Name -> Vikas kumar

github link -> https://github.com/Vikasprajapati7500/weather.git

Task 1

```
├── prisma/
|   └──schema.prisma
├── public/
├── styles/
|   └──globals.css
├── pages/
|   ├──api/
# Public assets (images, icons, etc.)
# Global CSS including Tailwind imports
|   |   ├──create-rule.ts    # API route for creating a rule
|   |   ├──combine-rules.ts  # API route for combining rules
|   |   └──evaluate-rule.ts  # API route for evaluating rules
|   └──index.tsx
├── lib/
|   ├──ruleEngine.ts
|   └──prisma.ts
├── components/
|   └──RuleForm.tsx
├── tailwind.config.js
├── postcss.config.js
├── tsconfig.json
├── package.json
├── prisma/
```

```
 |   ├──migrations/
 |   └──schema.prisma
 └── next.config.js
# Home page with form for rule creation
 # Rule engine logic (AST parsing, combining, evaluating)
 # Prisma client setup
 # React component for the rule form
 # Tailwind CSS configuration
 # PostCSS configuration for Tailwind CSS
 # TypeScript configuration
 # Project dependencies and scripts
 # Prisma migrations (auto-generated)
 # Prisma schema definition
 # Next.js configuration
 1
// schema.prisma
 model Rule {
  id      Int    @id @default(autoincrement())
  ast     Json   // Stores the AST structure as JSON
  createdAt DateTime @default(now())
 }
 model User {
  id         Int      @id @default(autoincrement())
  age        Int
  department String
  salary     Int
  experience Int
 }
 // api/rules.ts
```

```
import { NextApiRequest, NextApiResponse } from 'next';
import { parseRuleString, combineRuleASTs, evaluateRuleAST } from '../../lib/ruleEngine';
import { prisma } from '../../lib/prisma';  // Assuming Prisma is used for DB interaction
// POST /api/create-rule
export async function createRule(req: NextApiRequest, res: NextApiResponse) {
 const { ruleString } = req.body;
 try {
   const ast = parseRuleString(ruleString);  // Parse the rule string into an AST
   const rule = await prisma.rule.create({
     data: { ast: JSON.stringify(ast) }  // Store AST as JSON in PostgreSQL
   });
2
   res.status(200).json({ rule });
 } catch (error) {
   res.status(500).json({ error: 'Failed to create rule' });
 }
}
// POST /api/combine-rules
export async function combineRules(req: NextApiRequest, res: NextApiResponse) {
 const { ruleStrings } = req.body;
 try {
   const asts = ruleStrings.map((ruleString: string) => parseRuleString(ruleString));
   const combinedAST = combineRuleASTs(asts);  // Combine the ASTs
   res.status(200).json({ combinedAST });
 } catch (error) {
   res.status(500).json({ error: 'Failed to combine rules' });
 }
}
// POST /api/evaluate-rule
```

```
export async function evaluateRule(req: NextApiRequest, res: NextApiResponse) {
 const { ast, data } = req.body;  // AST and user data
 try {
   const isEligible = evaluateRuleAST(ast, data);  // Evaluate rule based on user data
   res.status(200).json({ isEligible });
 } catch (error) {
   res.status(500).json({ error: 'Failed to evaluate rule' });
 }
}
3
// pages/index.tsx
import { useState } from 'react';
export default function Home() {
 const [rule, setRule] = useState('');
 const [result, setResult] = useState(null);
 async function handleCreateRule() {
   const res = await fetch('/api/create-rule', {
     method: 'POST',
     headers: { 'Content-Type': 'application/json' },
     body: JSON.stringify({ ruleString: rule }),
   });
   const data = await res.json();
   setResult(data);
 }
 return (
   <div className="p-6">
     <h1 className="text-xl font-bold mb-4">Rule Engine</h1>
     <textarea
       className="border p-2 w-full"
```

```
      value={rule}
      onChange={(e) => setRule(e.target.value)}
      placeholder="Enter rule string..."
    />
    <button onClick={handleCreateRule} className="mt-2 bg-blue-500 text-white px-4 py-2">
4
      Create Rule
    </button>
    {result && <pre className="mt-4">{JSON.stringify(result, null, 2)}</pre>}
  </div>
 );
}
// lib/ruleEngine.ts
// Node structure for the Abstract Syntax Tree (AST)
export interface Node {
 type: 'operator' | 'operand';  // "operator" or "operand"
 operator?: 'AND' | 'OR';      // For operator nodes
 condition?: string;           // For operand nodes like "age > 30"
 left?: Node;                  // Left child (for operators)
 right?: Node;                 // Right child (for operators)
 value?: any;                  // Value for operand nodes, e.g., number, string
}
// Function to parse rule strings into AST
export function parseRuleString(ruleString: string): Node {
 // Parse rule string into an AST
 // Example: return abstract syntax tree (AST) for "age > 30 AND department = 'Sales'"
 return {
  type: 'operator',
  operator: 'AND',
```

```
  left: { type: 'operand', condition: 'age > 30' },
  right: { type: 'operand', condition: "department = 'Sales'" }
 };
5
}
// Function to combine multiple ASTs into one
export function combineRuleASTs(asts: Node[]): Node {
 // Combine multiple ASTs with 'AND' or 'OR' operators
 let combinedAST: Node = asts[0];
 for (let i = 1; i < asts.length; i++) {
  combinedAST = {
   type: 'operator',
   operator: 'AND',
   left: combinedAST,
   right: asts[i],
  };
 }
 return combinedAST;
}
// Function to evaluate an AST against user data
export function evaluateRuleAST(ast: Node, data: any): boolean {
 // Recursively evaluate the AST against the provided data
 if (ast.type === 'operand') {
  // Evaluate condition like 'age > 30'
  return evalCondition(ast.condition, data);
 } else if (ast.type === 'operator') {
  const leftEval = evaluateRuleAST(ast.left, data);
  const rightEval = evaluateRuleAST(ast.right, data);
  return ast.operator === 'AND' ? leftEval && rightEval : leftEval || rightEval;
```

```
 }
 return false;
6
}
// Helper function to evaluate a single condition like 'age > 30'
function evalCondition(condition: string, data: any): boolean {
const [field, operator, value] = condition.split(' ');
switch (operator) {
case '>': return data[field] > +value;
case '<': return data[field] < +value;
case '=': return data[field] === value.replace(/['"]/g, '');
default: return false;
}
}
```

Task 2 - > code

```
weather-app/
|
├── components/
|   └──WeatherCard.tsx
|
├── pages/
|   └──index.tsx
|
```

```
├── public/
│
├── services/
│   └──openWeather.ts
│
├── styles/
│   └──globals.css
│
├── utils/
│   └──weatherUtils.ts
│
├── .gitignore
├── next-env.d.ts
├── next.config.js
├── package.json
├── postcss.config.js
├── tailwind.config.js
├── tsconfig.json
└── README.md
# Component to display weather data and alerts
# Main page displaying weather data for multiple cities
# Static assets (optional)
# Service to interact with OpenWeatherMap API
# Tailwind CSS global styles
# Utility functions for temperature conversion and timestamp formatting
# Git ignore file
# Next.js TypeScript environment definitions
# Next.js configuration file
```

```
# Project dependencies and scripts

# PostCSS configuration for Tailwind CSS

# Tailwind CSS configuration file

# TypeScript configuration

# Project documentation (optional)

1

services/openWeather.ts

ts

Copy code

import axios from 'axios';

const API_KEY = '1b73a80d4682f7a2fa586674813aa6e8';

const API_URL = 'https://api.openweathermap.org/data/2.5/weather?units=metric&q=';

interface WeatherData {

 main: string;

 temp: number;

 feels_like: number;

 dt: number;

}

export const getWeatherData = async (city: string): Promise<WeatherData | null> => {

 try {

  const response = await axios.get(`${API_URL}${city}&appid=${API_KEY}`);

  const { main, weather, dt } = response.data;

  return {

   main: weather[0].main,

   temp: main.temp,

   feels_like: main.feels_like,

   dt,

2

};
```

```
} catch (error) {
console.error('Error fetching weather data:', error);
return null;
}
};
```

2. Create a utility for temperature conversion and time formatting.

utils/weatherUtils.ts

ts

Copy code

```
import dayjs from 'dayjs';
// Convert temperature from Kelvin to Celsius
export const kelvinToCelsius = (kelvin: number): number => kelvin - 273.15;
// Convert Unix timestamp to readable format
export const formatTimestamp = (timestamp: number): string => dayjs.unix(timestamp).format('YYYY-MM-DD HH:mm:ss');
// Calculate aggregate values for daily summaries
export const calculateDailyAggregates = (weatherData: Array<any>) => {
const tempSum = weatherData.reduce((acc, curr) => acc + curr.temp, 0);
const maxTemp = Math.max(...weatherData.map((data) => data.temp));
const minTemp = Math.min(...weatherData.map((data) => data.temp));
const dominantWeather = weatherData
.map((data) => data.main)
.sort((a, b) =>
weatherData.filter((v) => v === a).length - weatherData.filter((v) => v === b).length
)
3
  .pop();
 return {
  avgTemp: tempSum / weatherData.length,
  maxTemp,
```

```
  minTemp,
  dominantWeather,
 };
};
```

3. Create a component to display weather data and handle alerts.

components/WeatherCard.tsx

```
import { useEffect, useState } from 'react';
import { getWeatherData } from '../services/openWeather';
import { formatTimestamp } from '../utils/weatherUtils';
interface WeatherCardProps {
 city: string;
 threshold: number;
}
const WeatherCard: React.FC<WeatherCardProps> = ({ city, threshold }) => {
 const [weather, setWeather] = useState<any>(null);
 const [alert, setAlert] = useState<string | null>(null);
 useEffect(() => {
  const fetchWeather = async () => {
   const data = await getWeatherData(city);
   if (data) {
    setWeather(data);
4
    // Check for alert condition
    if (data.temp > threshold) {
     setAlert(`Alert: Temperature in ${city} exceeds ${threshold}°C!`);
    } else {
     setAlert(null);
    }
   }
```

```
  };

  // Fetch weather every 5 minutes

  fetchWeather();

  const interval = setInterval(fetchWeather, 300000);

  return () => clearInterval(interval);

 }, [city, threshold]);

 return (

  <div className="bg-white p-4 rounded-lg shadow-md">

   <h2 className="text-xl font-bold">{city}</h2>

   {weather ? (

    <>

     <p>Main: {weather.main}</p>

     <p>Temp: {weather.temp}°C</p>

     <p>Feels Like: {weather.feels_like}°C</p>

     <p>Last Update: {formatTimestamp(weather.dt)}</p>

    </>

   ) : (

    <p>Loading...</p>

5

   )}

   {alert && <p className="text-red-500 font-bold">{alert}</p>}

  </div>

 );

};

export default WeatherCard;
```

4. Create a page to display weather data for multiple cities and configure thresholds.

pages/index.tsx

tsx

Copy code

```
import WeatherCard from '../components/WeatherCard';

const cities = ['Delhi', 'Mumbai', 'Chennai', 'Bangalore', 'Kolkata', 'Hyderabad'];

const Home = () => {
 const threshold = 35; // Example threshold in °C
 return (
  <div className="container mx-auto p-4">
   <h1 className="text-2xl font-bold text-center mb-4">Weather Monitoring System</h1>
   <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
    {cities.map((city) => (
     <WeatherCard key={city} city={city} threshold={threshold} />
    ))}
   </div>
  </div>
 );
6
};
 export default Home
```