

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 la 16 sinon il faut modifier le POM)
- avoir installé Maven (version 3 ou plus)
- savoir forker un repository Github et 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

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

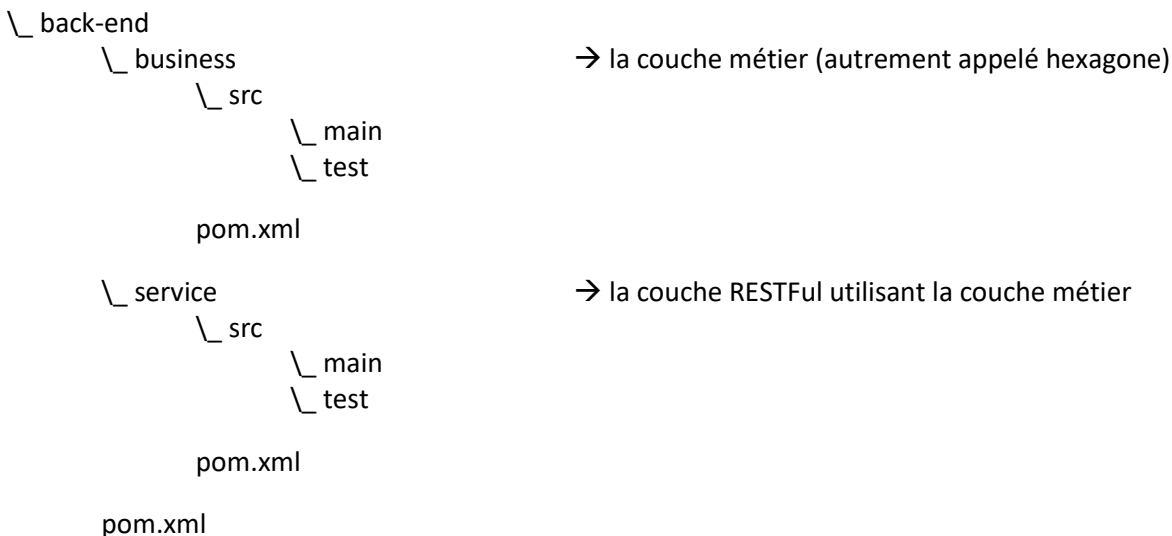
La première chose à faire est de le forker chez vous

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 scénarios
2. Exposer cette logique métier à travers une API RESTful

Organisation du projet Java

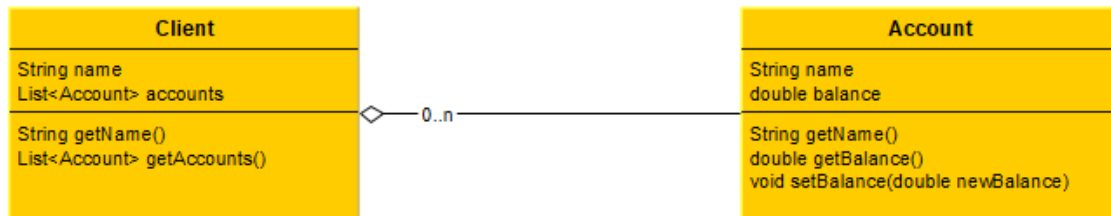
Le présent projet est un projet Maven parent contenant 2 modules :



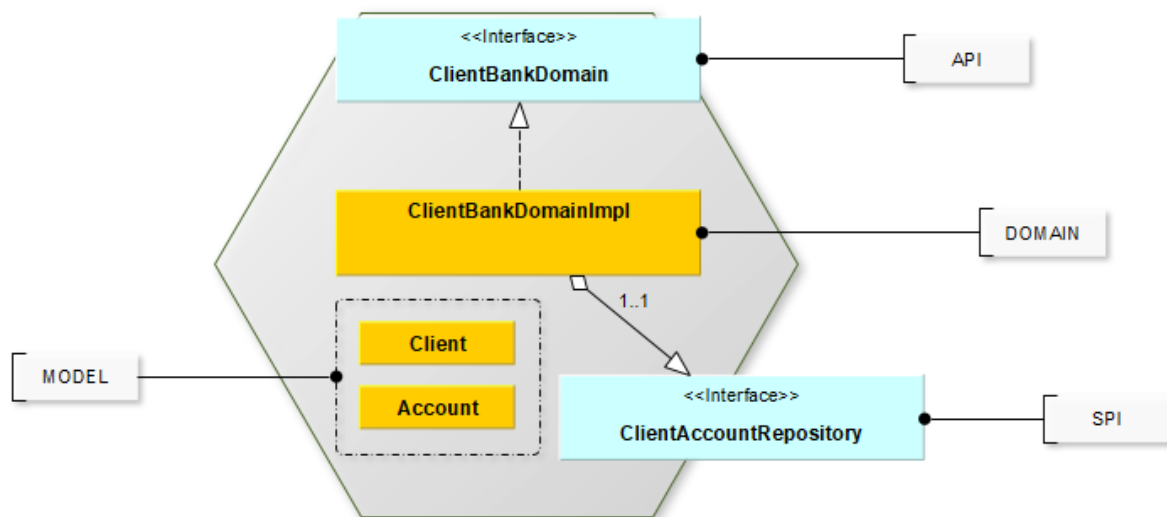
Module business

⚠ 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 aucun ou plusieurs comptes bancaires.



Votre objectif est d'écrire la classe **ClientBankDomainImpl** dans les normes de l'architecture hexagonale :



Et de répondre aux trois comportements suivants :

Scenario: a client should be able to read one of his accounts

Given I am **steve.jobs**

And I own **100.0** on my account **FORTUNEO**

And I own **30.0** on my account **N26**

Then balance of my account **FORTUNEO** should be **100.0**

And balance of my account **N26** should be **30.0**

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

Given I am **elon.musk**

And I own **100.0** on my account **BNP**

When I deposit **10.0** on my account **BNP**

Then balance of my account **BNP** should be **110.0**

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

Given I am **jeff.bezos**

And I own **30.0** on my account **BFORBANK**

When I withdraw **10.0** on my account **BFORBANK**

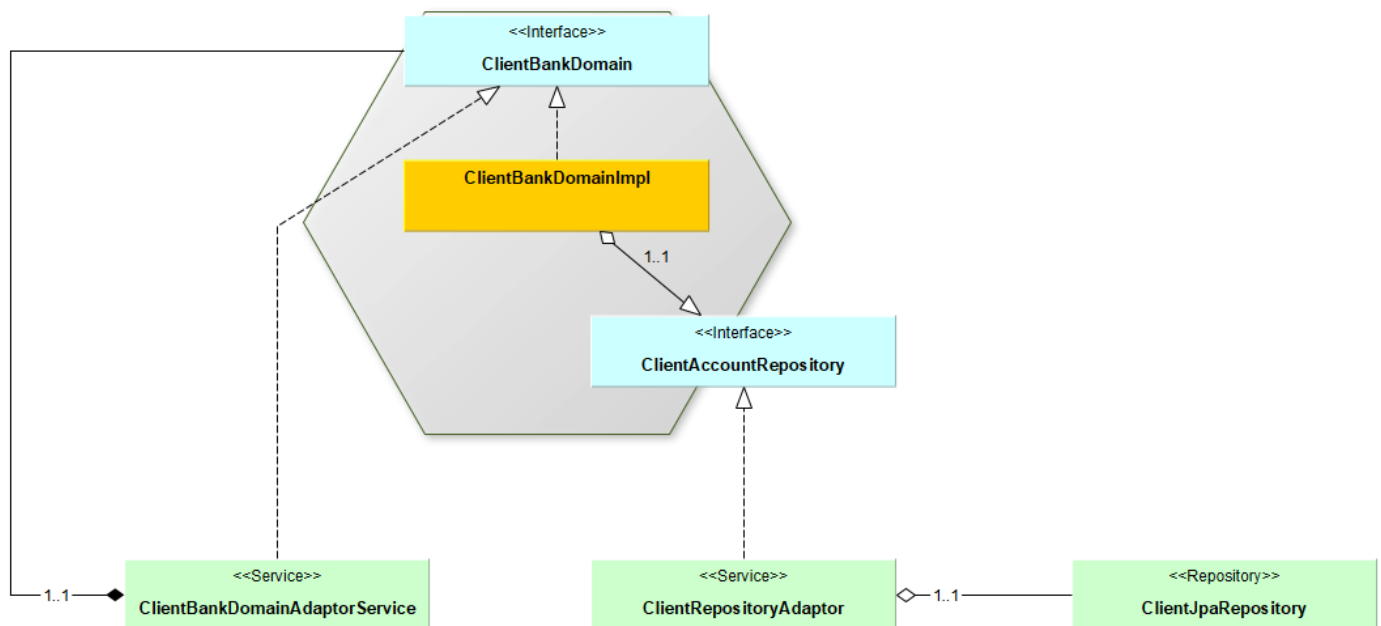
Then balance of my account **BFORBANK** should be **20.0**

⚠ Une attention toute particulière sera apportée 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

L'objectif est désormais d'exposer l'hexagone que vous venez d'implémenter à travers une API RESTful développée en Spring-Boot en respectant le découplage des parties métier et service.

Pour vous aider, voici une partie du diagramme de classe attendu :



Exemple d'utilisation de l'API RESTful

Account et Client peuvent être stockés dans une base de données relationnelle.

GET <http://localhost:8080/api/v1/clients/elon.musk/accounts/BNP>

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

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

```
[
  { "code": "FORTUNEO", "balance": 100.0 },
  { "code": "N26", "balance": 30.0 },
]
```

PATCH <http://localhost:8080/api/v1/clients/elon.musk/accounts/BNP/balances>

BODY : { "deposit" : 10.0 }

⚠ Une attention toute particulière sera apportée aux tests unitaires de ce service.