DOCUMENT DE CONCEPTION TECHNIQUE

Plateforme DigitalBank

Phase 1.2 : Architecture de la Solution

Groupe DevForce

Janvier 2026

Version 1.0

TABLE DES MATIÈRES

1. INTRODUCTION

Ce document présente l'architecture technique détaillée de la plateforme DigitalBank, une solution de gestion bancaire moderne intégrant des capacités d'analyse de données, de détection de fraude par intelligence artificielle et de monitoring en temps réel.

Objectifs du document :

- • Décrire l'architecture technique globale de la plateforme
- • Présenter le modèle de données et les relations entre entités
- • Détailler les mécanismes de sécurité implémentés
- • Justifier les choix technologiques effectués

Contexte du projet :

La plateforme DigitalBank s'inscrit dans le cadre d'un projet académique visant à concevoir et développer une solution bancaire moderne utilisant des technologies no-code/low-code. Le système gère actuellement :

- • 10 clients avec leurs comptes bancaires
- • 13 comptes actifs (épargne, courant)
- • 30 transactions historiques analysées
- • Détection de fraude en temps réel via Machine Learning

2. ARCHITECTURE TECHNIQUE GLOBALE

2.1 Vue d'ensemble

L'architecture de DigitalBank suit un modèle en couches (layered architecture) garantissant la séparation des responsabilités, la maintenabilité et la scalabilité du système.

Référence : Voir Schéma 1 - Architecture Technique Globale (annexe)

2.2 Composants principaux

COUCHE PRÉSENTATION

- • Metabase : Dashboards Business Intelligence pour l'analyse des données bancaires, visualisation des KPIs et reporting automatique.
- • Grafana : Monitoring de l'infrastructure et des performances système avec alerting en temps réel.
- • Interface Web : Application client responsive permettant aux utilisateurs d'accéder à leurs comptes.

COUCHE APPLICATION

- • Supabase : API REST auto-générée, authentification JWT + MFA, Row Level Security (RLS) et subscriptions temps réel.
- • n8n : Orchestration des workflows automatisés (alertes email, notifications Slack, rapports périodiques).
- • API Flask ML : Service de Machine Learning pour la détection de fraude utilisant un modèle Random Forest.

COUCHE DONNÉES

- • PostgreSQL 14 : Base de données relationnelle contenant 5 tables principales (customers, accounts, transactions, cards, audit_logs). Restaurée depuis la Phase 1 avec ajout de la table audit_logs pour la traçabilité.

MONITORING

- • Prometheus : Collecte des métriques système et applicatives.
- • Grafana : Visualisation des métriques et configuration des alertes.

3. MODÈLE DE DONNÉES

3.1 Structure des tables

Le modèle de données suit les principes de normalisation (3NF) et inclut 5 tables

principales. Référence : Voir Schéma 2 - Modèle de Données ERD (annexe)

TABLE CUSTOMERS

Stocke les informations clients avec chiffrement des données sensibles (email, phone).

- • Clé primaire : customer_id (INTEGER)
- • Attributs : first_name, last_name, email (chiffré), phone (chiffré), address, created_at

TABLE ACCOUNTS

Gère les comptes bancaires associés aux clients.

- • Clé primaire : account_id (INTEGER)
- • Clé étrangère : customer_id → CUSTOMERS
- • Attributs : account_type, balance, iban (chiffré), status, created_at

TABLE TRANSACTIONS

Enregistre toutes les transactions avec prédiction de fraude par ML.

- • Clé primaire : transaction_id (INTEGER)
- • Clé étrangère : account_id → ACCOUNTS
- • Attributs : amount, merchant, category, location, timestamp
- • Champs ML : is_fraud (BOOLEAN), fraud_score (FLOAT 0-1)

TABLE CARDS

Gère les cartes bancaires liées aux comptes.

- • Clé primaire : card_id (INTEGER)
- • Clé étrangère : account_id → ACCOUNTS
- • Attributs : card_number (chiffré), card_type, expiry_date, status

TABLE AUDIT_LOGS (NOUVELLE)

Table de traçabilité pour la conformité RGPD Article 30.

- • Clé primaire : log_id (SERIAL)
- • Attributs : user_id, action (SELECT/UPDATE/DELETE), table_name, record_id, timestamp, ip_address, user_agent
- • Rétention : 1 an minimum

3.2 Relations et contraintes

Le modèle suit des relations One-to-Many (1:N) :

- • CUSTOMERS (1) → (N) ACCOUNTS : Un client peut avoir plusieurs comptes
- • ACCOUNTS (1) → (N) TRANSACTIONS : Un compte génère plusieurs transactions

- • ACCOUNTS (1) → (N) CARDS : Un compte peut avoir plusieurs cartes

4. ARCHITECTURE DE SÉCURITÉ

L'architecture de sécurité suit une approche Defense in Depth avec 6 niveaux de protection.

Référence : Voir Schéma 3 - Architecture de Sécurité (annexe)

4.1 Authentification

NIVEAU 1 : Authentification multi-facteurs

- • Email + Mot de passe : Première couche d'authentification
- • MFA (Multi-Factor Authentication) : SMS ou TOTP (Google Authenticator)
- • JWT Tokens : Durée de vie 1 heure, algorithme HS256 (HMAC SHA-256)
- • Refresh Tokens : Validité 7 jours, stockés en httpOnly cookies
- • Session Timeout : 30 minutes d'inactivité

4.2 Contrôle d'accès (RBAC)

NIVEAU 2 : Role-Based Access Control

4 rôles définis selon le principe du moindre privilège :

- • admin : Accès complet (lecture/écriture/suppression sur toutes les tables)
- • analyst : Lecture seule sur toutes les données pour analyse
- • customer_service : Lecture + capacité de bloquer comptes/cartes
- • customer : Accès uniquement à ses propres données

4.3 Chiffrement

NIVEAU 3 : Row Level Security (RLS)

Filtrage automatique au niveau PostgreSQL : WHERE customer_id = auth.uid()

NIVEAU 4 : Chiffrement des
données At-Rest (stockage) :

- • Algorithme : AES-256 via extension pgcrypto de PostgreSQL
- • Données chiffrées : email, phone, iban, card_number
- • Gestion des clés : Supabase Vault (clé maître)

In-Transit (transmission) :

- • Protocole : TLS 1.3 (HTTPS obligatoire)
- • Certificat SSL : Let's Encrypt (renouvellement automatique)

4.4 Audit et traçabilité

NIVEAU 5 : Logging et audit

- • Table audit_logs : Enregistrement de toutes les actions sensibles
- • Actions tracées : SELECT, UPDATE, DELETE sur données sensibles
- • Informations capturées : user_id, timestamp, IP source, user_agent

- • Conformité : RGPD Article 30 (registre des activités de traitement)
- • Rétention : 1 an minimum, puis archivage ou suppression

5. CHOIX TECHNOLOGIQUES

5.1 Stack technique retenu

La stack technologique a été sélectionnée selon une approche pragmatique privilégiant les solutions no-code/low-code pour accélérer le développement.

5.2 Justifications et alternatives

BASE DE DONNÉES : PostgreSQL 14

Justification :

- • Base relationnelle robuste et mature
- • Support natif du JSON (flexibilité)
- • Extension pgcrypto pour chiffrement
- • Performance excellente (jusqu'à 10M lignes)
- • Open-source et gratuit

Alternatives considérées :

- • MySQL : Moins de fonctionnalités avancées
- • MongoDB : Moins adapté aux relations complexes

API LAYER : Supabase
Justification :

- • API REST auto-générée à partir du schéma
- • Authentification JWT intégrée
- • Row Level Security natif
- • Realtime subscriptions (WebSocket)
- • Gratuit jusqu'à 500MB (adapté au projet)

Alternatives considérées :

- • Hasura : Plus complexe, courbe d'apprentissage
- • Xano : Pricing élevé, moins de flexibilité

DASHBOARDS : Metabase
Justification :

- • Interface intuitive pour les non-techniciens
- • Connexion directe à PostgreSQL
- • Visualisations variées (graphiques, tableaux, cartes)
- • Rapports automatiques par email
- • Open-source et gratuit

Alternatives considérées :

- • Grafana : Meilleur pour métriques techniques que BI
- • Retool : Nécessite plus de

configuration WORKFLOW

AUTOMATION : n8n

Justification :

- • Interface visuelle drag-and-drop
- • 400+ intégrations (Email, Slack, PostgreSQL, HTTP)
- • Self-hosted (contrôle total des données)
- • Gratuit en version Community

Alternatives considérées :

- • Make.com (Integromat) : Payant dès 10 000 opérations/mois
- • Zapier : Coût élevé, pas de self-hosting

MONITORING : Prometheus + Grafana
Justification :

- • Stack standard de l'industrie
- • Prometheus : Time-series database pour métriques
- • Grafana : Dashboards temps réel et alerting
- • Open-source et gratuit

Alternatives considérées :

- • ELK Stack : Plus lourd, nécessite plus de ressources
- • Datadog : SaaS payant, coût élevé

MACHINE LEARNING : Python Flask + scikit-learn
Justification :

- • Python : Langage standard pour ML
- • Flask : Framework léger pour API REST
- • Random Forest : Algorithme robuste pour détection fraude
- • Temps de réponse : < 500ms

5.3 Critères de décision

Les choix technologiques ont été guidés par 7 critères principaux :

1. COÛT

- • Budget projet limité (contexte académique)
- • Privilégier l'open-source et les tiers gratuits
- • Stack actuelle : 0€/mois (sauf hébergement)

2. FACILITÉ D'UTILISATION

- • Interface visuelle préférée au code
- • Documentation claire et exemples nombreux
- • Communauté active pour support

3. PERFORMANCE

- • Temps de réponse API : < 200ms
- • Détection fraude : < 2 minutes (bout en bout)
- • Scalabilité jusqu'à 100 000 transactions/jour

4. SÉCURITÉ

- • Conformité RGPD obligatoire
- • Chiffrement end-to-end
- • Audit trail complet

5. INTÉGRATION

- • Compatibilité entre les outils
- • API REST standard
- • Webhooks pour temps réel

6. MAINTENABILITÉ

- • Code minimal (no-code privilégié)
- • Mise à jour automatique des dépendances
- • Backups automatiques

7. ÉVOLUTIVITÉ

- • Possibilité d'ajouter des features facilement
- • Architecture modulaire
- • Migration possible vers solutions entreprises

6. CONCLUSION

L'architecture technique de la plateforme DigitalBank a été conçue selon les meilleures pratiques de l'industrie tout en tenant compte des contraintes du projet (budget, temps, compétences).

Points forts de l'architecture :

- • Séparation claire des responsabilités (architecture en couches)
- • Sécurité multi-niveaux (Defense in Depth)
- • Stack moderne et évolutive
- • Coût d'exploitation minimal (0€/mois hors hébergement)
- • Performance satisfaisante (< 2min pour détection fraude)

Limitations identifiées :

- • Solutions no-code limitées pour personnalisation avancée
- • Dépendance aux tiers (Supabase, Metabase)
- • Scalabilité limitée (jusqu'à 100K transactions/jour)

Évolutions futures recommandées :

- • Migration vers Kubernetes pour orchestration
- • Implémentation de caching (Redis)
- • Ajout d'une API GraphQL (en complément REST)
- • Machine Learning : modèles plus avancés (Deep Learning)
- • Tests automatisés (CI/CD pipeline)

En conclusion, l'architecture proposée répond aux exigences fonctionnelles et non-fonctionnelles du projet tout en offrant une base solide pour des évolutions futures. La stack technologique retenue permet un développement rapide grâce aux solutions no-code/low-code, tout en maintenant des standards professionnels de sécurité et de performance.

Le système est prêt pour une mise en production dans un environnement contrôlé, avec un monitoring en place pour identifier rapidement d'éventuels problèmes.

ANNEXES

Les schémas d'architecture suivants sont disponibles en fichiers séparés :

- • Schéma 1 : Architecture Technique Globale (Schema_Architecture_Technique_Globale.png)
- • Schéma 2 : Modèle de Données ERD (Schema_Modele_Donnees_ERD.png)
- • Schéma 3 : Architecture de Sécurité (Schema_Architecture_Securite.png)
- • Schéma 4 : Flux de Détection de Fraude (Schema_Flux_Detection_Fraude.png)

Ces schémas illustrent de manière détaillée l'architecture, les flux de données et les mécanismes de sécurité décrits dans ce document.

FIN DU DOCUMENT

Groupe DevForce - Janvier 2026