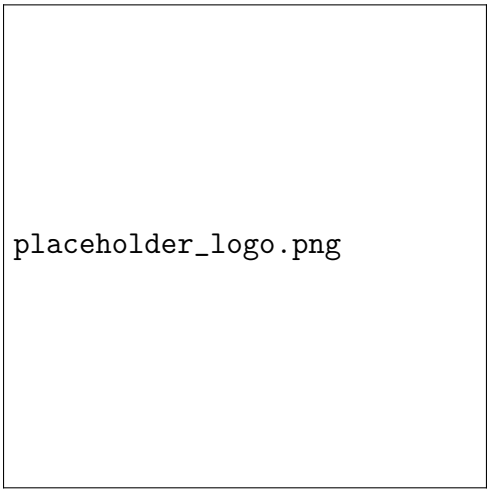
CrisisEye

Product Requirements Document

Automatic Flood Detection & Damage Assessment System



Hackathon: AI między orbitami

Data: January 10, 2026

Wersja: 2.1 (Poprawiona)

Status: Zaakceptowany

Product Owner: Zespół CrisisEye
Lead Developer: [Imię i nazwisko]
Data Scientist: [Imię i nazwisko]
UX Designer: [Imię i nazwisko]

Automatyczne wykrywanie powodzi z danych radarowych SAR i szacowanie strat w czasie rzeczywistym

Contents

1 Wizja Produktu

1.1 Cel Główny

CrisisEye to aplikacja webowa wykorzystująca dane radarowe (Sentinel-1 SAR) do automatycznego wykrywania obszarów powodziowych w czasie rzeczywistym, niezależnie od warunków atmosferycznych. System integruje dane satelitarne z mapami budynków, umożliwiając szybkie szacowanie strat finansowych i generowanie raportów operacyjnych.

1.2 Problem Statement

Podczas katastrof naturalnych:

- Służby ratownicze nie mają **dostępu do aktualnych danych** z powodu zachmurzenia
- Ocena zniszczeń opiera się na **pojedynczych zgłoszeniach**, a nie na pełnym obrazie
- Szacowanie strat odbywa się z **opóźnieniem tygodni/miesięcy**
- Dane optyczne (Sentinel-2) są **bezużyteczne przy zachmurzeniu**

1.3 Rozwiązanie

- **Wykrywanie powodzi w czasie rzeczywistym** z wykorzystaniem radaru SAR
- **Automatyczne szacowanie strat** poprzez integrację z OpenStreetMap
- **Generator raportów** PDF dla służb i samorządów
- **Dark mode interface** z efektywną wizualizacją danych

2 Wymagania Funkcjonalne

2.1 Główne Funkcjonalności

ID	Opis funkcjonalności	Priorytet
F1	Panel mapy interaktywnej z warstwami SAR i maską zalania	Must-have
F2	Automatyczne pobieranie i przetwarzanie danych Sentinel-1	Must-have
F3	Change detection między sceną "przed" i "w trakcie" powodzi	Must-have
F4	Integracja z OpenStreetMap (budynki)	Must-have
F5	Kalkulator strat finansowych	Should-have
F6	Generator raportów PDF	Should-have
F7	System powiadomień i alertów	Could-have
F8	Dashboard z statystykami	Could-have

2.2 Opis Szczegółowy

2.2.1 F1: Panel Mapy Interaktywnej

- Mapa oparta na OpenStreetMap z możliwością zoom i pan

- Dostępne warstwy:
  - Sentinel-1 (VH/VV polaryzacja)
  - Maska zalania (półprzezroczysty niebieski)
  - Budynki z OSM (kolorowane według poziomu zalania)
  - Infrastruktura krytyczna
- Suwak czasu do przeglądania zmian
- Narzędzia pomiarowe (odległość, powierzchnia)

2.2.2 F2: Przetwarzanie Danych SAR

- Automatyczne pobieranie z CREODIAS/Google Earth Engine
- Preprocessing: kalibracja, odszumianie, korekcja terenu
- Generowanie produktów pośrednich (sigma0, gamma0)

2.2.3 F3: Change Detection

```python
def detect_flood_change(current_scene, reference_scene, threshold=-3):
    """
    Wykrywanie zmian poprzez log-ratio
    current_scene: aktualna scena SAR (sigma0 w dB)
    reference_scene: scena referencyjna (mediana z okresu przed)
    threshold: próg dla wykrycia wody (w dB)
    """
    # Oblicz log-ratio
    log_ratio = 10 * np.log10(current_scene / reference_scene)

    # Wykryj wodę (ujemne wartości wskazują na wodę)
    water_mask = log_ratio < threshold

    return water_mask, log_ratio
```

2.2.4 F4: Integracja OSM

```python
import requests

def get_osm_buildings(bbox):
    """
    Pobiera budynki z OpenStreetMap dla danego bbox
    bbox: [min_lon, min_lat, max_lon, max_lat]
    """
    overpass_url = "https://overpass-api.de/api/interpreter"
    overpass_query = f"""
    [out:json];
    (
      way["building"]({bbox[1]},{bbox[0]},{bbox[3]},{bbox[2]});
      relation["building"]({bbox[1]},{bbox[0]},{bbox[3]},{bbox[2]});
    );
```

```
    out body;
    >;
    out skel qt;
    """

    response = requests.get(overpass_url, params={'data': overpass_query})
    return response.json()
```

2.2.5 F5: Kalkulator Strat

- Formuła: $strata = powierzchnia\_zalana \times warto\_za\_m^2 \times wspczynnik\_zniszczenia$
- Wartości domyślne:
  - Mieszkaniowy: 5000 PLN/m²
  - Komercyjny: 6000 PLN/m²
  - Przemysłowy: 3000 PLN/m²
- Współczynniki zniszczenia:
  - Płytka woda (<50cm): 0.3
  - Średnia (50-150cm): 0.7
  - Głęboka (>150cm): 0.95

3 Architektura Techniczna

3.1 Stack Technologiczny

Komponent	Technologia	Uzasadnienie
Frontend	React 18 + TypeScript	Nowoczesny, silne typowanie
Styling	Tailwind CSS + Framer Motion	Szybki development, animacje
Mapy	react-leaflet + OpenStreetMap	Darmowe, lekkie rozwiązanie
Backend	Python + FastAPI	Szybkie API, asynchroniczność
Przetwarzanie SAR	rasterio, xarray, geopandas	Lekkie, łatwe w instalacji
ML/AI	scikit-learn, TensorFlow	Proste modele + możliwość rozbudowy
Baza danych	SQLite (hackathon)	Brak konieczności setupu serwera
Hosting	Vercel (frontend) + Railway (backend)	Darmowe tier dla hackathonu

3.2 Schemat Architektury

```
crisis-eye/
 frontend/
    public/
    src/
       components/
          MapDashboard.tsx
          ControlPanel.tsx
          StatsPanel.tsx
          ReportGenerator.tsx
       hooks/
       services/
       App.tsx
    package.json
 backend/
    app/
       main.py
       api/
       services/
          satellite.py
          processing.py
          damage.py
       models/
    requirements.txt
    Dockerfile
 README.md
```

3.3 API Endpoints

```python
from fastapi import FastAPI, HTTPException, File, UploadFile
from pydantic import BaseModel
from typing import Optional, List
import numpy as np

app = FastAPI(title="CrisisEye API", version="2.1")

class AnalysisRequest(BaseModel):
    bbox: List[float]  # [min_lon, min_lat, max_lon, max_lat]
    date_flood: str    # "2024-07-15"
    days_before: int = 30
    asset_value_pln_per_m2: float = 5000.0

class AnalysisResponse(BaseModel):
    flood_mask_url: str
    flood_polygons_geojson: dict
    statistics: dict
    report_id: Optional[str] = None

@app.post("/api/analyze", response_model=AnalysisResponse)
async def analyze_flood(request: AnalysisRequest):
    """
    Główny endpoint do analizy powodzi
    """
```

```
    try:
        # 1. Pobierz dane SAR
        sar_data = await get_sar_data(
            request.bbox,
            request.date_flood,
            request.days_before
        )

        # 2. Przetwórz i wykryj zmiany
        flood_mask = process_flood_detection(sar_data)

        # 3. Pobierz budynki
        buildings = get_osm_buildings(request.bbox)

        # 4. Oblicz straty
        damage_report = calculate_damage(
            flood_mask,
            buildings,
            request.asset_value_pln_per_m2
        )

        # 5. Generuj wyniki
        return {
            "flood_mask_url": generate_tile_url(flood_mask),
            "flood_polygons_geojson": mask_to_geojson(flood_mask),
            "statistics": damage_report.statistics,
            "report_id": damage_report.id
        }

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.get("/api/report/{report_id}")
async def get_report(report_id: str):
    """
    Pobierz wygenerowany raport PDF
    """
    report_path = f"reports/{report_id}.pdf"
    if os.path.exists(report_path):
        return FileResponse(report_path)
    raise HTTPException(status_code=404, detail="Report not found")
```

4 Pipeline Przetwarzania Danych

4.1 Przepływ Danych

Step 1: **Trigger** - nowa scena Sentinel-1 dostępna w CREODIAS

Step 2: **Pobranie** - download produktu GRD (Ground Range Detected) lub użycie Google Earth Engine

Step 3: **Preprocessing**:

- Kalibracja radiometryczna (DN →) - może być pominięta jeśli używamy GEE

- Korekcja terenu (radiometric terrain correction)
- Speckle filtering (Refined Lee lub Median)
- Geokodowanie

Step 4: Change Detection:

- Wyszukanie sceny referencyjnej (mediana z okresu przed powodzią)
- Obliczenie log-ratio
- Thresholding automatyczny (Otsu) lub z ustalonym progiem

Step 5: AI Refinement (opcjonalnie):

- Użycie Random Forest do poprawy maski
- Usuwanie false positives (rzeki, jeziora stałe)
- Rozróżnienie wody stojącej od wody płynącej

Step 6: Analiza Strat:

- Pobranie budynków z OSM (lub Microsoft Building Footprints)
- Spatial join z maską zalania
- Obliczenie procentowego zalania każdego budynku
- Aplikacja modelu ekonomicznego

Step 7: Generowanie Wyników:

- Tiles dla mapy (png/jpeg)
- GeoJSON dla pobrania
- PDF raport z podsumowaniem

4.2 Model AI/ML

```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import rasterio

class FloodDetectionModel:
    def __init__(self):
        self.model = RandomForestClassifier(
            n_estimators=100,
            max_depth=10,
            random_state=42
        )
        self.features = ['vh_db', 'vv_db', 'vh_vv_ratio', 'texture', 'log_ratio']

    def extract_features(self, sar_image, reference_image):
        """
        Ekstrakcja cech dla każdego piksela
        sar_image: dict z kluczami 'vh', 'vv' (w dB)
        reference_image: dict z kluczami 'vh', 'vv' (w dB)
```

```
        """
        features = []

        # Podstawowe cechy
        features.append(sar_image['vh'])  # VH w dB
        features.append(sar_image['vv'])  # VV w dB
        features.append(sar_image['vh'] / sar_image['vv'])  # Ratio

        # Tekstura (wariancja w oknie 5x5)
        features.append(self.calculate_texture(sar_image['vh'], window_size=5))

        # Change detection feature
        log_ratio = 10 * np.log10(sar_image['vh'] / reference_image['vh'])
        features.append(log_ratio)

        # Stack features along the last axis
        return np.stack(features, axis=-1)

    def calculate_texture(self, image, window_size=5):
        """Oblicza teksturę jako wariancję w oknie"""
        from scipy.ndimage import uniform_filter
        mean = uniform_filter(image, window_size)
        mean_of_squares = uniform_filter(image**2, window_size)
        variance = mean_of_squares - mean**2
        return variance

    def train(self, X, y):
        """
        Trening na ręcznie oznaczonych danych
        X: cechy [n_samples, n_features]
        y: etykiety [n_samples], 0=ląd, 1=woda
        """
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )
        self.model.fit(X_train, y_train)

        # Ewaluacja
        accuracy = self.model.score(X_test, y_test)
        print(f"Model accuracy: {accuracy:.2%}")
        return accuracy

    def predict(self, features):
        """
        Predykcja maski wody
        """
        # Reshape for prediction
        original_shape = features.shape[:-1]
        features_flat = features.reshape(-1, features.shape[-1])

        predictions = self.model.predict_proba(features_flat)
        water_probability = predictions[:, 1]  # Prawdopodobieństwo wody

        # Reshape back
        return (water_probability > 0.5).reshape(original_shape)
```

5 Interfejs Użytkownika

5.1 Layout Komponentów

```tsx
import React, { useState } from 'react';
import { MapContainer, TileLayer, GeoJSON, useMapEvents } from 'react-leaflet';
import ControlPanel from './components/ControlPanel';
import StatsPanel from './components/StatsPanel';
import 'leaflet/dist/leaflet.css';

const CrisisEyeDashboard: React.FC = () => {
  const [bbox, setBbox] = useState<[number, number, number, number]>([17, 51,
  17.5, 51.5]);
  const [results, setResults] = useState<any>(null);
  const [loading, setLoading] = useState(false);

  const handleAnalyze = async () => {
    setLoading(true);
    try {
      const response = await fetch('/api/analyze', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          bbox,
          date_flood: '2024-07-15',
          days_before: 30
        })
      });

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }

      const data = await response.json();
      setResults(data);
    } catch (error) {
      console.error('Analysis failed:', error);
      alert('Analysis failed. Check console for details.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="min-h-screen bg-gray-900 text-white">
      {/* Header */}
      <header className="bg-gray-800 p-4 flex justify-between items-center">
        <div>
          <h1 className="text-2xl font-bold">CrisisEye</h1>
          <p className="text-cyan-400">Real-time flood detection with SAR</p>
        </div>
        <div className="text-sm">
          <span className="bg-red-500 px-2 py-1 rounded">Live</span>
          <span className="ml-2">Hackathon: AI między orbitami</span>
        </div>
      </header>
```

```
      <div className="flex h-[calc(100vh-80px)]">
        {/* Left Panel - Controls */}
        <div className="w-80 bg-gray-800 p-4 overflow-y-auto">
          <ControlPanel
            bbox={bbox}
            onBboxChange={setBbox}
            onAnalyze={handleAnalyze}
            loading={loading}
          />

          {results && <StatsPanel statistics={results.statistics} />}
        </div>

        {/* Main Map */}
        <div className="flex-1 relative">
          <MapContainer
            center={[51.25, 17.25]}
            zoom={10}
            className="h-full"
          >
            <TileLayer
              url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
              attribution='&copy; OpenStreetMap contributors'
            />

            {/* Flood mask overlay */}
            {results?.flood_mask_url && (
              <TileLayer
                url={results.flood_mask_url}
                opacity={0.6}
              />
            )}

            {/* Buildings overlay */}
            {results?.flood_polygons_geojson && (
              <GeoJSON
                data={results.flood_polygons_geojson}
                style={(feature) => {
                  const depth = feature?.properties?.depth;
                  let color = '#3b82f6'; // Default blue

                  if (depth === 'shallow') color = '#60a5fa';
                  if (depth === 'deep') color = '#1d4ed8';

                  return {
                    fillColor: color,
                    color: '#1e40af',
                    weight: 1,
                    opacity: 0.8,
                    fillOpacity: 0.5
                  };
                }}
              />
            )}
          </MapContainer>

          {/* Loading overlay */}
```

```
          {loading && (
            <div className="absolute inset-0 bg-black bg-opacity-50 flex
            items-center justify-center">
              <div className="text-white text-xl">
                Processing satellite data...
              </div>
            </div>
          )}
        </div>
      </div>
    </div>
  );
};

export default CrisisEyeDashboard;
```

5.2 Design System

Element	Specyfikacja
Primary Color	#00ffff (Cyan)
Secondary Color	#ff0000 (Red)
Background	#111827 (Gray-900)
Text Color	(White)
Font Family	Inter, system-ui, sans-serif
Map Tiles	OpenStreetMap (dark variant)
Animation	Framer Motion for smooth transitions
Glass Effect	backdrop-blur-lg bg-opacity-20

Table 1: Specyfikacja design system "Orbital Command"

6 Wymagania Niefunkcjonalne

6.1 Wydajność

- Czas przetwarzania: $<$ 10 minut dla obszaru 100km^2
- Czas odpowiedzi API: $<$ 3s dla 95
- Łączenie użytkowników: 50+ równocześnie
- Rozmiar danych: Kompresja rastrów do 8-bitowych PNG

6.2 Bezpieczeństwo

- CORS ograniczony do domeny aplikacji
- Walidacja inputów (bbox, daty)
- Rate limiting: 10 requestów/minutę na użytkownika
- Brak przechowywania danych osobowych

6.3 Skalowalność

- Stateless backend - łatwe skalowanie horyzontalne
- Cache'owanie wyników (Redis)
- Asynchroniczne przetwarzanie (Celery)
- CDN dla tile'ów mapy

7 Plan Implementacji na Hackathon

7.1 Podział Zadań (48h)

Członek zespołu	Rola	Zadania
Członek 1	Frontend Lead	React setup, komponenty mapy, UI
Członek 2	Backend Lead	FastAPI, integracja z CREODIAS/GEE, SAR processing
Członek 3	Data Scientist	Change detection, model ML, analiza strat
Członek 4	Full-stack	Integracja, raporty PDF, deployment

7.2 Harmonogram

Dzień 1 (20h):

- **Godziny 0-4:** Setup projektu, konfiguracja środowiska
- **Godziny 4-8:** Podstawowy frontend + backend
- **Godziny 8-12:** Integracja z CREODIAS/GEE
- **Godziny 12-16:** Implementacja change detection
- **Godziny 16-20:** Integracja OSM + podstawowy pipeline

Dzień 2 (20h):

- **Godziny 0-4:** Model ML (Random Forest)
- **Godziny 4-8:** Kalkulator strat + statystyki
- **Godziny 8-12:** Generator raportów PDF
- **Godziny 12-16:** Poprawki UI/UX, animacje
- **Godziny 16-20:** Testy, prezentacja, deployment

Kryterium	Metryka	Minimalna wartość
Działający pipeline	End-to-end przetwarzanie	100%
Czas przetwarzania	Od requestu do wyników	< 5 minut
Dokładność	Porównanie z ground truth	$> 75\%$ IoU
UI responsywność	Time to interactive	< 3s
Deployment	Dostępność online	24/7

7.3 Kryteria Sukcesu

8 Zależności i Wymagania

8.1 requirements.txt (Backend)

```
# Core
fastapi==0.104.0
uvicorn[standard]==0.24.0

# SAR Processing (LIGHTWEIGHT - NO SNAPPY!)
earthengine-api==0.1.378  # Google Earth Engine (łatwiejsze niż CREODIAS)
sentinelsat==1.6.0        # CREODIAS download (opcjonalnie)
rasterio==1.3.8           # Otwieranie GeoTIFF - LEKKIE!
geopandas==0.14.0
xarray==2023.10.0         # Alternatywa dla rasterio
rioxarray==0.15.0         # Raster I/O dla xarray

# ML/AI
scikit-learn==1.3.2
numpy==1.24.3
scipy==1.11.3             # WYMAGANE dla scikit-image i obliczeń
scikit-image==0.22.0      # Filtry i przetwarzanie obrazu
tensorflow==2.14.0        # Opcjonalnie dla zaawansowanych modeli

# Utilities
requests==2.31.0
pydantic==2.4.2
python-multipart==0.0.6
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4

# PDF Generation
reportlab==4.0.4
pdfkit==1.0.0
weasyprint==61.0         # Alternatywa dla reportlab

# Database
sqlalchemy==2.0.23
alembic==1.12.1
```

8.2 package.json (Frontend)

```
{
  "name": "crisis-eye-frontend",
  "version": "1.0.0",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "lint": "eslint . --ext ts,tsx --report-unused-disable-directives
    --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-leaflet": "^4.2.1",
    "leaflet": "^1.9.4",
    "axios": "^1.5.0",
    "recharts": "^2.8.0",
    "framer-motion": "^10.16.5",
    "tailwindcss": "^3.3.5",
    "autoprefixer": "^10.4.16",
    "postcss": "^8.4.31",
    "react-pdf": "^7.6.0"
  },
  "devDependencies": {
    "@types/react": "^18.2.33",
    "@types/react-dom": "^18.2.14",
    "@types/leaflet": "^1.9.8",
    "@typescript-eslint/eslint-plugin": "^6.0.0",
    "@typescript-eslint/parser": "^6.0.0",
    "@vitejs/plugin-react": "^4.1.0",
    "eslint": "^8.45.0",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.3",
    "typescript": "^5.2.2",
    "vite": "^4.5.0"
  }
}
```

9 Ryzyka i Środki Zaradcze

10 Rozszerzenia (Future Work)

1. **Predykcja rozprzestrzeniania** - użycie DEM do symulacji przepływu wody
2. **Mobile app** - aplikacja dla ratowników w terenie z offline maps
3. **Integracja z systemami GIS** - WMS/WFS dla samorządów
4. **Automatyczne alerty** - powiadomienia email/SMS dla subskrybentów

Ryzyko	Prawdopodobieństwo	Środek zaradczy
Brak dostępu do CREODIAS	Wysokie	Backup przez Google Earth Engine API
Wolne przetwarzanie SAR	Wysokie	Ogranicz obszar analizy, użyj preprocessed data
Słaba jakość danych OSM	Średnie	Użyj Microsoft Building Footprints jako fallback
Problemy z deploymentem	Średnie	Gotowe kontenery Docker, Vercel + Railway
Niski accuracy modelu	Wysokie	Prosty thresholding jako fallback, focus na UI
Limit czasu hackathonu	Wysokie	Minimal MVP: tylko core functionality

5. **Multi-hazard** - rozszerzenie na pożary, huragany, trzęsienia ziemi
6. **Social media integration** - agregacja zdjęć i doniesień z mediów społecznościowych

11 Źródła Danych i Licencje

11.1 Główne Źródła

- **Sentinel-1:** Copernicus Programme (ESA) - otwarty dostęp
- **OpenStreetMap:** ODbL license - darmowy do użycia z atrybucją
- **Google Earth Engine:** Free tier dla edukacji i non-profit
- **Microsoft Building Footprints:** MIT license - darmowy do użycia

11.2 Limity i Quoty

- CREODIAS: 100GB storage, bez limitów requestów
- Google Earth Engine: 250k requests/month na darmowym tier
- OSM Overpass API: 10k requests/day, 1GB download/day
- Vercel: 100GB bandwidth/month na hobby plan

12 Wnioski

CrisisEye przedstawia praktyczne rozwiązanie wykorzystujące dane satelitarne SAR do szybkiego wykrywania powodzi i szacowania strat. Dzięki focusowi na hackathonowy MVP, zespół może dostarczyć działający prototyp w 48 godzin, który demonstruje kluczowe funkcjonalności:

- **Detekcja w czasie rzeczywistym** - niezależnie od pogody

- **Automatyczna ocena strat** - z integracją danych przestrzennych
- **Profesjonalny interfejs** - dark mode z clear data visualization
- **Scalable architecture** - gotowa do rozbudowy po hackathonie

System ma potencjał realnego wpływu na zarządzanie kryzysowe, redukując czas reakcji z dni/tYGODNI do godzin.

— END OF DOCUMENT —