



Guide d'utilisation de l'application "IUT BANK"

Réalisé par Melosso Tom, Idoux Clément, Riese Tom, Sabatier Romain

I) Introduction.....	2
II) Installation du projet.....	3
1) Prérequis.....	3
2) Installation et configuration.....	3
III) Test unitaires.....	4

I) Introduction

Ce projet consiste à maintenir et améliorer l'application bancaire du département informatique de l'IUT de Metz. L'application actuelle contient des bugs et n'a pas de documentation.

Le but est donc de :

- Faire fonctionner l'application dans un environnement de développement.
- Corriger les bugs et ajouter les fonctionnalités demandées.
- Fournir une documentation claire pour l'installation, l'utilisation et la maintenance.
- Mettre en place des tests et des outils pour assurer la qualité du code.

Cette documentation a donc pour objectif de guider les futurs utilisateurs et développeurs dans l'installation, l'utilisation et la maintenance de l'application, tout en fournissant une vision complète de sa structure et de ses fonctionnalités.

II) Installation du projet

1) Prérequis

Avant de commencer, assurez-vous d'avoir installé :

- Java JDK 11
- Apache Tomcat 9
- IntelliJ IDEA Ultimate
- Wamp

2) Installation et configuration

1. Installer IntelliJ IDEA

- Télécharger IntelliJ IDEA : [lien officiel](#)

2. Cloner le projet

- Avec Git :

```
git clone https://github.com/TomRiese57/StickSpin.git
```

- Ou directement via le bouton "**Clone repository**" dans IntelliJ IDEA

2. Mise en place de la base de données

- Lancer Wamp et connecter vous à PHPMyAdmin : [ici](#)
- Créer la première base de données "bank_iut_prod", puis importer "dumpSQL.sql" (disponible dans le répertoire script du projet)
- Créer la seconde base de données "bank_iut_test", puis importer "dumpSQL_JUnitTest.sql"

(N'oubliez pas de garder Wamp ouvert lors de l'utilisation du site)

3. Installer le JDK

- Aller dans **File** → **Project Structure** → **SDK**
- Installer **JDK 11**

4. Compiler le projet en .war

- Aller dans **Run** → **Edit Configurations** → **New** → **Maven [Clean install]**
- Exécuter **Maven [clean install]**

5. Télécharger TomCat 9.0.10

- Télécharger TomCat : [lien officiel](#)

6. Configurer TomCat et lancer le serveur

- Aller dans **Run** → **Edit Configurations** → **New** → **Tomcat Server (local)**
- **Before launch** → ajouter **Build Artifact** → sélectionner **_00_ASBank2023.war**
- **Deployment** → ajouter **Artifact** → sélectionner **_00_ASBank2023.war exploded**
- **Run** → **TomCat**

III) Test unitaires

Voici l'ensemble des tests unitaires réalisés au lancement de l'application, qui vérifient son bon fonctionnement :

1. Mot de passe :

- **Hash non nul** : vérifie que le hash généré n'est jamais vide.
- **Longueur du hash** : vérifie que le hash SHA-256 fait bien 64 caractères.
- **Vérification mot de passe correct** : le mot de passe initial correspond au hash.
- **Vérification mot de passe incorrect** : un mot de passe différent est rejeté.

2. Base de données :

- **Récupération de compte** : vérifie qu'un compte existant peut être récupéré et qu'un compte inexistant renvoie null.
- **Type de compte** : teste que les comptes récupérés sont bien `CompteSansDecouvert` ou `CompteAvecDecouvert`.
- **Création de compte** : teste la création de comptes avec ou sans découvert, et que la création échoue si l'ID existe déjà.
- **Suppression de compte** : vérifie qu'un compte peut être supprimé et n'est plus récupérable après suppression.
- **Récupération d'utilisateur** : teste la récupération d'un utilisateur existant ou inexistant, et la distinction entre `Client` et `Gestionnaire`.
- **Création d'utilisateur** : vérifie la création de clients et de gestionnaires, et que la création échoue si l'ID existe déjà.
Suppression d'utilisateur : vérifie qu'un utilisateur peut être supprimé et n'est plus récupérable après suppression.
- **Vérification des droits** : teste la méthode `isUserAllowed` pour différents cas : login/mot de passe correct, mot de passe incorrect, utilisateur inexistant, login ou mot de passe vide ou null.

3. Manager de la banque:

- **Création de client** : vérifie qu'un client peut être créé correctement et que la création échoue si un compte avec le même numéro existe.
- **Suppression de compte avec découvert** : teste la suppression réussie si le solde est zéro, et l'échec si le solde est différent de zéro.
- **Suppression de compte sans découvert** : teste la suppression réussie si le solde est zéro, et l'échec si le solde est différent de zéro.
- **Suppression d'utilisateur** : vérifie que la suppression réussit si l'utilisateur n'a pas de comptes ou si ses comptes ont un solde nul.
- **Suppression interdite** : teste que la suppression du dernier gestionnaire ou d'un client avec comptes non nuls échoue correctement.

4. Client :

- **Vérification du format d'ID client** : teste différents cas valides et invalides pour l'identifiant utilisateur (userId), comme la présence de lettres, chiffres, points, caractères spéciaux, ou chaînes vides.
- **Vérification du format du numéro client** : teste que le numéro client contient exactement 10 chiffres et ne comporte ni lettres ni caractères spéciaux.
- **Possède des comptes à découvert** : vérifie que la méthode `possedeComptesADecouvert` renvoie correctement true ou false selon que le client possède des comptes avec ou sans découvert.
- **Comptes avec solde non nul** : teste que `getComptesAvecSoldeNonNul` retourne uniquement les comptes dont le solde est supérieur à zéro, en gérant différents types de comptes et combinaisons.

5. Compte :

- **Méthode créditer** : teste que le solde est correctement incrémenté avec un montant positif et que la méthode lève une exception pour un montant négatif.
- **Constructeur de compte** : vérifie que le constructeur lève une exception si le numéro de compte est au mauvais format.
- **Vérification du format du numéro de compte** : teste plusieurs cas valides et invalides, incluant le nombre de lettres au début, la présence de lettres au milieu ou à la fin, et le nombre exact de chiffres.

5. Compte avec découvert :

- **Vérification de la classe** : confirme que l'objet est bien de type `CompteAvecDecouvert`.
- **Méthode debiter** : teste que la méthode lève une exception pour un montant négatif, permet un débit réalisable selon le découvert autorisé, et renvoie une exception si le débit dépasse le découvert autorisé.

6. Compte sans découvert :

- **Vérification de la classe** : confirme que l'objet est bien de type `CompteSansDecouvert`.
- **Méthode debiter** : teste que la méthode lève une exception pour un montant négatif, permet un débit réalisable selon le solde disponible, et renvoie une exception si le débit dépasse le solde.

