

# Test Technique de Recrutement

L'objectif de ce test technique est de mesurer les capacités du candidat à produire une solution de bonne qualité, autour d'un simple cas d'utilisation.

## ⚠ Prérequis:

- avoir un environnement de développement avec un IDE
- avoir installé une [JDK](#) (de préférence  $\geq 8$ )
- avoir installé [Maven \(version 3 ou plus\)](#)
- savoir forker un repository Github et de le cloner en local (pour cela, il est nécessaire de posséder un compte Github)

⚠ **10 minutes** sont conseillées pour lire **attentivement** le présent document et pour prendre en compte les différents fichiers fournis dans le test.

## Déroulement du test (1H30)

Le test est stocké ici : <https://github.com/HiringTechnicalTest/BankAccountFullStackTest>

La première chose à faire est de le forker chez vous

Le test est découpé en 2 étapes qui **doivent** être prises en compte dans **l'ordre** suivant :

1. développer une logique métier à partir de scénarii (*module business*)
2. exposer cette logique métier à travers une API RESTful (*module service*)

## Organisation du projet

Le présent projet est un projet Maven parent contenant 2 modules :

\\_ back-end

    \\_ business

        \\_ src

            \\_ main

            \\_ test

        pom.xml

    \\_ service

        \\_ src

            \\_ main

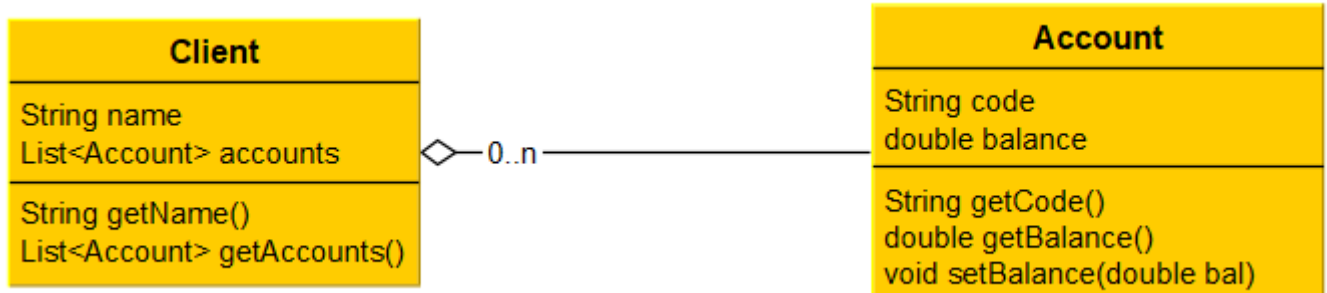
            \\_ test

        pom.xml

## Module business (40 minutes)

⚠ Ce module ne doit contenir **aucune dépendance avec Spring, JPA...** (pure Java). Il est important de conserver ce découplage car c'est lui qui garantit à terme que le modèle métier est maintenable (architecture hexagonale).

Soit un client qui peut posséder plusieurs comptes bancaires.



Votre objectif est d'écrire la classe ClientBankDomain afin de répondre aux trois comportements suivants :

**Scenario:** a client should be able to read his accounts

**Given** I am a client with '100.0' on my account 'TEST\_002'  
**When** I check my account 'TEST\_002'  
**Then** my balance should be '100.0'

**Scenario:** a client should be able to make a deposit on his accounts

**Given** I am a client with '100.0' on my account 'TEST\_002'  
**When** I deposit '10' on my account 'TEST\_002'  
**Then** my balance should be '110.0'

**Scenario:** a client should be able to make a withdraw from his accounts

**Given** I am a client with '100.0' on my account 'TEST\_002'  
**When** I withdraw '10' from my account 'TEST\_002'  
**Then** my balance should be '90.0'

⚠ Une attention toute particulière sera apportée par l'examineur aux tests unitaires.

⚠ L'implémentation des tests Cucumber n'est pas obligatoire. Mais c'est un plus. Les dépendances avec Cucumber ont déjà été ajoutées afin de ne pas perdre de temps à configurer.

## Module service (40 minutes)

L'objectif est d'exposer le contrat du service métier ClientBankDomain que vous venez de créer à travers une API RESTful développée en Spring-Boot.

**Exemple :** GET <http://localhost:8080/api/v1/clients/steve.jobs/accounts/14451>


```
{ "code": "14451", "balance": 100.0 }
```

Le model JPA est fourni ainsi que la configuration avec une base H2 en mémoire.

Console H2 : <http://localhost:8080/h2/console>

- jdbc url : jdbc:h2:mem:testDB
- login : sa
- password : [blank]

De même, Swagger est configuré : <http://localhost:8080/swagger-ui.html>

 Une attention toute particulière sera apportée par l'examineur aux tests unitaires de ce services.