BankSim :

Dossier de conception général

Référence : CCG-14030602-0A
Fournisseur
Date : 15/02/2024
Version/Édition : 0A
État : Préliminaire

Type de diffusion : Diffusion restreinte
Autre référence :

FICHE DE SUIVI DES AUTORISATIONS ET DIFFUSIONS

AUTORISATIONS PRESTATAIRE

	Fonction	Nom	Date	Visa
Auteur	Directeur de projet	Paul Bouglé		
Validé par	Architecte logiciel	Alexandre Bardiot		
Vérifié par				
Vérifié par				
Approuvé par				

AUTORISATIONS CLIENT

	Fonction	Nom	Date	Visa
Approuvé par				
Approuvé par				
Approuvé par				

DIFFUSION INTERNE

Nom	Fonction	Action	Date	Nb exemplaire(s)
Paul Bouglé	Directeur de Projet			
Alexandre Bardiot	Architecte logiciel			

DIFFUSION EXTERNE

Nom	Fonction	Action	Date	Nb exemplaire(s)
Yves Jehanno	Client			

Historique des révisions

Date	Description et justification de la modification	Auteur	Pages / Chapitre	Edition / Révision
06/03/2014	Création		Toutes	0A

Table des matières

1 Introduction

1.1 Objet du document

1.2 Objet du document

La conception générale d'un logiciel est une démarche rationnelle, qui consiste à réduire la complexité initiale en la décomposant en constituants de niveau inférieur.

La conception permet de préciser *comment* vont être réalisées les spécifications.

La conception préliminaire permet d'élaborer une solution technique répondant aux spécifications. Elle précise l'architecture de cette solution (interfaces internes, constituants principaux, ...), ainsi que son comportement dynamique (logique d'enchaînements, diagrammes d'état, parallélisme, synchronisation, ...).

L'architecture du matériel cible est présentée dans la conception générale si cela n'a pas été fait dans un autre document.

Il faut présenter succinctement la structure du document :

- description et analyse des ressources matérielles et logicielles,
- diagramme des catégories ou des classes principales,
- structure statique (interfaces externes, interfaces internes, ...),
- structure dynamique (synchronisation entre tâches, séquencement des traitements, passages entre les différents états ou mode de traitement, ...),
- implémentation sur le matériel cible (allocation des ressources CPU mémoire bande passante, adresse des périphériques, couche basse de protocoles spécifiques, ...).

La conception préliminaire doit permettre de réaliser le dossier d'intégration qui décrit les étapes permettant de vérifier que tous les composants identifiés au niveau de la conception préliminaire s'intègrent bien ensemble.

La revue de conception vérifie que tout le contenu des spécifications a bien été pris en compte par la conception préliminaire.

La conception générale peut être réalisée avec plus ou moins de détail en fonction de la taille du projet. Pour un *petit projet*, la conception générale est suffisante et ainsi le niveau de détail est le plus fin. Pour un *projet moyen*, la conception générale est suivie d'une conception détaillée. L'objectif de la conception générale est donc d'affecter chacune des données et des traitements définis dans les cas d'utilisations de la spécification à des classes. Les classes, attributs et opérations nécessaires à l'implémentation, mais inutiles pour la compréhension générale sont ignorés.

Pour un *gros projet*, la conception générale est suivie d'une ou plusieurs conceptions détaillées. Le rôle de la conception générale n'est pas de rentrer dans les détails, mais de présenter l'architecture générale du logiciel et de déterminer clairement le domaine de responsabilité de chaque catégorie ou classe à la fois en terme de données et de traitements vis à vis des cas d'utilisation et aussi en terme d'opérations que chaque développeur doit mettre au service des autres. La conception générale organise un découpage pour des développements parallèles.

1.3 Responsabilités

La rédaction de la conception générale est de la responsabilité du Chef de projet. Il juge de son état complet et décide de sa présentation en revue de conception.

La conception générale est souvent livrable, elle permet de décrire de façon complète le travail à réaliser.

1.4 Evolution

La conception générale fait partie de la référence de réalisation du système; toute modification de cette référence intervenant après le prononcé de revue de conception doit être traitée comme une demande d'évolution.

Quand dans la phase de développement des détails d'implémentation apparaissent, l'évolution du document de conception générale évolue en fonction des types de projets.

Pour un *petit projet*, la seule référence étant la conception générale, celle-ci doit être mise à jour.

Pour un *projet moyen*, les détails d'implémentation relèvent de la conception détaillée, sauf si les évolutions concernent des interfaces externes et ont donc fait l'objet d'une demande d'évolution.

Pour un *gros projet*, il en est de même, sauf que les évolutions des interfaces internes doivent également être prises en compte. La conception générale évolue que si les modifications ont un impact sur l'architecture du logiciel.

1.5 Outils utilisés

Les documents de base sont rédigés avec la suite bureautique Office, Rational Rose sous MS-Windows ou les outils LyX et Dia sous Unix. D'autres outils peuvent être utilisés dans le cadre du projet mais les règles énoncées ci-après restent valables sur toute la durée du projet.

Les schémas présentés respectent les conventions UML.

2 Terminologie

2.1 Abréviations

UML	**U**nified **M**odeling **L**anguage

2.2 Définitions des termes employés

attribut	un attribut est une information caractéristique mémorisée par un objet.
cas d'utilisation	cas d'utilisation du système, par extension il représente également la technique de modélisation mise en oeuvre dans UML (use case).
catégorie	une catégorie consiste en un regroupement logique de classes à forte cohérence interne et faible couplage externe, associée au concept UML de package. Ce concept permet une présentation plus synthétique du diagramme des classes d'un système réel.
classe	une classe définit un ensemble d'objets similaires potentiels. Elle fournit le modèle de la structure et les possibilités de chaque objet.
objet	un objet est une instance d'une classe, c'est une entité informatique unique possédant ses propres attributs et opérations
opération ou méthode	une opération est un traitement spécifique qu'un objet est en charge de fournir.
tâche	une tâche représente un élément manipulé par le système et ordonnançable de manière individuelle. cela peut représenter un process d'un système Unix, une tâche d'un moniteur temps-réel, un thread d'une application.
threads	codes exécutés de manière concurrente par le système d'exploitation mais partageant le même espace mémoire.

3 Description et analyse de l'environnement

3.1 Architecture matérielle

L'application de simulation de transaction par carte bancaire fonctionne sur un environnement matériel standard d'un ordinateur personnel. Aucune architecture matérielle spécifique n'est requise

3.2 Description des ressources matérielles

Les ressources matérielles utilisées pour exécuter l'application comprennent un ordinateur

personnel standard doté d'un processeur et de la mémoire suffisants pour exécuter une application Java.

Aucun autre équipement matériel spécifique n'est nécessaire.

3.3 Description des ressources logicielles

- **Système d'exploitation :** L'application est compatible avec les systèmes d'exploitation courants tels que Windows, macOS et Linux.
- **Java :** L'application est développée en utilisant le langage de programmation Java. Java Development Kit (JDK) doit être installé sur le système pour exécuter l'application.
- **JavaFX :** Les interfaces utilisateur de l'application sont développées à l'aide de la bibliothèque JavaFX. JavaFX est inclus dans le JDK depuis Java 8.
- **Gradle :** Le système de construction Gradle est utilisé pour compiler, tester et exécuter l'application. Le fichier `build.gradle` spécifie les dépendances et les tâches de construction.
- **SQLite JDBC Driver :** Pour la gestion de la base de données SQLite, le pilote JDBC SQLite est utilisé. Le fichier JAR `sqlite-jdbc-3.34.0.jar` est inclus dans le répertoire `lib`.Interfaces externes

L'application fonctionne en local sur l'ordinateur de l'utilisateur et n'a pas de dépendances externes. Aucune interface externe n'est requise pour son fonctionnement.

3.4 Organisation de l'espace de travail

Le code source de l'application est organisé dans le répertoire app/src. Les fichiers Java sont situés dans app/src/main/java, les ressources telles que les fichiers SQL sont situées dans app/src/main/resources. Les tests unitaires sont situés dans app/src/test.

Le système de construction Gradle est configuré dans le fichier build.gradle, et les tâches de construction sont exécutées à l'aide des scripts Gradle. Les résultats de la construction sont générés dans le répertoire app/build.

Arbre de structure du project BankSim :

```
app
├── build
│
├── build.gradle
└── src
    ├── main
    │   ├── java
    │   │   └── Banksim
    │   │       ├── App.java
    │   │       ├── Back
    │   │       │   ├── assets
    │   │       │   │   ├── BankAccount.java
    │   │       │   │   ├── Bank.java
    │   │       │   │   └── Card.java
    │   │       │   ├── bin
    │   │       │   ├── database
    │   │       │   │   └── DatabaseManager.java
    │   │       │   ├── Main.class
    │   │       │   ├── servers
    │   │       │   │   ├── AcquisitonServer.java
    │   │       │   │   ├── AuthorizationServer.java
    │   │       │   │   └── InterbankNetwork.java
    │   │       │   └── tpe
    │   │       │       └── Terminal.java
    │   │       └── Screen
    │   │           ├── BankScreen.java
    │   │           ├── MainScreen.java
    │   │           ├── TerminalScreen.java
    │   │           └── TransactionScreen.java
    │   └── resources
    │       └── sql
    │           ├── database_Banque A.db
    │           ├── database_Banque B.db
    │           └── database_giecb.db
    └── test
```

Remarques:
- build n’est pas développé car ils contient des informations relatives à Gradle pour la construction (gestion des dépendances et compilation des fichiers).
-bin non plus car il contient les .class des .java
-test est à l’identique de Banksim mais chaque fonction est la fonction de test associée.

4 Structure statique

4.1 Décomposition générale

La décomposition générale du projet peut être présentée comme suit :

- **Banksim** : Package racine du projet.
  - **Back** : Package contenant les composants liés à la logique métier et aux données.
    - **assets** : Package contenant les classes représentant les actifs du système.
    - **database** : Package contenant les composants de gestion de la base de données.
    - **servers** : Package contenant les serveurs applicatifs des banques et du GieCB.
    - **tpe** : Package contenant le terminal de paiement électronique.
  - **Screen** : Package contenant les classes représentant les écrans de l'interface utilisateur.
- **resources** : Répertoire contenant les ressources de l'application.
  - **sql** : Répertoire contenant les scripts SQL et les bases de données.

4.2 Allocations fonctionnelles et spécifications

Les grandes classes permettant la simulation :

Classe Terminal :

- **Rôle** : Cette classe représente un terminal de paiement électronique (TPE) dans le système. Elle est utilisée pour effectuer des transactions avec les cartes bancaires.
- **Lien avec les cas d'utilisation** :
  - Elle permet de faire la liaison entre le client et le réseau bancaire via son interface physique.

Classe Bank :

- **Rôle** : Cette classe représente une banque dans le système. Elle gère la création et la gestion des comptes bancaires, ainsi que le routage des transactions.
- **Lien avec les cas d'utilisation** :
  - Elle est responsable de la validation des requêtes de paiement grâce à sa base de données, ce qui correspond à la fonctionnalité d'**Authentification et de Validation.**
  - Elle est aussi en charge des déplacement de fonds et de leur suivi, correspondant aux fonctionnalités **Transactions et Sauvegarde.**

Classe InterBankNetwork:

- **Rôle** : Cette classe représente le réseau interbancaire dans le système (GieCB). Elle gère le routage des transactions entre les banques.
- **Lien avec les cas d'utilisation** :
  - Elle est responsable de la création d'une nouvelle banque et de son intégration dans le réseau via sa base de données correspondant à lafonctionnalité **Création et Suivi des Banques**.
  - Elle est responsable du routage des transactions vers les banques appropriées, ce qui est lié à la fonctionnalité de **Authentification et Autorisation**.
  - Elle peut être impliquée dans la fonctionnalité de **Suivi des Transactions** en enregistrant des informations sur le routage des transactions pour analyse ultérieure.

Ces classes interagissent ensemble à l'aide de classes secondaires pour fournir les fonctionnalités principales du système de simulation de transaction par carte bancaire, telles que décrites dans les spécifications.

4.3 Analyse statique

4.3.1 **Classe Terminal**

4.3.1.1 **Définition**

La classe Terminal représente un terminal de paiement électronique (TPE) dans le système de simulation de transactions bancaires par carte. Son rôle principal est de connecter une carte bancaire au réseau bancaire pour effectuer le paiement.

4.3.1.2 **Attributs**

- bank : Référence vers l'objet Bank associé au terminal.
- accountID : Identifiant du compte bancaire sur lequel la transaction sera effectuée.
- amount : Montant de la transaction.

4.3.1.3 **Opérations**

Les opérations offertes par la classe Terminal comprennent :

- **Terminal(Bank, accountID)** : Constructeur de la classe Terminal.
  - **Entrées** : Objet Bank représentant la banque associée au terminal, identifiant du compte bancaire (accountID) sur lequel la transaction sera effectuée.
  - **Sortie** : Instance de la classe Terminal.
- **getAmount()** : Récupère le montant de la transaction.
  - **Sortie** : Montant de la transaction.
- **setAmount(amount)** : Définit le montant de la transaction.
  - **Entrée** : Montant de la transaction (amount).
- **processTransaction(Card)** : Traitement de la transaction sur le terminal.
  - **Entrée** : Objet Card représentant la carte bancaire utilisée dans la transaction.
  - **Sortie** : Résultat de l'autorisation de la transaction.

4.3.2 **Classe Bank**

4.3.2.1 **Définition**

La classe Bank représente une banque dans le système de simulation de transactions bancaires par carte. Son rôle principal est de gérer les comptes bancaires, de router les transactions vers d'autres banques via le réseau interbancaire et de traiter les transactions financières.

4.3.2.2 **Attributs**

- DATABASE_PATH : Chemin vers la base de données de la banque.
- ID : Identifiant unique de la banque.
- Name : Nom de la banque.

4.3.2.3 **Opérations**

Les opérations offertes par la classe Bank comprennent :

- **newBank(DATABASE_PATH, ID, Name)** : Crée une instance de la classe Bank, certifiant son unicité
  - **Entrées** : Chemin vers la base de données (DATABASE_PATH), identifiant de la banque (ID), nom de la banque (Name).
  - **Sortie** : Instance de la classe Bank (déjà existante ou nouvelle).
- **getBank(ID)** : Récupère une instance de la banque à partir de son identifiant.
  - **Entrée** : Identifiant de la banque (ID).
  - **Sortie** : Instance de la classe Bank.
- **getID()** : Récupère l'identifiant de la banque.
  - **Sortie** : Identifiant de la banque.
- **getName()** : Récupère le nom de la banque.
  - **Sortie** : Nom de la banque.
- **getDatabasePath()** : Récupère le chemin vers la base de données de la banque.
  - **Sortie** : Chemin vers la base de données.
- **setName(Name)** : Définit le nom de la banque.
  - **Entrée** : Nom de la banque (Name).
- **setDatabasePath(path)** : Définit le chemin vers la base de données de la banque.
  - **Entrée** : Chemin vers la base de données (path).
- **routeRequest(Card, AccountID)** : Routage de la demande de transaction.
  - **Entrées** : Objet Card représentant la carte bancaire utilisée dans la transaction, identifiant du compte (AccountID) où la transaction sera effectuée.
  - **Sortie** : Résultat de l'autorisation de la transaction.
- **processBuyerTransaction(card)** : Traitement de la transaction côté acheteur.
  - **Entrée** : Objet Card représentant la carte bancaire utilisée dans la transaction.
  - **Sortie** : Résultat du traitement de la transaction.
- **getRandomCard()** : Récupère une carte bancaire aléatoire.
  - **Sortie** : Objet Card représentant une carte bancaire aléatoire.
- **getRandomAccountID()** : Récupère un identifiant de compte bancaire aléatoire.
  - **Sortie** : Identifiant de compte bancaire aléatoire.
- **initializeDatabase()** : Initialise la base de données de la banque avec des comptes bancaires aléatoires.

4.3.3 **Classe InterbankNetwork**

4.3.3.1 **Définition**

La classe InterbankNetwork représente le réseau interbancaire dans le système de simulation de transactions bancaires par carte. Son rôle principal est de router les demandes de transaction entre les différentes banques et de gérer le processus d'autorisation des transactions.

4.3.3.2 **Attributs**

- instance : Instance unique de la classe InterbankNetwork.
- DATABASE_PATH : Chemin vers la base de données du réseau interbancaire.
- CurrentCode : Code actuel utilisé pour la création d'identifiants de banque.

4.3.3.3 **Opérations**

Les opérations offertes par la classe InterbankNetwork comprennent :

- **getInstance()** : Récupère l'instance unique de la classe InterbankNetwork.
    - **Sortie** : Instance de la classe InterbankNetwork.
- **routeRequest(Card)** : Routage de la demande de transaction vers la banque appropriée.
    - **Entrée** : Objet Card représentant la carte bancaire utilisée dans la transaction.
    - **Sortie** : Résultat de l'autorisation de la transaction.
- **extractBankID(Card)** : Extrait l'identifiant de la banque à partir du numéro de carte.
    - **Entrée** : Objet Card représentant la carte bancaire utilisée dans la transaction.
    - **Sortie** : Identifiant de la banque.
- **findBuyerBank(bankID)** : Recherche et récupère la banque acheteuse à partir de son identifiant.
    - **Entrée** : Identifiant de la banque.
    - **Sortie** : Objet Bank représentant la banque acheteuse.
- **createBank(bankName)** : Crée une nouvelle instance de la classe Bank ou récupère la banque déjà existante avec cette ID.
    - **Entrée** : Nom de la banque.
    - **Sortie** : Instance de la classe Bank.
- generatePath(String Name) : Génère le chemin vers la base de données d'une banque à partir de son nom.
    - **Entrée** : Nom de la banque.
    - **Sortie** : Chemin vers la base de données.
- **initializeDatabase()** : Initialise la base de données du réseau interbancaire composé de la table Banks.

5 Aspects dynamiques

5.1 Description des tâches

- Terminal :
  - Gestion locale de la transaction et routage vers la banque du commerçant
  - Affiche le résultat de la transaction dans le réseau bancaire
- Banque commerçant :
  - Routage vers le GieCB
  - Retour vers le terminal une fois l'opération effectué
  - Ajout des fonds vers le compte commerçant lors d'une transaction nominale
- GieCB :
  - Recherche de la banque à créditer et routage
  - Retour vers la banque du commerçant une fois le routage effectué
- Banque client :
  - Recherche du compte à créditer
  - Retour vers le GieCB lors d'une transaction nominale

5.2 Comportement du logiciel en erreur

En cas d'erreur lors de la transaction, le terminal reçoit un code d'erreur qu'il retranscrit sur son interface graphique. L'erreur peut provenir dans cette ordre :

- Terminal
- GieCB
- Banque client
- Banque commerçant.

5.3 Modes de fonctionnement

Dans le cas d'une erreur, le Terminal se contente de montrer le code d'erreur et lemessage associé expliquant la cause. En interne, la transaction est avortée mais les requêtes sont sauvegardées.

6 Allocation des ressources

6.1 Répartition des tâches

- Carte Bancaire : Possède un numéro relié à son propriétaire et un code PIN pour la vérification physique.
- Terminal : possède les informations de la banque du commerçant.
- Banques : Elles possèdent leur base de données client et le chemin pour joindre le GieCB
- InterBankNetwork (GieCB) : possède une base de données comprenanttoutes les banques existantes

6.2 Gestion de la mémoire

- Les bases de données sont stockés dans le dossier « resources » dans le code source

- Le dossier « lib » contient le driver SQLite JDBC pour permettre l'utilisation des bases de données.
- Les autres dépendances telles que javaFX sont gérées par Gradle.