Projekt Spezifikation Infra-Improver

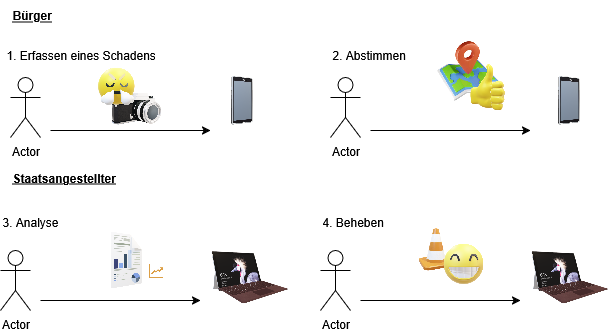
# Thema

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

# 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 werden.

# Aufgabenstellung & Ziel des Projektes



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

# Scope des Projektes

## Projekt Ziele (Must Haves)

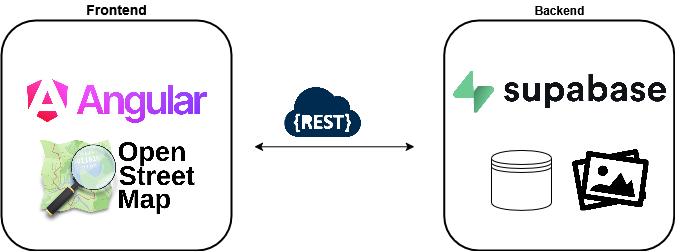
* **Generell:**
  + User Management wird nur sehr vereinfacht implementiert.
  + Die Benutzersprache ist Deutsch
* **Erfassen eines Schadens:**
  + Es soll möglich sein, ein Titel zu verfassen.
  + Es soll möglich sein, eine Textbeschreibung zu erfassen.
  + Es soll möglich sein, die erfassten Probleme zu kategorisieren z.B. Schlaglöcher, Bahnhöfe, Brückenpfeilern, etc.
  + Es soll möglich sein, ein oder mehrere Fotos hochzuladen.
  + Es soll möglich sein, die Position des Standortes via WebApi oder Foto Metadaten zu ermitteln.
* **Abstimmen:**
  + Es soll möglich sein, die erfassten Schäden in einer Karte darzustellen.
  + Schäden, die sich um den Standort vom Bürger befinden, werden angezeigt.
  + Es soll möglich sein, die erfassten Details eines Schadens anzusehen.
  + Es soll möglich sein, die erfassten Schäden mittels eines Priorisierungssystems zu bewerten.
* **Analyse:**
  + Es soll möglich sein, eine ganz simple Auswertung anhand der Abstimmungen zu erstellen.

## Optionale Ziele (Nice to Haves)

* **Generell:**
  + Nur read only Ansichten, falls Bürger nicht eingeloggt ist.
  + Es soll möglich sein, dass nur eingeloggte Bürger Schäden erfassen, editieren oder priorisieren können.
  + Es soll möglich sein, dass nur eingeloggte Staatsangestellte Schäden analysieren oder beheben können.
* **Erfassen eines Schadens:**
  + Es soll möglich sein, den erfassten Schaden zu bearbeiten.
  + Es soll möglich sein, ein Diskussions-Kommentar zu erfassen.
* **Analyse:**
  + Es soll möglich sein, Statistiken für unterschiedliche Kategorien darzustellen.
  + Es soll möglich sein, Statistiken für Erfassungsdaten darzustellen.
  + Erfasste Schäden können mittels Ortssuche gefunden werden.
* **Beheben:**
  + Es soll möglich sein, die erfassten Schäden als behoben zu markieren.
  + Es soll möglich sein, eine Statistik für die Bearbeitungszeiten darzustellen.

# Grob-Architektur und Design

## Architektur und Technologien

****

|  |  |
| --- | --- |
| **Backend** | Supabase: → Simples Backend, PostGIS ist integriert   * Supabase Storage * Supabase Auth * Supabase RPC * PostgreSQL * PostGIS: → eignet sich gut für Geolocation * Reverse Geocoding → um aus Koordinaten Ortsdaten zu ermitteln |
| **Frontend** | Angular: → Schweizweit etabliert   * PrimeNG → etabliertes Design, umfangreiche Komponenten * Supabase JS * OpenStreetMap – freie Alternative zu GoogleMaps * Web APIs: Geolocation API, File API * D3.js → etablierte Grafik-Lib |

# Screens-Flow

Siehe angehängtes interaktives PDF.

# Backend

## Supabase Datenbank

Die Schadenmeldungen sollen in die Supabase PostgreSQL Datenbank abgespeichert werden. Eine Schadensmeldung enthält die folgenden Daten:

{  
 "defect": {  
 "id": 1,  
 "user": "Max Muser",  
 "title": "",  
 "category": "damage",  
 "description": "This is a mess",  
 "state": "open/closed/deleted",  
 "numberOfVotes": 3,  
 "createdAt": "10.10.2025",  
 "updatedAt": "12.10.2025",  
 "location": {  
 "longitude": 1,  
 "latitude": 2  
 },  
 "photos":[  
 "photo\_id1",  
 "photo\_id2"  
 ]  
 }

* *category* und *state* sind als PostgreSQL enums definiert.
* Die *location* wird als PostGIS geometry object definiert
* Die *photos* werden als link zum bucket abgespeichert.

## Supabase Remote Procedure Call

Um die umliegenden Schadenmeldungen eines Standortes zu ermitteln, werden Supabase RPC Funktionen eingesetzt.

**SQL:**

create or replace function nearby\_stores(lat float, long float)

returns table (id public.stores.id%TYPE, name public.stores.name%TYPE, description public.stores.description%TYPE, lat float, long float, dist\_meters float)

language sql  
 as $$  
 select id, name, description, st\_y(location::geometry) as lat,

st\_x(location::geometry) as long, st\_distance(location, st\_point

(long,lat)::geography) as dist\_meters  
 from public.stores  
 order by location <-> st\_point(long, lat)::geography;

$$;

**JavaScipt:**

// Get all places with calculated distance$  
async getNearbyStores(lat: number, long: number) {  
const { data, error } = await this.supabase.rpc('nearby\_stores', {  
 lat,  
 long,  
 })  
 return data  
 }

## Supabase Storage

Die aufgenommenen Fotos werden in einem Supabase Bucket Storage abgelegt und mittels identifier mit der PostgreSQL Datenbank verlinkt.

## Supabase API

Zugriff via Supabase REST API via Supase-JS.

return this.supabase  
.from(defects)  
.select(`\*`)  
.eq('id', user.id)  
.single()

# Offene Fragen

Deployment der Frontend Anwendung.

# Studierende

Ngo-Dinh Sandrine

Weisskopf Beat

Jegge Yves