

Rapport de Projet Tutoré:

Sujet : Chatbot pour le support client des services Bancaire

Nom donné au chatbot : Yakfis-Bot

Nom :	OUEDRAOGO
Prénom:	Mohamed
Filière :	Génie Logiciel
Option:	Data Analyse
Niveau :	Licence 3

Professeur : Dr. OUEDRAOGO CHRISTIAN

Lien du dépôt GitHub:

<https://github.com/Mohamed-236/Projet-Tutore-Chatbot-de-support-client-pour-les-services-bancaires>

Introduction

▪ Contexte du projet

Avec la digitalisation croissante des services bancaires, les clients recherchent des moyens rapides et efficaces pour interagir avec leur banque. Les chatbots sont devenus des outils incontournables pour offrir un service client automatisé, disponible 24h/24, capable de répondre à des questions sur les comptes, les transactions, les cartes bancaires etc...

C'est dans ce contexte que **Yakfis-Bot** (le nom donné à mon chatbot) est conçu pour assister les clients bancaires en répondant à leurs questions, en fournissant un historique des transactions et en facilitant certaines opérations courantes, tout en conservant un suivi des interactions pour améliorer la qualité du service.

▪ Objectifs du projet

Le présent projet vise à développer un **chatbot bancaire intelligent** permettant d'améliorer l'expérience utilisateur et l'efficacité du service client. Les objectifs principaux sont :

⇒ Automatiser l'assistance bancaire

Mettre en place un assistant virtuel capable de répondre aux demandes des utilisateurs concernant :

- la gestion des comptes bancaires,
- le suivi des transactions,
- les informations liées aux cartes bancaires,
- et d'autres services bancaires courants.

⇒ Permettre l'exécution d'opérations financières

Offrir la possibilité aux clients d'effectuer certaines transactions automatiquement via le chatbot, sans intervention humaine directe (ex : virements internes).

⇒ Garantir la sécurité et la confidentialité

Assurer la protection des données sensibles des utilisateurs grâce à des protocoles sécurisés, tout en respectant les normes bancaires.

⇒ **Optimiser le travail des analystes bancaires**

Faciliter le suivi, la gestion et l'analyse des interactions clients afin de :

- améliorer la qualité du service,
- détecter des transactions anormales,
- réduire les erreurs humaines.

⇒ **Améliorer l'expérience utilisateur**

Proposer une interface intuitive, simple d'utilisation, disponible 24/7, avec une interaction fluide et personnalisée.

Impact attendu :

- **Réduction du temps de réponse** et de traitement des demandes client.
- **Amélioration notable de la satisfaction client** grâce à une assistance rapide et continue.
- **Optimisation du suivi des requêtes** permettant une meilleure détection des comportements suspects ou des anomalies.
- **Allègement de la charge opérationnelle** du personnel bancaire.
- **Renforcement de l'image moderne et digitale** de l'institution bancaire.

■ **Plateformes et outils utilisés**

- **Langage et framework** : Python, Flask pour le développement web.
- **Base de données** : PostgreSQL pour stocker les utilisateurs, conversations et interactions.
- **NLP** : SpaCy pour l'analyse des questions des utilisateurs et recherche de réponses.
- **Front-end** : HTML, CSS, et Bootstrap pour une interface responsive et un peu de javascript pour le scrolle automatique des pages.
- **Environnement de développement** : VS Code avec l'environnement virtuel venv pour gérer les dépendances.

Analyse des besoins

▪ Description des données collectées

- **Sources** : Base de données interne simulant les comptes bancaires et transactions.
- **Volume et type de données** :

Les données sont composées de 8 tables chacune avec un rôle spécifique à savoir :

utilisateur,trans_client,compte,interaction,conversation,faq,suspicion,carte

Les Données sont des données **structurées** stockées dans la base de données PostgreSQL.

▪ Analyse des utilisateurs et de leurs besoins

- **Utilisateurs principaux** : Ce sont les clients bancaires souhaitant accéder à des informations sur leurs comptes, effectuer des transactions, poser des questions pour avoir des réponses à leurs préoccupations etc...
- **Analystes bancaires** : Les analystes utilisent le Dashboard pour suivre les interactions des utilisateurs et valider les transactions nécessitant une analyse avant validation ou annuler les transactions jugées susceptible de fraude.
- **Besoins spécifiques** :

Les besoins spécifiques sont entre autres :

- Effectuer des transactions de manière automatiques via le bot en fournissant juste les informations nécessaires au bot
- Obtenir rapidement le solde ou l'historique des transactions.
- Effectuer des opérations sur sa carte bancaire (consultation du statut, blocage, remplacement)
- Poser des questions fréquentes et obtenir des réponses rapides.
- Obtenir rapidement des éclaircissements sur certaines situations d'urgence liée aux comptes
- Suivre les différentes opérations des transactions et prendre des décisions nécessaires

- Signaler des potentielles fraudes lier à son compte
- Rentrer en contact avec un agent humain pour plus d'éclaircissement

▪ Étude de marché et concurrence

Les banques utilisent déjà des chatbots, mais souvent limités aux FAQ simples.

Yakfis-Bot se distingue par :

- ✓ L'intégration de la gestion des transactions et des comptes par le bot de manière rapide et sécurisée
- ✓ La gestion automatique des cartes de crédit (consultation du statut, blocage, remplacement)
- ✓ La gestion des opérations suspecte liée aux comptes
- ✓ La gestion des historiques complète des transactions et interactions.
- ✓ La gestion des suivis des Operations de transactions
- ✓ Une interface simple, descriptif, moderne et responsive.

Conception et modélisation

▪ Choix des techniques

- **NLP** : L'utilisation de SpaCy notamment le large model(lg) pour comparer la similarité entre question utilisateur et FAQ.
- **Le langage SQL** : pour récupérer les comptes et transactions.
- **L'Architecture MVC** : Flask pour séparer la logique, la vue et le modèle.

▪ Architecture du système

- **Flux** : l'utilisateur envoie une question → Flask reçoit la requête → traitement NLP → récupération des données si nécessaire → réponse générée → affichage dans l'interface du bot.
- **Diagramme simplifié** :

Utilisateur → Flask → NLP → Base de données → Flask → Interface utilisateur

■ Modélisation de la base de données

Tables principales :

- ✓ utilisateur : Stocke les informations des clients.
- ✓ compte : stocke les informations sur les comptes.
- ✓ Trans_client : Sauvegarde les historiques des transactions.
- ✓ conversation : Sauvegarder les titres des conversations créer par l'utilisateur afin de grouper les historiques par titre pour une meilleure organisation.
- ✓ interaction : Sauvegarde les messages utilisateurs et réponses du bot.
- ✓ Carte : stocke les informations liées au carte bancaires
- ✓ Faq : Stocke les questions fréquentes et les réponses à ces questions

Implémentation

■ L'accueil :

Yakfis-bot a été conçu pour aider les clients bancaires et donc chaque client est censé disposer des informations lui permettant de s'authentifier afin de pouvoir accéder au bot, nécessaires pour la sécurité et la confidentialité. Pour cela une page d'accueil responsive avec des carrousels est créé pour un peu guider les utilisateurs de leur premier accès au bot, afin de leur expliquer un peu le rôle du chatbot. Sur cette page se trouve un lien menant à la page de connexion. Cette page de connexion joue deux rôles à savoir :

Elle récupère les informations valides d'un client et lui donne accès au chabot d'une part et d'autre part, elle récupère celles d'un conseiller et lui mène vers le dashboard,

Cela grâce aux informations de connexion spécifiques aux clients et aux conseiller enregistrées dans la base de données

■ La page de connexion :

Elle est conçue pour permettre à chaque utilisateur c'est à dire client légal de la banque de se connecter avec ses informations de connexion (email, mot de passe) afin de

pouvoir accéder au chatbot et effectuer des opérations sensibles ou poser des questions pour trouver des réponses pertinentes.

▪ La construction du dashboard :

Le Dashboard est l'espace réservé aux conseillers ou analystes de la banque qui ont pour rôle d'effectuer des contrôles approfondis en ce qui concerne les transactions des utilisateurs et décider d'une potentielle validation ou annulations en cas de fraude lier aux transactions

▪ Développement du chatbot

Le chatbot a été conçu pour offrir une interaction fluide, sécurisée et personnalisée aux utilisateurs. Plusieurs mécanismes techniques ont été mis en place :

- **Gestion des utilisateurs avec Flask Sessions**
 - Permet de conserver l'état de la conversation et les informations temporaires liées à chaque utilisateur (identité, compte sélectionné, étape en cours, etc.).
- **Gestion des formulaires via méthodes HTTP (POST / GET)**
 - Utilisée pour récupérer les requêtes et transmettre les réponses du chatbot.
- **Système d'analyse intelligente des questions**
 - Utilisation du NLP (SpaCy) pour :
 - normaliser la question (minuscules, suppression accents...)
 - extraire les mots-clés
 - gérer les variantes d'écritures et fautes mineures
- **Recherche intelligente dans la FAQ avec PostgreSQL + pg_trgm**
 - Permet de trouver la meilleure réponse même si la question n'est pas identique

Exemple : “ouvrir un compte ?” match aussi avec “comment créer un compte ?”
- **Gestion du contexte conversationnel**
 - Le bot suit un processus étape par étape (ex : virement bancaire)
 - demande du compte → du montant → validation

- **Contrôles métier et sécurité des transactions**
→ Vérification du solde, seuils de sécurité, fraude potentielle...
- **Réponses dynamiques et personnalisées**
→ Les messages du bot s'adaptent selon :
 - l'étape
 - l'utilisateur
 - le résultat de la requête (succès ou refus)
- **Gestion des intentions courantes**
→ Salutations, questions personnelles sur le bot, aide, erreurs...

■ Modules clés

Le système est structuré en plusieurs modules afin de faciliter la maintenance, la lisibilité du code et l'évolution du chatbot :

- **nlp.py**
→ Contient la logique de traitement du langage naturel (NLP)
 - fonction trouver_reponse() pour analyser la question utilisateur
 - recherche intelligente de réponses dans la FAQ
 - calcul de similarité sémantique
- **database.py**
→ Gère toutes les interactions avec la base PostgreSQL
 - récupération des comptes utilisateurs, soldes, transactions
 - ajout des transactions, suivi et suspicion
 - accès à la FAQ
- **flask_app.py**
→ Point central de l'application
 - gestion des routes (/chatbot, /connexion/deconnexion/dashbord, etc.)
 - gestion des sessions pour mémoriser l'état de conversation
 - appels au NLP et à la base selon l'intention détectée

- **Templates HTML / CSS / Bootstrap /js**
→ Interface utilisateur moderne et responsive
 - zones d'affichage du chat
 - formulaire d'envoi des messages
 - adaptation à tous les écrans (mobile / PC)
 - défilement automatique du chat
- **static/ (css/images)**
→ Améliore l'interactivité (défilement automatique du chat, animations...)

▪ Algorithmes

Le chatbot s'appuie sur plusieurs techniques d'intelligence artificielle et de traitement du langage pour comprendre et répondre aux utilisateurs :

- **Recherche par similarité sémantique (SpaCy)**
 - Chaque question posée est analysée en NLP
 - Une comparaison est effectuée avec les questions de la FAQ
 - L'algorithme renvoie la réponse la plus pertinente selon un score de similarité
→ Permet de comprendre même si la question est formulée différemment
- **Filtrage intelligent par mot-clés**
 - Utilisation de listes de mots-clés pour détecter les intentions
 - Exemple : *soldé, argent, balance* → consultation du solde
 - *envoyer, virement, transférer* → opérations bancaires
 - *carte, visa, bloquer* → gestion des cartes
→ Permet une reconnaissance rapide des catégories de demandes
- **Recherche textuelle optimisée en base (pg_trgm + GIN index)**
 - Accélère la recherche floue sur les questions FAQ
 - Tolère les fautes et variations dans la saisie
→ Améliore largement les performances du chatbot

- **Gestion d'état conversationnel**
 - Une transaction ou action se déroule en plusieurs étapes
 - Le système mémorise l'avancement grâce aux sessions Flask
 - Le bot sait à quelle étape se trouve l'utilisateur
- **Règles métier bancaires**
 - Vérification du solde avant transaction
 - Détection de fraude pour montants élevés
 - Suivi et enregistrement des opérations
 - Sécurisation des actions sensibles

Résultats de l'analyse

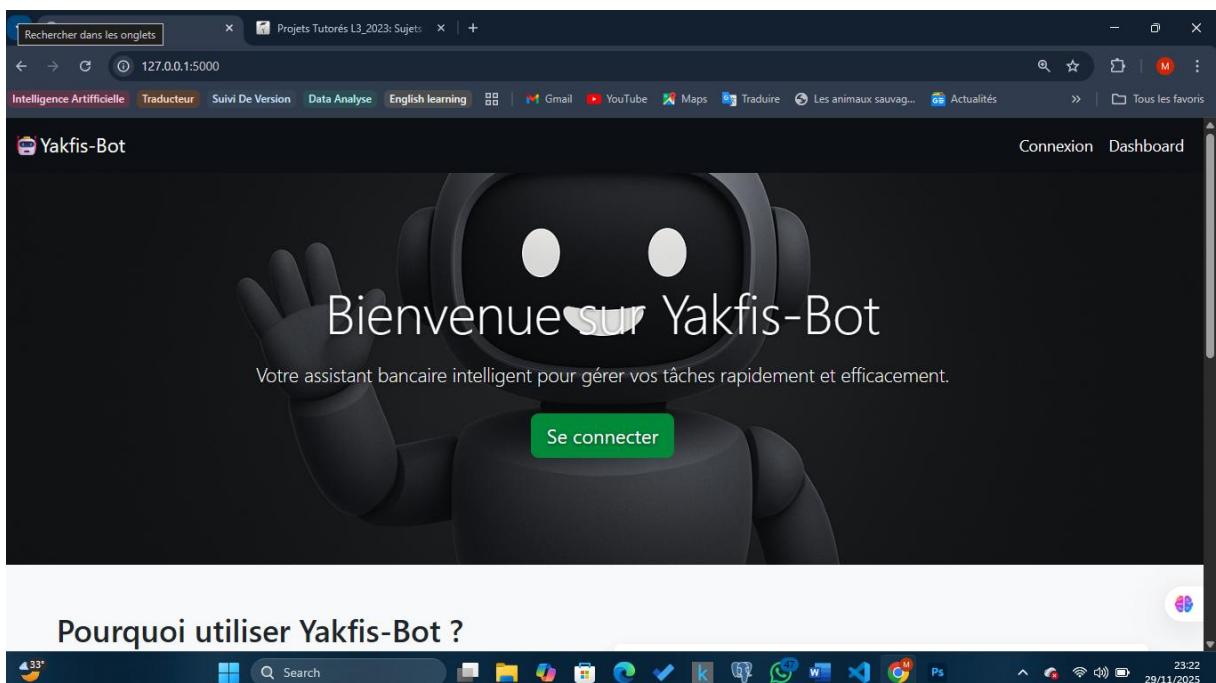
■ Fonctionnalités obtenues

Le système développé permet à l'utilisateur d'interagir avec son compte bancaire via le chatbot. Les fonctionnalités principales obtenues sont :

- **Consultation des soldes**
 - Affichage du solde de tous les comptes associés à l'utilisateur
 - Mise à jour en temps réel depuis la base de données
- **Historique des transactions**
 - Visualisation complète des opérations effectuées
 - Détails : montants, dates, statuts et type de transaction
- **Gestion des transactions bancaires**
 - Possibilité d'effectuer un virement vers un autre compte
 - Contrôles automatiques : montant valide, solde suffisant
 - Gestion des étapes de la transaction (workflow conversationnel)
- **Sécurité et détection de fraude**
 - Blocage automatique des transactions suspectes (ex. montants trop élevés)
 - Signalement dans une table dédiée au suivi des anomalies

- **Gestion des cartes bancaires**
 - Consultation de l'état de la carte (active, bloquée, expirée...)
 - Blocage d'une carte en cas de perte ou de vol
 - Assistance sur activation, renouvellement, plafond...
- **Gestion des questions fréquentes**
 - Réponse automatique aux questions fréquentes

■ Captures d'écran



Pourquoi utiliser Yakfis-Bot ?

Yakfis-Bot est votre assistant bancaire intelligent. Il vous permet de :

- Consulter vos soldes et l'historique de vos transactions en temps réel.
- Effectuer vos virements et paiements de façon sécurisée et rapide.
- Gérer vos cartes bancaires, bloquer ou débloquer vos cartes instantanément.
- Recevoir des notifications personnalisées sur vos finances et alertes de sécurité.
- Signaler des fraudes potentielles liées à votre compte
- Avoir une réponse automatique à des questions fréquentes(FAQ)

Grâce à une interface intuitive et conviviale, Yakfis-Bot rend vos opérations bancaires simples et agréables. Plus besoin de naviguer entre plusieurs menus: toutes les fonctionnalités sont accessibles directement dans le chat.

La sécurité est au cœur de notre solution. Vos données sont protégées et vos transactions sont chiffrées, vous offrant une expérience bancaire fiable et sans souci.



instantanément.

- Recevoir des notifications personnalisées sur vos finances et alertes de sécurité.
- Signaler des fraudes potentielles liées à votre compte
- Avoir une réponse automatique à des questions fréquentes(FAQ)

Grâce à une interface intuitive et conviviale, Yakfis-Bot rend vos opérations bancaires simples et agréables. Plus besoin de naviguer entre plusieurs menus: toutes les fonctionnalités sont accessibles directement dans le chat.

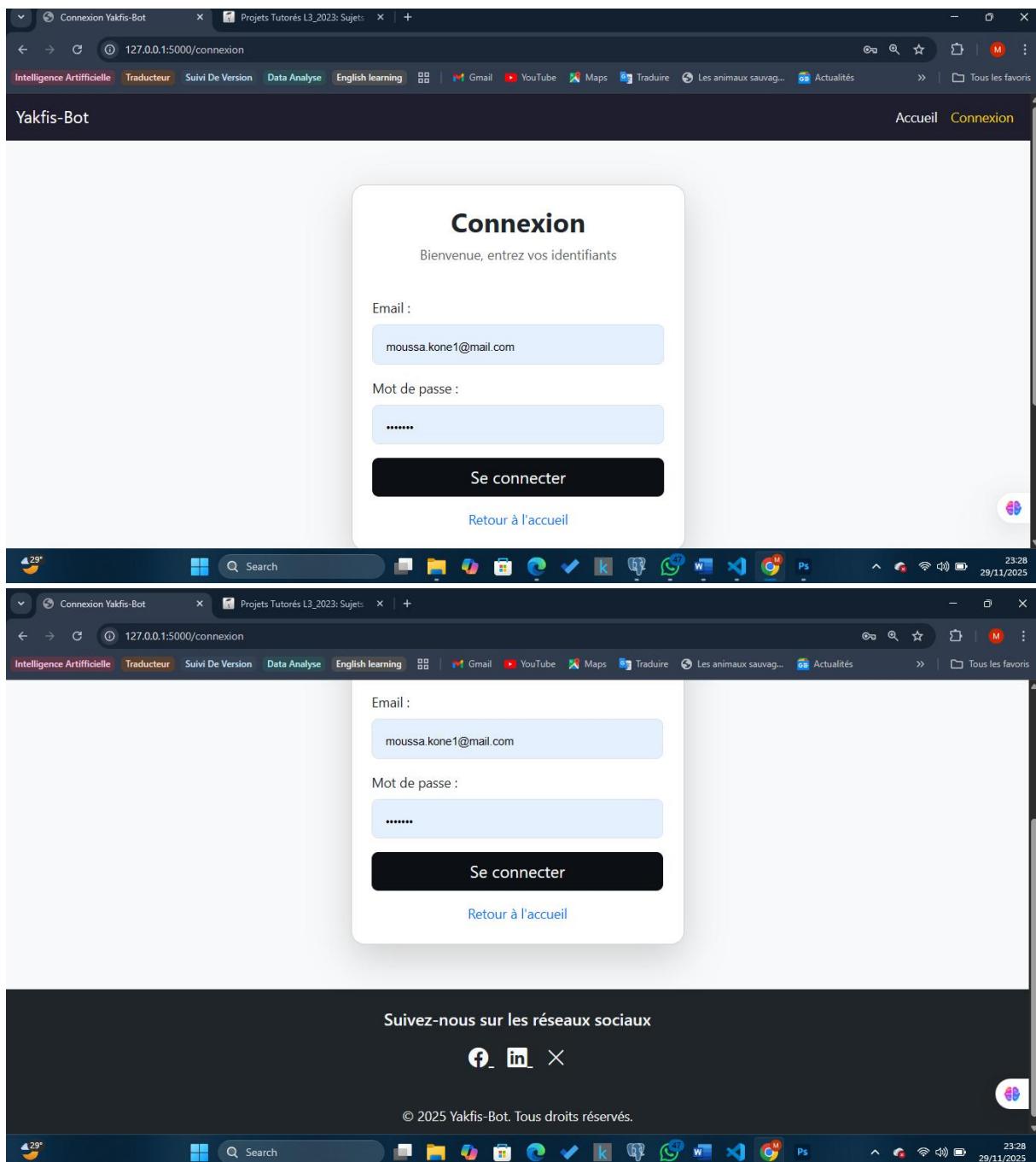
La sécurité est au cœur de notre solution. Vos données sont protégées et vos transactions sont chiffrées, vous offrant une expérience bancaire fiable et sans souci.



Suivez-nous sur les réseaux sociaux

© 2025 Yakfis-Bot. Tous droits réservés.

Page de connexion :

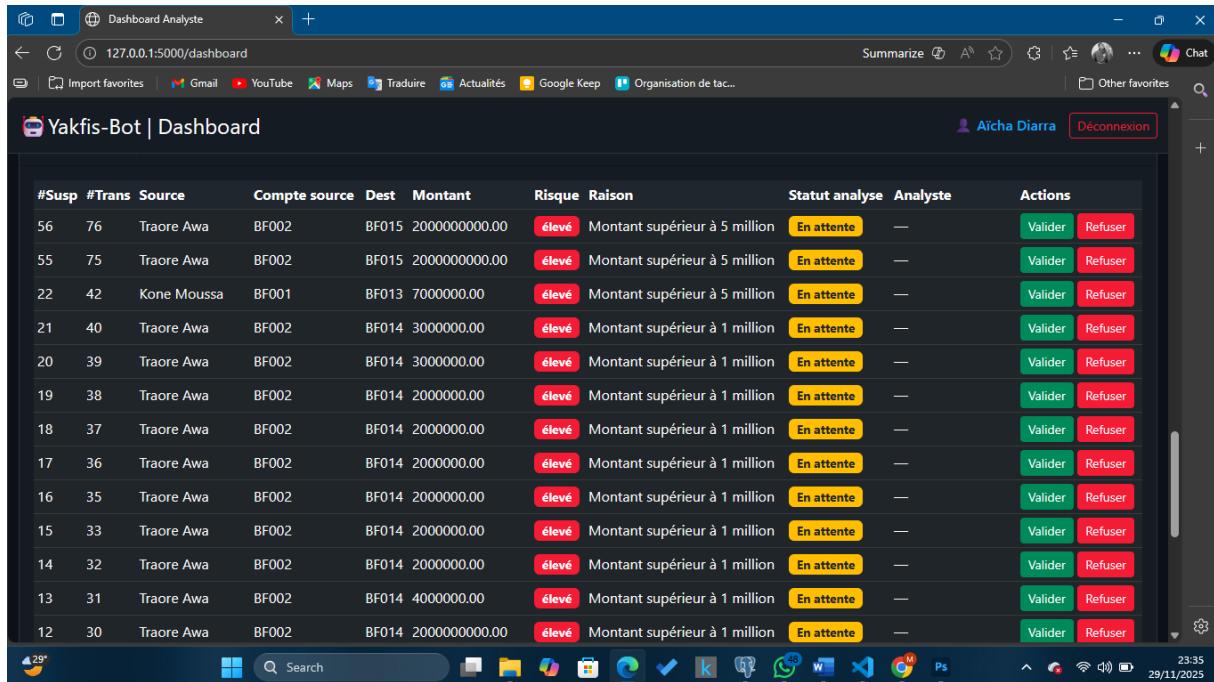


Page Dashboard pour analyste :

The screenshot shows two identical instances of a web application running on a local server at 127.0.0.1:5000. The top instance displays a summary with 'Total transactions' (44) and 'Suspects' (24). Below this is a table of transaction details:

ID	Source	Compte source	Destination	Montant	Type	Date	Statut	Suspecte
13	Kone Moussa	BF001	BF004	5000.00	envoi	None	Réussie	Non
16	Kone Moussa	BF001	BF013	90000000000.00	envoi	None	Refusée	Non
15	Kone Moussa	BF001	BF012	3000000000.00	envoi	None	Refusée	Non
14	Kone Moussa	BF001	BF008	6000000000.00	envoi	None	Refusée	Non
12	Traoré Mariam	BF009	BF007	2000.00	envoi	None	Réussie	Non
76	Traore Awa	BF002	BF015	2000000000.00	envoi	2025-11-29 11:33:21.074175	Refusée	Non
75	Traore Awa	BF002	BF015	2000000000.00	envoi	2025-11-29 11:25:24.456205	Refusée	Non
42	Kone Moussa	BF001	BF013	7000000.00	envoi	2025-11-26 14:10:59.809179	Refusée	Non

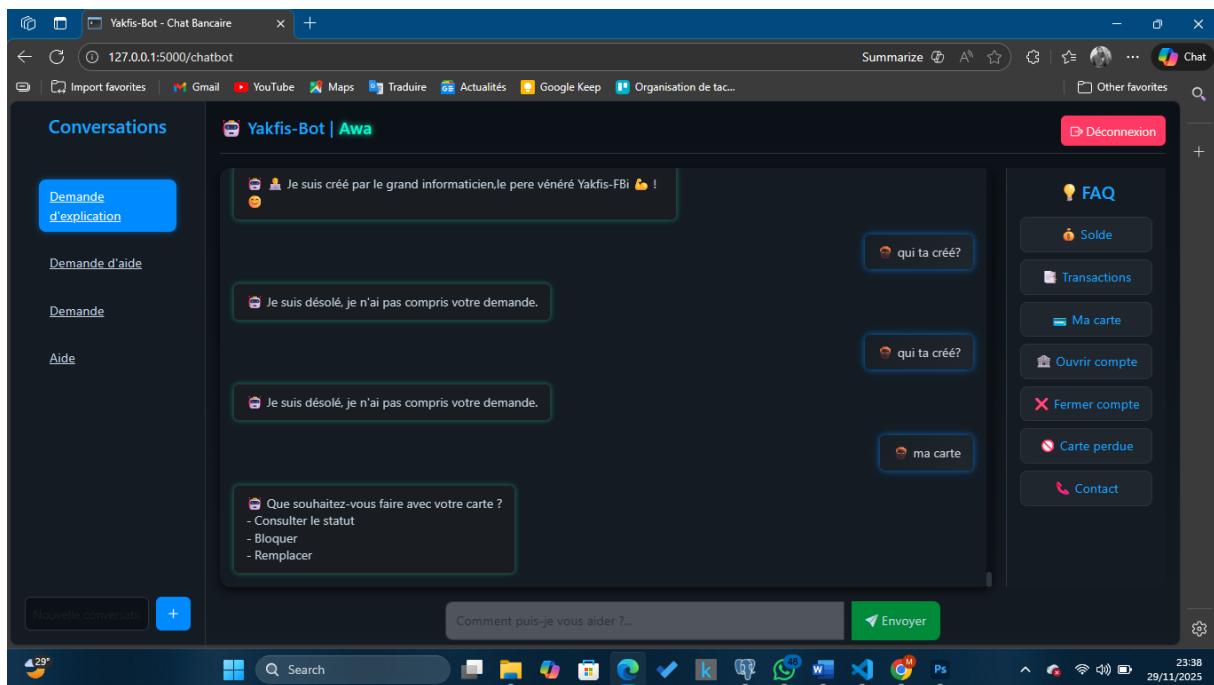
The bottom instance shows a similar table with a different set of transactions, including successful (réussie) and failed (Refusée) entries.



The screenshot shows a web-based dashboard titled 'Yakfis-Bot | Dashboard'. The interface includes a header with user information ('Aïcha Diarra') and a navigation bar with links like 'Summarize', 'Import favorites', 'Gmail', 'YouTube', etc. The main content is a table with the following columns: #Susp, #Trans, Source, Compte source, Dest, Montant, Risque, Raison, Statut analyse, Analyste, and Actions. The table lists 15 rows of data, each representing a transaction with specific details like account numbers, amounts, and risk levels. Buttons for 'Valider' (Green) and 'Refuser' (Red) are present in the 'Actions' column.

#Susp	#Trans	Source	Compte source	Dest	Montant	Risque	Raison	Statut analyse	Analyste	Actions
56	76	Traore Awa	BF002	BF015	2000000000.00	élevé	Montant supérieur à 5 million	En attente	—	<button>Valider</button> <button>Refuser</button>
55	75	Traore Awa	BF002	BF015	2000000000.00	élevé	Montant supérieur à 5 million	En attente	—	<button>Valider</button> <button>Refuser</button>
22	42	Kone Moussa	BF001	BF013	7000000.00	élevé	Montant supérieur à 5 million	En attente	—	<button>Valider</button> <button>Refuser</button>
21	40	Traore Awa	BF002	BF014	3000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
20	39	Traore Awa	BF002	BF014	3000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
19	38	Traore Awa	BF002	BF014	2000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
18	37	Traore Awa	BF002	BF014	2000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
17	36	Traore Awa	BF002	BF014	2000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
16	35	Traore Awa	BF002	BF014	2000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
15	33	Traore Awa	BF002	BF014	2000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
14	32	Traore Awa	BF002	BF014	2000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
13	31	Traore Awa	BF002	BF014	4000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>
12	30	Traore Awa	BF002	BF014	2000000000.00	élevé	Montant supérieur à 1 million	En attente	—	<button>Valider</button> <button>Refuser</button>

Page du chatbot :



The screenshot shows a web-based chat interface titled 'Yakfis-Bot - Chat Bancaire'. The interface includes a header with user information ('Aïcha Diarra') and a navigation bar with links like 'Summarize', 'Import favorites', 'Gmail', 'YouTube', etc. The main content is a conversation window with the bot 'Awa'. The conversation history is as follows:

- User: Je suis créé par le grand informaticien, le pere vénéré Yakfis-FBi !
- Bot: qui ta créé?
- User: Je suis désolé, je n'ai pas compris votre demande.
- Bot: qui ta créé?
- User: Je suis désolé, je n'ai pas compris votre demande.
- Bot: ma carte
- User: Que souhaitez-vous faire avec votre carte ?
- Consulter le statut
- Bloquer
- Remplacer

On the left side, there is a sidebar with categories: 'Demande d'explication', 'Demande d'aide', 'Demande', and 'Aide'. On the right side, there is a sidebar titled 'FAQ' with links: 'Solde', 'Transactions', 'Ma carte', 'Ouvrir compte', 'Fermer compte', 'Carte perdue', and 'Contact'.

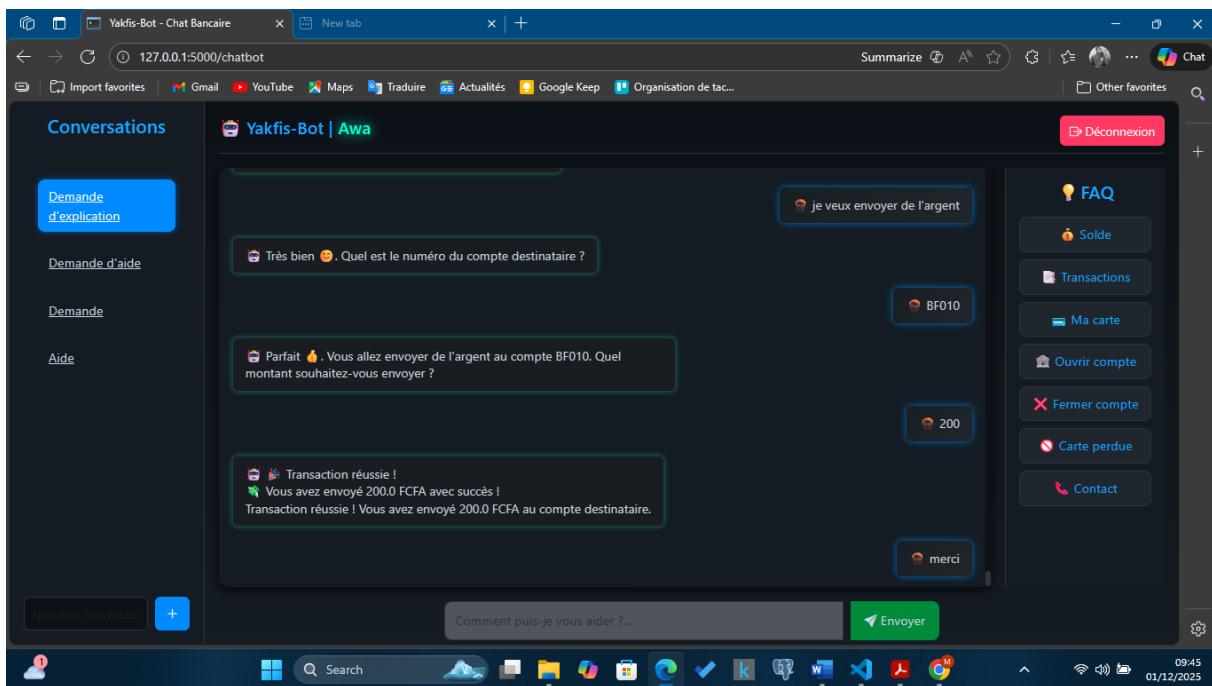
→ A gauche de la page du chabot, il y a la possibilité de créer une conversation afin de garder l'historique de manière organiser.

→ A droite de la page se trouve des boutons représentant les questions fréquemment posées par les utilisateurs pour faciliter l'expérience utilisateur

→ En haut de la page à droite, se trouve un bouton permettant à l'utilisateur de se déconnecter

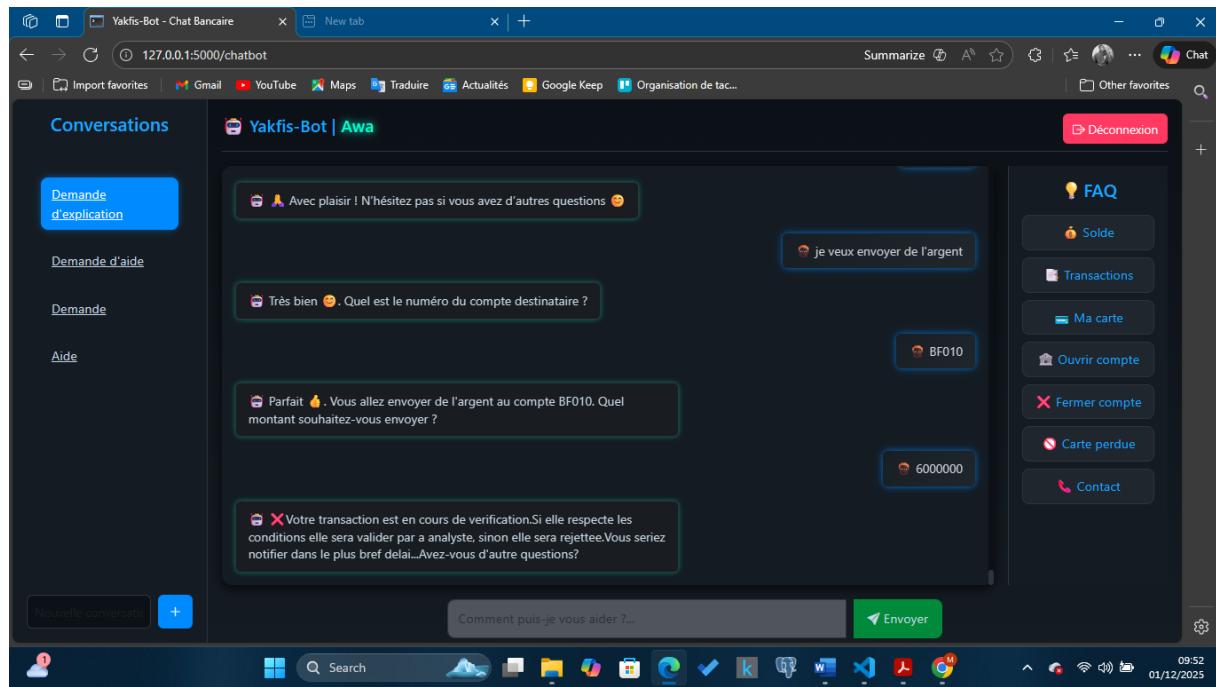
Fonctionnalités :

→ Transaction réussie :



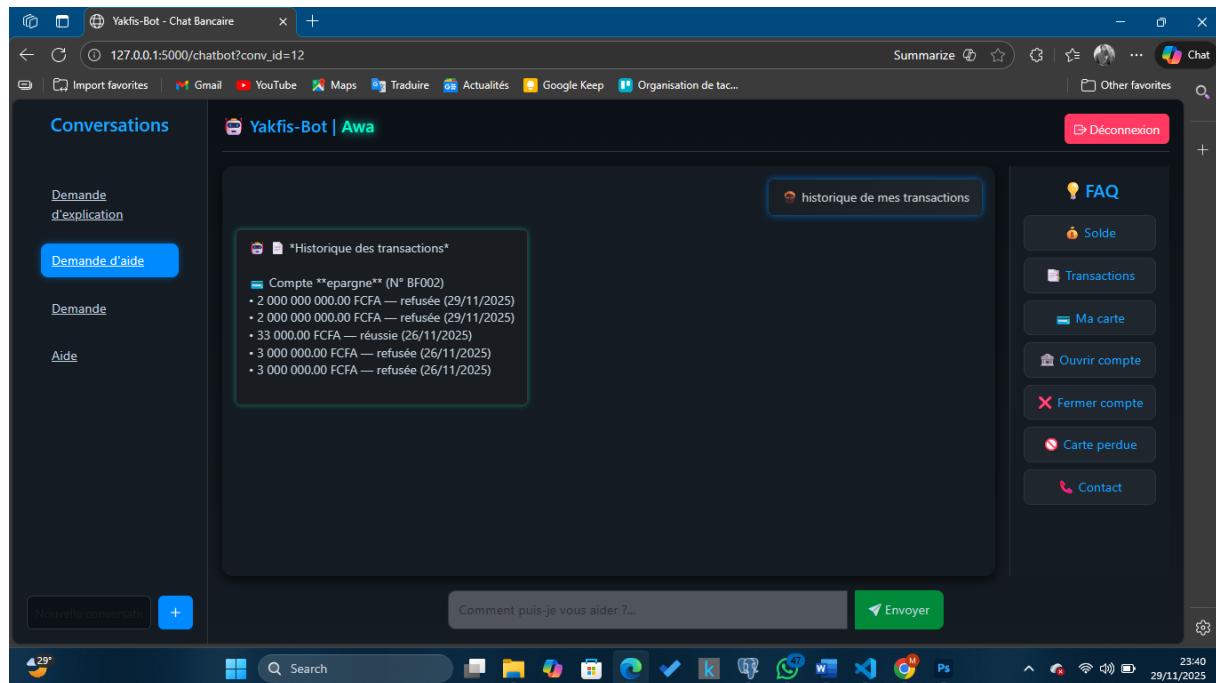
Nb: Les valeurs sont automatiquement calculées dans la base de données lorsque qu'une transaction est validée. Le solde de l'envoyeur se voit prélevé la somme envoyée et le solde du destinataire se voit augmenter par la somme reçue

→ Transaction annulée :

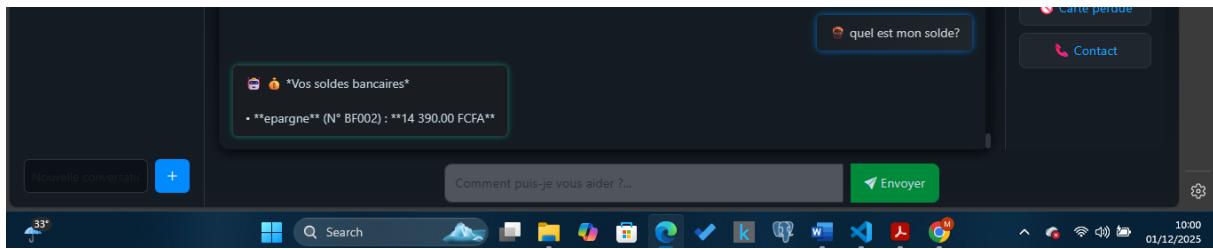


Nb: Un montant plafond a été fixé dans la base de données et si la somme que l'utilisateur veux envoyer est égale ou supérieure à ce montant, automatiquement la transaction fait l'objet d'une vérification au préalable avant validation ou annulation par nos analystes:

→ Historique des transactions :

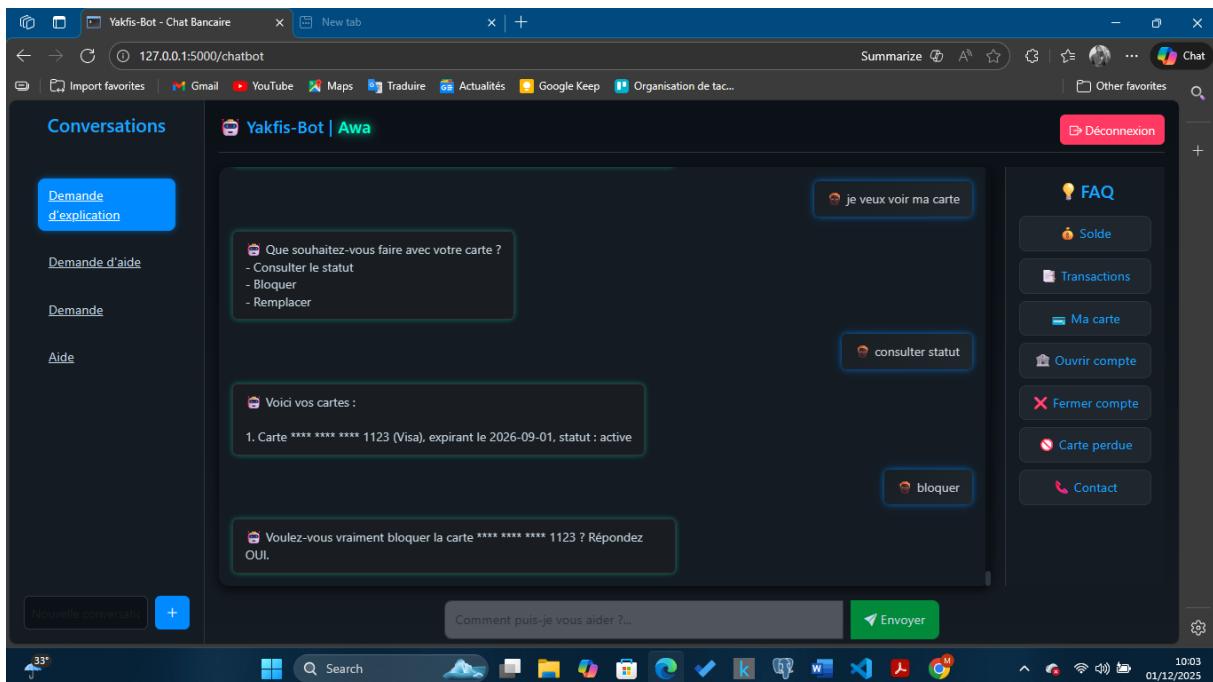


→ Solde :

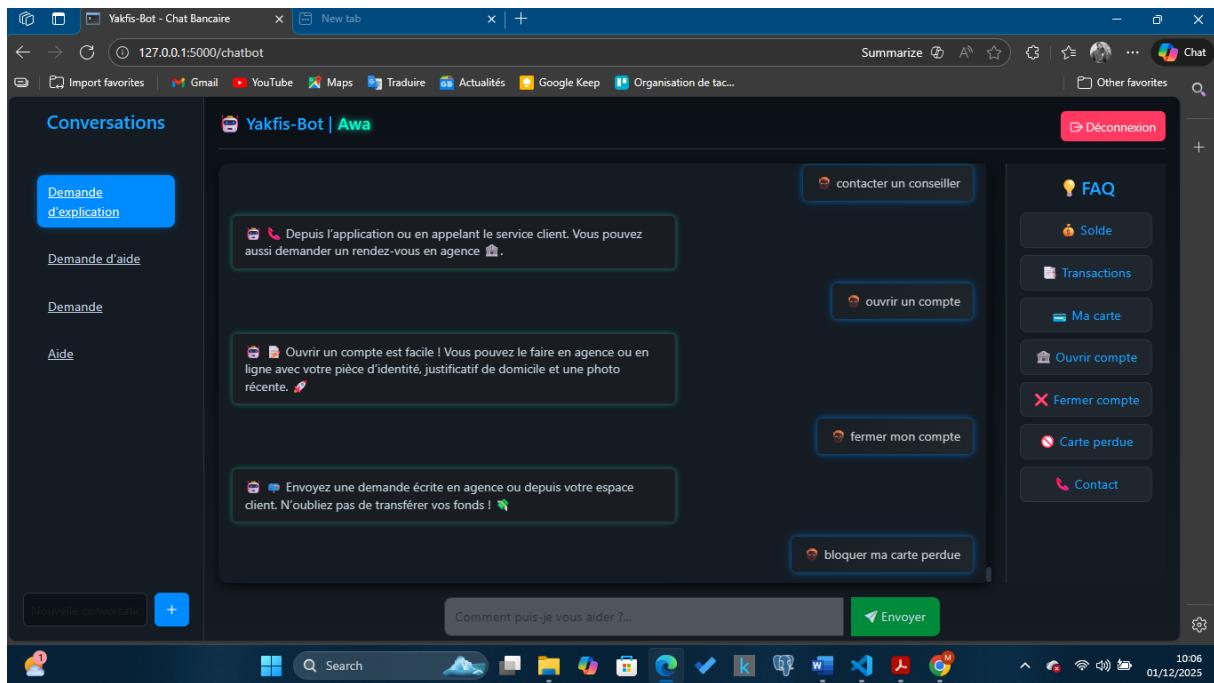


NB : Le solde est affiche pour différents comptes au cas où l'utilisation dispose de plusieurs comptes

→ Carte Bancaire (consulter le statut, bloquer en cas de perte par exemple, remplacer) :



→ Quelques questions fréquentes :



Tests et validation

Afin de garantir la fiabilité du chatbot bancaire et la conformité avec les besoins définis, plusieurs types de tests ont été réalisés :

- **Tests unitaires**
 - Vérification du bon fonctionnement des fonctions critiques telles que :
 - récupération des comptes depuis la base de données
 - extraction et affichage des transactions
 - gestion du statut des cartes bancaires
 - Résultat : toutes les fonctions retournent des données cohérentes.
- **Tests d'intégration**
 - Envoi de requêtes utilisateur complètes via l'interface (chat)
 - Vérification de la réponse appropriée selon le scénario (solde, virement...)
 - Test de la communication entre *Flask* → *logique du chatbot* → *base de données PostgreSQL*
 - Résultat : l'intégration entre les composants est stable.
- **Tests interactifs avec utilisateurs**

- Validation de l'ergonomie du chatbot
- Vérification que toute la conversation reste sur la même page
- Clarté et adaptation des messages de réponse (émoticônes, reformulations...)
- Résultat : bonne compréhension et satisfaction des utilisateurs testeurs.

✓ Conclusion des tests

Toutes les **fonctionnalités principales** ont été validées et répondent aux exigences établies : réponses pertinentes, sécurité des opérations, interface claire et fluide.

Performances et optimisation

Afin d'assurer une expérience utilisateur rapide et fluide lors des interactions avec le chatbot bancaire, plusieurs techniques d'optimisation ont été mises en œuvre :

- **Optimisation des requêtes SQL**
 - Utilisation de requêtes paramétrées et indexées pour réduire les temps de réponse de la base de données.
 - Suppression des requêtes répétitives inutilement lourdes.
- **Préchargement des données importantes**
 - Chargement anticipé de l'historique des comptes et transactions afin d'accélérer l'affichage et l'analyse.
 - Mise en cache des informations fréquemment utilisées.
- **Amélioration de la précision NLP**
 - Ajustement des seuils de similarité pour éviter les réponses incorrectes ou hors sujet.
 - Réduction des erreurs grâce à une meilleure gestion des mots-clés en complément du traitement linguistique.
- **Usage du modèle SpaCy Large (fr_core_news_lg)**
 - Le modèle **lg** contient plus de données et de vecteurs linguistiques, ce qui :
 - augmente la précision d'analyse des intentions utilisateur

- accélère la recherche de similarité
- améliore la compréhension des questions variées du client
- **Gestion des sessions**
 - Conservation du contexte conversationnel pour réduire les traitements redondants et offrir des réponses plus cohérentes.

Résultat : le temps de réponse du chatbot est nettement réduit et les erreurs de compréhension sont mieux maîtrisées.

Sécurité et protection des données

Dans le cadre d'une application bancaire, la sécurité est un élément essentiel. Plusieurs mécanismes ont été mis en place pour garantir la protection des données des utilisateurs :

- **Hachage sécurisé des mots de passe**
 - Utilisation de l'algorithme *bcrypt* assurant un chiffrement robuste et résistant aux attaques par force brute
 - Aucun mot de passe n'est stocké en clair dans la base de données
- **Sécurisation des sessions utilisateur**
 - Utilisation des cookies sécurisés de Flask pour maintenir l'authentification
 - Expiration automatique des sessions inactives afin de limiter l'usurpation
- **Protection contre les fuites de données**
 - Aucune donnée bancaire sensible (mot de passe, numéro de carte...) n'est exposée côté client
 - Données personnelles minimisées dans les réponses du chatbot
- **Communication sécurisée**
 - Chiffrement des échanges entre client et serveur (HTTPS recommandé)
 - Requêtes protégées contre les attaques de type injection SQL grâce aux requêtes paramétrées
- **Confidentialité renforcée**

- Possibilité d'anonymiser certaines données dans les historiques
- Respect des bonnes pratiques en conformité avec les normes de protection (type RGPD)

Conclusion

Ce projet a permis de concevoir et de réaliser un **chatbot bancaire intelligent**, capable d'interagir de manière fluide avec l'utilisateur à travers une interface moderne, responsive et intuitive. Les fonctionnalités mises en œuvre couvrent des besoins essentiels tels que :

- la **réalisation des transactions**,
- la **consultation des comptes et des transactions**,
- l'**exécution contrôlée des virements**,
- la **gestion des cartes bancaires**,
- et une **FAQ dynamique** exploitant le NLP pour apporter des réponses pertinentes en temps réel.

Grâce à l'utilisation d'outils performants comme **Flask**, **PostgreSQL** et **SpaCy**, ainsi qu'à l'intégration de mécanismes de sécurité conformes aux bonnes pratiques, le système garantit une expérience fiable et sécurisée.

Développement futur

Le développement futur du chatbot vise à renforcer son **accessibilité**, sa **performance** et sa **portée fonctionnelle**. Plusieurs améliorations majeures sont envisagées :

- **Déploiement sur un serveur cloud** (ex. : AWS, Azure, Heroku) afin de rendre le service disponible à distance 24/7 et accessible depuis n'importe quel terminal.
- **Support multilingue** pour permettre une utilisation dans plusieurs régions (par exemple : français, anglais, arabe...).
- **Intégration avec des applications mobiles** (Android/iOS) offrant une expérience plus fluide via des notifications en temps réel et des commandes rapides.
- **Amélioration du NLP** pour une meilleure compréhension du langage naturel, même en présence de fautes, d'abréviations ou d'expressions informelles.

- **Connexion avec d'autres services bancaires via des APIs** pour étendre la gamme de fonctionnalités (suivi des prêts, cartes virtuelles, gestion budgétaire...).
- **Renforcement de la sécurité** avec une meilleure détection des fraudes basée sur le Machine Learning.
- **Interface conversationnelle vocale** pour interagir via assistants vocaux ou appels téléphoniques.

Annexes

Diagrammes et visualisations complémentaires

Utilisateur



Interface (HTML / CSS / Bootstrap)



Flask (routes, sessions)



NLP (SpaCy : recherche par similarité)

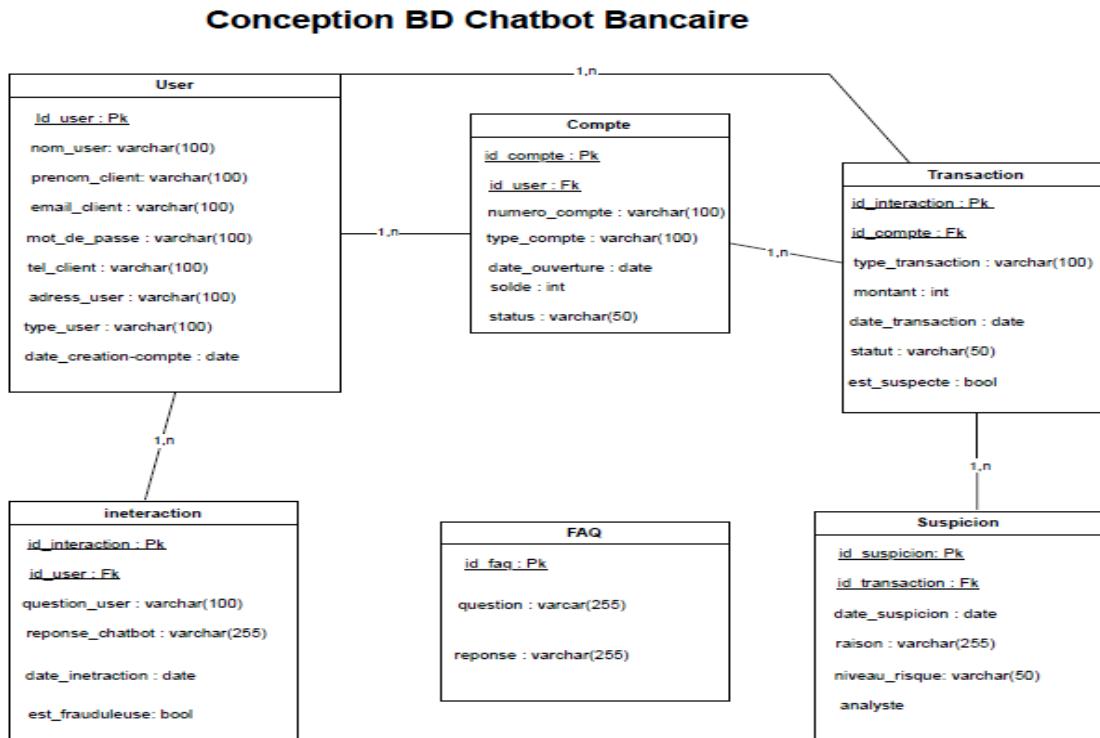


Base de données (comptes, transactions, FAQ)



Flask → Interface utilisateur

Conception de la base de données :



Nb : Diagramme incomplète, il y a des tables manquantes car j'ai envisager des les insérer plus tard et donc je les insèrerais plus tard dans le diagramme

Extraits de code source

```

# =====
# Fonction trouver_reponse
# =====

def trouver_reponse(question_user, user_id=None, current_conv=None):
    question_lower = question_user.lower().strip()

```

```
# Gestion solde

if user_id:

    if any(k in question_lower for k in ["solde", "consulter solde", "consulter argent",
"consulter mon argent"]):

        comptes = get_comptes_user(user_id)

        if not comptes:

            return "Vous n'avez aucun compte enregistré."

        texte = " 💰 *Vos soldes bancaires*\n\n"

        for c in comptes:

            solde = f"{c[3]:,}".replace(",", " ")

            texte += f"• **{c[2]}** (N° {c[1]}) : **{solde} FCFA**\n"

        return texte
```

```
# Historique transactions

if any(k in question_lower for k in ["historique", "mes transactions", "transactions"]):

    comptes = get_comptes_user(user_id)

    if not comptes:

        return " ❌ Vous n'avez aucun compte."

    texte = " 📋 *Historique des transactions*\n\n"

    for c in comptes:

        texte += f" 💼 Compte **{c[2]}** (N° {c[1]})\n"

        trans = get_transactions_compte(c[0])

        if not trans:

            texte += " — Aucune transaction récente\n\n"

            continue
```

```
for t in trans:  
  
    montant = f"{t[1]}".replace("", " ")  
  
    date = t[2].strftime("%d/%m/%Y") if hasattr(t[2], "strftime") else t[2]  
  
    texte += f" • {montant} FCFA — {t[3]} ({date})\n"  
  
  
    texte += "\n"  
  
texte = texte.replace("\n", "<br>")  
  
  
return texte  
  
  
# Gestion cartes  
  
keywords_cartes_bancaires = [  
    "bloquer", "verrouiller", "remplacer",  
    "renouveler", "statut", "etat",  
    "ma carte", "carte bancaire"  
]  
  
  
if user_id and any(k in question_lower for k in keywords_cartes_bancaires):  
    return gestion_cartes(question_lower, user_id)  
  
  
  
  
# Gestion transactions étape par étape  
  
if user_id and current_conv:  
    rep = gestion_transactions(question_lower, user_id, current_conv)  
    if rep:  
        return rep  
  
  
# =====
```

```
# NLP + pg_trgm pour FAQ

# =====

conn = get_connection()

cur = conn.cursor(cursor_factory=RealDictCursor)

question_lower = question_lower.strip()

# Récupération avec pg_trgm

cur.execute(""""

    SELECT question_faq, reponse_faq
    FROM faq
    WHERE similarity(question_faq, %s) > 0.5
    ORDER BY similarity(question_faq, %s) DESC
    LIMIT 10

""", (question_lower, question_lower))

faqs = cur.fetchall()

print(f"Nombre de FAQ récupérées : {len(faqs)}")

print(f"Exemple de FAQ : {faqs[:3]}")

cur.close()

conn.close()

if not faqs:

    return "Je suis désolé, je n'ai pas trouvé de réponse proche."


# Calcul de similarité sémantique

nlp = get_nlp_model()

doc_user = nlp(question_lower)
```

```
best_score = 0
best_answer = None
for item in faqs:
    doc_faq = nlp(item['question_faq'].lower())
    score = doc_user.similarity(doc_faq)
    if score > best_score:
        best_score = score
        best_answer = item['reponse_faq']
return best_answer if best_score >= 0.6 else "Je suis désolé, je n'ai pas compris votre demande."
```

Description :

- Fonction principale pour analyser les questions utilisateur.
- Utilise SpaCy pour calculer la similarité entre la question posée et les FAQ existantes.
- Seuil de confiance : 0.6 (pour ne retourner que des réponses fiables).

Gestion des transactions sécurisées :

```
# =====
# Gestion transactions
# =====
def gestion_transactions(question, user_id, current_conv):
    step_key = f"step_conv_{current_conv}"
    step = session.get(step_key, None)
    question_lower = question.lower().strip()
```

```
trigger_keywords = ["envoyer", "transférer", "virement", "faire une transaction","effectuer une transaction"]
```

```
if step is None and any(k in question_lower for k in trigger_keywords):
```

```
    session[step_key] = "dest"
```

```
    return "Très bien 😊 . Quel est le numéro du compte destinataire ?"
```

```
if step == "dest":
```

```
    numero_saisi = question.strip().upper().replace(" ", "")
```

```
    compte_dest = get_compte_by_numero(numero_saisi)
```

```
    if not compte_dest:
```

```
        return "❌ Ce numéro de compte n'existe pas. Réessayez ou tapez une autre commande."
```

```
        session["compte_dest"] = compte_dest
```

```
        session[step_key] = "montant"
```

```
        return f"Parfait 👍 . Vous allez envoyer de l'argent au compte {numero_saisi}. Quel montant souhaitez-vous envoyer ?"
```

```
if step == "montant":
```

```
    question_clean = question.strip().lower()
```

```
    if question_clean in ["annuler", "stop", "retour"]:
```

```
        session[step_key] = None
```

```
        session["compt_dest"] = None
```

```
        return "✅ Transaction annulée."
```

```
try:
```

```
    montant = float(question.replace(',', '.'))
```

```
except ValueError:
```

```
    session[step_key] = None
```

```
session["compte_dest"] = None  
return "✖ Veuillez saisir un montant valide."  
  
if montant <= 0:  
    return "✖ Le montant doit être positif."  
  
comptes = get_comptes_user(user_id)  
if not comptes:  
    session[step_key] = None  
    session["compte_dest"] = None  
    return "✖ Vous n'avez aucun compte pour effectuer cette transaction."  
  
compte_source = comptes[0]  
id_compte_source = compte_source[0]  
id_compte_dest = session["compte_dest"][0]  
  
succes, message = effectuer_transaction(id_compte_source, id_compte_dest,  
montant)  
if not succes:  
    session[step_key] = None  
    session["compte_dest"] = None  
    return message
```

Description :

- **Déetecte les montants élevés pour prévenir la fraude.**
- **Enregistre la suspicion dans la base de données et notifie l'utilisateur.**
- **Si le montant est valide, la transaction est effectuée normalement.**

Gestion des étapes de conversation via Flask

```
session[step_key] = "dest" # Début de la saisie du compte destinataire
```

```
session[step_key] = "montant" # Saisie du montant à envoyer
```

Description :

- Permet de suivre le flux de la conversation avec chaque utilisateur.
- Chaque étape (compte destinataire, montant, confirmation) est enregistrée dans la session Flask.

Documentation des algorithmes et modèles

1. NLP et recherche par similarité :

- SpaCy (lg) pour précision accrue dans l'analyse des questions.
- Comparaison des questions posées avec les questions existantes dans la table FAQ.
- Seuil minimal pour retourner une réponse : similarité > 0.75.

2. Règles spécifiques :

- Listes de mots-clés pour détecter des intentions : solde, transactions, cartes bancaires, prêts, etc.
- Flux conversationnel basé sur les étapes (step) de la session Flask pour gérer chaque utilisateur individuellement.

3. Base de données :

- Tables principales : utilisateur, compte, transaction, faq, suspicion, conversation, carte, interaction.
- Index trigramme sur faq.question_faq pour accélérer la recherche textuelle.
- Gestion sécurisée des données utilisateurs et transactions.

4. Sécurité et performance :

- Mots de passe hachés (bcrypt).
- Sessions sécurisées avec Flask.
- Préchargement des historiques pour réduire le temps de rendu.
- Optimisation SQL et ajustement des seuils NLP pour réduire les erreurs et accélérer les réponses.

Fin

**Merci
pour
votre
aimable
attention**