Cahier des Charges - Chacun Bouffe à Son Tour (CBT)

I. Introduction

Ce document décrit les exigences fonctionnelles et non fonctionnelles pour le développement de l'application de gestion de tontine "Chacun Bouffe à Son Tour (CBT)". L'objectif principal est de développer une application de gestion de tontine en mettant l'accent sur la manipulation directe de SQL et de la base de données.

II. Objectif Général

Développer une application desktop pour gérer les tontines, en maximisant la manipulation directe de SQL et de la base de données. L'objectif pédagogique est de se concentrer sur les requêtes SQL complexes, les transactions DB, les jointures, les triggers, les procédures stockées.

Le frontend est implémenté en Java, avec JDBC pour les interactions DB. L'application gère les membres, les groupes, les cotisations, les prêts, les votes, les sanctions, et les rôles administratifs.

III. Contexte et Principes de la Tontine

Une tontine est un système d'épargne collective où les membres contribuent régulièrement. Le principe de fonctionnement est le suivant : un minimum de membres par groupe (min 3). Les contributions sont collectées, et les fonds sont utilisés pour des distributions rotatives, des prêts, ou des activités collectives. Les types de tontines

Peuvent inclure des tontines de présence et les tontines opérationnelles.

Rôles clés :

- Membres standards : Inscription, connexion, visualisation de listes, soumission

  de propositions, visualisation de notifications et sanctions.
- Bureau (administratif) : Composé du Président, Gestionnaire Financier, Secrétaire, Censeur.
- Gestionnaire Financier : Gère la caisse, enregistre cotisations/prêts, calcule remboursements automatiques, gère entrées/sorties, rapports/indicateurs financiers.
- Secrétaire : Gère informations importantes, portail d'infos, rapports.
- Censeur : Applique sanctions/pénalités pour non-paiement, exclusions,

applique code de pénalité.

- Accès restreint : Les membres n'ont pas accès aux fonctionnalités du bureau.
- Langages : Java (frontend et backend logique), SQL (pour toutes les interactions

DB).

- Base de Données : MySQL (manipulation maximale : requêtes directes, jointures,

sous-requêtes, triggers pour audits/automatisations, procédures stockées pour calculs comme remboursements).

- Environnement : Développement sous Ubuntu (Linux).

Diagrammes Requis :

Diagramme de classes.

Diagramme de séquences.

Diagramme de cas d'utilisation.

Équipe : développeurs répartis en groupes de 3 personnes chacun. Chaque groupe se concentre sur un module, avec collaboration pour les intégrations DB.

- √ Groupe1 : Modélisation globale + Membres & Authentification.
- √ Groupe 2 : Groupes, Séances & Cotisations de Présence.
- √ Groupe 3 : Tontines Optionnelles & Tours.
- √ Groupe 4 : Crédits & Pénalités.
- √ Groupe 5 : Projets Collectifs, Votes & Rapports Globaux.

Portée : Application desktop (pas web/mobile).

Focus DB : Au moins 70% des fonctionnalités impliquent des manipulations SQL directes (e.g., inserts pour cotisations, selects avec jointures pour listes, updates pour sanctions).

Sécurité basique : Hashage des mots de passe en DB, sessions utilisateur.

- Exigences Fonctionnelles

basées sur les notes fournies, organisées par module.

- Gestion des Utilisateurs (Membres)

Inscription : Formulaire pour créer un compte (nom, email, mot de passe, rôle

potentiel). SQL : INSERT INTO users.

- Connexion : Authentification. SQL : SELECT avec vérification hash.

Visualisation : Liste des membres du groupe (jointure users-groups). Infos personnelles, historique sessions.

- notifications : Soumettre propositions, voir sanctions en cours. SQL : Triggers

pour notifier sur updates.

- Audit/Co-session : Logs d'activités (SQL : Table logs avec inserts sur actions).

Gestion des Groupes :

Création : Par président/admin. min membres. SQL : INSERT INTO groups, puis jointures.

Invitation : Envoyer/Accepter invitations. SQL : Table invitations avec status.

Cotisations : Valeur fixe/période (e.g., mensuelle). Enregistrer paiements. SQL : Table cotisations avec sums agrégées.

Type de Tontine : Solidaire (prêts inclus). SQL : Champ type dans groups.

Gestion Financière (Caisse)

Cotisations : Enregistrer, compléter. SQL : Procédure stockée pour calcul total.

Prêts : Demander, approuver, rembourser (calcul auto des restants). SQL :

Triggers pour updates sur remboursements.

Entrées/Sorties : Gérer flux (banque, audits). SQL : Jointures pour rapports.

Indicateurs : Soldes, histo prêts. SQL : Vues pour queries complexes.

Votes et Propositions

Créer Vote : Par président (e.g., pour activités/prêts).

Voter : Accepter/Refuser propositions. SQL : Table votes avec counts.

Soumettre Proposition : Par membres. SQL : INSERT, puis jointure pour status.

Rôles Administratifs et Sanctions

Président : Créer activités, gérer votes.

Gestionnaire Financier : Gérer caisse, prêts.

Censeur : Appliquer pénalités (e.g., sanctions pour non-paiement), exclusions.

SQL : Updates sur users avec flags.

Secrétaire : Portail infos, rapports.

Interface Utilisateur (Frontend Java Swing)

Page Connexion : Basique.

Dashboard Membre : Liste membres, notifications, propositions.

Dashboard Bureau : Selon rôle (e.g., prêts pour financier).

Historique : Sessions précédentes.

Sanctions : Voir/appliquer.