Workshop 2: Data System Architecture and Information Retrieval
Information Requirements for AgroClima Prediction System

César Andrés Torres Bernal (20191020147)
Juan David Duarte Ruiz (20191020159)

Universidad Distrital Francisco José de Caldas

High-level Architecture

The AgroClimaIQ platform is designed to operate as a scalable, data-driven system capable of ingesting, processing, storing, and delivering actionable climate intelligence to agricultural stakeholders. Given the system's reliance on large volumes of external data and personalized recommendations, a robust and modular data system architecture is essential.

This architecture follows a layered design pattern that supports continuous data ingestion from external APIs and simulated sources, distributed storage of structured and unstructured data, real-time and batch data processing, and secure delivery of insights through APIs and user interfaces. The architecture also integrates a Business Intelligence (BI) layer to enable data visualization and strategic decision-making. Each component in the system plays a specialized role—from collecting and validating weather data, to executing climate risk models, to generating field-specific recommendations. The architecture ensures high availability, scalability, and modularity to support a wide range of user roles, from farmers in the field to analysts and system administrators.

The following section shows the high-level architecture diagram and describes each architectural layer, its key components, the technologies employed, and how data flows between them to enable real-time decision support and long-term agricultural planning.

High-level Architecture Diagram

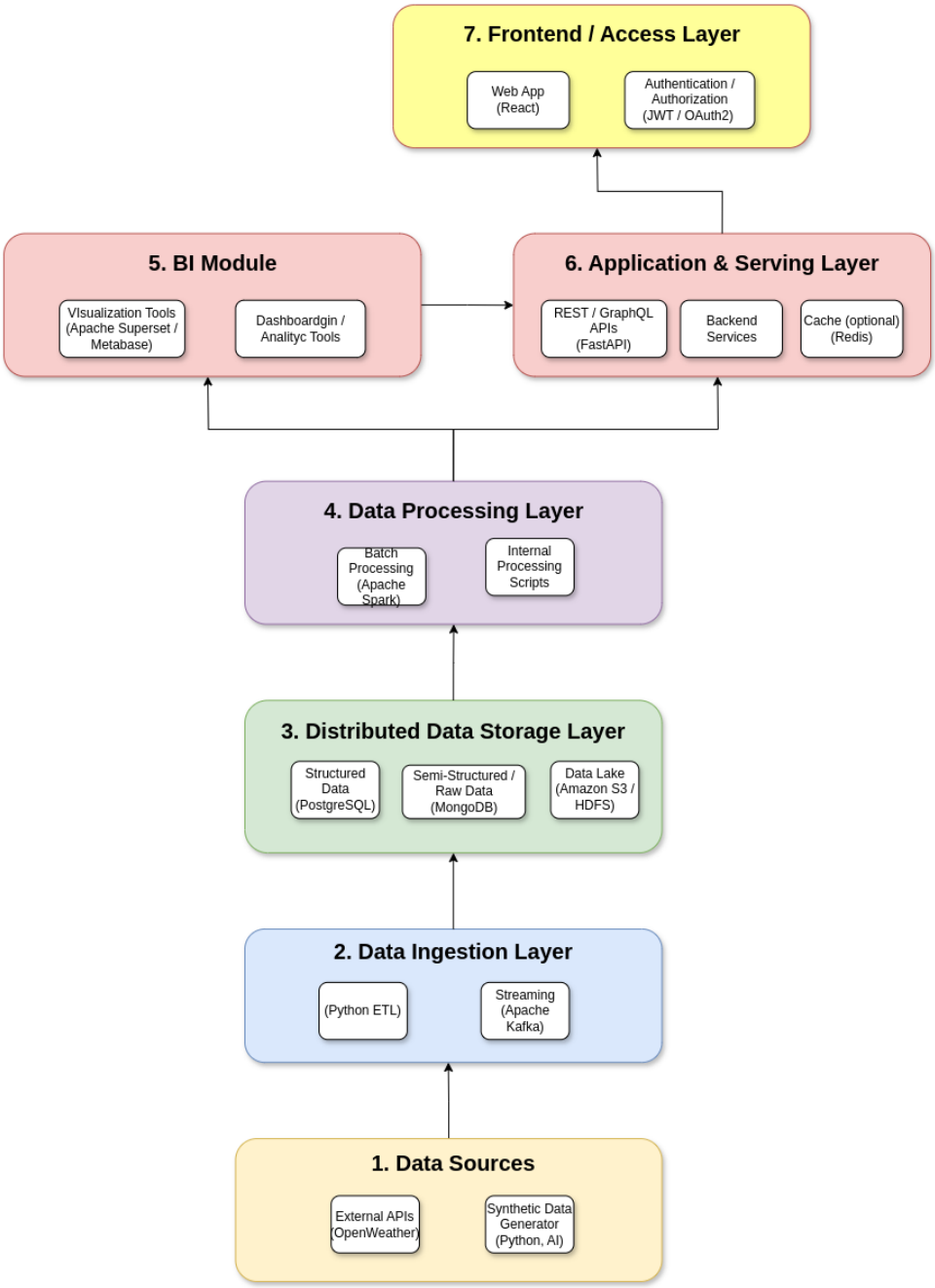


Figure 1: High-level Architecture Diagram

1. Data Sources Layer

Role: This layer serves as the entry point for all raw data entering the system. It is responsible for retrieving weather, climate, and agricultural data, both from external sources and synthetic generators.

Components & Technologies:

- **External APIs:** `OpenWeather`, `OpenMeteo`, `NOAA` – provide real-time and historical weather and climate data.
- **Synthetic Data Generator:** Python scripts using libraries such as `Faker`, `NumPy`, or `Pandas` – simulate agricultural data in academic or testing scenarios.

Interaction: These sources feed data directly into the ingestion layer through scheduled or real-time calls.

2. Data Ingestion Layer

Role: Responsible for acquiring, normalizing, validating, and routing incoming data from all sources to the system's internal data stores.

Components & Technologies:

- **Apache Kafka** (optional): For high-throughput, real-time stream ingestion where data events are published and consumed asynchronously.
- **Python ETL Scripts / Apache Airflow:** Used for scheduled jobs that run data transformation and enrichment pipelines.

Interaction: Data flows from external APIs or synthetic data generators → is collected and processed via custom Python ETL scripts → and is then stored directly in PostgreSQL (for structured data) and MongoDB (for unstructured or semi-structured data). Kafka (if used) serves to decouple real-time data streams from downstream processing and storage components.

3. Data Storage Layer

Role: This layer stores both raw and refined data, allowing persistence and long-term access. It includes both relational and `NoSQL` solutions.

Components & Technologies:

- **PostgreSQL:** Stores structured relational data: users, crops, alerts, predictions, sessions. Chosen for its `ACID` compliance, query performance, and relational integrity.
- **MongoDB:** Stores semi-structured data: `API` responses, recommendation documents, user feedback, and activity logs. Chosen for its schema flexibility and fast querying of document-based data.
- **Data Lake** (Optional): `Amazon S3` or local `HDFS` bucket for storing raw bulk data (e.g., climate archives) if needed.

Interaction: Data from ingestion flows into `PostgreSQL` for relational entities and into `MongoDB` for flexible or nested documents. These stores are later accessed by the processing and application layers.

4. Data Processing & Intelligence Layer

Role: Transforms raw data into useful insights. Executes `ML` models, risk analysis, climate forecasting, and generates recommendations.

Components & Technologies:

- **Apache Spark:** Performs batch processing on climate data, joins datasets from multiple sources, applies complex transformations.

– **Internal Processing Scripts:** Handle logic for alert generation, anomaly detection, and cleaning pipelines.

Interaction: Reads input data from `PostgreSQL` and `MongoDB` → runs models and transformations → writes back results to `PostgreSQL` (structured outputs like predictions) or `MongoDB` (rich recommendations and alerts).

5. Business Intelligence (BI) Module

Role: Provides data visualization and decision-making tools for analysts and administrators.

Components & Technologies:

– **Metabase / Apache Superset** Dashboarding and analytics tools connected to `PostgreSQL` and `MongoDB` for generating graphs, `KPIs`, reports.

Interaction: `BI` tools query `PostgreSQL` for analytics tables or materialized views and query `MongoDB` for custom visualizations or log-based dashboards.

6. Application & Serving Layer

Role: Serves processed data to external applications and end-users through interfaces and `APIs`. Implements business logic and manages user access.

Components & Technologies:

– **FastAPI** (Python): `REST/GraphQL APIs` expose data from both `PostgreSQL` and `MongoDB` securely.

– **Backend Services:** Handle user session management, request routing, access control, and processing `API` calls.

– **Redis** (optional): Used for caching frequently accessed predictions or alerts.

Interaction: `APIs` call `PostgreSQL` (for structured data like user profiles and prediction logs) and `MongoDB` (for dynamic recommendation responses). `Redis` may cache common or urgent responses for performance.

7. Frontend / Access Layer

Role: This is the interface through which users interact with the system – whether via web dashboards or mobile devices.

Components & Technologies:

– **React** Responsive `UIs` for displaying predictions, recommendations, alerts, and reports.

– **JWT / OAuth2:** Authentication and authorization systems ensure secure access per role (e.g., farmer, analyst, admin).

Interaction: The frontend communicates with the `API` layer → receives processed data from `PostgreSQL` or `MongoDB` → renders information tailored to the user's location, role, and preferences.

Information Requirements

The system must be able to retrieve various types of information to support both business needs and user stories effectively. Below are the key types of information that the system must retrieve:

User Information and Role Management

- **Description:** The system must store and manage detailed information about each user, including personal identification data (name, email, password hash), role (e.g., farmer, analyst, administrator), associated agricultural fields, and user preferences (such as alert preferences or notification channels). The system must also associate users with system access controls to enforce role-based permissions.
- **Business Value:** User information is critical for delivering tailored experiences, restricting access to sensitive data, and segmenting recommendations, alerts, and dashboards. It also enables monitoring of system engagement and effectiveness at the individual or organizational level.
- **Stored In:**
  - **User:** stores primary user profile and role.
  - **Agriculture_Field:** linked to users to define field ownership.
  - **Alert:** linked to users for personalized climate notifications.
  - **Recommendation:** indirectly linked to the user via their associated fields.
  - **Auth_Session** (optional): to store login session and authentication data if implemented.
- **Related User Needs:**
  - Farmers receive field-specific alerts and recommendations.
  - Analysts access aggregated data and reports.
  - Admins manage user access, roles, and system activity.

Climate Risk Prediction Data

- **Description:** The system must store and retrieve data related to climate risk predictions, including event type (e.g., flood, drought, frost), location, probability, associated severity level, and prediction timestamp. Each prediction must be traceable to the machine learning model used for generation.
- **Business Value:** This information is core to the system's analytical capacity and value proposition, enabling proactive decision-making for users by anticipating adverse weather conditions.
- **Stored In:**
  - **Prediction** (risk event, probability, timestamp, location, model)
  - **Event_type**, **Severity_level**, and **Model** (supporting metadata)
- **Related User Needs:**
  - Farmers receive alerts and reports based on prediction outcomes.
  - Managers assess the likelihood of risks across regions they oversee.

Personalized Agricultural Recommendations

- **Description:** The system must store actionable recommendations linked to specific climate predictions and tailored to the crop type, field location, and user context. Each recommendation includes a textual description and reference to its originating prediction.
- **Business Value:** These recommendations transform insights into guidance, helping farmers plan irrigation, crop rotation, pest control, and other field-level actions to optimize outcomes.
- **Stored In:**

  - **Recommendation** (linked to Prediction)
  - **Agriculture_Field**, **Crop**, and **User** (to contextualize each recommendation)
- **Related User Needs:**
  - Farmers access field-specific suggestions.
  - Managers oversee aggregated recommendations for strategic coordination.

Real-Time and Historical Weather Data

- **Description:** The system must collect and manage granular weather data (temperature, humidity, wind speed, timestamp) for each weather station and location. This data should be both real-time and historically queryable.
- **Business Value:** Serves as the foundational input for generating accurate predictions and trend analysis. Enables historical reporting and supports scientific credibility of the system's outputs.
- **Stored In:**
  - **Weather_Data** (linked to Weather_Station and Location)
  - **Weather_Data_Source** (to track origin and ingestion timeline)
- **Related User Needs:**
  - Farmers adjust practices based on live updates.
  - Analysts use historical trends to improve forecasting models.

Agricultural Field and Crop Information

- **Description:** The system must maintain detailed records of users' agricultural fields, including crop type, field `ID`, and location coordinates. Each user can be associated with one or more fields.
- **Business Value:** Essential for personalizing predictions and recommendations. Field-specific characteristics help determine risk sensitivity and agricultural practices.
- **Stored In:**
  - **Agriculture_Field**, **Crop**, and **Location** (with foreign key relations to User)
- **Related User Needs:**
  - Farmers see data related to their own plots.
  - Farm managers analyze performance by field and crop type.

Alert and Notification Records

- **Description:** The system must store alert records that include which user received what alert, for which prediction, when it was sent, and the current status (e.g., delivered, read, acknowledged).
- **Business Value:** Ensures timely risk communication, allows users to track and act upon weather-related risks, and provides administrators with traceability for compliance and auditability.
- **Stored In:**
  - **Alert** (linked to Prediction and User)
- **Related User Needs:**
  - Farmers stay informed of imminent threats.
  - Admins verify alert coverage and effectiveness.

Raw API Response Storage

- **Description:** The system must optionally store raw `JSON` payloads from external weather `APIs` (such as `OpenWeather`) for auditing, reprocessing, and debugging purposes. These payloads may vary in structure depending on the source, `API` version, or requested endpoint.
- **Business Value:** Storing raw `API` responses ensures transparency, traceability, and reusability. It allows the system to:
  - Compare predictions based on past input
  - Debug failures in ingestion or modeling logic
- **Stored In:**
  - **MongoDB Collection:** `api_responses`

User Interaction Logs and Feedback

- **Description:** The system must store user interaction data and optional feedback related to recommendations, alerts, and platform usage. These logs include timestamps, device type, interaction type (e.g., viewed recommendation, dismissed alert, submitted feedback), and additional metadata like geolocation, response time, or comments. This information can vary in structure depending on the event or action.
- **Business Value:** Tracking user behavior helps evaluate the effectiveness of alerts and recommendations, identify usability issues, and improve the system over time. Feedback also provides insight into user satisfaction and areas for platform improvement.
- **Stored In:**
  - **MongoDB Collection:** `interaction_logs`

Query Proposals

This section details the key information requirements for the system, along with SQL queries for data retrieval and their business context, all this based on the ER diagrama presented on the initial database architecture:

User Information and Role Management

- **Description:** The system must store and manage detailed information about each user, including personal identification data (name, email, password hash), role (e.g., farmer, analyst, administrator), associated agricultural fields, and user preferences (such as alert preferences or notification channels). The system must also associate users with system access controls to enforce role-based permissions.
- **SQL Query (PostgreSQL):**

```sql
SELECT user_id, user_name, role, email
FROM User
WHERE role = Role;
```

- **Purpose:** Retrieves a list of all user with that role.
- **Insight:** Supports role-based access, user segmentation, and administration.
- **Use Case:** Analytics (for admin dashboards or user management tools).

Climate Risk Prediction Data

- **Description:** The system must store and retrieve data related to climate risk predictions, including event type (e.g., flood, drought, frost), location, probability, associated severity level, and prediction timestamp. Each prediction must be traceable to the machine learning model used for generation.
- **SQL Query (PostgreSQL):**

```sql
SELECT p.prediction_id, e.event_name, l.name AS location, p.probability, s.severity_type, p.timestamp
FROM Prediction p
JOIN Event_type e ON p.event_type_id = e.event_type_id
JOIN Severity_level s ON p.severity_level_type_id = s.severity_level_id
JOIN Location l ON p.location_id = l.location_id
WHERE p.timestamp >= CURRENT_DATE - INTERVAL '7 days';
```

- **Purpose:** Retrieves recent predictions by event type and location.
- **Insight:** Helps users understand imminent or frequent risks.
- **Use Case:** Analytics (historical or weekly reports for managers and dashboards).

Personalized Agricultural Recommendations

- **Description:** The system must store actionable recommendations linked to specific climate predictions and tailored to the crop type, field location, and user context. Each recommendation includes a textual description and reference to its originating prediction.
- **SQL Query (PostgreSQL):**

```sql
SELECT u.user_name, c.crop_name, af.field_id, r.recommen_descrip, p.timestamp
FROM Recommendation r
JOIN Prediction p ON r.prediction_id = p.prediction_id
JOIN Agriculture_Field af ON r.field_id = af.field_id
JOIN Crop c ON af.crop_id = c.crop_id
```

```
JOIN User u ON af.user_id = u.user_id
WHERE u.user_id = 1
ORDER BY p.timestamp DESC;
```

- **Purpose:** Retrieves the recommendations for a given user.
- **Insight:** Shows which actions have been recommended based on the current crop and field context.
- **Use Case:** Real-time access for end-users.

Real-Time and Historical Weather Data

- **Description:** The system must collect and manage granular weather data (temperature, humidity, wind speed, timestamp) for each weather station and location. This data should be both real-time and historically queryable.
- **SQL Query (PostgreSQL):**

```
SELECT wd.timestamp, wd.temperature, wd.humidity, ws.station_name, l.name AS location
FROM Weather_Data wd
JOIN Weather_Station ws ON wd.station_id = ws.station_id
JOIN Location l ON ws.location_id = l.location_id
WHERE wd.timestamp BETWEEN NOW() - INTERVAL '1 DAY' AND NOW();
```

- **Purpose:** Retrieves real-time weather data for the last 24 hours.
- **Insight:** Enables immediate decision-making for field operations like irrigation.
- **Use Case:** Real-time access for farmers and analytics for analysts.

Agricultural Field and Crop Information

- **Description:** The system must maintain detailed records of users' agricultural fields, including crop type, field ID, and location coordinates. Each user can be associated with one or more fields.
- **SQL Query (PostgreSQL):**

```
SELECT af.field_id, af.field_size, l.name AS location, c.crop_name, u.user_name
FROM Agriculture_Field af
JOIN Crop c ON af.crop_id = c.crop_id
JOIN Location l ON af.location_id = l.location_id
JOIN User u ON af.user_id = u.user_id
WHERE u.user_id = 1;
```

- **Purpose:** Retrieves crop and field information for a specific user.
- **Insight:** Used to personalize recommendations and predictions.
- **Use Case:** Analytics for reports and real-time access to visualize fields on the app.

Alert and Notification Records

- **Description:** The system must store alert records that include which user received what alert, for which prediction, when it was sent, and the current status (e.g., delivered, read, acknowledged).
- **SQL Query (PostgreSQL):**

```sql
SELECT u.user_name, e.event_name, a.status, a.sent_at
FROM Alert a
JOIN User u ON a.user_id = u.user_id
JOIN Prediction p ON a.prediction_id = p.prediction_id
JOIN Event_type e ON p.event_type_id = e.event_type_id
WHERE u.user_id = 1
ORDER BY a.sent_at DESC;
```

- **Purpose:** Displays all alerts received by a user, including status and associated risk.
- **Insight:** Helps users and admins track how alerts are being delivered and acknowledged.
- **Use Case:** Real-time access for farmers, analytics for admins.

User Interaction Logs and Feedback

- **Description:** The system must store user interaction data and optional feedback related to recommendations, alerts, and platform usage. These logs include timestamps, device type, interaction type (e.g., viewed recommendation, dismissed alert, submitted feedback), and additional metadata like geolocation, response time, or comments. This information can vary in structure depending on the event or action.
- **NoSQL Query (MongoDB):**

```javascript
db.interaction_logs.find({
  interaction_type: "recommendation_viewed",
  user_id: "u001"
}).sort({ timestamp: -1 }).limit(5);
```

- **Purpose:** Retrieves the 5 most recent interactions where the user with `ID u001` viewed a recommendation.
- **Insight:** Helps monitor user engagement with the recommendation system, understand behavior patterns, and identify which recommendations are being read or ignored.
- **Use Case:** Real-time access for activity monitoring in the user dashboard. Analytics for `UX` evaluation and system improvement.

Raw API Response Storage

- **Description:** The system must optionally store raw `JSON` payloads from external weather `APIs` (such as `OpenWeather`) for auditing, reprocessing, and debugging purposes. These payloads may vary in structure depending on the source, `API` version, or requested endpoint.
- **NoSQL Query (MongoDB):**

```javascript
db.api_responses.find({
  source: "OpenWeather",
  "location.name": "Cali",
  timestamp: { $gte: ISODate("2025-05-27T00:00:00Z") }
}).sort({ timestamp: -1 }).limit(10);
```

- **Purpose:** Retrieves the last 10 raw weather `API` responses from the `OpenWeather` source for the city of Cali.
- **Insight:** Supports auditability of predictions, debugging of data pipelines, and retraining of climate models using original `API` inputs.
- **Use Case:** Analytics for historical comparison and model validation. Data engineering use for reprocessing or feature extraction.

References

[1] Stonebraker, M., & Çetintemel, U. (2005). *"One size fits all": An idea whose time has come and gone*. Proceedings of the 21st International Conference on Data Engineering (ICDE), 2–11. IEEE.

[2] Marz, N., & Warren, J. (2015). *Big Data: Principles and best practices of scalable real-time data systems*. Manning Publications.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Workshop No. 1

Databases II

César Andrés Torres Bernal
20191020147
Juan David Duarte Ruiz
20191020159

Engineering Faculty

1. Business Model

CFI

Key Partners

- Weather data providers (OpenWeather, Copernicus, IDEAM).

- Manufacturers and distributors of IoT sensors for agricultural use.

- Ministries of Agriculture, government agencies and NGOs that promote sustainable practices.

- Universities and climate and agricultural research institutes, which provide knowledge and scientific validation.

- Payment platforms and rural banking entities, which facilitate payments and subsidies.

- Agricultural input distributors that could partner for integrated offers.

Key Activities

- Design and management of weather forecasting and advisory services, adapted to different agricultural profiles.
- Execution of awareness and technology adoption campaigns in rural areas.
- Management of strategic relationships with governments, cooperatives, NGOs and agricultural associations.
- Technical support: development and maintenance of the digital system to offer the service.

Key Resources

- Network of strategic alliances with actors in the agricultural and climate sector.
- Access to reliable databases and satellite coverage.
- Supporting resource:
  - AI and machine learning engine trained on historical and real-time data.
  - Distributed databases and data ingestion systems such as Kafka, Kinesis, BigQuery, etc.

Value Propositions

<u>For farmers:</u>
- Accurate and actionable recommendations based on local climate predictions.
- Reduction of losses due to extreme events such as drought, frost or excessive rainfall.
- Optimization of the use of resources such as water and fertilizers.
- Ease of use, even without technical training.

<u>For institutions and companies:</u>
- Access to regional data and reports for agro-climatic decision making.
- Tools for planning subsidies, agricultural insurance and technical support.
- Improved public policies and agroclimatic disaster mitigation.

Customer Relationships

- Multichannel technical assistance (chat, telephone, mail) for rural users with limited connectivity.
- Training and support in the adoption of the platform, especially in areas with low digitalization.
- Automated and personalized alerts, without the need for constant interaction.

Channels

- Web platform accessible in rural areas with low connectivity.
- API access for institutions wishing to connect their agricultural management systems.
- Institutional partners (agricultural agencies, cooperatives) as implementation and support channels.

Customer Segments

<u>Individual users:</u>

- Small and medium farmers interested in protecting their crops.
- Technified producers who integrate IoT in their agricultural management.

<u>Institutional customers:</u>

- Regional and national governments that manage climate risks in the agricultural sector.
- Agribusiness companies seeking to optimize production and climate logistics.

Cost Structure

- Operation of alliances, training and customer service in the territory.

- Access and processing of climate and satellite data.

- Marketing, campaigns and community training.

- Investment in continuous improvement of the regional recommendation and adaptation model.

- Technical support: maintenance and technological operation costs servers, cloud, support.

Revenue Streams

- Monthly or annual subscription of farmers, with staggered plans according to needs.

- Institutional licenses for local or national governments.

- Additional services (consulting, premium data, risk maps).

- Freemium model with free basic functionalities and advanced paid plans.

- Alliances with insurance companies or input companies, which sponsor access to the service.

Functional Requirements

Number	Requirement	Description	Priority
FR1	User registration	Allow registration of farmers, technicians and managers via email or federated authentication (Google, Microsoft).	High
FR2	Authentication and roles	Provide secure login with role-based access control to customize visible functionality based on user type.	High
FR3	Continuous data ingestion	Integrate data from open weather sources in real time using tools such as Kafka or AWS Kinesis.	High
FR4	Distributed storage	Store weather and agricultural data on a distributed, globally accessible basis with minimal latency.	High
FR5	Weather data queries	Allow users to visualize real-time and historical information about weather conditions in their area.	High
FR6	Predictive analytics	Run machine learning models to predict climate risks such as drought, frost or heavy rainfall.	High
FR7	Customized recommendations	Issue intelligent suggestions for planting, irrigation or fertilization based on microclimates and analyzed data.	Media
FR8	Report generation	Produce customized reports for agricultural and governmental decision makers.	Media
FR9	Administrative panel	Enable user managemen general system statistics by the administrator.	High

No Functional Requirements

Number	Requirement	Description	Priority
NFR1	Performance	Process queries with a maximum latency of 5 seconds for weather searches and predictions in agricultural areas.	High

NFR2	Horizontal scalability	Allow adding nodes to handle large volumes of data and increase the number of concurrent users.	High
NFR3	High availability	Guarantee 99.9% uptime with failover and automatic failover mechanisms.	High
NFR4	Multi-location access	Allow access from multiple geographic regions with minimal load times through the use of CDNs or distributed clusters.	Media
NFR5	Interoperability	Integrate with third-party APIs such as OpenWeather and Copernicus, as well as IoT devices using standard protocols.	Media
NFR6	Usability	Design a responsive, intuitive and fast web interface, accessible from different devices with low response times.	Media
NFR7	Maintainability	Have a modular and well-documented architecture that facilitates system upgrades, corrections and scaling.	Media

User Stories (Client Role)

Title:	**Priority:**	**Estimate:**
Register as a Client	High	3

User Story:

As a client, I want to register using my email account so that I can access climate data and receive tailored recommendations.

Acceptance Criteria:

- The system allows registration via email, Google, or Microsoft.
- Client role is assigned automatically upon successful registration.
- A confirmation email is sent upon account creation.

Title:	**Priority:**	**Estimate:**
View Local Weather Conditions	High	4

User Story:

As a client, I want to view real-time and historical weather data for my region so that I can plan activities effectively.

Acceptance Criteria:

- Weather data is displayed for the client's location.
- Data includes temperature, humidity, precipitation, and forecast.
- A time filter (e.g., last 7 days, 1 month) is available.

Title:	**Priority:**	**Estimate:**
Receive Weather Alerts	High	4

User Story:

As a client, I want to receive automatic alerts for upcoming extreme weather events in my area so that I can take preventive actions.

Acceptance Criteria:

- Alerts are triggered based on predictive models.
- Alerts are sent via email and displayed in the user dashboard.
- Each alert includes event type, severity, estimated time, and recommendations.

Title:	**Priority:**	**Estimate:**
Access Historical Recommendations	Medium	3

User Story:

As a client, I want to view past recommendations and actions taken so I can evaluate their effectiveness over time.

Acceptance Criteria:

- A history tab shows all past recommendations received.
- Each entry includes date, context, and whether the advice was followed.
- The user can filter history by date range or type (irrigation, fertilization, etc.).

Title:	**Priority:**	**Estimate:**
Receive Farming Recommendations	Medium	5

User Story: As a client, I want to receive personalized recommendations based on weather predictions and soil conditions so I can optimize my crop yield.
Acceptance Criteria: ● Recommendations are shown on the dashboard. ● Data includes suggestions on irrigation, fertilization, and planting. ● Alerts are sent for critical weather risks (e.g., frost, drought).

User Stories (Analyst)

Title:	**Priority:**	**Estimate:**
Access and Analyze Climate Data	High	4
User Story: As an analyst, I want to access real-time and historical climate data from all monitored stations so I can analyze environmental trends.		
Acceptance Criteria: ● A dashboard shows station data in real time. ● Filters allow data segmentation by location and date.		

Title:	**Priority:**	**Estimate:**
Generate Analytical Reports	Medium	3
User Story: As an analyst, I want to generate reports about climate conditions and risk trends to support decision-making for stakeholders.		

Acceptance Criteria:
● Reports can be generated monthly or weekly. ● Data visualizations (charts, graphs) are included.

User Stories (Administrator)

Title:	**Priority:**	**Estimate:**
Manage Users and Roles	High	3
User Story: As an administrator, I want to manage user accounts and assign roles so that access control is enforced throughout the platform.		
Acceptance Criteria: ● Admin dashboard lists all users with filters by role. ● Roles can be assigned or changed. ● User accounts can be deactivated or deleted.		

Title:	**Priority:**	**Estimate:**
Audit User Activity Logs	Medium	4
User Story: As an administrator, I want to access logs of user activities so I can ensure compliance and investigate anomalies.		
Acceptance Criteria: ● Logs include login times, data access, configuration changes. ● Filters by user, date, and activity type are available. ● Logs can be exported in CSV format.		

Initial Database Architecture

To propose a robust and scalable initial database architecture for AgroClima, considering the requirements for big data and distributed databases, it is essential to design a system capable of handling large volumes of geospatial and time-series data from various sources. The platform must support continuous data ingestion, efficient processing to generate real-time alerts, the generation of historical reports, and the execution of complex models for climate and agricultural predictions.

1. Data Sources:

- External APIs: Integration with global providers such as OpenWeather, OpenMeteo and NOAA.
- Historical Climate Data: Retrospective datasets from various sources for trend analysis and model training.

2. Data Ingestion Layer:

This layer is responsible for collecting data from diverse sources, which can vary in format, velocity, and structure. Given the big data nature, a scalable and fault-tolerant ingestion system is required.

- Apache Kafka or Amazon Kinesis: For ingesting real-time and streaming data from weather stations. They allow decoupling data sources from processing systems.
- ETL Tools: For processing data in batches from historical files, climate models, or satellite imagery. Examples: Apache NiFi, Talend, or cloud services like AWS Glue or Google Cloud Dataflow.

3. Distributed Data Storage Layer:

This layer is the responsible to storage all the information, information both raw and already processed for use, to handle the diversity and volume of data, a combination of distributed databases is proposed, each optimized for different data types

- Data Lake: To store raw data from various sources before being processed and refined.
    - Technologies: Apache Hadoop HDFS, Amazon S3, Google Cloud Storage, Azure Data Lake Storage. They provide scalable and cost-effective storage.
- Distributed Storage: Processed data will be stored in a distributed database such as Apache Cassandra, Google Bigtable, or Amazon DynamoDB.

4. Distributed Data Processing Layer:

This layer is responsible for transforming, analyzing, and deriving valuable information from the stored data, supporting both real-time and batch processing.

Technologies: Apache Spark / Apache Flink:

- Handles both batch and stream processing.
- Cleans, transforms, and prepares data for storage and analysis.

5. Serving and Application Layer:

This layer interacts directly with end-users (logistics and agricultural companies), providing access to alerts, reports, and predictions through a user interface or APIs.

- Serving Database: Databases optimized for fast and concurrent reads by the application. These can be materialized views or aggregations of processed data.
    - Suggested Technologies: Relational databases (PostgreSQL, MySQL) for structured data like reports and user profiles, and key-value or document NoSQL databases for fast access data (Redis, MongoDB).
- API Services: To expose platform functionalities to external applications or the user interface.

6. Access and API Layer REST and GraphQL APIs:

Serve data securely to mobile and web clients. Provide modular endpoints for different user roles (farmers, analysts, admins). The Access and API Layer in AgroClima provides secure and scalable interaction between the platform's backend and its user-facing applications. Using REST and GraphQL APIs, the system exposes modular and role-specific endpoints that allow farmers, analysts, and administrators to access data and services tailored to their needs.

ER Diagram (Initial Version)

Below is a description of each entity, with its respective attributes, for the initial version of the ER (Entity-Relationship) diagram for the AgroClima project.

User

Description: Represents the individuals registered in the AgroClima platform. Users may have different roles such as farmers, analysts, or administrators and are the recipients of climate alerts and predictions.

Attributes:

- user_id (INT, PK): Unique identifier of the user.
- user_name (VARCHAR(50)): Full name of the user.
- email (VARCHAR(100), UNIQUE): Email address used for login and notifications.
- role (VARCHAR(50)): User role (e.g., Farmer, Analyst, Admin).
- password_hash (VARCHAR(50)): Encrypted password for authentication.

Alert

Description: Stores records of alerts sent to users when a relevant climate event is predicted. Alerts are based on predictions generated by the system.

Attributes:

- alert_id (INT, PK): Unique identifier of the alert.
- user_id (INT, FK): User who receives the alert.
- prediction_id (INT, FK): Prediction that triggered the alert.
- timestamp_sent (DATETIME): Date and time the alert was sent.
- status (VARCHAR): Status of the alert (e.g., Sent, Acknowledged, Dismissed).

Prediction

Description: Stores the results generated by machine learning models that predict extreme weather events that may affect specific locations.

Attributes:

- prediction_id (INT, PK): Unique identifier of the prediction.
- timestamp (DATETIME): Date and time the prediction was generated.
- event_type_id (INT, FK): Type of event predicted (e.g., drought, flood).
- probability (DECIMAL): Estimated probability of occurrence (%).
- severity_level_id (INT, FK): Estimated severity level of the event.
- model_id (INT, FK): ID of the model used to generate the prediction.

Event_type

Description: Defines the types of weather events that the system can predict.

Attributes:

- event_type_id (INT, PK): Unique identifier of the event type.
- evet_name (VARCHAR(100)): Name of the event (e.g., Flood, Heatwave).
- event_description (VARCHAR(200)): Description of the event type.

Severity_level

Description: Categorizes the severity level of a predicted event, helping to prioritize alerts and responses.

Attributes:

- severity_level_id (INT, PK): Unique identifier of the severity level.
- severity_type (VARCHAR(50)): Name of the severity level (e.g., Low, Medium, High).
- level_description (VARCHAR(200)): Description of the severity level.

Model

Description: Contains metadata about the predictive models used by AgroClima, including versioning and training information.

Attributes:

- model_id (INT, PK): Unique identifier of the model.
- model_name (VARCHAR(100)): Descriptive name of the model.
- version (DECIMAL): Version number of the model.
- training_timestamp (DATETIME): Date and time of the model's last training.

Weather_Station

Description: Represents the weather stations (physical or virtual) that collect meteorological data used by the system to feed predictions.

Attributes:

- station_id (INT, PK): Unique identifier of the station.
- name (VARCHAR(100)): Name or code of the weather station.
- location_id (INT, FK): Reference to the location where the station is situated.

Weather_Data

Description: Stores individual weather readings collected by a weather station at a specific timestamp. These readings are key inputs for generating predictions.

Attributes:

- data_id (INT, PK): Unique identifier of the data entry.
- station_id (INT, FK): Weather station that generated the data.
- timestamp (DATETIME): Date and time of the reading.
- temperature (FLOAT): Temperature reading (°C).
- humidity (FLOAT): Relative humidity (%).
- wind_speed (FLOAT): Wind speed (km/h or m/s).

Location

Description: Represents geographic locations associated with weather stations or prediction areas. Enables spatial organization of environmental data.

Attributes:

- location_id (INT, PK): Unique identifier of the location.
- name (VARCHAR(100)): Name of the location (e.g., city, region).
- latitude (DECIMAL): Latitude coordinate.
- longitude (DECIMAL): Longitude coordinate.
- type (VARCHAR(100)): Type of location (e.g., Station, Critical Zone, Rural Area).

Agriculture_field

Description: Represents a physical agricultural field owned or managed by a user. Each field is geolocated and can be associated with one or more crops. This entity allows AgroClima to tailor predictions and recommendations to specific land plots.

Attributes:

- field_id (INT, PK): Unique identifier of the agricultural field.
- user_id (INT, FK): Reference to the user (farmer) who owns or manages the field.
- field_type (INT): Type or classification of the field (e.g., irrigated, rainfed, greenhouse).

Crop

Description: Stores the crops cultivated in each agricultural field. This allows the platform to provide crop-specific recommendations and monitor weather impacts on individual crop types.

Attributes:

- crop_id (INT, PK): Unique identifier of the crop record.
- field_id (INT, FK): Field where the crop is grown.
- crop_name (VARCHAR(200)): Name of the crop (e.g., maize, rice, coffee).

Recommendation

Description: Stores system-generated recommendations based on predictions and field conditions. These are delivered to farmers to support agricultural decisions such as irrigation, fertilization, or harvesting.

Attributes:

- recommend_id (INT, PK): Unique identifier of the recommendation.
- recommend_descrip (VARCHAR(300)): Description of the recommendation.
- prediction_id (INT, FK): Reference to the prediction that triggered the recommendation.

Data_source

Description: Defines the external or internal sources of meteorological data used in the system. This allows tracking of data provenance and managing integration with third-party APIs.

Attributes:

- source_id (INT, PK): Unique identifier of the data source.
- name (VARCHAR(100)): Name of the source (e.g., OpenWeather, Copernicus).
- type (VARCHAR(50)): Type of source (e.g., API, physical station).
- url (VARCHAR(100)): URL or endpoint of the data source.

Weather_data_source

Description: This is an associative entity that links weather stations to their corresponding data sources. It allows tracking of when a station began using a specific data source.

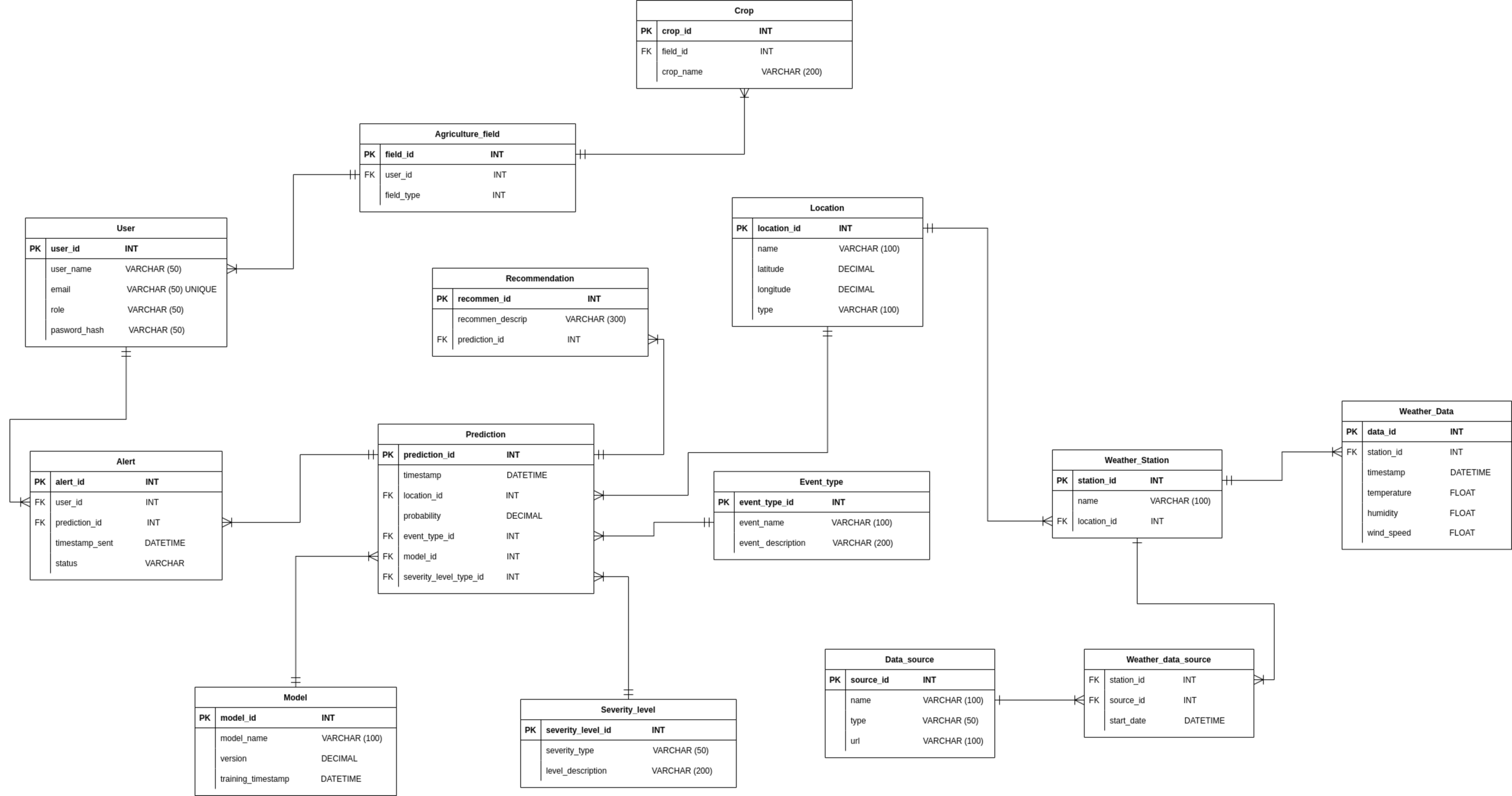
Attributes:

- station_id (INT, FK): Reference to the weather station.
- source_id (INT, FK): Reference to the data source.
- start_date (DATETIME): Date when this data source started being used for the station.

References

European Commission (2020). *Copernicus Climate Data Store (CDS) User Guide*.

- Link: https://cds.climate.copernicus.eu


Crop
PK crop_id INT
FK field_id INT
crop_name VARCHAR (200)
Agriculture_field
PK field_id INT
FK user_id INT
field_type INT
User
PK user_id INT
user_name VARCHAR (50)
email VARCHAR (50) UNIQUE
role VARCHAR (50)
pasword_hash VARCHAR (50)
Recommendation
PK recommen_id INT
recommen_descrip VARCHAR (300)
FK prediction_id INT
Location
PK location_id INT
name VARCHAR (100)
latitude DECIMAL
longitude DECIMAL
type VARCHAR (100)
Weather_Data
PK data_id INT
FK station_id INT
timestamp DATETIME
temperature FLOAT
humidity FLOAT
wind_speed FLOAT
Alert
PK alert_id INT
FK user_id INT
FK prediction_id INT
timestamp_sent DATETIME
status VARCHAR
Prediction
PK prediction_id INT
timestamp DATETIME
FK location_id INT
probability DECIMAL
FK event_type_id INT
FK model_id INT
FK severity_level_type_id INT
Event_type
PK event_type_id INT
event_name VARCHAR (100)
event_ description VARCHAR (200)
Weather_Station
PK station_id INT
name VARCHAR (100)
FK location_id INT
Model
PK model_id INT
model_name VARCHAR (100)
version DECIMAL
training_timestamp DATETIME
Severity_level
PK severity_level_id INT
severity_type VARCHAR (50)
level_description VARCHAR (200)
Data_source
PK source_id INT
name VARCHAR (100)
type VARCHAR (50)
url VARCHAR (100)
Weather_data_source
FK station_id INT
FK source_id INT
start_date DATETIME