Architecture & ETL Data Flow (Redis → MongoDB)

- Data Sources: IoT sensor readings / sample JSON documents
- ETL Flow:

1 Extract – Application requests data (API call)

2 Transform – If data found in Redis (cache hit) → return immediately; if cache miss, read from MongoDB and serialize into Redis

3 Load – Cached entries stored in Redis for faster subsequent reads

- Tools: Node.js API Server + MongoDB Atlas + Redis (local)

Diagram:

Client → Express API → Redis (Cache Layer) → MongoDB Atlas (Database)
↑______________________________↓ (Data refresh)

Lessons Learned from Implementing Redis Caching

- Explored Cache-Aside, Read-Through, and TTL-based strategies.
- Redis improved read performance by over 90% after first query.
- TTL expiration ensures automatic data freshness.
- Connected Redis and MongoDB Atlas using environment variables.
- Understood trade-offs between cache consistency and database accuracy.

- Visual Comparison:
- Before Redis → Slow (~130 ms)
- After Redis → Fast (~5 ms)

Challenges & Reflections

What Worked Well:

- Simple setup using createClient() and Mongoose.
- Easy caching validation through Thunder Client and Redis CLI.
- Major latency improvement after cache hits.

Challenges:

- Initial MongoDB URI and Redis connection issues.
- 'Cannot GET' error due to wrong server file execution.
- Managing TTL expiration timing.

Takeaway:

Integrating a caching layer transformed performance and taught balance between speed, consistency, and maintainability.