Architecture & ETL Data Flow (Redis → MongoDB)

- **Architecture Overview:**
- - Sensors / App data → Node.js API → Redis cache → MongoDB Atlas (AgriDB.readings)
- - Redis holds hot data for fast retrieval (Cache-Aside, Read-Through, TTL)
- - MongoDB stores persistent data for long-term analytics

- **ETL Flow:**
- 1. Extract – Node.js collects readings via API
- 2. Transform – Validate JSON, add metadata (_ingestedAt, _source, meta.author)
- 3. Load – Insert into MongoDB; update Redis cache

- Key Message: Redis = Speed Layer, MongoDB = Storage Layer

Lessons Learned from Extending the Cached Database Pipeline

- Integrating Redis reduced latency from ~120 ms → ~20 ms (after caching)
- Learned Cache-Aside, Read-Through, and TTL expiration patterns
- Improved ETL pipeline with metadata tracking and validation
- Thunder Client testing showed clear cache-hit and TTL behavior

- Key Takeaway: Combining caching logic with ETL ensures speed and data freshness.

Challenges, What Worked Well, and What Did Not Work Well

- **Worked Well:**
- - Reliable Redis + MongoDB connections
- - Cache-Aside pattern gave visible performance gain
- - TTL handled auto-expiration effectively

- **Challenges:**
- - Manual cache invalidation after DB updates
- - Debugging Redis connection errors
- - Measuring TTL expiry precisely

- **Next Steps:**
- Automate invalidation using Pub/Sub
- mplement Write-Through caching
- Add visualization for cache performance metrics