BTS SIO SLAM

Session 2025

Ateliers de Professionnalisation

API de virements et Application de gestion et suivi des virements

Table des matières

PRÉSENTATION DE L'ENTREPRISE "CRÉDIT GÉNÉRAL BANK"

Partie 1 : Le contexte d’entreprise :

Introduction

Le Crédit Général est une banque française, innovante et dynamique. Elle est spécialisée dans les services financiers et les solutions technologiques pour les entreprises. Forte d'une expertise solide dans le domaine bancaire et de l'informatique, elle propose une large gamme de services destinés à optimiser la gestion financière des entreprises clientes. Parmi ses clients elle compte, LMHV, BATIPRO, mais aussi Galaxy Swiss Bourdin (GSB) depuis Début 2025.

Organisation et Gouvernance

Le Crédit Général est structuré en plusieurs pôles stratégiques, pôles que l’on retrouve par ailleurs dans la plupart des grandes banques de crédits :

1. Pôle Finance

Un pole finance dont les activités principales sont :
- la gestion des comptes clients, entreprises et grands comptes.
- le conseil et l’apport de solutions de financement et de crédit.
- Le suivi des opérations bancaires et la mise en place de systèmes de prévention des fraudes.

2. Pôle Technologique et Numérique

Le pôle technologique et numérique de Crédit Générale est structuré autour de plusieurs départements spécialisés :

Département Développement : Il se concentre sur la conception d'applications bancaires innovantes et sur le développement de solutions API. Ce département joue un rôle clé dans l'amélioration continue des services numériques offerts aux clients.

Département Cybersécurité : Il est responsable de la sécurisation des transactions et de la détection des fraudes. Grâce à des technologies avancées, il assure la protection des données sensibles et garantit la confiance des clients dans les services bancaires.

Département Data & Analytics : Il exploite les données financières pour fournir une aide à la décision. En utilisant des outils d'analyse avancés, il permet à Crédit Générale de proposer des insights précieux et d'optimiser ses stratégies financières.

3. Pôle Service aux Entreprises

Le pôle service aux entreprises de Crédit Générale accompagne les entreprises dans leur gestion financière. Il propose des offres de solutions sur mesure pour la gestion des paiements, des virements, et des relances. Un service client dédié est disponible

pour répondre aux besoins spécifiques des entreprises partenaires, assurant ainsi un soutien personnalisé et efficace.

4. Pôle Juridique et Conformité

Le pôle juridique et conformité de Crédit Générale assure une veille réglementaire rigoureuse et met en œuvre les mesures nécessaires pour se conformer aux normes bancaires en vigueur. Il réalise des audits internes et contrôle les procédures de sécurité pour garantir l'intégrité et la fiabilité des opérations bancaires.

Infrastructure et Filiales

Le Crédit Général dispose d'une infrastructure déployée sur plusieurs régions :

- Le Siège Social est basé à Paris. Les services de direction et les services stratégiques y sont regroupés.
- Les Centres de Développement sont quant à eux installés à Lyon et Toulon/Bonaparte ;). ils sont spécialisés dans la création de solutions logicielles et d'applications bancaires.
- Les Centres de Données : infrastructures sécurisées pour l'hébergement des services bancaires et des API. Ils sont installés sur Sofia Antipolis et Marseille.
- Les agences bancaires : Elles sont plus de 500 et sont réparties sur l'ensemble du territoire, offrant un service de proximité à ses clients.

Par ailleurs CDB dispose de filiales Internationales présentes en Europe et en Amérique du Nord pour les services bancaires aux grandes entreprises.

Services et Solutions Proposés

1. Applications Bancaires pour Entreprises

Les applications bancaires pour entreprises proposées par le Crédit Générale offrent une gestion multi-comptes avancée, permettant aux entreprises de suivre efficacement leurs transactions sur différents comptes. Grâce à des plateforme **web** et mobile personnalisées et sécurisées, l'utilisateur peut accéder à ses informations financières à tout moment et en tout lieu. Ces solutions intègrent des fonctionnalités de suivi des transactions en temps réel, des alertes personnalisées, et des rapports financiers détaillés pour une meilleure prise de décision. Ces applications « *on demand* » sont adaptables et adaptées en fonction des besoins de chacune de ces organisations.

2. Les API de Virements Bancaires

Les API de virements bancaires de Crédit Générale propose des interfaces REST pour automatiser les virements et les paiements entre les entreprises et leurs partenaires. Cette solution permet d'intégrer facilement les processus de paiement dans les systèmes existants, réduisant ainsi les erreurs manuelles et améliorant l'efficacité opérationnelle. L'utilisateur peut gérer les flux financiers de manière transparente et sécurisée, tout en bénéficiant de notifications instantanées pour chaque transaction effectuée.

Cette solution propose donc de traiter les virements individuels, mais aussi de traiter les virements par lot pour la mise en paiement des salariés, des frais, ou des

fournisseurs. Elle permet de faire un suivi précis des transactions afin de planifier et de rejouer les transactions qui n'auraient pu aboutir.

Par ailleurs en plus des solutions de sécurité mise en œuvre quant à l'utilisation des api, les comptes cibles des virements des clients de CGB sont automatiquement vérifiées et doivent faire partie d'une liste blanche de comptes autorisés à être crédités.

3. Solutions de Relance et de Recouvrement

Le Crédit Générale offre un outil complet de suivi des factures impayées et des échéances de paiement, permettant aux entreprises de gérer efficacement leurs créances. Grâce à l'automatisation des relances, l'utilisateur peut envoyer des rappels personnalisés à ses clients, réduisant ainsi les délais de paiement.

4. Système d'Analyse Financière et Statistiques

Le système d'analyse financière de Crédit Générale propose un tableau de bord analytique intuitif pour visualiser l'activité financière de l'entreprise. L'utilisateur peut accéder à des indicateurs clés de performance, des graphiques interactifs, et des rapports personnalisés pour une analyse approfondie.

Partenariats et Clients

Crédit Général compte parmi ses clients des entreprises de divers secteurs, comme ceux de l'industrie pharmaceutique avec Galaxy Swiss Bourdin (GSB) pour la gestion des frais professionnels et des paiements des visiteurs médicaux.
Mais aussi celui des grandes entreprises du secteur industriel en fournissant des solutions de gestion des paiements et des crédits aux fournisseurs.
Et bien entendu celui des Startups et des PME en proposant des offres de financement et de l'accompagnement en transformation Numérique.

Résultats et Réalisations

Grâce aux services proposés à nos client, et la prise en compte des retours d'expérience, CGB à pu améliorer ses résultats et sa rentabilité sur les points suivants :

- L'augmentation du Nombre de Clients : Une augmentation de 20% du nombre de clients actifs grâce à l'amélioration des services numériques.
- La réduction des Impayés de nos clients à hauteur de 15% grâce au système de relance automatisé.
- L'amélioration de la satisfaction client, puisque le taux de satisfaction client est passé de 85 à de 95%, reflétant ainsi la qualité et la fiabilité de nos services.

Partie 2 : Les missions.

Missions à réaliser

Cette partie contient les différentes missions qui vous sont proposées et qui portent sur l'application , elle même basée sur le contexte présenté dans la partie 1.

▸ Pré-requis

- Avoir lu et assimilé le contexte de l'organisation
- Maitriser les concepts de la POO en java.
- Comprendre et connaître et avoir utilisé les concepts suivant afin d'aborder sereinement les projets Spring à savoir :
  - Type générique en java
  - Principe d'annotations et d'injection
  - Bases de données H2
  - Projets et cycle de vie de projet avec Maven
  - Persistance des Objets sous Spring avec JPA
  - Application web coté serveur avec Thymleaf.
  - Tests unitaires en java

1.1 Mission 1 : Etude et configuration de l'API de virement existente.

L'entreprise GSB qui est l'un de nos clients, nous à confié la mission de développer des api Rest permettant la mise en paiement des frais de déplacement de leurs visiteurs.

Vous venez d'être intégré à l'équipe en charge du développement des services de ce projet, afin d'améliorer et de mettre en production ces nouvelles API. Pour rappel :

- Virement par compte, (API existante en cours de développement et en test)
- Virement par lots, (A réaliser)
- Enregistrement des transactions de virements, (Réussies et en echec)
- Rejeux des transactions en echec.
- Sécurisation

L'objectif de cette mission est d'installer et de tester l'API REST de virement simple proposée habituellement aux différents clients. Pour l'instant, cette api est une api de tests qui s'interface à une base de données H2, différente de la base de données postgreSQL de production sur laquelle se trouvent les données réelles de la banque.

L'API REST de virements uniques a été développée pour permettre aux entreprises clientes, comme GSB, d'effectuer des virements bancaires entre le compte courant ou l'un des comptes de l'entreprise et des comptes cibles autorisés (ceux des visiteurs, des employés, des fournisseurs, ...). Cette mission vise à valider le bon fonctionnement de l'API et à proposer des améliorations basées sur les tests à réaliser.

Tâches à Réaliser

Installation de l'API

- Cloner le dépôt de code source de l'API depuis le gestionnaire de versions (Git).
  - Les informations du compte à partir duquel vous pouvez cloner le projet se trouvent dans le document installCGB.txt qui vous sera donné en classe, il contient l'adresse du dépôt et le grained-token pour pouvoir le cloner.
- Configurer l'environnement de développement pour exécuter l'API
  - Java, Spring Boot, base de données H2, Maven. Intégration du projet git à Eclipse (Confer document FT_ GitHub Eclipse fourni dans l'archive ou sur le site cours.fiard.free.fr):
- Etudiez le code pour identifier les *endpoints* à tester
- Préparez vos goals Maven (build) , ex : clean compile, validate

- Compiler puis Lancer l'API localement depuis eclipse et **Identifiez lors du lancement la valeur du jeton** que vous devrez utiliser pour une connexion en mode Authentification Basique.

Utilisation de l'API

Afin de tester manuellement l'API :

- Préparer des données de test pour un virement unique (montant, compte destinataire, référence).
- Configurer la requête ***postman*** pour une authentification en mode **Basic Auth** avec comme utilisateur : user et comme jeton celui identifié au lancement de l'application Spring
- Utiliser l'outil Postman pour envoyer des requêtes à l'API et exécuter le virement.
- Vérifier que le virement est correctement traité et que les comptes sont mis à jour en conséquence.

Tests unitaires et fonctionnels

- Écrire des tests unitaires pour valider les différentes fonctionnalités de l'API (validation des données, exécution des virements), Pour ce faire, vous devez utiliser les dépendances Mock (voir Annexe 2)
- Identifier et documenter les éventuels bugs ou comportements inattendus.

Amélioration de l'API

- Analyser les résultats des tests pour identifier des axes d'amélioration :
  - Réfléchir aux valeurs de retours de l'API, (Sont elles fondées, utiles, améliorables)
  - Réfléchir au mécanisme d'authentification des utilisateurs de l'API
- Proposer (pour l'instant) des modifications du code la sécurité, ou l'ergonomie de l'API. (Le chemin est il satisfaisant?)

Documentation et Rapport

- Comme vous pouvez le constater, votre prédécesseur n'a fourni aucune documentation du code. A l'aide d'annotations appropriées (confer cours Eclipse javadoc déjà vu), mettez à jour la documentation technique de l'API avec les informations recueillies lors des tests et des améliorations.
- Rédiger un rapport succinct :
  - sur les résultats des tests,
  - les améliorations à apporter,
  - et les recommandations pour les développements futurs.

1.2 Mission : Conformité IBAN.

Le développement de ce projet n'est pas conforme aux exigences du système bancaire, en effet, les comptes ne respectent pas la norme IBAN qui sont construits de la manière suivante.

Un IBAN est composé :

- Code du pays : Deux lettres représentant le pays (par exemple, "FR" pour la France).
- Chiffres de contrôle : Deux chiffres utilisés pour la validation de l'IBAN.
- BBAN (Basic Bank Account Number) : Le numéro de compte bancaire de base, dont la structure varie selon le pays. Il peut inclure le code de la banque, le code de l'agence, et le numéro de compte.

Création des utilitaires :

- Créer une classe utilitaire pour pouvoir valider les IBAN avant de les affecter aux comptes. Cette classe (Singleton) contiendra les méthodes :
  - getInstanceValidator() pour renvoyer une instance de CGBIbanValidator (Pattern singleton).
  - boolean isIbanStructureValide(String iban) qui vérifie que la structure de la chaine de l'IBAN respecte les régles syntaxiques.
  - Une autre méthode isIbanValide qui s'appuie sur la bibliothèque java suivante et qui permet de valider les IBAN avec vérification du CRC:

    https://commons.apache.org/proper/commons-validator/apidocs/org/apache/commons/validator/routines/IBANValidator.html
  - Des méthodes qui permettent chacune d'extraire :
    - Le Code du pays
    - Le Chiffre de contrôle
    - Le Basic Bank Account Number
- Créez une classe Abstraite ExceptionInvalideIBAN() et deux classes D'Exception dérivées InvalidIbanFormatException et InvalidUnCheckableIbanException :
  - Une pour lever une exception pour lorsque le format n'est pas respecté (Nombre de caractères, Code pays etc.).
  - l'autre pour lever une Exception lorsque les cas d'iban sont invalide : crc non conforme, invérifiable.
- Ajouter au projet la classe IbanGenerator FOURNIE dans l'annexe 4, qui permet de générer des IBAN, elle vous permettra de générer des IBAN aléatoires, mais valides pour tester votre classe.
- Modifier la classe DataBaseInitialiser afin de créer 20 comptes bancaires valides et remplacez votre base de données H2 avec votre nouvelle base valide.
- Réaliser une classe de tests Maven Junit pour tester vos méthodes, en prévoyant différents cas : format illégal, Format OK, mais Code de vérification incorrecte.

1.3 Mission : Traitement des Lots.

Afin de répondre à la demande de GSB qui désire une API permettant de traiter les virements par lots, on vous donne le cahier des charges pour le développement de cet API REST. Ce document détaille les exigences fonctionnelles et techniques pour vous guider dans ce projet.

Cahier des Charges : API REST pour Virements en Lots

1. Introduction

L'objectif de cette mission est de développer une API REST permettant de réaliser plusieurs virements bancaires en une seule opération. Cette API sera utilisée par les développeurs de nos clients comme par exemple GSB qui l'utilisera pour mettre en paiement leurs frais, voire le paiement des salaires.

2. Contexte

Crédit Générale dispose actuellement d'une API REST en cours de développement permettant d'effectuer des virements individuels entre les comptes courants des entreprises et des comptes autorisés (celle sur laquelle vous avez apporté des améliorations). La base de données H2 stocke (temporairement) les transferts effectués et les comptes autorisés qui ont

été enregistrés manuellement. L'objectif est d'étendre cette fonctionnalité pour permettre les virements en lot.

3. Objectifs

Permettre aux développeurs de GSB d'intégrer dans leurs applications nos services REST pour effectuer plusieurs virements en une seule opération, et donc faciliter l'intégration de nos services avec les systèmes existants de GSB.

Assurer la traçabilité des transactions de façon redondante par des mécanismes de logs, mais aussi par enregistrement des transactions en base de données.

- L'objectif est de prévoir des outils de rejeux en cas d'échec des transactions.
- La sécurisation n'est pas encore demandée, elle le sera dans une autre mission.

4. Exigences Fonctionnelles

4.1. Gestion des Virements en Lot

- L'API doit accepter une liste de virements à effectuer au format JSON, incluant les détails suivants pour chaque virement :
  - Montant du virement. : ***amount***
  - Compte destinataire (qui doit être autorisé et préalablement renseigner. Ce compte fera partie de la liste enregistrée de compte « bénéficiaires » associés au client : pour l'instant GSB). ***destAccount***
  - Référence ou description du virement. ***description***
  - Le lot sera identifié automatiquement par l'api et, daté : La date sera gérée par l'api, comme l'identifiant du lot.
- L'API doit valider les données d'entrée pour s'assurer :
  - que le compte source fait partie des comptes de l'entreprise. Cette vérification sera mise en œuvre plus tard lors de la mission d'authentification du client.
  - que tous les comptes destinataires des virements sont bien autorisés et enregistrées en tant que compte destinataires et que les montants sont disponibles sur le compte courant de GSB. Dans ce cas les transactions qui n'auront pas pu avoir lieu, seront « taguées » et stockées pour être rejouer ultérieurement.
- Par ailleurs cette API en développement devra pouvoir être utilisée par d'autre clients qui auront eux aussi leur liste de comptes bénéficiaires/autorisés et leurs comptes sources à partie desquels ils pourront effectuer les virements..

4.2. Exécution des Virements

- chaque virements du lot doit être traité dans une transaction unique pour garantir l'atomicité.
- En cas d'échec d'un virement, le virement doit être annulé et l'information doit être stockée, afin de pouvoir être, consultable, rejouer plus tard, voire annulée.
  - Remarque la notion d'annulée ne supprimera pas le virement des transactions enregistrées, c'est son état qui sera modifié. Pour rappel les états possibles sont : waiting, failure, success, canceled.
- Comme il s'agit d'un traitement par lot, l'api une fois invoquée, elle lancera le service de gestion des lots de façon asynchrone et reverra, sans attendre la fin du traitement, un objet JSON { numLot : 124 , dateLancement : 2025-03-14 , message : "Traitement Lancé", etat : "waiting" } avec waiting pour EnCours
- toutefois, pour des raisons de logique et de sécurité, un même lot de traitement ne peut pas être lancée deux fois.

4.4. Traçabilité et Conformité

- Toutes les transactions doivent être enregistrées dans la base de données H2 avec des détails complets pour assurer la traçabilité.
- Les logs doivent être conservés pour les audits et la conformité réglementaire.

1.4 Mission : Notification et rapport

Exigences Fonctionnelles (suite)

4.3. Notifications et Rapports

- L'API doit pouvoir générer un rapport détaillé des transactions effectuées au format JSON, incluant le statut de chaque virement (succès ou échec).
- L'API doit proposer une route pour pouvoir générer et consulter ce rapport à la demande
- Une notification doit être envoyée à l'utilisateur autorisé (Dans notre cas celui de GSB) à la fin du traitement du lot par mail donnant uniquement et ce pour des raisons de sécurités : l'identité du lot, sa date, le nombre de transactions en échecs, le nombre de transactions réussies.
- Un liste de virements en échecs doit pouvoir être obtenue en fonction :
  - du numéro de lot
  - d'un intervalle de dates
  - du numéro de compte destinataire.

1.5 Mission : Comptes courants et Comptes bénéficiaires :

Dans toute activité bancaire qui se respecte, les clients des banques disposent de comptes bancaires à partir desquels ils peuvent faire leurs virements.

De la même façon, tout virement ne peut se faire vers un autre compte bancaire qu'à la condition où ce un compte bénéficiaire a préalablement été associé à la liste des comptes bénéficiaires (recipient_account). Bien entendu les virements peuvent se faire vers ses propres comptes.

En résumé chaque client aura à disposition une liste de comptes sources autorisés et une liste de comptes bénéficiaires vers lesquels il pourra faire ses virements.

Cette première étape est en fait une contrainte de sécurité fonctionnelle pour éviter que l'on puisse faire des virements depuis n'importe quel compte vers n'importe quel compte.

Comme à terme, nous devrons aussi authentifier les utilisateurs à l'aide de jetons, que ces utilisateurs seront associés à des rôles : exemple 'ADMIN', 'COMPTABLE', 'UTILISATEUR', nous anticipons cette étape pour architecturer le reste de l'application en développant ces classes.

- On vous demande de mettre en place l'architecture de l'annexe 5, qui permet de répondre à cette demande et d'anticiper la future phase de sécurisation.

1.6 Mission : sécurisation :

Il va de soit qu'une api de la sorte ne peut être mise en production qu'à la condition que les utilisateurs/développeurs ne puisse accéder à ce service qu'avec un mécanisme d'Authentification et d'Autorisation suffisamment fort.

Exigences Fonctionnelles (suite)

- L'API devra donc implémenter un mécanisme d'authentification sécurisé (ex. : OAuth2) pour vérifier l'identité des utilisateurs.
- Seuls les utilisateurs autorisés doivent pouvoir accéder aux fonctionnalités de virement en lot (JWT).

- Pour l'instant, un jeton est généré automatiquement par l'application, et permet d'accéder en mode AuthBasic à l'API. Cette solution n'est pas adaptée et pas sécurisé. On veut mettre en place un mécanisme d'authentification par jeton que chaque utilisateur obtiendra une fois authentifié.
    - Grace à ce jeton, on veut pouvoir accéder aux fonctionnalités selon le rôle attribué aux utilisateurs, deux rôles seront définis le rôle comptable et le rôle utilisateur.
        - Par exemple le rôle COMPTABLE avec l'utilisateur Phil Adelphi aura le droit de lancer le traitements des lots, le rejeu des transactions en échec, et toutes autres transactions,
        - alors que Pat Atchaude avec le rôle UTILISATEUR ne pourra qu'utiliser les api en consultation pour voir les transactions, leurs états etc.
    - Il faudra prévoir que ces comptes puissent être actualisés, supprimés ou modifiés.
    - On doit aussi pouvoir en créer d'autres car cette Api pourra servir à d'autres client (entreprise / customer).
- Bien entendus ces utilisateurs devront être associés d'une façon ou une autres à l'entreprise, conformément au diagramme UML proposé en annexe 6. Cette liste correspond à la listes des comptes de l'entreprise à partir duquel on peut faire les virements.
- Par ailleurs, les virements ne pourront être effectués que sur des comptes autorisés, les comptes bénéficiaires qu'ils faudra associer à l'entreprise (Customer) comme présenté en Annexe 6.

Contraintes

1. Contraintes techniques

Technologies Utilisées

- Langage de programmation : Java.
- Framework : Spring Boot.
- Base de données : H2 (pour le développement et les tests).
- Sécurité : OAuth2 pour l'authentification.

Architecture

- L'API doit suivre les principes RESTful.
- Utilisation de services pour structurer le code et faciliter les tests unitaires.

Sécurité

- Implémentation de l'authentification multi-facteurs pour renforcer la sécurité.
- Chiffrement des données sensibles en transit et au repos.

2. Livrables

- Documentation technique de l'API.
- Code source de l'API.
- Scripts de tests unitaires et d'intégration.
- Guide d'utilisation pour les développeurs de GSB.

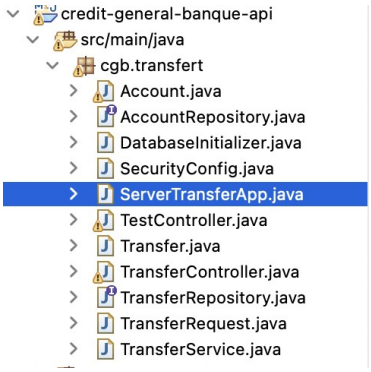
3. Qualité du travail attendu :

- L'API doit permettre d'effectuer des virements en lot de manière sécurisée et fiable.
- Les performances de l'API doivent être optimales pour traiter un grand nombre de virements simultanément.
- La documentation doit être complète et compréhensible pour les développeurs de GSB.

Annexes :

Annexe 1 : Brève Présentation de l'application :

L' application déjà développée est une api rest qui permet de réaliser des virements d'un compte à un autre. Cette application propose une fonctionnalité basique pour faire des virements individuel d'un compte à un autre. Cette application est constituée :



- La classe d'entrée et **ServerTransferApp**
- Elle offre des ***endpoints*** qui sont définis dans les controleurs.
- Elle permet pour l'instant d'effectuer des virements en soumettant au endpoint POST des JSON dont le formalisme est le suivant :
  - Body de la Requête post :

```
{
	"sourceAccountNumber": "123456789",
	"destinationAccountNumber": "678912345",
	"amount": 200.00,
	"currency": "EUR",
	"description": "Remboursement d'un prêt"
}
```

Réponse renvoyée :

```
{
	"id": 22,
	"sourceAccountNumber": "123456789",
	"destinationAccountNumber": "678912345",
	"amount": 40.0,
	"transferDate": "2024-08-15",
	"description": "Remboursement d'un prêt"
}
```

- Elle n'est pas mise en production, elle n'est actuellement que très peu sécurisée par un mécanisme d'authentification basique et jeton unique.
- Elle sert d'application de base pour effectuer les virements simples de compte à compte.
- Par ailleurs, elle ne respecte pas les conventions de structure d'une application Spring.

Annexe 2 : Le package Mock

Le package MockMvc, est un package de tests, disponible sous Spring qui permet de réaliser des tests sur les API rest. Il permet d'intégrer les tests sous Maven afin qu'ils soient exécutés avec le goal (Test)

Ci dessous un exemple de test qui utilise MockMvc sur une api REST :

```
import org.junit.jupiter.api.Test;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;


// @SpringBootTest

@WebMvcTest(Controller.class)
public class MonApplicationTest {

     @Autowired
    private MockMvc mockMvc;

    @Test
    public void testObtenirUtilisateur() throws Exception {
        Long id = 1L;

        mockMvc.perform(get("/user/{id}", id))
               .andExpect(status().isOk())
               .andExpect(content().contentType(MediaType.APPLICATION_JSON))
               .andExpect(content().json("{\"id\":1,\"nom\":\"Lavoisier\"}"));
    }

     @Test
     void contextLoads() {
     }
}
```

- L'api REST testée est une getMapping, elle permet de vérifier que la requête :

```
Get:    http://localhost:port/user/2
```

- retourne bien le JSON :

- ainsi que le code de retour 200 :

Si les tests (goals Maven) se déroulent bien, la console affichera quelque chose comme :

```
2025-02-18T22:08:10.967+01:00 INFO 87713 --- [ main]
c.bacasable.BacasableApplicationTests : Started BacasableApplicationTests in 0.635
seconds (process running for 1.327)

[INFO] [1;32mTests run: [0;1;32m2[m, Failures: 0, Errors: 0, Skipped: 0, Time
elapsed: 1.559 s - in coursBacAsable
```

Pour utiliser MockMvc, il faut intégrer les dépendances :

```
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
```

et ainsi utiliser les packages :

```
org.springframework.test.web.servlet.result.MockMvcResultMatchers
org.springframework.test.web.servlet.MockMvc
```

qui permettent de réaliser des tests en simulant des requêtes get ou post.

Pour plus d'informations Confer : Annexe du cours sur les test Spring du cours Spring-Jpa

Annexe 3 : comment sont validés les ibans ?

les IBAN (International Bank Account Numbers) incluent un mécanisme de validation intégré qui utilise un code de contrôle basé sur un algorithme de type CRC (Cyclic Redundancy Check). Ce code de contrôle est constitué de deux chiffres qui permettent de vérifier l'intégrité de l'IBAN.

Algorithme de validation des IBAN

- Réarrangement de l'IBAN : les quatre premiers caractères (le code du pays et les chiffres de contrôle) sont déplacés à la fin de l'IBAN.
- Conversion des lettres en chiffres : chaque lettre de l'IBAN est remplacée par deux chiffres, où A = 10, B = 11, ..., Z = 35.
- Calcul du modulo 97 : l'IBAN réarrangé est traité comme un grand nombre entier. On calcule ensuite le reste de la division de ce nombre par 97.
- Vérification : Si le reste est égal à 1, l'IBAN est valide, https://commons.apache.org/proper/commons-validator/apidocs/org/apache/commons/validator/routines/IBANValidator.html

Annexe 4 : IbanGenerator

```
package cgb.utils;


import java.math.BigInteger;
import java.util.Random;

public class IbanGenerator {
	private static final String COUNTRY_CODE = "FR";
	private static final int IBAN_LENGTH = 27;
	private static final Random RANDOM = new Random();

	public static String generateValidIban() {
		StringBuilder bban = new StringBuilder();
		for (int i = 0; i+3 < IBAN_LENGTH ; i++) {
			bban.append(RANDOM.nextInt(10));
		}
		String checkDigits = calculateCheckDigits(COUNTRY_CODE,
bban.toString());
		return COUNTRY_CODE + checkDigits + bban.toString();
	}

	private static String calculateCheckDigits(String countryCode, String bban)
{
		String countryNumeric = convertLettersToNumbers(countryCode) + "00";
		String ibanNumeric = bban + countryNumeric;
		BigInteger ibanNumber = new BigInteger(ibanNumeric);
		int checkDigits = 98 -
ibanNumber.mod(BigInteger.valueOf(97)).intValue();
		return String.format("%02d", checkDigits);
	}

	private static String convertLettersToNumbers(String letters) {
		StringBuilder result = new StringBuilder();
		for (char c : letters.toCharArray()) {
			result.append(c - 'A' + 10);
		}
		return result.toString();
	}

	public static void main(String[] args) {
		String validIban = generateValidIban();
		System.out.println("Generated Valid IBAN: " + validIban);
	}
}
```

Annexe 5 : Envoi du Lot

- Structure d'un objet de transfert permettant de soumettre à l'application le lot de virements à traiter.
- Exemple de body de la requête post

```
POST:    http://api.cgb.lan:port/send/lot/
```

- Body :

```
{
     "refLot":"2025-04-02-01",
     "sourceAccount":"FR3339003608488772753575242 6",
     "descriptionLot":"Traitement fiches Frais Février 2025",
     "virements":[
                {
                "destAccount":"FR3050053537324765509952 6542",
                "amount":"250.00",
                "description":"Remboursement frais forfait Février"
                },
                {
                "destAccount":"FR6605270223391088867221 0290",
                "amount":"190.00",
                "description":"Remboursement frais hors forfait Février"
                }
               ]
}
```

Consultation du lot

- Consultation du lot soumis à l'api REST par le biais de la requête :

```
POST:    http://api.cgb.lan:port/get/lot/12
```

- Résultat

```
{
  "id":12,
  "refLot":"2025-04-02-01",
  "sourceAccount":"FR3339003608488772753575242 6",
  "descriptionLot":"Traitement fiches Frais Février 2025",
  "dateLot":"2025-04-02",
  "state":"receive",
  "virements":[
                {
                      "id":243,
                      "destAccount":"FR3050053537324765509952 6542",
                      "amount":"250.00",
                      "description":"Remboursement frais forfait Février",
                      "completionDate":null,
                      "state":"pendling"
                },
                {
                      "id":246,
                      "destAccount":"FR6605270223391088867221 0290",
                      "amount":"190.00",
                      "description":"Remboursement frais hors forfait Février",
                      "completionDate":null,
                      "state":"pendling"
                }
           ]
}
```

Remarque : la référence du lot est basée sur la date + un numéro d'ordre.

Exemple : "refLot":"2025-04-02-01",

Annexe 6 : Diagramme de classe pour la gestion des authentifications.

- Les clients des banques comme GSB sont des customers,
- La structure n'affecte pas la gestion des comptes existants (ACCOUNT)
- Les entités Rôle et UserCGB sur lesquels on pourra s'appuyer pour les authentifications par jetons jwt.
- Pour éviter les confusions avec la table user des serveurs SQL on utilisera la classe UserCGB.
- Pour le reste consulté le cours sur L'authentification par jeton, utilisateur et rôles.
- Les comptes bénéficiaires sont associés aux clients, ce sont les comptes sur lesquels il aura le droit de faire des virements.