

CDAI : Banque en ligne

M. Diogo Sobral et M. Bekhouche Abdel

We Are You Bank



Table des matières

Introduction	3
Interface	4
L'architecture	5
Architecture applicative	5
Le modèle de données	6
Partis pris	8
Gestion de l'authentification	9
Fonctionnement	10
Déroulement classique	10
Problèmes rencontrés	14
Conclusion	17
Annexe : Scripts de création des tables	18

Introduction

Notre génération, digital native et ayant grandi avec les technologies, est une cible particulièrement prisée pour les banques en lignes. Parallèlement, nous constituons un ensemble de clients particulièrement exigeants pour ces institutions.

Nous-mêmes clients de certaines de ces banques (ING direct, Hello bank!, LCL filiale en ligne...), nous avons souhaité essayer de réaliser ce projet à l'image d'une banque de laquelle nous aimerions être clients.

Ce projet, réalisé dans le cadre du Master 2 SITN à Dauphine, consiste en la mise en place d'un serveur bancaire permettant à des clients d'effectuer des opérations simples, et aux employés de gérer ces clients.

Nous avons choisi de travailler sous Netbeans, ayant eu l'occasion de nous familiariser avec lors des TP.

Nous n'avons ni l'un ni l'autre un background ou un profil de développeur. Nous n'avons pas suivi, au cours de nos études, de formation poussée sur java ou les applications internet, et ne cherchons pas à nous orienter professionnellement vers du développement, aussi avons décidé de commencer ce projet très tôt, afin de pouvoir appréhender les problèmes, apprendre les techniques de code, d'architecture et de conception qui seraient à appliquer dans notre projet, et le perfectionner. Nous avons donc réfléchi à la modélisation et écrit nos premières lignes de code dès début janvier, intéressés par ce projet au sujet on ne plus d'actualité, et malgré tout rassurés par l'aboutissement de nos TP en cours.

Interface

Nous avons en tête de créer une banque en ligne à l'image de celles que l'on souhaiterait avoir à nos côtés. Pour nous, la première caractéristique d'une banque en ligne doit être la simplicité d'usage. Les clients choisissent une banque en ligne plus qu'une banque traditionnelle pour accélérer les processus bancaires, effectuer leurs opérations depuis n'importe quel endroit connecté, et accéder facilement à leurs comptes. Nous avons donc voulu une interface très simple, des couleurs apaisantes, et une interaction facilitée.

Afin de combler nos lacunes en programmation Web, nous avons décidé d'utiliser un thème HTML5 puis de le modifier en profondeur afin qu'il corresponde à nos attentes.

Voici la page d'accueil de notre banque en ligne :



We Are You

Votre banque en ligne 4G

Adresse e-mail

Mot de passe

Connexion

Comme nous l'avons évoqué, la simplicité était l'élément clé de notre démarche.

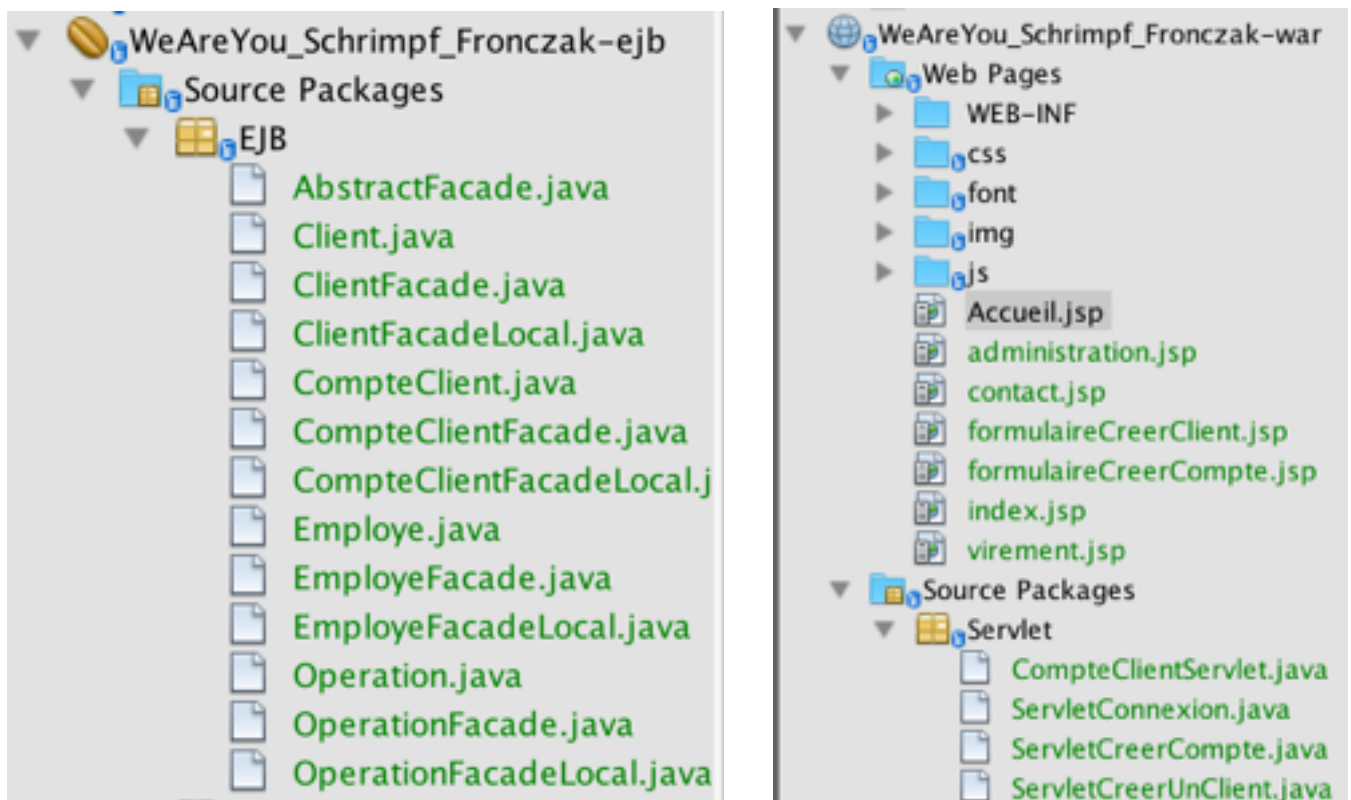
L'architecture

Architecture applicative

Pour architecturer notre application, nous avons décidé d'utiliser les Sessions Bean étant donné qu'elles sont parfaitement adaptées à des clients légers.

De plus, l'utilisation des sessions Bean couplées à celle de Servlet est en parfait accord avec le modèle MVC.

Voici l'arborescence de notre projet afin de détailler cette implémentation :



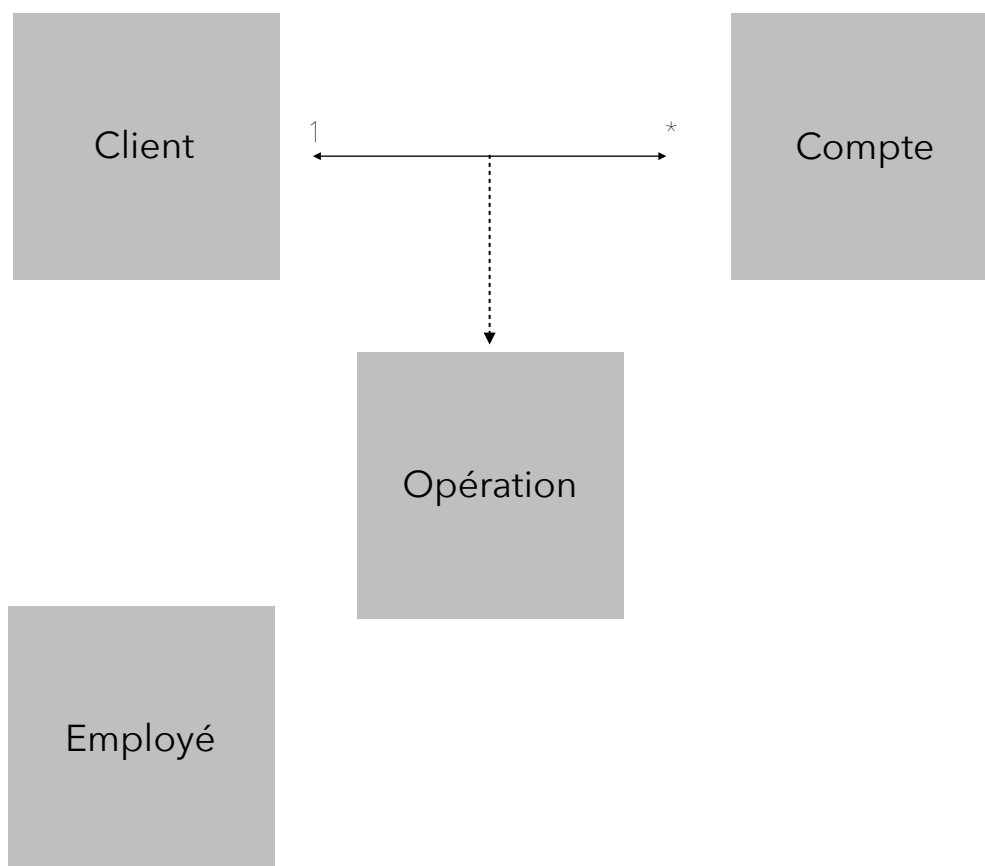
Toutes les classes sont des Bean générés automatiquement à partir de la base de données. Ainsi la cohérence des données est assurée.

Le modèle de données

Avant toute chose, il est nécessaire de préciser que nous avons décidé d'utiliser MySQL en tant que Système de base de données. Ce choix est motivé par le fait que lors de nos stages respectifs, MySQL tenait une place à part dans les solutions utilisées, et nous avons voulu nous rapprocher d'un milieu professionnel auquel nous pouvions nous référencer.

D'autre part, MySQL et Java appartenant tout deux à Oracle, il est logique de penser que les interactions entre ces deux composants ont été pensées et optimisées.

Voici une représentation de notre modèle de données :



- **CLIENT :**
 - ID
 - nom
 - prénom
 - mot de passe
 - téléphone
 - email
 - adresse ligne 1
 - adresse ligne 2
 - code postal
 - ville
 - autorisation de découvert

- **COMPTE :**
 - numéro de compte (id)
 - Type de compte (ex : compte courant, livret A...)
 - Solde
 - CLE ETRANGERE : ID client

- **OPERATION**
 - ID
 - Client débiteur
 - Client créditeur
 - Compte créditeur
 - Compte débiteur
 - Date
 - type (crédit / débit)
 - format (CB/virement...)
 - montant
 - CLE ETRANGERE : Client débiteur, Client créditeur, Compte débiteur, Compte créditeur

- **EMPLOYE :**
 - ID
 - Mot de passe
 - Nom
 - Prénom
 - Adresse e-mail

METHODES

- o Créer client
- o Créer compte
- o Virement
- o Consulter solde (affichage)

- o Suppression de compte (par l'employé)
- o Consulter historique : juste l'affichage de la liste opération
- o Modification des coordonnées d'un client par l'employé

Le schéma que nous avons choisi se veut basique mais néanmoins efficace.

Les identifiants sont tous auto-incrémentés.

Un client possède un ou plusieurs comptes, et des opérations sont effectuées entre différents comptes.

La classe Employé n'est relié à aucune table mais c'est elle qui définit le rôle d'administration.

Partis pris

Plusieurs choix de modélisation se sont imposés. Nous avons ici cité les moins instinctifs – en se basant, encore une fois, sur le modèle auquel nous avons prévu d'aboutir.

- Virements : il n'y a pas de vérification effectuée sur les numéros de compte rentrés. En effet, vérifier si le numéro de compte correspond bien à un compte présent dans notre base serait restreindre : une banque ne peut pas interdire à ses clients de faire des virements vers des comptes d'une autre banque ! We Are You ne déroge pas à la règle, elle n'a pas (pour l'instant) pour ambition de devenir la seule banque présente sur le marché mondial...
- Modification des informations personnelles : ceci n'est pas autorisé au client. Si un client souhaite modifier ses informations personnelles, il doit contacter un employé, qui alors pourra modifier les informations demandées. Nous avons fait ce choix par souci de sécurité.

Gestion de l'authentification

Pour la gestion de l'authentification, nous avons choisi, dans notre optique de simplicité, de ne mettre en place qu'une page de connexion.

L'algorithme de connexion détermine ensuite si l'utilisateur est un employé ou non et le redirige vers la page lui correspondant.

```
public String seConnecter(String emailClient, String motDePasseClient)
{
    String estAdmin = "Non";

    // On exécute une requête pour savoir si un client avec ce mot de passe là existe
    Query Q = em.createQuery("FROM Client client where client.emailClient =:emailClient "
        + "and client.motDePasseClient =:motDePasseClient", Client.class);
    Q.setParameter("emailClient", emailClient);
    Q.setParameter("motDePasseClient", motDePasseClient);

    // On vérifie si l'utilisateur est un employé
    Client TestClient = (Client) Q.getSingleResult();

    if (TestClient == null)
    {
        // On exécute une requête pour savoir si un EMPLOYE avec ce mot de passe là existe
        Query Qemp = em.createQuery("FROM Employee employee where employee.emailClient =:emailClient "
            + "and employee.motDePasseClient =:motDePasseClient", Employee.class);
        Qemp.setParameter("emailClient", emailClient);
        Qemp.setParameter("motDePasseClient", motDePasseClient);
        // On vérifie si l'utilisateur est un EMPLOYE
        Employee TestEmployee = (Employee) Q.getSingleResult();

        if (TestEmployee == null)
        {
            // Si les deux requêtes n'ont pas de resultat, alors l'utilisateur n'existe pas
            estAdmin = "Inconnu";
        }
        else
        {
            estAdmin = "Oui";
        }
    }
    return estAdmin;
}
```

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try
    {
        String emailClient = request.getParameter("emailClient");
        String motDePasseClient = request.getParameter("motDePasseClient");

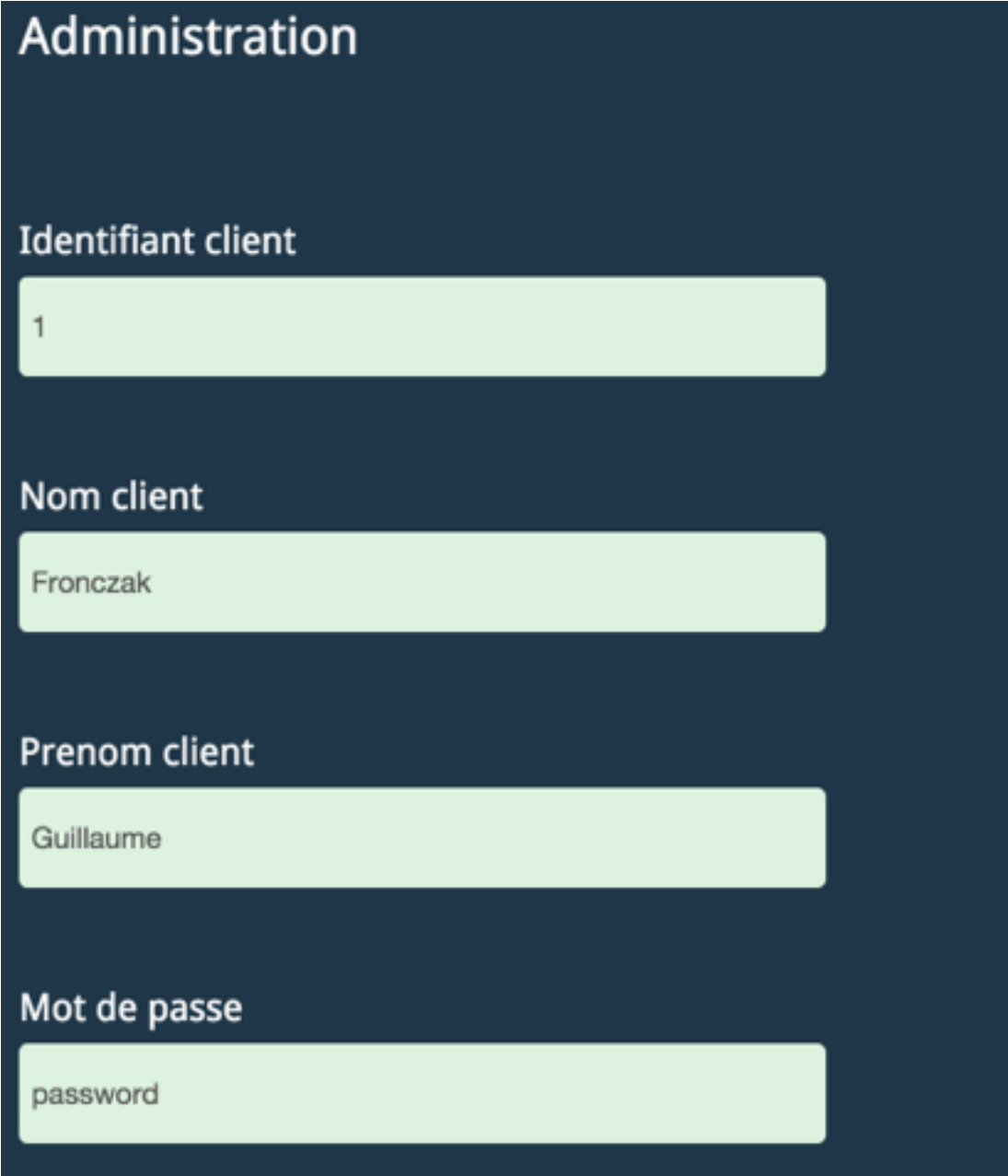
        // On appelle la méthode seConnecter qui définit si l'utilisateur est un employé ou non
        String testConnexion = CCF.seConnecter(emailClient, motDePasseClient);

        if (testConnexion == "Non")
        {
            response.sendRedirect("index.jsp");
        }
        else if (testConnexion == "Oui")
        {
            response.sendRedirect("administration.jsp");
        }
        else
        {
            response.sendRedirect("Accueil.jsp");
        }
    }
    catch (Exception EX)
    {
        System.out.println("Erreur de connexion" + EX.getMessage());
    }
}
```

Fonctionnement

Déroulement classique

Afin de comprendre la manière dont nous avons visualiser notre projet, voici un exemple pratique.
Mr Robert Duchemin souhaite ouvrir un compte courant dans la banque We Are You. Il a contacté les employés par téléphone et fourni les informations nécessaires.
L'employé utilise alors le formulaire « créer un client » et remplit les champs suivant :



The image shows a dark-themed user interface for an 'Administration' section. It contains a form with four input fields, each with a label above it. The labels are 'Identifiant client', 'Nom client', 'Prenom client', and 'Mot de passe'. The input fields are light green and contain the values '1', 'Fronczak', 'Guillaume', and 'password' respectively.

Label	Value
Identifiant client	1
Nom client	Fronczak
Prenom client	Guillaume
Mot de passe	password

Téléphone client

619844254

Adresse e-mail

guillaume.fronczak@gmail.com

Adresse Rue 1

Coucou

Adresse Rue 2

Code postal

75018

Ville Paris

Créer client

Le client apparaît alors dans la base de données.

	idClient	nomClient	prenomClient	motDePasseClient	telClient	emailClient	adresseRue1Client	adresseRue2Client	codePostalClient	villeClient
>	1	Fronczak	Guillaume	password	619844254	guillaume@fronczak.fr	Coucou	1234	75018	Paris

L'employé crée alors un compte courant associé au client Mr Duchemin en remplissant le formulaire suivant :

Num compte

Type compte

Solde compte

Autorisation decouvert

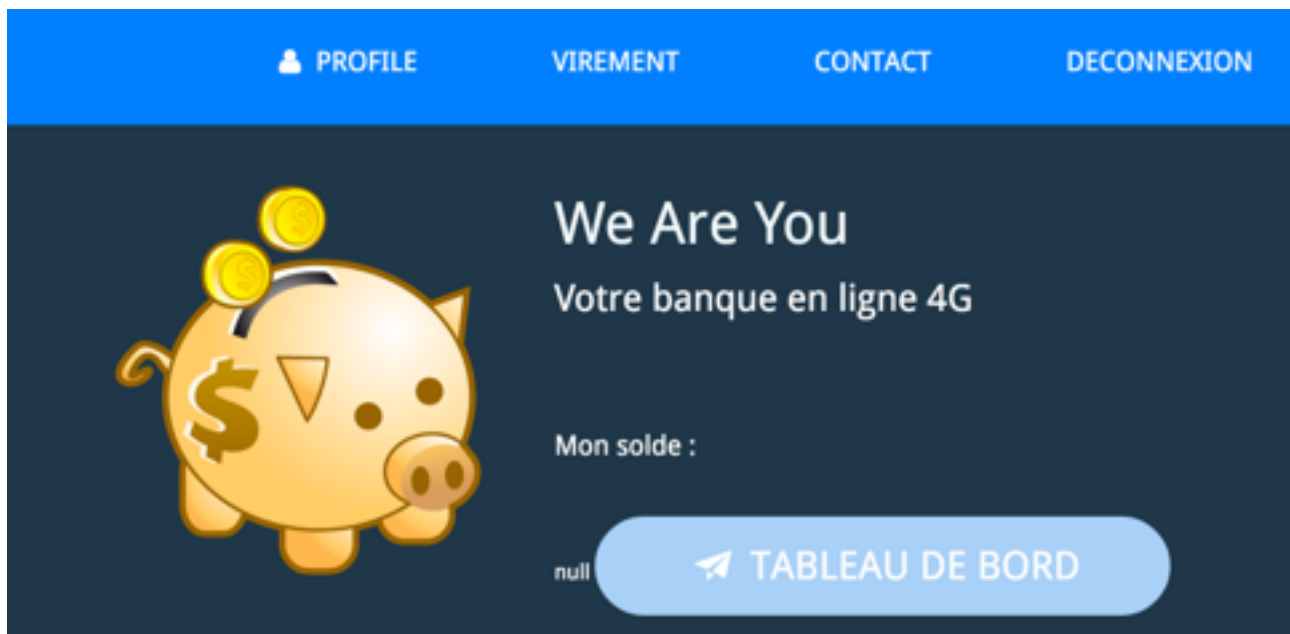
Identifiant client

Par la suite, l'employé peut également créer un autre compte associé à Mr Duchemin (livret, PEL,...) selon la même procédure.

Mr Duchemin peut alors se connecter.

Depuis son espace, il peut :

- Consulter le solde de ses comptes, affiché en accueil.
- Effectuer des virements en renseignant son compte (débiteur), et le numéro de compte créditeur. Si le solde du compte de Mr Duchemin est suffisant pour effectuer le virement (en prenant en compte l'autorisation de découvert), on passe à l'étape suivante, sinon, le virement est refusé. Par la suite : si le compte n'appartient pas à la banque, seule la partie débit de Mr Duchemin est gérée ; sinon, par une recherche BD, le compte correspondant est crédité du même montant, et ce de manière sécurisée.
- Consulter un historique de ses opérations.



Si Mr Duchemin souhaite supprimer son compte, il devra de même contacter par téléphone un employé.

Problèmes rencontrés

Nous avons au cours de ce projet rencontré de nombreux problèmes qui ont freiné notre avancement.

- 1) Difficulté quant à l'emplacement du serveur MySQL. En effet, au moment de la liaison entre Netbeans et notre base de données, il a été difficile de trouver l'adresse de la base de données, étant donnée que celle-ci n'était pas du tout conforme aux « normes »

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd" >
  <persistence-unit name="WeAreYou_Schrimpf_Fronczak-ejbPU" transaction-type="JTA">
    <!--
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    -->
    <jta-data-source>jdbc/WeAreYouPool</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>

    <properties>
      <property name="javax.persistence.schema-generation.database.action" value="create"/>
      <property name="serverName" value="localhost"/>
      <property name="portNumber" value="3306"/>
      <property name="databaseName" value="WeAreYou"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value="root"/>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/WeAreYouDB?zeroDateTimeBehavior=convertToNull"/>
      <property name="driverClass" value="com.mysql.jdbc.Driver"/>
    </properties>
  </persistence-unit>
</persistence>
```

- 2) La configuration de la persistance nous a elle aussi causé des problèmes, notamment quant au protocole mis en place pour que Netbeans reconnaisse le driver MySQL.
- 3) L'appel à la fonction `em.find()` ne semble pas fonctionner. Malgré de nombreuses recherches sur internet, malgré la comparaison avec des passages de codes parfaitement semblables, et qui, sur une autre machine, fonctionne, les appels venant de notre machine ne s'exécutent pas correctement. Après plusieurs tests et pistes de débogage, nous avons réussi à identifier plus précisément le problème : le binding des attributs des requêtes SQL générées automatiquement ne s'effectue pas. De fait, lorsque nous utilisons la méthode `em.find(Attribut, Object.class)`, le binding de « Attribut » n'est pas fait, et la méthode `em.find` ne retourne donc aucun résultat.

Caused by: Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.5.2.v20140319-9ad6abd): org.eclipse.persistence.exceptions.DatabaseException

Internal Exception:
com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Duplicate entry '1' for key 'PRIMARY'

Error Code: 1062

Call: INSERT INTO CompteClient (numCompte, autorisationDecouvert, soldeCompte, typeCompte, idClient) VALUES (?, ?, ?, ?, ?)

bind => [5 parameters bound]

Query: InsertObjectQuery(EJB.CompteClient[numCompte=1]) at org.eclipse.persistence.exceptions.DatabaseException.sqlException(DatabaseException.java: 331)

Nous n'avons jusqu'ici pas réussi à régler ce problème. Le contourner est impossible, car la fonction `em.find()` est nécessaire pour toutes les communications avec la base de données (exceptées la création d'entrées, qui elles sont donc fonctionnelles (création de client)). Par exemple, lorsque nous souhaitons créer un compte client, nous utilisons la méthode `em.find()` afin de retourner l'objet client correspondant à l'identifiant client passé en paramètre (le constructeur de `CompteClient` attend un objet de type `Client`).

```
@Override
public void creerCompte(int numCompte, String typeCompte, double soldeCompte, int autorisationDecouvert, int idClient)
{
    try
    {
        // Je récupère l'objet Client qui correspond à l'ID passé en paramètre
        Client MonClient = em.find(Client.class, idClient);

        CompteClient LeCompte = new CompteClient(numCompte, typeCompte, soldeCompte, autorisationDecouvert, MonClient);
        em.persist(LeCompte);
    }
    catch (Exception EX)
    {
        System.out.println(EX.getMessage());
    }
}
```

- 6) Le problème venant apparemment du Binding, même un passage en dur ne fonctionne pas (passer le client directement n'est pas reconnu non plus)

```
public String seConnecter(String emailClient, String motDePasseClient)
{
    String estAdmin = "Non";

    // On exécute une requête pour savoir si un client avec ce mot de passe là existe
    Query Q = em.createQuery("FROM Client client where client.emailClient = 'guillaume.fronczak@gmail.com' + ' and client.motDePasseClient = 'password' ", Client.class);
    // Q.setParameter("emailClient", emailClient);
    // Q.setParameter("motDePasseClient", motDePasseClient);

    // On vérifie si l'utilisateur est un employé
    Client TestClient = (Client) Q.getSingleResult();
}
```

Ici, l'email client et son mot de passe sont passés en dur. Cependant, une exception est levée :

Caused by: Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.5.2.v20140319-9ad6abd): org.eclipse.persistence.exceptions.DatabaseException
Internal Exception: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'OPTION SQL_SELECT_LIMIT=DEFAULT' at line 1

Error Code: 1064

Call: SELECT idClient, adresseRue1Client, adresseRue2Client, codePostalClient, emailClient, motDePasseClient, nomClient, prenomClient, telClient, villeClient FROM Client WHERE ((emailClient = ?) AND (motDePasseClient = ?))

bind => [2 parameters bound]

Query: ReadAllQuery(referenceClass=Client sql="SELECT idClient, adresseRue1Client, adresseRue2Client, codePostalClient, emailClient, motDePasseClient, nomClient, prenomClient, telClient, villeClient FROM Client WHERE ((emailClient = ?) AND (motDePasseClient = ?))")

- 8) Problèmes de matériel : nous avons été retardés, au cours de nos travaux, par le fait que nous travaillions sur un mac, parfois en proie à des problèmes de configuration. Nous ne disposions malheureusement pas de beaucoup de choix de matériel de base, et avons choisi d'office d'écarter les ordinateurs proposés par le CRIO de la fac (un précédent projet nous a fait découvrir leur manque de stabilité et leurs horaires d'accès réduit) Cela nous a donc ouvert les yeux sur le fait qu'il est très important, dans un projet, de sélectionner avec efficacité et bon instinct le matériel sur lequel travailler, pour ne pas se retrouver freinés par la suite. Dans notre vie professionnelle, c'est une question à laquelle nous serons régulièrement confrontés ; c'était donc plus qu'intéressant de pouvoir s'en rendre compte au cours de ce projet.

Certains problèmes énoncés ci-dessus ont bloqué notre avancement. En effet, le fait de ne pas réussir à utiliser `d'em.find()` et de correspondre ainsi avec la base de données nous a empêché de construire jusqu'au bout le scénario précédemment énoncé.

Nous avons alors fait le choix de continuer l'implémentation des méthodes, même sans pouvoir les exécuter sans rencontrer ce bug, et de continuer à chercher une solution à ce problème d'ici la soutenance - une fois ce problème résolu, le reste sera alors déjà en partie implémenté.

Conclusion

Ce projet, s'inscrivant dans le cadre du module de conception et développement d'applications internet, nous a permis de nous familiariser avec l'architecture, la programmation JEE, et l'utilisation du mapping relationnel objet. Nous avons aussi pu mettre en pratique la séparation des couches dictée par le modèle MVC, par l'emploi de servlet et de sessions Bean.

La conception de l'application nous a également fait découvrir le lancement à petite échelle d'études concernant le but de l'application, les interactions souhaitées, et les technologies utilisées.

Ce projet nous a également permis de nous renseigner avec précision, lors de recherches préalables, sur le fonctionnement des banques en ligne, et les questions de sécurité et conformité que ce type d'établissement soulève. Dans le contexte actuel, et au vu de nos études universitaires, c'était très intéressant de s'y intéresser « de l'autre côté du rideau », c'est-à-dire en tant que concepteur plus que client.

Cependant, comme nous l'avons évoqué précédemment, nous avons rencontré de nombreux problèmes au cours de la réalisation de ce projet, qui sont venus s'ajouter à notre connaissance relativement basique de la programmation web.

D'une part, ces problèmes nous ont permis d'améliorer notre capacité d'adaptation et de recherche de solutions, que nous allons devoir employer à de nombreuses reprises au cours de notre vie professionnelle. En effet, les imprévus ne sont pas rares et il est indispensable de pouvoir s'adapter et trouver de nouvelles solutions afin de ne pas retarder un projet important.

En conclusion, et malgré la frustration créée par le manque de solution finale satisfaisante, ainsi que le travail de recherches et d'essais répétés que nous avons effectué, nous pouvons affirmer que ce projet participe à notre insertion professionnelle en nous incitant à trouver des solutions alternatives lorsqu'un problème est rencontré, en ce point il a donc été très enrichissant pour nous. De plus, nous nous sommes beaucoup impliqués dans ce projet en passant de nombreuses heures à rechercher les réponses aux problèmes survenus, à repousser nos limites, et ce malgré la frustration engendrée. De fait, ce projet nous a également appris à ne pas baisser les bras face à une avalanche de problèmes, et à persévérer dans notre travail.

Annexe : Scripts de création des tables

```
-- MySQL dump 10.13 Distrib 5.6.19, for osx10.7 (i386)
--
-- Host: localhost Database: WeAreYouDB
--
-- Server version 5.6.23

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `Client`
--

DROP TABLE IF EXISTS `Client`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Client` (
  `idClient` int(11) NOT NULL AUTO_INCREMENT,
  `nomClient` varchar(45) NOT NULL,
  `prenomClient` varchar(45) NOT NULL,
  `motDePasseClient` varchar(45) NOT NULL,
  `telClient` int(11) NOT NULL,
  `emailClient` varchar(45) NOT NULL,
  `adresseRue1Client` varchar(45) NOT NULL,
  `adresseRue2Client` varchar(45) DEFAULT NULL,
  `codePostalClient` varchar(5) NOT NULL,
  `villeClient` varchar(45) NOT NULL,
  PRIMARY KEY (`idClient`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Client`
--
```

```

LOCK TABLES `Client` WRITE;
/*!40000 ALTER TABLE `Client` DISABLE KEYS */;
INSERT INTO `Client` VALUES (1,'Fronczak','Guillaume','password',
619844254,'guillaume@fronczak.fr','Coucou',NULL,'75018','Paris'),
(2,'Fronczak','Guillaume','motdepasse',619844254,'email@gmail.com','Rue
Laghouat',NULL,'75018','Paris'),(3,'Fronczak','Guillaume','motdepasse',
619844254,'email@gmail.com','Rue Laghouat',NULL,'75018','Paris'),
(4,'Fronczak','Guillaume','motdepasse',619844254,'email@gmail.com','Rue
Laghouat',NULL,'75018','Paris'),(5,'Guillaume','FRONCZAK','kujaZZZ',
619844254,'guillaume.fronczak@gmail.com','2 Rue de Rennes','test','75018','Paris'),
(6,'Guillaume','FRONCZAK','kuja2212',78,'guillaume.fronczak@gmail.com','2 Rue de
Rennes','test','75018','Paris'),(7,'Schrimpf','Ombeline','coucou',89,'omb@coucou.fr','2 Rue de
Rennes','test','75006','Paris');
/*!40000 ALTER TABLE `Client` ENABLE KEYS */;
UNLOCK TABLES;

```

```

--
-- Table structure for table `CompteClient`
--

```

```

DROP TABLE IF EXISTS `CompteClient`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `CompteClient` (
  `numCompte` int(11) NOT NULL,
  `typeCompte` varchar(45) NOT NULL,
  `soldeCompte` double NOT NULL,
  `autorisationDecouvert` int(11) NOT NULL,
  `idClient` int(11) NOT NULL,
  PRIMARY KEY (`numCompte`),
  KEY `idClient_idx` (`idClient`),
  CONSTRAINT `idClient` FOREIGN KEY (`idClient`) REFERENCES `Client` (`idClient`) ON DELETE
NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

```

```

--
-- Dumping data for table `CompteClient`
--

```

```

LOCK TABLES `CompteClient` WRITE;
/*!40000 ALTER TABLE `CompteClient` DISABLE KEYS */;
INSERT INTO `CompteClient` VALUES (1,'courant',1000,100,1);
/*!40000 ALTER TABLE `CompteClient` ENABLE KEYS */;
UNLOCK TABLES;

```

```

--
-- Table structure for table `Employe`

```

--

```
DROP TABLE IF EXISTS `Employe`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Employe` (
  `idEmploye` int(11) NOT NULL,
  `nomEmploye` varchar(45) NOT NULL,
  `prenomEmploye` varchar(45) NOT NULL,
  `mdpEmploye` varchar(45) NOT NULL,
  `emailEmploye` varchar(45) NOT NULL,
  PRIMARY KEY (`idEmploye`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;
```

--

-- Dumping data for table `Employe`

--

```
LOCK TABLES `Employe` WRITE;
/*!40000 ALTER TABLE `Employe` DISABLE KEYS */;
/*!40000 ALTER TABLE `Employe` ENABLE KEYS */;
UNLOCK TABLES;
```

--

-- Table structure for table `Operation`

--

```
DROP TABLE IF EXISTS `Operation`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Operation` (
  `idOperation` int(11) NOT NULL,
  `idClient` int(11) NOT NULL,
  `dateOperation` datetime NOT NULL,
  `formatOperation` varchar(45) NOT NULL,
  `beneficiaireOperation` varchar(45) NOT NULL,
  `debiteurOperation` varchar(45) NOT NULL,
  `numCompteBeneficiaire` int(11) NOT NULL,
  `numCompteDebiteur` int(11) NOT NULL,
  `montantOperation` double NOT NULL,
  `libelleOperation` varchar(45) NOT NULL,
  `typeOperation` varchar(45) NOT NULL,
  PRIMARY KEY (`idOperation`),
  KEY `numCompteBeneficiaire_idx` (`numCompteBeneficiaire`),
  KEY `numCompteDebiteur_idx` (`numCompteDebiteur`),
  CONSTRAINT `numCompteBeneficiaire` FOREIGN KEY (`numCompteBeneficiaire`) REFERENCES
`CompteClient` (`numCompte`) ON DELETE NO ACTION ON UPDATE NO ACTION,
```

```
        CONSTRAINT `numCompteDebiteur` FOREIGN KEY (`numCompteDebiteur`) REFERENCES
`CompteClient` (`numCompte`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Operation`
--

LOCK TABLES `Operation` WRITE;
/*!40000 ALTER TABLE `Operation` DISABLE KEYS */;
/*!40000 ALTER TABLE `Operation` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2015-02-18 23:27:08
```