

# *Systèmes d'exploitation Programmation réseaux*

TP 5 - Mise en place d'une architecture client-serveur Java classique avec sécurité logicielle hybride symétrique et asymétrique.

*Liège, Avril 2015*

HEPL - INGÉNIEUR INDUSTRIEL



**Province  
de Liège**

**Enseignement**

Haute Ecole de la Province de Liège

Thibault Reinders  
Nassim Rahali  
Kaoutare Ahallouch

M 18

Professeur : C. Vilvens

## Table des matières

<b>1</b>	<b>Méthodologique de travail</b>	<b>2</b>
<b>2</b>	<b>Bank Server</b>	<b>2</b>
<b>3</b>	<b>Bank Client</b>	<b>3</b>
<b>4</b>	<b>Agent Bancaire</b>	<b>3</b>
4.1	Vers Bank Server . . . . .	3
4.2	Vers Data Server . . . . .	4
<b>5</b>	<b>Data Server</b>	<b>4</b>
<b>6</b>	<b>Web Server</b>	<b>4</b>
<b>7</b>	<b>AS &amp; TGS</b>	<b>4</b>

# 1 Méthodologique de travail

Nous nous sommes tout d'abord réparti le travail selon les 3 serveurs à faire (Bank, Data et Web).

- Bank Server : Thibault
- Data Server : Kaoutare
- Web Server : Nassim

Ensuite, en fonction de l'avancement de chacun dans son serveur, nous avons fait le reste du travail comme suit

- Kerberos (AS & TGS) : Nassim et Thibault
- Bank Client : Thibault
- Agent Bancaire : Kaoutare et Thibault (chacun la partie utilisant son serveur.)

Pour faire les 3 serveurs, nous n'avons pas eut besoin de nous mettre d'accord puisqu'il n'y a pas d'interactions directes entre les différents serveurs, hormis le Web Server qui devait avoir un accès direct aux bases de données des deux autres serveurs.

Pour l'Agent Bancaire, les interactions entre la partie utilisant le Bank Server et celle utilisant le Data Server n'est pas très importante et nous n'avons pas eut de problème pour développer les 2 en parallèle.

Pour l'AS et le TGS, il nous a fallu travailler ensemble et communication par communication pour ne pas se tromper dans qui chiffre quoi et avec quelle clé.

## 2 Bank Server

Le serveur a 2 tâches à remplir. Il communique avec les clients Bank Client et Agent Bancaire. On utilise donc 2 *ThreadPoolExecutor* pour qu'un trop grand nombre de clients d'un type ne bloque pas l'autre type de client. Logiquement, on donnera plus de threads au pool destiné à communiquer avec les Banques Clientes qu'à celui pour les Agents Bancaires.

Le serveur a accès à une base de données *MongoDB* qui contient les banques clientes et leurs demandes d'opérations de crédit et de débit avec notre banque. Typiquement

```
1 {  
2   "_id" : 1,  
3   "type" : "debit",  
4   "montant" : 1000,  
5   "valide" : "false",  
6   "date" : "2015-04-15"  
7   "banque" :  
8   {  
9     "_id" : 1,  
10    "name" : "Belfius"  
11  }  
12 }
```

La communication avec les Banques Clientes se fait par SSL, on a donc créé des clés (vérifiées par un CA) enregistrées dans un *keystore*, côté client et côté serveur. Pour ne pas alourdir le code, la création d'une socket SSL a été mise dans un petit jar.

Par défaut, les demandes de crédit ou de débit des Banques Clientes sont à *false*. Il faudra l'intervention d'un Agent Bancaire pour les valider.

L'Agent Bancaire peut demander à voir toutes les opérations ou alors les filtrer selon la Banque ou le type d'opération, pour ensuite en sélectionner une ou plusieurs et les valider.

### 3 Bank Client

La Banque Cliente communique donc par SSL avec son serveur. Lors de la connexion, le client doit choisir une banque partenaire de notre banque sous peine d'être refusé. Il peut ensuite faire des demandes de débit et/ou de crédit ou vérifier l'état d'une opération appartenant à sa banque.

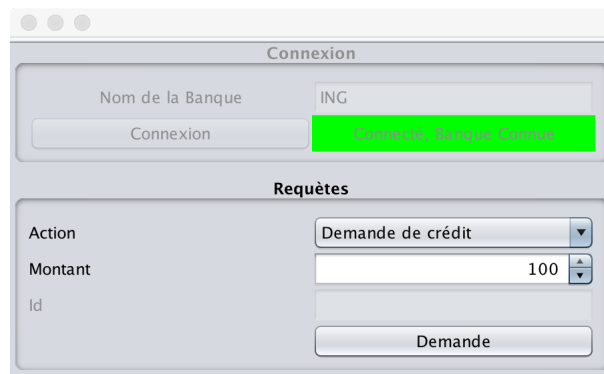


FIGURE 1 – Application Banque Cliente

## 4 Agent Bancaire

### 4.1 Vers Bank Server

L'Agent Bancaire se connecte au Bank Server et s'authentifie au moyen de l'architecture Kerberos. Il peut demander à voir l'ensemble des opérations, ou alors filtrer celles-ci par *Banque* et *Etat*.

Pour valider une ou plusieurs opérations, l'Agent Bancaire va envoyer la liste des *ids* à valider et le serveur va se charger de faire les updates correspondants sur la base de données.

The screenshot shows a web application interface for a 'Bank Server'. At the top, there are tabs for 'Bank Server' and 'Data Server'. Below the tabs is a section titled 'Contrôles' containing a search bar with the text 'Recherche réussie : 12 résultats' in green. Below the search bar is a table with 5 columns: 'Id', 'Banque', 'Type', 'Montant', and 'Validé'. The table contains 12 rows of transaction data. The 'Validé' column has checkboxes for each row, with some already checked. Below the table is a large empty grey area.

Id	Banque	Type	Montant	Validé
1	Belfius	debit	1000	<input type="checkbox"/>
2	ING	debit	2000	<input type="checkbox"/>
3	Belfius	credit	1500	<input checked="" type="checkbox"/>
4	ING	credit	500	<input checked="" type="checkbox"/>
5	BNP Paribas Fortis	credit	2500	<input type="checkbox"/>
6	ING	credit	100	<input checked="" type="checkbox"/>
7	ING	credit	150	<input type="checkbox"/>
8	BNP Paribas Fortis	debit	10000	<input checked="" type="checkbox"/>
9	Belfius	credit	100	<input checked="" type="checkbox"/>
10	ING	credit	90	<input checked="" type="checkbox"/>
11	ING	credit	456	<input type="checkbox"/>

FIGURE 2 – Agent Bancaire (Bank Server)

## 4.2 Vers Data Server

## 5 Data Server

## 6 Web Server

## 7 AS & TGS

C'est la partie qui a été la plus 'délicate' à implémenter. En effet presque toutes les applications de l'architecture sont impactées et il faut travailler en groupe et étape par étape pour être sûr de ne pas chiffrer la mauvaise information ou avec la mauvaise clé pour que le suivant puisse bien décrypter les données.

Point de vue technique, on notera juste l'utilisation de *SealedObject* pour encapsuler des objets chiffrés puisqu'on ne peut pas simplement crypter un objet et en récupérer les bytes.