Cached Sensor Readings Pipeline

- Node.js API connects to **Redis** (cache) and **MongoDB Atlas** (AgriDB.readings).
- App reads by **cache-aside**: Request → check Redis → on miss, fetch from MongoDB → write to Redis → return.
- **TTL route** stores items in Redis with an expiration (60s) so stale data is auto-evicted.
- Seed script inserted **2,000 sensor readings** into AgriDB.readings (sensorId, reading, unit, updatedAt, meta.author).
- Tools: VS Code, MongoDB Compass, local Redis.

Client → Node API → (1) Redis → (2) MongoDB → Redis

Lessons Learned

- Caching patterns (cache-aside, read-through, TTL) are mostly the same code — the main difference is **when** you load Redis.
- Measured response time showed the second request was faster (15 ms → 9 ms) when data came from Redis.
- Having a consistent document shape in Mongo (meta.author) made filtering and screenshots easy.
- Environment variables matter: the app only worked when the Mongo URI matched the actual cluster (cluster0.lixbqmp.mongodb.net).
- It’s easier to seed data with a script than to hand-insert 2,000 rows in Compass.

Challenges & What Worked / Didn’t

- **Worked well:** Node connected to Redis and Mongo; seed script successfully inserted 2,000 docs; cache routes returned correct source flags (mongo, redis, redis (not expired)).
- **Didn’t at first:** “bad auth : Authentication failed” when using the wrong Atlas cluster (lab2cluster… vs cluster0…).
- **Fixed by:** Hard-coding the correct Mongo URI and then moving it back into .env.
- **Observation:** TTL makes performance “bounce” — fast while cached, back to slower after expiry.
- **Next steps:** Add write-through or invalidation on updates so Redis and Mongo never drift.