· FloodSafe Project Guide

Welcome! This guide is designed to help you understand the **FloodSafe** codebase, its architecture, and the vision behind it. Whether you are a student or a contributor, this document will walk you through everything from the folder structure to the future roadmap.

·· 1. Understanding the Codebase

We use a **Monorepo** structure. This means all our different projects (Backend, Frontend, IoT) live in one single repository. This makes it easier to share code and run everything together.

· Directory Structure

```
FloodSafe/
··· apps/                  # Where the actual code lives
·   ··· backend/           # The Brain (Python/FastAPI)
·   ··· frontend/          # The Face (React/Vite)
·   ··· iot-ingestion/     # The Ears (Python/IoT)
··· docker-compose.yml     # The Orchestra Conductor
··· pnpm-workspace.yaml    # The Glue (links everything)
```

· The Backend (`apps/backend`)

Tech: Python, FastAPI, SQLAlchemy, PostGIS. **Pattern**: **Clean Architecture**.

- **Why?** We separate the "Business Logic" (Domain) from the "Database" (Infrastructure). This means if we change the database later, our core logic doesn't break.
- **Key Files**:
  - `src/domain/models.py`: Defines what a "User" or "Report" *is* (Pydantic models).
  - `src/api/reports.py`: Handles the logic when someone uploads a flood report (including Geotag verification!).

· The Frontend (`apps/frontend`)

Tech: React, TypeScript, MapLibre GL. **Key Feature**: **Vector Maps**.

- **Why?** Instead of loading heavy images for the map, we use "Vector Tiles" (`.pmtiles`). This makes the map fast and zoomable without needing a generic Google Maps API key.
- **Key Files**:
  - `src/lib/map/useMap.ts`: The hook that loads the map and handles the layers.

· IoT Ingestion (`apps/iot-ingestion`)

Tech: Python (FastAPI). **Purpose**: To listen to thousands of sensors.

- **Why separate?** If 10,000 sensors send data at once, we don't want the main website to slow down. This service handles the noise.

· 2. Current Features (MVP)

This is what works **right now**:

1. **Interactive Flood Map**:
   - You can see a detailed map of the city (Bangalore/Delhi) with flood zones.
2. **Geotagged Reporting**:
   - Users can upload a photo of a flood.
   - **Smart Check**: The backend reads the GPS data (EXIF) from the photo to verify if it was actually taken at that location.
3. **Real-time Sensor Data**:
   - The system can receive and display water level data from IoT sensors.
4. **SOS Webhook**:
   - A placeholder endpoint exists to receive WhatsApp Location SOS messages.

· 3. The Mobile App Path

We designed this system to be **Mobile-First**. Here is how we turn this into an app:

Strategy: "Capacitor" (The Fast Way)

We don't need to rewrite the code. We can use a tool called **Capacitor** to wrap our React website into a native Android/iOS app.

1. **Install Capacitor** in `apps/frontend`.
2. **Build**: It takes the web code and puts it inside a native container.
3. **Deploy**: You get an `.apk` file to install on your phone.

Why this is cool:

- The **Backend** doesn't care if it's a web or mobile app. It just serves JSON.
- We can access native features like **Push Notifications** and **Background Geolocation** using Capacitor plugins.

· 4. Production Features & Gaps

These are the "Big League" features we have architected but haven't fully built yet.

· Gaps (For You to Build)

1. **Video Reporting (Deferred)**
   - **The Goal**: Allow users to upload videos.
   - **The Gap**: Storing large videos on a server is hard. We need to connect to **AWS S3** or **Google Cloud Storage** to handle this efficiently.
   - **Code Prep**: The `Report` model already has `media_type="video"` ready to go !
2. **AI Prediction (Prophet)**

- **The Goal**: Predict floods 2 hours before they happen.
- **The Gap**: We need historical data to train the AI.
- **Code Prep**: We created an interface `IPredictionService`. You just need to write the code that fills in the blanks.

3 **WhatsApp SOS Integration**

- **The Goal**: Real-time SOS via WhatsApp.
- **The Gap**: We need a **Twilio** account and WhatsApp Business approval.
- **Code Prep**: The endpoint `/api/webhooks/whatsapp` is already written and waiting for the connection.

· Final Advice for Students

- **Start Small**: Try to run the `docker-compose up` command and get the map showing.
- **Explore**: Look at `apps/backend/src/domain/models.py`. Try adding a new field to the `User` model.
- **Break Things**: It's the best way to learn. If the map stops loading, check the browser console!

Happy Coding! ·