Lab-4: Redis + MongoDB

Name: Ravi Pareshbhai Kakadia

Architecture & ETL Data Flow

- Node.js server connects Redis (cache) and MongoDB Atlas (persistent store)
- ETL Flow:
  - Extract – IoT sensor data read from Redis (if cached) or MongoDB
  - Transform – Data enriched with metadata and timestamps
  - Load – Records stored in MongoDB “AgriDB.readings” Collection.
- Cache-Aside & Read-Through patterns implemented for efficient reads
- TTL (Expiration) ensures stale data automatically refreshes
- Architecture reduces DB load and improves read latency
- Redis handles fast access; MongoDB stores complete, reliable dataset

Lessons Learned

- Caching dramatically improves response time (100 ms → 5 ms average)
- TTL expiration helps maintain data freshness without manual cleanup
- Combining **Cache-Aside** and **Read-Through** clarifies real-time vs. persistent data roles
- MongoDB's flexible schema easily stores IoT data with nested metadata
- Redis proved ideal for temporary, high-speed operations
- Proper environment setup (.env, ports, WSL Redis) is critical for smooth integration

Challenges & Reflections

- **What Worked Well**
  - Redis connection and PING test successful inside WSL
  - Caching logic integrated smoothly with MongoDB queries
  - Noticeable performance boost with cached reads
  - Schema validation and indexes improved query reliability
- **What Didn’t Work Well**
  - Initial **CORS and body-parser** setup issues caused undefined requests
  - TTL testing required careful timing to observe expiration
  - Managing sync between cache and DB after updates can be complex
  - Redis memory management and eviction policies require tuning for scale

Thank You