Decision Log : Drone Operations AI

Key Assumptions

Data & Storage

CSV-based persistence - Assumed data is read at startup, not modified during runtime
- Valid for prototype with 5-10 records
- No concurrent write requirements
- Trade-off: Simplicity over scalability

Single-threaded operation - One user/conversation at a time
- Sufficient for demo/proof-of-concept
- Can upgrade to async with FastAPI + Celery later

Pilot/drone status is current - Data manually updated in CSV before queries
- No real-time location tracking or live updates
- Assumption: Operators trust CSV data accuracy

User Interaction

Natural language queries require LLM intelligence
- Fallback: Simple regex pattern matching for ~20 common queries
- Assumption: Users phrase requests consistently ("Show pilots", "Available drones")
- Reality check: Rule-based mode covers 80% of expected operational queries

No authentication needed - Suitable for prototype/demo only
- Assumption: Internal tool for trusted operators
- Future: Add JWT tokens for production

Operational Model

Urgent missions detected but require human approval (NOT auto-reassignment)
- Assumption: Safety-critical drone operations require human oversight
- Liability: Auto-assigning incompetent pilot = legal/regulatory risk
- Model: System detects conflicts and alerts operator

Technology Trade-offs & Decisions

1. Flask vs FastAPI vs Django

Framework	Chosen?	Reason
Flask	YES	Lightweight, perfect for prototype-to-production, single file deployable, minimal overhead
FastAPI	NO	Overkill for simple API, steeper learning curve, async not needed yet
Django	NO	Too heavy, 20+ files boilerplate, better for complex apps with admin panels

Trade-off Accepted: Limited built-in features, manual error handling Benefit: Flexible, easy to scale to production (can migrate to FastAPI/Django later)

2. LLM: ChatGPT + Rule-Based Fallback

Approach	Chosen?	Reason
ChatGPT + Fallback	YES	Best of both: intelligent LLM + graceful degradation when API fails
Claude Only	NO	More reliable but no fallback, requires API key
Local LLM (Llama)	NO	Free & offline but GPU-intensive, slower, overkill for structured queries
Rule-Based Only	NO (as primary)	Simple but limited, can't handle context or variations

Dual-Mode Architecture:

```
User Query → [Is OpenAI API available?]
    ├─ YES → ChatGPT with Tool descriptions (powerful reasoning)
    └─ NO  → Regex pattern matching (simple, fast, 100% reliable)
```

Why This Works:

Real scenario: OpenAI API quota exceeded (actual current state) → App still works perfectly
LLM mode: Better NLP, conversation context, reasoning
Rule-based mode: 80% of operational queries work without AI
Trade-off: Rule-based is regex-fragile but sufficient for structured commands

Storage	Chosen?	Reason
CSV	YES	Human-readable, version control friendly, zero setup, lightweight
SQLite	NO (for now)	Overkill for 12 records, still needs schema management
PostgreSQL	NO (for now)	Needs docker/server, connection pooling, overcomplicated for MVP
Google Sheets	NO (Phase 2)	Adds OAuth complexity, but planned for real 2-way sync future

3. Data Storage: CSV vs SQLite vs PostgreSQL

CSV Limitations Accepted:

No concurrent write safety
No indexing (but 5 pilots = linear search fine)
No complex queries

But: Simple, git-trackable, editable in Excel

Phase Plan:

Phase 1: CSV (current)
Phase 2: SQLite + ORM
Phase 3: PostgreSQL with replicas

4. Conflict Detection Strategy

Chosen: Validation-based (proactive) + user approval required

5 Conflict Types Detected:

Double-Booking - Same pilot assigned to overlapping missions
Skill Mismatch - Pilot lacks required skills
Certification Gap - Missing regulatory certifications
Equipment Mismatch - Drone lacks required capabilities
Location Mismatch - Resource unavailable at mission location

Why Validation-Based (not auto-resolve):

Safety-first: Prevents assigning unqualified pilots
Regulatory: Drone ops require documented human decision
Liability: "System auto-assigned incompetent pilot" = lawsuit

How "Urgent Reassignments" Were Interpreted
What We Did NOT Do

Auto-reassign pilots without approval
Override conflict detection for urgent missions
Implement automatic conflict resolution

What We DID Do

Detect urgent missions (priority field)
Flag conflicts immediately when urgent mission is affected
Return actionable suggestions: "Reassign pilot X to mission Y to free up Z"
Require human dispatcher to make final decision

Why Monolithic App (Not Microservices)?

Current: Single Flask app with all logic
Benefits: Single docker deployment, no inter-service latency, easy to debug
Limitation: Can't scale conflict detection independently
Scale Plan: Split into Agent Service + Conflict Service + DB Service when needed

Why Rule-Based Fallback > Hard Failure?

Alternative: Fail with error if API down
Our choice: Gracefully degrade to pattern matching
Real-world impact: Demo at 2am when OpenAI quota exceeded? Works anyway ✓
User experience: No API key requirement for basic queries