Weather Application - WGU Capstone

Taylor Ketterling

C964 - Task 2

1/29/2026

Letter of Transmittal: Weather Intelligence Decision-Support Application

1/20/2026

To: Senior Leadership Team
From: Taylor Ketterling
Subject: Transmittal – Weather Intelligence Decision-Support Data Product (Weatherapp)

Dear Leadership Team,

I am submitting the Weather Intelligence Decision-Support Application, Weatherapp, for your review. This project addresses a common operational challenge in that when weather impacts safety, staffing, equipment, or potentially revenue, decision-makers often rely on fragmented sources (phone apps, delayed alerts, or generalized forecasts) that lack operational context and reduce the ability to respond proactively.

I am requesting for approval of the deployment of the Weatherapp as a lightweight data product that consolidates NOAA/NWS weather data into a single, map-based view of current conditions, forecast changes, and weather risk indicators.

This provides faster situational awareness and supports informed planning, with the option for targeted alerts based on the organization's defined tolerance conditions. The application converts raw weather data into actionable visual insight, reducing the time and effort required to interpret traditional weather reports. It helps teams quickly identify conditions that may disrupt operations and plan around short-term patterns.

The project will require minimal funding and uses open-source software. If an existing compute device is not available, **the total initial funding requirement is $832.88** (web hosting, a used LattePanda Sigma, and a domain).

I have academic training in computer engineering (CSUN) and I am soon to be completing a Computer Science degree at WGU. Professionally, I work as an IT Director managing Linux servers, networking, security, and cloud administration. These skills directly support building and operating the system reliably (API services, database, reverse proxy, and ML service).

Thank you for your time and consideration. I'm available to walk through the application at a high level and answer any questions related to rollout, operating expectations, and ongoing maintenance of the system.

Sincerely,

Taylor Ketterling
Enclosed: Project Proposal, Quick Start Guide, Executive Summary, Codebase

1/21/2026

Project Proposal

Taylor Ketterling

Problem Summary

This project will deliver a web-based weather intelligence application designed to support organizations with decision making where weather conditions directly impact operations, safety, or financial outcomes. The application will consolidate publicly available weather data from NOAA and/or NWS and present it through an interactive, map based interface that highlights current conditions, forecasted changes, and weather-related risk indicators.

The project will be developed for organizations that operate facilities, equipment, or personnel in environments where weather uncertainty creates any operational risk. Currently decision-makers often rely on fragmented weather reports such as their phone's weather app, delayed alerts, or too generalized forecasts that lack any geographic or operational context. This fragmentation limits the ability to respond proactively to changing conditions, such as shutting a ride down as the weather passes set tolerances including wind, rain, or thunder head risk.

The proposed application will meet organizational needs by centralizing weather data into a single map platform that enables faster situational awareness and more informed planning along with potential for live alerts sent directly to decision makers if they utilize polygon alerting and setting their tolerance conditions, removing the need to consistently monitor the app. The project will deliver a functional data product that visualizes weather layers, supports time based exploration, and provides predictive risk indicators derived from historical and forecast data. Upon completion, the organization will gain a practical decision support tool that improves weather awareness and reduces uncertainty.

Application Benefits

The proposed application will directly address organizational needs related to risk management, operations planning, and situational awareness of weather patterns that may affect an organization's interests. By transforming raw weather data into actionable visual insights, the application will reduce the cognitive and time burden associated with interpreting traditional weather reports when utilizing the map.

The organization will benefit from being able to fastly identify any weather condition that may disrupt any operations and improve the organization's ability to plan around short-term weather patterns.

Application Description

The Weather App is a responsive web application that provides users with real-time weather conditions and multi-day forecasts for any city or location within the US (where data sources are available). The app allows users to search by interacting with the map to show local weather data. The goal is to help users quickly view current temperature, precipitation, wind speed, and expected conditions so they can plan daily activities or facility operations.

Data Description

Raw data comes from NOAA stations in the form of csv files. The daily-summaries-latest.tar.gz contains thousands of these records in a ~7gb file. The tar file must be ingested through a tar buffer that looks through each compressed csv file for new data. The stations have time-series data, with many stations having many days of data, with each data containing daily temperature mean, precipitation, wind, and more depending on the available data from the station. Not all stations have all data, according to NOAA, half of all stations only record precipitation.

Some types of data used include nominal data such as station id, station name, and flags.

Quantitative data includes temperature, wind speed and atmospheric pressure.

Temporal data includes data and time of observation.

Dependent variables include tomorrow's predictions of Tmin, Tmax, Precipitation and Wind.

Independent variables include logged historic data and station context features such as latitude and longitude.

Anomalies include weird data and missing data. Weird data includes LAX which is reporting a yearly min and max of 0 → 5 * C , that's not normal for LA, nearby Hawthorne Airport reports a more reasonable 10* → 30 * C yearly min max. LAX's NWS live data also disagrees with its historic NOAA data.

Missing data exists for stations no longer in service or stations missing data ingests. The program must be able to work with expired data and ignore stations with missing data.

Objectives and Hypothesis

The objective is to enable a fully featured map application that allows the user to digest and interpret NOAA and NWS data; then use said data with machine learning algorithms to predict future weather patterns. The Machine learning capabilities will be successful if they produce predictions with an accuracy of 90%.

Methodology

Development methodology:
CRISP-DM (Cross-Industry Standard Process for Data Mining) implemented with an agile build style (build → test → refine each ingestion + API + ML piece in small cycles).

CRISP-DM supports the project as its an end-to-end data pipeline + ML + API system: where the program must ingest NOAA data, store it reliably, generate predictions, and serve it to a web client. CRISP-DM is built for this exact flow (problem → data → prep → model → evaluate → deploy), and smaller iterative cycles are practical because working increments can be shipped one at a time. Such as setting up the Database, then moving on the API, before working on machine learning capabilities.

CRISP-DM is appropriate for this project as it can document what the app is solving (business understanding) separately from how NOAA data is structured/cleaned (data understanding & prep), and separately from ML and deployment. CRISP-DM also will help with data issues like the LAX anomaly, missing values, and station coverage gaps are normal in climate datasets; CRISP-DM explicitly includes data understanding & preparation phases to address this before modeling. This methodology allows us to loop back as new information comes to light.

Finally CRISP-DM ends with deployment and monitoring, which is exactly the final goal of this program, to be operationally, useful, and monitored.

CRISP-DM Phases:

1) Business Understanding
   Define what the system must do and how success is measured.
   a) This is when we must identify user needs, such as “Must have at least 1 week of forecast predictions” or “Must have access to live data”
   b) Define measurable outcomes: API must respond in less than 20 seconds, station data must update weekly
   c) Decide project scope

2) Data Understanding
   Learn what data exists, what it looks like, and what can go wrong with the data
   a) Inspect the NOAA daily summaries structure
   b) Profile data ranges per station
   c) Identify missingness patterns, station coverage differences, and outliers

3) Data Preparation
   a) Make the NOAA dataset usable and consistent for storage and Machine learning programs.
   b) Normalize and validate units before filtering flagged records and transforming into the database schema.

4) Modeling
   a) Train and run ML forecasts using prepared historical data, choose a baseline approach first, then train models using a consistent time-based split
   b) Store model runs and predictions into ML tables

5) Evaluation
   a) Validate prediction accuracy using available metrics
   b) Evaluate system correctness such as ingestion completeness, schema constraints, API endpoints, response payloads

6) Deployment and monitoring
    a) Operate the system reliably in a production-like environment
    b) Deploy Java API + Python ML as systemd services, exposed through the caddy service
    c) Add logging and monitoring: ingestion job logs, counts, timings, error alert
    d) Ongoing maintenance plan (retrain schedule, drift checks, anomaly detection)

CRISP-DM is agile in nature and will feature iterative sprints that aim to launch working features module by module.

Funding Requirements

The project needs minimal resources to launch and uses open-source software to keep cost down and reduce development overhead.

- Front-end Web Hosting : $155.88 / yr

Title	Paid at	Amount
ketterling.space	2026-01-22	US$ 155.88

- Back-end compute device:
    - Used: Latte Panda Sigma $648

16GB RAM, SN770 500GB SSD, AX211 WiFi 6E

$648.00

- Domain:
    - Domains average $20 yearly for .com and .space domains.

- Software cost: **$0**, entire tech stack is open source.
    - Ubuntu
    - Caddy, postgres, postgis, scikit

- If no compute device available, project funding requirement is **$832.88**

Impact of the Solution on Stakeholders

Non-technical users will benefit from a simple interface for interpreting weather information. The map view and charts will reduce the learning curve and allow users to answer common questions quickly (What is happening now? What is likely next? Is there an active alert near us? Think Lahaina's wild fire)

Data stakeholders will gain a structured dataset that can be queried, audited, and reused. Historic station data, forecast snapshots, and ML predictions will be stored in normalized tables, enabling repeatable analytics, validation, and model improvement over time. Data anomalies will be easier to detect because they will be visible through dashboards and data-quality checks; for example, when I noticed LAX's data is an outlier, I visually saw it as abnormally cold for the area.

Executive stakeholders will see that the Weather App data product will improve decision speed and consistency by providing a single source of truth for weather risk across the entire organization. Leadership will be able to make go/no-go calls faster, reduce avoidable downtime associated with weather, and justify decisions with logged alerts, forecasts, and historical context rather than any informal assumptions.

Data Precautions

- Sensitive or protected data: **none**
  - No PII , PHI or Student data is collected, stored, or used.

- If applicable, describe necessary precautions which will be taken.
  - Both APIs have ambiguous rate limits to "what is reasonable for the data you are collecting" so data such as NWS alerts are acceptable to request on a 30 second basis whereas hourly summaries are acceptable to request every 30 minutes.
  - Offline ingest from the daily-summaries-latest.tar.gz is typically a couple days old.

The data being used is publicly available datasets from government agencies. NOAA station history can be found here. NWS API information can be found here.

Developer's Expertise

The developer, Taylor Ketterling, has academic training in computer engineering (CSUN) and is completing a Computer Science degree at WGU. Professionally, he has worked as an IT Director and managed real-world infrastructure including Linux servers, networking, security, and cloud administration.

These qualifications match the needs of this project because the system requires building and maintaining production-style services (systemd), deploying behind a reverse proxy (Caddy), designing and querying a relational database (PostgreSQL/PostGIS), implementing backend APIs (Java), and integrating an ML service (Python). The project reflects a practical application of this training and work experience.

Executive Summary: Weather Intelligence Decision-Support Application

1/26/2026

Part B: Executive Summary

Problem Statement

Organizations with facilities, equipment, or personnel exposed to weather risk will continue to experience operational uncertainty because weather data is typically consumed from fragmented sources (consumer phone apps, delayed alerts, generalized forecasts) that lack geographic and operational context. This fragmentation will slow response time, increase the likelihood of weather-related disruptions, and create inconsistent decision-making when conditions cross operational tolerances.

The proposed Weather Intelligence Decision-Support Application, Weatherapp,. will address this problem by centralizing NOAA/NWS weather data into a single platform with map-driven situational awareness and predictive machine learning indicators derived from historical and forecast data.

Client Summary

The intended client is an organization with operational risk exposure to weather (facilities operations, logistics, outdoor venues, field teams, public safety support, or any organization that must make time-sensitive operational decisions based on changing weather).

The Weather App will resolve the problem by converting raw public weather data into an operationally usable data product with a single-pane-of-glass map interface. Backed by a backend architecture designed for reliability through DB persistence, service isolation, reverse proxy, and scheduled ingestion/model runs. The application will support both descriptive (visualization, historical summaries) and non-descriptive (ML prediction) functions, enabling proactive planning rather than reactive weather monitoring.

Existing System Analysis

Current tools used include employees phone’s weather apps, browsing raw NOAA/NWS resources, and manually interpreting alerts and forecasts. No centralized data offerings available via web interface.

The current environment will be insufficient because it will not provide a consolidated geospatial view tied to the organization’s operational footprint, nor will it support consistent tolerance-based decisioning without continuous manual monitoring. The current tool do not provide a structured, queryable historical dataset and ML-based predictive indicators inside the same system

Data

Raw dataset: The raw historical dataset will originate from NOAA station “daily summaries” CSV records distributed in daily-summaries-latest.tar.gz, which will contain thousands of per-station CSV files and represent ~7GB of compressed data.

The project will also use live/near-live weather and alert data from NWS endpoints (queried on an interval appropriate to rate limits and operational needs).

Data structure and types: Each station dataset will be time-series in nature (station × day), containing daily observations such as temperature, precipitation, wind, and other fields depending on station capability. According to NOAA, 50% of weather stations only collect precipitation data. The dataset will include nominal identifiers (station id/name, flags), quantitative measures (temperature, wind speed, pressure), and temporal fields (date/time of observation). The dependent (target) variables will include next-day predictions for Tmin, Tmax, precipitation, and wind, while independent variables will include historic lag features and station context (latitude/longitude).

Data collection, processing, and management across the lifecycle: A PostgreSQL database with the PostGIS extension will be used to store station metadata, historical daily summaries, ML predictions, and cached live API outputs. The historic NOAA archive will be ingested using a streaming/buffered tar read approach (tar buffer) to avoid RAM exhaustion and to process per-station CSV entries incrementally. The Java service will handle ingestion and API serving, persisting normalized records into Postgres/PostGIS. The Python ML service will train/infer using scikit-learn and will write predictions back to the ML tables in the database.

Historic ingestion will be repeatable and resumable after failures. Live API data will be cached and refreshed on a schedule aligned to rate limits (alerts more frequent than hourly summaries).

Outliers/weird station ranges: Stations with suspicious ranges (e.g., LAX reporting yearly min/max around 0–5°C) will be flagged for exclusion, unit validation, or downgraded trust. Stations with missing or expired data will be skipped, and the system will remain functional even when a subset of stations is unavailable. Since many stations will not record all fields (NOAA indicates many are precipitation-only), the ML pipeline will require feature gating and fallbacks for stations lacking required predictors.

Project Methodology

The project will use CRISP-DM (Cross-Industry Standard Process for Data Mining) executed in iterative increments (agile-like sprints) to deliver usable capabilities early and refine based on data findings and operational constraints. CRISP-DM will be appropriate because this project is an end-to-end data pipeline with ML and API system where data quality, feature availability, and deployment reliability must be addressed explicitly. The methodology will support revisiting earlier phases when issues such as station anomalies, missingness, or feature coverage gaps are discovered.

The planned development by phases under CRISP-DM:

1. Business Understanding: Requirements, success metrics, scope boundaries.

2. Data Understanding: Inspect NOAA archive structure, profile station distributions, identify missingness/outliers.

3. Data Preparation: Normalize/validate units, filter flagged records, transform into DB schema, build features.

4. Modeling: Train baseline and improved models, use time-based splits, store model runs/predictions.

5. Evaluation: Validate Machine Learning model accuracy and verify end-to-end correctness

6. Deployment/Monitoring: Deploy Java and Python services as systemd units behind Caddy; implement all logging, health checks, and ongoing monitoring.

Project Outcomes (Deliverables)

The project will deliver the following:

1. A functional Weatherapp (the data product)
    a. Map-based front end using MapLibre
    b. Backend platform consisting of Java REST API + ingest service, Python ML API, PostgreSQL/PostGIS Database, and Caddy reverse proxy
2. Quick Start Guide
    a. Step-by-step instructions for OS hardening, installing dependencies, DB initialization, service setup, and operations
3. Deployment artifacts
    a. Database initialization SQL
    b. systemd service units for Java and Python components
    c. Caddyfile reverse proxy configuration limiting public exposure to 443/TLS endpoints.

Implementation Plan

Development Phases:

1. Infrastructure and security setup and hardening
    a. Ubuntu host provisioning
    b. Firewall rules and hardening
2. Database rollout
    a. Install PostgreSQL/PostGIS
    b. Create weatherdb and least-privilege DB user
    c. initialization DB with included SQL, verify schema and privileges
3. Backend API/Ingest rollout
    a. Build Java service with Maven and deploy as a restricted service user
    b. Verify ingests from tar archive and validate row counts / data ranges from consol
4. ML service rollout
    a. Create Python venv, install dependencies (FastAPI/ Uvicorn/ Pandas/ NumPy scikit-learn)
    b. Deploy as systemd service and validate inference endpoints.
5. Front-end rollout
    a. Host public_html on webhost
    b. configure CORS to allow webhost source

Services will be restarted and verified using documented commands, and logs will be validated through journald.

Evaluation Plan

The Weatherapp will perform schema verification of tables, indexes, constraints, and PostGIS extension availability through startup checks & database init script execution. Ingestion verification will occur through ingest counts per station entry, detection of new rows, and run logging (newRows counters). The API can be evaluated through endpoint health checks, response payload structure, and proxy routing (Caddy allowed paths). Finally the machine learning program will be verified through successful model run completion, prediction writes to DB, and inference endpoint response validation.

Once completed, completion validation will confirm:

- The map UI will display live and historic layers correctly.
- The system will return predictions for target variables (Tmin/Tmax/precip/wind) and store them consistently.
- The ML component will meet the project's stated success target of 90% accuracy (as defined in the project objectives).
- The deployment will operate through HTTPS behind a reverse proxy with reduced public exposure.

Resources and Costs

Hardware

- Back-end compute device (used LattePanda Sigma): $648.00
- Front-end web host : $155.88 / year

Software

- Domain: ~$20 / year
- Software licensing: $0.00
    - (open-source stack: Ubuntu, Caddy, PostgreSQL/PostGIS, scikit-learn)

Estimated Human Resources: 120 hours development labor

- Infrastructure and security setup and hardening: 20h
- Data ingest + DB work: 20h
- Back-end API + proxy integration: 30h
- Back-end ML training/inference integration: 30h
- Front-end UI integration: 15h
- Testing + dos: 5h

Total funding requirement : $832.88 + development labor

Timeline

Milestone	**Milestone Implementation Goals**	**Start**	**End**
Milestone 1 (2 days): Requirements & Success Criteria	finalize scope, tolerance targets, endpoints, acceptance criteria	2026-01-16	2026-01-17
Milestone 2 (2 days): Environment & Security	Setup Ubuntu host + firewall + Caddy proxy	2026-01-18	2026-01-19
Milestone 3 (2 days) : Database + PostGIS	schema init, privileges, spatial readiness	2026-01-19	2026-01-20
Milestone 4 (2 days): Historic NOAA Ingestion - Java Ingest	tar-buffer ingest pipeline + validation & anomaly handling	2026-01-20	2026-01-21
Milestone 5 (2 days): Java API Layer	REST endpoints + cached layer outputs	2026-01-21	2026-01-22

Milestone 6 (2 days): ML Service	scikit-learn training/inference + DB prediction writes	2026-01-22	2026-01-23
Milestone 7 (2 days): UI Integration	MapLibre layers + time navigation + prediction overlays	2026-01-24	2026-01-25
Milestone 8 (2 days) : System Testing & Hardening	end-to-end verification, performance checks, logging readiness	2026-01-26	2026-01-27
Milestone 9 (1 day) : UAT & Documentation	evaluator run path, quick start validation	2026-01-28	2026-01-28
Milestone 10 (1 day) : Release	final packaging, deployment readiness sign-off	2026-01-29	2026-01-29

Codebase: Weather Intelligence Decision-Support Application

1/29/2026

Taylor Ketterling

GitLab Repo: *Weatherapp*

This application features the use of multiple parts, starting with a front-end enabled through *Hostinger*(domain, webhost), a *firewalla purple* for network management and forwarding 443 and 80 port access to the *Latte Panda*. The *Latte Panda* will host the ML application, Database, API server, and a reverse proxy to help avoid exposing the APIs directly to the internet, the reverse proxy reduces exposed open ports and is made easy through a utility called Caddy. All dependencies such as caddy, postgres, postgis, mapLibre, jackson, hikari, and javelin are all open source resources.

A quick start guide is provided which covers launching the backend systems(Database, Java API/ingestion service, Machine learning service / API, and reverse proxy). public_html is hosted by Hostinger, it can be locally hosted to test; due to caddy public_html/index.html cannot be accessed in browser via file:c://…

The public_html directory must be hosted due to CORS origin handling. If you host public_html on a custom domain you need to add them as an accepted cors_allow_origins under WeatherApp\ml_service\settings.py and

One descriptive method and one nondescriptive (predictive) method

Multiple graphs are used to describe the data visually over the course of time.

The predictive model is then visually displayed in comparison with the NWS forecast mean

Collected or available datasets

Both the NOAA and NWS API are pinged to fetch current or historic data via API. But the primary source of historic data comes from daily-history-sumaries.tar.gz and its collection of weather stations and their daily history summaries.

daily-summaries-latest.tar.gz

Decision support functionality

The application will provide decision support by translating weather conditions and forecasts into operational signals, such as threshold-based triggers, location-based impact awareness, and predictive indicators (next-day temperature/precip/wind estimates). This will allow leaders to make faster go/no-go decisions and plan staffing and operations based on forecasted risk.

Ability to support features, parsing, cleaning, and wrangling datasets

The system implements a repeatable ingestion pipeline that:

- Streams and parses station CSV files from the compressed tar archive without loading the full dataset into memory
- Validate schemas, enforce data types, and normalize units)
- Handles missing values and preserves or filters NOAA quality/source flags
- Stores cleaned records into a normalized Postgres/PostGIS schema with constraints and indexes to support analytics and API retrieval

Methods and algorithms supporting data exploration and preparation

Data exploration and summarizing is done through the machine learning service program, it manages predictions in predictions.py and pulls daily NOAA data from noaa.py. Simple filters are used to hide rows that are missing or include invalid data, this is managed in series.py.

Daily records are sorted and aligned by date so models learn the time pattern in series.py. Then the mean and delta are calculated from NOAA daily values in features.py, seasonal encoding then occurs and rolling content is applied such as last-7-day mean and trend, helping the model see recent changes in weather.

Data visualization functionalities for data exploration and inspection

Data is presented on a mapLibre map, the user may select a place on the map, placing a pin. More data is then visually presented to the user including forecasts for that point over the next 24 hrs. Historic data over last year, ML prediction graph for coming week. There is also a temperature and precipitation heatmap visualizing the data available on the map, not just a graph.

Implementation of interactive queries
The application will support interactive queries through the UI and API, including:

- Querying weather history by station and date range
- Querying forecast by gridpoint and time window
- Querying active alerts by bounding box/area, alert type, severity, and time validity
- Querying ML predictions by location and a forecast horizon
- Querying for external API updates

These queries are implemented through REST endpoints backed by indexed Postgres queries (including all spatial queries via PostGIS).

Implementation of machine-learning methods and algorithms

The system implements supervised gaussian curve learning through the gpr.py program to predict seasonal weather patterns. The system also implements a ridge regression forecast for daily weather predictions, within ridge.py.

Machine learning is made possible with the scikit library.

Functionalities to evaluate the accuracy of the data product

MAE/RMSE standard metrics used to evaluate regression model performance. They are used to compare against NOAA daily actuals to determine the accuracy of the data product. The /metrics/accuracy route can be used to compute the accuracy.

Industry-appropriate security features

The system includes standard security controls appropriate for an internal data product such as TLS termination through Caddy with HTTPS-only access along with a reverse proxy routing to restrict direct access to internal service ports. Secrets are stored in environment variables or protected configuration files (***not committed to source, NOAA API Token, DB password***). The application applies least-privilege database user accounts and role-based permissions and has firewall rules limiting exposure to required ports only (primarily 443). Extensive logging for audit and incident troubleshooting. UFW is used to manage ports easily.

Tools to monitor and maintain the product

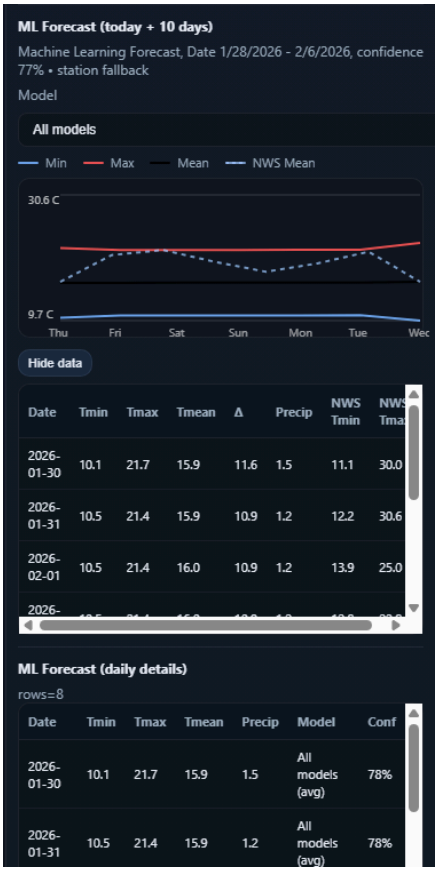
To manage the services, systemd service management is used for API and ML services (restart policies, auto-start on boot, service configuration). journald logging with persistent logs for debugging and traceability, helps with quantifying status of the backend and seeing API paths being handled. Health-check endpoints (API + ML) will be established to confirm service availability. Basic job/run tracking tables in Postgres are available to record ingestion runs, row counts, durations, and failures to better track system performance.

A user-friendly, functional dashboard that includes three visualization types

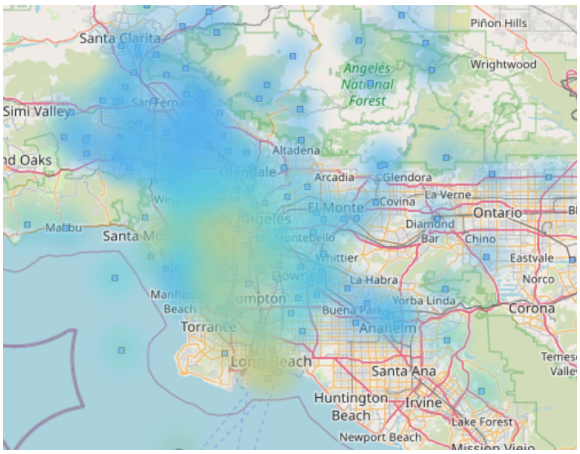
map.ketterling.space

The dashboard was designed for non-developer users while still supporting IT inspection. It includes several visualization types:

1. Geospatial map visualization
2. Time-series line chart
3. Heatmaps
4. Summary tables



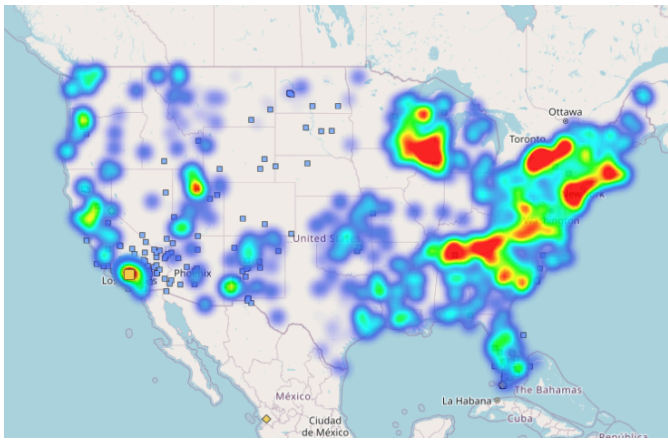
Temperature heatmap across los angeles:



The Dashboard supports tracking and naming specific locations using a simple CRUD interface.
All cards on the dashboard are collapsable to condense information.

Selecting a location gives you all the historic data, current data, and predictive data.

Precipitation heatmap:



Quick Start Guide: Weather Intelligence Decision-Support Application

1/23/2026
Taylor Ketterling

Quick Start Guide: *Deploying the Application*

This application will feature the use of multiple parts to work, starting with a front-end enabled through *Hostinger*(domain, webhost), a *firewalla purple* for network management and forwarding 443 and 80 port access to the *Latte Panda*. The *Latte Panda* will host the ML application, Database, API server, and a reverse proxy to help avoid exposing the APIs directly to the internet, the reverse proxy reduces exposed open ports and is made easy through a utility called Caddy.

Requirements:
Access to a webhost(This project uses Hostinger) for **domain** services and basic **file hosting** of the mapLibre front-end.

- Enables users to use the web app via a Domain and Web hosting services. *Caddy will cause issues with using local host File:c://… , included caddy file does not include work around to enable unauthorized origins. use either a local webhost or a service like hosting CORS allow origins must be set under the settings.py file.*

Access to a **compute device** that can run **Linux 24.04 LTS** (This project uses a Latte Panda Sigma with a 500GB M.2 Drive)

- If using a cloud based Virtual Machine, there are no additional networking requirements.
- If you are using a locally hosted compute device such as this project is, you must have a **networking device**, such as the firewalla purple, that can enable port forwarding; enabling the front-end hosted by Hostinger and the client's machine, to communicate with the backend running on the compute device.

The **front end** is a MapLibre based project (open source)

The **Back end** includes a python machine learning application which is using the (1) scikit-learn, Machine Learning in Python, opensource toolset. To store Historic data, store machine learning predictions, and to cache live data from the NWS API a (2) PostgreSQL (open source) Database will be used with the PostGIS (open source) extension which enables storing spatial indexes and other map related data types better. To enable reverse proxy and HTTPS functionality, (3) Caddy (open source) will be used, reducing the total open ports and preventing our local apps from direct internet exposure. The developed (4) Java REST API must be launched in order to enable front-end communication with backend resources. These 4 applications make up the backend which ultimately stores persistent data, enables Machine Learning models, and allows the front end to safely and securely communicate with the backend via the API and Proxy.

The OS the backend will run on is Ubuntu 24.04.3 LTS (open source)

Use the following steps 1 - 10 to setup and harden a Linux OS to serve as the back end for the Weather App to deploy systems identified on page one *(Backend services 1-4).*

Prereq an Ubuntu OS: How to Install Linux in 2024 - A Beginners Guide

In order to connect via shh I have pre-given the latte panda my .pub key via usb. I also know my latte panda's IP address both from ifconfig and the firewalla Firewall and Network controller. I have configured the device to be a static IP within firewalla.

Store your pub key in:

~/.ssh/authorized_keys

Guide to Keys and SSH Learn SSH In 6 Minutes - Beginners Guide to SSH Tutorial

With your pub key stored you can shh to the latte panda with:

ssh appadmin@192.168.108.170

This will help AFTER step 3, by letting you work from your desktop not your deployment device.

Step 1) Harden the OS by updating everything and enabling security upgrades:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y unattended-upgrades
```

then

```
sudo dpkg-reconfigure --priority=low unattended-upgrades
```

Say Yes after popup

This will prevent CVEs from getting unpatched and ensure the OS is ready for the project.

Step 2) Create an appadmin user and disable SSH passwordAuthentication as it can be susceptible to brute force, once created stop using root or any personal users for the project after setup.

```
sudo adduser appadmin
sudo usermod -aG sudo appadmin
sudo nano /etc/ssh/sshd_config
    >> Set or verify:
        PermitRootLogin no
        PasswordAuthentication no
        PubkeyAuthentication yes

sudo systemctl restart ssh
```

Step 3) Setup the basic firewall offered onboard a linux system using ufw.

```
sudo ufw default deny incoming
sudo ufw default allow outgoing

sudo ufw allow 22/tcp          # SSH   (allow us to interact and control the server from our desktop)
sudo ufw allow 443/tcp         # HTTPS (the API + reverse proxy)

sudo ufw enable
sudo ufw status verbose
```

Once completed check all enabled services and ensure ufw is one of them:

```
systemctl list-unit-files --type=service | grep enabled
```

sudo systemctl disable AngryBirds *#example of terminating an unknown service*

Install and enable fail2ban and check its status that it is running. This enables automatic IP banning for repeated failures of people attempting ssh connection.

```
sudo apt install -y fail2ban
sudo systemctl enable --now fail2ban
sudo fail2ban-client status
```

Step 4) Install Caddy and it's dependencies:

```
sudo apt install -y debian-keyring debian-archive-keyring apt-transport-https

curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/gpg.key' | sudo tee
/usr/share/keyrings/caddy-stable.asc

curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/debian.deb.txt' | sudo tee
/etc/apt/sources.list.d/caddy-stable.list

sudo apt update
sudo apt install -y caddy
```

Setup Caddy file so that no app ports are exposed publicly, allowing a separation between services, TLS is automatic:

sudo nano /etc/caddy/Caddyfile

```
api.ketterling.space {
 encode gzip

 handle_path /ml/* {
  reverse_proxy 127.0.0.1:8000
 }

 # Allowed API paths -> Java API
 @apiAllowed path / /health /api/* /layers/*
 handle @apiAllowed {
  reverse_proxy localhost:8080
 }

 # Drop all other requests
 handle {
  respond "Not found" 404
 }

 header {
  Strict-Transport-Security "max-age=31536000"
  X-Content-Type-Options "nosniff"
  X-Frame-Options "DENY"
 }
}
```

sudo systemctl reload caddy

sudo systemctl start caddy // if caddy is not running *caddy.service: Unit cannot be reloaded because it is inactive.*

Step 5) Setup the directory and assign them to appadmin, using {db,logs,env} creates multiple sub directories with weather-app for each sub service.

```
sudo mkdir -p /opt/weather-app
sudo mkdir -p /opt/weather-app/{db,logs,env}
sudo chown -R appadmin:appadmin /opt/weather-app
```

Here's what the files structure looks like:

```
/opt/weather-app/
 ├── db/        (DB scripts, migrations)
 ├── env/       (Environment Files(where to post 2 .env files)
 ├── WeatherApp/        (Imported project priles)
        ├── ml_service/         (Python Machine learning program / API)
        ├──weatherapp/          (Java API and Ingest server)
        ├──'Database init'/     (Database init sql)
        ├──public_html/         (Front-end)
 └── logs/      (a place for centralized logs)
```

I am hosting the public_html on hostinger, a cloud provider. map.ketterling.space

Step 6) Install Java and maven, and create a user for the API

```
sudo apt install -y openjdk-17-jdk maven
java -version
```

Create a dedicated user for the API , assign it to the weather-app/api directory

```
sudo useradd -r -s /usr/sbin/nologin weatherapi
sudo chown -R weatherapi:weatherapi /opt/weather-app/api
```

The api is then isolated and can be called via:

```
sudo -u weatherapi java -jar weather-api.jar
```

Step 7) Install latest python (3) and it's project handler pip, setup the virtual environment(venv), only use pip when in the virtual environment:

```
sudo apt install -y python3 python3-venv python3-pip
```

Create a virtual environment for the machine learning algorithm:

```
cd /opt/weather-app/WeatherApp/ml_service
python3 -m venv venv
source venv/bin/activate
```

Install the core open source dependency for the Machine Learning model:

```
pip install fastapi uvicorn pandas numpy scikit-learn joblib
deactivate
```

Step 8) Now lets focus on the database, getting the postgreSQL and PostGIS dependencies and hardening the database, then creating the weatherapp user that has access to the weatherdb:

```
sudo apt install -y postgresql postgresql-contrib postgis
```

After installing postgres(or if it's pre pre-installed), the postgres OS user will exist. Create a new database user and assign a password; write this down somewhere and update env variable(1) and env variable(2)

```
sudo -i -u postgres
psql
CREATE USER weatherapp WITH PASSWORD 'supeRduppeRstrongpassword';
CREATE DATABASE weatherdb OWNER weatherapp;
\c weatherdb
CREATE EXTENSION postgis;

exit
```

Lock down the DB to local network access by adjusting:

```
sudo nano /etc/postgresql/*/main/postgresql.conf
```

and setting:

```
listen_addresses = 'localhost'
```

restart to apply changes:

```
sudo systemctl restart postgresql
```

Step 9) Enable logging and monitoring, starting with enabling journald persistence.

```
sudo mkdir -p /var/log/journal
sudo systemctl restart systemd-journald
```

You can view logs by using:

```
journalctl -u caddy
journalctl -u weather-api-service
```

Step 10) Go to the fridge and have a drink. The OS and system are ready for the 4 services to be launched. Make sure to write down any created credentials so they are not lost.

Welcome back from the fridge. The first service, the Caddy reverse proxy has already been enabled, congrats. you can check that it is running by:

```
sudo systemctl status caddy
```

The Java ingest/API service, along with the python ML API both interact with the DB, so let's set that up next:

Setting up the Database

Use the 01_init_weatherapp.sql file to initialize the database:

Ensure the .sql file is owned by postgres or you will run into permission errors:

```
sudo chown postgres:postgres 01_init_weatherapp.sql
sudo chmod 600 init.sql

sudo -u postgres psql -d weatherdb -f 01_init_weatherapp.sql
```

Then grant appropriate rights to the weatherapp db user:

```
sudo -u postgres psql -d weatherdb

GRANT CONNECT ON DATABASE weatherdb TO weatherapp;
GRANT USAGE ON SCHEMA public TO weatherapp;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO weatherapp;
ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO weatherapp;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO weatherapp;
GRANT SELECT, INSERT, UPDATE, DELETE ON cached_grid_agg TO weatherapp;
```

Note: I had to run the init file twice, constraints were satisfied the second run once the tables were created from the first run. We will no longer interact with the postgres user, rather instead use the weatherapp user for all DB interactions.

Deploying the Java API

This is a maven java project and must have maven installed, then you can compile and run the project to enable the weatherapp API. Make sure to update *application.properties* to reflect correct database credentials and NWS user information. NWS API has you sign your requests with some basic info.

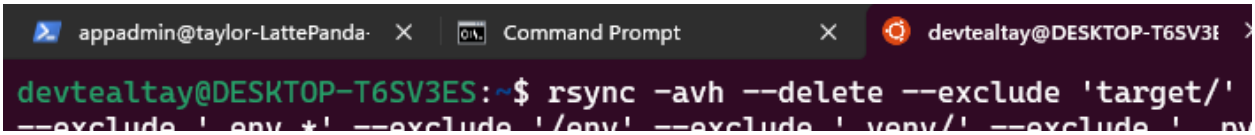
if maven is not installed use:

sudo apt install maven

To copy the maven project from a desktop to the Linux server over ssh use:

scp -r "C:\Users\Taylor\Desktop\WeatherApp" appadmin@192.168.108.170:/opt/weather-app/

I had to use a linux dev terminal on windows to use scp and rsync commands to send code from my desktop to my Latte Panda(dev device).



Set the env variables:

sudo nano /opt/weather-app/env/weatherapp-ml.env
sudo chown root:root /opt/weather-app/env/weatherappl.env
sudo chmod 600 /opt/weather-app/env/weatherapp.env

once the maven project has been copied over it must be compiled in order to run:

cd /opt/weather-app/WeatherApp/weatherapp/
maven -q package

then run the project by calling the created jar file:

java -jar target/*-all.jar the API will then be listening on port 8080

Leaving the API as a launchable jar file is an option, and you can call it as shown above. Instead lets create **a service** that monitors its status and launches at runtime(so on reboots).

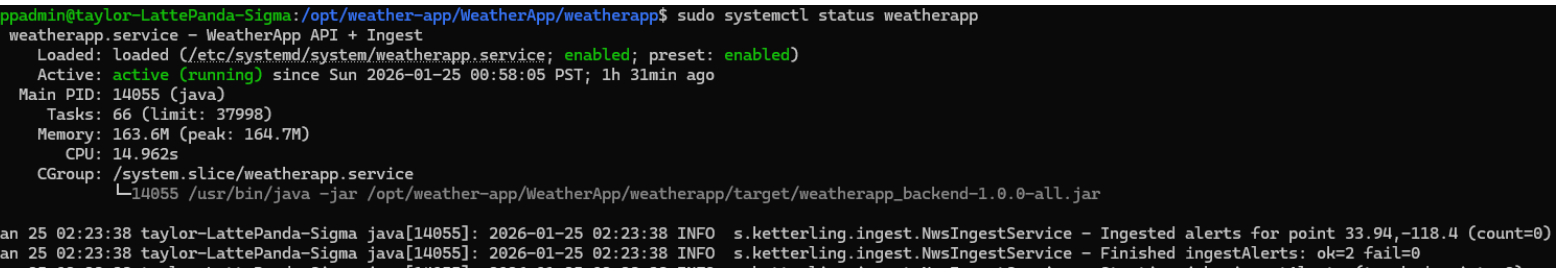
sudo nano /etc/systemd/system/weatherapp.service

```
[Unit]
Description=WeatherApp API + Ingest
After=network.target postgresql.service

[Service]
User=appadmin
WorkingDirectory=/opt/weather-app/WeatherApp/weatherapp
ExecStart=/usr/bin/java -jar
/opt/weather-app/WeatherApp/weatherapp/target/weatherapp_backend-1.0.0-all.jar
Restart=always
RestartSec=3
Environment=JAVA_OPTS=-Xms512m -Xmx6g -XX:ActiveProcessorCount=10
CPUQuota=400%
MemoryMax=6G
TasksMax=2048
LimitNOFILE=65535
NoNewPrivileges=true
PrivateTmp=true
ProtectSystem=strict
ProtectHome=true

[Install]
WantedBy=multi-user.target
```

sudo systemctl daemon-reload
sudo systemctl enable --now weatherapp
sudo systemctl status weatherapp



We can then monitor the output of the API/ingest program using:

journalctl -u weatherapp -f

Update the program using **rysnc** rather than scp, this comprehensive one ignores various files to avoid over riding them on the device:

rsync -avh --delete --exclude 'target/' --exclude '.git/' --exclude '.idea/' --exclude '.vscode/' --exclude '*.properties' --exclude '*.csv' --exclude '.env' --exclude '.env.*' --exclude '/env' --exclude '.venv/' --exclude '**/venv/**' --exclude '__pycache__/' --exclude '*.pyc' /mnt/c/Users/Taylor/Desktop/WeatherApp/ appadmin@192.168.108.170:/opt/weather-app/WeatherApp/

I know it's a long command, but this helps you avoid destroying that venv you just created, and any env variables, it also prevents sending the .properties file to the Latte Panda.

Build the Machine Learning Program Environment:

Create its env variables:

sudo nano /opt/weather-app/env/weatherapp-ml.env

```
DB_HOST=127.0.0.1
DB_PORT=5432
DB_NAME=weatherdb
DB_USER=weatherapp
DB_PASSWORD=YOUR super SecreET PASSWORD

HOST=127.0.0.1
PORT=8000
```

Change the owner to root and set it to read and set it 600 (-rw-------)

```
sudo chown root:root /opt/weather-app/env/weatherapp-ml.env
sudo chmod 600 /opt/weather-app/env/weatherapp-ml.env
```

```
cd /opt/weather-app/WeatherApp/ml_service
python3 -m venv .venv
./.venv/bin/pip install --upgrade pip
./.venv/bin/pip install -r requirements.txt
```

Create the Service:

sudo nano /etc/systemd/system/weatherapp-ml.service

enter>>

```
[Unit]
Description=WeatherApp ML Service (FastAPI)
After=network.target postgresql.service
Wants=postgresql.service

[Service]
User=appadmin
WorkingDirectory=/opt/weather-app/WeatherApp/ml_service
EnvironmentFile=/opt/weather-app/env/weatherapp-ml.env

# FastAPI app lives in app.py and is named "app" -> app:app
ExecStart=/opt/weather-app/WeatherApp/ml_service/.venv/bin/python -m uvicorn app:app --host
${HOST} --port ${PORT} --workers 4
Environment=OMP_NUM_THREADS=8
Environment=MKL_NUM_THREADS=8
Environment=OPENBLAS_NUM_THREADS=8
Environment=NUMEXPR_NUM_THREADS=8
CPUQuota=900%
MemoryMax=8G
TasksMax=4096
LimitNOFILE=65535

Restart=always
RestartSec=3

NoNewPrivileges=true
PrivateTmp=true
ProtectSystem=strict
ProtectHome=true

[Install]
WantedBy=multi-user.target
```

Then turn the service on with:

sudo systemctl daemon-reload && sudo systemctl enable --now weatherapp-ml && sudo systemctl status weatherapp-ml --no-pager && journalctl -u weatherapp-ml -f

To restart the ML service / ML API:

sudo systemctl restart weatherapp-ml && sudo systemctl status weatherapp-ml && journalctl -u weatherapp-ml -f

Example the system is running properly:
Java API / Ingest will report successfully ingests from the tar archive file:

Jan 29 09:53:11 taylor-LattePanda-Sigma java[74509]: 2026-01-29 09:53:11.066 [] [startup-local-historic] INFO s.k.ingest.StationHistoricCsvIngest - Ingested archive entry MXN00014173.csv -> newRows=10728

```
Jan 29 10:51:49 taylor-LattePanda-Sigma java[74509]: 2026-01-29 10:51:49.065 [] [startup-local-historic] INFO  s.k.ingest.StationHistoricCsvIngest - Ingested archive entry UZM00038586.csv -> newRows=5726
```

(this archive ingest has been running for days, so far it has imported 3270 historic records of stations and their data, ranging from hundred to thousands of rows of daily summaries).

I originally built a csv ingest, but kept crashing my computer trying to drag all the csv files from the archive to my project, I then found you can work with tar.gz files directly through a buffer (meaning it won't slam your RAM, the archive file is 7.2 GB). While my Latte Panda has 16GB of RAM, the Java VM is running max 0.5 GB and won't happily handle a 7GB file without utilizing buffering. Now I just upload the daily-summary-archive.tar.gz, on startup the Program begins ingesting all the individual compressed .csv record files out of the tar.gz . Each record is ingested and reported in a log,

The python program will report the forecast progress after several thousand command line outputs of it tuning its model.

```
Jan 29 13:13:34 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:13:34,150 INFO weatherapp-ml - Forecast bootstrap progress: station_preds=9200 skipped=1472
Jan 29 13:13:39 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:13:39,454 INFO weatherapp-ml - Forecast bootstrap complete: station_preds=9200 grid_preds=560 skipped=1472
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:14:19,374 INFO weatherapp-ml - OPTIONS /predict/store -> 200 (0 ms)
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: INFO:     192.168.108.1:0 - "OPTIONS /predict/store HTTP/1.1" 200 OK
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:14:19,380 INFO weatherapp-ml - Predict store request date=2026-01-25 neighbors=8 source_type=point
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:14:19,392 INFO weatherapp-ml - POST /predict/store -> 200 (12 ms)
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: INFO:     192.168.108.1:0 - "POST /predict/store HTTP/1.1" 200 OK
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:14:19,423 INFO weatherapp-ml - OPTIONS /predict -> 200 (0 ms)
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: INFO:     192.168.108.1:0 - "OPTIONS /predict HTTP/1.1" 200 OK
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:14:19,429 INFO weatherapp-ml - Predict POST request date=2026-01-25 neighbors=8
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:14:19,440 INFO weatherapp-ml - POST /predict -> 200 (11 ms)
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: INFO:     192.168.108.1:0 - "POST /predict HTTP/1.1" 200 OK
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: 2026-01-29 13:14:19,492 INFO weatherapp-ml - OPTIONS /weather/latest -> 200 (0 ms)
Jan 29 13:14:19 taylor-LattePanda-Sigma python[365623]: INFO:     192.168.108.1:0 - "OPTIONS /weather/latest?sourceType=point&sourceId=null&lat=33.938286281703725&lon=-118.38421
```

Historic NOAA data →

Not every NOAA station has relevant data, some stations are skipped because they have irrelevant data to the modeling task at hand. According to NOAA , most stations are precipitation stations.

ML Forecast prediction →

Tip: I found this to be a clean way to restart both of my services. Depending on the service I want to monitor immediately after restarting, I modify the journalctl command:

When inside the maven project root:

```
cd /opt/weather-app/WeatherApp/weatherapp
```

Rebuild the java project, reload the daemon to fetch new variables, restart both services and print their startup status. Finally start a jounalctl feed on one of the services to see the program output:

```
mvn -q clean package && sudo systemctl daemon-reload && sudo systemctl restart weatherapp && sudo systemctl status weatherapp && sudo systemctl restart weatherapp-ml && sudo systemctl status weatherapp-ml && journalctl -u weatherapp-ml -f
```

Vision Statement: Weather Intelligence Decision-Support Application

Weatherapp , Prototype
Owner: Taylor Ketterling
1/24/2026

Vision statement: Deliver a *"single pane of glass"* web app that centralizes NOAA/NWS weather data into a map-driven interface, adds relevant operational context, and includes Machine Learning based predictive indicators so organizations can make faster, more consistent weather-related decisions.

Problem statement: Organizations with weather-exposed operations experience uncertainty because weather info is consumed from fragmented sources (phone apps, delayed alerts, generalized forecasts) with limited geographic + operational context, leading to slower response times and inconsistent decision-making around tolerances.

Opportunity: The Weatherapp can turn raw public weather data into an operationally usable data product via a centralized, map-based weather/climate awareness(historical/visual) and non-descriptive (ML predictive) features to support proactive planning instead of reacting to predictable events.

Goals of the Weatherapp include the centralization of NOAA station history and the incorporation of NWS live data. along with map-based situational awareness indicators such as alerts and forecasts. and finally to offer predictive models that are trained on stored data.

Metrics to identify success include predictions meeting the target of 90% accuracy. Data ingestion is repeatable and resumable, allowing for large archives to be ingested and fail over capacity. The UI correctly renders the available data and API returns all expected payloads from behind the reverse proxy.

Misc Items: Weather Intelligence Decision-Support Application

Weatherapp , Prototype
Owner: Taylor Ketterling
1/29/2026

Provide raw and cleaned datasets with the code and executable files used to scrape and clean data (if applicable):

Raw dataset (source data)

The raw dataset is the NOAA daily summaries archive: daily-summaries-latest.tar.gz. This archive contains many station CSV files, where each CSV holds daily records for a single station

How raw data is scraped/ingested

The application reads the tar archive using a buffered stream rather than extracting the entire archive to disk or loading it into memory. As each station CSV entry is encountered, the program parses records row-by-row. To prevent duplicate inserts and reduce processing time, ingestion is incremental. The program checks whether the station CSV contains new daily rows not currently stored. If new records exist, only the new rows are parsed and inserted into the database, If no new records exist, the station file is skipped.

Cleaned dataset (processed data)

The cleaned dataset is the standardized, validated representation of the raw station records after preprocessing and normalization. Cleaning includes type normalization (dates, numeric fields), handling missing values / “null-like” values, and removing or skipping invalid rows (such as bad dates). The data is further cleaned through enforcing consistent units/column mapping and deduplication via database constraints/keys. Cleaned results are stored in PostgreSQL tables (noaa_daily_summary) and are used as the source for downstream analytics and ML feature generation.

The method `ingestDailySummariesTarGzIfPresent` is responsible for handling compressed tar files and extracting new rows of data out of the csv and upserting the data into the Database.

Provide code used to perform the analysis of the data and construct a descriptive, predictive, or prescriptive data product:

GPR model training

Ridge model training

Machine Learning Program README.md

Assessment of the product's accuracy

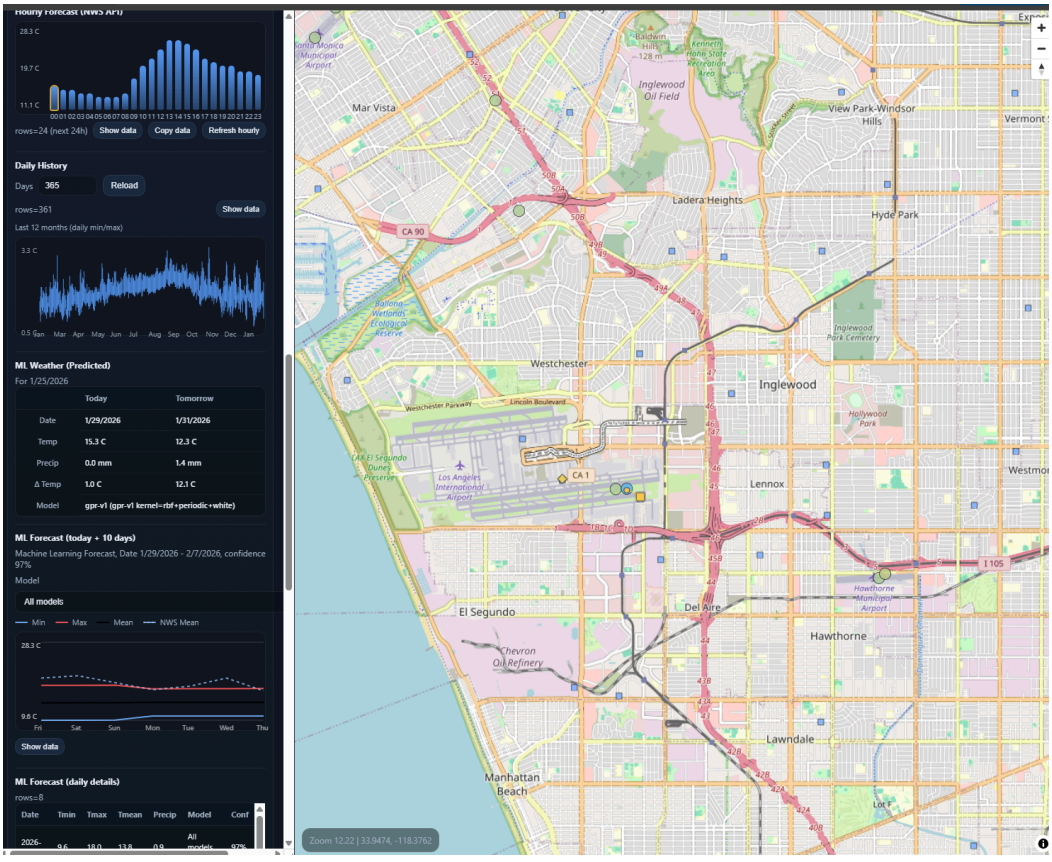
The Weatherapp includes a predictive data product (machine learning regression) that estimates next-day weather variables for a selected location. Because these are continuous numeric outputs, "accuracy" is assessed using regression error metrics rather than classification accuracy. The model is evaluated by comparing predicted values to actual observed values from the NOAA daily summary dataset. The dataset is split into training and testing data so evaluation is performed on unseen records. Then model predictions are generated for the test set and error is measured using standard regression metrics MAE and RMSE.

A lower MAE/RMSE score indicates better accuracy of the data model. Accuracy is considered acceptable if MAE/RMSE values are within a reasonable tolerance for the user's decision-making, and the model performs consistently across multiple evaluation runs and stations.

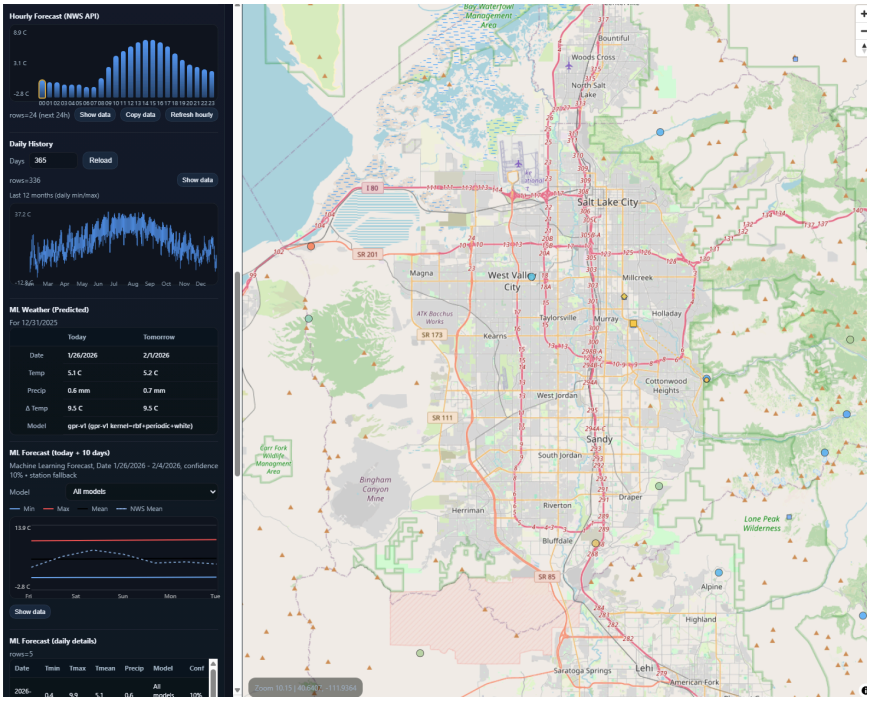
The function `accuracy_metrics` in predictions.py calculates the MAE/RMSE.

Results from the data product testing, revisions, and optimization based on the provided plans, including screenshots

map.ketterling.space



Looking at WGU's campus we can click the map to pull data



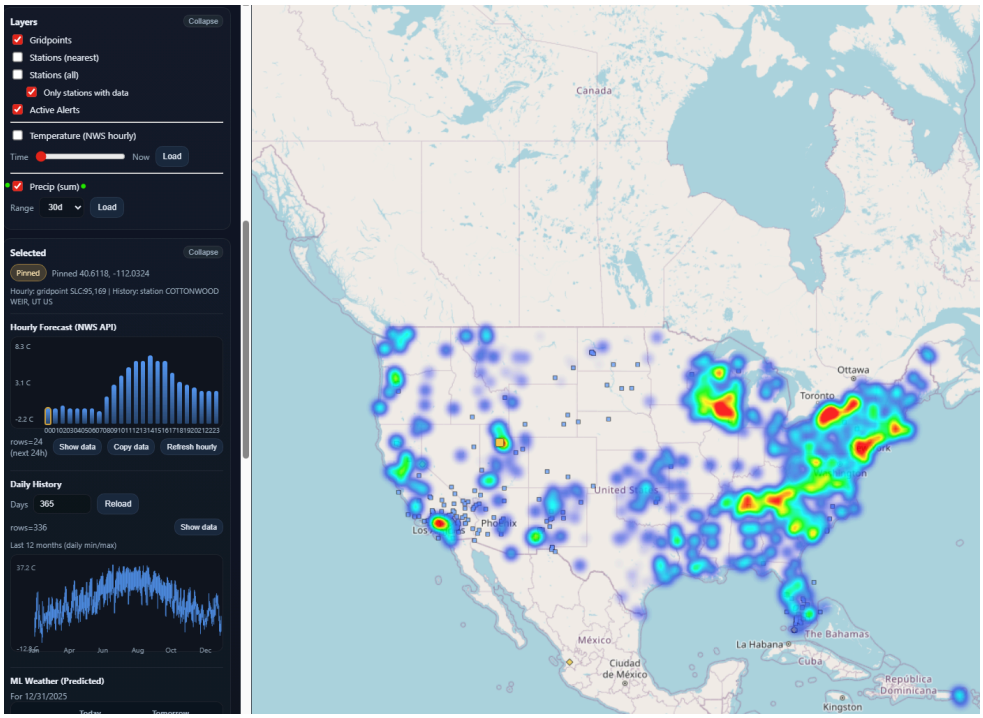
It will pull data from the nearest station *with* data, in this case COTTONWOOD WEIR, UT US. We can see what station our pinpoint is collecting data from via the dotted yellow line.

Date	Tmin	Tmax	Tmean	Precip	Model	Conf
2026-01-31	0.4	9.9	5.1	0.6	All models (avg)	10%
2026-02-01	0.4	10.0	5.2	0.7	All models (avg)	10%
2026-02-02	0.4	10.0	5.2	0.7	All models (avg)	10%
2026-02-03	0.5	10.1	5.3	0.6	All models (avg)	10%
2026-02-04	0.5	10.2	5.3	0.5	All models (avg)	10%

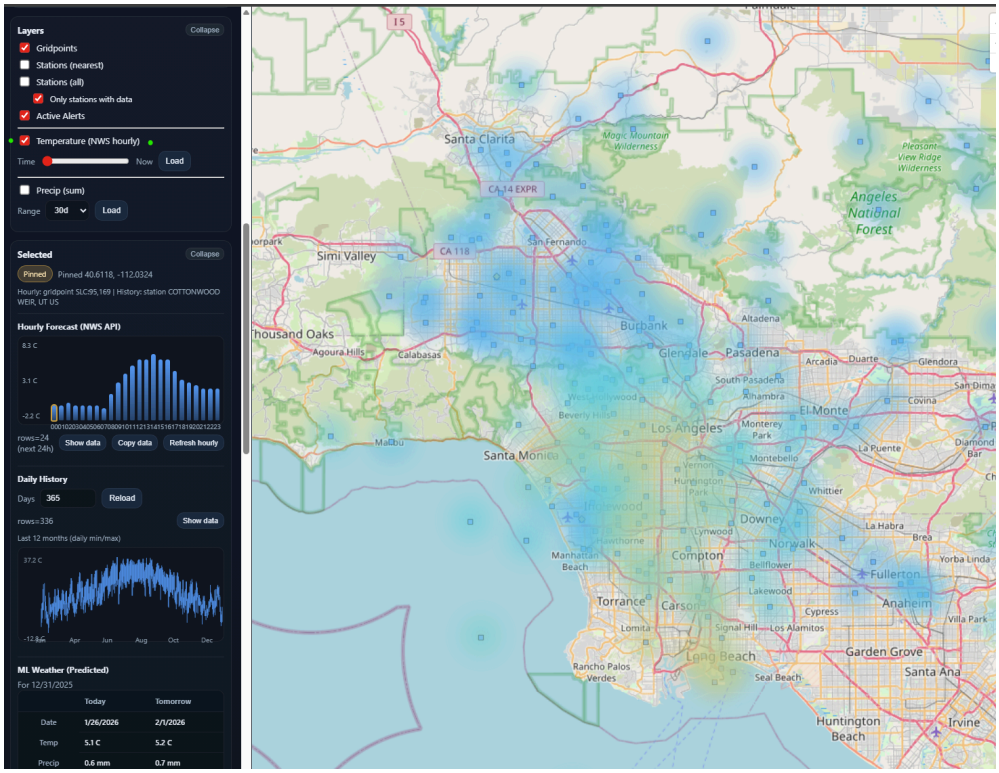
We can see the GPR prediction and confidence

The map also supports visualizing data as layers on the map.Temperature layer and precipitation layer can be checked to visual interpret data.

Precipitation sum across the US over the last 30 days:



The Temperature layer visualizes gridpoint NWS data.



You can create a grid point by clicking on the map, new grid points with pre-existing LOX grid points will be removed after creation. New LOX grid points will be kept. LOX grid points represent NWS API accessible data.

Acknowledgements

Websources:

"1.7. Gaussian Processes — Scikit-Learn 0.21.3 Documentation." Scikit-Learn.org, 2015, scikit-learn.org/stable/modules/gaussian_process.html.

- Utilized for research on the Scikit library and using gaussian regression to predict curves in a future x direction(weather forecast).

Mentor

Corbin Beebe, m.s. Software Engineering, Application Reliability Engineer at Natera, linkedin.com/in/corbin-beebe

- Supported midproject in advising on how to best break down the then extensively long Java program from a monolithic structure into modules. Mentor's advice has directly impacted the file structure chosen which has supported enabling my first multi-language, multi-service project.

Course Experience

Linux Foundations - D281

- Experience learned from this course allowed me to comfortably approach a SBC, my Latte Panda Sigma, and confidently set up a secure and capable Linux environment that ultimately delivered on hosting 4 simultaneous services.

Advanced Data Management - D326

- This course paved my foundations on how to effectively manage data through a postgres database system and write sql to interact with a database.

Data Structure and Algorithms II - C950

- Learned to write self improving applications, experience that supported when researching to use reinforce machine learning libraries on data to create predictions.

Software Design and Quality Assurance - D480

- Through this course I learned to identify and translate business requirements into software designs and identified required software systems and dependencies.