# Step-By-Step Instructions

## 1. Set Up the Project Environment

Before coding, ensure the project is correctly set up.

**1. Clone the repository and navigate to the project folder**

- Run:
```
git clone <repository-url>
cd Weather-Dashboard
```

**2.Install Dependencies**

- Run:
```
npm run install
```

This will install dependencies for both the client and server.

**3.  Create Environment Variables for API Keys**

- Navigate to Weather-Dashboard > server and create a .env file.
- Add:
```
OPENWEATHER_API_KEY=your_api_key_here
```

**4. Ensure TypeScript and Vite Configuration is Correct**

- `tsconfig.json` for the client (`Weather-Dashboard > client > tsconfig.json`) and server (`Weather-Dashboard > server > tsconfig.json`) should match project settings.
- Ensure `vite.config.js` (`Weather-Dashboard > client > vite.config.js`) has:

```
export default defineConfig({
    server: {
        port: 3000,
        open: true,
        proxy: {
            '/api': {
                target: 'http://localhost:3001',
                changeOrigin: true,
                secure: false,
            },
        },
    },
});
```

## 2. Implement the Backend (Server)

### 2A. Create an API to Fetch and Store Weather Data

**1. Implement OpenWeather API Fetching** (`Weather-Dashboard > server > src > service > weatherService.ts`)

- Add a function to fetch weather data:

```
import axios from 'axios';
import dotenv from 'dotenv';
dotenv.config();

const API_KEY = process.env.OPENWEATHER_API_KEY;
const BASE_URL = 'https://api.openweathermap.org/data/2.5/forecast';

export const getWeatherData = async (lat: string, lon: string) => {
  try {
    const response = await axios.get(`${BASE_URL}?lat=${lat}&lon=${lon}&appid=${API_KEY}`);
    return response.data;
  } catch (error) {
    console.error('Error fetching weather data:', error);
    throw new Error('Failed to fetch weather data');
  }
};
```

**2. Implement Search History Storage** (`Weather-Dashboard > server > src > service > historyService.ts`)

- Store search history in a JSON file:

```
import fs from 'fs';
import { v4 as uuidv4 } from 'uuid';

const FILE_PATH = 'server/db/searchHistory.json';

export const saveCity = (cityName: string) => {
  const history = getSearchHistory();
  const newCity = { id: uuidv4(), name: cityName };
  history.push(newCity);
  fs.writeFileSync(FILE_PATH, JSON.stringify(history, null, 2));
  return newCity;
};

export const getSearchHistory = () => {
  try {
    return JSON.parse(fs.readFileSync(FILE_PATH, 'utf8'));
  } catch {
    return [];
  }
};
```

## 2B. Set Up Routes

**1. Create API Routes** (`Weather-Dashboard > server > src > routes > api > weatherRoutes.ts`)

- Add routes for fetching weather data and handling search history:

```
import express from 'express';
import { getWeatherData } from '../../service/weatherService';
import { saveCity, getSearchHistory } from '../../service/historyService';

const router = express.Router();

router.get('/history', (req, res) => {
  res.json(getSearchHistory());
});

router.post('/', async (req, res) => {
  const { cityName, lat, lon } = req.body;
  if (!cityName || !lat || !lon) return res.status(400).json({ error: 'Missing parameters' });

  const weatherData = await getWeatherData(lat, lon);
  const newCity = saveCity(cityName);
  res.json({ city: newCity, weather: weatherData });
});

export default router;
```

**2. Set Up Server** (`Weather-Dashboard > server > src > server.ts`)

- Initialize Express and configure routes:

```
import express from 'express';
import cors from 'cors';
import weatherRoutes from './routes/api/weatherRoutes';

const app = express();
app.use(cors());
app.use(express.json());

app.use('/api/weather', weatherRoutes);

app.listen(3001, () => console.log('Server running on port 3001'));
```

## 3. Implement the Frontend (Client)

### 3A. Handle API Calls (`Weather-Dashboard > client > src > main.ts`)

**1. Create Functions to Fetch Weather Data**

```
async function fetchWeather(city: string) {
  const response = await fetch(`/api/weather`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ cityName: city })
  });
  return response.json();
}

async function fetchHistory() {
  const response = await fetch(`/api/weather/history`);
  return response.json();
}
```

**2. Implement UI Updates**

```
document.getElementById('search-button')?.addEventListener('click', async (event) => {
  event.preventDefault();
  const city = (document.getElementById('search-input') as HTMLInputElement).value;
  if (city) {
    const weather = await fetchWeather(city);
    displayWeather(weather);
  }
});

function displayWeather(weather: any) {
  document.getElementById('search-title')!.innerText = weather.city.name;
  document.getElementById('temp')!.innerText = `Temperature: ${weather.weather.main.temp}°F`;
  document.getElementById('wind')!.innerText = `Wind: ${weather.weather.wind.speed} MPH`;
  document.getElementById('humidity')!.innerText = `Humidity: ${weather.weather.main.humidity}%`;
}
```

## 3B. Implement UI Updates

**1. Modify the `fetchWeather` function to include forecast data**

- In `Weather-Dashboard > client > src > main.ts`:

```
async function fetchWeather(city: string) {
  const response = await fetch(`/api/weather`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ cityName: city })
  });
  const data = await response.json();
  displayWeather(data);
  displayForecast(data.weather.list);
}
```

**2. Create a function to display the 5-day forecast**

- Inside `Weather-Dashboard > client > src > main.ts`, add:

```
function displayForecast(forecastList: any[]) {
  const forecastContainer = document.getElementById('forecast');
  if (!forecastContainer) return;
  forecastContainer.innerHTML = ''; // Clear previous forecasts

  // OpenWeather returns forecasts for every 3 hours, so we filter to one per day
  const dailyForecasts = forecastList.filter((_, index) => index % 8 === 0);

  dailyForecasts.forEach((forecast) => {
    const date = new Date(forecast.dt * 1000).toLocaleDateString();
    const temp = forecast.main.temp;
    const wind = forecast.wind.speed;
    const humidity = forecast.main.humidity;
    const icon = `https://openweathermap.org/img/wn/${forecast.weather[0].icon}.png`;

    const forecastCard = `
      <div class="card forecast-card">
        <div class="card-body text-center">
          <h5 class="card-title">${date}</h5>
          <img src="${icon}" alt="${forecast.weather[0].description}">
          <p class="card-text">Temp: ${temp}°F</p>
          <p class="card-text">Wind: ${wind} MPH</p>
          <p class="card-text">Humidity: ${humidity}%</p>
        </div>
      </div>
    `;

    forecastContainer.innerHTML += forecastCard;
  });
}
```

**3. Ensure this function is called in `fetchWeather`**

- Now, when a user searches for a city, both **current weather** and **5-day forecast** will be displayed.

## 4. Deploy the Application

**1. Deploy to Render**

- Follow Render’s Guide
- Deploy server as a web service
- Deploy client as a static site

**2. Set Environment Variables in Render**

- Add OPENWEATHER_API_KEY in Render's settings.

**3. Ensure App Works in Production**

- Visit the deployed site.
- Test API calls.
- Debug errors.

## 5. Bonus: Add DELETE Functionality

**1. Add DELETE Route** (`Weather-Dashboard > server > src > routes > api > weatherRoutes.ts`)

```
router.delete('/history/:id', (req, res) => {
  const { id } = req.params;
  const history = getSearchHistory().filter((city) => city.id !== id);
  fs.writeFileSync(FILE_PATH, JSON.stringify(history, null, 2));
  res.json({ message: 'City deleted' });
});
```

**2. Implement Frontend Delete Button**

```
async function deleteCity(id: string) { await fetch(`/api/weather/history/${id}`, { method:
'DELETE' }); loadHistory(); }
```

# GPT Suggested Improvements

**1. Ensure the server loads the `index.html` file for all routes**

- You need to serve the frontend for users visiting the app directly.
- Add this to `Weather-Dashboard > server > src > routes > htmlRoutes.ts`:

```
import express from 'express';
import path from 'path';

const router = express.Router();

router.get('*', (req, res) => {
  res.sendFile(path.resolve(__dirname, '../../../client/dist/index.html'));
});

export default router;
```

- Then **import this route into `server.ts`**:

```
import htmlRoutes from './routes/htmlRoutes';
app.use(htmlRoutes);
```

**2. Ensure Weather API Requests Include Units (`metric` or `imperial`)**

- The OpenWeather API returns temperature in Kelvin by default.
- Modify `getWeatherData` in `weatherService.ts`:

```
const response = await
axios.get(`${BASE_URL}?lat=${lat}&lon=${lon}&appid=${API_KEY}&units=imperial`);
```

- This ensures temperature is displayed in **Fahrenheit (°F)**

**3. Ensure Previously Searched Cities Load on Page Refresh**

- Modify `fetchHistory` in `main.ts` to **update the UI**:

```
async function fetchHistory() {
  const history = await fetch(`/api/weather/history`).then(res => res.json());
  const historyContainer = document.getElementById('history');
  if (!historyContainer) return;
  historyContainer.innerHTML = '';

  history.forEach((city: { id: string; name: string }) => {
    const button = document.createElement('button');
    button.textContent = city.name;
    button.className = 'history-btn btn btn-primary';
    button.onclick = () => fetchWeather(city.name);
    historyContainer.appendChild(button);
  });
}
```

- Call `fetchHistory()` **on page load** to auto-load previous searches