Title: Resolving the Key Flaws in the SmartFarm 360 Architecture

---

⚠️ Flaw 1: No Load Balancer

Problem: All traffic is handled by a single backend server. If this server crashes or becomes slow, everything breaks.

Why it's bad:

- Not fault-tolerant (no backup)
- Not scalable for many users

Solution:

- Add a **load balancer** (like NGINX, AWS ELB, or Render Auto-scaling)
- Split traffic among multiple backend instances to handle load efficiently

Result:

- Better uptime and performance

⚠️ Flaw 2: ML Training and Prediction Happen on the Same Engine

Problem: Your ML training and live predictions share the same code or server.

Why it's bad:

- Training uses heavy computation(CPU usage) and can slow down real-time predictions

Solution:

- Separate into two parts:
  - train_model.py for periodic training (offline)
  - predict_model.py (runs 24/7 for user input) in server

Result:

- Stable and fast predictions, even while retraining happens in the background

⚠️ Flaw 3: No Caching Layer (Everything Hits MongoDB)

Problem: Every dashboard load pulls data directly from MongoDB, even if it's already been fetched recently.

Why it's bad:

- Slower app speed
- More load on MongoDB

Solution:

- Use **Redis** to store frequent or repeated data (e.g., past recommendations)
- Use Redis as the first check; fallback to MongoDB if Redis misses

Result:

- Faster dashboard performance and reduced database load

⚠️ Flaw 4: No Drone Task Validation or Security

Problem: Your backend sends spray commands without verifying if they were executed. Also, there's no security — anyone could try to send a command.

Why it's bad:

- Duplicate tasks or missed execution
- Risk of unauthorized drone control

Solution:

- Secure APIs with **JWT authentication**
- Drones must send back an acknowledgment (like "Yes,task completed successfully") when they complete a task
- Use **MQTT** for secure and reliable task delivery (optional)

Result:

- Reliable and secure drone task handling

⚠️ Flaw 5: No HTTPS and Weak Login Security

Problem: Data is being transferred over HTTP. Also, no protection for APIs — anyone can call them.

Why it's bad:

- Hackers can steal data
- No access control

Solution:

- Use **HTTPS** for all requests
- Use **JWT tokens** for login sessions
- Secure each API endpoint by checking the user's role

Result:

- Strong data protection and user-level access control

⚠️ Flaw 6: No Logs or Health Monitoring

Problem: If something goes wrong (backend crashes, ML fails), you won't know. There are no logs or health checks.

Why it's bad:

- Hard to debug
- No way to detect downtime

Solution:

- Add a /status route to every service
- Log all critical events (login, tasks, prediction, errors)
- Use tools like **UptimeRobot**, **Loggly**, or **Grafana** (optional)

Result:

- System is observable and easier to maintain

⚠️ Flaw 7: No Role-Based Dashboard Access

Problem: All users see the same dashboard regardless of whether they're an admin, drone operator, or farmer.

Why it's bad:

- Security risks
- Confusing UX (wrong features shown to wrong people)

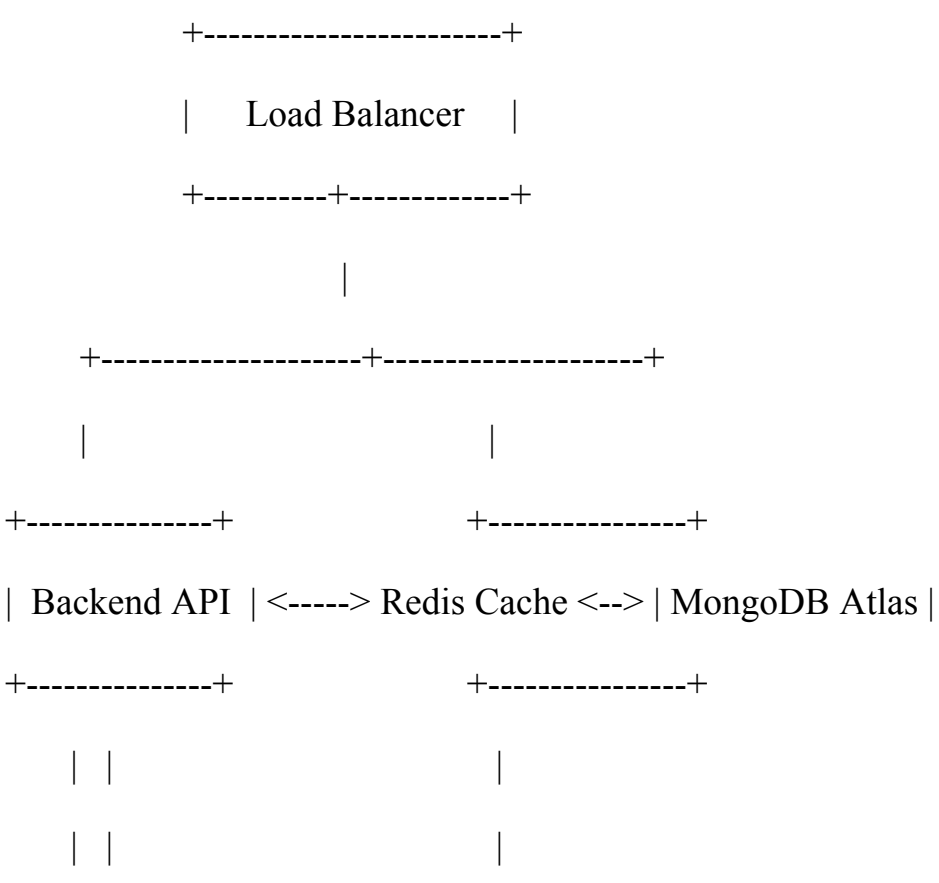
Solution:

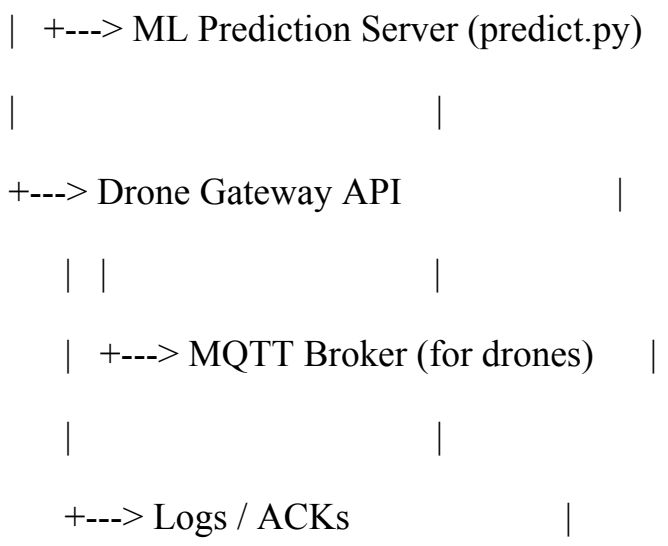
- Store user role during login (in JWT token)
- On frontend, show different components/pages based on role,means different dashboard for users,admin and drone controller
- Backend checks role before doing sensitive operations

Result:

- Clean, secure user experience with proper role access

TRY TO UNDERSTAND THIS





Frontend: React App (Role-based Dashboard)

- Farmer view
- Drone operator view
- Admin view

Monitoring: UptimeRobot + Log tracking service