System Design Document (SDD)  
KOMEZA BANK – Churn Intelligence Platform for Bank of Kigali  
Version: 0.1  
Date: October 2025

**Document Control and Information**

| **Name** | **Title** | **Action** |
| --- | --- | --- |
| Rose NGABIRE | Chief Operations Officer | Approve |
| Nicholas MURIMI | Chief Product Development Officer | Review |
| Samuel MUTINDA | Head, Data & Analytics | Review |
| Steven SHYAKA | Machine Learning Engineer | Compile |

**Revision History**

| **Version** | **Author** | **Summary of Change** |
| --- | --- | --- |
| 0.1 | Steven SHYAKA | Initial SDD draft aligned with SRS and BRS |

**Distribution List**

| **Name** | **Area** | **Action** |
| --- | --- | --- |
| Samuel MUTINDA | Data Science and Analytics | Review |
| Herve TWAHIRWA | Data Science and Analytics | Solution Development |
| Alphonse MUHAYIMANA | Business Applications | CRM Integration |
| Sandra UWADEDE | Business Applications | CRM Integration |
| Francine UMULISA | Customer Experience | Comment/Noting |

**Glossary**

| **Term/Abbreviation** | **Definition** |
| --- | --- |
| ETL | Extract, Transform, Load |
| CDC | Change Data Capture |
| API | Application Programming Interface |
| CI/CD | Continuous Integration / Continuous Delivery |
| SHAP | SHapley Additive exPlanations |
| AUC | Area Under Curve |
| MLOps | Machine Learning Operations |

**1. Purpose**

This System Design Document (SDD) details the technical architecture, components, data flows, interfaces, and operational considerations for the KOMEZA BANK Churn Intelligence Platform tailored for Bank of Kigali. The document translates SRS requirements into design artifacts that developers, data engineers, and DevOps teams can implement.

**2. Design Principles**

1. **Security-first:** Sensitive data is protected by encryption, access controls, and anonymization prior to modelling.
2. **Modularity:** Loosely-coupled components (ingest, storage, model, serving, UI) for maintainability.
3. **Scalability:** Horizontal scaling for ETL and model serving to handle 1M+ customers.
4. **Observability:** Metrics, logs, and alerts for data pipelines and model performance.
5. **Compliance:** Design aligns with Law No. 058/2021 and BK internal policies.

**3. High-level Architecture**

Components:

* **Data Sources:** T24 Core DB, CRM, Digital App logs, Call Center logs, Loan system, Surveys.
* **Ingestion Layer:** Secure connectors (JDBC, APIs, SFTP) with CDC or batch jobs.
* **Raw Data Lake:** Immutable storage (object store) for raw extracts.
* **Data Warehouse / Feature Store:** Curated, cleaned tables and features for ML.
* **ETL / Feature Engineering:** Orchestrated pipelines (Airflow or Prefect).
* **Model Training Environment:** Containerized training using Python, scikit-learn/XGBoost, TensorFlow optional.
* **Model Registry:** MLflow or similar for version control and metadata.
* **Model Serving / Scoring Engine:** RESTful API (FastAPI) or batch scoring jobs.
* **Explainability Module:** SHAP service producing per-customer explanations.
* **Application Layer:** Dashboards (Power BI/Streamlit) and CRM integration APIs.
* **Monitoring:** Prometheus + Grafana for infra and model metrics; ELK stack for logs.

Architecture Diagram (conceptual):

[T24] [CRM] [Digital App] [Call Center] [Loan System]

\ | | | /

\ | | | /

--> Ingestion Layer (JDBC/API/SFTP/CDC)

↓

Raw Data Lake (Object Store)

↓

ETL / Feature Engineering (Airflow)

↓

Data Warehouse & Feature Store (SQL / Parquet)

↓

Model Training ----> Model Registry ----> Model Serving (REST)

↓ ↘

Explainability (SHAP) CRM / Dashboard

↓

Monitoring & Logging (Prometheus / ELK)

**4. Component Design**

**4.1 Data Ingestion**

**Requirements:** Secure extraction from T24, CRM, Digital App, Call Center.

**Design:**

* Use a combination of **CDC** (where available) and scheduled **batch ETL** jobs (daily or hourly).
* Connectors: JDBC for databases, REST for APIs, SFTP for flat files.
* Data validation hooks at extraction time (row counts, schema checks).
* Mask/remove PII during or immediately after ingestion in a secure staging zone.

**Implementation Notes:**

* Tooling: Airbyte / Fivetran (if licensed) or custom Python ETL scripts using dbt for transformations.
* Use service accounts with least privilege access.

**4.2 Raw Data Lake**

**Requirements:** Immutable storage of raw extracts for audit and reprocessing.

**Design:**

* Object store (S3-compatible) with folder layout: /raw/{source}/{YYYY}/{MM}/{DD}/.
* Retention policy for raw data per BK policy.
* Encryption at rest (server-side) and in transit.

**4.3 ETL & Feature Engineering**

**Requirements:** Clean, standardize, and derive features required by models.

**Design:**

* Orchestrator: Airflow / Prefect.
* Use modular DAGs: ingestion → validation → transformation → feature compute → load to feature store.
* Feature Store: Feast or homemade SQL-based feature tables refreshed daily/weekly.
* Example features: avg\_balance\_3mo, txn\_count\_3mo, digital\_login\_freq, days\_since\_last\_txn, complaints\_count.

**Data Quality Rules:**

* Null thresholds per column (alert if >10%).
* Range checks for numeric features (balances, counts).
* Unique key checks for Customer\_ID.

**4.4 Data Warehouse & Feature Store**

**Requirements:** Fast queries for dashboards and model training.

**Design:**

* Warehouse: PostgreSQL / Amazon Redshift / Snowflake depending on BK’s infra.
* Store features as time-stamped tables to support historical backfills and rolling windows.
* Indexing for fast lookups by Customer\_ID and Branch\_ID.

**4.5 Model Training Module**

**Requirements:** Reproducible training with versioning and experiment tracking.

**Design:**

* Containerized training jobs (Docker) executed on Kubernetes or managed compute.
* Use MLflow for experiment tracking and model registry.
* Baseline models: Logistic Regression, Random Forest, XGBoost. Candidate deep models if necessary.
* Training pipeline steps: data pull → split (time-based) → feature selection → hyperparameter tuning → evaluation → register model.

**Compute:**

* GPU not required for tree-based models. Use CPU instances (4–16 cores) depending on dataset size.

**4.6 Model Registry and Deployment**

**Requirements:** Version control for models and ability to rollback.

**Design:**

* Use MLflow or DVC + artifact store.
* Model metadata includes: training data snapshot, feature list, hyperparameters, performance metrics, and SHAP baseline.
* Promote models through stages: staging → production after validation.

**4.7 Model Serving / Scoring Engine**

**Requirements:** Provide batch and online scoring; low-latency API for CRM calls.

**Design:**

* **Batch Scoring:** Scheduled job runs nightly/weekly to score full customer base and write results to warehouse.
* **Online Scoring:** REST API (FastAPI) exposing /score?customer\_id= endpoint for on-demand scoring (<= 300ms target).
* Containerized service behind API Gateway with auth (JWT / mTLS).

**Scaling:**

* Horizontal scaling with Kubernetes HPA based on CPU and request latency.

**4.8 Explainability Module**

**Requirements:** Per-customer explanation of top churn drivers.

**Design:**

* Use SHAP library to compute feature contributions.
* Pre-compute explanations during batch scoring for all customers and store top-5 features per customer.
* Online explainability endpoint computes SHAP on-demand (with caching for recent requests).

**4.9 Application Layer (Dashboard & CRM Integration)**

**Requirements:** Interactive BI dashboards and CRM display of churn score.

**Design:**

* Dashboard: Power BI for executive use; Streamlit/React-based app for data teams (if custom UI needed).
* CRM Integration: API or scheduled CSV export to push churn scores and recommended actions into CRM.
* Role-based views and data masking in dashboards for sensitive fields.

**4.10 Monitoring, Logging & Alerting**

**Requirements:** Observability into data pipelines, model metrics, and infra.

**Design:**

* Metrics: Prometheus for infra and application metrics (latencies, error rates, scoring throughput).
* Logs: ELK (Elastic) or hosted logging for pipeline and application logs.
* Model Monitoring: Track data drift (feature distribution changes), prediction drift, and AUC over time.
* Alerts: PagerDuty / email for pipeline failures, low model performance, or data quality anomalies.

**5. API Specifications**

**5.1 Authentication**

* OAuth2 / JWT tokens issued by BK internal identity provider.
* mTLS for internal service-to-service communication.

**5.2 Endpoints**

**POST /api/v1/score/batch**

* Description: Trigger batch scoring for customer segments (internal use).
* Auth: Service account
* Payload: { "segment": "retail", "date": "YYYY-MM-DD" }
* Response: Job ID and status.

**GET /api/v1/score/{customer\_id}**

* Description: Retrieve latest churn score and top-5 drivers for a customer.
* Auth: JWT
* Response: { "customer\_id": "C123", "score": 0.78, "drivers": [{"feature":"days\_since\_last\_txn","impact":0.22}, ...] }

**GET /api/v1/model/status**

* Description: Returns current model version and training metrics.
* Auth: JWT
* Response: { "model\_version": "v1.2.0", "auc": 0.87, "trained\_on": "2025-09-15" }

**POST /api/v1/feedback**

* Description: Accept feedback from CRM/Telesales about recommended actions (accepted/rejected).
* Payload: { "customer\_id": "C123", "action": "accepted", "product": "Credit Card", "outcome\_date": "YYYY-MM-DD" }
* Use: Feed into model retraining and recommender refinement.

**6. Security Design**

* **Network Security:** Private subnets for sensitive systems, VPC peering for services.
* **Data Encryption:** AES-256 for data at rest, TLS 1.2+ for in transit.
* **Identity & Access Management:** RBAC via BK’s identity provider; MFA for admin access.
* **Secrets Management:** Vault or managed secrets service for DB credentials and API keys.
* **PII Handling:** PII hashed or tokenized in staging; only authorized roles can view raw PII in secure environment.

**7. Scalability & Performance**

* **Scaling:** Kubernetes for horizontal scaling, autoscaling for model-serving pods.
* **Caching:** Redis for caching recent scores and explanations to reduce compute.
* **Performance Targets:** Online scoring < 300ms; Dashboard queries < 3s for aggregated views.
* **Load Testing:** Locust scripts to simulate CRM and dashboard traffic.

**8. Operations & Maintenance**

* **CI/CD:** GitHub Actions / GitLab CI for build, test, and deployment pipelines.
* **Backups:** Nightly backups of feature store and model artifacts; weekly full backups of warehouse.
* **Runbooks:** Incident runbooks for pipeline failures, model degradation, and security incidents.
* **SLA:** 99.5% availability for core scoring and dashboard services.

**9. Testing Strategy**

* **Unit Tests:** For ETL logic, model preprocessing, and API handlers.
* **Integration Tests:** End-to-end tests from ingestion to model serving.
* **Model Validation:** Hold-out temporal split; cross-validation and performance threshold checks.
* **UAT:** Business users (Marketing, CX, BI) to validate dashboards and CRM workflows.
* **Load & Stress Tests:** Validate autoscaling and estimate resource needs.

**10. Deployment Plan**

**Phased Approach:**

1. **PoC Phase:** Minimal ingestion, baseline model, manual export to CRM.
2. **Pilot Phase:** Full ETL, scheduled batch scoring, Power BI dashboards, selected branches.
3. **Production Phase:** Automated pipelines, API integration with CRM, monitoring and SLAs.

**Rollback Strategy:**

* Maintain previous stable model in registry; API routes can be switched to previous model version.
* DB snapshots for quick restore.

**11. Risks & Mitigation (Technical Focus)**

| **Risk** | **Likelihood** | **Mitigation** |
| --- | --- | --- |
| Data schema changes in T24 | Medium | Schema registry and schema validation in ETL. |
| Unexpected data drift | High | Automated drift detection alerts and retraining cadence. |
| Infra cost overrun | Medium | Cost monitoring and budget alerts. |
| Latency spikes during peak | Low-Medium | Autoscaling and caching strategies. |

**12. Appendix**

**12.1 Example Feature List**

* avg\_balance\_3mo
* txn\_count\_3mo
* digital\_logins\_3mo
* days\_since\_last\_txn
* complaints\_count\_6mo
* loan\_outstanding\_amount

**12.2 Sequence: Scoring Request Flow (Online)**

1. CRM user requests score via /score/{customer\_id}.
2. API Gateway authenticates request.
3. Model Serving retrieves features from feature store or cache.
4. Model returns score; Explainability module provides top drivers.
5. Response returned to CRM and logged for audit.

**12.3 Sequence: Batch Scoring Flow**

1. Scheduler triggers batch job for date range.
2. ETL assembles feature snapshot and writes to feature store.
3. Batch scoring job loads model from registry and scores all customers.
4. Scores and top-5 drivers written to warehouse and pushed to CRM via export or API.

**Conclusion**

This SDD provides the technical blueprint for implementing the KOMEZA BANK Churn Intelligence Platform for Bank of Kigali. It balances scalability, security, and maintainability while aligning with BK's operational needs. The document is ready for review by Data Engineering, DevOps, and Security teams for implementation planning.