Infra-Improver

Ngo-Dinh Sandrine,

Weisskopf Beat

Jegge Yves

Inhaltsverzeichnis

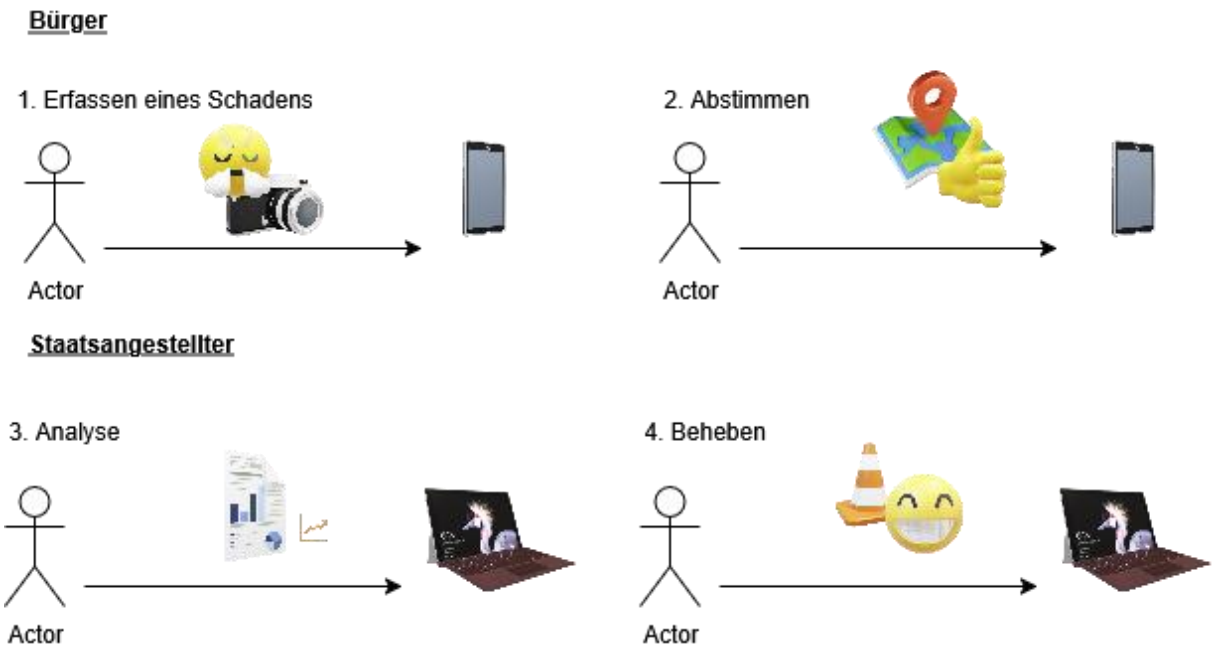
1. Thema

Mobile Website zum Erfassen von Infrastruktur-Problemen wie Schlaglöcher, Defekte an Brückenpfeilern, Bahnhöfen, Schienenabschnitten, blindengerechten und rollstuhlgängigen Bereichen.

2. Umfeld, Ausgangslage

Es gibt viele Infrastruktur-Probleme, die nicht alle im Blickfeld sind. Mit dieser Lösung kann jeder unkompliziert Infrastrukturprobleme melden. Über ein Voting können Prioritätslisten erstellt werden, sodass die dringendsten Anliegen schneller erkannt und behoben werden.

3. Aufgabenstellung & Ziel des Projektes



1.) Bürger, die eine defekte Infrastruktur vorfinden, können mittels eines Fotos, dem aktuellen Standort sowie einer kurzen Beschreibung dies der zuständigen Stelle melden. Dies soll mittels eines mobilen Gerätes möglich sein.
2.) Alle Bürger können die gemeldeten Infrastruktur-Mängel ansehen und diese mittels eines Abstimmungs-Systems priorisieren. Dies soll ebenfalls mittels eines mobilen Gerätes möglich sein.
3.) Die zuständigen Stellen können diverse Statistiken und Filter auf die gemeldeten Mängel anwenden. Dies soll mittels eines normalen Desktop-Computers möglich sein.
4.) Die zuständigen Stellen können die behobenen Infrastruktur-Mängel als gelöst markieren. Dies soll mittels eines normalen Desktop-Computers möglich sein.

4. Scope des Projektes

Features

- **Generell:**
  - User Management wurde nur in vereinfachter Form umgesetzt. Es existiert derzeit keine Rollenverwaltung, sodass beispielsweise die zuständige Stelle keine eigene Rolle besitzt.
  - Die Anwendung ist aktuell ausschliesslich in deutscher Sprache verfügbar. Eine Mehrsprachigkeit ist nicht umgesetzt.
  - Erfasste Mängel können mittels Ortssuche gefunden werden.

- **Erfassen eines Schadens:**
  - Nutzer können einen kurzen, prägnanten Titel für den Mangel erfassen.
  - Es soll möglich sein, eine detaillierte Beschreibung des Mangels zu verfassen, um weitere Informationen bereitzustellen.
  - Mängel können einer vordefinierten Kategorie zugeordnet werden (z. B. Strasse, öffentlicher Verkehr, Brücke, Spielplatz, fehlende Barrierefreiheit, Veloverkehr, Fussverkehr, Andere).
  - Nutzer können ein oder mehrere Fotos hochladen, um den Mangel zu dokumentieren.
  - Die Position des Mangels kann automatisch über die Web-API (z. B. Geolocation des Geräts), über die Metadaten (EXIF-Daten) der hochgeladenen Fotos ermittelt oder mittels Ortssuche eingetragen werden.
  - Erfasste Mängel können vom Ersteller bearbeitet bzw. gelöscht werden.

- **Abstimmen:**
  - Die erfassten Mängel werden in einer Karte dargestellt.
  - Erfasste Mängel, die sich um den Standort vom Bürger befinden, werden angezeigt.
  - Die Details eines erfassten Mangels können angesehen werden.
  - Die erfassten Mängel können mittels eines Abstimmungssystems priorisiert werden.

- **Analyse:**
  - Torten-Diagramm mit Kategorien von allen erfassten Mängeln
  - Linien-Diagramm mit Bearbeitungszuständen
  - Tabellenübersicht mit allen erfassten Mängeln
  - Mängel können über die Tabellenübersicht auf Karte angesehen werden

- **Beheben**
  - In der Analysenansicht kann in der Tabelle der Bearbeitungsstatus geändert werden.

Zukünftige Features

- **Generell:**
  - User Management mit Rollenverwaltung, so dass nur eingeloggte zuständige Stellen die Analyseansicht sehen können.
  - Nicht eingeloggte User erlauben, die keine Bearbeitungs- und Abstimmungsrechte besitzen.
  - Mehrsprachigkeit (Englisch, Italienisch, Französisch)
  - Kantonsgrenzen einblenden.
  - Zuständige Stellen können nur Mängel sehen, die für sie relevant sind.
  - End-To-End-Testing und Unit Tests hinzufügen
  - Admin-UI für Konfigurationen wie das Erstellen von Kategorien.

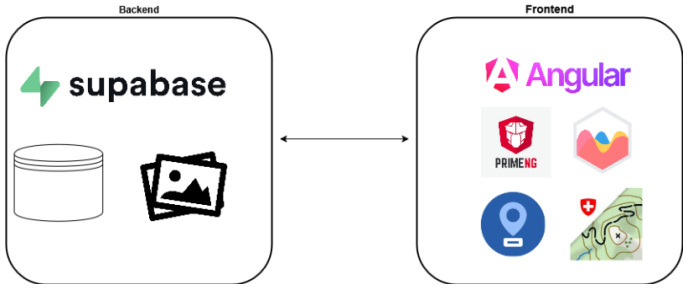
- **Erfassen eines Schadens:**
  - Kommentarfunktion hinzufügen

- **Beheben**:
  - Es soll möglich sein, eine Statistik für die Bearbeitungszeiten darzustellen.

5. Systemübersicht, Grob-Architektur, Deployment

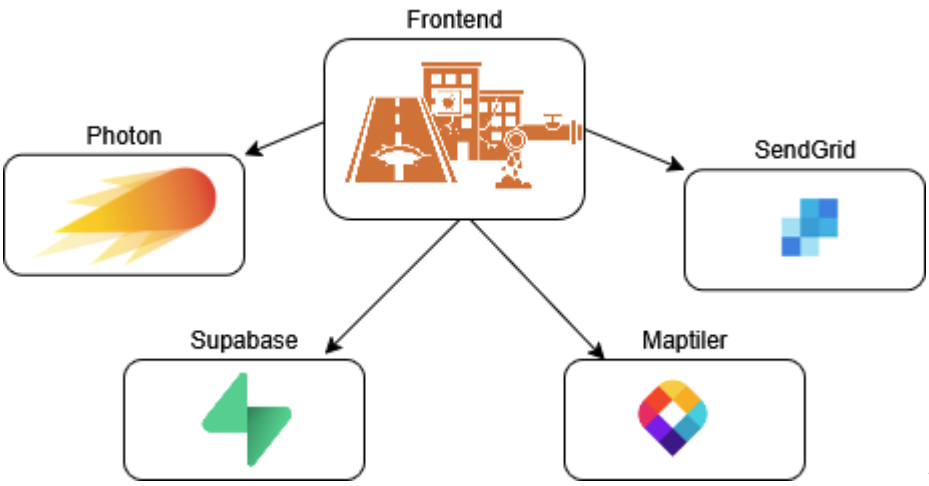
Grob-Architektur

Als Backend wird Supabase verwendet und als Frontend Angular.



Umsysteme

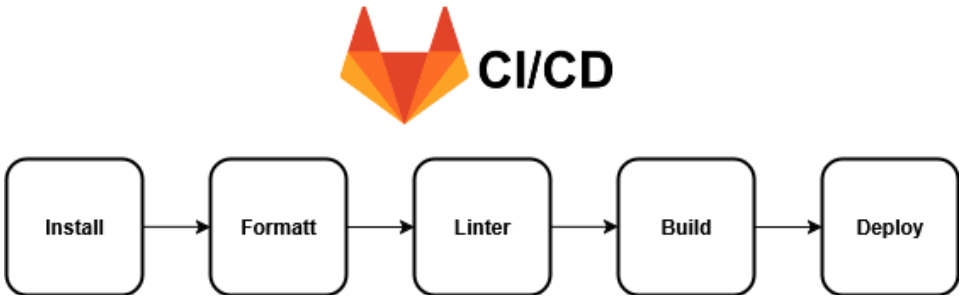
- Supabase Backend-Storage
- GitLab Pages: für Bereitstellung der Applikation aus dem GitLab-Repository
- Maptiler: für die Bereitstellung des Kartenmaterials (SwissTopo, OpenStreetMap), die in der Angular-Applikation für die Mängel-Anzeige verwendet werden
- SendGrid für das Versenden von Emails
- Photon: GeoCoding API – transformiert Koordinaten in Standortdaten und umgekehrt.



^

Deployment

Das Backend wird über die Supabase Cloud deployed, während das Frontend via GitLab Pages gehostet wird. Um das Frontend zu deployen, wurde eine GitLab CI/CD Pipeline mit fünf Schritten erstellt.



Die Pipeline sorgt dafür, dass die Angular-App bei jedem Merge Request folgende Schritte automatisch ausführt:

1. Installation von externen Dependencies
2. Gewährleistung einer einheitlichen Code-Formatierung.
3. Überprüfung von Linting-Regeln
4. Building des Frontend
5. Veröffentlichung auf GitLab Pages und Erstellung eines Git-Tags. Dies wird jedoch erst nach einem erfolgreichem merge durchgeführt

6. Eingesetzte Technologien

Backend

- **Supabase PostgreSQL**:
  - *Beschreibung*: Vollwertige relationale Datenbank, gut für strukturierte Daten wie Mängel und Abstimmungen.
  - *Technologienwahl*: Einfaches Management, Skalierbarkeit, unterstützt komplexe Queries.

- **PostGIS Extension**:
  - *Beschreibung*: Geodaten-Unterstützung in PostgreSQL.
  - *Technologienwahl*: Ermöglicht räumliche Abfragen wie „alle Meldungen in View“ oder „alle Meldungen in einem Umkreis“

- **Supabase Auth**:
  - *Beschreibung*: Benutzer-Authentifizierung & -Autorisierung
  - *Technologienwahl*: in Supabase-Framework enthalten

- **Supabase Storage**:
  - *Beschreibung*: Speicherung für Fotos.
  - *Technologienwahl*: in Supabase-Framework enthalten

- **Supabase RPC (Remote Procedure Calls)**:
  - *Beschreibung*: Serverseitige Funktionen direkt in PostgreSQL.
  - *Technologienwahl*: Logik wie “Vote erhöhen”, “Defect in View” effizient serverseitig ausführen.

- **Maptiler:**

  - *Beschreibung*: Kartenmaterial-Provider
  - *Technologienwahl*: Unterschiedliches Kartenmaterial von SwissTopo, OpenStreetMap etc. Kostenloses Kontingent für kleine Apps.

- **Photon:**

  - *Beschreibung*: Umwandlung von GPS-Koordinaten in Adressen und umgekehrt.
  - *Technologienwahl*: Meldungen können automatisch mit einer Adresse angezeigt werden, ohne dass Nutzer die Adresse manuell eingeben müssen. Autocomplete wird auch unterstützt.

- **SendGrid:**

  - *Beschreibung*: Ermöglicht das Versenden von E-Mails.
  - *Technologienwahl*: Leider bietet Supabase keinen E-Mail-Service an, deshalb musste eine Alternative gefunden werden.

Frontend

- **Angular:**
  - *Beschreibung*: Single-Page-App (SPA) Framework
  - *Technologienwahl*: Interesse, weil es weltweit eingesetzt ist und zum Teil in unseren Firmen eingesetzt wird.

- **Supabase JS:**
  - *Beschreibung*: JavaScript Library für den direkten Supabase-Zugriff.
  - *Technologienwahl*: Weil wir Supabase verwenden.

- **PrimeNG:**
  - *Beschreibung*: Fertige UI-Komponenten wie Tabellen, Buttons, Dialoge.
  - *Technologienwahl*: Bietet mehr Komponenten an als z.B. Angular Material. Hat auch eine gute Dokumentation und unterstützt Tailwind CSS, was das Styling auch sehr vereinfacht.

- **Chart.js:**
  - *Beschreibung*: Visualisierung von Daten wie Vote-Statistiken.
  - *Technologienwahl*: Mit PrimeNg kompatible interaktive, aber verbesserungswürdige, Diagramm-Bibliothek ohne viel Eigenentwicklung.

- **Tailwind CSS:**
  - *Beschreibung*: Flexibles, modernes CSS-Framework.
  - *Technologienwahl*: Von PrimeNG empfohlenes Styling Framework, mit dem man responsive Designs leichter umsetzen kann.

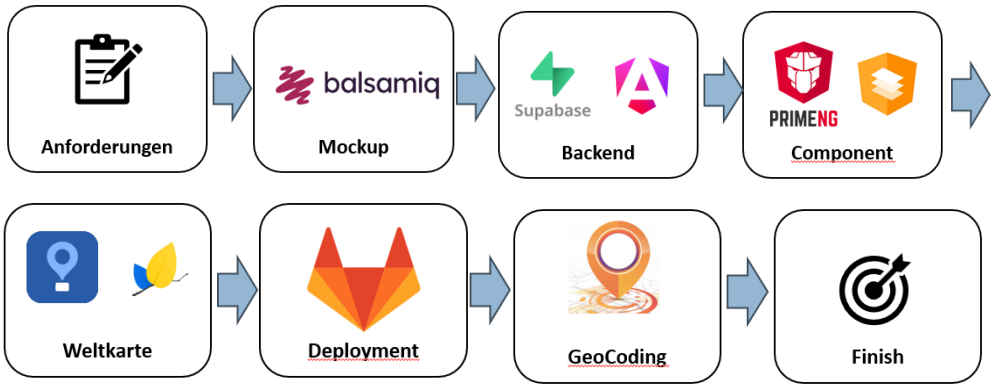
- **ExifReader:**
  - *Beschreibung*: GPS-Daten aus Fotos auslesen.
  - *Technologienwahl*: Defekte mit Foto können automatisch Standort enthalten, Benutzerfreundlichkeit steigt.

- **maplibre & ngx-maplibre:**
  - *Beschreibung*: Open-Source-Kartenbibliothek für Angular.
  - *Technologienwahl*: Vektor-basiertes Kartenmaterial und interaktive Karten, Marker – perfekt für Geo-Visualisierung von Defekten; viele Möglichkeiten

7. Design und Umsetzung

Development Prozess

Die Entwicklung des Infra-Improver ging wie folgt von statten:



Anforderungen

- Wurden im Team erstellt und während der Umsetzung angepasst. Anforderungen wurden klassifiziert in must-haves und nice-to-haves.

Balsamiq

- Ein Mockup wurde mit Balsamiq erstellt und Testbenutzern vorgestellt. Diese haben uns Feedback gegeben, die in die Applikation eingeflossen sind.

Backend

- Eine einfache Angular-Applikation mit Authentifizierung mit Supabase-Datenbank und BLOB-Storage wurde mithilfe von Tutorials erstellt und als Grundlage für die weitere Entwicklung genommen.

Component

- Zudem wurde mit Angular Material und PrimeNG experimentiert.

Weltkarte

- Bei der Karte haben wir mit leaflet begonnen und uns letztlich für das vektorbasierte map-libre entschieden, da die Karten damit schneller geladen wurden.

Deployment

- Eine Deployment-Pipeline wurde erstellt, die auch auf GitLab-Pages deployed hat. Dies hat auch geholfen, die Applikation auf verschiedenen Geräten zu testen.

GeoCoding

- Einbindung von Features wie die Ortssuche und der Ortsrückwärtssuche (von Koordinaten zu einem Standort) wurde unter Verwendung der photon API umgesetzt.

Iterative Weiterentwicklung

- Umsetzung in kleinen Schritten mit Tests und Feedback im Team.
- Häufige Anpassungen am UI und Datenmodell.
- Umsetzung von chart.js Grafiken
- Diverse Verbesserungen an der Weltkarte (Marker, flyToDefect, Sateliten View)

Projekt-Aufbau und -Strukturierung

Die Infra-Improver-Projekt-Struktur hat sich an einer mittelgrossen Projekt-Struktur wie in (medium.com, 2025) orientiert. Diese sieht wie folgt aus:

```
/src
–– app
  |   ├–– core
  |   ├––  ├–– models
  |   ├––  ├––  ├–– ...
  |   |   └–– ...
  |   ├–– shared
  |   |   ├–– components
  |   |   |   └–– ...
  |   ├–– features
  |   |   └–– ...
  |   ├–– app.ts
  |   └–– app.routes.ts
  –– assets/
  –– environments/
  –– main.ts
  –– index.html
  –– styles.css
```

Routing

Das Angular Frontend Routing wurde relativ einfach gehalten und sieht wie folgt aus:

```
/
├──login
|   ├── signup
|   ├── restore
├── home
├── create-defect
├── analyse-defects
```

Services

Services sorgen für eine klare Trennung von Businesslogik und Applikationslogik.

Service	**Funktion**
Supabase-Service	Kommunikation mit Supabase-Backend: Authentifizierung, Datenbank Zugriff, RPC-Aufrufe, Storage
Location-Service	Geolocation API zur Standortermittlung, Ermittlung von GPS-Daten aus Foto-Metadaten
Photon-Service	Für Ortssuche und Reverse-Geocoding
Toast-Service	Für standardisierte PrimeNg Toast-Benachrichtigungen

Features

Die Haupt-Features wurden in eigene Komponenten ausgelagert.

Komponente	**Funktion**
user-authentication	Registrierung, Login, Passwort-Reset
show-defects	Interaktive Karte mit allen Mängeln in einem Bereich
create-defect	Formular zum Erstellen eines Mangels
analyse-defects	Diagramme, Statistiken und Tabellen

Shared

Shared Komponenten sind Komponenten, die mehr als einmal verwendet werden oder in Zukunft in anderen Komponenten Verwendung finden könnten.

Komponente	**Funktion**
navigation	Navigation zwischen home, create-defect, analyse-defects
photo-upload	Upload von Fotos (inkl. GPS-Bestimmung)
photos-viewer	Anzeige von hochgeladenen Fotos
search-location	Standortsuche
street-map	Karte zur Anzeige von Defekten
street-map-drawer	Drawer, der über die Karte angezeigt wird

Supabase Datenbank

Es wurden zwei Tabellen defects und votes erstellt. Sowie eine Email_queue für das Versenden von E-Mails bei Änderung des Bearbeitungsstatus.

Defects

Name	**Type**	**Beschreibung**
id	uuid	Identifier vom Mangel
title	text	Titel vom Mangel
category	Categories	Enum types von Kategorien, z.B. Fussverkehr
description	text	Beschreibung vom Mangel
state	States	Bearbeitungsstatus vom Mangel
numberOfVotes	int8	Anzahl von Votes für den Mangel
createdAt	timestamptz	Timestamp, wann der Mangel erstellt wurde
updatedAt	timestamptz	Timestamp, wann der Mangel bearbeitet wurde
location	geometry	Enthält Standortdaten vom Mangel
user_id	uuid	User, der den Mangel erstellt hat

Votes

Tabelle wurde erstellt, um zu ermitteln, ob ein User bereits für einen Mangel abgestimmt hat.

Name	**Type**	**Beschreibung**
id	int4	Identifier vom Vote/Abstimmung
user_id	uuid	User, der abgestimmt hat
defect_id	uuid	Mangel, für den abgestimmt wurde

Email_Queue

Tabelle wurde erstellt für das Senden von Emails bei der Änderung des Bearbeitungsstatus.

Name	**Type**	**Beschreibung**
id	int4	Identifier vom Vote/Abstimmung
defect_id	uuid	Mangel, bei der der Bearbeitungsstatus geändert wurde
user_id	uuid	User, der den Mangel erstellt hat
subject	text	Betrefftext
body	text	Text, der E-Mail
sent	bool	Zeigt an, ob die E-Mail schon versendet wurde
created_at	timestamp	Wann der Eintrag erstellt wurde
email	text	E-Mail-Adresse, an den die E-Mail versendet werden soll

Supabase Storage/Bucket

Fotos werden im Supabase Storage mit folgender Struktur abgespeichert:

```
–– defects
  |   ├–– defect_id
  |   ├––  ├–– photo1.jpg
  |   ├––  ├–– ...
```

Supabase Remote Procedure Call

Name	Beschreibung
defects_in_view	Liefert alle Mängel, die sich innerhalb eines Rechtecks befinden zurück.
get_defects_in_radius	Liefert alle Mängel, deren Standort innerhalb eines angegebenen Radius um einen bestimmten Mittelpunkt liegt.
get_enum_values	Gibt Enum-Werte zurück (für States, Categories)

Supabase Trigger

Name	Beschreibung
handle_votes_change	Aktualisiert die numberOfVotes in der defects-Tabelle – Bei einem Upvote wird numberOfVotes um eins erhöht, bei einem Downvote um eins vermindert.
defect_state_change_trigger	Prüft, ob sich der Status eines Mangels geändert hat, und legt in diesem Fall einen neuen Eintrag in der E-Mail-Warteschlange mit Betreff und Nachricht für den betroffenen Benutzer an.

8. Herausforderungen

Map-Entwicklung:

- Die Integration von Kartenfunktionen erwies sich komplexer als erwartet, z.B. MapLibre bietet viele Funktionalitäten, aber die Dokumentation ist verbesserungswürdig.
- Es wurde auch Leaflet getestet, aber das Laden des Kartenmaterials war sehr langsam, da es imagebasiert ist, sowie war die Funktionalität eingeschränkt. MapLibre ist vektorbasiert, wodurch es schneller ist.

Responsives Design:

- PrimeNG bietet eine grosse Komponenten-Auswahl, jedoch war die Anpassung für mobile Endgeräte sehr schwierig.

Gerätevielfalt:

- Unterschiedliche Browser- und Geräteverhalten (vor allem bei Geolocation und Foto-Upload) sind aufwändig zum Handhaben. Aus diesem Grund mussten mehrere Möglichkeiten für die Standortbestimmung eines Mangels implementiert werden.

Mockup vs. Realität:

- Nicht jedes Mockup-Screen liess sich wie geplant implementieren.
- Flexibel sein.

CORS Policy beim Reverse GeoCoding:

- Browser-Sicherheitsbeschränkungen machten zusätzliche Massnahmen notwendig.
- Edge Function wurde dazu implementiert, um dies zu umgehen.

Datenflüsse:

- Die Parent-Komponente soll die States der Applikation verwalten. Die Kinder-Komponenten dürfen diese nicht manipulieren.
- Herausfordernd war es diese richtig zu verwalten, vor allem bei einem Refactoring von den Komponenten.

9. Lessons Learned

Library-Analyse

- Sich mehr Zeit für die Evaluierung von Libraries, z.B. Component oder Weltkarten nehmen, da es schwierig eine Entscheidung zu treffen, da es eine sehr grosse Auswahl gibt. Ausserdem ist ein Technologiewechsel sehr zeitaufwändig.

Mockups mit Balsamiq

- Erste Entwürfe der Benutzeroberfläche wurden in Balsamiq erstellt, was sich als nützlich erwies, da dadurch Probleme wie z.B. “Logout-Button war zu prominent und so zu verlockend zu drücken” entdeckt werden konnten. Feedback konnte dementsprechend eingearbeitet werden.
- Obwohl wir es für ein mobiles Gerät gemockt haben, konnten wir es am Ende so nicht implementieren, da es so nicht bedienbar gewesen wäre. Unser Mockup beinhaltete zu viele übereinanderliegende Widgets.

Deployment

- Frühes Erstellen einer Build-Pipeline mit Linter und Formatter hat sich bewährt. Der Linter hat auf inkonsistenten sowie deprecated Code, z.B. ngModule, *ngIf hingewiesen.
- Das frühzeitige Deployment hat auch sehr geholfen, die Applikation auf verschiedenen Geräten zu testen sowie von verschiedenen Testusern testen zu lassen.

Responsives Styling

- Zeitaufwand darf man nicht unterschätzen, vor allem auf mobilen Geräten ist das Styling sehr herausfordernd. Tailwind CSS hat das Styling massiv erleichtert vor allem bei nicht vorhandenen CSS-Kenntnissen.

Typing

- Es ist nicht so einfach, die Types zwischen Frontend und Backend konsistent zu halten. Es wäre eventuell sinnvoll beim nächsten Mal Libraries zu verwenden, die helfen solche Probleme zu finden (z.B. Zod).

Was hat gut funktioniert?

- **Linting**: Das automatische Linting hat nicht nur für konsistente Code-Formatierung gesorgt, sondern auch beim Erkennen von Fehlern unterstützt. Zusätzlich konnten damit obsoleter oder unnötiger Code frühzeitig identifiziert und entfernt werden.

- **Mockups (Balsamiq)**: Die Erstellung von Mockups war sehr hilfreich, um frühzeitig eine klare Vorstellung der Benutzeroberfläche zu bekommen, auch wenn sie mit der Zeit modifiziert wurde. Die Erstellung der Mockups war nicht nur für das Team hilfreich, sondern auch, um Feedback von Testnutzern zu erhalten. Da die Darstellung bewusst einfach gehalten ist, konnten Personen ohne technisches Vorwissen Rückmeldungen zur Benutzerführung und Verständlichkeit geben. Dadurch liess sich frühzeitig erkennen, ob die UI logisch und intuitiv aufgebaut ist.

- **Supabase-Integration**: Die Einbindung von Supabase in das Projekt funktionierte reibungslos. Besonders positiv war die einfache Einrichtung der Authentifizierung und Datenbankfunktionen, wodurch schnell ein erster funktionierender Prototyp erstellt werden konnte.

- **PrimeNG**: Die Nutzung von PrimeNG ermöglichte eine schnelle Umsetzung benutzerfreundlicher UI-Komponenten (z.B. Foto-Upload, Drawer). Dadurch konnten wir viel Zeit sparen.

- **Tailwind CSS**: Mit Tailwind war es möglich, das Styling schnell anzupassen. Dadurch konnten die Standardkomponenten von PrimeNG an das gewünschte Design angepasst werden, ohne aufwändige Überschreibungen im CSS/SCSS vorzunehmen. Zudem hat sich die Kombination aus PrimeNG und-Tailwind als sehr effektiv erwiesen – PrimeNG stellte eine solide Basis an Komponenten bereit, während Tailwind die Feinanpassung des Designs vereinfachte.