**Rôle et Objectif :** Tu es un architecte de solutions cloud expert, spécialisé dans la conception d'applications web pour le secteur de la restauration et de l'hôtellerie (RMS Lounge Bar). Ta mission est de proposer une architecture technique et fonctionnelle complète pour une application de gestion destinée à un restaurant lounge bar à Cotonou, Bénin. L'objectif principal et non négociable est de minimiser les coûts d'infrastructure en utilisant l'écosystème AWS.

Pour une approche plus agile, il faut me proposer une breakdown du projet en phases (MVP puis features avancées) et une estimation des coûts AWS mensuels basée sur l'architecture proposée.

**Contexte du Projet :**

* **Nom du projet :** Application de gestion pour "Le RMS Lounge Bar à Cotonou" (ADM-Lounge).
* **Type d'établissement :** Restaurant Lounge Bar.
* **Localisation :** Cotonou, Bénin.
* **Utilisateurs cibles :** Le gérant, les serveurs, le barman, le chef de cuisine, le comptable.
* **Actifs existants :**
  + Un nom de domaine : grouperms.com, géré via GoDaddy.
  + Un compte AWS actif.
* **Contrainte majeure :** L'architecture doit être pensée "serverless-first" pour réduire au maximum les coûts fixes. Le paiement doit être autant que possible lié à l'utilisation réelle (pay-as-you-go).

**Spécifications Fonctionnelles Détaillées :** Je souhaite que l'application intègre les modules suivants :

**1. Module de Ventes (Point of Sale - POS) :**

* Prise de commande intuitive sur tablette ou ordinateur.
* Gestion des tables et des zones (ex: Bar, Terrasse, VIP).
* Gestion des modes de paiement multiples : Espèces, TPE (carte bancaire), et **surtout Mobile Money (MTN MoMo, Moov Flooz)**.
* Division et fusion de factures.
* Génération de reçus (impression et/ou envoi par email/WhatsApp).
* Gestion des réductions et des offres spéciales.

**2. Module de Gestion des Stocks et Inventaire :**

* Gestion des articles : boissons, ingrédients pour les plats et cocktails.
* Décompte automatique des stocks en temps réel à chaque vente.
* Alertes de stock bas.
* Gestion des fournisseurs et des bons de commande.
* Fonctionnalité d'inventaire physique (comptage et ajustement des stocks).

**3. Module de Gestion Financière et Comptable :**

* **Tableau de bord principal** affichant les KPIs : Chiffre d'Affaires (CA) du jour, semaine, mois ; bénéfice brut ; panier moyen.
* **Suivi du Chiffre d'Affaires :** Rapports de ventes détaillés par article, par serveur, par période.
* **Gestion des Charges et Frais :**
  + Enregistrement de toutes les dépenses : salaires, loyer, achats fournisseurs, factures (SBEE, SONEB), marketing, taxes, etc.
  + Catégorisation des charges.
  + Possibilité d'attacher une photo du reçu à chaque dépense.
* **Gestion des Investissements :** Suivi des achats d'équipements et d'immobilisations.
* **Gestion de la Trésorerie (Ressources) :**
  + Visualisation en temps réel de la position des différentes caisses :
    - Caisse physique (espèces).
    - Compte bancaire principal.
    - Comptes Mobile Money (solde du compte MoMo de l'entreprise).
  + Enregistrement des flux entre ces différentes ressources (ex: dépôt des espèces à la banque, retrait MoMo vers la caisse).

**4. Module de Gestion du Personnel (RH) :**

* Fiches d'information des employés (nom, contact, poste, salaire).
* Gestion des rôles et des permissions (un serveur ne voit pas la comptabilité, le gérant voit tout).
* Gestion simple des plannings de travail.
* Suivi du temps de travail (pointage basique).

**Spécifications Techniques et Architecturales (AWS) :**

1. **Architecture Backend :** Proposer une architecture 100% serverless.
   * **API :** Utiliser **Amazon API Gateway** (RESTful ou HTTP API).
   * **Logique Métier :** Utiliser des fonctions **AWS Lambda** (en Node.js ou Python de préférence).
   * **Base de Données :** Proposer la solution la plus économique et scalable. Comparer **Amazon DynamoDB** (NoSQL, souvent le moins cher pour ce type de cas avec son modèle de paiement à la requête) et **Amazon RDS avec Aurora Serverless v2** (relationnel, si la structure des données l'exige). Justifier le choix.
   * **Authentification :** Utiliser **Amazon Cognito** pour gérer les utilisateurs et les mots de passe de manière sécurisée et à faible coût.
2. **Architecture Frontend :**
   * Proposer un framework JavaScript moderne comme **React.js, Vue.js ou Svelte**.
   * L'application doit être une **Progressive Web App (PWA)** pour être accessible depuis n'importe quel navigateur (PC, tablette, smartphone) et fonctionner hors ligne pour les prises de commande.
   * **Hébergement :** Héberger le site statique sur **Amazon S3** et le distribuer via **Amazon CloudFront** pour la performance et le cache, ce qui est l'option la plus économique.
3. **Intégration et Déploiement :**
   * Expliquer comment connecter le domaine grouperms.com à l'infrastructure AWS (via **Amazon Route 53**).
   * Suggérer un pipeline CI/CD simple pour automatiser les déploiements (par exemple avec **AWS Amplify** ou **AWS CodePipeline**).

**Livrables Attendus :** Je souhaite que tu me fournisses une réponse structurée en plusieurs parties :

1. **Résumé de l'Architecture Technique :** Un schéma simple ou une description textuelle claire de l'interaction entre les services AWS proposés (Frontend -> CloudFront -> API Gateway -> Lambda -> DynamoDB/RDS).
2. **Choix des Services AWS :** Une liste des services recommandés avec une justification claire pour chacun, axée sur la **minimisation des coûts**.
3. **Modèle de Données :** Une ébauche des principaux modèles de données (ex: tables Vente, Produit, Employe, Charge, RessourceFinanciere) pour la base de données choisie.
4. **Estimation Qualitative des Coûts :** Sans donner un chiffre exact, expliquer les principaux postes de coût (ex: nombre d'appels API, Go de stockage, temps de calcul Lambda) et comment l'architecture proposée les maintient au plus bas pour une activité de démarrage.
5. **Plan d'Action Simplifié :** Les grandes étapes pour développer et déployer cette application, de la configuration d'AWS au déploiement final.

**Fin du Prompt**