**1) Estructura de carpetas**

weather-backend/

├─ app/

│ ├─ main.py # FastAPI + CORS + endpoint /api/weather/predict

│ ├─ schemas.py # Pydantic models (request/response)

│ ├─ predictor.py # Lógica de predicción + métricas/estadística

│ ├─ providers/

│ │ ├─ base.py # Interfaz de proveedor de datos históricos

│ │ ├─ mock\_provider.py # Datos sintéticos (MVP runnable)

│ │ └─ opendap\_provider.py # (plantilla) Lectura real vía OPeNDAP/xarray

│ └─ utils.py # Helpers (HI, dirección del viento, etc.)

├─ requirements.txt

├─ .env.example

└─ README.md

**2) requirements.txt**

fastapi==0.115.2

uvicorn[standard]==0.30.6

pydantic==2.9.2

python-dotenv==1.0.1

numpy==2.1.2

pandas==2.2.3

xarray==2024.9.0

netCDF4==1.7.2

Más adelante, cuando actives NASA real, ya tienes xarray + netCDF4 para OPeNDAP.

**3) .env.example**

# CORS

ALLOWED\_ORIGINS=http://localhost:5173,http://127.0.0.1:5173

# Proveedor de datos (mock|opendap)

DATA\_PROVIDER=mock

# (para OPeNDAP/NASA más adelante)

EARTHDATA\_USERNAME=

EARTHDATA\_PASSWORD=

**4) Código**

**app/schemas.py**

from pydantic import BaseModel, Field, field\_validator

from typing import List, Optional, Literal

class PredictionRequest(BaseModel):

latitude: float = Field(..., ge=-90, le=90)

longitude: float = Field(..., ge=-180, le=180)

targetDate: str # "YYYY-MM-DD"

@field\_validator("targetDate")

@classmethod

def validate\_date(cls, v: str) -> str:

# Solo valida formato básico; el front ya asegura "futura"

import datetime as dt

try:

dt.date.fromisoformat(v)

except Exception:

raise ValueError("targetDate debe tener formato YYYY-MM-DD")

return v

class HistoricalRow(BaseModel):

year: int

temperatureC: float

humidity: float

windSpeed: float

conditions: str

class LocationOut(BaseModel):

latitude: float

longitude: float

name: Optional[str] = None

class PredictionOut(BaseModel):

temperatureC: float

temperatureMin: float

temperatureMax: float

humidity: float

humidityMin: float

humidityMax: float

windSpeed: float

windSpeedMin: float

windSpeedMax: float

windDirection: str

conditions: str

precipitation: str

visibility: str

confidence: float # 0-100

class AnalysisOut(BaseModel):

yearsAnalyzed: int

dataPoints: int

trends: str

notes: str

class PredictionResponse(BaseModel):

targetDate: str

location: LocationOut

prediction: PredictionOut

historicalData: List[HistoricalRow]

analysis: AnalysisOut

**app/utils.py**

import math

def wind\_deg\_to\_compass(deg: float) -> str:

dirs = ["N","NNE","NE","ENE","E","ESE","SE","SSE","S",

"SSW","SW","WSW","W","WNW","NW","NNW"]

ix = int((deg/22.5)+0.5) % 16

return dirs[ix]

def heat\_index\_c(temp\_c: float, rh: float) -> float:

"""

Heat Index aproximado en °C (conversión de la fórmula NOAA sobre °F).

"""

# pasa a F

t\_f = temp\_c \* 9/5 + 32

R = rh

HI\_f = (-42.379 + 2.04901523\*t\_f + 10.14333127\*R

- 0.22475541\*t\_f\*R - 0.00683783\*(t\_f\*\*2)

- 0.05481717\*(R\*\*2) + 0.00122874\*(t\_f\*\*2)\*R

+ 0.00085282\*t\_f\*(R\*\*2) - 0.00000199\*(t\_f\*\*2)\*(R\*\*2))

# vuelve a C

return (HI\_f - 32) \* 5/9

**app/providers/base.py**

from typing import List, Dict

from dataclasses import dataclass

@dataclass

class DailyRecord:

date: str # "YYYY-MM-DD"

t2m\_c: float # temperatura °C

rh: float # humedad %

wind\_ms: float # m/s

wind\_dir\_deg: float # 0-360

precip\_mm: float # mm/day

conditions: str # etiqueta descriptiva opcional

class IDataProvider:

"""

Interfaz: devuelve series diarias históricas para un punto (lat, lon)

y ventana DOY ±k durante un rango de años.

"""

def get\_historical\_window(self,

lat: float,

lon: float,

target\_date: str,

window\_days: int = 7,

year\_start: int = 2001,

year\_end: int = 2024) -> List[DailyRecord]:

raise NotImplementedError

**app/providers/mock\_provider.py**

import datetime as dt

import math

import random

from typing import List

from .base import IDataProvider, DailyRecord

class MockProvider(IDataProvider):

"""

Genera datos 'realistas' con estacionalidad suave.

Sirve para probar el front/back sin credenciales NASA.

"""

def get\_historical\_window(self, lat: float, lon: float,

target\_date: str,

window\_days: int = 7,

year\_start: int = 2001,

year\_end: int = 2024) -> List[DailyRecord]:

tgt = dt.date.fromisoformat(target\_date)

doy = tgt.timetuple().tm\_yday

out: List[DailyRecord] = []

for y in range(year\_start, year\_end+1):

# ventana DOY ± window\_days

for off in range(-window\_days, window\_days+1):

d = dt.date.fromordinal(

dt.date(y,1,1).toordinal() + (doy-1 + off)

)

season = math.sin(2\*math.pi\*(d.timetuple().tm\_yday/365.0))

base\_t = 20 + 12\*season + (lat/90)\*5 # °C

noise\_t = random.gauss(0, 2)

t2m\_c = base\_t + noise\_t

base\_rh = 65 - 10\*season

rh = max(20, min(100, base\_rh + random.gauss(0,7)))

base\_w = 3 + 2\*(1-abs(season))

wind\_ms = max(0, base\_w + random.gauss(0,1.2))

wind\_dir = random.uniform(0,360)

rain\_prob = 0.25 + 0.15\*(1-abs(season))

precip\_mm = random.gammavariate(2, 3) if random.random() < rain\_prob else 0.0

cond = "Rain" if precip\_mm>=1 else ("Clear" if rh<60 and t2m\_c>15 else "Clouds")

out.append(DailyRecord(

date=d.isoformat(),

t2m\_c=t2m\_c,

rh=rh,

wind\_ms=wind\_ms,

wind\_dir\_deg=wind\_dir,

precip\_mm=precip\_mm,

conditions=cond

))

return out

**app/providers/opendap\_provider.py (plantilla para NASA)**

# NOTA: Esta es una guía; puedes completarla con URLs reales cuando actives Earthdata.

import datetime as dt

from typing import List

import xarray as xr

import pandas as pd

from .base import IDataProvider, DailyRecord

class OpendapProvider(IDataProvider):

"""

Ejemplo para abrir MERRA-2/IMERG vía OPeNDAP con xarray (lazy).

Debes:

1) Resolver URLs de colección/variable (CMR/Harmony).

2) Abrir datasets con xr.open\_dataset(opendap\_url).

3) Seleccionar punto más cercano o interpolar.

"""

def get\_historical\_window(self, lat: float, lon: float,

target\_date: str,

window\_days: int = 7,

year\_start: int = 2001,

year\_end: int = 2024) -> List[DailyRecord]:

# Pseudocódigo (rellenar con colecciones/variables):

# ds\_t = xr.open\_dataset(T2M\_OPENDAP\_URL)

# ds\_w = xr.open\_dataset(WIND10M\_OPENDAP\_URL) # u10, v10

# ds\_p = xr.open\_dataset(IMERG\_DAILY\_URL)

# subset = ds.sel(lat=..., lon=..., time=slice(...))

# construir registros DailyRecord...

raise NotImplementedError("Implementar con URLs/variables reales")

**app/predictor.py**

import datetime as dt

import numpy as np

import pandas as pd

from typing import List, Tuple

from .providers.base import IDataProvider, DailyRecord

from .utils import wind\_deg\_to\_compass, heat\_index\_c

def \_summary(arr: np.ndarray) -> Tuple[float,float,float]:

return (float(np.nanmean(arr)),

float(np.nanmin(arr)),

float(np.nanmax(arr)))

def \_conditions\_text(temp\_c: float, rh: float, precip\_mm: float, wind\_ms: float) -> Tuple[str,str,str]:

# conditions / precipitation / visibility

if precip\_mm >= 20:

precip = "Lluvia fuerte"

cond = "Muy húmedo"

vis = "Reducida"

elif precip\_mm >= 1:

precip = "Lluvia ligera"

cond = "Húmedo"

vis = "Parcial"

else:

precip = "Sin precipitación"

cond = "Seco" if rh < 45 else "Templado"

vis = "Buena"

# Ajustes por calor/frío/viento

if temp\_c >= 32: cond = "Caluroso"

if temp\_c <= 0: cond = "Frío"

if wind\_ms >= 10: cond += " y ventoso"

return cond, precip, vis

def \_confidence(hits: int, total: int) -> float:

# confianza simple: cobertura + consistencia

if total == 0: return 0.0

p = hits/total

# intervalo (Wilson) aproximado → más datos = más confianza

n = total

conf = (0.5 + (n/ (n+20))) \* (0.4 + 0.6\*abs(p-0.5)\*2) # 0..1

return round(conf\*100, 1)

def predict\_for\_point(provider: IDataProvider,

lat: float, lon: float, target\_date: str,

window\_days: int = 7,

year\_start: int = 2001, year\_end: int = 2024):

recs: List[DailyRecord] = provider.get\_historical\_window(

lat, lon, target\_date, window\_days, year\_start, year\_end

)

if not recs:

raise ValueError("No hay datos históricos para este punto/fecha")

df = pd.DataFrame([{

"date": r.date,

"year": int(r.date[:4]),

"t2m\_c": r.t2m\_c,

"rh": r.rh,

"wind\_ms": r.wind\_ms,

"wind\_dir\_deg": r.wind\_dir\_deg,

"precip\_mm": r.precip\_mm,

"conditions": r.conditions

} for r in recs])

# Estadística base por día (agregado simple)

t\_mean, t\_min, t\_max = \_summary(df["t2m\_c"].to\_numpy())

h\_mean, h\_min, h\_max = \_summary(df["rh"].to\_numpy())

w\_mean, w\_min, w\_max = \_summary(df["wind\_ms"].to\_numpy())

# “dirección” dominante (mediana)

wind\_dir = float(np.nanmedian(df["wind\_dir\_deg"]))

# Selecciona un “día típico” por medianas para formar la tarjeta

# y estima HI para condiciones “desconfort”

hi = heat\_index\_c(t\_mean, h\_mean)

cond\_text, precip\_text, vis\_text = \_conditions\_text(

temp\_c=t\_mean, rh=h\_mean,

precip\_mm=float(np.nanmedian(df["precip\_mm"])),

wind\_ms=w\_mean

)

# Probabilidades simples de excedencia para algunas reglas

very\_hot\_hits = int((df["t2m\_c"] >= 32).sum())

very\_wet\_hits = int((df["precip\_mm"] >= 20).sum())

very\_windy\_hits = int((df["wind\_ms"] >= 10).sum())

total = len(df)

# Confianza (mezcla) — puedes refinar con varianza interanual

conf = \_confidence(

hits = very\_hot\_hits + very\_wet\_hits + very\_windy\_hits,

total = total\*3

)

# Tabla histórica (una fila por año con promedios en la ventana)

hist = (df.groupby("year")

.agg(temperatureC=("t2m\_c","mean"),

humidity=("rh","mean"),

windSpeed=("wind\_ms","mean"))

.reset\_index())

# Etiqueta simple por año

hist["conditions"] = np.where(hist["temperatureC"]>=32,"Caluroso",

np.where(hist["temperatureC"]<=0,"Frío","Templado"))

historical\_rows = hist.round(2).to\_dict(orient="records")

# Tendencia rápida (lineal) para temp media

yrs = hist["year"].to\_numpy()

tvals = hist["temperatureC"].to\_numpy()

if len(yrs) >= 3:

slope = float(np.polyfit(yrs, tvals, 1)[0]) # °C/año

trend\_text = f"Tendencia de +{slope\*10:.2f} °C por década en temperatura media (ventana DOY±{window\_days})."

else:

trend\_text = "Muestra insuficiente para tendencia robusta."

notes = "Estadística basada en ventana climatológica (±{0} días) a lo largo de {1}-{2}. No es un pronóstico determinista.".format(

window\_days, int(yrs.min()), int(yrs.max())

)

response = {

"temperatureC": round(t\_mean, 1),

"temperatureMin": round(t\_min, 1),

"temperatureMax": round(t\_max, 1),

"humidity": round(h\_mean, 1),

"humidityMin": round(h\_min, 1),

"humidityMax": round(h\_max, 1),

"windSpeed": round(w\_mean, 1),

"windSpeedMin": round(w\_min, 1),

"windSpeedMax": round(w\_max, 1),

"windDirection": wind\_deg\_to\_compass(wind\_dir),

"conditions": cond\_text,

"precipitation": precip\_text,

"visibility": vis\_text,

"confidence": conf

}

return response, historical\_rows, trend\_text, notes, len(yrs), len(df)

**app/main.py**

import os

from fastapi import FastAPI, HTTPException

from fastapi.middleware.cors import CORSMiddleware

from dotenv import load\_dotenv

from .schemas import PredictionRequest, PredictionResponse, LocationOut, PredictionOut, AnalysisOut

from .predictor import predict\_for\_point

from .providers.mock\_provider import MockProvider

# from .providers.opendap\_provider import OpendapProvider

load\_dotenv()

ALLOWED = os.getenv("ALLOWED\_ORIGINS","http://localhost:5173").split(",")

PROVIDER = os.getenv("DATA\_PROVIDER","mock").lower()

app = FastAPI(title="Weather Probability API")

app.add\_middleware(

CORSMiddleware,

allow\_origins=ALLOWED,

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

# Selección de proveedor

if PROVIDER == "mock":

provider = MockProvider()

elif PROVIDER == "opendap":

# provider = OpendapProvider()

raise RuntimeError("Activar OpendapProvider cuando esté implementado.")

else:

raise RuntimeError("DATA\_PROVIDER desconocido.")

@app.post("/api/weather/predict", response\_model=PredictionResponse)

def predict(req: PredictionRequest):

try:

pred, hist\_rows, trends, notes, years\_an, points = predict\_for\_point(

provider=provider,

lat=req.latitude,

lon=req.longitude,

target\_date=req.targetDate,

window\_days=7,

year\_start=2001,

year\_end=2024

)

except Exception as e:

raise HTTPException(status\_code=500, detail=str(e))

return PredictionResponse(

targetDate=req.targetDate,

location=LocationOut(latitude=req.latitude, longitude=req.longitude),

prediction=PredictionOut(\*\*pred),

historicalData=hist\_rows,

analysis=AnalysisOut(

yearsAnalyzed=years\_an,

dataPoints=points,

trends=trends,

notes=notes

)

)

**5) Cómo ejecutar**

# 1) Crear venv

python -m venv .venv

source .venv/bin/activate # Windows: .venv\Scripts\activate

# 2) Instalar dependencias

pip install -r requirements.txt

# 3) Configurar .env

cp .env.example .env

# (si tu front corre en http://localhost:5173, ya está ok)

# 4) Lanzar API

uvicorn app.main:app --reload --port 4000

Tu front puede apuntar a:

fetch('http://localhost:4000/api/weather/predict', { ... })

**6) Prueba rápida (cURL)**

curl -X POST http://localhost:4000/api/weather/predict \

-H "Content-Type: application/json" \

-d '{

"latitude": 6.2442,

"longitude": -75.5812,

"targetDate": "2025-12-20"

}' | jq

Obtendrás exactamente el **JSON** que espera tu WeatherDetail.

**🛰️ Activar datos NASA reales (cuando estés listo)**

1. Cambia DATA\_PROVIDER=opendap en .env.
2. Implementa OpendapProvider:
   * Resolver URLs de colecciones (MERRA-2: t2m, u10/v10; IMERG: precip).
   * Abrir con xarray.open\_dataset(opendap\_url) (requiere Earthdata Login).
   * Seleccionar el **punto** (o interpolar) y filtrar por **ventana DOY ±7** en rango de años.
   * Mapear a DailyRecord y devolver la lista.

Si prefieres **Harmony** para recortes por polígono y descarga de NetCDF ya “listo”, puedes crear un harmony\_provider.py con la misma interfaz IDataProvider y mantener la app intacta.

**🧩 Conexión con tu front (tal cual lo definiste)**

En src/components/WeatherNavigator.tsx solo cambia la URL:

const response = await fetch('http://localhost:4000/api/weather/predict', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(request)

})