommen werden soll und dann aus einer Übersicht einer von dessen CO_2-Werten als Grundwert ausgewählt werden.

  Es können neue Gruppen von Nährwerten erstellt und diesen Gruppen neue Nährwerte, eine Referenzangabe, sowie die Länder, für die die Werte gültig sind, hinzugefügt werden.

  Dem Benutzer wird über ein rotes Speichersymbol angezeigt, wenn veränderte Attribute noch nicht auf den Server übertragen wurden (siehe Abbildung 6.11b). Stimmt die lokale Repräsentation des Produktes wieder mit jener in der Datenbank überein, erlischt die Anzeige.

6.4.4 Konvertierung

Aus dem Hauptfenster heraus können Daten, welche im bisherigen JSON-Format vorliegen, für die Datenbank konvertiert werden. Dabei werden die Informationen aus den Produkt- und Nährwert-Dateien verarbeitet und in neue Entitäten, also Produkte, CO_2-Werte, Nährwerte und Referenzen transformiert, entsprechend der Restrukturierung des Datenmodells. Entstehen aufgrund von Inkonsistenzen in den alten Daten Konflikte, existieren beispielsweise mehrere Produkte mit der gleichen ID, werden diese wo möglich bereinigt und Änderungen werden protokolliert.

Kapitel 7

Schlussfolgerung

Mit dem restrukturierten Datenmodell und dem implementierten Prototypen bieten sich Eaternity grundlegende Verbesserungsmöglichkeiten für die Verwaltung von Lebensmitteldaten.

7.1 Resultate

Das entstandene Datenmodell weist klare Vorteile gegenüber dem bisherigen auf. Es ermöglicht, wie bisher einem Produkt Werte jeglicher Art zuzuweisen, welche gemäss Datenlieferanten zu einem anderen Produkt gehören. Dies ist aber jetzt im relationalen Modell nachvollziehbar. Ausserdem erlaubt es, für ein Produkt die Werte aus unterschiedlichen Referenzen zu übernehmen und realitätsgetreu für verschiedene Länder unterschiedliche Werte zu erfassen. Im neuen Datenmodell können Werte für abstrakte Produkte hinterlegt werden, welche dann vom System von konkreten Produkten unterschieden werden können. Diese abstrakten Produkte können beispielsweise Produktkategorien abbilden. Die generalisierte Modellierung der unterschiedlichen erfassten Daten ermöglicht in Zukunft die unkomplizierte Erweiterung des Datensatzes um weitere Umwelteinflüsse wie Wasserverbrauch und Energieverbrauch.

Der Prototyp demonstriert mit seiner modularen Architektur sowohl die nötigen Schritte zur Implementationen der wichtigsten Komponenten für einen zentralen Web-Service als auch für jene einer Desktop-Applikation. Durch die Verwendung von Electron ist die Benutzeroberfläche nicht nur auf den drei häufigsten Betriebssystemen nutzbar: Da der grösste Teil der Benutzerschnittstelle mit AngularJS implementiert wurde, ist er auch in einer Browser-Applikation wiederverwendbar. Anhand der implementierten Benutzeroberfläche wurde aufgezeigt, wie den Mitarbeitenden bei Eaternity Aufgaben zur Verknüpfung von Werten, Erfassung von Referenzen und Ansicht der Daten erleichtert werden können. Die Applikation erfordert zudem keine Autorisierung, ausser zum Abgleich der Daten mit dem Github-Repository. Der Prototyp sowie die technische Dokumentation befinden sich auf der CD, welche dieser Arbeit beiliegt.

Neben der für einen Prototypen typischen Unvollständigkeit hat das vorgeschlagene System auch Schwächen, welche in Zukunft zu überwinden sind: Die Verwendung des Object Relation Mappers SQLAlchemy bringt für das schlanke Backend mit dem geringen Umfang an Anwendungslogik kaum Vorteile. Im Gegenteil stellt er einen weiteren Konvertierungsschritt zwischen der relationalen Datenbank und der JSON-Repräsentation für die REST-Schnittstelle dar, welcher sowohl die Performance des Systems, als auch die Übersichtlichkeit des Projekts beeinträchtigt. Für die Implementation eines produktiven Systems anhand der gemachten Vorschläge empfiehlt es sich, auf die Verwendung zu verzichten, es sei denn, es ergäbe sich das Bedürfnis nach der Verwendung von Python-Klassen im Backend. Das für den Prototypen entwickelte Modul für die Anzeige von Vorschlägen während der Eingabe von Referenzen, Allergenen und Nutrients ist für die Verwendung in einer produktiven Umgebung ebenfalls noch einmal zu überdenken oder das Modul ist so zu überarbeiten, dass es die Performance des Frontends nicht beeinträchtigt.

7.2 Ausblick

Der Zweck des Prototypen ist es, die Umsetzbarkeit der ausgearbeiteten Vorschläge sowie die Zweckmässigkeit des neuen Datenmodells zu demonstrieren. Es ist jedoch nicht der Zweck des Prototypen, das bisherige System bereits zu ersetzen. In diesem Sinne bietet der Prototyp den Anstoss für die eigentliche Entwicklungsarbeit. Auch wurden nicht alle erhobenen Anforderungen bei der Entwicklung des Prototypen berücksichtigt.

Eine Anforderung, welche Potential für spannende Fortsetzungen bietet, ist die programmatische Erfassung der Werte eines Produktes anhand dessen Kategorisierung. Daten zu Lebensmitteln werden oft kategorisch erhoben, beispielsweise nach Herkunft, nach Saison und nach Art. Fallen Produkte in mehrere Kategorien, widersprechen sich die Daten der Kategorien und sind keine spezifischen Informationen bekannt, führt dies unweigerlich in die Thematik der multiplen Vererbung, eine bekannte Herausforderung in der Datenmodellierung.

Eine der Anforderungen, welche eine andere Richtung als die im Prototypen behandelten Funktionen einschlägt, ist die Visualisierung des Datenbestandes. Das zugrunde liegende Bedürfnis für die Visualisierung des Datenbestandes ist das Auffinden von wenig abgedeckten Produktgruppen, um die Nachforschungen des Teams bei Eaternity in deren Richtung zu lenken. Diese Problemstellung fällt in das Gebiet der Datenanalyse und könnte durch Methoden des maschinellen Lernens und des Data Mining unterstützt werden.

In eine andere Disziplin fällt die Optimierung der Benutzeroberfläche. Die professionelle Entwicklung nach Prinzipien der Human Computer Interaction (HCI) hätte das Potential, die Produktivität der Mitarbeiter zu erhöhen.

Literaturverzeichnis

Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, Old Tappan, NJ, 3rd edition, 2003.

Sara L Beckman and Michael Barry. Innovation as a Learning Process: Embedding Design Thinking. *California Management Review*, 50(1):25–56, 2007.

Claudio Beretta, Franziska Stoessel, Urs Baier, and Stefanie Hellweg. Quantifying food losses and the potential for reduction in Switzerland. *Waste Management*, 33(3):764–773, 2013. ISSN 0956053X. doi: 10.1016/j.wasman.2012.11.007.

Nadya Direkova. Design Sprint Methods, 2015. ISSN 1098-6596. URL `https://developers.google.com/design-sprint/downloads/DesignSprintMethods.pdf`.

R Elmasri and S Navathe. *Fundamentals of Database Systems*. Addison-Wesley, Boston, MA, 6th edition, 2011. ISBN 9780136086208.

Fkostadinov. Source Code for Eaternity DBMS Solution based on GitHub, Jekyll, Heroku and Prose, 2015. URL `https://github.com/Eaternity/eaternity-database-public`.

Kevin D. Hall, Juen Guo, Michael Dore, and Carson C. Chow. The progressive increase of food waste in America and its environmental impact. *PLoS ONE*, 4(11):9–14, 2009. ISSN 19326203. doi: 10.1371/journal.pone.0007940.

Ulrich Kampffmeyer. Enterprise Content Management. *Legal Information Management*, 7(7):160–164, 2007. ISSN 1472-6696.

Ulrich Kampffmeyer and Jörg Rogalla. Grundsätze der elektronischen Archivierung. Technical report, VOI Verband Organisations-und Informationssysteme eV, Darmstadt, 1997.

Jake Knapp, John Zeratsky, and Braden Kowitz. *Design Sprint*. Transworld Publishers, London, 1.0 edition, 2016.

U Leser and F Naumann. *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. dpunkt-Verlag, Heidelberg, 2007. ISBN 9783898644006.

M Masse. *REST API Design Rulebook*. Oreilly and Associate Series. O'Reilly Media, Sebastopol, CA, 2012. ISBN 9781449310509.

G Simsion and G Witt. *Data Modeling Essentials*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco, CA, 2004. ISBN 9780080488677.