Spécifications Techniques de l'Application Web pour INTIA Assurance

1. Introduction

Ce document présente les spécifications techniques pour le développement de l'application web de gestion des clients et des utilisateurs pour la société **INTIA Assurance**. L'application permettra de gérer les clients (ajout, modification, suppression et consultation des informations), ainsi que les utilisateurs (administrateurs et agents), tout en offrant une interface web pour interagir avec les données. Elle sera conçue en utilisant une architecture web moderne, stable, et scalable, garantissant ainsi une bonne expérience utilisateur et une efficacité opérationnelle.

2. Objectifs et Contexte

L'application a pour objectif de fournir une plateforme fiable pour le suivi et la gestion des clients d'**INTIA Assurance**. Cela inclut:

- L'ajout, la modification et la suppression des informations des clients.
- La consultation des informations des clients (affichées sous forme de liste ou de détails individuels).
- La gestion des utilisateurs avec des rôles d'admin et d'agent, et l'authentification via JWT (JSON Web Token).

3. Architecture du Système

3.1 Architecture Générale

L'application est basée sur une architecture **MVC** (Modèle-Vue-Contrôleur) avec un backend développé en **Python** utilisant le framework **Flask** et un frontend moderne. Les principales technologies et outils sont les suivants :

- **Backend**: Flask (Python)
- **Base de données** : SQLite (par défaut, mais elle peut être facilement changée en PostgreSQL ou MySQL)
- **Frontend** : HTML/CSS/Bootstrap pour une interface simple et reactive
- **Authentification** : Utilisation de **JWT** pour la gestion de l'authentification sécurisée des utilisateurs
- **Serveur** : Serveur local Flask en mode développement (en production, un serveur WSGI comme **Gunicorn** ou **Uvicorn** serait utilisé)
- **Déploiement** : Dockerisation pour faciliter le déploiement et l'intégration continue via **CI/CD avec GitHub Actions**

3.2. Flux de données et interactions

- **Backend** : Le backend gère toutes les requêtes HTTP, traite la logique métier et interagit avec la base de données. Flask gère les routes API et le traitement des données.
- **Frontend** : L'interface utilisateur est construite en HTML et CSS avec **Bootstrap** pour une présentation responsive. Elle est responsable de la collecte et de l'envoi des données via des formulaires.
- **Base de données** : La base de données stocke toutes les informations des clients et utilisateurs, permettant ainsi un accès rapide et structuré aux données. Nous utilisons **SQLAlchemy** pour interagir avec la base de données.

4. Modélisation de la Base de Données

4.1 Modèle Client

Le modèle **Client** représente les informations relatives aux clients d'INTIA Assurance. Les champs de la table sont les suivants :

Champ	Type	Description
id	INT (Primary Key)	Identifiant unique du client
nom	VARCHAR(255)	Nom du client
prenom	VARCHAR(255)	Prénom du client
email	VARCHAR(255) UNIQUE	Adresse email (unique)
telephone	VARCHAR(20)	Numéro de téléphone
adresse	TEXT	Adresse postale du client
succursale	ENUM('Douala', 'Yaoundé')	Succursale d'affiliation (Douala ou Yaoundé)

4.2 Modèle Utilisateur

Le modèle **Utilisateur** représente les informations relatives aux utilisateurs (agents et administrateurs). Les champs sont :

Champ	Type	Description
id	INT (Primary Key)	Identifiant unique
nom_utilisateur	VARCHAR(255) UNIQUE	Nom d'utilisateur (unique)
email	VARCHAR(255) UNIQUE	Email de connexion
mot_de_passe	VARCHAR(255)	Mot de passe hashé
role	ENUM('Admin', 'Agent')	Rôle de l'utilisateur

5. Fonctionnalités Clés

5.1 Gestion des Clients

Les principales fonctionnalités de gestion des clients sont les suivantes :

- **Création d'un client** : Un administrateur ou agent peut ajouter un client avec son nom, prénom, email, téléphone, adresse et succursale d'affiliation.
- **Modification des informations d'un client** : Modification des informations de contact ou d'affiliation d'un client existant.
- **Suppression d'un client** : Suppression complète d'un client de la base de données.
- **Consultation de la liste des clients** : Affichage d'un tableau listant tous les clients enregistrés.
- **Consultation des détails d'un client** : Affichage des informations détaillées d'un client individuel en cliquant sur son nom dans la liste.

5.2 Gestion des Utilisateurs

L'application inclut une gestion des utilisateurs permettant :

- **Authentification via JWT** : Gestion de l'inscription et de la connexion des utilisateurs via des tokens JWT pour sécuriser les routes sensibles.
- **Gestion des rôles d'utilisateur** : Les utilisateurs sont classés en deux rôles : **Administrateur** et **Agent**, avec des permissions différentes.

6. API REST

6.1 Endpoints Clients

- **POST /clients** : Créer un client
    - Corps de la requête : Données JSON du client
- **GET /clients** : Lister tous les clients
    - Réponse : Liste des clients avec leurs informations
- **GET /clients/{id}** : Détails d'un client
    - Réponse : Informations complètes du client avec l'id spécifié
- **PUT /clients/{id}** : Modifier un client
    - Corps de la requête : Données JSON modifiées du client
- **DELETE /clients/{id}** : Supprimer un client
    - Réponse : Confirmation de suppression

6.2 Endpoints Utilisateurs

- **POST /auth/signup** : Inscription d'un utilisateur
    - Corps de la requête : Données de l'utilisateur (nom, email, mot de passe)
- **POST /auth/login** : Connexion d'un utilisateur
    - Corps de la requête : Données de connexion (email, mot de passe)
- **GET /users** : Liste de tous les utilisateurs
    - Réponse : Liste des utilisateurs (agents et administrateurs)

7. Sécurité

- **Authentification et autorisation** : Utilisation de JWT pour protéger les routes sensibles. Les rôles (Admin/Agent) sont gérés et les permissions sont contrôlées.
- **Protection contre les attaques CSRF et CORS** : Mise en place de protections pour sécuriser les formulaires et les requêtes HTTP.
- **Hashage des mots de passe** : Utilisation de bibliothèques sécurisées pour le hashage des mots de passe des utilisateurs avant stockage dans la base de données.

8. Déploiement

- **Docker** : L'application sera contenue dans un conteneur Docker pour faciliter le déploiement sur différents environnements.
- **CI/CD** : Utilisation de **GitHub Actions** pour mettre en place un pipeline CI/CD, permettant de tester, builder et déployer l'application automatiquement à chaque modification du code source.