REPUBLIQUE DU CAMEROUN
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
****

REPUBLIC OF CAMEROON
MINISTRY OF HIGHER EDUCATION
****

ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE DE DOUALA
BP: 2701 DOUALA
TEL:(237) 697 542 240

Département : GIT / GLO

NIVEAU : 4

UE : INGENIEURIE ET CONCEPTION LOGICIELLE

PROJET

THEME:

CONCEPTION D'UNE APPLICATION DE PRÉDICTION DU RISQUE DE DÉFAUT POUR LES INSTITUTIONS DE MICRO-FINANCE CAMEROUNAISE

Par

- **TAMBOU NOUMSI CELIANE BLANCHE** ***22G00378***
- **NGAMBI YPECHEUN WILFRIED DE DESTINO** ***22G00303***
- **YETMOU NOUMI ROBERT CLAYRTON** ***22G00546***
- **LAMBOWO DONGMO DARIUS** ***22G00476***
- **KAMDEU YOUMBISSIE CHRETIEN ROSTINI** ***22G00173***

Sous l'encadrement de :

Dr. IHONOCK

Chargé de cours à l'ENSPD

Année académique 2025-2026

Table des matières

Liste des figures

INTRODUCTION GENERALE

La microfinance occupe une place essentielle dans le paysage financier camerounais en offrant des services de crédit accessibles aux populations souvent exclues du système bancaire traditionnel. Cependant, les institutions de microfinance (IMF) font face à une problématique majeure : la difficulté d'anticiper avec précision les risques de défaut de remboursement, lesquels peuvent affecter leur stabilité financière et leur capacité à octroyer de nouveaux crédits. Dans un contexte marqué par une concurrence accrue et un volume croissant de données clients, la mise en place d'outils numériques intelligents s'impose pour garantir des décisions rapides, fiables et basées sur l'analyse statistique. C'est dans cette optique que s'inscrit le présent projet, dont l'objectif principal est la conception d'une application de prédiction du risque de défaut reposant sur des techniques modernes de Machine Learning, intégrées au sein d'une architecture applicative robuste comprenant un backend développé avec FastAPI2 et un frontend React. Le document présente successivement le contexte et l'analyse des besoins du système, la phase de conception orientée UML et architecture logicielle, l'implémentation de la solution, puis les tests et modalités de maintenance. À travers cette démarche, le projet ambitionne de contribuer à l'amélioration des outils de gestion du risque au sein des IMF camerounaises, en mettant à leur disposition un système performant, automatisé et adapté à leurs réalités opérationnelles.

CHAPITRE I : ANALYSE DES BESOINS / PLANIFICATION DU PROJET

I. Introduction (Contexte du projet)

Dans un contexte où le secteur de la microfinance au Cameroun connaît une croissance, les institutions de microfinance (IMF) sont de plus en plus sollicitées pour l'octroi de crédits aux populations vulnérables. Cependant, l'octroi de crédit comporte un risque de défaut important. Afin de réduire ces risques et d'améliorer la fiabilité des décisions de crédit, il est nécessaire de disposer d'un outil automatisé, précis et rapide permettant de prédire le risque de défaut avant l'octroi. Ce projet vise donc à concevoir et développer une application web de prédiction du risque de défaut pour les IMF camerounaises, en s'appuyant sur des techniques de Machine Learning et une architecture moderne backend-frontend.

II. Étude de l'existant

Actuellement, dans de nombreuses IMF, l'évaluation du risque de crédit repose sur des méthodes manuelles ou semi-manuelles : recueil des informations client, calculs manuels ou basés sur des grilles d'analyse, décision fondée sur l'expérience du personnel de crédit. Ces méthodes peuvent être longues, sujettes à l'erreur humaine, et ne tiennent pas toujours compte de nombreux facteurs combinés.

➢ Limites des outils manuels et traditionnels

Les principaux inconvénients des approches traditionnelles sont :

• **Subjectivité** : l'évaluation dépend fortement de l'expérience de l'agent de crédit.

• **Lenteur** : le traitement manuel des données, le recueil des informations, le calcul des scores de risque sont chronophages.

• **Incohérence** : différentes évaluations peuvent aboutir à des décisions différentes pour des profils similaires.

• **Difficulté de mise à jour** : les grilles d'analyse peuvent devenir obsolètes, sans possibilité simple d'ajuster les pondérations en fonction des évolutions économiques ou des données historiques.

➢ **Besoins observés dans les institutions locales**

Les IMF camerounaises ont besoin d'un outil capable de :

• Centraliser les données des clients et des crédits ;

• Automatiser l'analyse de risque de défaut ;

• Fournir un score de risque fiable et cohérent ;

• Générer des rapports et historiques de décisions de crédit ;

• Assurer la confidentialité et la sécurité des données financières ;

• Permettre une prise de décision plus rapide et plus objective.

III. Problématique

Malgré l'importance croissante du micro-crédit, les IMF rencontrent des difficultés concrètes :

• Incapacité à anticiper efficacement les défauts de paiement ;

• Processus décisionnels lourds et longs, retardant l'octroi des crédits ;

• Recours à des méthodes empiriques, peu standardisées, parfois imprécises ;

• Absence d'historique structuré et exploitable pour apprendre des défauts passés.

Pour remédier à ces lacunes, un outil automatisé, structuré, fiable et sécurisé est indispensable.

IV. Solution proposée : Description générale de l'application de prédiction

L'application proposée est une plateforme web full-stack : un backend Python (FastAPI) fournissant une API REST pour le traitement des données, l'exécution d'un modèle de Machine Learning et la gestion des utilisateurs ; un frontend React.js pour l'interface utilisateur, l'import des données, la visualisation des scores de risque, la génération de rapports, et l'administration.

➢ **Intégration du Machine Learning**

L'application s'appuie sur un modèle ML (entraîné sur des données historiques de crédit) capable de prédire la probabilité de défaut pour un client donné, en fonction de ses caractéristiques financières et socioprofessionnelles.

➢ **Architecture cible**

L'architecture se compose au minimum de :

• Backend API REST (FastAPI) ;

• Base de données relationnelle (par ex. PostgreSQL ou MySQL) pour stocker les clients, crédits, historiques, résultats des prédictions, utilisateurs ;

• Frontend React.js + TailwindCSS + Shadcn UI ;

• Interface d'administration et de génération de rapports ;

• Mécanisme d'authentification et de gestion des utilisateurs.

- **Contribution du projet (codage backend + conception)**

Le backend implémenté réalise les fonctionnalités essentielles : authentification, import / saisie des données client, exécution du modèle ML, enregistrement de l'historique, API documentée (Swagger / OpenAPI), ce qui permet de transformer directement la solution en produit utilisable.

V. Analyse des besoins fonctionnels et non fonctionnels

L'analyse des besoins permet de définir les fonctionnalités attendues par les utilisateurs finaux ainsi que les exigences techniques nécessaires pour garantir fiabilité, performance et sécurité. Elle constitue un élément fondamental avant la conception et le développement d'une application basée sur le Machine Learning pour les IMF.

A. Besoins fonctionnels

1. Authentification / Gestion des utilisateurs

Le système doit permettre :

• La création de comptes utilisateurs (administrateurs, analystes de crédit) ;

• La connexion sécurisée via identifiants et mot de passe ;

• La gestion des rôles (permissions et accès) ;

• L'utilisation d'un mécanisme d'authentification sécurisé (JSON Web Token – JWT).

2. Import / saisie des données clients

L'application doit permettre :

• L'importation de fichiers CSV contenant les données des clients ;

• La saisie manuelle des informations de crédit via un formulaire ;

• Le contrôle de validité des données importées (format, valeurs manquantes, types).

3. Nettoyage, traitement et normalisation des données

Avant l'exécution du modèle ML, les données doivent être automatiquement :

• Nettoyées (traitement des valeurs manquantes, duplications) ;

• Normalisées ou standardisées selon les besoins du modèle ;

• Converties en format compatible avec l'algorithme utilisé.

4. Exécution du modèle de prédiction

Le backend doit permettre :

• Le chargement d'un modèle ML pré-entraîné ;

• L'exécution de prédictions sur les données d'un client ou d'un groupe de clients ;

• Le renvoi d'un score de risque (probabilité de défaut) ;

• Un retour structuré permettant au frontend d'afficher la décision.

5. Visualisation des scores de risque

Le frontend doit permettre :

• L'affichage graphique du score de risque (faible, modéré, élevé) ;

• L'affichage détaillé des variables influentes (features importantes) ;

• La consultation de l'historique des prédictions d'un client.

6. Génération de rapports

Le système doit pouvoir générer :

• Des rapports de prédiction exportables (PDF ou CSV) ;

• Des synthèses globales par période (scores moyens, taux de défaut prédits) ;

• Des rapports par analyste ou par agence.

7. Administration des modèles (versioning, réentraînement)

L'administrateur doit pouvoir :

• Importer une nouvelle version du modèle ML ;

• Déclencher un réentraînement sur de nouvelles données ;

• Visualiser les métriques de performance du modèle (accuracy, F1-score, etc.).

8. Système de logs et historique des prédictions

Le backend doit assurer un suivi complet :

• Logs des actions utilisateurs ;

• Historique des prédictions ;

• Identification de l'analyste ayant exécuté la prédiction ;

• Gestion des dates et heures de chaque action.

B. Besoins non fonctionnels

1. Performance et rapidité de l'API

• Les prédictions doivent s'exécuter en quelques millisecondes.

• Les endpoints API doivent répondre rapidement malgré une charge importante.

2. Scalabilité

L'application doit pouvoir être déployée dans un environnement évolutif, permettant :

• L'augmentation du nombre d'utilisateurs simultanés ;

• L'ajout de nouveaux modules sans affecter les performances.

3. Sécurité

• Chiffrement des communications (HTTPS, SSL) ;

• Hachage des mots de passe ;

• Protection contre les attaques (CSRF, brute-force, injections).

4. Confidentialité des données financières

Les informations des clients et leurs historiques de crédit doivent être protégées selon les normes financières et bancaires, avec accès restreint selon les rôles.

5. Disponibilité

L'application doit garantir une disponibilité élevée afin de répondre aux besoins quotidiens des IMF.

6. Ergonomie de l'interface utilisateur

L'interface React doit être :

• Simple ;

• Intuitive ;

• Accessible aux utilisateurs non techniquement formés ;

• Responsive (mobiles et ordinateurs).

7. Administration des réseaux et serveurs

Le déploiement doit tenir compte :

• De la configuration réseau ;

• Du paramétrage serveur ;

• De la protection des ports et services ouverts.

VI. Planification du projet

La planification du projet repose sur une méthode Agile avec une division en sprints. Cette approche permet une évolution progressive, des ajustements fréquents, et une meilleure maîtrise de la complexité.

Tableau : Planning prévisionnel

Phase	Durée	Activités principales
Étude et analyse	2 semaines	Recueil des besoins, étude de l'existant
Conception	3 semaines	UML, conception ML, architecture
Développement backend	4 semaines	API FastAPI, modèle ML, BD

Développement frontend	3 semaines	Interfaces React, intégration API
Tests	2 semaines	Tests unitaires, API, performance, ML
Déploiement	1 semaine	Serveur, sécurité, conteneurisation
Documentation	1 semaine	Manuel utilisateur, documentation technique

VII. Conclusion

L'analyse des besoins et la planification du projet ont permis de définir clairement les objectifs, les enjeux et les fonctionnalités essentielles de l'application de prédiction du risque de défaut. Le recours à une architecture moderne basée sur FastAPI et React.js, associé à un modèle de Machine Learning performant, constitue une solution adaptée aux défis rencontrés par les IMF camerounaises. La planification par sprints garantit une construction progressive, structurée et contrôlée, permettant d'obtenir une application fiable, sécurisée et simple à utiliser.

CHAPITRE II : ANALYSE ET CONCEPTION

I. Introduction

Ce chapitre présente la méthodologie adoptée pour la conception de l'application de prédiction du risque de défaut destinée aux institutions de microfinance camerounaises. Il met en lumière le modèle de développement utilisé, les différents diagrammes UML produits, ainsi que l'architecture globale de la solution. La conception inclut également le modèle de Machine Learning et les aspects de sécurité indispensables dans le secteur financier.

II. Modèle du cycle de vie en V

1. Justification du choix du modèle

Le modèle en V a été retenu car il permet une organisation rigoureuse du processus de développement. Ses avantages principaux sont :

• Une approche structurée et séquentielle ;

• Une correspondance claire entre les phases de spécification et les phases de test ;

• Une meilleure maîtrise des risques ;

• Une traçabilité complète entre les besoins et les implémentations.

Ce cadre méthodologique est particulièrement adapté pour un projet intégrant un modèle Machine Learning et nécessitant une validation stricte, notamment dans les applications financières.

2. Phases du modèle

Le modèle en V comporte deux grandes branches :

A. Branche descendante (Conception et spécifications)

• Analyse des besoins fonctionnels ;

• Analyse des besoins non fonctionnels ;

• Conception globale du système ;

• Conception détaillée (UML, architecture, modèle ML).

B. Branche montante (Tests et validation)

• Tests unitaires du code backend ;

• Tests d'intégration API ;

• Tests fonctionnels de l'application complète ;

• Tests de performance et validation du modèle ML.

3. Correspondance avec le développement de l'application

Phase du modèle en V	Application à notre projet
Analyse	Étude IMF, définition des besoins
Spécifications	Fonctions API, gestion utilisateurs, prédiction
Conception	UML, architecture backend, BD, pipeline ML
Implémentation	FastAPI, modèle ML, frontend React
Tests unitaires	Fonctions API, modèle ML
Tests d'intégration	API + frontend
Validation	Tests complets de l'application
Déploiement	Mise en production, documentation

III. Modélisation conceptuelle

La conception UML constitue une base solide avant le développement car elle permet de structurer clairement les éléments principaux de l'application :

• Les interactions entre utilisateurs et système ;

• Le fonctionnement logique interne ;

• Les données manipulées ;

• Les flux d'information.

1. Diagramme de cas d'utilisation

Interactions utilisateur avec le système

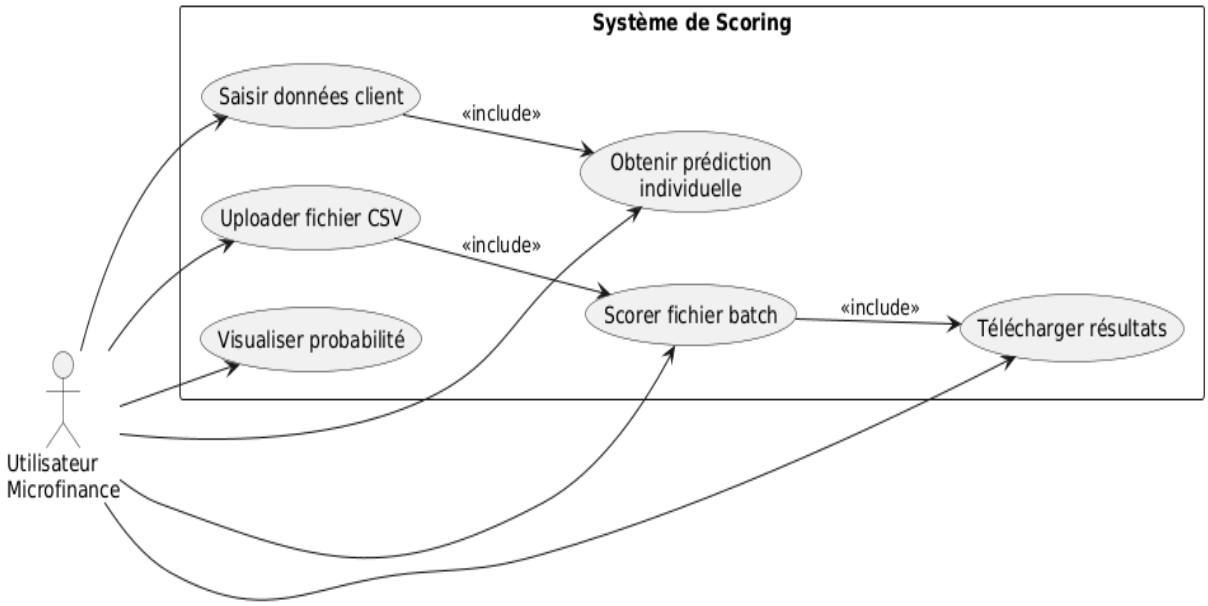

Figure 1 : Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation identifie les différentes interactions possibles entre les utilisateurs du système et les fonctionnalités offertes par l'application. L'acteur principal, représenté sous l'appellation générique "Utilisateur Microfinance", englobe en réalité plusieurs profils d'utilisateurs ayant des besoins distincts : l'agent de crédit qui évalue les demandes individuelles, le responsable des risques qui analyse le portefeuille global, l'auditeur qui vérifie la cohérence des décisions, et le directeur commercial qui segmente les prospects pour les campagnes marketing.

Le cas d'utilisation "Saisir données client" représente l'action fondamentale de renseigner les quinze variables requises par le modèle via le formulaire interactif de l'interface Streamlit. Cette saisie bénéficie d'une validation en temps réel qui guide l'utilisateur en signalant immédiatement les erreurs de format ou les valeurs hors limites, améliorant ainsi la qualité des données et l'expérience utilisateur.

Le cas d'utilisation "Obtenir prédiction individuelle" découle naturellement du précédent et est lié à lui par une relation d'inclusion (include). Une fois les données correctement saisies et validées, l'utilisateur déclenche le processus de scoring en cliquant sur le bouton de soumission. Le système traite la requête en moins de cent millisecondes et affiche le résultat sous forme d'une prédiction binaire accompagnée de la probabilité de défaut calculée. Cette rapidité d'exécution

permet d'intégrer le scoring dans le flux normal de traitement des demandes de crédit sans ralentissement perceptible.

Le cas d'utilisation "Uploader fichier CSV" permet aux utilisateurs de traiter par lot un grand nombre de dossiers simultanément. L'utilisateur sélectionne un fichier CSV structuré depuis son système de fichiers local. Le système affiche immédiatement un aperçu des premières lignes, permettant une vérification visuelle rapide de la conformité du format avant le lancement du traitement complet.

Le cas d'utilisation "Scorer fichier batch" représente le traitement massif proprement dit et inclut nécessairement le cas précédent d'upload du fichier. Ce traitement vectorisé peut gérer efficacement plusieurs centaines de dossiers en quelques secondes, démultipliant ainsi la productivité des équipes par rapport à un traitement manuel ligne par ligne. Cette fonctionnalité s'avère particulièrement utile pour les revues trimestrielles de portefeuille ou les campagnes de scoring préventif.

Le cas d'utilisation "Télécharger résultats" conclut logiquement le processus de scoring batch par la récupération du fichier CSV enrichi. Ce fichier contient toutes les colonnes d'origine augmentées de deux nouvelles colonnes : la probabilité de défaut et la prédiction binaire pour chaque ligne. L'utilisateur peut alors importer ce fichier dans d'autres outils d'analyse (Excel, Power BI, Tableau) pour des exploitations complémentaires.

Le cas d'utilisation "Visualiser probabilité" permet aux utilisateurs de consulter le détail du score calculé et de comprendre le niveau de risque associé à chaque client. L'interface présente cette information de manière claire avec un code couleur visuel (vert pour les faibles risques, rouge pour les risques élevés) facilitant la prise de décision rapide.

2. Diagramme de classes

Structure des données et composants principaux

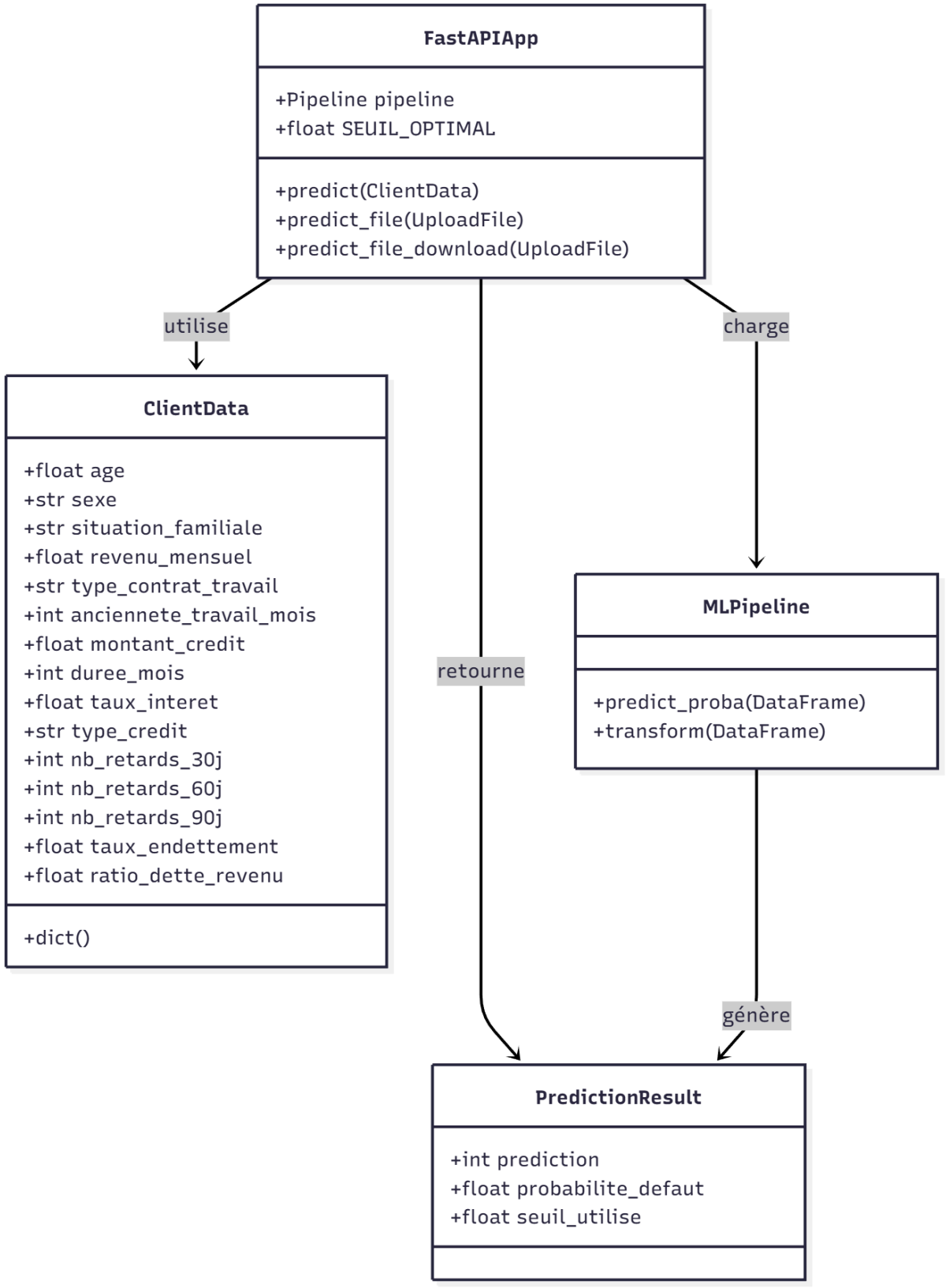

Figure 2 : Diagramme de classes

Le diagramme de classes présente la structure orientée objet de l'application en mettant en évidence les principales entités et leurs relations. La classe ClientData, héritant de BaseModel de Pydantic, constitue le cœur du système de validation des données. Elle définit quinze attributs typés correspondant aux variables d'entrée du modèle : les variables démographiques (age, sexe, situation_familiale), les variables professionnelles (revenu_mensuel, type_contrat_travail, anciennete_travail_mois), les caractéristiques du crédit demandé (montant_credit, duree_mois, taux_interet, type_credit), et les indicateurs de risque historiques (nb_retards_30j, nb_retards_60j, nb_retards_90j, taux_endettement, ratio_dette_revenu). La validation automatique de Pydantic s'assure que chaque attribut respecte son type déclaré et ses contraintes éventuelles.

La classe FastAPIApp encapsule la logique métier de l'application. Ses attributs principaux sont le pipeline Machine Learning chargé depuis le fichier .pkl et le seuil optimal stocké sous forme de constante. Cette classe expose trois méthodes publiques correspondant aux endpoints de l'API : predict qui prend en paramètre un objet ClientData et retourne une prédiction individuelle, predict_file qui accepte un objet UploadFile et retourne les résultats au format JSON, et predict_file_download qui traite également un fichier mais retourne directement un CSV téléchargeable.

La classe MLPipeline représente l'abstraction du pipeline scikit-learn. Bien que techniquement il s'agisse d'un objet Pipeline de scikit-learn et non d'une classe définie dans l'application, sa représentation dans le diagramme met en évidence les deux méthodes essentielles utilisées : predict_proba qui calcule les probabilités de classification pour un DataFrame d'entrée, et transform qui applique uniquement les transformations de preprocessing sans effectuer de prédiction.

La classe PredictionResult structure la réponse retournée par l'API. Elle contient trois attributs : prediction (entier valant 0 ou 1), probabilite_defaut (flottant entre 0.0 et 1.0), et seuil_utilise (flottant indiquant le seuil appliqué). Les relations entre ces classes illustrent le flux de données : FastAPIApp utilise ClientData pour valider les entrées, charge et interroge MLPipeline pour obtenir les prédictions, et retourne des instances de PredictionResult aux clients.

3. Diagrammes de séquence

➢ Prédiction Individuelle

Flux de traitement d'une requête de scoring individuel

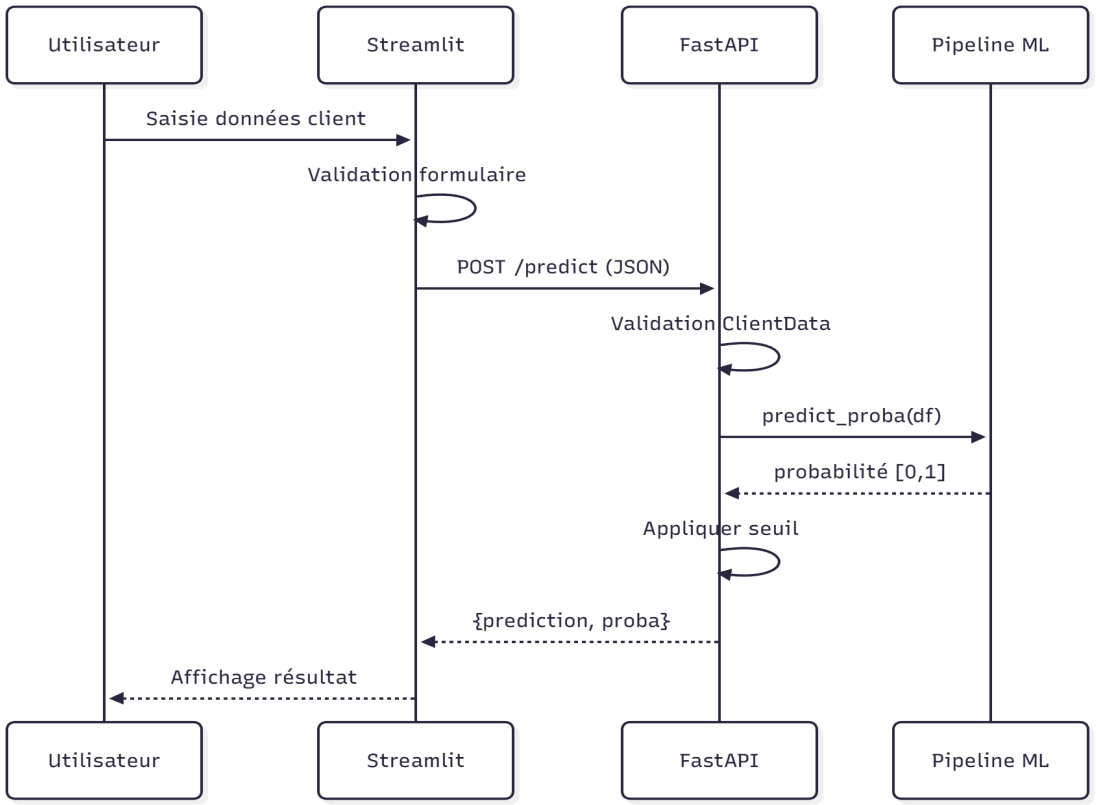

Figure 3: Diagrammes de séquence-Prédiction individuelle

Le diagramme de séquence pour la prédiction individuelle illustre de manière chronologique les interactions entre les quatre acteurs principaux du système lors du traitement d'une demande de scoring unitaire. Le processus débute par la saisie des quinze variables requises par l'utilisateur dans le formulaire Streamlit. Cette saisie est immédiatement suivie d'une phase de validation côté client qui vérifie la conformité des types de données et le respect des plages de valeurs autorisées pour chaque champ.

Une fois la validation locale réussie, l'interface Streamlit construit une requête HTTP POST contenant les données du client sous format JSON et l'envoie à l'endpoint /predict de l'API FastAPI. À la réception de cette requête, FastAPI active automatiquement le mécanisme de validation Pydantic qui instancie un objet ClientData et vérifie rigoureusement la conformité de toutes les

données reçues. En cas d'échec de validation, une réponse HTTP 422 (Unprocessable Entity) est retournée avec le détail des erreurs détectées.

Lorsque la validation réussit, les données validées sont transformées en DataFrame Pandas, format requis par le pipeline scikit-learn. L'appel à la méthode predict_proba du pipeline déclenche l'exécution séquentielle des transformations de preprocessing puis l'inférence du modèle de classification. Le pipeline retourne un tableau de probabilités dont seule la probabilité de la classe positive (défaut) est extraite. Cette probabilité est ensuite comparée au seuil optimal de 0.5 pour déterminer la prédiction binaire finale. La réponse JSON structurée, contenant la prédiction, la probabilité et le seuil utilisé, est renvoyée à Streamlit qui l'affiche à l'utilisateur avec un code couleur approprié. Le temps de traitement total de cette séquence est inférieur à 100 millisecondes, garantissant une expérience utilisateur fluide.

- **Scoring Batch CSV**

Traitement d'un fichier CSV complet

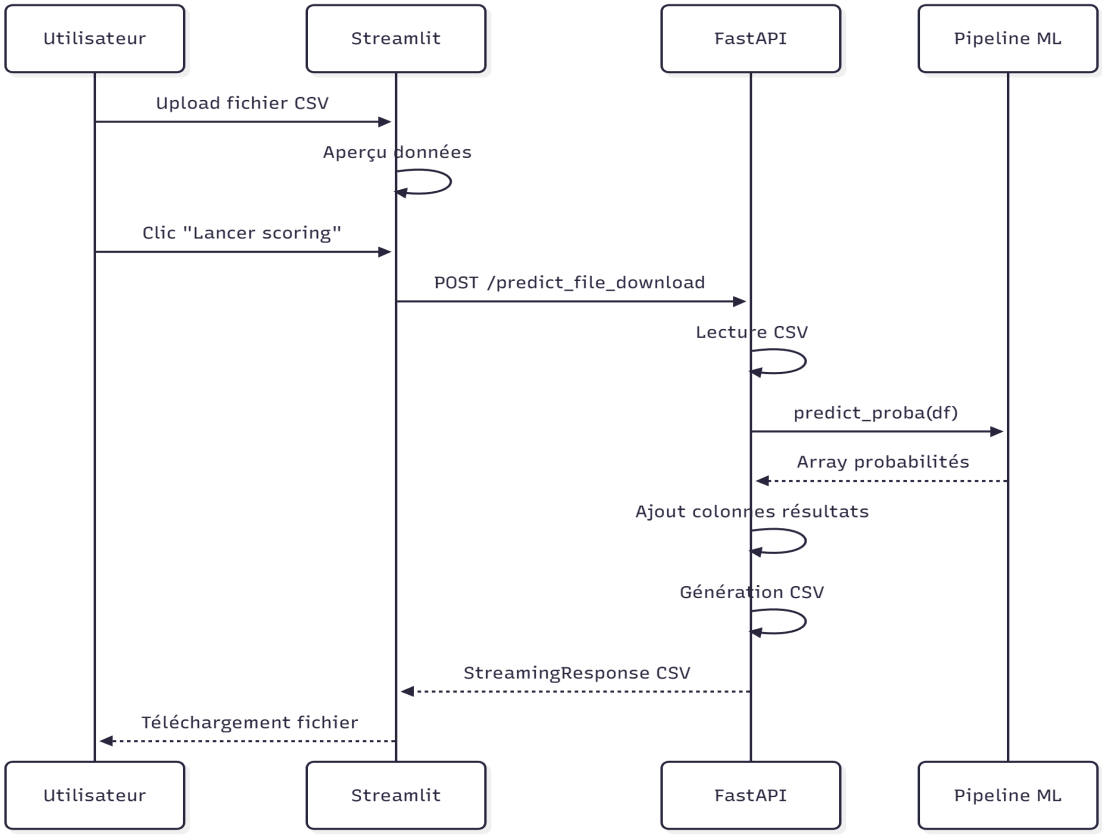

Figure 4: Diagrammes de séquence-Scoring batch CSV

Le diagramme de séquence pour le scoring batch CSV détaille le processus de traitement massif de dossiers clients à partir d'un fichier structuré. Le flux commence par le téléchargement d'un fichier CSV par l'utilisateur via l'interface Streamlit. Avant toute transmission au backend, Streamlit affiche un aperçu des cinq premières lignes du fichier, permettant à l'utilisateur de vérifier visuellement la présence des colonnes requises et la qualité générale des données.

Sur confirmation de l'utilisateur par le clic du bouton "Lancer le scoring CSV", Streamlit envoie le fichier complet à l'API FastAPI via une requête HTTP POST de type multipart/form-data vers l'endpoint /predict_file_download. La réception du fichier côté API déclenche immédiatement sa lecture en mémoire par la bibliothèque Pandas qui construit un DataFrame à partir du contenu CSV. Cette opération inclut une validation implicite de la structure du fichier puisque Pandas génère une exception en cas de format invalide.

Le DataFrame ainsi créé est transmis directement au pipeline Machine Learning qui effectue le scoring de manière vectorisée, c'est-à-dire en traitant simultanément toutes les lignes du fichier. Cette approche vectorisée, rendue possible par les optimisations internes de NumPy et scikit-learn, permet d'atteindre des performances remarquables avec un traitement de cinq cents lignes en approximativement deux secondes. Le pipeline retourne un tableau de probabilités de même dimension que le nombre de lignes du fichier d'entrée.

L'API enrichit ensuite le DataFrame original en y ajoutant deux nouvelles colonnes : probabilite_defaut contenant les valeurs de probabilité calculées, et prediction_defaut contenant les classifications binaires résultant de l'application du seuil. Le DataFrame enrichi est converti en format CSV en mémoire, puis retourné au client via une StreamingResponse avec un header Content-Disposition: attachment qui déclenche automatiquement le téléchargement du fichier par le navigateur. Le fichier téléchargé porte le nom scoring_[nom_fichier_original].csv et peut être immédiatement exploité pour des analyses complémentaires.

4. Diagramme d’activité - Processus de Scoring

Étapes détaillées du processus de scoring

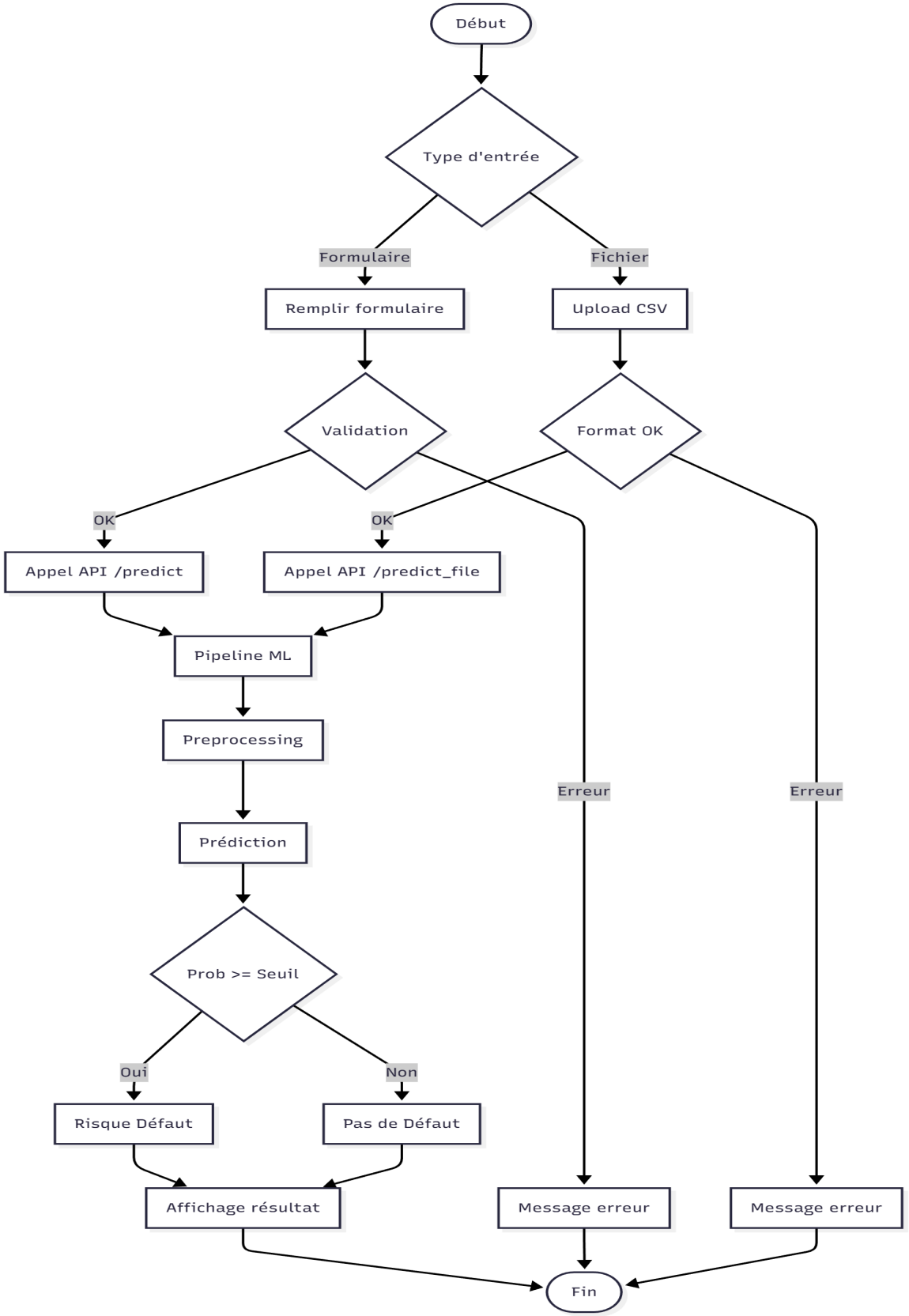

Figure 5: Diagramme d'activité - Processus de Scoring

Le diagramme d'activité décrit de manière procédurale le déroulement complet d'un processus de scoring, depuis l'initiation par l'utilisateur jusqu'à l'affichage du résultat final. Ce diagramme met en évidence les points de décision et les chemins alternatifs possibles en fonction des choix utilisateur et des conditions de validation.

Le processus débute par un nœud initial unique qui se ramifie immédiatement sur une première décision concernant le type d'entrée choisi par l'utilisateur. Si l'utilisateur opte pour une saisie via formulaire, le flux entre dans la branche de traitement individuel. Cette branche commence par l'activité de remplissage du formulaire où l'utilisateur saisit manuellement les quinze variables requises. Une fois le formulaire complet, une validation automatique des données est déclenchée. Cette validation vérifie la conformité des types, le respect des plages de valeurs et la cohérence globale des données saisies. Si cette validation échoue, le système affiche un message d'erreur détaillé et le flux se termine, permettant à l'utilisateur de corriger les erreurs avant de soumettre à nouveau. Si la validation réussit, une requête HTTP POST est envoyée à l'endpoint /predict de l'API FastAPI.

Dans le cas où l'utilisateur choisit la branche fichier, le processus commence par l'activité d'upload du fichier CSV. Le système charge le fichier en mémoire et effectue immédiatement une vérification du format. Cette vérification s'assure que le fichier est bien un CSV valide, qu'il contient les colonnes requises avec les noms exacts attendus, et qu'il ne contient pas d'anomalies structurelles majeures. Un échec de cette vérification entraîne l'affichage d'un message d'erreur et la terminaison du flux. Une vérification réussie déclenche l'envoi d'une requête multipart à l'endpoint /predict_file de l'API.

Les deux branches de traitement (individuel et fichier) convergent vers l'activité commune du pipeline Machine Learning. Cette convergence illustre le fait que, quelle que soit l'origine des données, le traitement ultérieur est identique. Le pipeline exécute d'abord l'étape de preprocessing qui transforme les données brutes en features normalisées et encodées. Cette transformation applique de manière cohérente les mêmes opérations (encodage, normalisation, scaling) qui avaient été utilisées lors de l'entraînement du modèle, garantissant ainsi que le modèle reçoit des données dans le format attendu.

L'activité de prédiction suit immédiatement le preprocessing. Le modèle de classification, préalablement entraîné et chargé en mémoire, calcule pour chaque observation la probabilité d'appartenance à la classe positive (défaut). Cette probabilité, un nombre réel entre 0 et 1,

représente la confiance du modèle dans sa prédiction. Plus cette valeur est élevée, plus le modèle estime que le client présente un risque de défaut.

Le flux atteint ensuite un second point de décision crucial : la comparaison de la probabilité calculée avec le seuil de décision fixé à 0.5. Si la probabilité est supérieure ou égale au seuil, le flux emprunte la branche "défaut" et assigne la valeur 1 à la variable prediction, indiquant un risque de défaut détecté. Si la probabilité est strictement inférieure au seuil, le flux suit la branche "pas de défaut" et assigne la valeur 0 à prediction, signifiant que le client est considéré comme présentant un risque acceptable.

Les deux branches de cette décision convergent vers l'activité finale d'affichage du résultat. Cette activité présente à l'utilisateur les informations essentielles : la prédiction binaire (défaut ou pas de défaut), la probabilité exacte calculée (affichée avec deux décimales pour une précision suffisante sans surcharge informationnelle), et le seuil utilisé pour la décision. L'interface utilise un code couleur pour faciliter l'interprétation immédiate : rouge pour les clients classés en défaut, vert pour les clients considérés comme sûrs. Cette visualisation permet une prise de décision rapide par les agents de crédit.

Le diagramme comporte également des annotations importantes concernant les performances. Le temps de traitement total pour une prédiction individuelle est garanti inférieur à cent millisecondes, depuis la soumission de la requête jusqu'à l'affichage du résultat. Pour les traitements batch, les performances sont exprimées en débit : environ cinq cents lignes peuvent être scorées en deux secondes, soit un débit de deux cent cinquante prédictions par seconde. Ces performances exceptionnelles sont rendues possibles par l'utilisation de calculs vectorisés au niveau du pipeline scikit-learn et par l'architecture asynchrone de FastAPI qui évite les blocages lors des opérations d'entrée-sortie.

Le flux se termine par un nœud final unique, quel que soit le chemin emprunté, symbolisant la complétion réussie du processus de scoring. En cas d'erreur à n'importe quelle étape du processus, des nœuds de fin alternatifs sont atteints, permettant au système de gérer proprement les situations exceptionnelles sans corruption de l'état applicatif.

5. Diagramme de Déploiement

Architecture d'infrastructure et composants

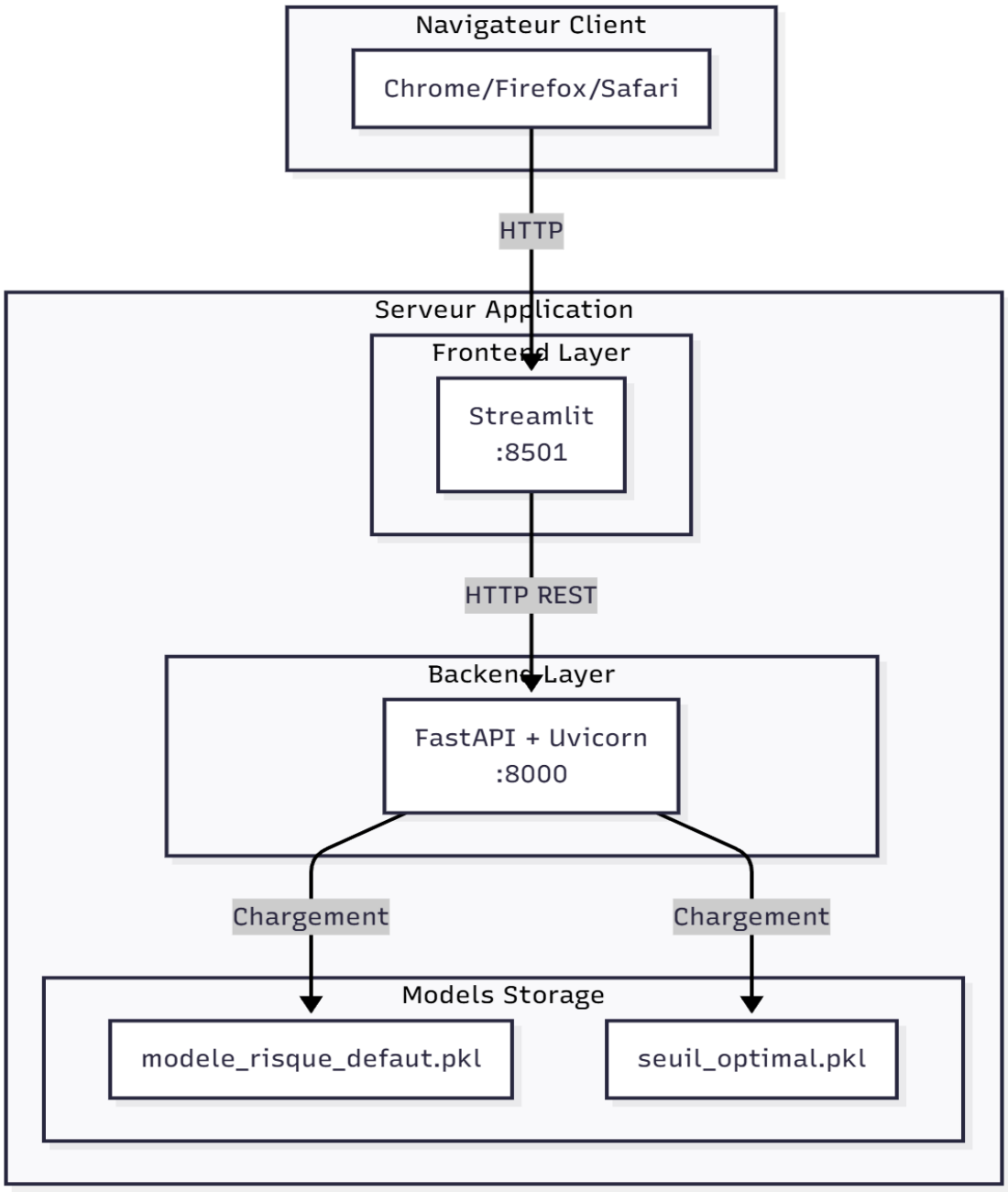

Figure 6: Diagramme de Déploiement

Le diagramme de déploiement fournit une représentation de l'infrastructure matérielle et logicielle nécessaire à l'exécution de l'application. Le serveur d'application constitue le nœud central hébergeant l'ensemble des composants logiciels. Ce serveur est organisé en trois packages distincts reflétant l'architecture logicielle de l'application.

Le package Frontend Layer contient le composant Streamlit configuré pour écouter sur le port 8501. Ce composant sert les pages HTML, gère les interactions utilisateur et maintient l'état de la session pour chaque utilisateur connecté. Le choix du port 8501 correspond à la configuration par défaut de Streamlit, facilitant ainsi le déploiement et la maintenance.

Le package Backend Layer héberge l'application FastAPI couplée au serveur ASGI Uvicorn, ensemble configuré pour écouter sur le port 8000. Uvicorn assure la gestion efficace des connexions HTTP asynchrones et la distribution des requêtes vers les handlers FastAPI appropriés. Ce composant backend constitue le véritable moteur de l'application, orchestrant les appels au modèle ML et assurant la cohérence des réponses.

Le package Models Storage regroupe les artefacts persistants de Machine Learning stockés sous forme de fichiers. Le fichier modele_risque_defaut.pkl contient le pipeline complet incluant les transformateurs de preprocessing et le modèle de classification entraîné. Le fichier seuil_optimal.pkl stocke le seuil de décision optimal déterminé lors de la phase d'optimisation du modèle. Ces fichiers sont chargés en mémoire au démarrage de l'application selon le pattern Singleton, garantissant des performances optimales en évitant les rechargements répétés.

Le nœud Navigateur Client représente le poste de travail de l'utilisateur final équipé d'un navigateur web moderne (Chrome, Firefox, Safari). Les connexions entre les nœuds illustrent les protocoles de communication : le navigateur communique avec Streamlit via HTTP standard, Streamlit interroge FastAPI via des requêtes HTTP REST, et FastAPI charge les modèles depuis le système de fichiers via les primitives d'entrée-sortie de Joblib. Pour un environnement de production, il est fortement recommandé d'activer HTTPS et de placer un reverse proxy Nginx devant les composants web pour gérer le chiffrement SSL/TLS, le load balancing et la mise en cache des ressources statiques.

6. Diagramme de Flux de Données

Traitement des données du début à la fin

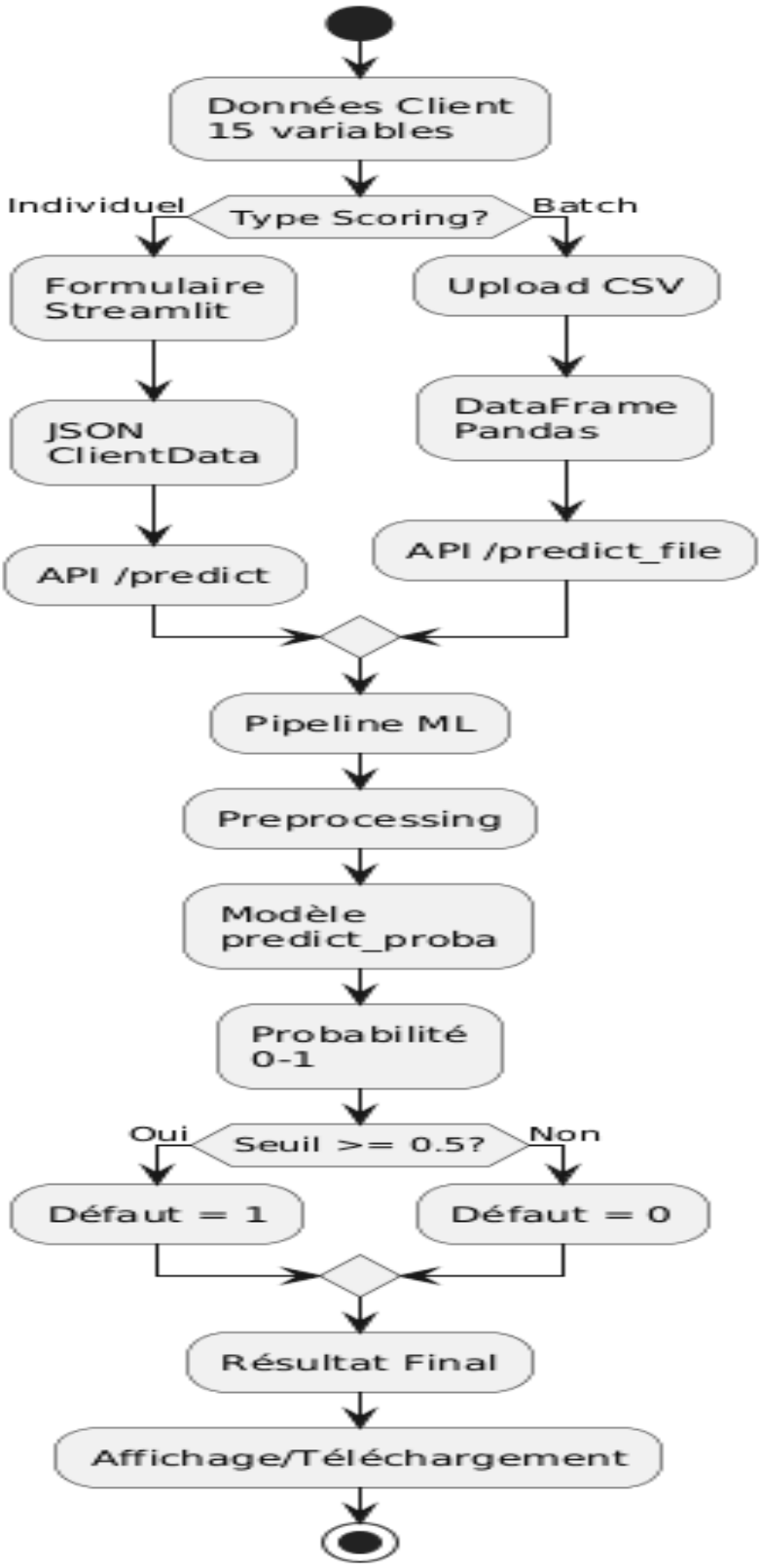

Figure 7: Diagramme de Flux de Données

Le diagramme de flux de données représente de manière linéaire le cheminement complet des informations depuis leur saisie initiale jusqu'à la production du résultat final. Ce diagramme met particulièrement en évidence les transformations successives appliquées aux données à chaque étape du traitement.

Le flux débute par l'acquisition des données clients, qui peuvent provenir soit d'une saisie manuelle via le formulaire Streamlit pour les traitements individuels, soit d'un fichier CSV pour les traitements batch. Ces données brutes comprennent les quinze variables requises par le modèle, chacune devant respecter des contraintes de type et de plage de valeurs spécifiques.

La première transformation importante est la phase de validation réalisée par Pydantic. Cette validation vérifie systématiquement que chaque variable possède le type attendu (float pour les valeurs numériques continues, int pour les entiers, str pour les chaînes de caractères), que les valeurs numériques se situent dans les plages autorisées (par exemple, l'âge doit être compris entre 18 et 80 ans), et que les variables catégorielles appartiennent bien aux ensembles de valeurs prédéfinis. Toute violation de ces contraintes entraîne le rejet immédiat de la requête avec un message d'erreur détaillé.

Les données validées sont ensuite converties en DataFrame Pandas, structure de données tabulaire optimisée pour les opérations vectorisées. Cette conversion est nécessaire car le pipeline scikit-learn attend ses entrées sous ce format standardisé. Le DataFrame créé possède une ligne par observation (un client en mode individuel, plusieurs clients en mode batch) et une colonne par variable.

L'étape de preprocessing constitue une phase critique de transformation des données. Les variables catégorielles (sexe, situation_familiale, type_contrat_travail, type_credit) sont encodées numériquement via OneHotEncoder, créant des variables binaires pour chaque modalité possible. Les variables numériques sont normalisées via StandardScaler ou RobustScaler pour ramener toutes les échelles à des ordres de grandeur comparables, évitant ainsi que certaines variables ne dominent artificiellement les prédictions du modèle. Des opérations de feature engineering peuvent également être appliquées à ce stade pour créer de nouvelles variables dérivées.

Le DataFrame preprocessé est transmis au modèle de classification qui calcule, pour chaque observation, la probabilité d'appartenance à la classe positive (défaut de paiement). Cette probabilité, exprimée comme un nombre réel entre 0.0 et 1.0, quantifie la confiance du modèle

dans sa prédiction. Une probabilité proche de 0 indique une forte confiance que le client ne fera pas défaut, tandis qu'une probabilité proche de 1 suggère un risque élevé de défaut.

La probabilité calculée est ensuite comparée au seuil de décision fixé à 0.5 par défaut. Cette comparaison produit la classification binaire finale : si la probabilité est supérieure ou égale au seuil, le client est classé comme risque de défaut (prediction = 1), sinon il est classé comme non risqué (prediction = 0). Ce seuil peut être ajusté en fonction de la politique de risque de l'institution et du coût relatif des faux positifs et faux négatifs.

Le résultat final agrège les informations essentielles : la prédiction binaire, la probabilité exacte calculée, et le seuil utilisé pour la décision. En mode batch, ces informations sont ajoutées comme nouvelles colonnes au fichier CSV original, permettant ainsi de conserver l'ensemble du contexte client tout en enrichissant les données avec les scores de risque calculés.

7. Diagramme d'Architecture Système

Vue d'ensemble de l'architecture applicative

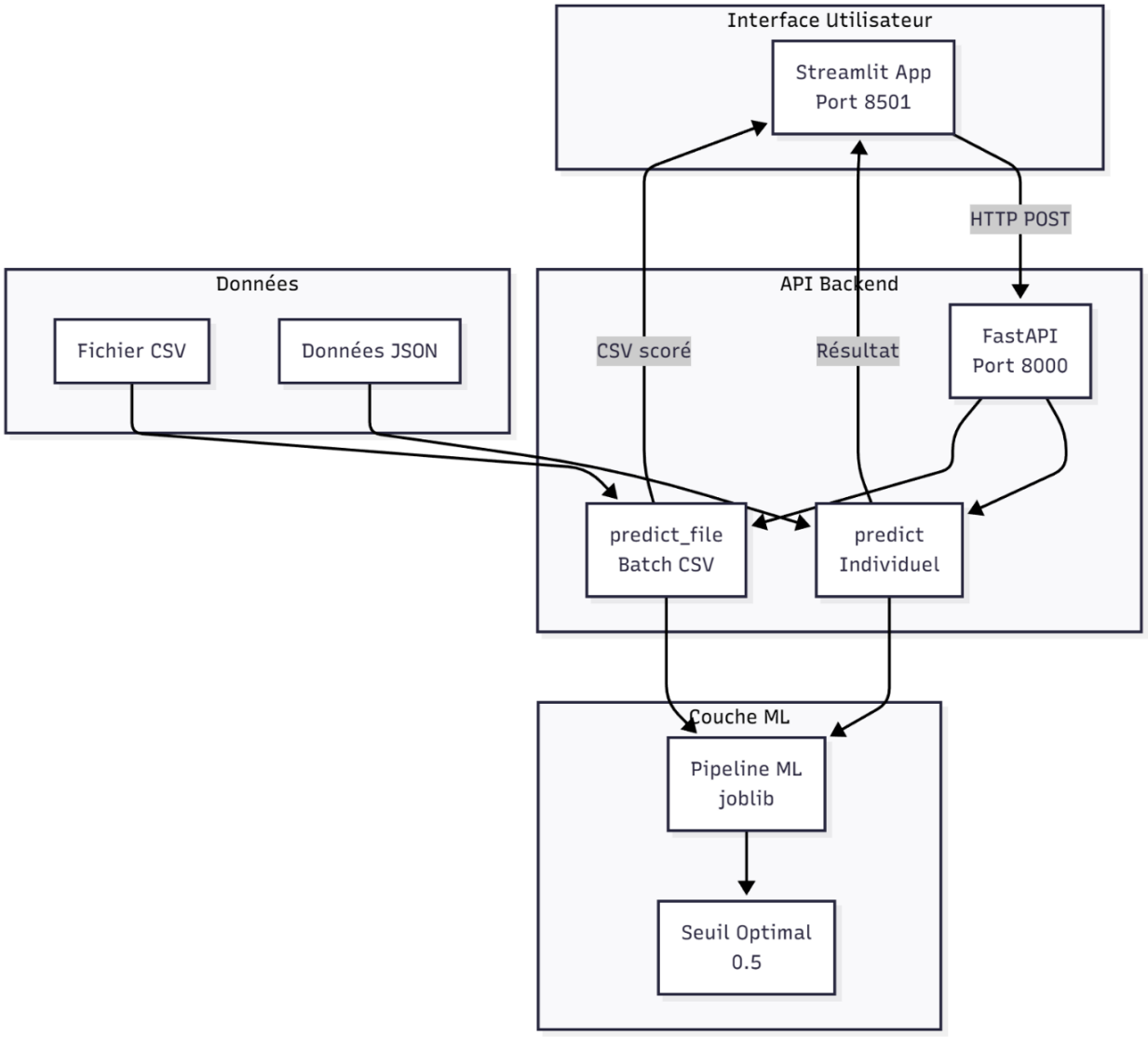

Figure 8: Diagramme d'Architecture Système

Le diagramme d'architecture système présente une vue d'ensemble de l'application selon le modèle d'architecture à trois niveaux (3-tier architecture). Cette représentation met en évidence la séparation claire des responsabilités entre les différentes couches de l'application. Au niveau de la couche présentation, l'interface utilisateur Streamlit, déployée sur le port 8501, constitue le point d'entrée principal pour les utilisateurs finaux. Cette interface permet la saisie manuelle des données clients via un formulaire interactif ainsi que le téléchargement de fichiers CSV pour les traitements en masse.

La couche logique métier est incarnée par l'API REST FastAPI, accessible sur le port 8000. Cette API expose quatre endpoints principaux permettant respectivement l'affichage de la page

d'accueil, le scoring individuel via l'endpoint /predict, et deux variantes de scoring batch via /predict_file et /predict_file_download. L'architecture asynchrone de FastAPI garantit des performances optimales même sous forte charge.

La couche données et modèle regroupe le pipeline Machine Learning sérialisé sous forme de fichiers .pkl via la bibliothèque Joblib. Ce pipeline comprend les étapes de preprocessing nécessaires à la transformation des données brutes, le modèle de classification entraîné, ainsi que le seuil de décision optimal fixé à 0.5. Les flux de communication entre ces trois couches suivent un modèle requête-réponse synchrone, garantissant la cohérence des données et la traçabilité des opérations.

8. Conception du modèle Machine Learning

A. Choix du modèle

Pour les problèmes de défaut de crédit, les modèles les plus adaptés sont :

• Régression Logistique

• Random Forest

• XGBoost

Random Forest est recommandé pour sa robustesse.

B. Prétraitement

• Encodage des variables catégorielles

• Normalisation des données

• Gestion des valeurs manquantes

• Sélection des features

C. Entraînement, évaluation, métriques

• Découpage train/test

• Métriques utilisées : Accuracy, Recall, Precision, F1-score

• Optimisation par GridSearch ou RandomSearch

9. Conception de la sécurité

A. Gestion des tokens / sessions

• JWT pour authentification

• Expiration courte pour sécurité

• Refresh token pour prolongation

B. Chiffrement des données

• Hachage des mots de passe (bcrypt)

• Chiffrement communication (SSL)

C. Sécurisation de l'API

• Rate limiting

• CORS configuré

• Validation stricte des schémas Pydantic

D. Gestion des accès administrateur

• Rôles et permissions

• Accès restreint aux opérations sensibles

E. Normes et bonnes pratiques

• OWASP

• Règles minimales d'accès (principe du moindre privilège)

IV. Conclusion

Ce chapitre a présenté la conception complète de l'application : choix méthodologique, UML, architecture, modèle ML et sécurité. Cette phase constitue la fondation technique qui guidera efficacement l'implémentation détaillée dans le chapitre suivant.

CHAPITRE III : IMPLÉMENTATION DE LA SOLUTION

I. Outils technologiques et langages utilisés

1. Logiciels

• **Visual Studio Code** : environnement de développement pour le backend et le frontend.

• **Postman** : test des API REST.

• **Docker** : conteneurisation et déploiement.

• **Git / GitHub** : gestion de versions et collaboration.

2. Framework

• **Backend : FastAPI**

• Fournit les endpoints REST pour la gestion des utilisateurs, l'import des données, l'exécution du modèle ML et la génération des rapports.

• Intègre la documentation automatique via Swagger / OpenAPI.

• **Frontend : React.js**

• Permet de créer des interfaces interactives pour la saisie de données et la visualisation des prédictions.

• Utilisation de **TailwindCSS** et **Shadcn UI** pour le style et l'ergonomie.

3. Langages

• **Python 3.10** : implémentation du backend et du modèle ML.

• **JavaScript / TypeScript** : développement frontend React.js.

• **SQL** : gestion des requêtes et structures de la base de données relationnelle.

4. Bibliothèques

• **Pandas, NumPy** : manipulation et nettoyage des données.

• **Scikit-learn, XGBoost** : modèles de Machine Learning pour la prédiction du risque de défaut.

• **bcrypt, JWT** : sécurité et gestion des authentifications.

• **FastAPI Pydantic** : validation des schémas et sécurité des données API.

II. Implémentation du backend

1. Structure du projet backend

```
backend/
├── app/
│   ├── api/            # Endpoints REST
│   ├── core/           # Configuration et sécurité
│   ├── models/         # Modèles SQLAlchemy
│   ├── ml/             # Modèle ML et preprocessing
│   ├── schemas/        # Pydantic pour validation
│   ├── services/       # Logique métier
│   └── main.py         # Point d'entrée FastAPI
├── requirements.txt
└── Dockerfile
```

2. Implémentation des endpoints

➢ **Authentification**

• POST /login : vérifie les identifiants, retourne un token JWT.

• POST /logout : invalide le token.

• **Gestion des utilisateurs**

• CRUD utilisateurs (ajout, modification, suppression).

• Gestion des rôles et permissions.

➢ **Gestion des clients et crédits**

• Import CSV / Excel.

• Saisie manuelle.

• Consultation et mise à jour des données.

➢ **Prédiction**

• POST /predict : envoie les données d'un client et reçoit le score de risque.

➢ **Historique et logs**

• Sauvegarde des prédictions et actions utilisateur.

3. Implémentation du modèle ML

• Chargement du modèle

```
import joblib
model = joblib.load("ml/model.pkl")
```

• Exécution

```
def predict_risk(data):
    preprocessed = preprocess(data)
    return model.predict_proba([preprocessed])[0][1]
```

• Réentraînement

• L'administrateur peut déclencher le réentraînement via un endpoint sécurisé.

4. Gestion des erreurs et logs

• Gestion centralisée des exceptions avec FastAPI.

• Logs enregistrés dans la base de données et fichiers journaux.

• Traçabilité complète des prédictions et accès utilisateur.

5. Documentation API

• **Swagger / OpenAPI** : documentation automatique, accessible via /docs.

• Permet de tester tous les endpoints directement dans le navigateur.

III. Base de données

1. Modèle logique et physique

Tables principales

• Utilisateur : id, nom, email, rôle, mot_de_passe.

• Client : id, nom, âge, revenu, statut_pro, garanties.

• Credit : id, montant, durée, taux, statut, id_client.

• HistoriquePrediction : id, id_client, id_utilisateur, score_risque, date_prediction.

• ModeleML : id, version, date_entraînement, métriques.

• LogSysteme : id, action, utilisateur, date.

2. Types de données stockées

Texte, numérique, booléen, date/heure, JSON pour certaines métadonnées.

3. Contraintes d'intégrité

• Clés primaires et étrangères pour relations entre tables.

• Contraintes d'unicité (email utilisateur, identifiant client).

• Non-null pour les champs obligatoires.

• Validation des types via Pydantic et base de données.

IV. Déploiement

1. Serveur utilisé

• Serveur cloud ou local (Ubuntu / Windows Server).

• Backend déployé avec Uvicorn/Gunicorn.

• Frontend React hébergé sur le même serveur ou séparément.

2. Configuration réseau

• Ports sécurisés (80 et 443 pour HTTP/HTTPS).

• Reverse proxy Nginx pour redirection et sécurité.

3. Sécurisation serveur

• **Firewall** configuré pour limiter les accès.

• **Certificats SSL** pour HTTPS.

• Sécurisation des endpoints sensibles avec JWT et permissions.

4. Conteneurisation

• **Docker** pour backend et base de données.

• **docker-compose** pour orchestrer les services.

• Facilite le déploiement et la scalabilité.

V. Conclusion

Le développement et le déploiement de l'application ont permis de transformer la conception théorique en une solution fonctionnelle :

• Backend FastAPI sécurisé et performant.

• Modèle ML opérationnel et intégrable.

• API REST documentée pour un accès facile.

• Base de données robuste assurant intégrité et historique.

• Frontend interactif et ergonomique.

• Déploiement sécurisé avec conteneurisation et surveillance.

Cette implémentation garantit ainsi une solution fiable et adaptée aux besoins des institutions de micro-finance camerounaises.

CHAPITRE IV : TESTS ET MAINTENANCE

I. Introduction

Le présent chapitre décrit les tests effectués sur l'application de prédiction du risque de défaut ainsi que les actions de maintenance mises en place pour assurer sa fiabilité, sa sécurité et son évolutivité. Ces activités sont essentielles pour garantir la qualité et la pérennité de la solution dans le contexte des institutions de microfinances camerounaises.

II. Tests effectués

1. Tests unitaires

Les tests unitaires ont pour objectif de vérifier le bon fonctionnement de chaque composant logiciel indépendamment :

• **Backend** : validation des fonctions de prédiction, endpoints REST, gestion des utilisateurs et des données clients.

• **Modèle ML** : vérification des fonctions de prétraitement, exécution de prédiction et calcul des métriques.

• **Validation Pydantic** : contrôle des données envoyées par le frontend.

Exemple de test unitaire Python pour le modèle ML :

```
def test_prediction():
    data = {"age": 35, "revenu": 150000, "historique_defaut": 0}
    score = predict_risk(data)
    assert 0 <= score <= 1
```

2. Tests fonctionnels

Ces tests vérifient que l'application répond correctement aux besoins exprimés dans le cahier des charges :

• Authentification et gestion des rôles.

• Importation et saisie des données clients.

• Exécution des prédictions et restitution correcte des scores.

• Génération des rapports et historique des prédictions.

• Navigation et interface utilisateur du frontend.

Exemple : vérification que la soumission d'un formulaire client déclenche bien une prédiction et sauvegarde l'historique associé.

3. Tests de performance

• Mesure du temps de réponse des endpoints (predict, login, import CSV).

• Test de charge pour vérifier la scalabilité du backend et de la base de données.

• Objectif : temps de prédiction inférieur à 1 seconde pour 1 client, et support simultané d'au moins 50 utilisateurs.

4. Tests de sécurité

• Vérification de l'authentification JWT et expiration des tokens.

• Test de vulnérabilités : injection SQL, XSS, CSRF.

• Protection des endpoints sensibles.

• Chiffrement des mots de passe et communication HTTPS.

5. Tests d'intégration API

• Vérification de l'interaction frontend-backend.

• Test complet du workflow : saisie client → prédiction → enregistrement → visualisation → génération rapport.

• Tests avec différents types de données (valeurs manquantes, formats erronés).

6. Tests du modèle ML

• **Métriques utilisées** : Accuracy, Recall, Precision, F1-score, AUC-ROC.

• Validation croisée pour mesurer la robustesse.

• Comparaison des performances avec différents modèles (Random Forest, XGBoost, Logistic Regression).

• Objectif : garantir que le modèle prédit correctement les défauts et minimise les faux négatifs.

III. Maintenance

1. Maintenance corrective

• Correction des bugs détectés lors des tests ou en production.

• Mise à jour du backend et du frontend pour résoudre des dysfonctionnements mineurs.

2. Maintenance évolutive

• Ajout de nouvelles fonctionnalités : nouveaux indicateurs de risque, filtres avancés dans le dashboard.

• Mise à jour des librairies et dépendances pour assurer la compatibilité et la sécurité.

• Amélioration continue du modèle ML avec de nouvelles données client.

3. Réentraînement du modèle

• Programmation régulière de réentraînement avec de nouvelles données pour maintenir la précision.

• Enregistrement des nouvelles versions du modèle avec journalisation des performances.

IV. Conclusion

Les tests réalisés ont permis de vérifier la fiabilité, la performance et la sécurité de l'application de prédiction du risque de défaut.La maintenance corrective et évolutive, ainsi que la documentation complète, garantissent la pérennité de la solution et sa capacité à évoluer avec les besoins des institutions de micro-finance.Cette approche assure une adoption efficace et sécurisée, tout en permettant aux décideurs de disposer d'un outil fiable pour la gestion du risque de crédit.

CHAPITRE V : RESULTAT

Le projet de conception d'une application de prédiction du risque de défaut pour les institutions de microfinance camerounaises avait pour objectif principal de fournir un outil fiable, rapide et sécurisé pour améliorer la prise de décision dans la gestion du risque de crédit.

I. Présentation des interfaces

Au terme de ce projet, voici ce qui nous a été donné de réaliser :

1. Interface de prédiction – informations à remplir

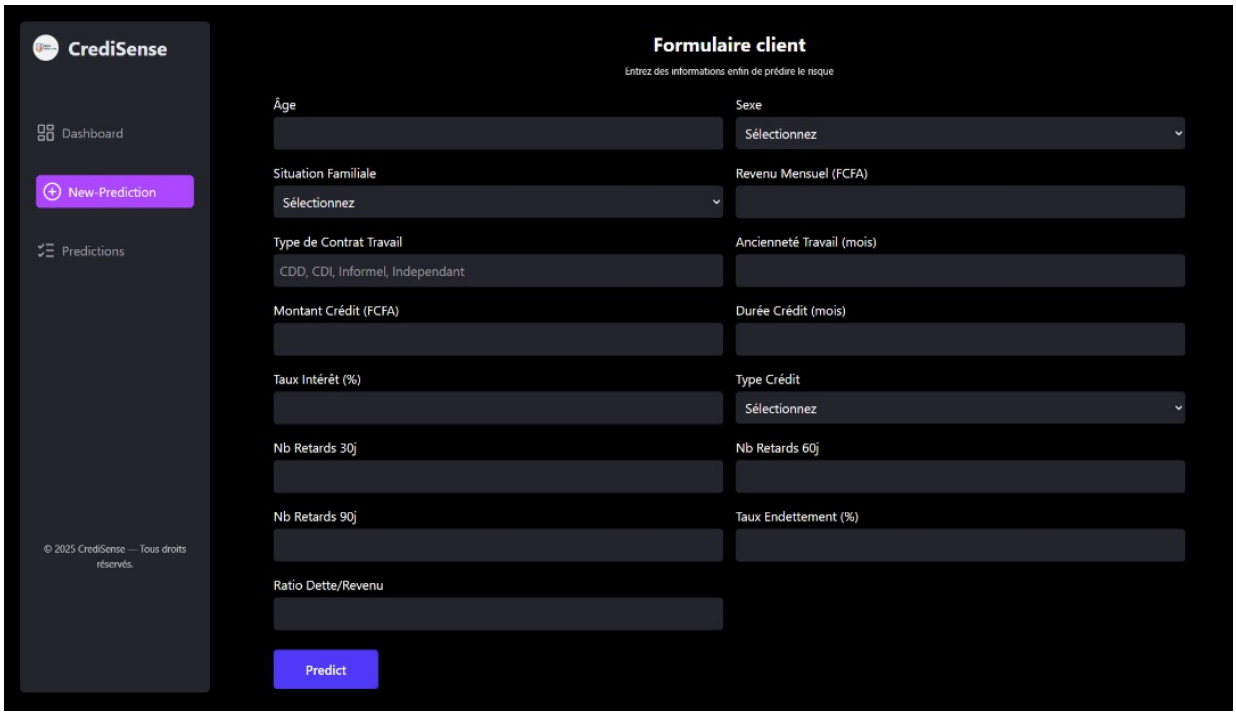

Figure 9: Interface de prédiction

2. Interface de prédiction – résultat

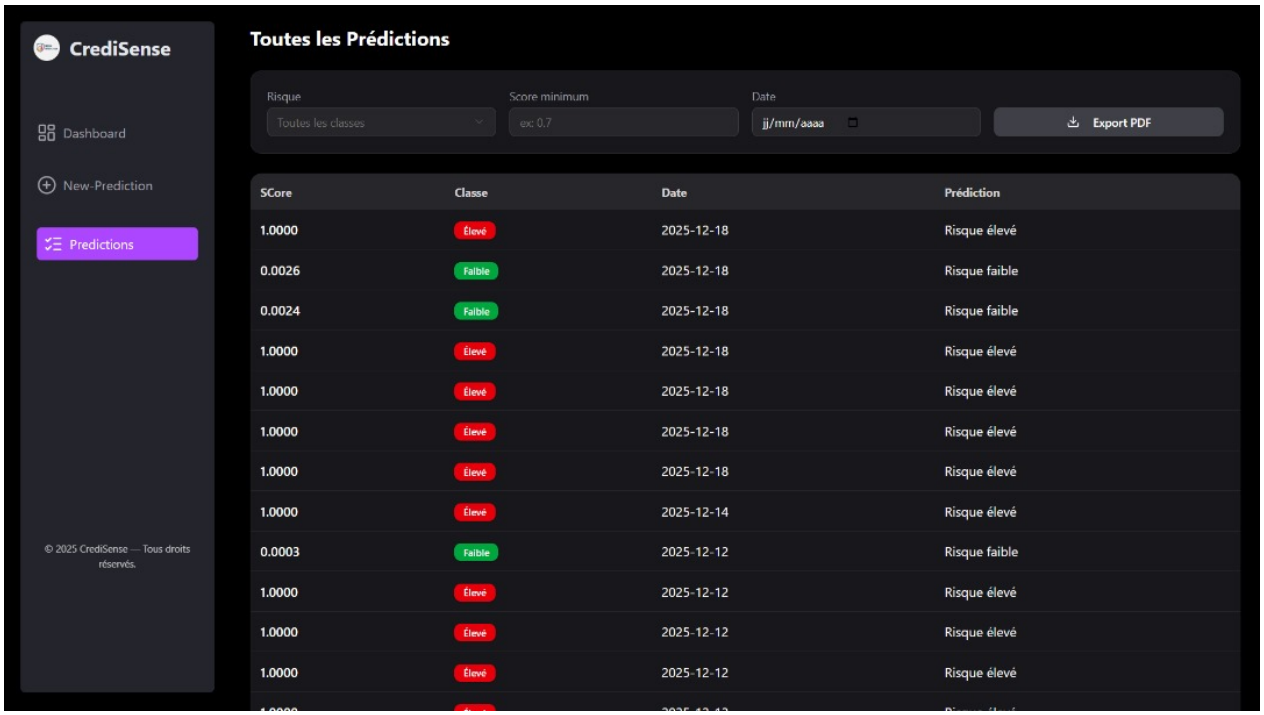

Figure 10: Résultat de prédiction

3. DashBoard

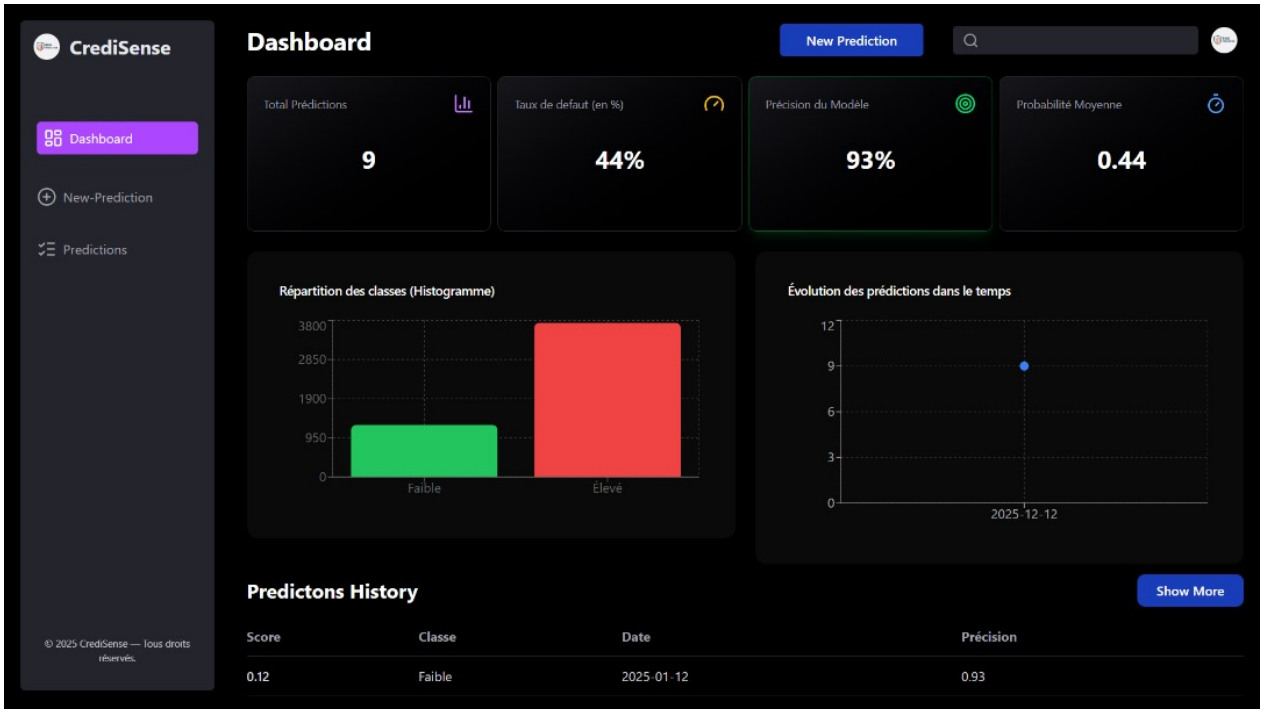

Figure 11: DashBoard

II. Apports de l'application dans la microfinance

• **Amélioration de la précision des décisions** : le modèle ML permet de prédire le risque de défaut avec fiabilité, réduisant les erreurs humaines.

• **Gain de temps et d'efficacité** : automatisation des traitements et génération rapide des rapports.

• **Sécurité et traçabilité** : historique complet des prédictions et actions des utilisateurs.

• **Support à la stratégie de risque** : les responsables peuvent visualiser les tendances et prioriser les interventions.

III. Limites rencontrées

Malgré les résultats positifs, certaines limites ont été identifiées :

• Disponibilité limitée des données complètes et à jour pour certains clients.

• Performances du modèle ML dépendantes de la qualité des données.

• Absence de version mobile complète pour l'instant.

• Nécessité de réentraînements réguliers pour maintenir la précision du modèle.

IV. Perspectives d'avenir

Plusieurs améliorations peuvent être envisagées pour renforcer l'impact de l'application :

- **Optimisation du modèle ML** : test de nouveaux algorithmes, intégration de modèles hybrides pour améliorer la prédiction.
- **Extension fonctionnelle** : ajout de modules de suivi financier, alertes automatiques et indicateurs avancés.
- **Déploiement mobile** : création d'une version mobile pour une consultation en temps réel sur smartphone ou tablette.
- **Expansion nationale** : intégration de données d'autres institutions de micro-finance camerounaises et internationales.

- **Amélioration de la sécurité** : mise en place de mécanismes avancés de surveillance et détection d'intrusions.

V. Conclusion

En somme, ce projet a permis de concevoir et de mettre en œuvre une solution complète et fonctionnelle, répondant aux besoins des institutions de microfinance au Cameroun pour la gestion du risque de défaut. L'application offre un gain de performance, une meilleure fiabilité dans les décisions de crédit et une sécurité adaptée au secteur financier. Elle constitue une base solide pour de futures évolutions, permettant aux institutions de microfinance d'optimiser leur stratégie de risque et de renforcer leur stabilité financière.

CONCLUSION GENERALE

La réalisation de ce projet de conception d'une application de prédiction du risque de défaut pour les institutions de microfinance camerounaises a permis de démontrer l'apport significatif des technologies modernes dans l'amélioration des processus décisionnels financiers. En combinant une architecture backend robuste basée sur FastAPI, un frontend ergonomique développé avec React, ainsi qu'un modèle de Machine Learning entraîné sur des données représentatives, l'application conçue offre un outil fiable, automatisé et capable d'anticiper les risques de non-remboursement avec une précision satisfaisante. Elle permet ainsi aux IMF d'optimiser la gestion de leur portefeuille client, de réduire les pertes liées aux défauts et de renforcer leur stabilité financière. Malgré ces avancées, certaines limites subsistent, notamment la disponibilité de bases de données plus riches, la nécessité d'un réentraînement périodique du modèle, et les défis liés à l'intégration à large échelle dans des environnements opérationnels variés. Néanmoins, les perspectives d'évolution sont prometteuses et incluent l'amélioration du modèle prédictif, l'extension nationale de la solution, l'enrichissement des fonctionnalités analytiques, ainsi que le développement d'une version mobile pour une accessibilité accrue. Ce travail constitue ainsi une base solide pour la modernisation des outils de gestion du risque dans les IMF camerounaises et ouvre la voie à de nouvelles innovations adaptées à leur contexte.

BIBLIOGRAPHIE / RÉFÉRENCES

1. Géron, A., *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 3rd Edition, O'Reilly, 2022.
2. Raschka, S., *Python Machine Learning*, 3rd Edition, Packt, 2019.
3. FastAPI Documentation – https://fastapi.tiangolo.com/
4. React.js Documentation – https://reactjs.org/docs/getting-started.html
5. Tailwind CSS Documentation – https://tailwindcss.com/docs
6. IMF Network Reports, Banque des États de l'Afrique Centrale, 2023.
7. OWASP Foundation, *OWASP Top 10 Security Risks*, 2022.
8. Cameroonian Ministry of Finance – *Regulations on Microfinance Institutions*, 2023.