

## Databases II

# AgroClima: A Smart Agro-Climatic Decision Support Platform

César Andrés Torres Bernal

20191020147

Juan David Duarte Ruiz

20191020159



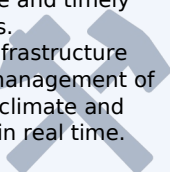

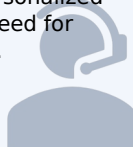




Universidad Distrital Francisco José de Caldas  
School of Engineering

## Introduction to the Business Model

*AgroClima* is a smart agro-climatic decision support platform designed to provide advanced climate-based solutions for farmers and agricultural organizations. The core idea of this project is to leverage climate data and artificial intelligence tools to deliver accurate and timely recommendations that enhance agricultural decision-making in the face of adverse weather conditions.

The main problem addressed by *AgroClima* is the high vulnerability of the agricultural sector to extreme weather events such as droughts, frosts, and heavy rainfall, which result in substantial crop losses, reduced productivity, and threats to food security. Currently, many farmers lack effective and accessible tools that would allow them to anticipate and adapt to these climate-related challenges.

The scope of the project includes the development of a final product: a user-friendly web platform specifically adapted for rural areas with limited internet connectivity. This platform will deliver personalized climate alerts, practical recommendations for agricultural management, and optimization strategies for resource use such as water and fertilizers. Furthermore, it will provide access to key regional climate risk information for public and private institutions. The overall goal of *AgroClima* is to empower small-scale farmers and relevant stakeholders in making informed strategic and operational decisions.

<h3>Key Partners</h3> <ul style="list-style-type: none"> <li>- Weather data providers (OpenWeather, Copernicus, IDEAM).</li> <li>- Manufacturers and distributors of IoT sensors for agricultural use.</li> <li>- Ministries of Agriculture, government agencies and NGOs that promote sustainable practices.</li> <li>- Universities and climate and agricultural research institutes, which provide knowledge and scientific validation.</li> <li>- Payment platforms and rural banking entities, which facilitate payments and subsidies.</li> <li>- Agricultural input distributors that could partner for integrated offers.</li> </ul> 	<h3>Key Activities</h3> <ul style="list-style-type: none"> <li>- Design and management of weather forecasting and advisory services, adapted to different agricultural profiles.</li> <li>- Execution of awareness and technology adoption campaigns in rural areas.</li> <li>- Management of strategic relationships with governments, cooperatives, NGOs and agricultural associations.</li> <li>- Technical support: development and maintenance of the digital system to offer the service.</li> </ul>  <h3>Key Resources</h3> <ul style="list-style-type: none"> <li>- Network of strategic alliances with actors in the agricultural and climate sector.</li> <li>- Access to reliable databases and satellite coverage.</li> <li>- Intelligent predictive analytics and machine learning system to generate accurate and timely recommendations.</li> <li>- Technological infrastructure for the efficient management of large volumes of climate and agricultural data in real time.</li> </ul> 	<h3>Value Propositions</h3> <p><u>For farmers:</u></p> <ul style="list-style-type: none"> <li>- Accurate and actionable recommendations based on local climate predictions.</li> <li>- Reduction of losses due to extreme events such as drought, frost or excessive rainfall.</li> <li>- Optimization of the use of resources such as water and fertilizers.</li> <li>- Ease of use, even without technical training.</li> </ul> <p><u>For institutions and companies:</u></p> <ul style="list-style-type: none"> <li>- Access to regional data and reports for agro-climatic decision making.</li> <li>- Tools for planning subsidies, agricultural insurance and technical support.</li> <li>- Improved public policies and agroclimatic disaster mitigation.</li> </ul> 	<h3>Customer Relationships</h3> <ul style="list-style-type: none"> <li>- Multichannel technical assistance (chat, telephone, mail) for rural users with limited connectivity.</li> <li>- Training and support in the adoption of the platform, especially in areas with low digitalization.</li> <li>- Automated and personalized alerts, without the need for constant interaction.</li> </ul>  <h3>Channels</h3> <ul style="list-style-type: none"> <li>- Web platform accessible in rural areas with low connectivity.</li> <li>- API access for institutions wishing to connect their agricultural management systems.</li> <li>- Institutional partners (agricultural agencies, cooperatives) as implementation and support channels.</li> </ul> 	<h3>Customer Segments</h3> <p><u>Individual users:</u></p> <ul style="list-style-type: none"> <li>- Small and medium farmers interested in protecting their crops.</li> <li>- Technified producers who integrate IoT in their agricultural management.</li> </ul> <p><u>Institutional customers:</u></p> <ul style="list-style-type: none"> <li>- Regional and national governments that manage climate risks in the agricultural sector.</li> <li>- Agribusiness companies seeking to optimize production and climate logistics.</li> </ul> 
<h3>Cost Structure</h3> <ul style="list-style-type: none"> <li>- Operation of alliances, training and customer service in the territory.</li> <li>- Access and processing of climate and satellite data.</li> <li>- Marketing, campaigns and community training.</li> <li>- Investment in continuous improvement of the regional recommendation and adaptation model.</li> <li>- Technical support: maintenance and technological operation costs servers, cloud, support.</li> </ul> 		<h3>Revenue Streams</h3> <ul style="list-style-type: none"> <li>- Monthly or annual subscription of farmers, with staggered plans according to needs.</li> <li>- Institutional licenses for local or national governments.</li> <li>- Additional services (consulting, premium data, risk maps).</li> <li>- Freemium model with free basic functionalities and advanced paid plans.</li> <li>- Alliances with insurance companies or input companies, which sponsor access to the service.</li> </ul> 		

# Requirements

## 1. Functional Requirements

ID	Requirement	Description	Priority
FR1	User registration	Allow farmers, technicians, and administrators to create an account using email or third-party login providers.	High
FR2	Authentication and access control	Provide secure login with differentiated access levels based on user roles.	High
FR3	Continuous data ingestion	Incorporate weather data in near real time from open or authorized sources, ensuring constant availability.	High
FR4	Distributed data storage	Store and access climate and agricultural data efficiently from any location with low latency.	High
FR5	Weather data queries	Enable users to consult and visualize current and historical weather data relevant to their agricultural zone.	High
FR6	Climate risk prediction	Offer reliable predictions about droughts, frosts, or heavy rainfall to support preventive decision-making.	High
FR7	Agricultural recommendations	Generate personalized recommendations for planting, irrigation, and fertilization based on local microclimates.	Medium
FR8	Report generation	Allow users to create customized reports for farmers and institutions to support strategic decisions.	Medium
FR9	Administrative panel	Provide administrative tools to manage users, roles, and global system statistics.	High

## 2. Non-Functional Requirements

ID	Requirement	Description	Priority
NFR1	Performance	The system must respond to queries and predictions within acceptable times for the expected data volumes.	High
NFR2	Horizontal scalability	Must support efficient scaling as the number of users or data ingestion volume increases.	High
NFR3	High availability	Must minimize downtime through automatic recovery mechanisms, ensuring operational continuity.	High
NFR4	Multi-region access	Must provide reasonable load times to geographically distributed users, regardless of physical location.	Medium
NFR5	Interoperability	Must integrate with external weather and agricultural data sources using standardized protocols.	Medium
NFR6	Usability	The interface should be intuitive, responsive, and usable on various devices, even with limited connectivity.	Medium
NFR7	Maintainability	The system should have a modular and well-documented architecture to support updates and scaling.	Medium

### 3. Prioritization Strategy — Applied Criteria

- **Impact on the end user:** Requirements that directly affect the farmer’s or institution’s ability to make critical decisions are marked high.
- **Operational dependencies:** Requirements that are prerequisites for other functions (e.g., authentication before access) are rated high.
- **Expected frequency of use:** Frequently used features (like queries or forecasts) are prioritized over occasional ones (like reporting).
- **Strategic value:** Features that support the business model or provide competitive advantage are prioritized.
- **Incremental benefit vs. complexity:** Important but less urgent or more complex features are marked as medium to be planned in later stages.

Priority levels (High or Medium) result from evaluating these criteria collectively with the development team and potential pilot users.

### 4. Performance and Capacity Analysis

#### System Dimensioning Assumptions

- Registered users: 2,000 in year one.
- Peak concurrent users: 500 (25%).
- Data sources: 1,200 sensors / 1 reading per minute  $\rightarrow$  1.7M records/day.
- Peak usage windows: 5:00–8:00 AM and 6:00–9:00 PM.
- Rural limitations: 2–5 Mbps connections, unstable latencies.

### Target Metrics

Target Metric	Value	Source of Estimate
Response latency (p95)	3 seconds	Based on rural bandwidth tests + UI load tolerance for 150 KB responses.
Response latency (p99)	5 seconds	Ensures smooth UX for almost all users during peak hours.
Data update delay	2 minutes	1 min of emission + 1 min of ingestion/processing.
Ingestion throughput	2,000 rec/s sustained (6,000 peak)	Derived from 1.7M daily recs $\times$ peak load factor $\times$ 3.
Availability	99.5% (3.6 h downtime/month)	Balanced cost-benefit for mid-tier infrastructure.
Payload size	150 KB	Suitable for 3G/4G connections to ensure $\leq$ 0.5s load.
Scalability	Double capacity in $\leq$ 1 hour	Tested via auto-provisioning in pilot environment.

### Source of Numbers

- Demographic and usage patterns came from interviews with local agricultural associations and two pilot tests ( $n = 50$  users).
- Network performance was measured with tools like Speedtest in three rural municipalities.
- Peak values include safety factors  $\times 3$ – $5$  over observed averages to cover climate emergency scenarios and system growth.
- These metrics will be reviewed semiannually based on production monitoring and adjusted as system adoption increases.

### Sizing Methodology

- **Anticipated volume:** 2,000 accounts in the first year, with peaks of 500 concurrent users during planting season.

- **Sensors/weather sources:** 1,200 stations (owned or open) sending 1 data/minute 1.7 million records/day.
- **Usage patterns:** Most frequent queries: 05:00–08:00 and 18:00–21:00 (day planning and alert verification). On average, a user executes 3 queries per session.
- **Limitations:** Rural connectivity: 2–5 Mbps links and unstable latencies.

## User Stories

This section outlines the user stories that guide the functional development of the AgroClima Climate Risk Prediction System. User stories are structured descriptions of system features from the perspective of end users, helping the development team stay aligned with user needs. To ensure consistency, clarity, and prioritization in planning and execution, this section also includes:

- **Priority Rubric:** A framework for classifying stories as High, Medium, or Low priority based on business impact, user value, and technical dependencies.
- **Estimation Metrics:** A story point-based system used to estimate the relative effort required for implementing each user story.

Each user story is presented in a standardized format and includes clearly defined acceptance criteria to support testing and validation. The stories are grouped by user role (Client, Analyst, Administrator) to reflect the different system interactions.

## Priority Rubrics

The priority rubric defines how each user story is categorized based on its importance to the system's core functionality, user impact, and time sensitivity. This classification helps the team focus on delivering the most critical features first, ensuring that essential needs are met in early development phases. It should be noted that the priority levels assigned to each user story are based on the defined scope of the project. This does not imply that some stories are inherently more important than others, but rather that certain functionalities are more critical to meet the objectives of the current delivery stage of the system.

Priority Level	Description	Typical Characteristics
High	Essential for system functionality or user satisfaction. Must be delivered in the MVP (Minimum Viable Product).	<ul style="list-style-type: none"><li>- Core business functionality</li><li>- Direct impact on user safety or decision-making</li><li>- Regulatory or compliance requirement</li></ul>
Medium	Important but not critical. Enhances usability or performance. Can be delivered after core features.	<ul style="list-style-type: none"><li>- Improves user experience</li><li>- Optimizes workflows or data processing</li><li>- Moderate business value</li></ul>
Low	Non-essential. Can be deferred without major consequences.	<ul style="list-style-type: none"><li>- Low usage frequency</li><li>- Non-mandatory functionality</li></ul>

Figure 1: Priority Rubrics

## Estimation Rubrics

Estimation metrics provide to evaluate the relative effort required to implement each user story. These metrics will help the team to plan sprints more accurately, balance workloads, and assess development complexity without relying on exact time measurements. Points are assigned based on factors such as technical complexity, risk, dependencies, and required resources.

Story Points	Description	Characteristics
1	Very simple task	- No dependencies or logic - Easy to test and deploy
2	Simple feature with minimal logic	- Few input validations - Slight data manipulation - Low technical risk
3	Moderate complexity	- Requires backend/frontend interaction - Conditional logic - Complex read/write transactions with database
4	High complexity	- Multiple components involved - Integration with APIs - High performance in the database

Figure 2: Estimation metrics

## User Stories

Based on the priority rubrics and estimation metrics defined earlier, we developed a set of user stories for the AgroClima system to identify and address the primary needs of its users. The user stories are organized by user roles within the system: the Client, typically a farmer or crop owner; the Analyst, who processes, analyzes, and visualizes environmental data; and the Administrator, who manages user access and ensures proper and secure system operation.

### User Stories (Client)

<b>Title:</b> Register as a Client	<b>Priority:</b> High	<b>Estimate:</b> 3
<b>User Story:</b> As a client, I want to register using my email account so that I can access climate data and receive tailored recommendations.		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"> <li>• The system allows registration via email, Google, or Microsoft.</li> <li>• Client roles are assigned automatically upon successful registration.</li> <li>• A confirmation email is sent upon account creation.</li> </ul>		

Figure 3: User Story 1



<b>Title:</b> View Local Weather Conditions	<b>Priority:</b> Medium	<b>Estimate:</b> 4
<b>User Story:</b> <p>As a client, I want to view real-time and historical weather data for my region so that I can plan activities effectively.</p>		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"> <li>• Weather data is displayed for the client's location.</li> <li>• Data includes temperature, humidity, precipitation, and forecast.</li> <li>• A time filter (e.g., last 7 days, 1 month) is available.</li> </ul>		

Figure 4: User Story 2

<b>Title:</b> Receive Weather Alerts	<b>Priority:</b> Medium	<b>Estimate:</b> 4
<b>User Story:</b> <p>As a client, I want to receive automatic alerts for upcoming extreme weather events in my area so that I can take preventive actions.</p>		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"> <li>• Alerts are triggered based on predictive models.</li> <li>• Alerts are displayed in the user dashboard.</li> <li>• Each alert includes event type, severity and estimated time.</li> </ul>		

Figure 5: User Story 3

<b>Title:</b> Access Historical Recommendations	<b>Priority:</b> High	<b>Estimate:</b> 3
<b>User Story:</b>  As a client, I want to view past recommendations and actions taken so I can evaluate their effectiveness over time.		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"> <li>• A history tab shows all past recommendations received.</li> <li>• Each entry includes date, context, and whether the advice was followed.</li> <li>• The user can filter history by date range or type.</li> </ul>		

Figure 6: User Story 4

<b>Title:</b> Receive Farming Recommendations	<b>Priority:</b> Low	<b>Estimate:</b> 4
<b>User Story:</b>  As a client, I want to receive personalized recommendations based on weather predictions and soil conditions so I can optimize my crop yield.		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"> <li>• Recommendations are shown on the dashboard.</li> <li>• Data includes suggestions on irrigation, fertilization, and planting.</li> <li>• Alerts are sent for critical weather risks (e.g., frost, drought).</li> </ul>		

Figure 7: User Story 5

## User Stories (Analyst)

Title:	Priority:	Estimate:
Access and Analyze Climate Data	Medium	4
<b>User Story:</b>  As an analyst, I want to access real-time and historical climate data from all monitored stations so I can analyze environmental trends.		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"><li>• A dashboard shows station data in real time.</li><li>• Filters allow data segmentation by location and date.</li></ul>		

Figure 8: User Story 6

Title:	Priority:	Estimate:
Generate Analytical Reports	Medium	3
<b>User Story:</b>  As an analyst, I want to generate reports about climate conditions and risk trends to support decision-making for stakeholders.		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"><li>• Reports can be generated monthly or weekly.</li><li>• Data visualizations (charts, graphs) are included.</li></ul>		

Figure 9: User Story 7

## User Stories (Administrator)

Title:	Priority:	Estimate:
Manage Users and Roles	High	2
<b>User Story:</b>  As an administrator, I want to manage user accounts and assign roles so that access control is enforced throughout the platform.		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"><li>• Admin dashboard lists all users with filters by role.</li><li>• Roles can be assigned or changed.</li><li>• User accounts can be deactivated or deleted.</li></ul>		

Figure 10: User Story 8

<b>Title:</b> Audit User Activity Logs	<b>Priority:</b> Low	<b>Estimate:</b> 4
<b>User Story:</b>  As an administrator, I want to access logs of user activities so I can ensure compliance and investigate anomalies.		
<b>Acceptance Criteria:</b> <ul style="list-style-type: none"> <li>• Logs include login times, data access, configuration changes.</li> <li>• Filters by user, date, and activity type are available.</li> <li>• Logs can be exported in CSV format.</li> </ul>		

Figure 11: User Story 9

# Data System Architecture

The AgroClima 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

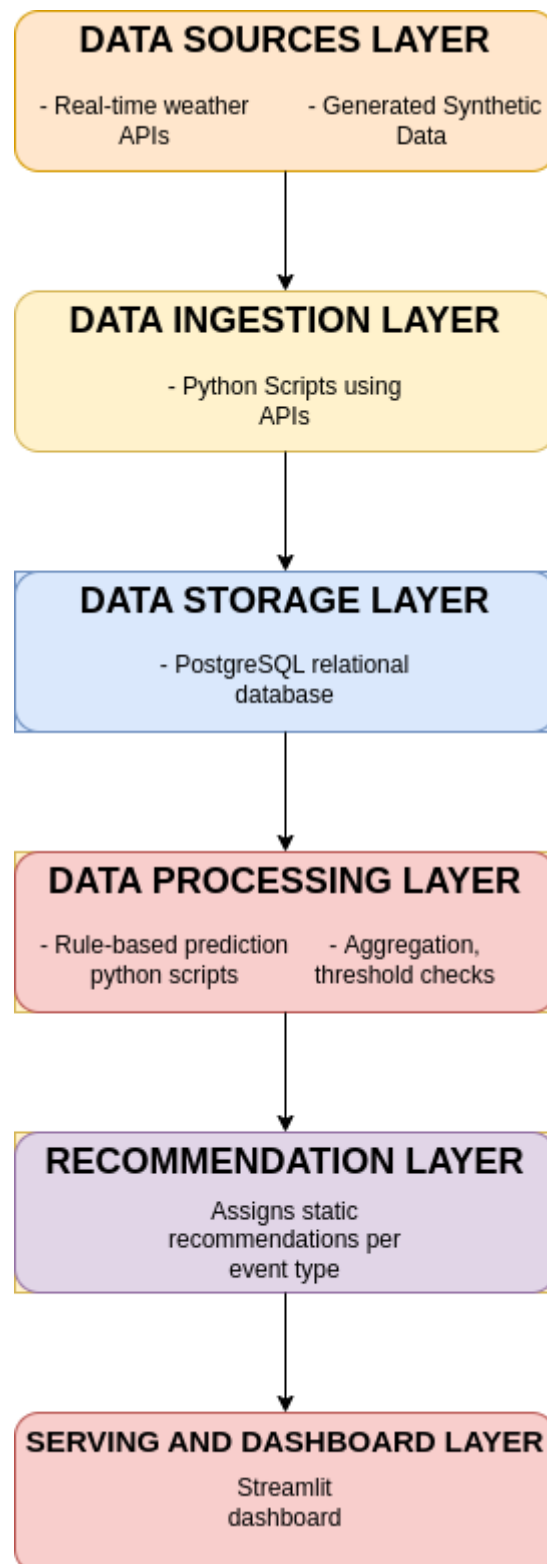


Figure 1: High-level Architecture Diagram

## 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` – 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:**

- **Ingestion Scheduler:**Manages the frequency and timing of data acquisition. Can use tools like `cron` or `APScheduler` (Python) (e.g., Railway jobs).
- **Data Fetchers:** Python-based scripts or microservices that call external APIs (`OpenWeather`) using libraries like `requests`. These fetchers retrieve JSON data, parse it, and transform it into internal representations.
- **Insertion Handler:** Performs database inserts using `psycopg2`, managing transactions and constraints. This component may include logic to prevent duplication using `ON CONFLICT DO NOTHING` or upsert patterns.

**Interaction:** The Data Ingestion Layer receives weather and agricultural data from the Data Sources Layer via scheduled API calls or synthetic generators. After validating and formatting the data, it writes structured records to PostgreSQL, primarily into tables such as *Location*, *Weather\_Station*, and *Weather\_Data*, preserving referential integrity and preparing the data for further processing.

## 3. Data Storage Layer

**Role:** This layer stores both raw and refined data, allowing persistence and long-term access.

**Components & Technologies:**

- **PostgreSQL:** Stores structured relational data: users, crops, alerts, predictions, sessions. Chosen for its `ACID` compliance, query performance, and relational integrity.

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

## 4. Data Processing Layer

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

**Components & Technologies:**

**Interaction:** Reads input data from `PostgreSQL` runs models and transformations, then writes back results to `PostgreSQL` (structured outputs like predictions).

## 5. Recommendation Layer

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

**Components & Technologies:**

- Python Scripts to generate generic recommendations based on predictions and location.

**Interaction:** Static mapping of events to field-specific actions.

## 6. Serving and Dashboard Layer

**Role:** Serves processed data to external applications and end-users through interfaces and APIs.

**Components & Technologies:**

- **Streamlit** (Python): **Streamlit** expose data from PostgreSQL.
- **Backend Services:** Handle user session management, request routing, access control, and processing API calls.

**Interaction:** Streamlit is in charge to show data as Predictions and Recommendations into a visual table for user.



# 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):**

---

```
1 SELECT user_id, user_name, role, email
2 FROM User
3 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):**

---

```
1 SELECT p.prediction_id, e.event_name, l.name AS location, p.probability, s.severity_type, p.timestamp
2 FROM Prediction p
3 JOIN Event_type e ON p.event_type_id = e.event_type_id
4 JOIN Severity_level s ON p.severity_level_type_id = s.severity_level_id
5 JOIN Location l ON p.location_id = l.location_id
6 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):**

---

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

```

6 JOIN User u ON af.user_id = u.user_id
7 WHERE u.user_id = 1
8 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):**

```

1 SELECT wd.timestamp, wd.temperature, wd.humidity, ws.station_name, l.name AS location
2 FROM Weather_Data wd
3 JOIN Weather_Station ws ON wd.station_id = ws.station_id
4 JOIN Location l ON ws.location_id = l.location_id
5 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):**

```

1 SELECT af.field_id, af.field_size, l.name AS location, c.crop_name, u.user_name
2 FROM Agriculture_Field af
3 JOIN Crop c ON af.crop_id = c.crop_id
4 JOIN Location l ON af.location_id = l.location_id
5 JOIN User u ON af.user_id = u.user_id
6 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):**

---

```
1 SELECT u.user_name, e.event_name, a.status, a.sent_at
2 FROM Alert a
3 JOIN User u ON a.user_id = u.user_id
4 JOIN Prediction p ON a.prediction_id = p.prediction_id
5 JOIN Event_type e ON p.event_type_id = e.event_type_id
6 WHERE u.user_id = 1
7 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):**

---

```
1 db.interaction_logs.find({
2   interaction_type: "recommendation_viewed",
3   user_id: "u001"
4 }).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.

# Concurrency Analysis

In data-intensive and distributed systems such as AgroClima, concurrency control is a fundamental aspect of system integrity, availability, and performance. Given that multiple components operate in parallel, including real-time data ingestion, predictive modeling, alert generation, and user interaction, there are several critical scenarios where concurrent access to shared data resources occurs. If not properly managed, such concurrency can lead to inconsistencies, data corruption, race conditions, or system bottlenecks.

## Concurrency Scenarios in AgroClima Prediction System

### a. Real-Time Data Ingestion and User Querying

While the ingestion layer pulls real-time weather data from external APIs or synthetic generators and writes to *Weather.Data* in PostgreSQL or MongoDB, users and services may simultaneously read from these collections to view live conditions, generate alerts, or run predictive models. This read-write concurrency requires careful isolation to avoid reading uncommitted or inconsistent data.

### b. Simultaneous User Activity Logging

Multiple users interacting with the platform generate high-frequency concurrent writes to collections such as *UserActivityLogs* or *UserPreferences*. As these are time-sensitive and may vary in structure, conflicts or race conditions may emerge if operations are not atomic.

### c. Prediction Generation vs. Alert Dispatch

Machine learning models running in batch mode (via Apache Spark or internal scripts) periodically generate new *ClimatePredictions*. In parallel, the alerting service consumes these predictions to trigger *RiskAlerts* to end-users. If both services access or update overlapping records without transactional guarantees, there is a risk of duplicate alerts, missed triggers, or inconsistent alert content.

### d. Dashboard Analytics During Data Updates

Business Intelligence modules may execute heavy analytical queries on datasets like *HistoricalWeatherData* or *ClimatePredictions* for visualization or reporting. At the same time, backend processes might update these same records, leading to inconsistent visualizations or slow query performance if locking is not managed.

## Potential Concurrency Problems and Solutions

AgroClima operates in a distributed and partially parallel data environment, where data is continuously ingested, transformed, and queried across multiple layers. These operations occur in relational (PostgreSQL) and non-relational (MongoDB) databases, some of which may be replicated or partitioned across nodes. While this enables performance and scalability, it also introduces several complex concurrency challenges that must be addressed at both the architectural and transactional levels.

## Race Conditions

Race conditions occur when two or more processes access shared data concurrently and at least one process modifies the data, causing unpredictable results depending on the execution sequence.

A Spark job is updating the *ClimatePredictions* table while the recommendation engine is querying the same record to generate an advisory. If the update partially completes before the read, the user may receive incorrect recommendations based on stale data.

## Mitigation Strategies

- Use row-level locking in PostgreSQL when predictions are being updated and joined with recommendations.
- Apply synchronization logic in application services to restrict simultaneous write access when needed.

## Lost Updates

Occurs when two concurrent updates to the same record overwrite each other without being aware of the other's changes, resulting in data loss.

Two backend processes update the same *UserPreferences* document within milliseconds of each other, causing one update to be unintentionally discarded.

### Mitigation Strategies

- Implement Optimistic Concurrency Control (OCC) by using version numbers or timestamps in both PostgreSQL and MongoDB.

## Deadlocks

A deadlock arises when multiple transactions each hold a lock on a resource and attempt to acquire a lock on the other's resource, leading to a cycle of waiting that can never resolve naturally.

A recommendation service locks the *Prediction* table and waits for *Alert*, while the alerting engine does the reverse — causing both to halt.

### Mitigation Strategies

- Keep transactions short and narrow in scope, locking only the required rows or documents.
- Rely on database-level deadlock detection: PostgreSQL, for instance, can automatically abort one transaction to break the cycle.

## Stale Reads

A stale read occurs when a query retrieves outdated or incomplete data due to asynchronous replication delays or transaction visibility rules.

A BI tool queries *HistoricalWeatherData* from a PostgreSQL read replica while the primary node is still ingesting and committing recent updates. The report reflects incomplete data, potentially misleading decisions.

### Mitigation Strategies

- In MongoDB, apply read concern "majority" to ensure that only fully replicated and acknowledged writes are visible to the reader.
- BI dashboards should query from synchronized nodes or use materialized views that update periodically but remain internally consistent.

## Replica Inconsistency and Latency

In replicated database architectures, write operations typically go to the primary node, while reads can be served by secondaries. If replication is asynchronous, delays can cause inconsistencies between nodes. An administrator views alert history from a MongoDB secondary that hasn't yet received recent writes from the primary, resulting in missing or outdated records.

### Mitigation Strategies

- Use Redis caching only for data with tolerable staleness and implement time-to-live (TTL) mechanisms.
- For PostgreSQL, query the primary node when absolute consistency is required.

## Partitioning and Sharding Conflicts

In systems that partition data horizontally across nodes (e.g., by region or time), queries or writes that span multiple partitions can become bottlenecks or require distributed coordination.

A prediction model processes weather data across regions that are stored in separate shards, causing inter-partition joins or requiring distributed transactions.

### **Mitigation Strategies**

- Use partition-aware queries that minimize cross-shard operations.
- Apply distributed transaction coordination only when required, and favor eventual consistency for non-critical updates.

# Performance Improvement Strategies

To meet the architectural demands and functional requirements of the AgroClima platform—namely, fast query execution, constant data ingestion, support for BI modules, distributed access, and real-time recommendations—several performance improvement strategies grounded in parallelism and distribution have been implemented. These strategies are necessary to ensure system responsiveness, fault tolerance, and scalability in a high-volume, big data context.

## Horizontal Scaling

Horizontal scaling increases the platform’s processing capacity by distributing workloads across multiple instances of services, rather than vertically upgrading a single machine. This is particularly important in cloud-native architectures that must respond dynamically to varying user demand and ingestion volumes.

### Application in AgroClima System

- **FastAPI microservices** are deployed across multiple containers or virtual machines behind a load balancer to handle concurrent API requests from farmers, analysts, and administrators.
- **PostgreSQL read replicas** are used to offload read-heavy workloads (e.g., BI dashboards and user queries) from the primary transactional database.
- **MongoDB replica sets** support distributed and geo-aware read operations for unstructured data like raw API payloads and activity logs.

### Benefits:

- Enables high availability, load balancing, and disaster recovery.
- Supports low-latency access to services from multiple geographical regions.

### Potential Challenges:

- Requires sophisticated orchestration and monitoring tools such as Kubernetes.
- Must externalize session state and user data using caching layers like Redis to maintain consistency across instances.

## Data Partitioning and Sharding

Data partitioning divides large datasets into smaller, independent partitions based on keys such as location, time range, or user ID. This allows each partition to be processed, queried, or stored in parallel, improving performance and scalability.

### Application in AgroClima System

- PostgreSQL partitioned tables are used for time-series data such as *Weather\_Data*, where each partition corresponds to a day, week, or region.

### Benefits:

- Speeds up query execution by reducing the amount of data scanned.
- Enables localized computation and model execution (e.g., per region or farm zone).

### Potential Challenges:

- Poorly chosen partition keys may lead to data skew or hotspots.
- Cross-partition queries become more complex and may require distributed joins or data federation strategies.

## Replication and Caching

Replication and caching are complementary strategies used to improve read performance and ensure availability. Replication involves maintaining multiple synchronized copies of data, while caching stores frequently accessed data in memory.

### Application in AgroClima System

- PostgreSQL read replicas and MongoDB secondaries are used to serve non-critical reads (e.g., dashboard metrics, mobile queries), reducing latency and database contention.
- Redis is implemented to cache:
  - Most recent *ClimatePredictions* per region.
  - Frequently accessed recommendations.
  - User profile and session data for quick retrieval.

#### Benefits:

- Drastically reduces load on primary databases.
- Provides real-time responsiveness for critical user-facing operations.

#### Potential Challenges:

- Requires careful cache invalidation strategies to prevent stale data delivery.
- Replication lag can affect consistency in systems with high write throughput.
- Configuration and monitoring overhead increase with data redundancy and distributed cache layers.

## Parallel Query Execution

Parallel query execution is a technique in which a single SQL or analytical query is divided into multiple smaller tasks that are executed simultaneously across different processor cores or worker threads.

### Application in AgroClima System

- AgroClima processes large volumes of structured and semi-structured data, particularly for:
  - Historical weather analysis
  - Batch predictions over multiple regions
  - Dashboards and risk summaries across all users or farms
- PostgreSQL supports parallel sequential scans, parallel joins, and parallel aggregates. When enabled, PostgreSQL divides a single query into chunks processed by multiple worker processes concurrently. This is valuable for:
  - Aggregating *ClimatePredictions* per region or per crop.
  - Computing rolling statistics on *Weather\_Data*.
- Dashboards and reports are powered by parallel-executed SQL queries on materialized views or aggregates, ensuring near-instantaneous insights for managers.

#### Benefits:

- Reduced Query Time: Especially for analytical queries that involve millions of records.
- Improved User Experience: Dashboards and reports respond faster, even during peak usage.
- Supports Big Data Use Cases: Enables scalable processing of historical climate datasets or user interactions.



**Potential Challenges:**

- Higher Memory Use: Parallel queries consume more memory, especially when joining large datasets.
- Configuration Required: PostgreSQL parallelism must be enabled and tuned.
- Not All Queries Can Be Parallelized: Very simple queries or those with functions that are not parallel-safe will still execute sequentially.
- Coordination Overhead: In distributed systems like Spark, synchronization between tasks introduces some latency.

These strategies ensure that AgroClima maintains optimal performance even under conditions of high user load, intensive data processing, and geographically distributed access. By combining horizontal scalability, intelligent data partitioning, and efficient read optimization mechanisms like replication and caching, the platform remains responsive, resilient, and capable of supporting both real-time and historical climate intelligence services.