Integrations-Projekt CAS HTML5

Studierende:
Luca Brandt
Miro Rutscho
Dominique Müller

Projekt-Dokumentation FridgeBuddy

Eine Initiative zur Reduzierung von Food Waste

Inhaltsverzeichnis

1 Thema

FridgeBuddy ist eine Plattform für den Austausch von Lebensmitteln zur Reduzierung von Food Waste und Förderung von günstigen Lebensmitteln.

2 Umfeld, Ausgangslage

Gemäss Bundesamt für Umwelt BAFU werden in Schweizer Haushalten pro Jahr 778‘000 Tonnen Lebensmittel verschwendet. Dies entspricht ungefähr einem Drittel des gesamten Food Waste in der Schweiz.
Die Schweizer Haushalte verschwenden dadurch pro Jahr über 5 Milliarden Franken. Pro Kopf werfen wir jährlich ungefähr 100kg Lebensmittel im Wert von 620.- in die Tonne.
Diverse Projekte fördern bereits den Austausch von Lebensmitteln, beispielsweise in öffentlich zugänglichen Kühlschränken (https://www.madamefrigo.ch/).
FridgeBuddy setzt sich nun zum Ziel, den Austausch von Lebensmittel auf einer Online-Plattform zu unterstützen.

3 Aufgabenstellung & Ziel des Projektes

Das primäre Ziel von FridgeBuddy ist, Lebensmittelspender und Konsumenten zusammen zu bringen. Spender, welche ein Überangebot an Lebensmittel haben, können Lebensmittel als Spende in FridgeBuddy publizieren.

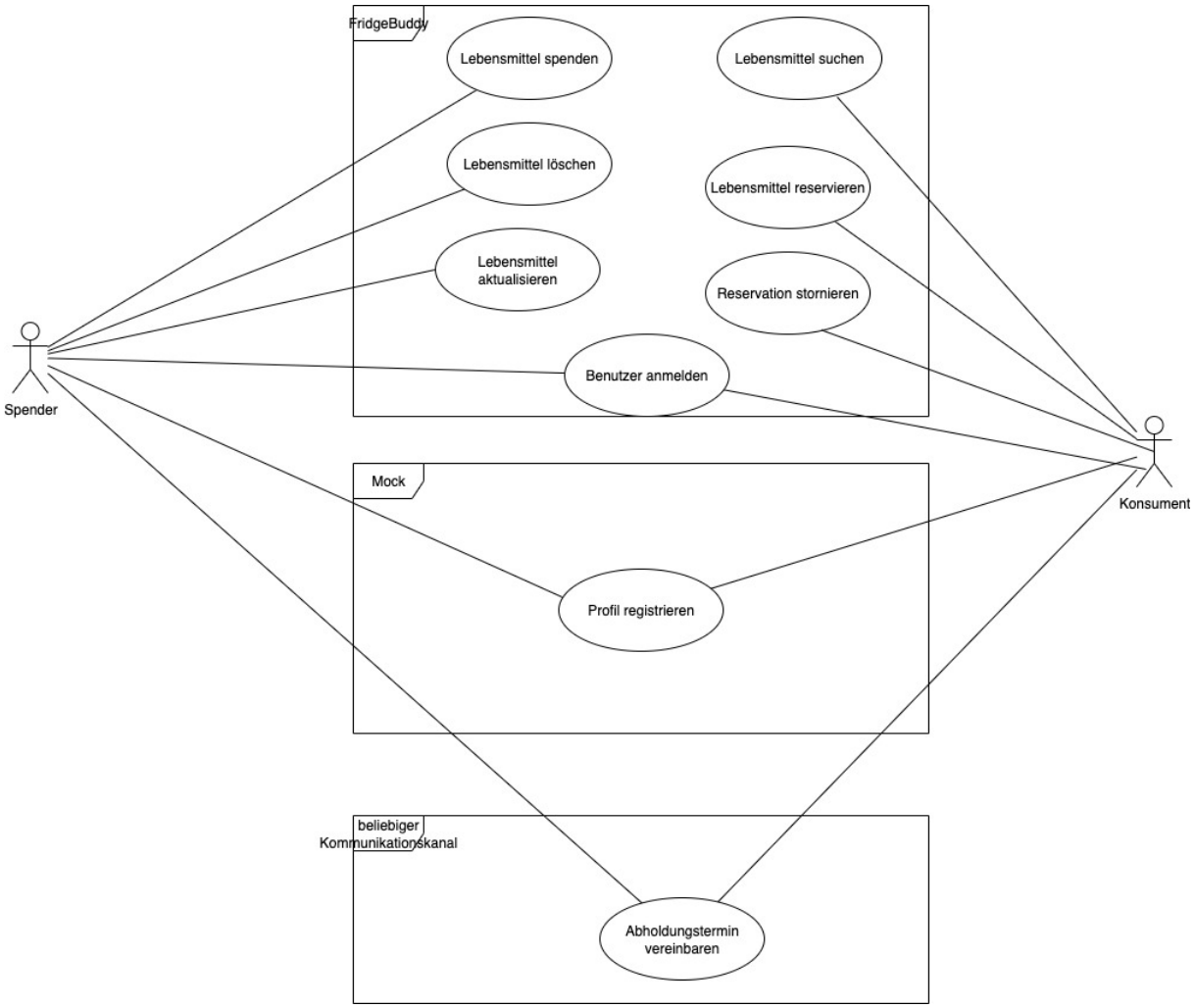
Interessierte können die Lebensmittel online reservieren und die Kontaktdaten austauschen.

Sowohl Spender als auch Konsumenten profitieren von der Plattform. Die Abnehmer der Lebensmittel profitieren finanziell von günstigen Lebensmitteln, wobei die Spender den eigenen Fussabdruck optimieren können.

4 Scope des Projektes

4.1 Funktionale Anforderungen

Nachfolgend ein Use-Case-Diagramm unseres Produkts. Auf die spannenden Use-Cases wird weiter unten detaillierter eingegangen.



4.1.1 Lebensmittel spenden

Die Spender können Lebensmittel anbieten. Hierfür können diverse Eigenschaften erfasst werden, wie beispielsweise die folgenden:

- Titel des Produkts

- Beschreibung
- Label (Bio, IP-Suisse)
- Foto
- Menge
- Essgewohnheit: vegetarisch, vegan

4.1.2 Lebensmittel suchen

Die Konsumenten können Lebensmittel nach diversen Kriterien suchen.
Filterkriterien können beispielsweise die eigenen Essgewohnheiten oder präferierte Label sein.

4.1.3 Lebensmittel reservieren

Die Konsumenten können Lebensmittel reservieren und bekommen anschliessend die Kontaktdaten des Anbieters angezeigt.

4.1.4 Reservation stornieren

Die Konsumenten können ihre eignen Reservationen löschen.

4.1.5 Benutzer anmelden

Die Anmeldung des Users ist im Frontend möglich.

4.1.6 Profil registrieren

Das Backend kann zwar User registrieren aber wir werden im Frontend nur vorgefertigte User verwenden und keinen echten Login-Mechanismus implementieren.

4.2 Nicht-Funktionale Anforderungen

4.2.1 Relevante Endgeräte

Nur Smartphones sind aktuell relevante Endgeräte für FridgeBuddy. FridgeBuddy wird nicht für andere Endgeräte optimiert.

4.3 Scope Änderungen im Vergleich zu Projektspezifikation

4.3.1 Fokus auf Wesentliches

In der Projektspezifikation waren teilweise sehr viele Details spezifiziert ohne wesentlichen Begeisterungseffekt. Beispielsweise war geplant, dass man für ein Produkt sehr viele Felder wie zB „Lebensmittelart“, „Herkunftsland“ oder „mindestens haltbar“ erfasst kann. Wir haben diesbezüglich auch in Absprache mit den Dozenten festgestellt, dass nicht sämtliche Felder umgesetzt werden. Wir haben uns auf einige wesentliche Felder fokussiert, zB Fotos, Ablaufdatum oder die Menge.

So konnten wir uns auch im Usability-Testing darauf Fokussieren, die konzeptuelle Idee und den Reservationsprozess zu überprüfen, statt uns auf Details zu fokussieren.
Diese Anpassung war sicherlich ein wichtiges Learning und passt auch zur agilen Software-Entwicklung, wo man sich auf schnelle User-Feedbacks konzentriert.

4.3.2 Learnings aus Usability-Testing

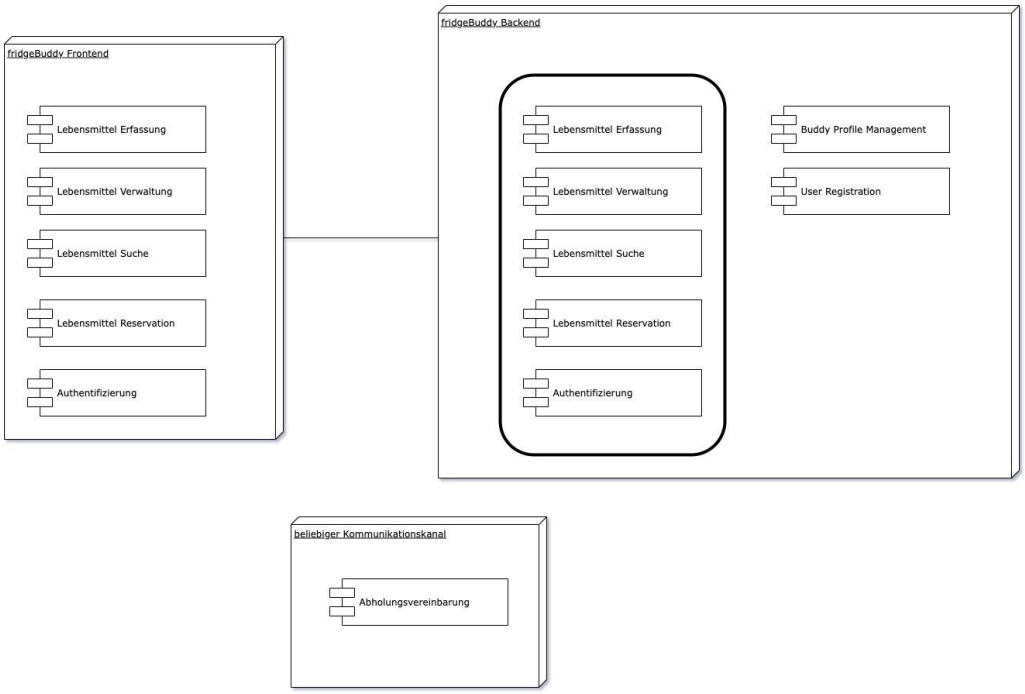
Einige Feedbacks aus dem Usability-Testing konnten direkt in das Endprodukt einfliessen. Beispielsweise wurde die Filterung nach Demeter-Produkten entfern. Weiter konnten wir den Umgang mit den Eigenschaften vegetarisch und vegan konkretisieren. Wir wussten lange nicht, wie wir bei der Erfassung und Sortierung von Produkten damit umgehen sollen, dass ja vegetarische Produkte automatisch vegan sind. Auf der einen Seite wollten wir, dass es nicht möglich ist, vegane Produkte zu erfassen, die nicht vegetarisch sind. Dies wäre fachlich nicht korrekt. Auf der anderen Seite wollten wir sowohl die Erfassung als auch die Suche intuitiv und effizient gestalten.
Schlussendlich haben wir dies so gelöst, dass bei der Erfassung von veganen Produkten die Checkbox «vegetarisch» automatisch aktiviert und disabled wird.

5 Systemübersicht, Grob-Architektur, Deployment

5.1 Wichtigste Links und Artefakte

Artefakt	Link
Lauffähige Installation	https://fridge-buddy-project.web.app/
Backend	https://fridge-buddies.onrender.com/api-auth/login/
Präsentation	https://bernerfachhochschule-my.sharepoint.com/:p:/r/personal/mulld7_bfh_ch/Documents/fridgeBuddy.pptx?d=waebd76b50fae456dab37c9c8f3723ca6&csf=1&web=1&e=Cb3qlc

5.2 Systemübersicht und Bestandteile des Systems



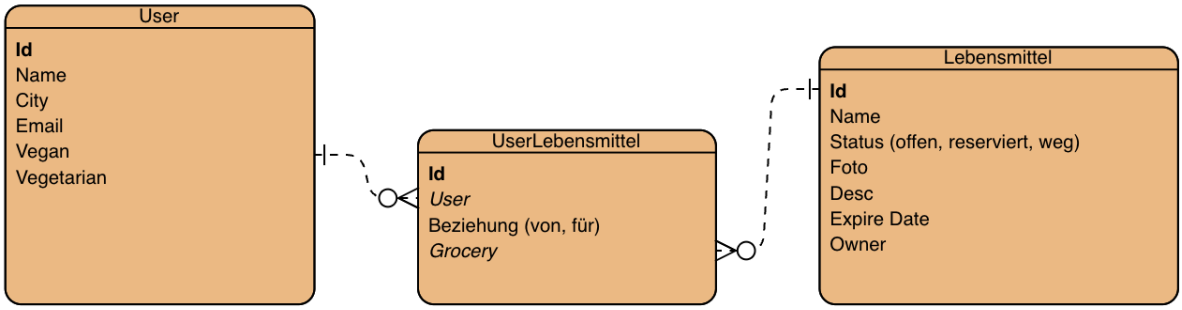
Obenstehende Grafik veranschaulicht die Bestandteile des Systems. Im Frontend wurde der Fokus auf die Bestandteile gelegt, welche auch ein entsprechendes Interesse für die Demo haben: Lebensmittel Erfassung, Lebensmittel Verwaltung, Lebensmittel Suche, Lebensmittel Reservation, Authentifizierung.

Das Buddy-Profile Management und die User-Registrierung wurde nur im Backend umgesetzt. Diese Bestandteile wurden nicht im Frontend umgesetzt.

Die Vereinbarung vom Abholungstermin ist komplett ausserhalb des Systems und wurde weder im Backend noch im Frontend umgesetzt. Die Beteiligten tauschen auf unserer Plattform lediglich die Kontaktdaten aus und wechseln anschliessend in einen beliebigen Kommunikationskanal.

5.3 Lösung

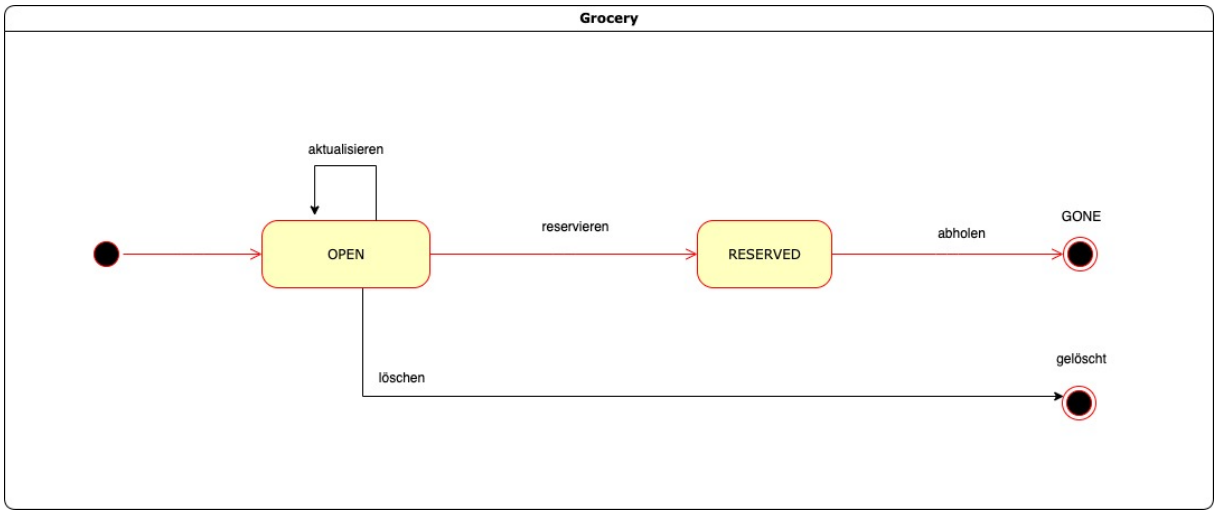
5.3.1 Datenmodell



Ein User kann mehrere Lebensmittel haben und ein Lebensmittel kann zu mehreren Usern gehören. Damit kann ein User ein Lebensmittel erstellen und ein anderer kann es reservieren. Der aktuelle Status wird in der Zwischentabelle gespeichert.

5.3.2 Statusdiagramm

Ein Lebensmittel kann fachlich betrachtet im Lebenszyklus die untenstehenden Stati durchlaufen.



5.3.3 Statemanagement

Die Daten für “Lebensmittel suchen” werden aus dem Backend geholt. Das funktioniert mit Pagination und Filtern. Wird ein Lebensmittel angeklickt, wird mit der ID dieses Lebensmittels eine neue Abfrage gemacht, um alle Daten zu bekommen. Diese werden auf der Detail-Seite angezeigt. Reserviert der Benutzer das Lebensmittel, dann wird eine weitere Abfrage gesendet, um den Status zu aktualisieren.
Von “Meine Produkte bearbeiten” wird erst eine Abfrage ans Backend gesendet, um alle eigenen Lebensmittel zu erhalten. Bei einer Änderung wird eine Abfrage ans Backend gesendet, um dieses eine Lebensmittel zu aktualisieren.

Die Updates im UI erfolgen nach dem pessimistischen Pattern, d.h. wir warten auf die Antwort des Backends, bevor wir die View aktualisieren. Aufgrund der Grösse des Projektes verzichten wir zudem auf eine Statemanagement-Library wie ngrx oder Akita und es wird keine client-seitige Persistenz und Offline-Fähigkeit implementiert.

5.4 Deployment-Struktur

5.4.1 Deployment Backend

Das Deployment des Django Rest Frameworks für unser Projekt erfolgt über die Plattform Render, wobei eine persistente Festplatte sowie eine PostgreSQL-Datenbank genutzt werden. Zunächst wurde das Projekt lokal entwickelt und alle Abhängigkeiten in der requirements.txt dokumentiert. Für das Deployment wird die Datei “backend.py” entsprechend angepasst.
Die Konfiguration der Umgebungsvariablen für die Datenbankverbindung und andere sensible Informationen erfolgte über die Render-Einstellungen, um die Sicherheit zu gewährleisten. Die persistente Festplatte wurde eingerichtet, um statische und Medien-Dateien zu speichern, wobei die entsprechenden Pfade in den Django-Einstellungen angegeben wurden.
Dieser Prozess stellt sicher, dass unsere Anwendung effizient und zuverlässig in einer Produktionsumgebung läuft. Um Änderungen zu deployen, wird der lokale Code zu GitHub “gepushed” was automatisch zu einem neuen Build und Migration in Render führt.

5.4.2 Deployment Frontend

Das Frontend ist auf Firebase installiert.

6 Komponenten-Hierarchie

6.1 Mapping zwischen Routen und Hauptkomponenten

Die folgende Tabelle zeigt die Routen mit der entsprechenden Parentkomponente.

Zweck	Route	Parentkomponente
Login	'login'	app-login
Home	''	app-home
Grocery erfassen und mutieren	'grocery-form'	app-grocery-form
Meine Reservationen	'my-reservations'	app-my-reservations
Meine Angebote	'my-groceries'	app-my-groceries
Grocery Detail	'product-detail/:groceryId'	app-grocery-detail

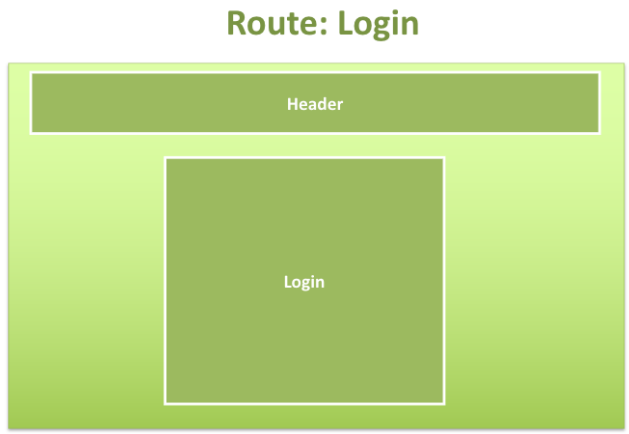
6.2 Komponenten-Hierarchie

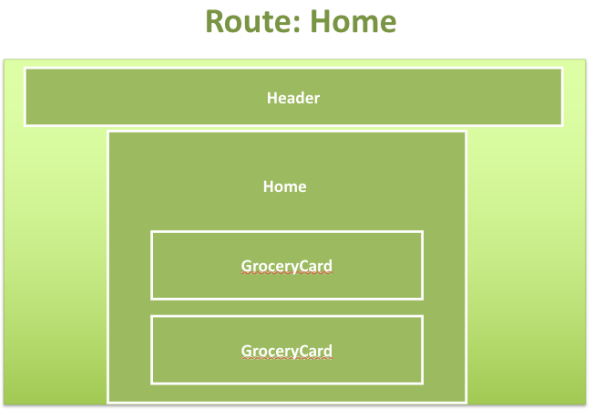
Die folgende Tabelle zeigt, welche Komponenten von den Hauptkomponenten verwendet werden.

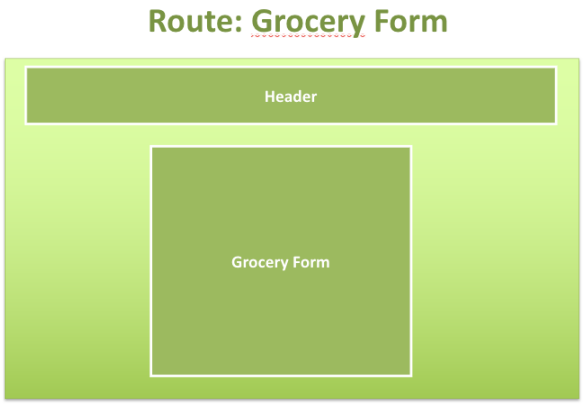
Parentkomponente	Childkomponente
app-login	-
app-home	app-grocery-card
app-grocery-form	-
app-my-reservations	app-grocery-card
app-my-groceries	app-grocery-card
app-grocery-detail	-

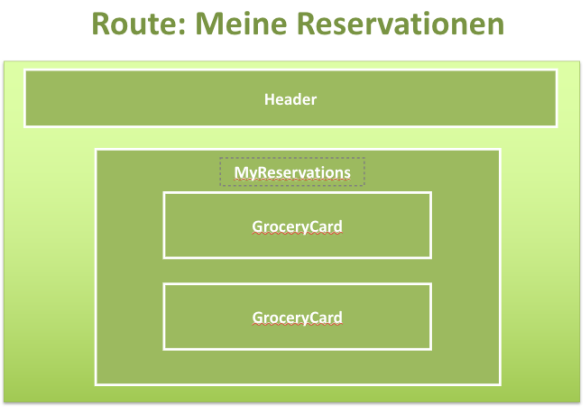
6.3 Grafische Repräsentation

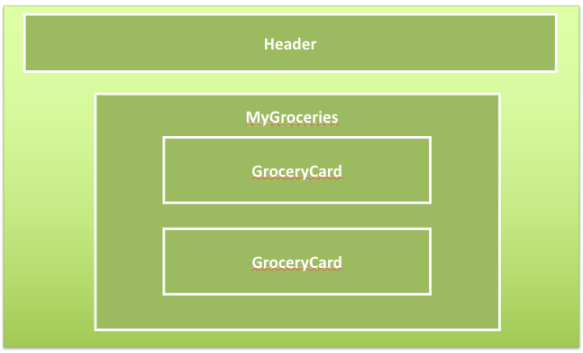
Nachfolgend befindet sich zum besseren Verständnis eine grafische Repräsentation zu den oben enthalten Informationen. Diese Informationen sind auch in der Power-Point Präsenation enthalten.

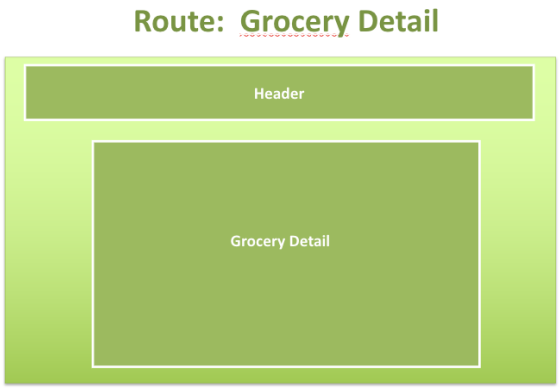





Route: Grocery Form
Header
Grocery Form


Route: Meine Reservationen
Header
MyReservations
GroceryCard
GroceryCard


Route: Meine Angebote
Header
MyGroceries
GroceryCard
GroceryCard



7 API-Dokumentation

7.1 Authentication & User Management

Die folgenden Endpoints können verwendet werden, um User Accounts zu verwalten.
Folgend eine Übersicht der Endpoints, welche weiter unten präziser dokumentiert werden.

Method	URL	Beschreibung
GET & POST	/api/token	Eingabe von Username & Passwort, Erhalt von access & refresh token
POST	/api/token/refresh	Erhalt von neuem Access Token (Refresh token senden)
GET & POST	/users/v1/buddy-profile	Profil erstellen & updaten von eingeloggtem User
POST	/users/v1/register	Neuen User Registrieren
GET	/users/v1/user-detail/{id}	Gibt Daten zu einem User zurück

7.1.1 JWT-Token erhalten

Endpoint	POST /api/token/
View	TokenObtainPairView
Zweck	Benutzer-Authentifizierung und Ausstellung von JSON Web Tokens (JWT)
Beschreibung	Ein Benutzer übermittelt seinen Benutzernamen und sein Passwort im Request-Body. Wenn die Anmeldedaten gültig sind, gibt die API ein Token-Paar zurück: ein Access-Token und ein Refresh-Token. Das Access-Token wird verwendet, um nachfolgende Anfragen an geschützte Endpoints zu authentifizieren.

Berechtigungen	Öffentlich zugänglich

7.1.2 JWT-Token erneuern

Endpoint	POST /api/token/refresh/
View	TokenRefreshView
Zweck	Erneuerung des Access Tokens, sobald das bestehende Token abgelaufen ist.
Beschreibung	Der Benutzer übermittelt sein gültiges (und nicht abgelaufenes) Refresh-Token im Request-Body. Die API stellt daraufhin ein neues Access-Token aus. Dadurch können Benutzer eingeloggt bleiben, ohne ihre Zugangsdaten erneut eingeben zu müssen.
Berechtigungen	Öffentlich zugänglich (aber gültiges Refresh-Token wird benötigt).

7.1.3 User Registrierung

Endpoint	POST /users/v1/register/
View	UserCreateAPIView
Zweck	Erstellung eines neuen Benutzer-Accounts
Beschreibung	Ein potenzieller Benutzer sendet eine POST-Anfrage mit den erforderlichen Benutzerinformationen (wie Benutzername, Passwort, E-Mail usw.). Daraufhin wird ein neues Benutzerobjekt in der Datenbank erstellt.
Berechtigungen	Öffentlich zugänglich, weil es der Einstiegspunkt für neue Benutzer ist.

7.1.4 Buddy Profile Management

Endpoint	GET, POST /users/v1/buddy-profile/
View	BuddyProfileAPIView
Zweck	Erstellung oder Ansicht des „Buddy-Profils“, welches mit dem angemeldeten Benutzer verknüpft ist.
Beschreibung	GET: Ruft das Buddy-Profil des aktuell angemeldeten Benutzers ab und zeigt es an. POST: Erstellt ein neues Buddy-Profile. Die Methode perform_create stellt dabei zwei wichtige Logiken sicher:

	1) Das neue Profil wird automatisch mit dem Benutzer verknüpft, der die Anfrage stellt (serializer.save(user=self.request.user)). 2) Zudem wird verhindert, dass ein Benutzer mehr als ein Buddy-Profil erstellt. Falls bereits ein Profil existiert, wird ein Fehler ausgelöst.
Berechtigungen	Erfordert Authentifizierung (IsAuthenticated). Nur angemeldete Benutzer können auf das eigene Profil zugreifen.

7.2 Grocery Management

Diese Endpoints verwalten den Kern der Applikation. Die Verwaltung von Groceries. Folgend eine Übersicht der Endpoints, welche weiter unten präziser dokumentiert werden.

Method	URL	Beschreibung
GET & POST	/groceries/api/v1/groceries	Alle Items filterbar nach Name, Status und Ablaufdatum. Sortiert nach Ablaufdatum. Gefiltert auf Status=Open
GET	/groceries/v1/groceries/{id}	Gibt Details zu einem Produkt zurück
POST & DELETE	/groceries/v1/groceries/{id}/request	Reservieren von Produkt Reservation Stornieren
GET & PUT & PATCH & DELETE	/groceries/v1/my-groceries/{id}	Erstellen, updaten und löschen von eigenem Produkt
GET	/groceries/v1/my-offered-groceries-unreserved	Alle angebotenen Produkte von eingeloggtem User
GET	/groceries/v1/my-requests	Alle angeforderten Produkte von eingeloggtem User

7.2.1 Neues Grocery erstellen

Endpoint	POST /groceries/v1/groceries/ (Get siehe weiter unten)
View	GroceryListAPIView
Zweck	Dient als Erstellungspunkt für neue Groceries.
Beschreibung	Beim Erstellen eines Lebensmittels wird die perform_create-Methode ausgelöst. Sie prüft, ob der angemeldete Benutzer ein zugehöriges Buddy-Profil hat. Falls ja, wird dieses Buddy-Profil als Spender des neuen Lebensmittels gesetzt. Falls nein, wird ein Validierungsfehler ausgelöst – damit wird ein Buddy-Profil zur Voraussetzung für das Anlegen eines Lebensmittels.
Berechtigungen	Nur angemeldete Benutzer können neue Groceries erstellen.

7.2.2 Zeige alle Groceries mit Pagination und Filter

Endpoint	GET /api/v1/groceries/
View	FilteredGroceryListAPIView
Zweck	Eine umfassende, öffentlich zugängliche Liste aller Lebensmittel mit umfangreichen Filter- und Sortierungsfunktionen.
Beschreibung	Wenn ein GET-Request gestellt wird, gibt die View eine Liste aller Grocery-Objekte zurück. Diese Liste wird durch folgende Funktionen erweitert; **1. Pagination:** Der Output ist aus Performancegründen automatisch paginiert. Standardmässig warden 15 Groceries pro Seite zurückgegeben. In der Response werden die folgenden Informationen zurückgegeben: • count: Die Anzahl Groceries, welche die Suchkriterien erfüllen. • next: Die URL für die nächste Seite mit Suchresultaten (oder null, wenn es die letzte Seite ist.). • previous: Die URL für die vorherige Seite mit Suchresultaten (oder null, wenn es die erste Seite ist.). • results: Das Array der Grocery item Objekten der aktuellen Seite: Verwendung: Erhalte Sucherergebnisse von Seite 2: GET /v1/groceries/?page=2 Wechsle Anzahl Groceries pro Seite zu 20: GET /v1/groceries/?page_size=20 (max 25). **2. Filterung** Die Liste kann mittels Verwendung von query-Parametern in der URL gefiltert werden **Filter nach Status:** Gibt Groceries zurück, bei welchen der Status exakt mit dem Parameter übereinstimmt. • **Parameter:** status • **Beispiel:** GET

	/v1/groceries/?status=open **Filter nach Name:** Gibt Groceries zurück, bei welchen ein Teil des Namens mit dem Parameter übereinstimmt (case-insensitive). • **Parameter:** name • **Beispiel:** GET /v1/groceries/?name=milk (will match "Milk", "Almond milk", etc.) **Filter nach Ablaufdatum(after):** Gibt Groceries zurück, welche frühestens an einem bestimmten Datum ablaufen (Ablaufdatum = Stichdatum oder später). • **Parameter:** expire_after • **Format:** YYYY-MM-DD • **Example:** GET /v1/groceries/?expire_after=2025-07-20 **Filter nach Ablaufdatum(Before):** Gibt Groceries zurück, welche spätestens an einem bestimmten Datum ablaufen (Ablaufdatum = Stichdatum oder früher). • **Parameter:** expire_before • **Format:** YYYY-MM-DD • **Beispiel:** GET /v1/groceries/?expire_before=2025-08-01 **Beispiel mit kombinierten Filtern:** Um alle Groceries mit “Käse” im Name zu finden, welche ab dem 15. Juli 2025 ablaufen: GET /v1/groceries/?status=open&name=cheese&expire_after=2025-07-15
Berechtigungen	Anonymous/ Alle Benutzer(GET) können die Liste mit Groceries sehen und filtern

7.2.3 Erstelle Grocery

Endpoint	POST /api/v1/groceries/
View	FilteredGroceryListAPIView
Zweck	Endpoint für die Erstellung von neuen Groceries
Beschreibung	Ein angemeldeter Benutzer kann ein neues Grocery erstellen, indem er eine POST-Anfrage mit den Lebensmittel-Daten (z. B. Name, Beschreibung, Ablaufdatum) sendet. Die perform_create-Methode stellt folgendes sicher: • Der Benutzer, der die Anfrage stellt, muss ein zugehöriges Buddy-Profil haben. • Das Buddy-Profil des Benutzers wird als Eigentümer mit dem neuen Grocery verknüpft. • Falls kein Buddy-Profil für den Benutzer gefunden wird, wird ein ValidationError ausgelöst, der die Erstellung verhindert.
Berechtigungen	Nur angemeldete Benutzer können neue Groceries erstellen.

7.2.4 Meine Groceries verwalten

Endpoint	GET, PUT/PATCH, DELETE /groceries/v1/my-groceries/<int:pk>/
View	MyGroceryDetailAPIView
Zweck	Anzeigen, Aktualisieren oder Löschen von Grocery, welches mir „gehört“
Beschreibung	• Die Verwaltung der Groceries basiert immer auf einer einzelnen Instanz, identifiziert mittels Primary Key in der URL. Es gibt keine Stapelverarbeitungen. • In der get_object_method wird geprüft, dass der Eigentümer des Groceries mit dem aktuell angemeldeten Benutzer übereinstimmt. Falls der angemeldete Benutzer nicht mit dem Eigentümer übereinstimmt, wirft das System den Error

	„PermissionDenied“. Dadurch verhindert das System, dass Benutzer Groceries aktualisieren oder löschen, welche nicht ihnen gehören.
Berechtigungen	Nur angemeldete Benutzer können Grocery verwalten.

7.2.5 Reserviere ein Grocery

Endpoint	POST /groceries/v1/groceries/<int:pk>/request/ (Reservation stornieren siehe weiter unten)
View	RequestGroceryAPIView
Zweck	Reserviere Grocery
Beschreibung	Angemeldeter Benutzer sendet POST-Request, um Grocery zu reservieren. Business Logik: • **Keine eigenen Groceries:** Das System prüft, dass ein Benutzer keine Groceries reservieren kann, von welchen der angemeldete Benutzer der Eigentümer ist. • **Verfügbarkeits-Check** Das System prüft, dass der Status des Groceries OPEN ist. DRAFT, RESERVED, GONE können nicht reserviert werden. Nach erfolgreicher Reservation aktualisiert das System den Status nach RESERVED und das Grocery wird mit dem reservierenden Benutzer verknüpft.
Berechtigungen	Nur angemeldete Benutzer können Groceries reservieren

7.2.6 Storniere eine Reservation

Endpoint	DELETE /groceries/v1/groceries/<int:pk>/request/
View	RequestGroceryAPIView
Zweck	Lösche bestehende Reservation
Beschreibung	Der Benutzer, welche vorher eine Reservation getätigt hat, sendet einen DELETE-Request, um die Reservation zu stornieren. Business Logik: • **Keine Reservationen von anderen Benutzern stornieren**: Der Benutzer, welche den Request macht muss mit dem Benutzer übereinstimmen, welcher das Grocery vorher reserviert hat. Man kann keine Reservationen von anderen Benutzern stornieren. • **Status = RESERVED**: Der Grocery-Status muss RESERVED sein. Mach erfolgreicher Stornierung setzt das System den Grocery-Status zurück zu OPEN und löscht die Verknüpfung mit dem Benutzer, welcher das Grocery ursprünglich reserviert hat.
Berechtigungen	Nur angemeldete Benutzer können bestehende Reservationen stornieren

7.2.7 Zeige Groceries, welche der angemeldete Benutzer reserviert hat

Endpoint	GET /v1/my-requests/
View	MyRequestedGroceriesListAPIView
Zweck	Stellt eine Liste von Groceries zur Verfügung, welche vom angemeldeten Benutzer reserviert wurden.
Beschreibung	Die Methode get_queryset geht wie folgt vor • Zuerst eine Liste mit Groceries abrufen, bei welchen der aktuelle Benutzer das Grocery reserviert hat. • Das System extrahiert die grocery_ids dieser Groceries

	• Abschliessend wird eine Liste von Groceries zurückgegeben, deren IDs mit den zuvor gefundenen übereinstimmen. So werden dem Benutzer alle Artikel angezeigt, die er reserviert hat.
Berechtigungen	Nur angemeldete Benutzer können eine Liste der reservierten Groceries anzeigen

8 Eingesetzte Technologien

Das Frontend ist mit Angular umgesetzt. Angular bietet eine solide Grundlage für strukturierte Webanwendungen mit klarer Trennung von Komponenten, Datenlogik und UI. Für FridgeBuddy ist Angular besonders geeignet, weil es skalierbar ist. Falls wir in Zukunft fridgeBuddy um weitere Funktionen ergänzen möchten, dann können diese Features schrittweise einbauen, ohne dass die Technologie grundlegend angepasst werden muss. Zudem unterstützt Angular Routing und Typensicherheit direkt. Schlussendlich profitieren wir von einer grossen Community im Angular Ökosystem.

Das Backend wird mit Python umgesetzt, weil im Team bereits “Django Rest Framework” – Wissen vorhanden ist und weil Django “out of the box” sehr viel bietet. Von Django kommt ein Admin-Panel, eine Browsable API fürs Entwickeln und Testen und ein Datenmodell, welches z.B. User bereits beinhaltet.

9 Herausforderungen

9.1 Technische Herausforderungen

Folgende Themen waren herausfordernd und in der Präsentation werden wir kurz darauf eingehen:

- Zusammenspiel von Tailwind 4 und Biome mit Angular
- Angular httpResource in der Praxis
- Wechselnde Anforderungen an die API

9.2 Weitere Herausforderungen

Für mich (Dominique) war es trotz der theoretischen Grundlage im CAS eine Herausforderung, den Einstieg in das Angular-Projekt zu finden. Schon nach ein paar Wochen war das Projekt relativ weit ausgebaut, neuer Code kam laufend dazu. Da mir die Erfahrung in der professionellen Softwareentwicklung fehlt, hatte ich dadurch teilweise Mühe, mit der hohen Pace der erfahreneren Studenten mitzuhalten.

10 Lessons Learned

- Agile Softwareentwicklung auch in Schulprojekt anzuwenden, hat Sinn gemacht. Dadurch haben wir uns aufs Wesentliche fokussiert und die Aufgaben laufend priorisiert.
- Zudem war das Usability Testing sehr aufschlussreich. Die Learnings daraus wurden bereits weiter oben festgehalten.

11 Anhang – Artefakte aus dem Block Interaction Design und Usability

11.1 Miro-Board

In Miro wurden die Drafts für die Mockups gemacht.
https://miro.com/app/board/uXjVIi8QpK0=/

11.2 Mockups in Figma

Figma wurde verwendet, um die Drafts aus Miro klickbar zu machen und darauf basierend Usability Tests durchzuführen.

https://www.figma.com/design/BbgSUgRMYzFJZ372YVaFweWA/FridgeBuddy?node-id=2-7&t=cLiwowTVkVTmrHKu-0

11.3 Dokumentation Usability-Testing

Usability Testing.docx
Aeiou.xlsx