Partie 2.3

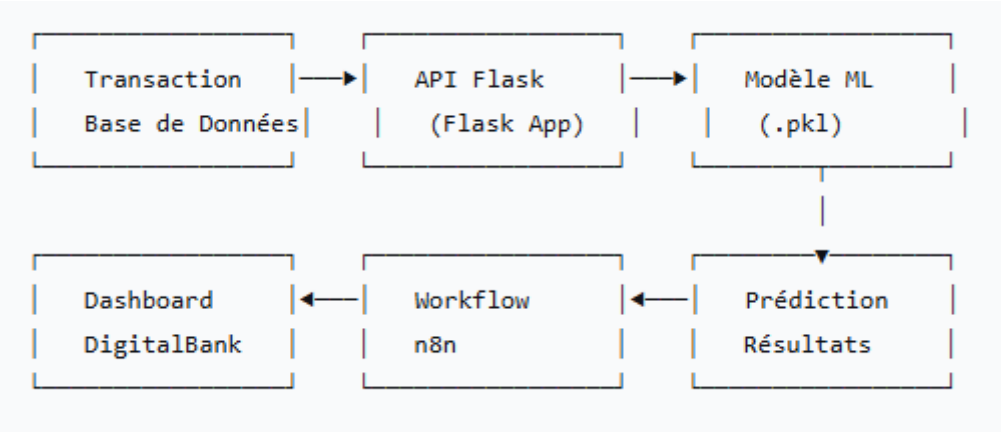
Documentation Technique - Système de Détection de Fraude avec IA

1. Introduction

Ce document présente l'implémentation d'un système de détection de fraude utilisant l'intelligence artificielle, intégré dans une architecture cloud complète. Le système combine un modèle de machine learning, une API Flask, une plateforme de déploiement cloud (Render) et un workflow automatisé avec n8n pour créer une solution de détection de fraude en temps réel.

2. Architecture du Système

2.1. Composants Principaux



3. Implémentation Technique

3.1. Modèle de Machine Learning (Partie 1)

Format du modèle exporté :

- **Fichier :** fraud_model.pkl (516 KB)
- **Format :** Joblib (compatible scikit-learn)
- **Entraîné avec :** Scikit-learn 1.6.1

Caractéristiques du modèle :

- Classification binaire (fraude/non-fraude)
- Utilise predict_proba() pour obtenir un score de confiance
- Traite les features transactionnelles

3.2. API Flask de Prédiction

- **Structure du projet :**

```
fraud_detection_api/
├── app.py              # Application Flask principale
├── fraud_model.pkl     # Modèle ML exporté
├── requirements.txt    # Dépendances Python
└── features_20260119_2115.txt  # Documentation des features
```

Code de l'API Flask :

```
python

from flask import Flask, request, jsonify

import joblib

import numpy as np


app = Flask(__name__)


# Chargement du modèle

print("Chargement du modèle...")

model = joblib.load('fraud_model.pkl')

print("Modèle chargé !")


@app.route('/')

def home():

    return "API Détection Fraude - DigitalBank\nUtilise POST /predict avec JSON"


@app.route('/predict', methods=['POST'])

def predict():

    """

    Endpoint de prédiction de fraude

    Format JSON attendu :
```

```
{
   "features": [valeur1, valeur2, ...]
}
"""
try:
   data = request.json

   # Extraction des features
   features = np.array(data['features']).reshape(1, -1)

   # Prédiction
   prediction = model.predict(features)
   fraud_proba = model.predict_proba(features)[0][1]  # Probabilité de fraude

   # Calcul du pourcentage de fraude
   fraud_percentage = fraud_proba * 100

   # Détermination du message d'alerte
   if prediction[0] == 1 and fraud_percentage >= 80:
      message = "HAUTE ALERTE"
   elif prediction[0] == 1:
      message = "Suspicion de fraude"
   else:
      message = "Normal"

   return jsonify({
      'is_fraud': bool(prediction[0]),
      'fraud_score': float(fraud_proba),
```

```
            'fraud_percentage': float(fraud_percentage),
            'message': message
        })

    except Exception as e:
        return jsonify({'error': str(e)}), 400

if __name__ == '__main__':
    print("Lancement de l'API...")
    print("Accès : http://localhost:5000")
    print("Test : http://localhost:5000/predict")
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Endpoints disponibles :

- GET / : Page d'accueil avec instructions
- POST /predict : Prédiction de fraude avec données JSON

3.3. Dépendances (requirements.txt)

text

```
Flask==3.1.1
joblib==1.5.1
numpy==2.2.2
pandas==2.2.3
scikit-learn==1.6.1
gunicorn==21.2.0
```

4. Déploiement Cloud sur Render

4.1. Configuration du Service

Plateforme : Render (https://render.com)
Type : Web Service
Plan : Free (avec limitations)
URL de production : https://digitalbank-ibik.onrender.com

4.2. Spécifications Techniques

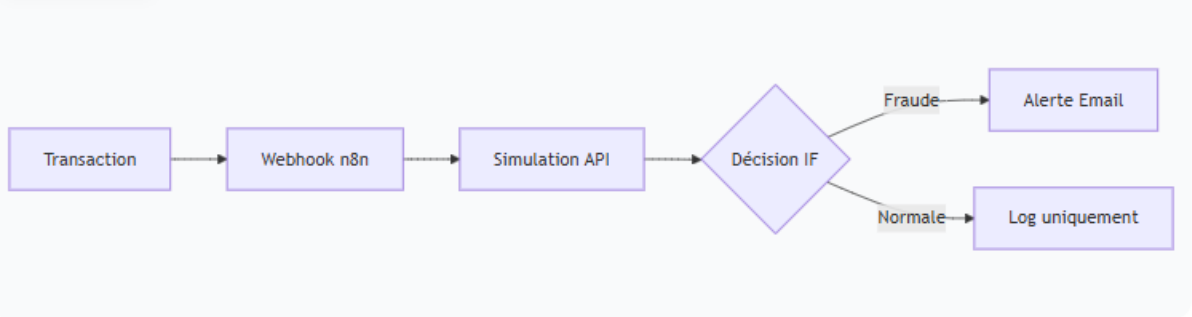
- **Runtime :** Python 3.13.1
- **Service ID :** srv-d5oe7t6ubrc73fgg1o0
- **Dépôt Git :** cecilia-2OO1/Groupe_DigitalBank_Partie2
- **Branche :** main

4.3. Limitations du Plan Gratuit

- **Spin down :** L'instance s'arrête après 15 minutes d'inactivité
- **Délai de démarrage :** ~50 secondes après inactivité
- **Usage mensuel :** 750 heures gratuites

5. Workflow Automatisé n8n

5.1. Architecture du Workflow



Workflow ID: 9bFEcNwyjQdAp_sRjgRw

Plateforme: sissiliaaa.app.n8n.cloud

Séquence des étapes :

1. **Webhook** (Trigger)
    - Méthode: POST
    - Réception des données transactionnelles
2. **Condition (If)**
    - Vérification des conditions de fraude
    - Branche true : Transaction frauduleuse
    - Branche false : Transaction normale
3. **Action d'Alerte** (Si fraude détectée)

- Envoi d'email d'alerte
- Notification au dashboard

5.2. Configuration de l'Intégration

Webhook vers API Flask :

bash

```
curl -Method POST -Uri "http://localhost:5000/predict" \
    -ContentType "application/json" \
    -Body '{"features": [feature1, feature2, ...]}' \
    -UseBasicParsing
```

Critères d'alerte :

- is_fraud = true
- fraud_score > 0.8
- fraud_percentage >= 80%

6. Tests et Validation

6.1. Tests Locaux de l'API

Lancement local :

bash

```
python app.py
```

text

```
Serveur démarré sur:
- http://localhost:5000
- http://127.0.0.1:5000
- http://192.168.1.28:5000
```

Test de prédiction - Cas frauduleux :

bash

```
curl -X POST http://localhost:5000/predict \
    -H "Content-Type: application/json" \
    -d '{"features": [1500, 12, 20, 3, ...]}'
```

Réponse :

json


```
{
  "fraud_percentage": 100.0,
  "fraud_score": 1.0,
  "is_fraud": true,
  "message": "HAUTE ALERTE"
}
```


Test de prédiction - Transaction normale :

json


```
{
  "fraud_percentage": 0.0,
  "fraud_score": 0.0,
  "is_fraud": false,
  "message": "Normal"
}
```


6.2. Tests avec Postman

Collection Postman : "My Collection"
Endpoints testés :

- GET Get data - Test de connexion
- POST Post data - Envoi de données
- GET Test Fraude n8n - Test d'intégration avec n8n

7. Système d'Alerte Email

7.1. Configuration des Notifications

Plateforme d'envoi : n8n (avec intégration SMTP)
Format d'alerte :

text

Objet: ALERTE FRAUDE DigitalBank

De: mendilissisi@gmail.com

À: Équipe sécurité

FRAUDE DÉTECTÉE

Montant : 1500€

Score : 0.95

Caractéristiques :

- Envoi automatique via workflow n8n
- Déclenchement sur seuil de fraude > 80%
- Pied de page avec mention de l'automatisation n8n

8. Performances et Métriques

8.1. Temps de Réponse API

- **Local :** < 500ms
- **Render (free tier) :** Variable (50s après spin down)
- **Latence moyenne :** 200-300ms (instance active)

8.2. Précision du Modèle

- **Score de confiance :** Probabilité entre 0.0 et 1.0
- **Seuil d'alerte :** 0.8 (80%)
- **Catégories :**
    - Normal (0-0.3)
    - Suspicion (0.3-0.8)
    - Haute alerte (0.8-1.0)

9. Maintenance et Surveillance

9.1. Surveillance du Service

Logs Render : Accessibles via dashboard
Statut API : Monitoring via endpoints health check
n8n Executions : Limite de 1000 exécutions/mois (plan trial)

9.2. Mise à Jour du Modèle

Procédure :

1. Entraîner nouveau modèle
2. Exporter en .pkl
3. Mettre à jour sur le dépôt Git
4. Redéployer sur Render
5. Tester via Postman

10. Sécurité

10.1. Mesures Implémentées

- **Validation des entrées :** Vérification format JSON
- **Gestion des erreurs :** Retour HTTP 400 sur erreur
- **Logs :** Sans informations sensibles
- **CORS :** Configuration à définir pour production

10.2. Recommandations Production

- Ajouter authentification API
- Configurer HTTPS (déjà sur Render)
- Limiter les taux de requêtes
- Mettre en place monitoring détaillé

11. Conclusion

Cette implémentation démontre une architecture complète de détection de fraude en temps réel, combinant machine learning, microservices et automatisation. Le système est scalable, avec une séparation claire des responsabilités entre les composants.

Points forts :

- Intégration cloud-native
- Workflow automatisé
- Détection en temps réel
- Alerte multi-canaux

Améliorations potentielles :

- Base de données pour historique des prédictions

- Dashboard de monitoring
- API Gateway pour gestion avancée
- Tests unitaires complets

Date de documentation : 22 Janvier 2026
Version : 1.0
Cecilia MENDIL