Technical assessment

Requirements for the below:

* Use [this dataset](https://drive.google.com/drive/folders/120NBGrMC7DSutH8C3Z5BgxqzRCzY0uA7) for the tasks below
* Use **docker to containerise your work**.
* Provide a README describing how to run your code

Additional requirements will be specified within each of the tasks below.

Task 1

Import the CRM events and CRM call center logs tables into a PostgreSQL database. Use SQL to join the tables and summarize the average time to resolve complaints across a number of different dimensions.

Provide a short presentation / slidedeck (submitted as a PDF) that summarizes your findings.

Task 2

Import the Luxury Loan Portfolio into pandas dataframes and use [Plotly dash](https://dash.plotly.com/) to create a web app that displays 3 charts of different types that show interesting business metrics.

Task 3

How would you create a LLM chat (a chatgpt-like solution or RAG applications) that uses only some company internal database? Please give details step by step, including data preparation, model evaluation, etc. It would be **a plus** if you could create strategies for reducing or preventing LLM hallucinations.

Task 4

How would you create a data platform end-to-end system? The data might have internal data or external data, but the end data would be stored into cloud platforms like Google Cloud Platform or Azure Platform or AWS Platform. Please give details step by step, including data preparation, model evaluation, etc.

**Hint for task 3 and task 4:**

* You can use a vector database or API for getting internal data.
* You can use MIRO or draw io or similar tools for creating framework / pipelines.

Task 5

Write a few short paragraphs of no more than 500 words on how you see the future of fintech developing when it comes to investment banking. Feel free to express your opinions and be creative, there is no single correct answer.

Submission Instruction

* **Time Limit:** You will be given 5 days to complete the test.
* **Deliverable :** Include the **link** to your completed test **in the body of your reply email**.
* **Submission:** Submit all deliverables (presentation file, code files, `README.md`) in a single linked folder to handik.yuwono[handik.yuwono@mandirisekuritas.co.id](mailto:handik.yuwono@mandirisekuritas.co.id)

and [valerian.pratama@mandirisekuritas.co.id](mailto:valerian.pratama@mandirisekuritas.co.id)

Task 2

Import the Luxury Loan Portfolio into pandas dataframes and use [Plotly dash](https://dash.plotly.com/) to create a web app that displays 3 charts of different types that show interesting business metrics.

# Task 3 — Build an internal-only LLM chat / RAG (step‑by‑step)

## 1) Data prep (from internal DB/files)

* **Inventory sources:** DB tables.
* **Access controls:** service account, fine‑grained RBAC/ABAC; never use a superuser key.
* **Export/ingest:**
  + DB: SQL extract views (mask PII at source when possible).
  + Files: use a text extraction pipeline (PDF, DOCX, HTML → plain text + metadata).
* **Normalize & clean:** remove boilerplate, headers/footers, menus, duplicated blocks.
* **Chunking:** split docs into **semantic chunks** (e.g., ~500–1,200 tokens) with **10–20% overlap**. Keep **metadata** (doc\_id, title, author, created\_at, acl\_tags, system\_version, url).
* **Embeddings:** choose a domain-suited model; store **vector + metadata**.
  + Vector stores: **pgvector** (Postgres).
* **Indexing:** build **hybrid** indexes: BM25 (keyword) **+** vector.

## 2) Retrieval pipeline

1. **Query rewriting** (optional): expand acronyms, apply HyDE/synonyms from glossary.
2. **Hybrid search:** top‑k from keyword (BM25) and vector; combine via **RRF** or weighted sum.
3. **Rerank:** cross‑encoder (e.g., MiniLM) to rerank top‑50 into top‑5 passages.
4. **Filter by ACL/metadata:** enforce row/document‑level security (team, region, data\_class).
5. **Compose context:** select ~2–5 passages, deduplicate, keep total tokens under your model’s limit.

## 3) Generation pipeline

* **Prompt template (system):** “Answer only from the provided context. Cite sources. If not in context, say you don’t know.”
* **User message:** their question.
* **Context message:** concatenated passages (with doc\_id + snippet).
* **Output spec:** JSON schema (answer, citations[], confidence, used\_docs[]).

### Minimal Python skeleton (framework-agnostic)

python

# Pseudocode: swap in your libs (LangChain, LlamaIndex, Haystack)

def retrieve(query, acl\_tags):

q2 = rewrite\_query(query) # optional

hits\_kw = bm25.search(q2, top\_k=50, filters=acl\_tags)

hits\_vec = vectordb.search(q2, top\_k=50, filters=acl\_tags)

merged = reciprocal\_rank\_fuse(hits\_kw, hits\_vec)[:50]

reranked = cross\_encoder\_rerank(q2, merged)[:5]

return reranked # [{text, doc\_id, score, meta}]

def generate\_answer(query, passages):

context = format\_passages(passages) # add doc ids + titles

prompt = render\_prompt(query=query, context=context)

out = llm.generate(prompt, response\_format=ANSWER\_SCHEMA) # enforce JSON

return out

def answer\_question(query, user\_acl):

ctx = retrieve(query, acl\_tags=user\_acl)

if not sufficient\_coverage(ctx): # score threshold / coverage check

return {"answer":"I don't know.", "citations":[], "confidence":0.2}

return generate\_answer(query, ctx)

## 4) Hallucination‑reduction strategies (practical)

* **Grounding & citations by design:** always pass only retrieved text in a separate context block; **require citations** (doc\_id + line).
* **Abstention:** if **relevance score < τ** or **coverage < threshold**, answer “I don’t know.”
* **Hybrid retrieval + reranking:** reduces missed evidence (recall) and off-topic context (precision).
* **Tight prompts:** “Do not guess. Quote exact lines when possible.”
* **Structured outputs:** JSON schema validation; reject if missing citations.
* **Context hygiene:** de‑duplicate, strip low-signal text, limit to top‑N passages.
* **Temporal filters:** prefer **latest** doc version; decay old docs.
* **Guardrails:** PII/secret scanning on outputs; allow only certain functions/knowledge.
* **Eval loop:** log “no context” and “low confidence” cases for curation/KB improvements.
* **Fine‑tuning (optional):** instruction‑tune on internal Q&A pairs + correct citations.

## 5) Evaluation (offline + online)

* **Retrieval metrics:** nDCG@k, Recall@k, MRR on labeled query→gold passage pairs.
* **Answer quality:**
  + **Groundedness/Faithfulness** (LLM-as-judge with rubric): does answer match context?
  + **Correctness** vs gold answers (Exact Match / F1 for extractive, rubric for abstractive).
  + **Citation accuracy:** does each citation support its sentence?
* **Safety:** red‑teaming prompts, data leakage checks, PII.
* **Latency & cost:** p50/p95 latency; $/1k queries.
* **Online:** A/B prompts, human feedback, ticket deflection %, CSAT.

## 6) Security & privacy

* **Network:** private VPC/VNet, no public endpoints; egress control.
* **Secrets:** KMS/Key Vault/Cloud KMS; per‑service IAM.
* **RBAC/ABAC:** propagate user identity; filter retrieval by ACL.
* **At-rest & in-flight encryption.**
* **Audit logs** for queries, retrieved docs, and returned citations.

## 7) Deploy & operate

* **Containerize** (Docker), deploy.
* **Observability:** structured logs (prompt, context hash, citations), traces, dashboards.
* **Caching:** query & embedding cache (e.g., Redis); pre‑warm popular docs.
* **Index refresh:** CDC from DB; nightly re-embed changed docs; blue/green index swaps.

# Task 4 — Design an end-to-end data platform to the cloud

## 0) Target outcomes

* Reliable pipelines for **internal & external data**.
* Medallion‑style layers (**Bronze/Raw → Silver/Cleansed → Gold/Curated**).
* Warehouse/lakehouse for BI + **ML/LLM** workloads.
* Governed, secure, cost‑efficient, observable.

## 1) Reference architecture (map to any cloud)

**Ingestion**

* **Batch:** Files/DB extracts → Object storage (ADLS).
* **Stream:** Kafka /Event Hubs.

**Storage**

* **Data Lake:** Parquet /ADLS (Bronze/Silver/Gold).
* **Warehouse/Lakehouse:** BigQuery/Snowflake /Databricks SQL.

**Processing**

* **Transform:** dbt (SQL ELT), Spark/Databricks,
* **Quality:** Great Expectations / Deequ; quarantine bad records.
* **Orchestration:** Airflow/Composer / Azure Data Factory

**Serving**

* **BI:** Looker/Power BI/Apache Superset
* **ML/LLM:** Vertex AI /Azure ML; feature store; RAG vector DB.
* **APIs:** FastAPI/Cloud Run/Lambda + API Gateway.

**Governance & Security**

* **Catalog/Lineage:** Data Catalog (GCP) / Purview (Azure) / Glue Data Catalog (AWS), OpenLineage.
* **Access:** IAM + Lake/Warehouse row/col policies; tokenization; DLP scanners.
* **Monitoring:** Cloud Logging/Monitoring + custom data SLAs (freshness, completeness).
* **IaC:** Terraform for everything; CI/CD with GitHub Actions/Azure DevOps/CodePipeline.

## 2) Cloud mappings (pick your stack)

| **Layer** | **GCP** | **Azure** | **AWS** |
| --- | --- | --- | --- |
| Object store | Cloud Storage | ADLS Gen2 | S3 |
| Stream | Pub/Sub | Event Hubs | Kinesis |
| Compute batch | Dataproc / Dataflow | Synapse / Databricks | EMR / Glue |
| Orchestration | Cloud Composer | Data Factory | Step Functions / MWAA |
| Warehouse | BigQuery | Synapse | Redshift / Athena |
| Lakehouse | BigLake/Iceberg/Delta via Dataproc | Delta Lake on ADLS | Delta/Iceberg on S3 |
| Catalog | Data Catalog | Purview | Glue Catalog |
| ML | Vertex AI | Azure ML | SageMaker |
| RAG vector | Matching Engine / AlloyDB PGVector | Azure AI Search / Cosmos DB + vectors | OpenSearch / Aurora PGVector / Pinecone |

## 3) Data lifecycle (step‑by‑step)

1. **Landing zone & foundations**
   * Org, projects/subscriptions, VPCs, subnets, Private Service Connect/Peering.
   * Centralized IAM & secrets management; audit logging enabled; budgets/quotas.
2. **Ingestion**
   * **Batch:** scheduled dbt seeds or extractor jobs (Airflow/ADF) → **Bronze** (as‑is).
   * **Stream:** CDC from OLTP via Debezium/DMS to Kafka/Kinesis → **Bronze** topics.
   * Apply **schema registry**, contracts, and PII tagging at ingress.
3. **Transformation**
   * **Silver:** standardize types, dedupe, conform dimensions, SCD2 for entities.
   * **Quality gates:** Great Expectations tests on each model (nulls, ranges, uniqueness).
   * **Gold:** business-ready marts (star schemas) for BI and ML features.
4. **Serving**
   * **BI:** publish approved datasets; row/column security for users.
   * **ML:** register features/models; enable RAG indexes built from curated docs/tables.
   * **APIs:** expose gold data via read‑only endpoints (rate-limited, cached).
5. **Governance**
   * Auto‑catalog new tables; classify PII; lineage from jobs to columns.
   * Policies: data retention, deletion, legal holds; data contracts with producers.
6. **Ops & FinOps**
   * SLAs/SLOs for pipelines; on‑call alerts (freshness, volume anomalies).
   * Cost dashboards per workspace/project; lifecycle policies (tiering, TTL).
   * Blue/green deploys for pipelines and warehouses; backfills via parameterized runs.

## 4) CI/CD & IaC (sketch)

* **Terraform** modules: network, storage, compute, secrets, catalog, warehouse, ML.
* **Data CI:** pre‑commit hooks, SQL lint, unit tests on dbt, data diffs (e.g., Elementary).
* **App CI:** build Docker images; scan vulnerabilities; deploy via ArgoCD/GitOps.

## 5) Evaluation & SLAs

* **Data quality KPIs:** freshness lag, % valid rows, duplication rate, schema drift events.
* **Pipeline KPIs:** success rate, p95 latency, backfill duration.
* **Consumption KPIs:** BI query latency, ML feature staleness, RAG answer groundedness.

## 6) Example dbt + Great Expectations pattern (concise)

sql

SalinEdit

-- models/silver/customers.sql

SELECT DISTINCT

id,

LOWER(email) AS email,

SAFE\_CAST(created\_at AS TIMESTAMP) AS created\_at,

country\_code

FROM {{ ref('bronze\_customers') }};

yaml

SalinEdit

# great\_expectations/expectations/customers.yml

expect\_column\_values\_to\_not\_be\_null:

column: email

expect\_column\_values\_to\_match\_regex:

column: email

regex: '^[^@]+@[^@]+\.[^@]+$'

## 7) RAG in the platform (where it fits)

* Build a **curated documents** zone in **Gold**, then index to your **vector DB**.
* Expose a **“/ask”** service (Vertex AI/SageMaker/Azure ML or custom) with the Task‑3 pipeline.
* Enforce **row‑level security** by propagating user identity to retrieval filters.

## Bonus — How to sketch this in Miro/draw.io quickly

* **Swimlanes:** Ingestion, Storage, Processing, Serving, Governance, Observability.
* **Shapes:**
  + Cylinders for storage (S3/GCS/ADLS, Warehouse).
  + Stacked docs for Bronze/Silver/Gold.
  + Gears for jobs (Airflow/ADF/Composer).
  + Magnifier for Catalog/Lineage.
  + Brain icon for LLM/RAG service.
* **Connectors:** solid arrows for data flows; dashed for control/metadata (catalog, lineage).
* **Callouts:** list SLAs, security policies, and cost controls next to each lane.

Task 5

The future of fintech in investment banking will likely be defined by deeper integration of technology into every layer of the deal-making, advisory, and capital-raising process. While automation has already transformed back-office operations, the next wave will move into the high-value, client-facing side of investment banking. Advanced AI models will assist bankers in market analysis, generating tailored financial strategies, and even identifying cross-border deal opportunities by processing vast datasets faster than any human team could. This won’t replace human expertise but will augment it, allowing bankers to spend more time on nuanced negotiation and relationship-building.

Blockchain and tokenization may also reshape the way securities and assets are issued and traded. Investment banks could facilitate instant settlement of complex financial products using blockchain infrastructure, reducing counterparty risk and operational costs. Tokenized equity, debt instruments, and even real assets could make capital markets more liquid and accessible to a broader range of investors while creating new fee streams for banks. The firms that adapt early will benefit from a first-mover advantage in this emerging ecosystem.

The winning banks will combine human skill with strong tech tools. They’ll use smart data to give clients investment plans designed just for them, while automated systems will handle most compliance and paperwork. Personalized, data-driven investment products will become the norm, powered by client behavioral analytics and real-time portfolio tracking. Regulatory technology (RegTech) will ensure compliance in an increasingly complex environment by automating reporting, risk management, and fraud detection. Cybersecurity will be paramount, as the same technologies that enable innovation will also attract more sophisticated threats. This will let bankers focus on building relationships and making big strategic decisions, instead of spending nights buried in spreadsheets.

Fintech will not erase the art of investment banking; it will refine it, making the practice more transparent, efficient, and client-centric. a hybrid between a technology company and a financial institution—leaner, faster, and more globally connected, but still anchored in human trust and judgment.