ChatGPT

AI-Powered Climate Change Prediction & Mitigation Platform

Final Project Documentation (2026 GC Software Engineering)

Stack: Next.js + NestJS + Python AI

Table of Contents

---

1. Project Overview

Objective

Build a scalable AI platform that predicts climate disasters (floods, wildfires) using satellite/IoT data and suggests mitigation strategies.

Problem Statement

Extreme weather events due to climate change require predictive tools for proactive disaster response and mitigation.

Key Features

- Real-time climate dashboard (Next.js)
- AI risk prediction (Python + TensorFlow)

- Secure API layer (NestJS)
- Multi-language support (i18n)
- Role-based access (Policymakers, NGOs, Public)

Compliance & Standards

- **API:** OpenAPI 3.0 (Swagger)
- **Frontend:** WCAG 2.1 accessibility
- **Backend:** RESTful API best practices
- **Data:** GDPR-compliant storage

---

2. System Architecture

Diagram

```
[Frontend - Next.js] ↔ [Backend - NestJS API] ↔ [AI Model - FastAPI] ↔ [Database - TimescaleDB]
          ↘ [WebSockets Layer] ↘ [Auth - JWT/OAuth 2.0]
```

Data Flow

1. User submits region data on UI
2. Backend forwards to Python AI service
3. AI model returns prediction
4. Backend stores result in DB and notifies via WebSocket
5. UI displays alerts/visualizations

---

3. Technology Stack

Component	Technology
Frontend	Next.js 14, TailwindCSS, i18next
Backend	NestJS, TypeORM, Swagger
AI/ML	Python, TensorFlow, FastAPI
Database	PostgreSQL (TimescaleDB)
Auth	JWT, OAuth 2.0
Deployment	Vercel (Frontend), AWS (Backend)
DevOps	GitHub Actions
Monitoring	Prometheus + Grafana

4. Frontend (Next.js)

Key Pages

- **Dashboard:** Map visualization (Mapbox/Kepler.gl)
- **Disaster Alerts:** Real-time notifications via WebSocket
- **Carbon Calculator:** User form + AI suggestions

API Communication

```
const fetchFloodRisk = async (region: string) => {
  const res = await fetch(`${NESTJS_API}/climate/predict-flood?region=${region}`);
  return res.json();
};
```

Auth

- JWT stored in `HttpOnly` cookies
- Interceptors to attach tokens to requests

i18n

- `next-i18next` with translation JSONs (e.g., `en.json`, `es.json`)

5. Backend (NestJS)

Structure

```
src/
├── climate/        # Climate data logic
├── ai/             # Python AI service calls
├── auth/           # JWT + OAuth
├── database/       # TypeORM models
└── main.ts         # Entry point
```

Swagger Setup

```
import { SwaggerModule } from '@nestjs/swagger';
const config = new DocumentBuilder()
  .setTitle('Climate API')
  .setVersion('1.0')
  .build();
const document = SwaggerModule.createDocument(app, config);
SwaggerModule.setup('api-docs', app, document);
```

URL: `https://api.climate-app.com/api-docs`

Middleware

- Global exception filters
- Guards for role-based access
- Rate limiter via `nestjs-rate-limiter`

---

6. AI/ML Integration (Python)

Workflow

1. Fetch IoT/satellite data
2. Clean and normalize
3. Train models (e.g., LSTM for floods)
4. Serve via FastAPI

FastAPI Endpoint

```
@app.post("/predict-flood")
def predict_flood(region: str):
    prediction = model.predict(region)
    return {"risk": prediction}
```

Model Performance

- Accuracy: 87.3%
- Metrics tracked via MLflow

---

7. API Communication

Frontend ↔ Backend

- **Protocol:** HTTPS (REST)
- **Auth:** JWT in headers
- **Format:** JSON

Backend ↔ AI Service

- **Options:** REST or gRPC

```
service ClimatePredictor {
  rpc PredictFlood (RegionRequest) returns (RiskResponse);
}
```

CORS

- Enabled for frontend origin

---

8. Database Design

Tables

Table	Purpose
`users`	Auth (roles: admin, public)
`climate_data`	Satellite time-series data
`predictions`	AI model results
`alerts`	Real-time user notifications

TimescaleDB Example

```sql
CREATE TABLE climate_data (
  time TIMESTAMPTZ NOT NULL,
  region VARCHAR(50),
  temperature FLOAT
);
SELECT create_hypertable('climate_data', 'time');
```

---

9. Deployment Strategy

Component	Tool/Service	URL
Frontend	Vercel	https://climate-app.vercel.app
Backend	AWS Elastic Beanstalk	https://api.climate-app.com
AI Model	AWS Lambda + FastAPI	Serverless endpoint

Secrets

- Handled via AWS SSM + Vercel env vars

---

10. Testing & Validation

Frontend

- **Tools:** Jest, Cypress
- **Tests:** Page load, i18n, form submission

Backend

- **Tools:** Jest, Supertest
- **Tests:** Auth routes, prediction API, DB queries

AI

- **Tools:** PyTest
- **Tests:** Model accuracy, data range validation
- **CI:** GitHub Actions + Coverage Badges

---

11. Future Enhancements

- **Blockchain:** Carbon credit tracking
- **Edge AI:** TensorFlow.js for browser-side inference
- **AR Visualization:** 3D climate overlays (Three.js + WebXR)

---

12. References & Standards

- **API Docs:** OpenAPI 3.0
- **i18n:** i18next docs
- **Database:** TimescaleDB Docs
- **ML:** MLflow, TensorFlow
- **Data Sources:** NASA EarthData, NOAA Climate Data

Deliverables

- GitHub Repo (`https://github.com/user/climate-platform`)
- Swagger Docs (`/api-docs`)
- Deployed Demo (`https://climate-app.vercel.app`)
- Video Walkthrough (Loom/YouTube)

- Postman Collection (`/docs/climate-api.postman_collection.json`)
- Sample Dataset (`/data/sample-climate.csv`)
- Architecture Diagram (`/docs/architecture.png`)