Lebensmittel und ihre Geheimnisse

Eine Web-App zum Aufdecken von Lebensmittelintoleranzen

Semesterarbeit an der Fernfachhochschule Schweiz

Philipp Tuor
Untersteinstrasse 7, 9450 Altstätten

CAS Full-Stack Development

Abgabedatum: 01.03.2025

Fachvorsteher CAS Full-Stack Development:

Markus Geuss
Fernfachhochschule Schweiz
Gleisarena Campus Zürich, Zollstrasse 17, CH-8005 Zürich

Dozenten:

Philipp Lauwiner
Fernfachhochschule Schweiz
Gleisarena Campus Zürich, Zollstrasse 17, CH-8005 Zürich

Dieter Rüetschi
Fernfachhochschule Schweiz
Gleisarena Campus Zürich, Zollstrasse 17, CH-8005 Zürich

Inhaltsverzeichnis

1 Ausgangslage

In den folgenden Kapiteln wird die Ausgangslage beschrieben, welche die Grundvoraussetzung zur Erstellung der Projektarbeit darstellt. Dabei wird die Aufgabenstellung sowie die Applikationsidee für die Projektarbeit erläutert. Anschliessend wird die geplante Vorgehensweise zur Umsetzung der Projektarbeit aufgezeigt.

1.1 Aufgabenstellung

Im CAS Full-Stack Development wird das Wissen vermittelt, um Web-Anwendungen von der Konzeption und der Entwicklung bis hin zum Deployment, mit den modernsten client- und serverseitigen Web-Technologien umzusetzen. Im Fokus ist dabei das praktische Anwenden etablierter Entwicklungswerkzeuge sowie die neuesten Trends in den Bereichen Usability und Benutzererfahrung. (FFHS, 2025)
Teil dieses CAS ist eine Projektarbeit bei der das erlernte Wissen angewendet und vertieft werden soll. Als technologische Rahmenbedingung wird React für das Frontend und Django als Backend vorausgesetzt. Des Weiteren ist es wichtig, dass das Frontend modern und responsiv gestaltet wird, um eine optimale Benutzererfahrung zu gewährleisten. Das gesamte Projekt sollte einer sauberen Full-Stack-Architektur entsprechen und gängige Patterns wie MVC, IoC und ORM integrieren. Zudem müssen Tests die Grundfunktionalität der Applikation abdecken und bei der Abgabe erfolgreich sein. Die Applikation muss für mehrere User ausgelegt sein und eine entsprechende Authentifizierung enthalten.

1.2 Applikationsidee

Es gibt Lebensmittelintoleranzen, die nur sehr schwer ausfindig gemacht werden können. Oft wird manuell Tagebuch geführt, was zu welchem Zeitpunkt gegessen wurde, und wann welche Symptome aufgetreten sind. Die Auswertung dieser Tagebücher ist mühsam und Aufwändig. Die zu entwickelnde Applikation zielt darauf ab, Menschen mit Lebensmittelintoleranzen eine benutzerfreundliche Plattform zu bieten, um ihre Ernährung und die damit verbundenen Symptome zu dokumentieren. Oftmals ist es für Betroffene eine Herausforderung, die Ursachen ihrer Beschwerden zu identifizieren, da die Symptome häufig erst Stunden oder Tage nach dem Verzehr bestimmter Lebensmittel auftreten. Die Applikation ermöglicht es den Nutzern, ein detailliertes Logbuch über die konsumierten Lebensmittel zu führen und gleichzeitig die aufgetretenen Symptome zu erfassen. Durch die Analyse dieser Daten wird die App in der Lage sein, mögliche Korrelationen zwischen bestimmten Lebensmitteln und den Symptomen aufzuzeigen. Dies geschieht ohne medizinische Empfehlungen oder Diagnosen, sondern dient lediglich der Unterstützung der Nutzer bei der

Identifikation von potenziellen Auslösern ihrer Beschwerden. Die intuitive Benutzeroberfläche und die einfache Handhabung der Applikation sollen es den Nutzern erleichtern, ihre Daten schnell und effizient einzugeben. Ziel ist es, den Prozess der Selbstbeobachtung zu vereinfachen und den Nutzern wertvolle Einblicke in ihre Ernährung und deren Auswirkungen auf ihr Wohlbefinden zu bieten. Die Auswertungs- und Analysemöglichkeiten der Applikation kann zukünftig erweitert werden und auch nach Abschluss der Semesterarbeit mit weiteren Funktionalitäten ergänzt werden.

1.3 Vorgehensweise

Agiles Projektmanagement ist in der Softwareentwicklung eine weit verbreitete Methode, um komplexe Projekte umzusetzen. Im CAS Full-Stack Development sind die Meilensteine der Semesterarbeit bereits vorgegeben. Die Umsetzung der Applikation orientiert sich an diese Meilensteine. Die Umsetzung des Projekts wird mittels GitLab Issues geplant und getrackt. Die Meilensteine der Projektarbeit werden als GitLab-Milestones definiert. Die Anforderungen der Applikation werden in Form von Issues erfasst und den entsprechenden Meilensteinen zugewiesen. Auf diese Weise kann der Fortschritt der Implementierung effektiv überwacht und gesteuert werden.

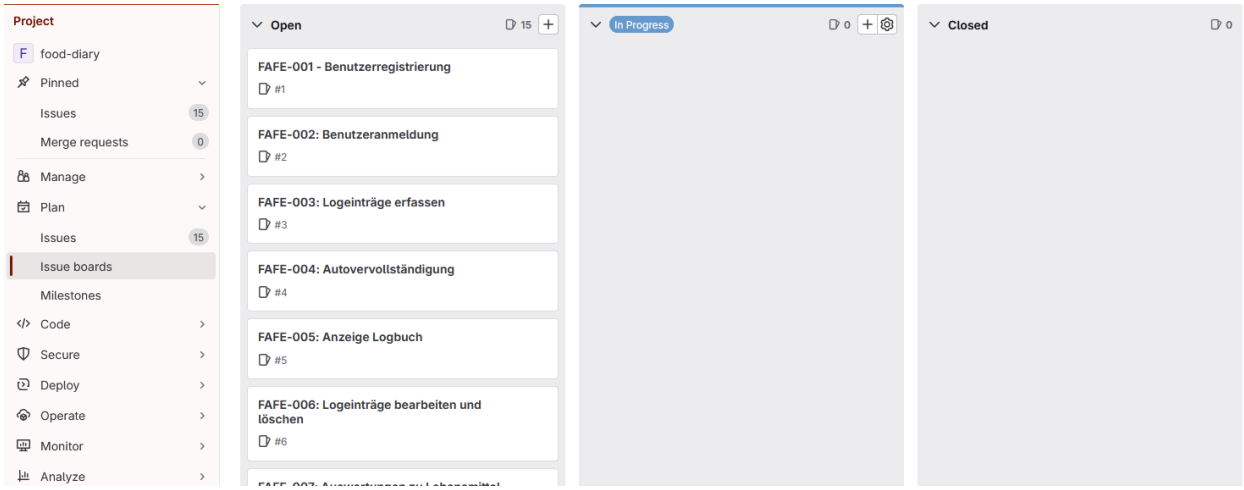

Abbildung 1:Issue Board innerhalb GitLab Project

1.4 Projektplanung

In einem ersten Schritt erfolgt die Projekt Initialisierung, bei der die Anforderung der Applikation definiert und die Umsetzung geplant wird. Am Meilenstein 1 erfolgt die Abgabe der Projektspezifikation mit dem Anforderungskatalog. Aufgrund des engen Zeitplans für die Umsetzung der Applikation beginnt parallel zur Besprechung und Freigabe der Projektspezifikation bereits die Implementierung der Applikation mit Sprint 1. Die Entwicklung der Applikation erfolgt in mehreren Iterationen, wobei der aktuelle Stand jeweils an den

Meilensteinen M1 - M4 den Dozenten zur Verfügung gestellt werden. Ziel ist es am Ende jedes Sprints einen funktionierenden Applikationsstand zu haben, sodass nach jedem Sprint die eingeplanten Features komplett umgesetzt sind.

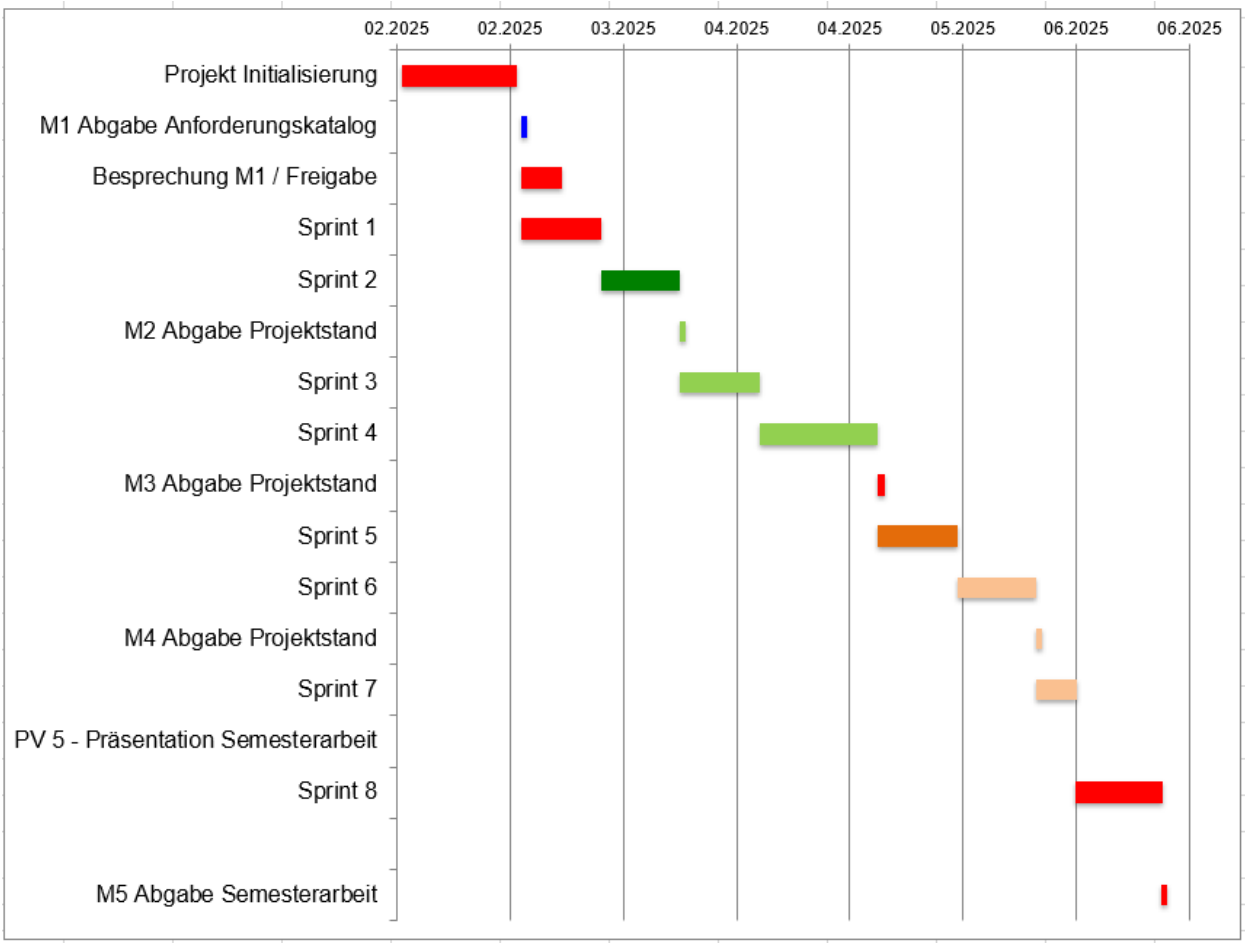

Abbildung 2: Zeitplan der Projektarbeit

2 Use Case Beschreibung

2.1 Use Case Diagramm

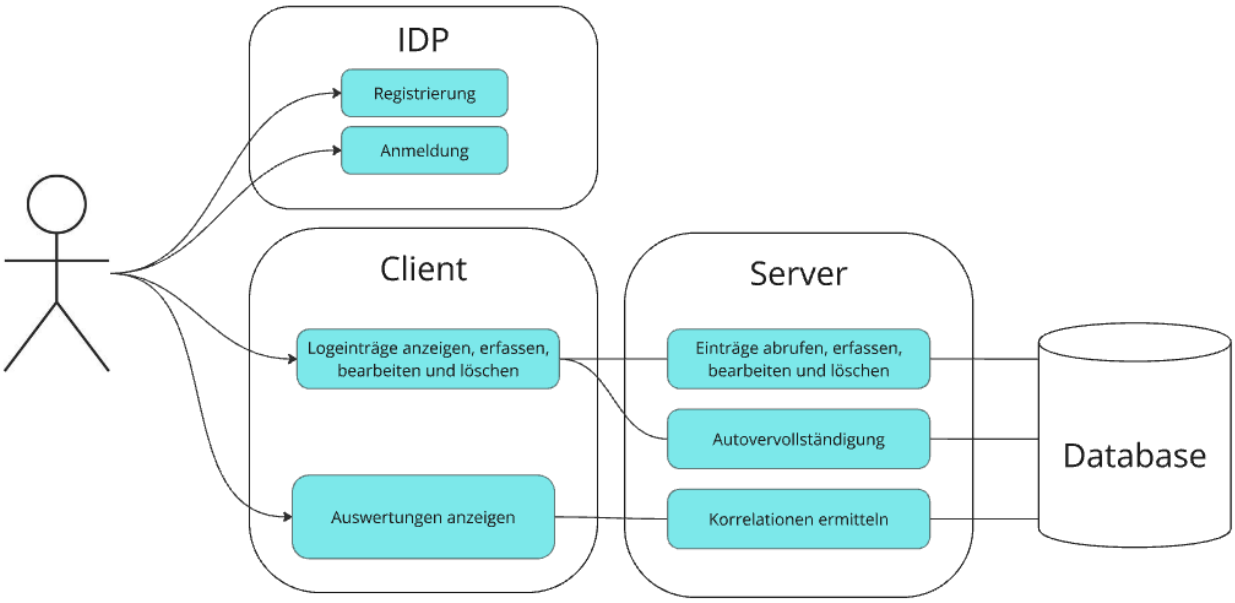

Abbildung 3: Use Case Diagramm

Das Use Case Diagramm visualisiert die Interaktionen zwischen den Benutzern und der Applikation, indem es die verschiedenen Anwendungsfälle darstellt, die die Benutzer durchführen können. Die Hauptakteure sind die Benutzer, die die Applikation nutzen, um Logeinträge im Ernährungslogbuch zu erfassen, sowie Analysen durchzuführen. Zudem interagieren die Benutzer mit dem IdP (Identity Provider) um sich zu registrieren und anzumelden. Um die Interaktion mit der Applikation zu vereinfachen, werden dem Benutzer bei der Erfassung eines Logeintrags geeignete Vorschläge in Form einer Autovervollständigung vorgeschlagen.

2.2 Wireframes

Die Wireframes dienen als konzeptionelle Entwürfe für die Erstellung der Benutzeroberflächen. Die Benutzeroberfläche muss sowohl für Desktopanwendungen wie auch für Mobileanwendungen optimiert sein. Wenn ein noch nicht angemeldeter Benutzer die Applikation öffnet, wird er auf eine Startseite geleitet, die ihm die Möglichkeit bietet, sich entweder einzuloggen oder zu registrieren.

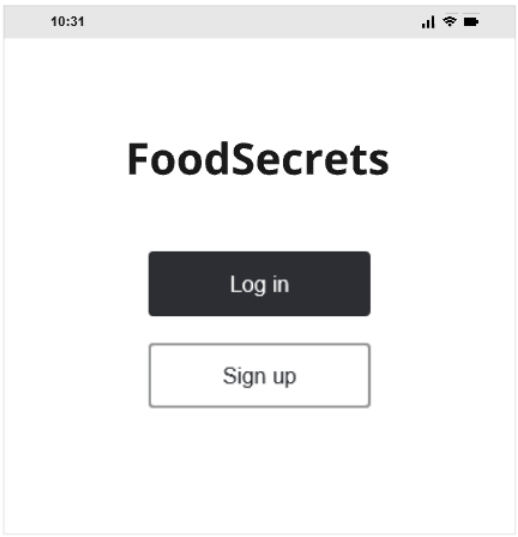

Abbildung 4: Login Seite

Auf der Registrierungsseite muss der Benutzer die Pflichtfelder ausfüllen und kann die Registrierung bestätigen.

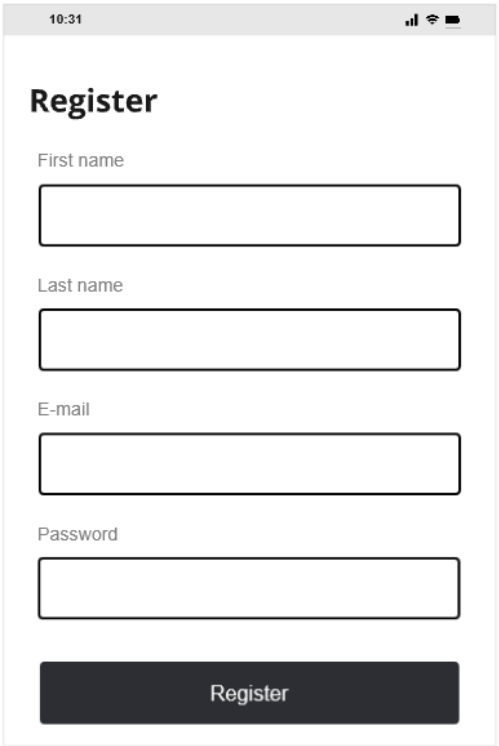

Abbildung 5: Benutzer Registrierung

Die Hauptseite eines angemeldeten Benutzers ist das Logbuch. Dort sind alle erfassten Logeinträge ersichtlich. Auf der linken Seite befindet sich eine Sidebar, welche die Navigation zu weiteren Seiten ermöglicht.

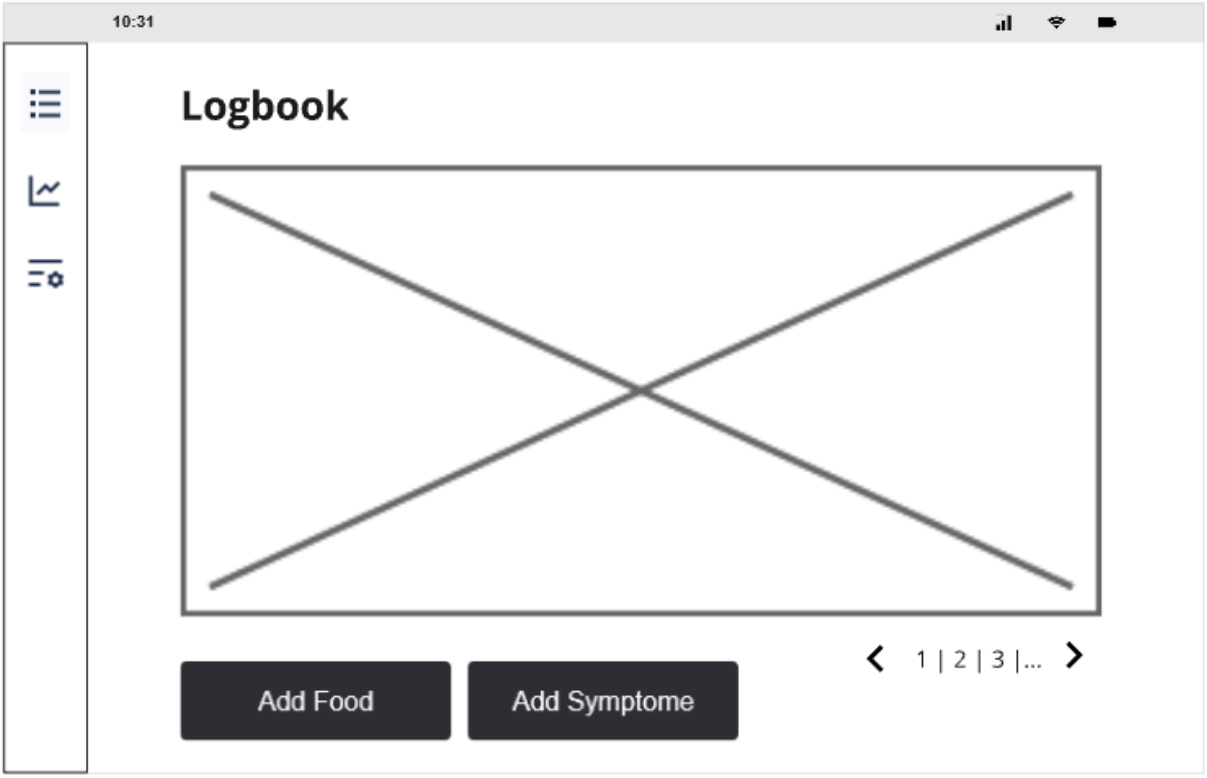

Abbildung 6: Logbuch

Neue Einträge können direkt mittels Anwählen der entsprechenden Schaltfläche erfasst werden. Dabei öffnet sich ein Popup Fenster, welches die Eingaben des Benutzers entgegennimmt und basierend auf bereits vorhandenen Daten Vorschläge in Form von einer Autovervollständigung macht.

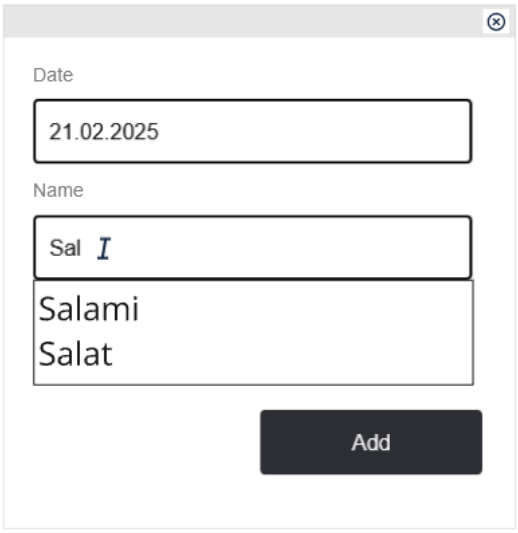

Abbildung 7: Logeintrag erfassen

Die Autovervollständigung basiert auf bereits zuvor erfasste Einträge. Im Falle, dass ein Lebensmittel mit einem Tippfehler erfasst wurde, wird dies zukünftig auch so in der Autovervollständigung vorgeschlagen. Um dieses unerwünschte Verhalten zu beeinflussen, muss der Fehlerhafte Eintrag korrigiert werden.
In der Mobile Ansicht der Hauptseite entfällt die Sidebar. Die Navigation zu weiteren Seiten erfolgt über ein Menüpunkt in der oberen rechten Seite. Logeinträge werden in der Mobile Ansicht durch Scrollen automatisch nachgeladen, sodass der Benutzer keine spezifische Seitenzahl anwählen muss, um ältere Einträge zu sichten.

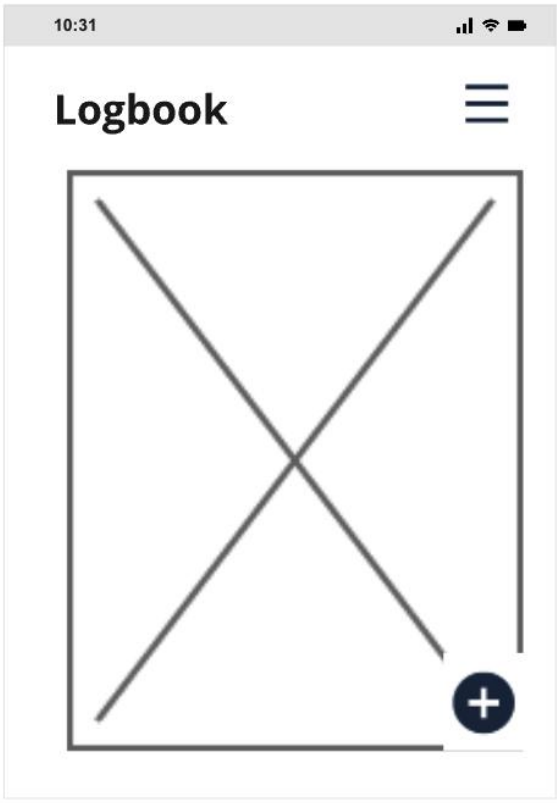

Abbildung 8: Logeintrag erfassen

Aus Platzgründen wird für das Erfassen von Lebensmittel und Symptomen keine dedizierte Schaltfläche vorgesehen. Beim Anwählen der «Plus» Schaltfläche kann der Benutzer Auswählen, ob ein Lebensmittel oder ein Symptom erfasst werden soll.

Abbildung 9: Logbuch - Einträge erfassen in der Mobile Ansicht

Auf der Analyseseite kann der Benutzer optional einen gewünschten Zeitraum angeben sowie bei Bedarf Lebensmittel Ignorieren. Eine zentrale Tabelle zeigt die Lebensmittel in einer Spalte und die Symptome in den anderen Spalten. Jede Zelle der Tabelle könnte einen Korrelationswert anzeigen, der angibt, wie stark das Lebensmittel mit dem jeweiligen Symptom korreliert. Werte könnten von -1 (starke negative Korrelation) bis +1 (starke positive Korrelation) reichen.

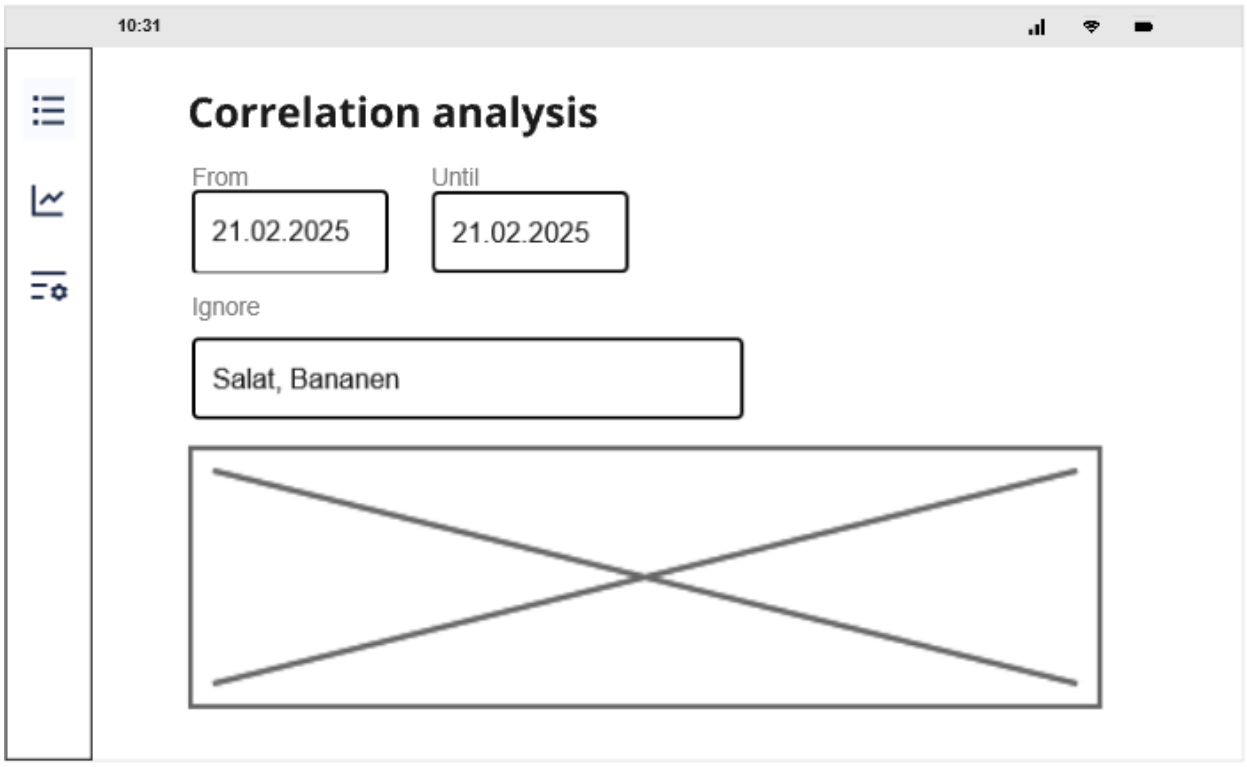

Abbildung 10: Analyseseite

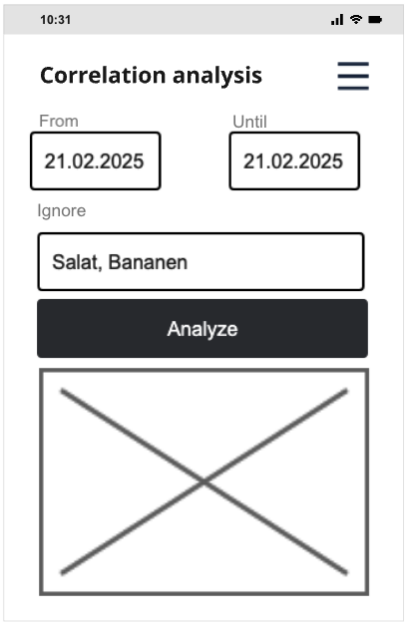

Abbildung 11: Analyseseite – Mobile Ansicht

3 Anforderungen

Das Backlog wird in funktionale und nicht-funktionale Anforderungen unterteilt. Zudem erfolgt eine Kategorisierung basierend auf dem Umsetzungsbereich (Frontend / Backend). Anhand der ID lässt sich erkennen, in welche Kategorie eine Anforderung eingeordnet wurde:

ID Präfix	**Bedeutung**
FABE	Funktionale Anforderung Backend
FAFE	Funktionale Anforderung Frontend
NFABE	Nicht-funktionale Anforderung Backend
NFAFE	Nicht-funktionale Anforderung Frontend

Tabelle 1: Gruppierung der Anfoderungen

Die Priorisierung der Einträge erfolgt nach dem MoSCoW Schema. Die Einträge werden folgendermassen klassifiziert:

- MUST (unbedingt für die Umsetzung erforderlich)
- SHOULD (Umsetzen, wenn alle Must have’s erledigt sind)
- COULD (kann umgesetzt werden, wenn höherwertige Anforderungen nicht beeinträchtigt werden)
- WON’T (wird nicht umgesetzt, evtl. für spätere Umsetzungen interessant)

3.1 FAFE-001: Benutzerregistrierung

- **Ziel**: Ein Benutzer kann sich selbst über eine Registrierungsmaske in der Applikation registrieren.
- **Ereignis**: Der Benutzer hat ihre Daten in die Registrierungsmaske eingegeben und klickt auf den Button "Registrieren".
- **Vorbedingung**:
    - Der Benutzer befindet sich auf der Registrierungsseite.
    - Der Benutzer hat alle erforderlichen Felder (z. B. E-Mail, Passwort, Bestätigung des Passworts) ausgefüllt.
    - Die Passwortlänge beträgt mindestens 8 Zeichen und enthält mindestens eine Zahl und ein Sonderzeichen.
- **Standardablauf**:
    1. Der Benutzer besucht die Registrierungsseite.
    2. Der Benutzer füllt die erforderlichen Felder (E-Mail, Passwort, Passwortbestätigung) aus.
    3. Der Benutzer betätigt den "Registrieren" Button.
    4. Die Registrierung wird an das Backend geschickt und die Daten werden persistiert [FABE-15] (Verweis auf andere Anforderung).
    5. Der Benutzer erhält eine Erfolgsmeldung, dass die Registrierung erfolgreich war.
- **Alternativablauf**:
    - Der Benutzer erhält eine entsprechende Fehlermeldung, wenn eine E-Mail bereits registriert ist.
    - Der Benutzer erhält eine entsprechende Fehlermeldung, wenn die Passwortanforderungen nicht erfüllt sind.
- **Nachbedingungen Erfolg**:
    - Der Benutzer erhält eine Erfolgsmeldung.
    - Der Benutzer wird zur Login-Seite weitergeleitet.
- **Nachbedingung Fehler**:
    - Der Benutzer erhält eine Fehlermeldung.
    - Die eingegebenen Daten bleiben im Registrierungsformular bestehen.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

3.2 FAFE-002: Benutzeranmeldung

- **Ziel**: Nur authentifizierte Benutzer können die Applikation nutzen.
- **Ereignis**: Der Benutzer versucht, auf die Applikation zuzugreifen.
- **Vorbedingung**:
    - Der Benutzer hat ein Benutzerkonto erstellt. **[FAFE-001]**
    - Der Benutzer befindet sich auf der Anmeldeseite der Applikation.
- **Standardablauf**:
    1. Der Benutzer besucht die Anmeldeseite der Applikation.
    2. Der Benutzer gibt ihren Benutzernamen und ihr Passwort in die entsprechenden Felder ein.
    3. Der Benutzer klickt auf den "Anmelden" Button.
    4. Der Identity Provider überprüft die Anmeldedaten.
    5. Bei erfolgreicher Authentifizierung wird der Benutzer zur Hauptseite der Applikation weitergeleitet.
- **Alternativablauf**:
    - Wenn die Anmeldedaten nicht korrekt sind, erhält der Benutzer eine Fehlermeldung, dass Benutzername oder Passwort falsch sind.
- **Nachbedingungen Erfolg**:
    - Der Benutzer hat Zugang zur Applikation und kann alle Funktionen nutzen.
- **Nachbedingung Fehler**:
    - Bei fehlerhaften Anmeldedaten bleibt der Benutzer auf der Anmeldeseite und kann es erneut versuchen.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

3.3 FAFE-003: Logeinträge erfassen

- **Ziel:** Ein Benutzer kann Logeinträge für die zu sich genommenen Lebensmittel oder für die aufgetretenen Symptomen erfassen.
- **Ereignis**: Der Benutzer hat die Details zu einem Lebensmittel/Symptom eingegeben und klickt auf den Button "Eintrag speichern".
- **Vorbedingung**:
    - Der Benutzer befindet sich auf der Seite zum Erfassen von Logeinträgen.
    - Der Benutzer hat die erforderlichen Felder (z. B. Lebensmittelname/Symptomname, Datum) ausgefüllt.
- **Standardablauf**:
    1. Der Benutzer besucht die Seite für Logeinträge.
    2. Der Benutzer wählt, ob ein Lebensmittel oder ein Symptom erfasst werden soll
    3. Der Benutzer füllt die erforderlichen Felder (Lebensmittelname/Symptomname, Datum) aus.
    4. Der Benutzer betätigt den "Eintrag speichern" Button.
    5. Der Logeintrag wird an das Backend geschickt und persistiert **[FABE-001]**
    6. Der Benutzer erhält eine Erfolgsmeldung, dass der Logeintrag gespeichert wurde.
- **Alternativablauf**:
    - Der Benutzer erhält eine entsprechende Fehlermeldung, wenn eines der erforderlichen Felder leer ist.
- **Nachbedingungen Erfolg**:
    - Der Benutzer erhält eine Erfolgsmeldung.
    - Der Logeintrag wird in der Übersicht der Logeinträge angezeigt.
- **Nachbedingung Fehler**:
    - Der Benutzer erhält eine Fehlermeldung.
    - Die eingegebenen Daten bleiben im Formular bestehen, damit der Benutzer Eingaben korrigieren kann.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

3.4 FAFE-004: Autovervollständigung

- **Ziel**: Aufgrund bereits erfassten Logeinträgen, sollen bereits bekannte Lebensmittel/Symptome bei der Eingabe als Auto-Vervollständigung angeboten werden.
- **Ereignis**: Der Benutzer erfasst einen Logeintrag **[FAFE-003],** bei der Eingabe des Lebensmittel- oder Symptomnamen werden Vorschläge zur Autovervollständigung vorgeschlagen.
- **Vorbedingung**:
    - Der Benutzer befindet sich auf der Seite zur Eingabe von Logeinträgen.
    - Der Benutzer hat bereits mindestens ein Logeintrag.
- **Standardablauf**:
    1. Der Benutzer besucht die Seite zur Eingabe von Logeinträgen.
    2. Der Benutzer beginnt, den Namen eines Lebensmittels/Symptoms in das Eingabefeld einzugeben.
    3. Das System schlägt automatisch Lebensmittel/Symptom vor, die mit der Eingabe übereinstimmen. **[FABE-005]**
    4. Der Benutzer kann ein vorgeschlagenes Lebensmittel/Symptom auswählen oder die Eingabe fortsetzen.
    5. Der Benutzer speichert den Logeintrag. **[FABE-001]**
- **Alternativablauf**:
    - Wenn keine übereinstimmen gefunden wurde oder noch kein Logeinträge vorhanden sind, erhält der Benutzer keinen Vorschlag für die Autovervollständigung. Der Benutzer gibt den ganzen Lebensmittel- oder Symptomnamen ein und speichert den Logeintrag.
- **Nachbedingungen Erfolg**:
    - Der Benutzer erhält Vorschläge, basierend auf bereits vorhandenen Daten.
- **Nachbedingung Fehler**:
    - Wenn die Auto-Vervollständigung nicht funktioniert, bleibt das Eingabefeld unverändert und er Benutzer erhält keine Vorschläge.
- **Klassifizierung**: Funktional, SHOULD

 Aufwand: MITTEL

3.5 FAFE-005: Anzeige Logbuch

- **Ziel**: Ein Benutzer sieht auf der Hauptseite all seine Logeinträge.
- **Ereignis**: Der Benutzer hat sich erfolgreich angemeldet und besucht die Hauptseite der Applikation. Auf der Hauptseite werden dem Benutzer seine bereits erfassten Einträgen angezeigt.
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat Logeinträge erstellt.
    - Der Benutzer befindet sich auf der Hauptseite der Applikation.
- **Standardablauf**:
    1. Der Benutzer besucht die Hauptseite der Applikation.
    2. Das System lädt alle Logeinträge der benutzenden Person. **[FABE-002]**
    3. Der Benutzer sieht eine übersichtliche Liste aller Logeinträge, sortiert nach Datum.
    4. Jeder Logeintrag zeigt relevante Informationen an (z. B. Lebensmittelname, Datum, Uhrzeit).
    5. Der Benutzer hat die Möglichkeit, jeden Logeintrag auszuwählen, um ihn zu bearbeiten. **[FAFE-006] [FABE-003]**
- **Alternativablauf**:
    - Wenn der Benutzer noch keine Logeinträge erstellt hat, wird eine entsprechende Nachricht angezeigt, dass keine Logeinträge vorhanden sind.
    - Wenn das Laden der Logeinträge fehlschlägt, erhält der Benutzer eine Fehlermeldung, und die Seite bietet die Möglichkeit, es erneut zu versuchen.
- **Nachbedingungen Erfolg**:
    - Der Benutzer sieht alle ihre Logeinträge auf der Hauptseite.
    - Der Benutzer kann die Logeinträge jederzeit verwalten.
- **Nachbedingung Fehler**:
    - Bei einem Fehler beim Laden der Logeinträge bleibt die Hauptseite unverändert.
    - Der Benutzer erhält eine Fehlermeldung, die den Grund des Problems erläutert.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

3.6 FAFE-006: Logeinträge bearbeiten und löschen

- **Ziel**: Ein Benutzer kann erfasste Logeinträge bearbeiten oder löschen.
- **Ereignis**: Der Benutzer möchte einen bestehenden Logeintrag ändern oder entfernen.
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat bereits Logeinträge erstellt.
    - Der Benutzer befindet sich auf der Seite, die ihre Logeinträge anzeigt.
- **Standardablauf**:
    1. Der Benutzer besucht die Seite mit seinen Logeinträgen.
    2. Der Benutzer wählt den Logeintrag aus, den er bearbeiten oder löschen möchte.
    3. **Bearbeiten**:
        - Der Benutzer klickt auf den "Bearbeiten" Button neben dem gewählten Logeintrag.
        - Die Applikation öffnet ein Formular mit den aktuellen Daten des Logeintrags.
        - Der Benutzer nimmt die gewünschten Änderungen vor und klickt auf den "Speichern" Button.
        - Die Applikation aktualisiert den Logeintrag und zeigt eine Erfolgsmeldung an.
    4. **Löschen**:
        - Der Benutzer klickt auf den "Löschen" Button neben dem gewählten Logeintrag.
        - Die Applikation zeigt eine Bestätigungsaufforderung an
        - Der Benutzer bestätigt die Löschung.
        - Der Eintrag wird gelöscht. Eine Erfolgsmeldung wird angezeigt. **[FABE-004]**
- **Alternativablauf**:
    - Wenn der Benutzer die Bearbeitung abbricht, bleibt der Logeintrag unverändert.
    - Wenn der Benutzer ungültige Daten eingibt, erhält sie eine Fehlermeldung und wird aufgefordert, gültige Daten einzugeben.
    - Wenn der ausgewählte Logeintrag nicht mehr vorhanden ist, erhält der Benutzer eine Fehlermeldung.
- **Nachbedingungen Erfolg**:
    - Bei einer Löschung wird der Logeintrag nicht mehr angezeigt.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während des Speicherns oder Löschens bleibt die Seite unverändert.
    - Der Benutzer erhält eine Fehlermeldung, die den Grund des Problems erläutert.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

3.7 FAFE-007: Auswertungen zu Lebensmittel-Symptom-Korrelationen

- **Ziel**: Ein Benutzer kann auf einer weiteren Seite Auswertungen zu den Korrelationen von Lebensmitteln und Symptomen auf Basis der erfassten Daten sehen.
- **Ereignis**: Der Benutzer navigiert zur Seite für Auswertungen um Korrelationen anzuzeigen.
- **Vorbedingung**:
    - Der Benutzer hat Logeinträge zu Lebensmitteln und Symptomen erstellt.
    - Der Benutzer befindet sich auf der Seite für Auswertungen.
- **Standardablauf**:
    1. Der Benutzer besucht die Seite für Auswertungen.
    2. Die Applikation lädt die erfassten Daten zu Lebensmitteln und Symptomen.
    3. Der Benutzer sieht eine Übersicht der Korrelationen zwischen Lebensmitteln und Symptomen.
    4. Der Benutzer kann Parameter auswählen, um die Analyse anzupassen:
        - Der Benutzer gibt einen bestimmten Zeitraum an, innerhalb dessen die Daten analysiert werden sollen.
        - Der Benutzer kann Symptome oder Lebensmittel auswählen, die in der Analyse nicht berücksichtigt werden sollen.
    5. Nach der Anpassung der Parameter klickt der Benutzer auf den "Analysieren" Button.
    6. Das System zeigt die angepasste Analyse der Korrelationen basierend auf den gewählten Parametern an. **[FABE-006]**
- **Alternativablauf**:
    - Wenn keine erfassten Daten vorhanden sind, zeigt das System eine entsprechende Nachricht an, dass keine Auswertungen verfügbar sind.
    - Wenn ungenügend Daten vorhanden sind oder keine Korrelationen gefunden werden, erhält der Benutzer eine entsprechende Meldung.
- **Nachbedingungen Erfolg**:
    - Der Benutzer sieht die angepasste Analyse der Korrelationen zwischen Lebensmitteln und Symptomen auf der Seite.
- **Nachbedingung Fehler**:
- Bei einem Fehler beim Laden der Daten bleibt die Seite unverändert.
- Der Benutzer erhält eine Fehlermeldung, die den Grund des Problems erläutert.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: HOCH

3.8 FABE-001: Neuen Logeintrag erfassen

- **Ziel**: Ein Benutzer kann über die API neue Logeinträge erfassen.
- **Ereignis**: Der Benutzer möchte einen neuen Logeintrag erstellen.
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat Zugang zur API.
    - Die erforderlichen Daten für den Logeintrag sind vorhanden (z. B. Lebensmittelname, Symptom, Datum).
- **Standardablauf**:
    1. Der Benutzer sendet eine POST-Anfrage an den API-Endpunkt mit den erforderlichen Daten im Request-Body.
    2. Die API empfängt die Anfrage und validiert die Daten.
    3. Wenn die Daten gültig sind, wird der neue Logeintrag in der Datenbank gespeichert.
    4. Die API gibt eine Erfolgsmeldung zurück, einschließlich der ID des neu erstellten Logeintrags.
    5. Der Benutzer erhält eine Bestätigung, dass der Logeintrag erfolgreich erfasst wurde.
- **Alternativablauf**:
    - Wenn die gesendeten Daten ungültig sind (z. B. fehlende Pflichtfelder), gibt die API einen HTTP Status Code 400 (Bad Request) mit den Gründen für den Validierungsfehler zurück.
- **Nachbedingungen Erfolg**:
    - Der neue Logeintrag ist in der Datenbank gespeichert und kann von der benutzenden Person eingesehen werden.
    - Die API gibt die Details des erstellten Logeintrags in der Antwort zurück.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während des Speicherns bleibt der Logeintrag unverändert.
    - Der Benutzer erhält eine Fehlermeldung, die den Grund des Problems erläutert.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

3.9 FABE-002: Alle Logeinträge abrufen

- **Ziel**: Ein Benutzer kann über die API alle seine Logeinträge abrufen.
- **Ereignis**: Der Benutzer erhält eine Liste all seiner Logeinträge.
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat Zugang zur API.
- **Standardablauf**:
    1. Der Benutzer sendet eine GET-Anfrage an den API-Endpunkt.
    2. Die API lädt alle Logeinträge der benutzenden Person aus der Datenbank.
    3. Die API gibt eine JSON-Antwort zurück, die alle Logeinträge der benutzenden Person enthält. (Pagination)
    4. Der Benutzer erhält die Antwort der API mit den Logeinträgen.
- **Alternativablauf**:
    - Wenn für der Benutzer keine Logeinträge in der Datenbank gespeichert sind, gibt die API eine leere Liste zurück.
- **Nachbedingungen Erfolg**:
    - Der Benutzer erhält eine JSON-Antwort mit allen ihren Logeinträgen.
    - Der Benutzer kann die Logeinträge in ihrer Anwendung verarbeiten und anzeigen.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während des Ladens der Logeinträge gibt die API eine Fehlermeldung zurück.
    - Der Benutzer erhält eine Fehlermeldung, die den Grund des Problems erläutert.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

3.10 FABE-003: Logeintrag bearbeiten

- **Ziel**: Ein Benutzer kann über die API bestehende Logeinträge bearbeiten.
- **Ereignis**: Logeintrag wurde anhand der Eingaben des Benutzers aktualisiert.
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat Zugang zur API.
    - Der Benutzer hat bereits einen Logeintrag erfasst.
- **Standardablauf**:
    1. Der Benutzer sendet eine PUT-Anfrage an den API-Endpunkt. Im Pfad ist die ID des zu bearbeitenden Logeintrags enthalten.
    2. Die API validiert die gesendeten Daten im Anfrage-Body.
    3. Wenn die Daten gültig sind, wird der Logeintrag in der Datenbank aktualisiert.
    4. Die API gibt eine Erfolgsmeldung zurück
    5. Der Benutzer erhält eine Bestätigung, dass der Logeintrag erfolgreich bearbeitet wurde.
- **Alternativablauf**:
    - **Ungültige Daten**: Wenn die gesendeten Daten ungültig sind (z. B. fehlende Pflichtfelder), gibt die API einen HTTP Status Code 400 (Bad Request) mit den Gründen für den Validierungsfehler zurück.
    - **Logeintrag nicht gefunden**: Wenn der angegebene Logeintrag nicht existiert, gibt die API einen HTTP Status Code 404 zurück.
- **Nachbedingungen Erfolg**:
    - Der Logeintrag ist in der Datenbank aktualisiert und kann von der benutzenden Person eingesehen werden.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während des Aktualisierens bleibt der Logeintrag unverändert.
    - Der Benutzer erhält eine Fehlermeldung, die den Grund des Problems erläutert.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: GERING

3.11 FABE-004: Logeintrag löschen

- **Ziel**: Ein Benutzer kann über die API bestehende Logeinträge löschen.
- **Ereignis**: Logeintrag wurde in der Datenbank entfernt
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat Zugang zur API.
    - Der Benutzer hat bereits Logeinträge erstellt, die gelöscht werden können.
- **Standardablauf**:
    1. Der Benutzer sendet eine DELETE-Anfrage an den API-Endpunkt. Im Pfad ist die ID des zu bearbeitenden Logeintrags enthalten.
    2. Die API sucht den Logeintrag in der Datenbank anhand der übergebenen ID.
    3. Wenn der Logeintrag gefunden wird, wird er aus der Datenbank gelöscht.
    4. Die API gibt eine Erfolgsmeldung zurück, dass der Logeintrag erfolgreich gelöscht wurde.
    5. Der Benutzer erhält eine Bestätigung, dass der Logeintrag gelöscht wurde.
- **Alternativablauf**:
    - **Logeintrag nicht vorhanden**: Wenn der angegebene Logeintrag nicht existiert, gibt die API eine Erfolgsmeldung zurück. (Kein HTTP Status Code 404)
- **Nachbedingungen Erfolg**:
    - Der Logeintrag ist aus der Datenbank gelöscht und kann von der benutzenden Person nicht mehr eingesehen werden.
    - Die API gibt eine Bestätigung des Löschvorgangs in der Antwort zurück.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während des Löschens bleibt der Logeintrag unverändert.
    - Der Benutzer erhält eine Fehlermeldung, die den Grund des Problems erläutert.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: GERING

3.12 FABE-005: Autovervollständigung für Lebensmittel und Symptome

- **Ziel**: Ein Benutzer kann über die API Vorschläge für Lebensmittel und Symptome basierend auf seiner aktuellen Eingabe abrufen.
- **Ereignis**: Der Benutzer gibt einen Suchbegriff in das Eingabefeld ein und erhält während der Eingabe Vorschläge für Lebensmittel oder Symptome erhalten.
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat Zugang zur API.
    - Der Benutzer gibt mindestens 1 Zeichen in das Eingabefeld ein.
- **Standardablauf**:
    1. Der Benutzer gibt einen Lebensmittel oder Symptom in das Eingabefeld ein.
    2. Die Anwendung sendet eine GET-Anfrage an den API-Endpunkt. In der Anfrage ist die aktuelle Eingabe des Benutzers enthalten, sowie ob Vorschläge für Lebensmittel oder Symptomen erwartet wird.
    3. Die API empfängt die Anfrage und verarbeitet die eingegebenen Daten.
    4. Die API sucht nach passenden Vorschlägen in der Datenbank.
    5. Die API gibt eine Liste von Vorschlägen zurück, die mit den bisherigen Eingaben des Benutzers übersinstimmen.
    6. Der Benutzer erhält die Vorschläge zur Auswahl.
- **Alternativablauf**:
    - Wenn der Benutzer weniger als 1 Zeichen eingibt, gibt die API eine leere Liste zurück.
    - Wenn keine Übereinstimmungen gefunden werden, gibt die API eine leere Liste zurück.
- **Nachbedingungen Erfolg**:
    - Der Benutzer erhält eine Liste von Vorschlägen, die mit seiner bisherigen Eingabe überbeinstimmen und kann diese auswählen.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während der Anfrage gibt die API eine Fehlermeldung zurück. Der Benutzer erhält keine Fehlermeldung und auch keine Vorschläge die zu Auswahl stehen. Der Benutzer kann mit der Eingabe fortfahren.
- **Klassifizierung**: Funktional, SHOULD
- **Aufwand**: MITTEL

3.13 FABE-006: Korrelationen auswerten

- **Ziel**: Ein Benutzer kann über die API eine Auswertung der Korrelationen basierend auf bestimmten Parametern wie Zeitraum, Symptome oder Lebensmittel anfordern, die von der Analyse ausgeschlossen werden sollen.
- **Ereignis**: Der Benutzer erhält eine Analyse basieren auf den erfassten Daten.
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat Zugang zur API.
    - Der Benutzer gibt die optionalen Parameter (z. B. Zeitraum, zu ignorierenden Symptome oder Lebensmittel) in die Anfrage ein.
- **Standardablauf**:
    1. Der Benutzer sendet eine GET-Anfrage an den API-Endpunkt mit den optionalen Parameter.
    2. Die API empfängt die Anfrage und validiert die eingegebenen Parameter.
    3. Die API führt die Analyse der Korrelationen basierend auf den übergebenen Parametern und den vorhandenen Daten durch.
    4. Die API erstellt eine Auswertung und berechnet die Korrelationen.
    5. Die API gibt die Ergebnisse der Analyse zurück.
    6. Der Benutzer erhält die Auswertung der Korrelationen.
- **Alternativablauf**:
    - Wenn die übergebenen Parameter ungültig sind (z. B. falsches Datumsformat), gibt die API einen HTTP Status Code 400 (Bad Request) mit den Gründen für den Validierungsfehler zurück.
- **Nachbedingungen Erfolg**:
    - Der Benutzer erhält eine detaillierte Auswertung der Korrelationen, anhand der angegebenen Parameter und den vorhandenen Daten.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während der Analyse gibt die API eine Fehlermeldung zurück.
    - Wenn nicht genügend Daten vorhanden sind oder keine Korrelationen gefunden wurden, enthält die Antwort eine leere Auswertung.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: HOCH

3.14 FABE-007: Zugriff auf die API nur für authentifizierte Benutzer

- **Ziel**: Sicherstellen, dass der Zugriff auf die fachliche API nur für authentifizierte Benutzer möglich ist.
- **Ereignis**: Der Benutzer kann die API nur nutzen, wenn er sich zuvor beim IdP registriert und authentifiziert hat.
- **Vorbedingung**:
    - Der Benutzer hat ein gültiges Benutzerkonto beim IdP. **[FAFE-001]**
    - Der Benutzer muss sich erfolgreich authentifizieren, bevor er auf die API zugreifen kann. **[FAFE-002]**
- **Standardablauf**:
    1. Der Benutzer sendet eine Authentifizierungsanfrage an den entsprechenden Endpunkt des IdP mit den erforderlichen Anmeldedaten (z. B. Benutzername und Passwort).
    2. Der IdP überprüft die Anmeldedaten.
    3. Die API gibt ein Token zurück, das der Benutzer für zukünftige Anfragen bei der API verwenden kann.
    4. Der Benutzer sendet eine Anfrage an die fachliche API, z. B. um alle Logeinträge abzurufen.
    5. Die API überprüft das bereitgestellte Token mittels Intorspect beim IdP oder mittels Public Key Validierung.
    6. Wenn das Token gültig ist, verarbeitet die API die Anfrage und gibt die angeforderten Daten zurück.
    7. Der Benutzer erhält die gewünschten Daten.
- **Alternativablauf**:
    - Wenn die Anmeldedaten falsch sind, gibt der IdP eine Fehlermeldung zurück und der Benutzer erhält kein Token.
    - Wenn der Benutzer die API mit einem ungültigen Token aufruft, gibt die API einen HTTP Status Code 401 (Unauthorized) zurück.
- **Nachbedingungen Erfolg**:
    - Der Benutzer erhält Zugriff auf die API und kann Daten abrufen oder ändern, solange das Token gültig ist.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während der Authentifizierung oder beim API-Zugriff erhält der Benutzer eine entsprechende Fehlermeldung.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

3.15 FABE-008: Zugriffsschutz für Benutzerdaten

- **Ziel**: Sicherstellen, dass ein authentifizierter Benutzer über die API nicht auf die Daten eines anderen Benutzers zugreifen kann.
- **Ereignis**: Anfragen eines Benutzers auf eine Ressource eines anderen Benutzers werden von der API abgelehnt.
- **Vorbedingung**:
    - Der Benutzer ist authentifiziert und hat ein gültiges Token.
    - Der Benutzer kennt oder errät die ID eines anderen Benutzers
- **Standardablauf**:
    1. Der Benutzer sendet eine GET-Anfrage an den API-Endpunkt um Logeinträge eines anderen Benutzers abzurufen. Dabei wird die ID eines anderen Benutzers in der Anfrage mitgegeben.
    2. Die API überprüft das bereitgestellte Token und die Authentifizierung des Benutzers.
    3. Die API überprüft, ob das Token für die Benutzer ID ausgestellt wurde, für die die Abfrage getätigt wird.
    4. Wenn die IDs übereinstimmen, gibt die API die angeforderten Daten zurück.
    5. Der Benutzer erhält die angeforderten Daten.
- **Alternativablauf**:
    - Wenn die angeforderte Benutzer-ID nicht mit der ID des authentifizierten Benutzers übereinstimmt, gibt die API einen HTTP Status Code 403 (Forbidden) zurück.
- **Nachbedingungen Erfolg**:
    - Der Benutzer erhält nur Zugriff auf seine eigenen Daten und keine Informationen anderer Benutzer.
- **Nachbedingung Fehler**:
    - Bei einem Fehler während der Überprüfung oder beim Zugriff auf die Daten gibt die API eine Fehlermeldung zurück.
    - Der Benutzer kann nicht auf die Daten eines anderen Benutzers zugreifen.
- **Klassifizierung**: Funktional, MUST
- **Aufwand**: MITTEL

4 Architektur

4.1 Technologie Stack

Gemäß der Aufgabenstellung sind die technologischen Rahmenbedingungen bereits festgelegt. Für das Frontend wird React verwendet, während Django für das Backend vorgesehen ist. Für das Styling im Frontend kommt SASS als Preprocessor zum Einsatz. Zur einfachen Integration von Icons wird die Icon-Bibliothek Remix Icon genutzt. Als Identity Provider (IdP) wird Keycloak in Betracht gezogen, da es über ausgezeichnete Export- und Importfunktionalitäten verfügt, die das lokale Setup für weitere Entwickler erheblich erleichtern würden. Hinsichtlich der Datenbanktechnologie sind sowohl SQL- als auch NoSQL-Datenbanken denkbar.

4.2 Applikationsarchitektur

Im Mittelpunkt jeder Webanwendung steht ein Webservice, der auf eingehende Anfragen mit den entsprechenden Inhalten reagiert. Die Clientanwendung stellt Anfragen an diesen Webservice, der dann die gewünschten Informationen bereitstellt. Die Kommunikation zwischen Client und REST API des Webservice erfolgt mittels HTTP. Die Architektur der Applikation wird durch die Integration eines Identity Providers (IdP) erweitert, der eine zentrale Rolle im Authentifizierungs- und Autorisierungsprozess spielt. Der Einsatz eines IdP ermöglicht es, Benutzeridentitäten sicher zu verwalten und den Zugriff auf die Anwendung zu steuern. Die Kommunikation zwischen der React-Anwendung, dem Django-Backend und dem Identity Provider erfolgt über standardisierte Protokolle wie OAuth 2.0 oder OpenID Connect. Der Benutzer authentifiziert sich beim IdP, der ein Token zurückgibt, das dann für den Zugriff auf die REST API verwendet wird. Diese Architektur ermöglicht eine sichere und skalierbare Lösung für die Benutzerverwaltung.

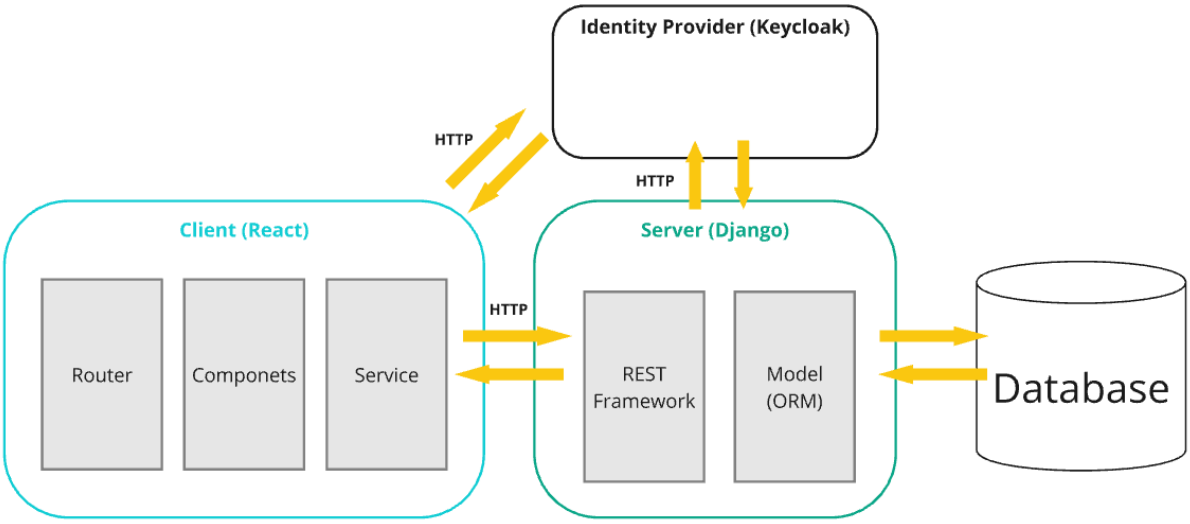

Abbildung 12: Applikationsarchitektur

Die Architektur der Applikation basiert auf einer modernen, komponentenbasierten Struktur, die React für das Frontend und Django für das Backend nutzt. Diese Kombination ermöglicht eine klare Trennung zwischen Benutzeroberfläche und Serverlogik, was die Wartbarkeit und Skalierbarkeit der Anwendung verbessert.

4.2.1 Clientanwendung (Frontend)

Der React Router ermöglicht die Navigation innerhalb der Anwendung, indem er URLs bestimmten Komponenten zuordnet. Dies sorgt für eine nahtlose Benutzererfahrung, da die Benutzer zwischen verschiedenen Ansichten wechseln können, ohne die Seite neu laden zu müssen. Die React Anwendung besteht aus verschiedenen wiederverwendbaren Komponenten, die jeweils spezifische Funktionen oder UI-Elemente darstellen. Diese Komponenten sind modular aufgebaut, was die Entwicklung und das Testen erleichtert. Eine Service-Schicht wird implementiert, um die Kommunikation mit dem Backend über HTTP zu ermöglichen. Diese Services sind verantwortlich für das Senden von Anfragen an die REST API und das Verarbeiten der Antworten, wodurch die Trennung von Logik und Darstellung gewahrt bleibt.

4.2.2 Serveranwendung (Backend)

Das Django REST Framework wird verwendet, um eine RESTful API zu erstellen, die es der Clientanwendung ermöglicht, Daten effizient abzurufen und zu manipulieren. Es bietet Funktionen wie Serialisierung, Authentifizierung und Berechtigungsmanagement, die die Entwicklung der API erheblich erleichtern. Die Datenmodelle in Django definieren die Struktur der Daten, die in der Anwendung verwendet werden. Diese Modelle sind eng mit der Datenbank verbunden und ermöglichen eine einfache Interaktion mit den Daten. Die Datenbank speichert alle relevanten Informationen der Anwendung. Django bietet ein integriertes ORM (Object-Relational Mapping), welches die Interaktion mit der Datenbank vereinfacht und die Datenintegrität gewährleistet. (MCLaughlin, 2025)

5 Implementierung

In diesem Kapitel wird die Implementierung der Web-Applikation detailliert behandelt, wobei die Schwerpunkte auf der Datenbank, der Serveranwendung (Backend) und der Clientanwendung (Frontend) liegen. Um die zuvor definierten Use Cases der Applikation erfolgreich umzusetzen, erfolgt die Implementierung gemäß den festgelegten Anforderungen.

5.1 Datenbank

Zur Datenspeicherung wird die open source Datenbank PostgreSQL verwendet. Hintergrund für diese Entscheidung ist, dass diese Datenbank bereits für die Keycloak Instanz benötigt wird. Dies vereinfacht das finale Deployment und die spätere Wartung im Betrieb, da nur eine Datenbanktechnologie zum Einsatz kommt. Um das lokale Setup zu vereinfachen, wird für die lokale Entwicklung die Default Datenbank von Django (SQLite) verwendet.

5.1.1 Datengrundlage

Um alle Anforderungen der Applikation zu erfüllen, müssen einige Daten in der Datenbank gespeichert werden. Neben den fachlichen Daten wie die Einnahme von Lebensmittel und das Auftreten von Symptomen, benötigt es auch Informationen zu den Usern. Diese Userinformationen sind notwendig, um die fachlichen Daten einem bestimmten User zuzuordnen. Da die gesamte Benutzerverwaltung in Keycloak als Identity Provider ausgelagert wird, benötigt es eine Möglichkeit diese Benutzer zur Serveranwendung zu synchronisieren. Bei der Integration von Keycloak in die Serveranwendung hat sich herausgestellt, dass dies automatisch passiert. Bei den eingehenden Requests erfolgt bei der Validierung der Token auch ein Abgleich mit der Django internen Usertabelle. Ist ein User in dieser Tabelle noch nicht bekannt, wird dieser automatisch angelegt. Dies vereinfacht das Setup enorm, da die Django interne Usertabelle für ein Admin Konto bereits notwendig ist und somit wiederverwendet werden kann.
Bei den fachlichen Daten war initial die Idee, dass die User ihre Lebensmittel und Symptome selber erfassen und verwalten. Vor allem bei den Lebensmitteln würde sich dadurch jedoch eine hohe Redundanz bei den Daten ergeben. Zudem währe es zukünftig schwierig, Korrelation über mehrere User zu ermitteln. Dies wäre z.B. für Studienzwecke spannen, welche mit der Einwilligung der User durchgeführt werden könnte und so nützliche Einblicke für die Forschung geben könnten. Dass die Lebensmitteldatenbank von allen Usern zusammen erfolgreich aufgebaut und gepflegt werden kann ist jedoch sehr unwahrscheinliches. Es wäre beispielsweise schwierig Missbrauch wie absichtliches erfassen von obszönen Wörtern welche dann alle User sehen könnten zu unterbinden.

Etwas Recherche hat ergeben, dass es eine Schweizer Nährwertdatenbank (https://naehrwertdaten.ch/de/) gibt, welche vom Bundesamt für Lebensmittelsicherheit und Veterinärwesen (BLV) öffentlich zur Verfügung gestellt wird. Neben der öffentlich zugänglichen Webseite gibt es auch ein frei zugängliche REST API sowie ein Datensatz als CSV. Der Datensatz enthält über 1000 Einträge und eignet sich ideal als Datengrundlage für die Applikation. Um externe Abhängigkeiten zu minimieren, wir das CSV per Skript in die eigene Datenbank importiert. Der Datensatz der Schweizerischen Nährwertdatenbank ist sehr umfangreich und enthält zu jedem Lebensmittel auch deren Nährwert. Aufgrund des Zeitmangels muss die Applikation etwas einfacher gehalten werden, wodurch die Nährwerte ignoriert werden. Beim Import in die eigene Datenbank werden lediglich die Lebensmittel, deren Kategorie sowie deren Synonym für die verbesserte Suche berücksichtigt.
Eine medizinische Datenbank für die Symptome wurde auch nach längerer Recherche nicht gefunden. Aus diesem Grund wird jeder User seine eigenen Symptome erfassen und verwalten. Da ein User in der Regel an der Korrelation der Lebensmittel zu einem bestimmten Symptom interessiert ist, sollte es bei den Symptomen zu deutlich geringeren Datenmengen als bei Lebensmittel kommen. Redundante Daten, sollten sich dort deshalb im Rahmen halten und sind deshalb vertretbar.

5.1.2 EER-Diagramm

Das Datenbankschema wird anhand eines EER-Diagramms dokumentiert. Vor der Erstellung des EER-Diagramms sollten die zu speichernden Daten normalisiert werden. Dadurch kann sichergestellt werden, dass keine Redundanzen in der Datenbank gespeichert werden.

Da die Usertabelle von Django auch mit externem idP wiederverwendet werden kann, muss diese nicht selbst definiert werden. Diese Tabelle stellt sich wie folgt zusammen:

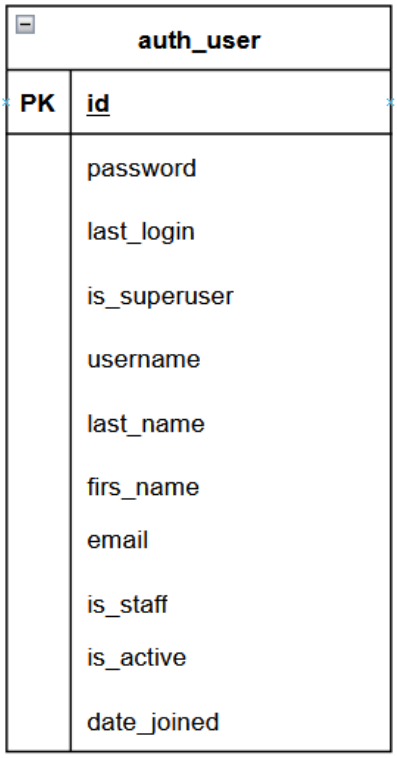

Abbildung 13: Datenbank - Usertabelle

Die Lebensmittel der Schweizerischen Lebensmitteldatenbank werden anhand der Lebensmittelkategorie gruppiert. Ein Lebensmittel kann dabei einer oder mehreren Kategorien zugeordnet werden. Eine Kategorie kann widerum einem oder Mehreren Lebensmittel zugeordnet sein. Das gleiche gilt für die Synonyme. Ein Lebensmittel kann mehreren Synonymen zugeordnet werden und umgekehrt. Daraus ergibt sich folgendes Datenbank Schema.

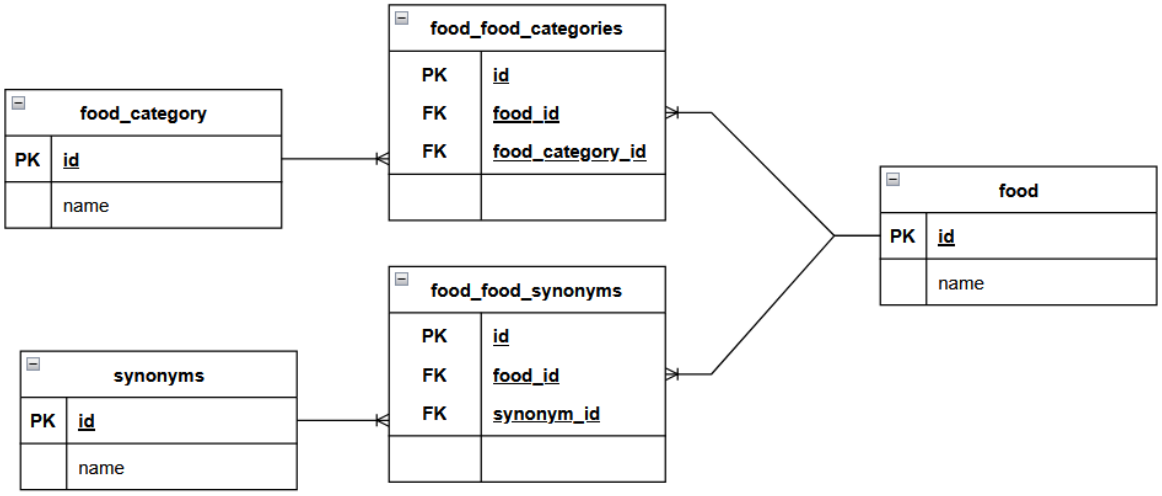

Abbildung 14: Datenbak - Lebensmittel Schema

Die Tabelle der Symptome wird bewusst einfach gehalten. Aus diesem Grund wird für ein Symptom nur dessen Namen in die Datenbank gespeichert.

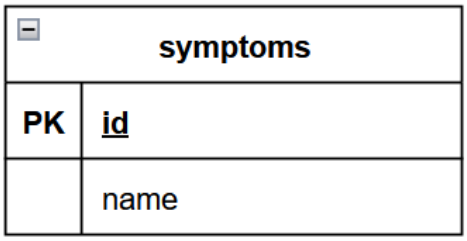

Abbildung 15: Datenbank - Symptomtabelle

Das finale EER-Diagramm befindet ist im Anhang des Dokuments angehängt.

5.2 Serveranwendung

Die Implementierung der Serveranwendung erfolgt gemäss Vorgabe in Python Webframework Django. Die Serveranwendung ist für die fachliche Verarbeitung der Daten zuständig und bietet eine entsprechend REST API für die Interaktion mit der Serveranwendung.

5.2.1 REST API

Wenn zwei Systeme so eng aneinandergekoppelt werden, dass sich diese nicht mehr trennen lassen, entsteht daraus ein grosses, verschmolzenes Gesamtsystem. Solche monolithische Riesensysteme lassen sich oft nur im Ganzen oder gar nicht einsetzen, aktualisieren oder modifizieren. In den meisten Fällen wird deshalb eine modulare Welt aus möglichst unabhängigen Teilsystemen angestrebt, welche über Schnittstellen miteinander kommunizieren. Durch den Einsatz von REST (Representational State Transfer) lässt sich die Koppelung zwischen zwei Systemen nicht vollständig verhindern, jedoch erheblich reduzieren. (Tilkov, 2015)
Die REST API ist für die Clientanwendung die einzige erreichbare Schnittstelle. Drüber werden sowohl daten gelesen wie auch Daten geschrieben.

5.2.1.1 Lebensmittel

Da die Lebensmittel bereits vorerfasst sind, wird über die REST API der Clientanwendung nur lesenden Zugriff auf die Lebensmittel gewährt. Es können alle Lebensmittel sowie ein spezifisches Lebensmittel anhand dessen ID abgerufen werden. Zudem kann mittels Query Parameter nach Lebensmittel gesucht werden. Dies ermöglicht der Clientanwendung eine Autovervollständigung für Lebensmittel anzubieten.

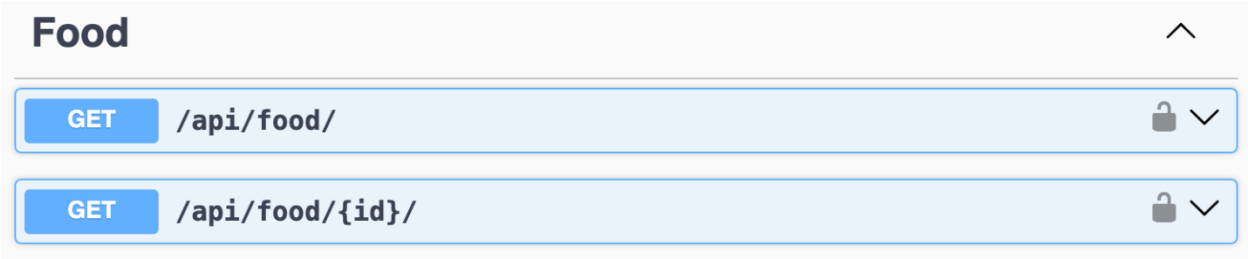

Abbildung 16: API - Lebensmittel

5.2.1.2 Lebensmittel Einnahmen

Zur Verwaltung der Einnahme von Lebensmittel stehen sowohl lesende wie auch schreibende Endpunkte in der API zur Verfügung. Dies ermöglicht den Usern über die Clientanwendung ihre Einnahmen von Lebensmitteln zu erfassen, bearbeiten und löschen.

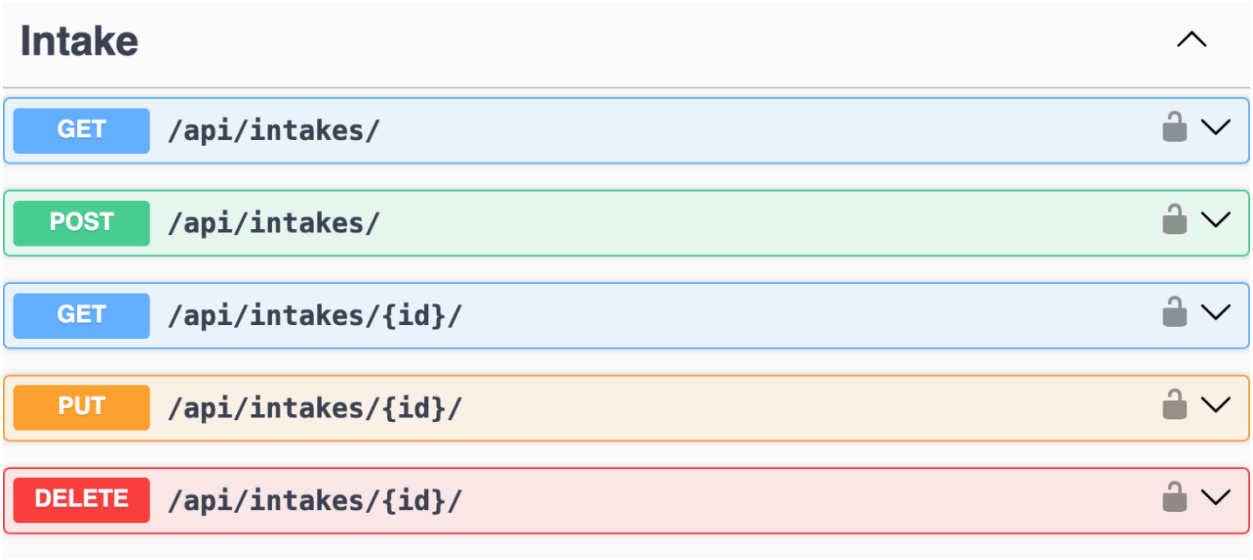

Abbildung 17: API - Lebensmittel Einnahmen

5.2.1.3 Symptome

Da die Symptome nicht vorerfasst sind und von jedem User selbst erfasste werden müssen, bietet die API entsprechende Endpunkte zum Verwalten der Symptome an.

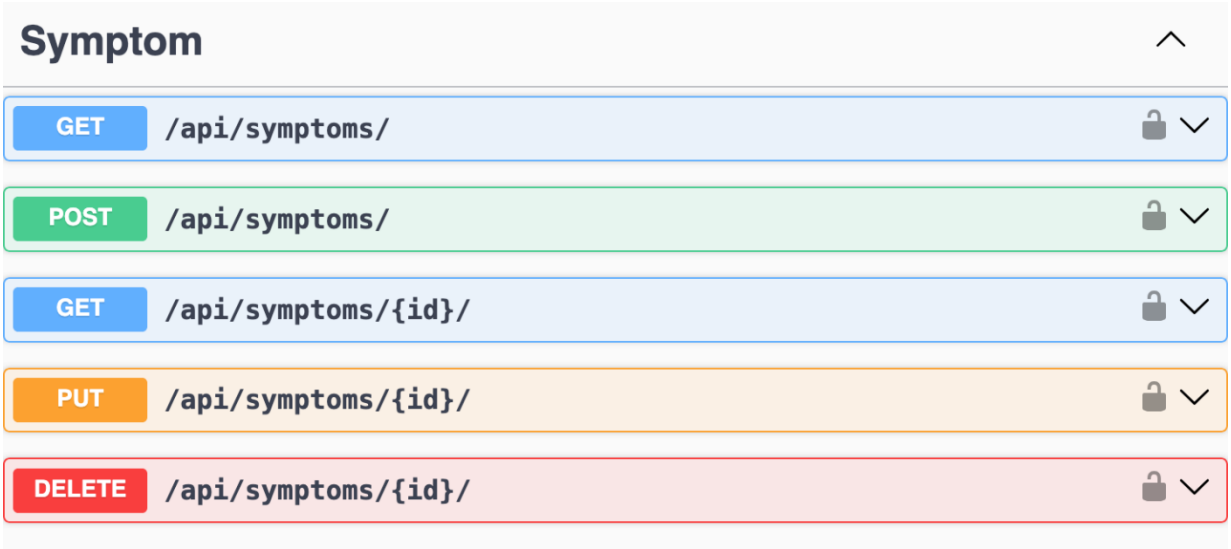

Abbildung 18: API - Symptome

5.2.1.4 Symptom Auftreten

Ist ein Symptom aufgetreten, hat ein User die Möglichkeit dies über die Clientanwendung zu erfassen. Zudem können diese Ereignisse bearbeitet oder gelöscht werden.

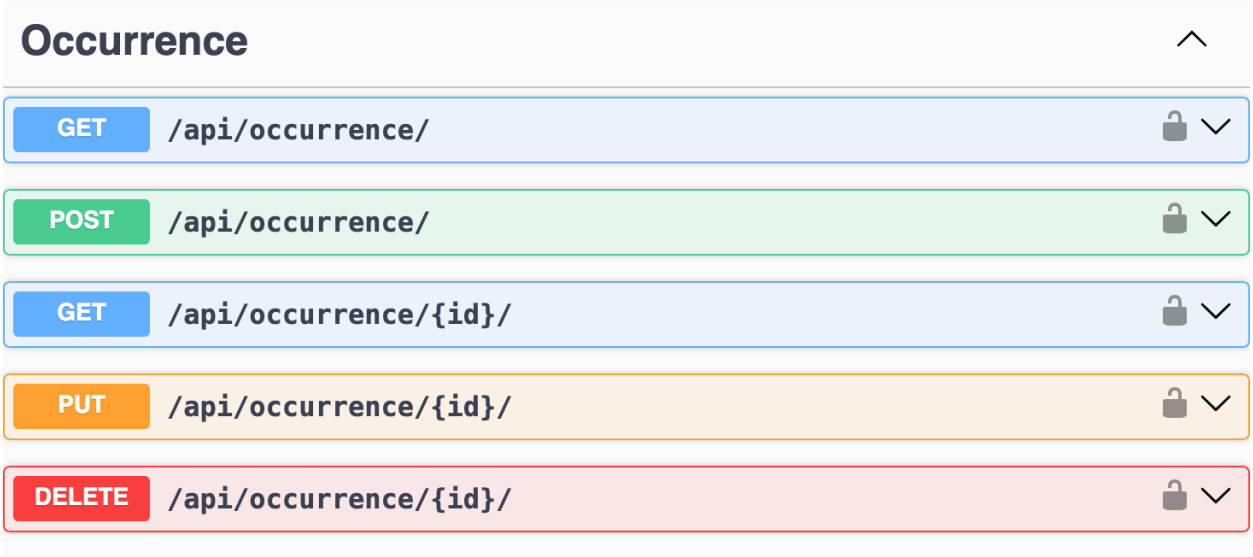

Abbildung 19: API - Symptom Auftreten

5.2.1.5 Logbuch

Die Hauptansicht der Clientanwendung enthält das Logbuch. Dort ist ersichtlich, welche Nahrungsmittel wann eingenommen wurden und wann welches Symptom aufgetreten ist. Zum Abrufen des Logbuchs wird über die API ein einzelner Endpunkt zur Verfügung gestellt. Datengrundlage für diesen Endpunkt sind sowohl die Eingenommenen Lebensmittel wie auch die aufgetretenen Symptome. Durch das zusammenfassen in nur einen Endpunkt verlagert sich diese Logik in die Serveranwendung. Dies vereinfacht nicht nur die Clientanwendung, sondern verbessert auch deren Performance, da die Clientanwendung für die Hauptansicht nicht mehrere Anfragen an den Server senden muss. Der Enpunkt des Logbuchs bietet zudem eine Paginierung, was das Gezielte aufteilen der Daten auf mehrere Seiten ermöglicht.

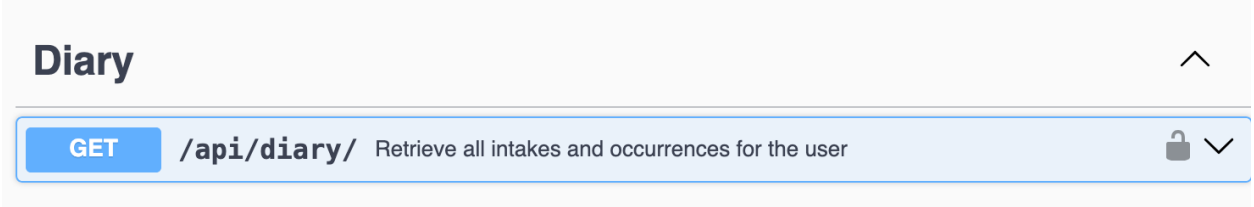

Abbildung 20: API – Logbuch

6 Logbuch

6.1 Projektinitialisierung

Die Initialisierung des Projekts erfolgt bis zum Meilenstein 1. Dabei wurden die Anforderungen sowie die Zielarchitektur der Applikation definiert und in der Projektdokumentation festgehalten. Zudem wurden das GitLab Projekt erstellt und die Anforderungen dort in Form von Issues erfasst.

Datum	Tasks
08.02.2025	Brainstorming & Projekt Pitch
13.02.2025	Sparringpartner anschreiben
14.02.2025	Aufgabenstellung studieren und Applikationsidee vertiefen, Dokumentstruktur erstellen
15.02.2025	Ausgangslage in der Projektdokumentation dokumentieren
16.02.2025	Erfassen erster Anforderungen
17.02.2025	Gitlab Projekt anlegen, Abschnitt zur Vorgehensweise im Kapitel Ausgangslage ergänzt
18.02.2025	Verfassen weiterer Anforderungen, Review aktueller Stand von Sparringpartner, Übermittlung aktueller Stand an Sparringpartner zur Sichtung
21.02.2025	Anforderungen verfassen, Wireframes erstellen, Voranalyse Integration IdP
22.02.2025	Zielarchitektur Dokumentieren
23.02.2025	Dokumentation überarbeiten, GitLab Issues erfassen, Übermittlung aktueller Version an Sparringpartner zur Sichtung
28.02.2025	Wireframes ergänzen, Dokumentation finalisieren, Logbuch nachtragen, Abgabe Projektdokumentation (M1)

Tabelle 2: Logbuch – Projekt Initialisierung

6.2 Sprint 1

Ziel des ersten Sprints war das Erstellen der Projektstruktur. Aufgrund eines defekten Laptops musste die Projektdokumentation auf Basis der Abgabe des Meilensteins 1 wiederhergestellt werden. Aus diesem Grund konnte nicht der gewünschte Aufwand in den Aufbau der Projektstruktur investiert werden

Datum	Tasks
07.03.2025	Grundstruktur Clientanwendung erstellt, Grundfunktionalitäten für SASS implementiert
11.03.2025	Besprechung Meilenstein 1, Wiederherstellen Projektdokumentation
12.03.2025	Keycloak als idP integrieren
13.03.2025	Keycloak als Docker Compose bereitstellen
14.03.2025	Clientanwendung im Docker Compose integrieren
15.03.2025	Vereinfachung der Konfiguration von Umgebungsvariablen für die Clientanwendung

Tabelle 3: Logbuch – Sprint 1

6.3 Sprint 2

Ziel des zweiten Sprints war die Integration der Serveranwendung in die Projektstruktur sowie die Integration des idP in der Serveranwendung. Zudem stand das Docker Compose Deployment aller Komponenten im Fokus. Beim Aufsetzen der Serveranwendung sind einige Probleme aufgetreten. Grund dafür waren die geringen Kenntnisse des Web-Frameworks Django. Auch erwies sich die Konfiguration des Django Rest Frameworks nicht als intuitiv. Vor allem bei der Integration des Identity Providers Keycloak mussten einige Bibliotheken ausprobiert werden. Auch eine eigene Implementation der Authentifizierung führte nicht direkt zum gewünschten Resultat. Obwohl die Integration eines idP eine Standardanforderung bei einem Web-Framework ist, erschien dies bei Django eher ein Umweg zu sein.

Datum	Tasks
16.03.2025	Anpassung Keycloak Konfiguration, README Anleitung ergänzen
20.03.2025	Recherche Custom Keycloak Theming
22.03.2025	Erstellung Custom Keycloak Theme
23.03.2025	Erstellung Grundstruktur Serveranwendung
24.03.2025	Erstellung Grundstruktur Serveranwendung
25.03.2025	Integration der Server Anwendung im Docker Compose Deployment
28.03.2025	Migration der Serveranwendung auf PostgreSQL Datenbank, Anapssung Deployment, Aktuallisierung der README Anleitung, Erstellung PlantUML
29.03.2025	API-Authentifizierung und Aufrufen der API durch die Clientanwendung, Styling anpassen, Docker Compose Deployment und Anleitungen finalisieren, Abgabe Meilenstein 2

Tabelle 4: Logbuch – Sprint 2

6.4 Sprint 3

Ziel des dritten Sprints war es, die Daten der Schweizerischen Lebensmitteldatenbank automatisiert in die Datenbank zu importieren sowie erste fachliche Endpunkte bei der Serveranwendung zu implementieren. Leider konnte aus zeitlichen Gründen das Sprintziel nicht erreicht werden.

Datum	Tasks
29.03.2025	Authentifizierung API, Erweiterung Readme, Recherche Lebensmitteldatenbank
30.03.2025	Spezifikation DB Schema für Lebensmitteldatenbank

Tabelle 5: Sprint 3

6.5 Sprint 4

Der vierte Sprint ist der Längste Sprint im Projekt. Zum einen sollte ein grossteil der fachlichen API sowie die Datenhaltung implementiert sein, sowie ein Grossteil der Clientanwendung umgesetzt sein. Im vierten Sprint konnte einiges an Zeit gutgemacht werden und grosse Fortschritte erreicht werden.

Datum	Tasks
08.04.2025	Lebensmittel Kategorien automatisiert in die Datenbank anlegen
11.04.2025	Restliche Daten der Lebensmitteldatenbank automatisiert in die Datenbank anlegen
14.04.2025	Support für Windows Deployment verbessern, Implementierung Food API
15.04.2025	API Authentifizierung vereinfachen, OpenAPI Spezifikaiton hinzufügen (Swagger)
16.04.2025	API für Lebensmitte Einnahmen
17.04.2025	Finalisierung der API für Lebensmittel Einnahmen, Testen der API
18.04.2025	API fürs Erstellen von Symptomen, API fürs Erfassen von aufgetretenen Symptomen, Testen der API
19.04.2025	Implementierung und Testen des Pagination Patterns, DRF UI deaktivieren im Produktiven Deployment
20.04.2025	Automatisiertes Hinzufügen von Demodaten beim Demostrationsdeployment, Testen der unteschiedlichen Deployment (Lokal, Docker, Docker Compose), Wrapper für UI Logs implementiert
22.04.2025	UI Layout verbessern
24.04.2025	Sidebar und Header Navigation implementiert

25.04.2025	Diverse UI Komponenten implementiert, Pagination Pattern im UI implementiert
26.04.2025	Footer implementiert, Layout optimiert, Diverse Styling Anpassungen
28.04.2025	Styling Anpassungen
29.04.2025	Styling Anpassungen
30.04.2025	Exception Banner implementiert, Keycloak Theming angepasst
01.05.2025	Autocomplete implementiert, Hinzufügen von Lebensmittel und Symptome über die Clientanwendung
03.05.2025	Dokumentation ergänzt

Tabelle 6: Sprint4

Abkürzungsverzeichnis

REST	Representational State Transnfer
API	Application Programming Interface
URL	Uniform Resource Locator
idP	Identity Provider

Literaturverzeichnis

FFHS. (2025). *https://www.ffhs.ch*. Von https://www.ffhs.ch/de/weiterbildung/cas-full-stack-development abgerufen

MCLaughlin. (Februar 2025). *opensource.com*. Von https://opensource.com/article/17/11/django-orm. abgerufen

Tilkov, S. (2015). *REST und HTTP - Entwicklung und Integration nach dem Architekturstil des Web.* Heidelbert: dpunkt.verlag GmbH.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

7 EER-Diagramm