DOKUMENTATION DER STUDIENARBEIT

NutriMate-App
RESTful Web Services

Gruppe MATCH

Marina Waller
Alexandra Waas
Tim Wagner
Celine Grenz
Dang Phuong Hien Le

Inhaltsverzeichnis

1. Einleitung **3**
2. Idee **4**
3. Anforderungen **6**
4. Beschreibung unserer API-Schnittstelle **7**
5. Beschreibung der benutzten Fremddienste **8**
5.1 Open Food Facts API 8
5.2 Edamam API 8
6. Hypermedia und Caching **9**
7. Client-Umsetzung **11**

1. Einleitung

Dieses Projekt beschäftigt sich mit der Implementierung eines Web Services entsprechend den REST-Prinzipien. Kurz erklärt ist ein RESTful Web Service ein leichter, wartungsfreundlicher und skalierbarer Dienst, der auf der Representational State Transfer (REST)-Architektur basiert. Diese Dienste nutzen das HTTP-Protokoll und ermöglichen die Bereitstellung und Wiederherstellung von APIs auf sichere, konsistente und zustandslose Weise. Dabei können Clients vordefinierte Vorgänge für diese REST-Dienste ausführen.

Ein RESTful-Dienst hat mehrere wichtige Anforderungen, die ihm einen effizienten und sicheren Betrieb ermöglichen:

- Zunächst einmal muss eine kohärente URL-Struktur mit spezifischen Ressourcenkennungen vorhanden sein, die aber dennoch leicht verständliche Pfade enthalten. Diese Pfade geben die hierarchische Beziehung der Ressourcen an.
- Zweitens ist die Verwendung von HTTP-Methoden wie GET, POST, PUT, PATCH und DELETE für den Zugriff auf die Erstellung, Aktualisierung und Löschung von Ressourcen unerlässlich. Darüber hinaus basiert ein RESTful-Dienst auf dem Prinzip der Zustandslosigkeit, was bedeutet, dass jede API-Anfrage alle für die Verarbeitung erforderlichen Informationen enthalten muss, ohne dass eine serverseitige Sitzungsverwaltung erforderlich ist. HTTP-Statuscodes wie 200 OK, 201 Created und 404 Not Found werden dabei verwendet, um die Ergebnisse klar zu kommunizieren.
- Drittens sind es die Sicherheitsmaßnahmen, die eine zentrale Rolle spielen: So ist es nötig, API-Authentifizierungsmethoden wie Basic Auth und tokenbasierte Authentifizierung zu implementieren sowie OAuth für eine sichere Zugriffskontrolle zu verwenden. Um die Sicherheit zu gewährleisten, sollte die Kommunikation zudem über HTTPS verschlüsselt erfolgen.
- Für den Datenaustausch werden Standardformate wie JSON oder XML verwendet, um ein einheitliches Schema für Anfragen und Antworten beizubehalten. HATEOAS (Hypermedia As The Engine Of Application State) wird eingesetzt, um Links in API-Antworten einzubetten und die Navigation zwischen Ressourcen zu erleichtern.
- Caching ist ein weiteres wichtiges Element. Caching verbessert die Leistung und reduziert die Serverlast durch die Unterstützung von HTTP-Caching-Headern wie Cache-Control und ETag.
- Um Entwickler bei ihrer Arbeit zu unterstützen, bedarf es außerdem einer klaren und ausreichenden Dokumentation der API-Endpunkte (hier wird z.B. OpenAPI/Swagger genutzt) und ihrer Verwendung mit Anfrage- und Antwortbeispielen.
- Schließlich sollte die API-Versionierung unterstützt werden, damit Änderungen und Weiterentwicklungen ohne Unterbrechung des Dienstes möglich sind.

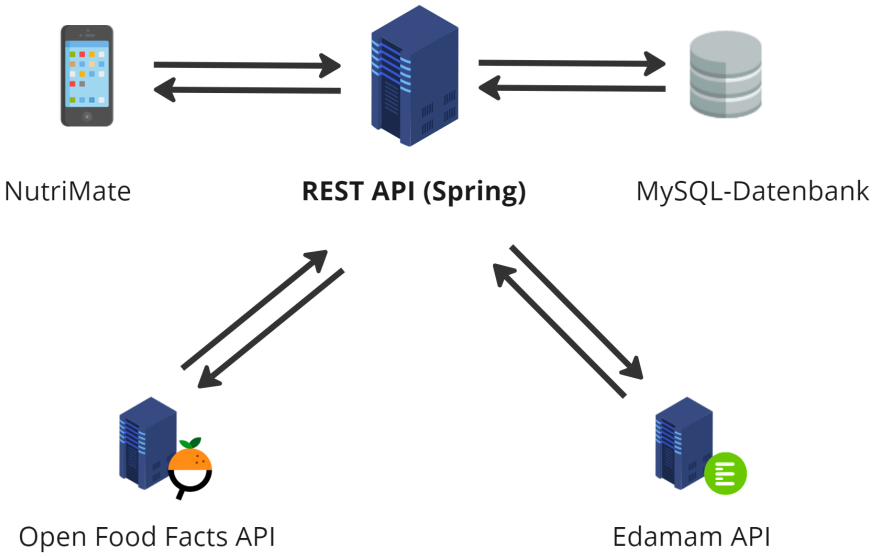
In diesem Projekt werden wir die Implementierung eines RESTful Web Services anhand der oben genannten Anforderungen und Best Practices durchführen.

2. Idee

Für dieses Modul haben wir die App “NutriMate” entwickelt, die die Prinzipien von REST verfolgt. NutriMate bietet die Möglichkeit, Lebensmittelprodukte effizient zu verwalten und kulinarische Inspirationen zu finden. Der Nutzer kann den Barcode eines gekauften Lebensmittelprodukts scannen, woraufhin das Produkt in seinem digitalen Kühlschrank gespeichert wird. Sobald der Barcode gescannt ist, kann der Nutzer die detaillierten Informationen (Nährwertangaben, Zutatenlisten und weitere wichtige Details) zum Produkt abrufen. Zusätzlich bietet unsere App eine Funktion zur Rezeptempfehlung. Basierend auf im Kühlschrank gespeicherten Produkten werden passende Kochideen (Rezepte) angezeigt.

In die App haben wir zwei APIs integriert: Open Food Facts und Edamam. Open Food Facts ist eine Lebensmitteldatenbank, die von allen für alle erstellt wurde. Wenn man einen Barcode scannt, wird unser REST-Dienst aktiviert und nutzt die API von Open Food Facts, um die Informationen von dem Lebensmittel zurückzugeben, die dem Benutzer direkt in der App angezeigt werden. Für Rezeptvorschläge nutzen wir die API von Edamam, die eine umfangreiche Datenbank mit Rezepten bereitstellt. Die Edamam-API erhält die Produkte des Nutzers als Stichworte und schlägt darauf basierend kreative und schmackhafte Rezepte vor, die der Nutzer einfach nachkochen kann. Mehr hierzu folgt in Kapitel 5.

Wie funktioniert die grundlegende Kommunikation zum REST-Server und zwischen den verschiedenen Komponenten in unserer Architektur?



- Wenn der Nutzer den Barcode eines Produkts scannt, sendet die App eine HTTP POST-Anfrage an unseren REST API-Server, der mit dem Spring Framework implementiert ist. Diese Anfrage enthält die Barcode-Daten des gescannten Produkts.
- Der REST-Server empfängt die Anfrage und leitet sie an die Open Food Facts API weiter, um detaillierte Produktinformationen abzurufen. Die Open Food Facts API antwortet mit den entsprechenden Daten, die der REST-Server dann in der MySQL-Datenbank speichert. Gleichzeitig sendet der Server die Produkt -informationen an die NutriMate App zurück, die diese dem Nutzer anzeigt.

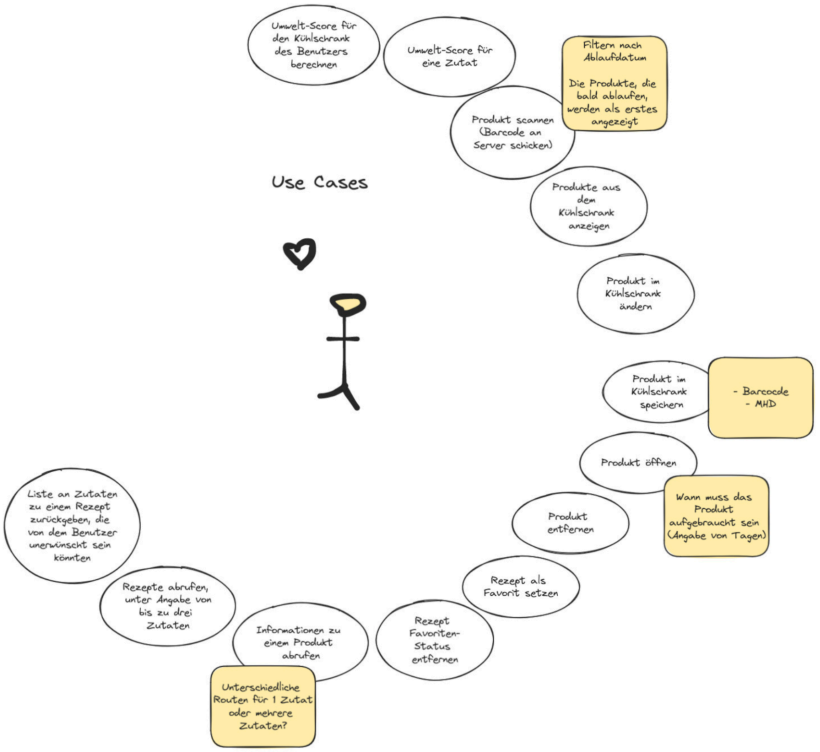
- Für Rezeptvorschläge nutzt der REST-Server die Edamam API. Wenn der Nutzer Rezeptvorschläge anfordert, sendet die NutriMate App eine HTTP GET-Anfrage an den Server. Der Server ruft die relevanten Produktinformationen aus der MySQL-Datenbank ab und leitet diese Informationen an die Edamam API weiter, um passende Rezepte zu erhalten. Die Edamam API antwortet mit einer Liste von Rezepten, die der REST-Server an die NutriMate App zurücksendet, welche die Rezepte dem Nutzer präsentiert.

Wie speichern wir die Produkte des Benutzers?
Jeder Benutzer hat seinen eigenen Kühlschrank. Jedoch ist der Kühlschrank nur für die angemeldete Benutzer verfügbar. Nach der Anmeldung können Benutzer Produkte in ihrem digitalen Kühlschrank speichern. Hierbei werden nicht nur die Produktinformationen, sondern auch das Mindesthaltbarkeitsdatum (MHD) erfasst. Nachdem man ein neues Produkt im Kühlschrank gespeichert hat, kann man das Produkt als “geöffnet” markieren und/ oder das MHD ändern.
Zusätzlich können Nutzer Einstellungen in ihrem Benutzerkonto vornehmen, die Informationen zu Allergien, Unverträglichkeiten und bevorzugten Ernährungsformen wie “vegetarisch”, “vegan” oder “low-carb” enthalten. Diese individuellen Einstellungen werden bei der Suche und Empfehlung von Rezepten berücksichtigt, um sicherzustellen, dass die vorgeschlagenen Rezepte den persönlichen Bedürfnissen und Vorlieben des Nutzers entsprechen.
Ein weiteres Highlight der App ist die Möglichkeit, Lieblingsrezepte zu speichern. Diese Rezepte verweisen direkt auf die ID oder URL der Edamam API, was eine einfache und schnelle Wiederauffindbarkeit der bevorzugten Rezepte gewährleistet.

Mit unserer NutriMate App bieten wir den Nutzern eine Reihe von Mehrwerten, die den Alltag erleichtern und eine gesunde sowie umweltbewusste Ernährung unterstützen. Mit der App können die Nutzer zuerst einfach und bequem den Barcode eines Lebensmittelprodukts scannen, um sofort Nutri-Score und Nährwerte angezeigt zu bekommen und das Produkt in ihrem digitalen Kühlschrank zu speichern. Zweitens ermöglicht unsere App die Suche nach Rezepten unter Angabe von bis zu drei Produkten aus dem Kühlschrank. Zusätzlich wird der Umwelt-Score eines Produkts aus den Open Food Facts API-Daten berechnet, wodurch die Nutzer die Umweltfreundlichkeit ihrer Einkäufe besser einschätzen können, und die App warnt auch bei kritischen Inhaltsstoffen (Allergien). Außerdem werden gespeicherte Produkte nach Ablaufdatum sortiert, um Lebensmittelverschwendung zu reduzieren. Zuletzt werden personalisierte Rezeptvorschläge angezeigt, die die Allergien, Unverträglichkeiten und Ernährungspräferenzen berücksichtigen.

3. Anforderungen

Das Ziel der Anwendung ist es Benutzern zu helfen, ihre Lebensmittel effizient zu verwalten und umweltbewusst zu konsumieren. Dafür wurden zu Beginn alle Anforderungen an die App und das Backend formuliert und in einem Use Case Diagramm festgehalten.



Eine wichtige Anforderung ist, dass es dem Benutzer möglich sein soll, den Barcode eines Produkts in der App zu scannen. Dieser Barcode soll von der App an den Server geschickt werden. Der Server soll den Barcode verarbeiten und die Informationen des Produkts zurückgeben. Jeder Benutzer soll einen digitalen Kühlschrank besitzen, in dem er alle Lebensmittel speichern kann und so einen Überblick über seine Vorräte behalten kann.

Der Nutzer soll ein Produkt zu seinem Kühlschrank hinzufügen können, indem er den Barcode des Produkts scannt und das Mindesthaltbarkeitsdatum hinzufügt. Bei Bedarf sollen Produkte bearbeitet oder entfernt werden können. Beim Bearbeiten soll es möglich sein, anzugeben, wann ein Produkt geöffnet wurde, und wie viele Tage es nach dem Öffnen verbraucht werden soll. Zusätzlich soll die App es ermöglichen, detaillierte Informationen zu einem Produkt abzurufen.

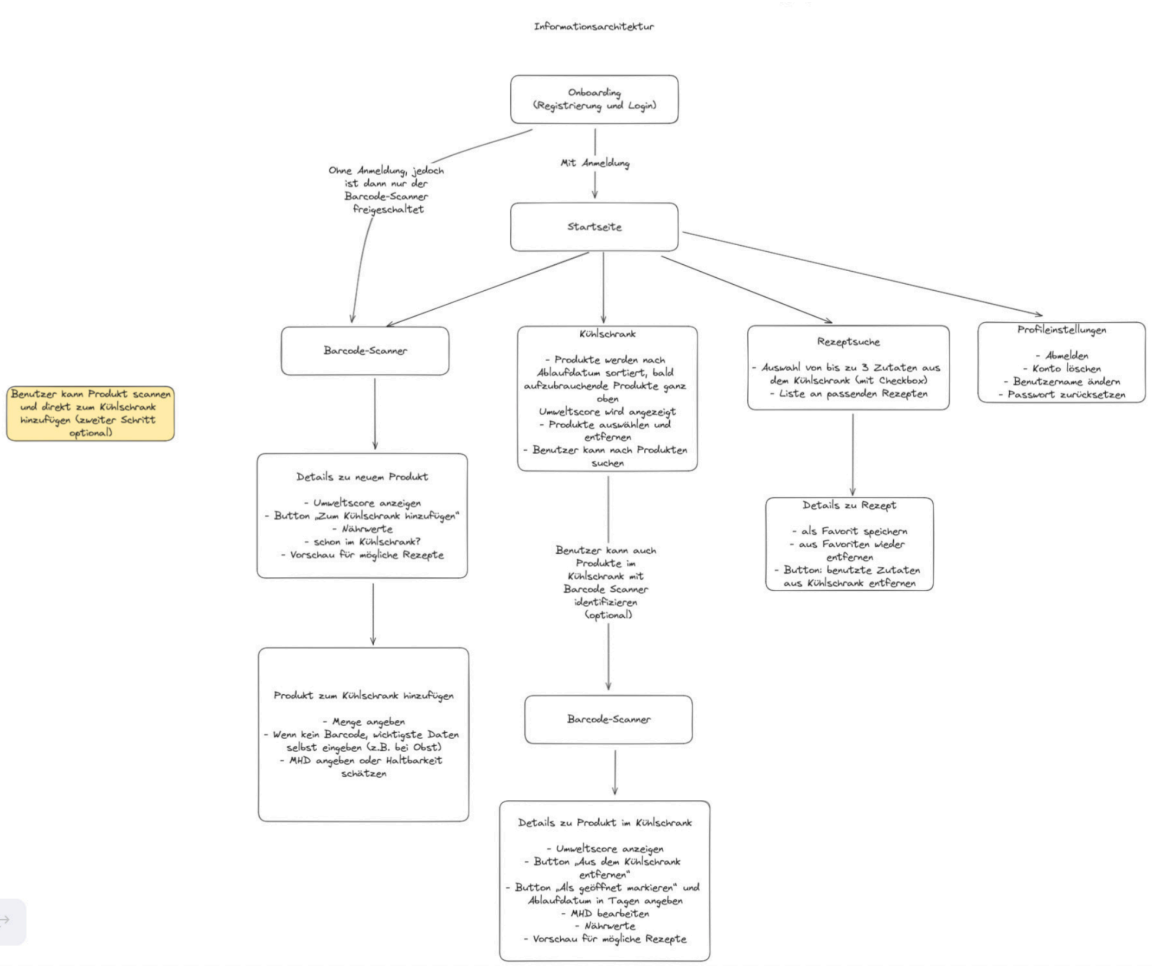
Es soll für den gesamten Kühlschrank sowie für einzelne Zutaten ein Umwelt-Score berechnet werden. Produkte sollen zusätzlich nach Ablaufdatum sortiert werden können, sodass die Lebensmittel, die als nächstes das Mindesthaltbarkeitsdatum überschreiten, zuerst angezeigt werden.

Dem Benutzer soll es möglich sein, unter Angabe von bis zu drei Zutaten, empfohlene Rezepte abzurufen und diese als Favoriten zu markieren und die Markierung wieder zu entfernen. Es soll eine Funktion geben, die es ermöglicht, Zutaten eines Rezepts zurückzugeben, die der Benutzer nicht verträgt. Unverträglichkeiten soll der Benutzer speichern können und diese sollen bei der Rezeptsuche berücksichtigt werden.

Die technischen Anforderungen für das Backend beinhalten einen Server, der Barcode verarbeitet, Produkte speichert und die Umwelt-Scores berechnen kann. Zudem ist eine Datenbank nötig, die Produktinformationen, die Unverträglichkeiten des Benutzers und favorisierte Rezepte speichern kann. Es sollen zwei Schnittstellen verwendet werden, um die Kommunikation zwischen der App und dem Backend zu ermöglichen.

Die App erfordert einen Barcode-Scanner und eine benutzerfreundliche Schnittstelle zur Verwaltung der Produkte und Anzeige von Rezepten. Filtern und Sortieren von Produkten nach Ablaufdatum und anderen Kriterien sollen möglich sein.

Die beschriebenen Anforderungen wurden als Leitfaden für die Entwicklung und Implementierung der App und des Backends genutzt. Darauf basierend wurde zudem folgende Informationsarchitektur erstellt, die die Interaktion des Users mit dem Client beschreibt:



4. Beschreibung unserer API-Schnittstelle

Unsere Swagger-Dokumentation beschreibt die API-Endpunkte, mit denen das Backend der Nutrimate-App angesprochen werden kann.

Die yaml-Dokumentation befindet sich im Verzeichnis nutrimate-backend/specs/swagger dieses Repositorys. Sie kann bei vorhandener Docker-Installation auf dem Rechner mit dem Befehl “docker compose up” als Docker-Container gestartet werden und ist dann im Browser unter http://localhost:8089 abrufbar.

Im aktuellen Verzeichnis (documentation) befindet sich zudem eine html-Version der Schnittstellenbeschreibung (beschreibung-der-schnittstelle.html). Diese kann unabhängig von Docker betrachtet werden.

5. Beschreibung der benutzten Fremddienste

5.1 Open Food Facts API

Open Food Facts ist eine Online-Datenbank, in der Lebensmittel mithilfe ihres Namens oder Barcodes gesucht werden und Informationen wie Nährwerte, Allergene oder Umweltauswirkungen des Lebensmittels nachgelesen werden können. Der Dienst bietet eine kostenfreie API an, mithilfe derer Abfragen aus der Datenbank möglich sind. Diese ist unter https://world.openfoodfacts.org/files/api-documentation.html dokumentiert.

Der Endpunkt GET https://[countrycode].openfoodfacts.org/api/v0/product/[barcode].json wurde in unserer Anwendung mit dem countrycode "en" für den englischsprachigen Raum sowie dem jeweiligen Barcode des Produktes angesprochen. Die Nutzung erfolgte in den beiden Endpunkten zum Scannen eines Barcodes und zum Speichern eines gescannten Lebensmittels im eigenen Kühlschrank. Bei letzterem Endpunkt werden ausgewählte Teile der Antwort, beispielsweise der Zuckergehalt oder einer der Bild-URLs, in unsere Datenbank übernommen. Als Kategorie, welche für die Suche nach Rezepten in der Edamam-Datenbank genutzt wird, wurde jeweils der letzte und somit spezifischste Eintrag im Array CategoryTags genutzt, damit ein allgemeiner Name des Lebensmittels zur Suche verwendet werden kann.

Da Open Food Facts eine offen bearbeitbare Datenbank ist, gibt es zahlreiche unvollständige und auch fehlerhafte Einträge. Es wurde sich dennoch für Open Food Facts entschieden, da die Datenbank vollkommen kostenlos ist und sich die Fehler bei den meisten Produkten in Grenzen halten. Aus diesen Gründen wurden als Allergene auch nur die 14 wichtigsten Allergene, die von Edamam und Open Food Facts unterstützt werden, in die App aufgenommen.

5.2 Edamam API

Edamam ist eine Datenbank mit APIs, die zahlreiche Schnittstellen im Zusammenhang mit Lebensmitteln anbietet. Sie lässt sich nur mit API-Key und App-Code nutzen, weshalb ein Account für unsere App angelegt wurde.

Die Dokumentation der für unser Projekt verwendeten Recipe Search API ist zu finden unter https://developer.edamam.com/edamam-docs-recipe-api. In unserem Backend wird der Endpunkt GET https://api.edamam.com/api/recipes/v2 mit den Query-Parametern q (Suchbegriffe) und health (Allergene) aufgerufen (sowie den verpflichtenden Parametern app-key und app-id). Hierdurch wird nach Rezepten gesucht, die bestimmte Zutaten enthalten und bestimmte Allergene nicht enthalten. Die Zutaten-Suche wird mithilfe des zuvor erwähnten Category-Attributs umgesetzt.

Edamam bietet eine sehr umfangreiche Rezeptsammlung an. Schwierigkeiten verursachten die nicht ganz mit Open Food Facts übereinstimmenden Allergen-Namen, die in unserem Backend daher umgewandelt werden mussten. Zudem stellt die Suche mithilfe der Open Food Facts Category noch ein Problem dar, denn bei zu spezifischen Kategorien (z.B. "chocolate and hazelnut spread" für Nutella) kann kein passendes Rezept gefunden werden. Hier wäre für die reale Umsetzung der App eine Lösung mit Einbindung einer KI denkbar.

6. Hypermedia und Caching

Als zusätzliche Anforderungen an die App wurden die Implementierung von Hypermedia sowie das Cachen zweier Ressourcen auf unterschiedliche Art und Weise verlangt.

Als Hypermedia-Ressource wurde der Inhalt des Kühlschranks eines Users (Ressource “Food”) gewählt, da die Paginierung des Kühlschranks ein sinnvolles Einsatzgebiet für Hypermedia-Links bietet. Mithilfe der Paginierungsfunktion des Spring Boot Repositories, der Zugriffsschnittstelle auf die Datenbank, sowie der Implementierung des entsprechenden Endpunktes “getAllFood” in der Food-Controller-Klasse wurde sichergestellt, dass das Backend auf die Query-Parameter “page” (aktuelle Seitenzahl) und “size” (maximale Anzahl an Objekten auf einer Seite) reagiert. Somit liefert das Backend beim Ansprechen des Endpunkts GET http://localhost:8080/fridge/food?page=0&size=3 z.B. folgende Antwort:

```
{
   "_embedded": {
      "foodDTOResponseList": [
         {
            "id": 1,
            "barcode": "123456789",
            "expireDate": "2024-06-28T07:01:42.334256",
            "category": "spread",
            "name": "Nutella",
            …                                    // weitere Attribute
            "_links": {
               "self": {
                  "href": "http://localhost:8080/fridge/food/1"
               }
            }
         },
            …          //  weitere Food-Objekte mit self-Link
      ]
   },
"_links": {
      "first": {
         "href": "http://localhost:8080/fridge/food?page=0&size=3"
      },
      "self": [
         {
            "href": "http://localhost:8080/fridge/food?page=0&size=3"
         },
         {
            "href": "http://localhost:8080/fridge/food?q="
         }
      ],
      "next": {
         "href": "http://localhost:8080/fridge/food?page=1&size=3"
      },
      "last": {
         "href": "http://localhost:8080/fridge/food?page=1&size=3"
      }
   },
   "page": {
      "size": 3,
      "totalElements": 6,
      "totalPages": 2,
      "number": 0
   }
}
```

Der Client kann nun mit den URLs im “_links”-Abschnitt der Response arbeiten und zur ersten, vorherigen, nächsten sowie zur letzten Seite des Kühlschranks weiterleiten.

Das Caching wurde ebenfalls anhand der Ressource “Food” implementiert. Für die erste Variante des Cachings mithilfe von E-Tags im Header von Response und Request wurde der Endpunkt GET localhost:8080/food/{barcode} ausgewählt, da dieser auf die Open-Food-Facts-API zugreift und dementsprechend eine längere Wartezeit bis zum Empfangen der Daten hat. In diesem Fall ist es sinnvoll, diese Zeit durch Caching zu verkürzen, da in der Datenbank von Open-Food-Facts nur in unregelmäßigen Abständen Änderungen erfolgen. Im Backend wird für jedes abgerufene Food ein eindeutiger Hash-Wert berechnet und als E-Tag dem Header der API Response angefügt. Diesen E-Tag kann das Frontend speichern und bei einer erneuten Anfrage als “If-None-Match”-Headerparameter an den Endpunkt mitsenden. Entspricht der E-Tag der bereits gesendeten Ressource, antwortet das Backend mit dem Status-Code 304 - Not Modified und einem leeren Body.

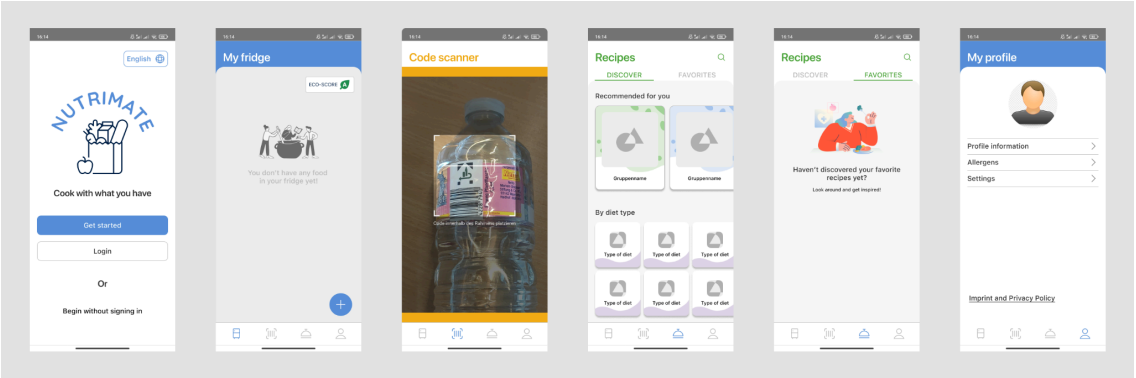
Die zweite Variante des Caching, das Senden eines Cache-Control Headers, wurde für die Ressource “Allergens” im Endpunkt GET localhost:8080/allergens umgesetzt. Diese ändert sich nur, wenn der Betreiber der Nutrimate-API neue Allergene hinzufügt oder vorhandene Allergene entfernt, d.h. es treten nur sehr selten Änderungen auf. Daher wurde im Cache-Control Header das max-age der Ressource auf 86400 (24 Stunden) gesetzt. Dies bedeutet, dass nach Ablauf dieser Zeit der Client die erhaltene Ressource neu holen soll, da sich Änderungen ergeben haben könnten. Die Response des Endpunkts besitzt somit eine Gültigkeit von 24 Stunden.

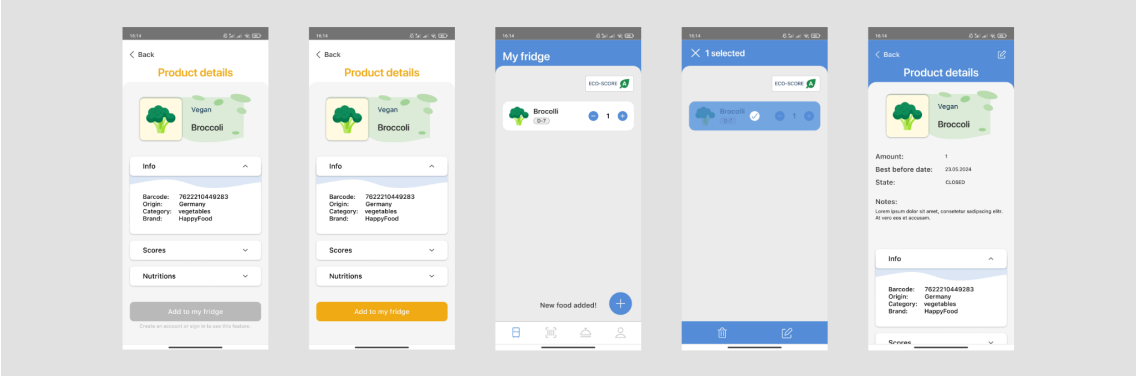
7. Client-Umsetzung

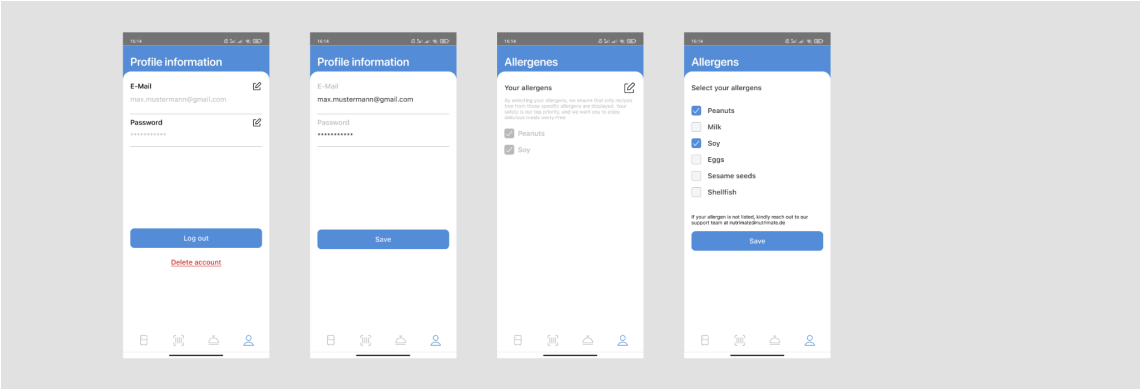
Bezüglich der Implementierung eines Clients, mit dem auf das Backend zugegriffen werden kann, entschieden wir uns für eine Flutter-App als leichtgewichtige und vielseitige Cross-Platform-Anwendung.

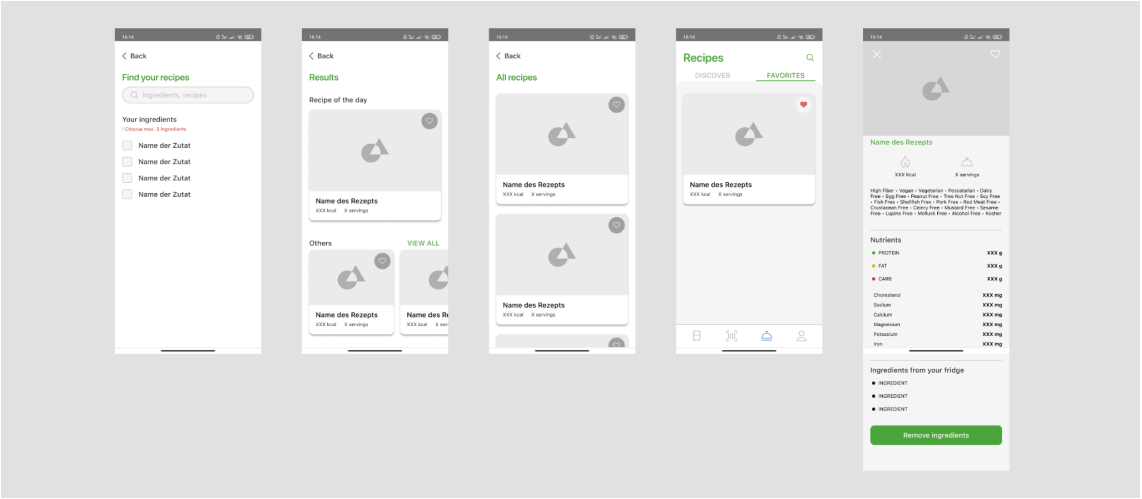
Bevor mit der Entwicklung des Frontends begonnen werden konnte, wurde zunächst eine Prototyp erstellt, um die Anforderungen und Funktionen der App vor der Programmierung zu visualisieren und ansprechend zu gestalten.Mit der Software “Figma” wurde ein einheitliches Design entwickelt, Probleme frühzeitig erkannt und behoben sowie überprüft, ob alle Funktionalitäten den Anforderungen entsprechen.

Nachfolgend sind die einzelnen Screens der App dargestellt.









Der vollständige Prototyp kann unter diesem Link eingesehen werden:
https://www.figma.com/design/M0ySdOkNb6yLsvfNZEOEha/RESTful_NutriMate_Design_MATCH?node-id=1-2&t=uwkUpD6B5sf211fd-1