Project Proposal Submission Form

Course: Distributed Systems
Instructor: Palak Mejpara
Semester: 25 Fall
Group ID / Team Name: Group 7
Submission Date: Oct / 28th / 2025

---

0. GitHub Repository

- **GitHub URL of project repository:** https://github.com/Yuvi-5/Distributed-Weather-App
  (e.g., `https://github.com/your-team/yourproject`)
- Ensure the repository is **public** (or shared with instructor) and will be used for all code, documentation, issues, and sprint tracking.

---

1. Team Information

	Member Name	Student ID	Role (including Scrum Master)	Feature Owned
1.	Dhruv Kalra	5143420	Backend Developer	Data Stream Handler
2.	Janaranjanee Sendhil Kumar		Backend Developer	Alert & Broadcast Module
3.	Dipal Nirmal		Frontend Developer	Visualization Dashboard
4.	Minhajuddin Muhammad	5144183	Backend Developer	Aggregator & Forecast Engine
5.	Yuvraj Singh Palh	5143317	Scrum Master	Edge Node Data Service

Note: Number of group members = number of major features

---

2. Project Title & Short Description

- **Project Title:**

  Distributed Weather Data Aggregator

- **Short Description (2-3 sentences):**

A distributed, event-driven system that collects and broadcasts live weather updates from multiple nodes and predicts weather trends for the next seven days. Each node gathers real-time readings (temperature, humidity, and wind) and streams data to a central aggregator. The aggregator processes the data, detects anomalies, and uses an external API for 7-day forecasting. The dashboard visualizes all readings, alerts, and predictions in real time.

3. Features Summary

List all major features you plan to implement (excluding login/signup). Each feature corresponds to one team member.

Feature #	Feature Name	Brief Description	Assigned Member
1	Edge Node Data Service	Simulates or retrieves real-time local weather data (temperature, humidity, wind speed) and sends it to the broker.	
2	Data Stream Handler	Ensures reliable message delivery, manages buffering, and handles retries when nodes or network fail.	
3	Aggregator & Forecast Engine	Collects data from all nodes, computes regional statistics, detects anomalies, and integrates 7-day forecast from OpenWeatherMap API.	
4	Alert & Broadcast Module	Generates regional weather alerts and broadcasts live weather data streams to all connected dashboards.	
5	Visualization Dashboard	Real-time UI showing live readings, trends, node health, alerts, and 7-day forecast charts.	

4. Architecture Overview

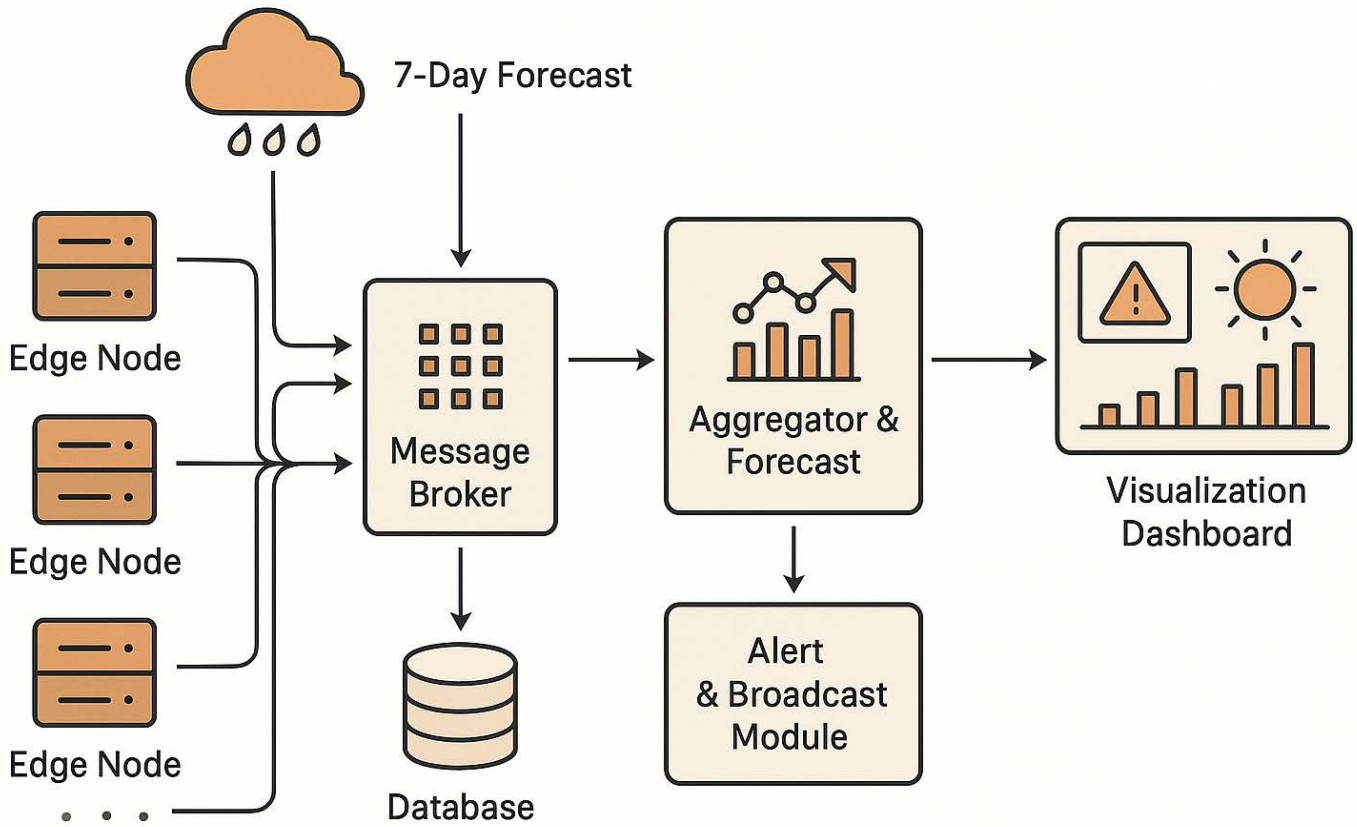
- **Proposed System Architecture:**
  (E.g., microservices, event-driven, peer-to-peer, using message queue, etc.)
- **Attach Architecture Diagram:**
  The diagram must show:
  - Components/services/nodes
  - How they communicate (APIs, queues, RPC, etc)
  - Data flows, storage layers (database(s), cache(s))
  - Any external systems or integrations
- **Reasoning / Justification for the Architecture:**
  Explain why you chose this architecture in terms of distributed-systems concerns: scalability, fault-tolerance, consistency, partitioning, latency, etc.

Proposed System Architecture:
A microservices-based event-driven architecture using a publish–subscribe model.

Edge Nodes: Produce local weather data and publish to Kafka/RabbitMQ.
Aggregator Service: Consumes data, processes statistics, stores them in MongoDB, and integrates 7-day forecast data from the OpenWeatherMap API.
Alert & Broadcast Module: Detects anomalies and pushes updates to WebSocket endpoints.
Dashboard: Subscribes to the broadcast for live display of metrics and forecasts.

Reasoning / Justification:

Scalability: Each node runs independently and new ones can join seamlessly.
Fault Tolerance: Queue-based communication ensures no data loss during downtime.
Consistency: Eventual consistency maintained across nodes and central aggregator.
Low Latency: WebSockets and message queues ensure fast live updates.
Partition Tolerance: Nodes can temporarily operate offline and resync later


7-Day Forecast
Edge Node
Edge Node
Edge Node
Message Broker
Database
Aggregator & Forecast
Alert & Broadcast Module
Visualization Dashboard

5. Agile Methodology & Team Roles

- **Scrum Master:** Yuvraj
- **Other team roles (e.g., Developer, QA, DevOps):**
  -
  - Mentioned above
- **Sprint Structure:**
  - Sprint length: 1 weeks
  - Number of sprints: 3
  - Sprint deliverables (example): Sprint 1 → Node setup + data stream
    Sprint 2 → Aggregator + forecast integration
    Sprint 3 → Dashboard + alert broadcasting
- **Tools for tracking & collaboration:**
  (e.g., Trello, Jira, GitHub Projects, Slack, etc)

  GitHub Projects (Kanban), Slack (communication), Google Meet (stand-ups)

- **Stand-up Cadence:**
  (e.g., daily at ___ time, 2 times per week)

  Twice weekly

- **If your group has only two members**, tick here: □ *We will communicate separately with instructor prior to starting*

6. Timeline & Milestones

Milestone	Description	Target Date
Proposal submission & approval	Submit this document and get instructor approval	/ / Oct 28
Sprint 1 end		Nov / 7 / 2025
Sprint 2 end		Nov / 14 / 2025
Mid-semester demo/update		Nov / 21 / 2025
Final project submission	Working distributed system + report + demo	End of/Sem /

7. Risk Assessment & Mitigation

- **Risks:**
  1. Forecast API limitations or downtime.
  2. Data loss due to network or broker failure.
  3. Latency increase during concurrent updates.
- **Mitigations / Contingency Plans:**
  1. Cache API results and simulate fallback data.
  2. Use persistent message queues with retry mechanisms.
  3. Implement throttling and batching for heavy data flow.

---

8. Dependencies & External Tools

- List any **external libraries / frameworks / services / APIs** that your project depends on:

  Frameworks: FastAPI / Node.js, Kafka or RabbitMQ, MongoDB / InfluxDB, React.
  External APIs: OpenWeatherMap (for live and forecast data).

- Are there any **hardware or infrastructure** dependencies? (e.g., servers, cloud, nodes, IoT)

  Docker Compose for distributed deployment.

- Are there any **known constraints** (time zones, availability, data privacy, third-party limits)?

  Free API limits, local machine resource allocation.

---

9. Evaluation Metrics & Success Criteria

- **How will you evaluate your system's success?** (e.g., throughput, latency, fault recovery, number of nodes supported, user operations per second, etc.)

  End-to-end latency (node → dashboard)
  Throughput (messages/sec)
  Forecast accuracy (%)
  Alert latency and reliability
  Node fault recovery time (s)

- **What are your target metrics/goals?**

  Latency < 300 ms
  Forecast accuracy ≥ 85%
  0% message loss with ≥ 5 active nodes
  Fault recovery within 5 seconds

- **How will you demonstrate/measure these?**

  Use system logs, real-time dashboard metrics, and stress testing under simulated failures.

10. Approval Section (Instructor)

- Instructor Comments:

- Approved: □ Yes □ No
- Date: ____ / ____ / ____
- Signature: _____

Instructions to students:

1. Fill in all sections above completely and accurately.
2. Provide the architecture diagram as required in Section 4.
3. Submit this form **before you begin implementation**.
4. Do **not** start coding, designing in detail or building the system until you receive approval.