

# fridgeBuddy

Food Saving – Semesterarbeit

- Miro Rutscho
- Dominik Müller
- Luca Brandt

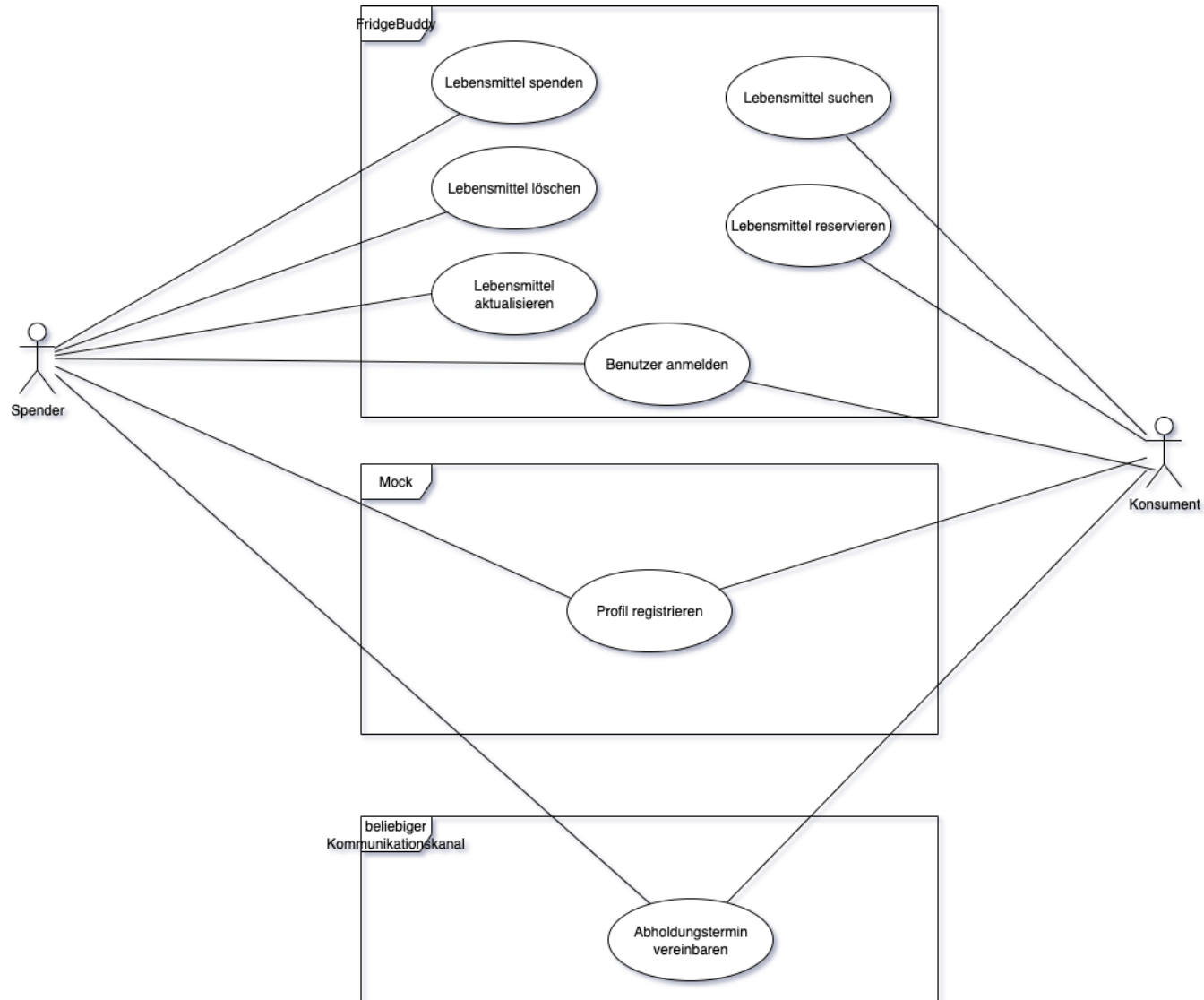
# Agenda

- 🥕 Motivation & Fachlichkeit
- 🥕 Architektur
- 🥕 Projekt-Setup
- 🥕 API-Spezifikation
- 🥕 Admin & Dev Tools Backend
- 🥕 Demo
- 🥕 Routen & Components Frontend
- 🥕 Retro
- 🥕 Code Snippets

# Motivation & Fachlichkeit




- 🥕 Reduzierung von Food Waste
- 🥕 Förderung von günstigen Lebensmitteln
- 🥕 Zielpublikum, sowohl als Anbieter & Konsument:
  - 🥕 Personen mit Bewusstsein für Nachhaltigkeit
  - 🥕 Studenten
  - 🥕 Familien
  - 🥕 Personen mit reduziertem Budget

# Unsere Lösung: fridgeBuddy







# Projekt-Setup

## Backend

-  Django Rest Framework
-  PostgreSQL
-  Deployment via Render

## Frontend

-  Angular
-  Angular Material
-  Tailwind
-  Deployment via Firebase

# API-Spezifikation: Auth & Users

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

# API-Spezifikation: Groceries

Method	URL	Beschreibung
GET & POST	/groceries/api/v1/groceries	Alle Items filterbar nach Name, Status und Auslaufdatum. 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 und unreservieren von Produkt
GET & PUT & PATCH & DELETE	/groceries/v1/my-groceries/{id}	Erstellen, updaten und löschen von eigenem Produkt
GET	/groceries/v1/my-offered-groceries	Alle angebotenen Produkte von eingeloggtem user
GET	/groceries/v1/my-requests	Alle angeforderten Produkt von eingeloggtem user

# Admin & Browseable API

- Browseable API
- Admin Panel
- Allowed Actions



# Allowed Actions

```
def get_allowed_actions(self, obj):
    actions = []
    request = self.context.get("request")

    if not request or not request.user.is_authenticated:
        return actions

    buddy = Buddy.objects.filter(user=request.user).first()
    if not buddy:
        return actions

    if obj.status == Grocery.OPEN and obj.owner != buddy:
        actions.append("reserve")

    elif obj.status == Grocery.RESERVED and obj.requester == buddy:
        actions.append("unreserve")

    if obj.owner == buddy and obj.status != Grocery.RESERVED:
        actions.extend(["update", "delete"])

    if obj.owner == buddy and obj.status == Grocery.RESERVED:
        actions.append("confirm")

    return actions
```

# Route: Login

Header

Login

# Route: Home

Header

Home

GroceryCard

GroceryCard

# Route: Grocery Form

Header

Grocery Form

# Route: Meine Reservationen

Header

MyReservations

GroceryCard

GroceryCard

# Route: Meine Angebote

Header

MyGroceries

GroceryCard

GroceryCard

# Route: Grocery Detail

Header


Grocery Detail

# Demo






# Retro

## Fokus auf Wesentliches

-  nicht sämtliche Eigenschaften eines Produkts modellieren wie Herkunftsland, Ablaufdatum etc. Fokus auf User Flow und Akzeptanz der Lösung

## Nicht für alle einfach «ins Projekt zu finden»

## Learning aus Usability Testing

-  Filterung nach Demeter entfernt
-  Positives Feedback zu Registrierungsprozess
-  Befragung zu «Next Step» nach Produkterfassung

## Tailwind 4 und Biome mit Angular 🥺

## Angular httpResource

## Wechselnde Anforderungen an die API

# Code Snippet

## Lebensmittel reservieren mit httpResource

```
// create separate instances of httpResource
createGroceryDetailsResource(groceryId: Signal<string | undefined>) {
  return httpResource<GroceryData>(() => {
    if (groceryId() === undefined) {
      return undefined;
    }
    return `${BASE_URL}/groceries/v1/groceries/${groceryId()}`;
  });
}

createBuddyDetailsResource(buddyId: Signal<number | undefined>) {
  return httpResource<Buddy>(() => {
    if (buddyId() === undefined) {
      return undefined; // No request will be made
    }
    return `${BASE_URL}/users/v1/user-detail/${buddyId()}`;
  });
}
```

# Code Snippet

## Lebensmittel reservieren mit httpResource

```
// query param injected by withComponentInputBinding()  
// path: 'grocery-details/:groceryId'  
groceryId = input.required<string>();  
groceryDetails = this.groceriesApi.createGroceryDetailsResource(  
    this.groceryId  
);  
  
// as soon as there we have the grocery's owner, we fetch buddyDetails  
buddyId = computed(() => this.groceryDetails.value()?.owner);  
buddyDetails = this.buddyApi.createBuddyDetailsResource(this.buddyId);
```

# Code Snippet

## Lebensmittel reservieren mit httpResource

```
this.reservationService
  .modifyReservation(this.groceryId(), 'reserve')
  .subscribe({
    next: () => {
      // reload details so we get the new status
      this.groceryDetailsResource.reload();

      this.notificationService.showNotification(
        'success',
        'Lebensmittel erfolgreich reserviert!',
        'temporary',
      );
    }
  });
```