System Design - End Term - Yerlan Dias

[Github repo link]

TASK 5

Functional requirements:

1. Registration and authentication: MFA
2. Vote casting
3. Anonymity/ Vote confidentiality
4. Encrypted storage
5. Results publishing
6. User friendly UI
7. Role-based access admin/voter

Non-functional requirements:

1. Scalability: handle peak load of 1M of concurrent users
2. Vote casting roundtrip latency: ≤ 200 ms (under normal load)
3. Tallying time (post-vote-close): ≤ 5 minutes for national scale (~10M votes)
4. Vote Storage Nodes (Encrypted): Min 250 GB × 3 nodes (replication factor = 3)
5. Audit Logs (WORM or IPFS-backed): ~100 GB/day (logs, proofs, votes)
6. Application servers (Ballot, Voter, Auth): 16 GB RAM minimum
7. Tally Orchestrator + Homomorphic compute nodes: 32 GB–64 GB RAM preferred
8. Mixnet nodes: 8–16 GB RAM
9. E2E encryption for all communication
10. Vote privacy: not personal info should be stored with the vote
11. Min 99.99% uptime during voting period
12. Continuity under highload

13. Usability for people of any category(disability)
14. Vote casting complete under <2 sec
15. >10K TPS encryption of votes

In terms of Security the system has strong auth with MFA, which prevents frauds, but it may frustrate non-technical voters. The mitigation is to provide fallback mechanisms (in-person registration, backup codes) and clear UX flows.

In terms of Anonymity mixnets, encrypted storage and ZKPs protect anonymity, however it makes troubleshooting, vote tracking and real-time verification more complex. The potential solution is to use ZKPs, Merkle trees, and observer dashboards to achieve verifiability without linkage

In terms of complexity of system asynchronous systems may scale and decouple logic, but Introduce complex failure modes, event replay bugs, and eventual consistency. The solution might be the Use idempotent consumers, event replay safeguards, and DLQs (Dead Letter Queues)

In terms of Performance homorphyc encryption enables secure, privacy-preserving tallying, but it's very CPU/memory intensive, especially at scale. Instead we can sse optimized libraries (e.g., SEAL, HElib), pre-aggregation, and GPU acceleration where possible