

# ProxiBanque

---

Création d'un outil de gestion bancaire

# Besoin client

---

## ■ Contexte

- Client : ProxiBanque, agence bancaire
- Besoin : application de gestion bancaire
  - Conseiller : gérer les clients et effectuer des virements
  - Directeur : auditer l'agence et consulter la liste des conseillers

## ■ Cahier des charges

| Besoin                         | Solution proposée            |
|--------------------------------|------------------------------|
| S'authentifier                 | Identifiant et mot de passe  |
| <b>Conseiller</b>              |                              |
| Lister les clients             | Liste de clients             |
| Editer un client               | Bouton d'édition/suppression |
| Créer un client                | Formulaire de création       |
| Lister les clients à découvert | Audit                        |
| Virement compte à compte       | Outil de virement            |
| <b>Directeur</b>               |                              |
| Suivre les transactions        | Graphique                    |
| Lister les conseillers         | Liste de conseillers         |
| Lister les clients à découvert | Audit                        |

# Organisation du projet

---

## ■ Présentation de l'équipe

- Samuel Bouchet : Master Ingénierie Mathématique
- Aurélie Potier : Master Sciences du Climat
- Maëva Soulabaille : Ingénieure Informatique Industrielle
- Ghania Bouzemame : Master 2 Chimie

# PROXIBANQUE : PRESENTATION DE L'APPLICATION

Besoin client

Organisation (2/2)

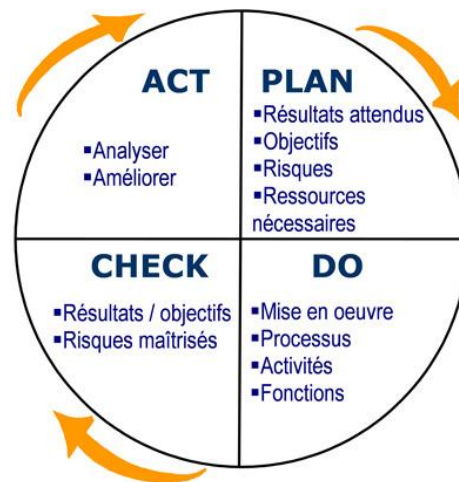
Conception

Fonctionnalités

Bilan

Conclusion

## ■ Organistation

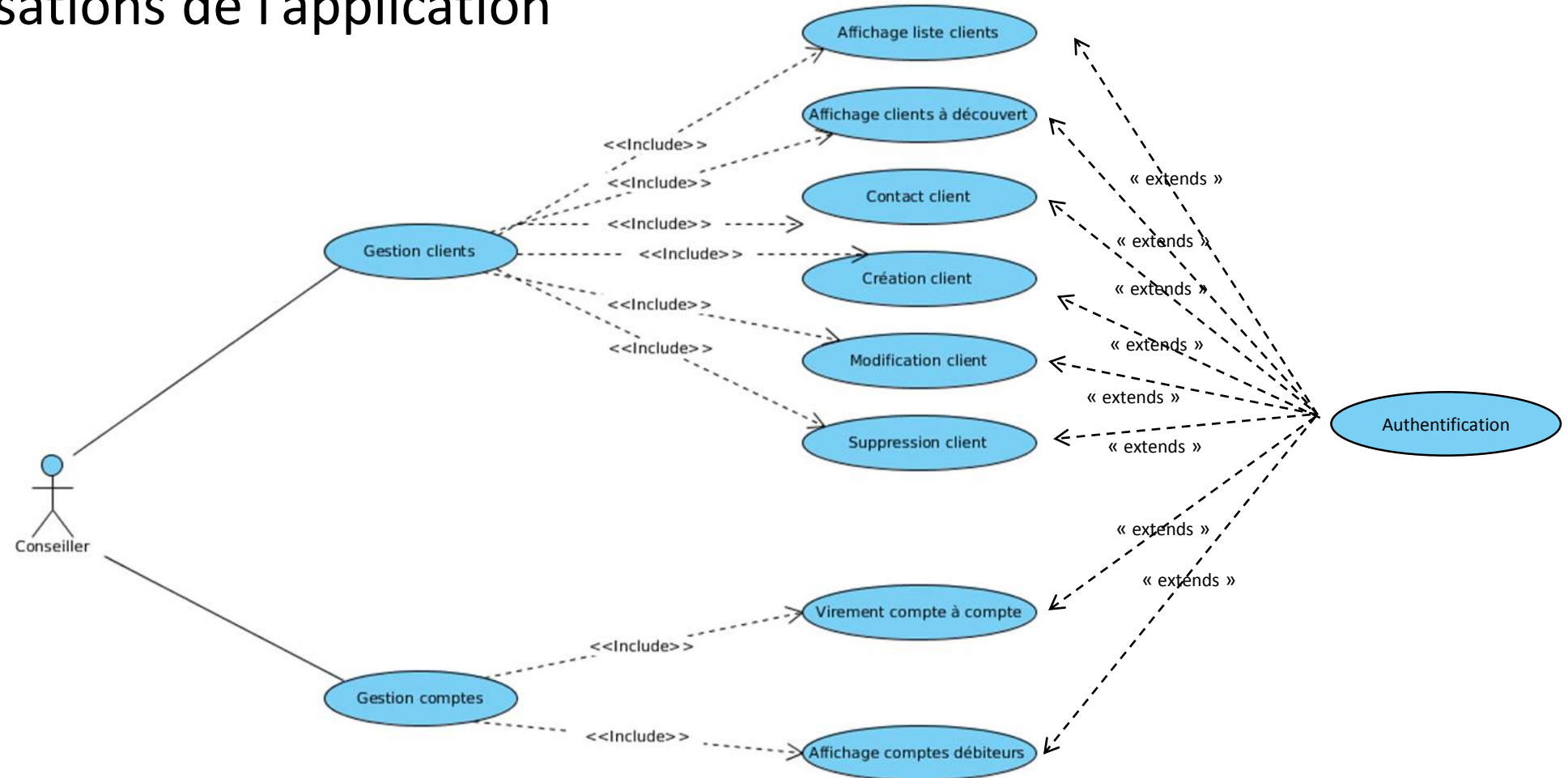


# Conception

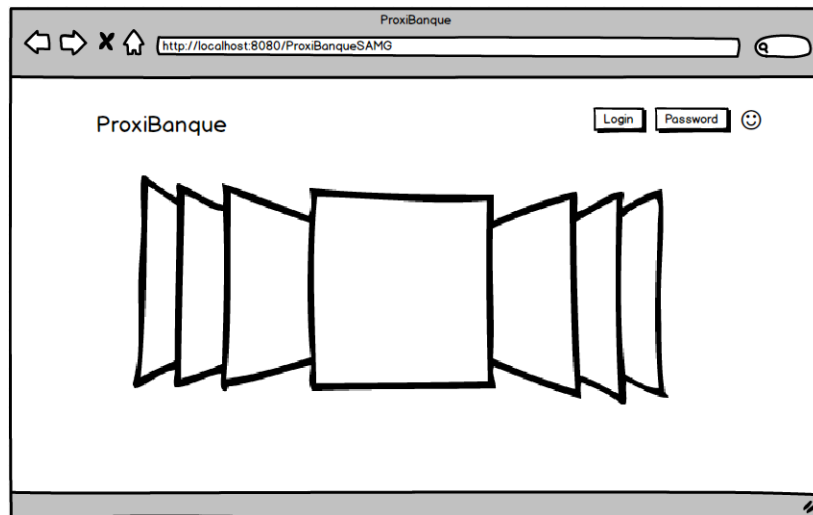
---



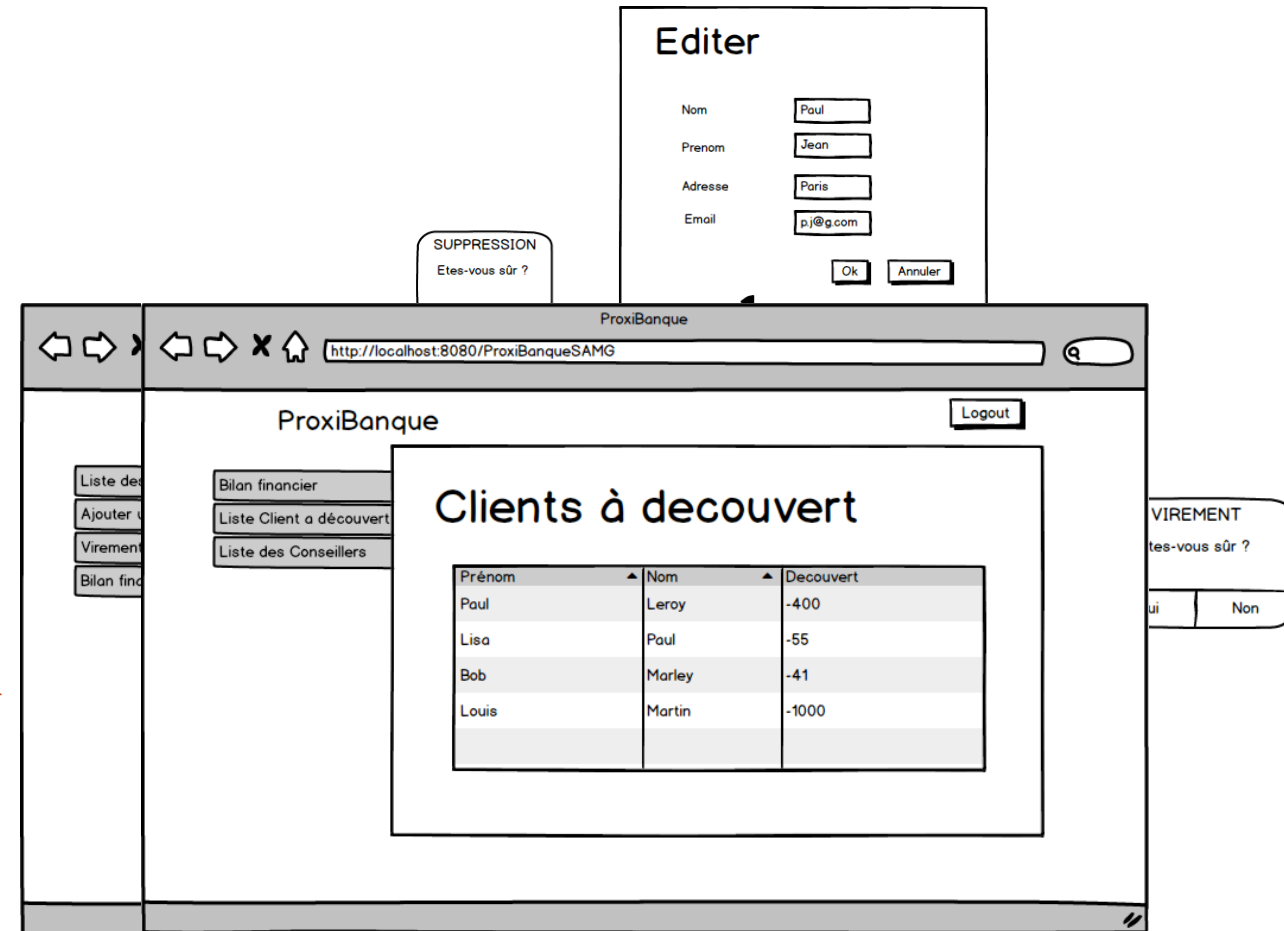
## ■ Cas d'utilisations de l'application



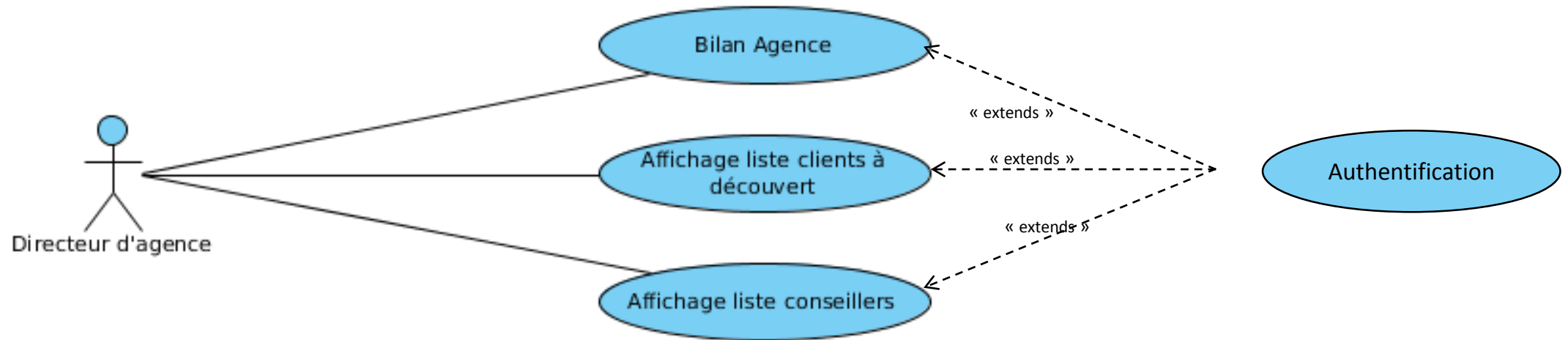
## ■ Maquette – Rôle conseiller



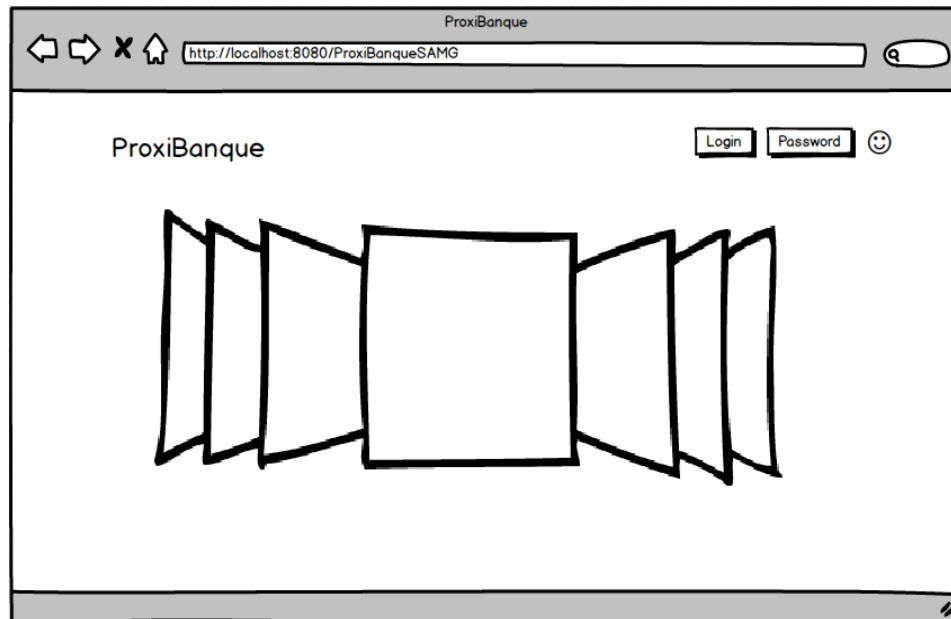
Page d'accueil



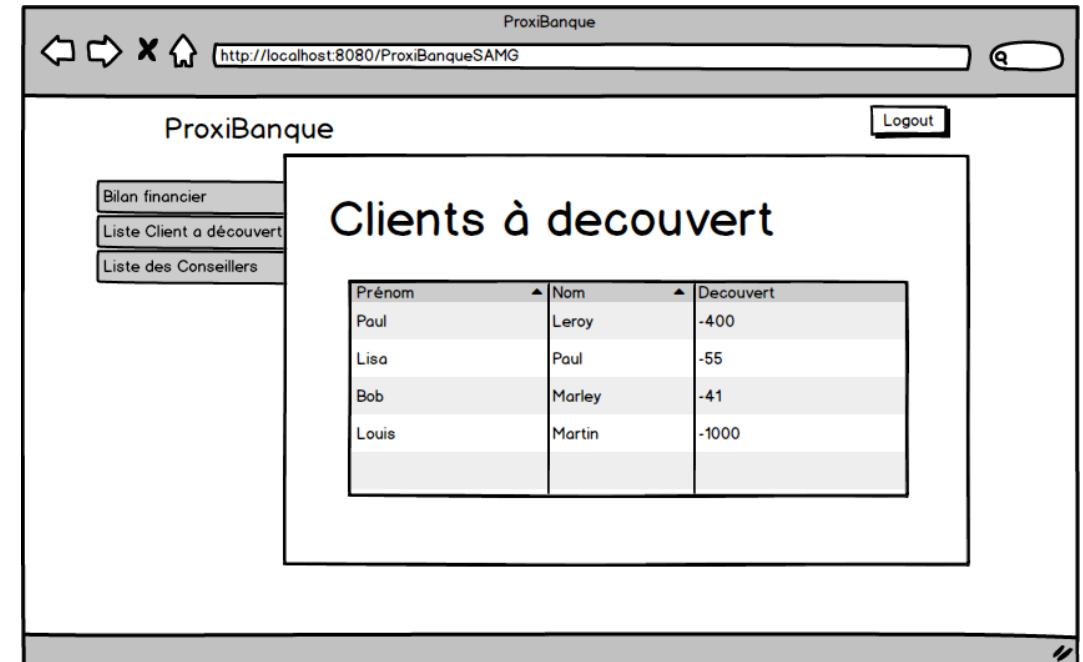
## ■ Cas d'utilisations de l'application



## ■ Maquette – Rôle directeur



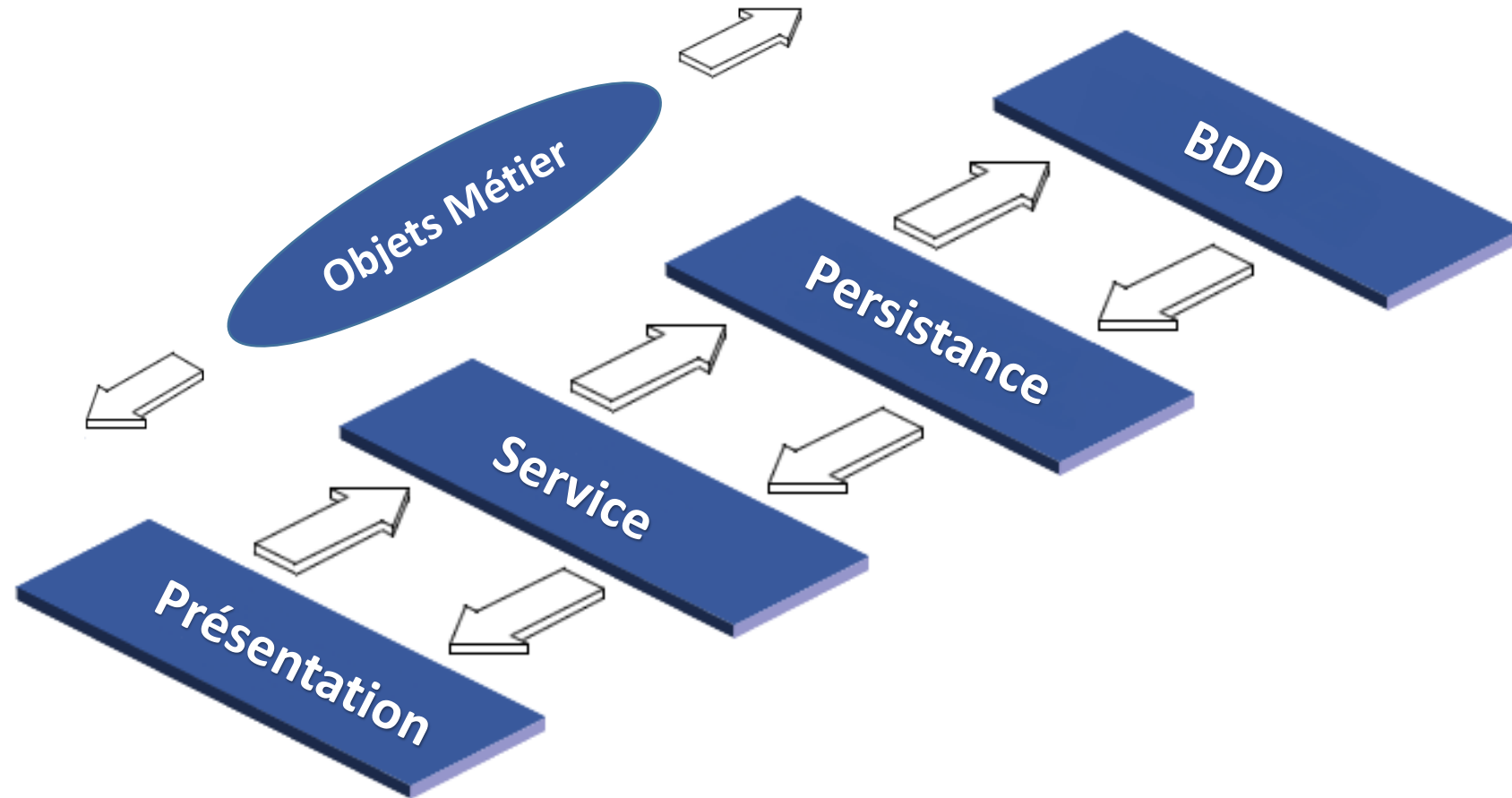
Page d'accueil



## ■ Environnement et outils de développement



## ■ Modèle en couche



# Fonctionnalités

---

## ■ Fonction 1 : Authentication

- Besoin client : l'application permet à deux profils de se connecter
- Solution retenue :
  - Création de deux champs : "Login" et "Mot de passe"



A mockup of a login form. It consists of two adjacent input fields on a light gray background. The first field is labeled 'Login' in a small, light blue font. The second field is empty. To the right of the second field is a square button with a blue border and a black silhouette of a person's head and shoulders.

- Technologie mise en œuvre
  - Champ/Controller



## ■ Fonction 1 : Authentification



```
<b:navBar brand="ProxiBanque" brandHref="#">

  <h:form styleClass="navbar-form navbar-right">
    <div class="form-group">
      <p:inputText placeholder="Login" value="#{userController.login}" />
    </div>
    <div class="form-group">
      <h:inputSecret value="#{userController.password}" />
    </div>

    <b:commandButton action="#{userController.checkPassword()}"
      icon="user">
      <f:param name="testId" value="33" />
    </b:commandButton>
  </h:form>
</b:navBar>
```

.xhtml

```
package org.proxib.presentation;

import java.util.HashMap;

@Component(value = "userController")
@SessionScoped
public class UserController {

    private static Logger LOGGER = LoggerFactory.getLogger(UserController.class);

    private String login, password;
    private Adviser adviser = new Adviser();

    private static Map<String, Adviser> users;
    static {
        users = new HashMap<String, Adviser>();
        users.put("conseiller", new Adviser("conseiller", "toto"));
        users.put("directeur", new Adviser("directeur", "tata"));
    }

    public String checkPassword() {
        Adviser u = users.get(login);
        if (u != null && password.equals(u.getPassword())) {
            if ("conseiller".equals(u.getLogin())) {
                return "accueil_conseiller";
            }
            else if ("directeur".equals(u.getLogin())) {
                return "bilan";
            }
        }
        notificationError("Identifiants");
        return "";
    }

    public void notificationSuccess(String operation) {

        LOGGER.info("Operation " + operation + " success");
        FacesMessage msg = null;
        msg = new FacesMessage(FacesMessage.SEVERITY_INFO, "Notification", "Success");
        FacesContext.getCurrentInstance().addMessage(null, msg);
    }
}
```

## Ajout client

**Veuillez saisir les caractéristiques du client**

Nom

Prénom

Adresse

Email

Compte courant

Compte épargne

Créer

Effacer

## ■ Fonction 2 : créer un client

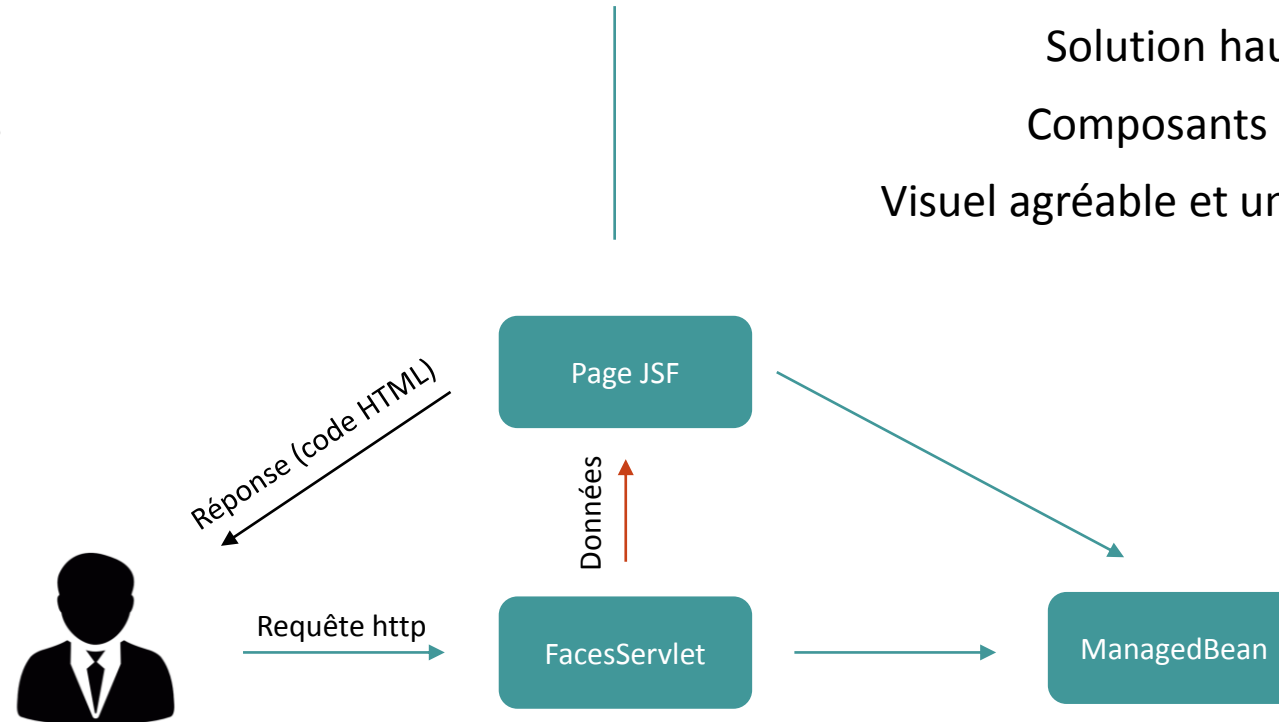
### • Technologies retenues

JSF

PrimeFaces

Bootstrap

Solution haut niveau  
Composants "ajaxisé"  
Visuel agréable et uniformisé



## ■ Fonction 2 : créer un client

- Contrôle des champs

Compte courant

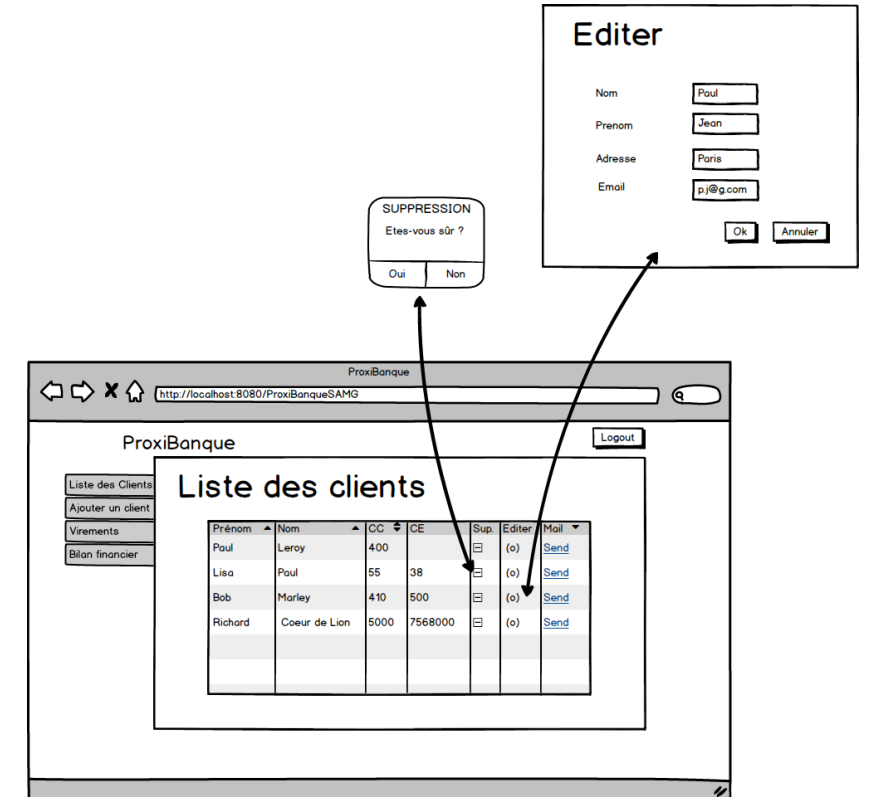
✖ Compte Courant : erreur de validation. La valeur est inférieure à la valeur minimale autorisée, "1".

```
<h:outputText value="Compte courant"
    style="padding-left: 30px; padding-right: 30px;" />
<p:inputText type="number" style="text-align: right"
    id="currentAccount" value="#{clientController.balanceCurrent}"
    label="Compte Courant" required="true">
    <f:validateLongRange minimum="1" />
</p:inputText>
<p:message for="currentAccount"></p:message>
```



## ■ Fonction 3 : Gérer les clients existants

- Besoins du conseiller
  - Afficher la liste des clients et les comptes associés
  - Supprimer et mettre à jour les clients
- Solution proposée
  - Création d'une page « Accueil conseiller »
  - Limiter la navigation entre différentes pages



## ■ Fonction 3 : Gérer les clients existants (couche présentation)

Menu

- Actions clients
- Liste des clients
- + Ajouter un client
- Transactions
- Virement
- Bilan
- Découverts

### Liste des clients

Clients à découverts

⚠ Vous avez 2 clients à découvert

| Prénom ↕      | Nom ↕     | CC ↕    | CE ↕    | Sup. | Editer | Mail |
|---------------|-----------|---------|---------|------|--------|------|
| Leroy         | Paul      | 1000.0  | 10000.0 | ×    | ✎      | Send |
| marley        | bobby     | -50.0   | 1111.0  | ×    | ✎      | Send |
| Lis           | Paula     | 50800.0 | 29000.0 | ×    | ✎      | Send |
| marley        | bob       | 400.0   |         | ×    | ✎      | Send |
| Potter        | Harry     | 77007.0 |         | ×    | ✎      | Send |
| Dujardin      | Jean      | 217.0   | 117.0   | ×    | ✎      | Send |
| Le grand      | Alexandre | 757.0   | 2000.0  | ×    | ✎      | Send |
| Coeur de Lion | Richard   | 562.0   |         | ×    | ✎      | Send |
| Lamy          | Alexandra | -1000.0 |         | ×    | ✎      | Send |
| Ricci         | Riccu     | -200.0  |         | ×    | ✎      | Send |

## ■ Fonction 3 : Gérer les clients existants

- Technologies mises en œuvre :
  - JPA - Java Persistence API / Hibernate
  - ORM - Object Relationnal Mapping
- Avantages :
  - JPA – spécifications standard JEE
  - Hibernate : fournisseur de persistance le plus utilisé, open source



## ■ Fonction 3 : Gérer les clients existants (objet métier)

Fichier de configuration de l'application

```
@Configuration
@EnableJpaRepositories(basePackages = { "org.proxib" })
@EnableTransactionManagement
@PropertySource("classpath:application.properties")
@EnableAspectJAutoProxy
@ComponentScan(basePackages = { "org.proxib" })
public class ApplicationConfig {
```

Création en base de données

```
@Entity
public class Client implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name= "idClient")
    private Long idClient;
    private String firstName;
    private String lastName;
    private String address;
    private String email;

    @OneToOne(cascade = { CascadeType.ALL })
    @JoinColumn(name = "currentAccount_id")
    private CurrentAccount currentAccount;
```

## ■ Fonction 3 : Gérer les clients existants (objet métier)

Table Client

```
SELECT * FROM 'client'
```

☐ Tout afficher

Nombre de lignes :































25

Filtrer les lignes:

Trier sur l'index:

Aucune

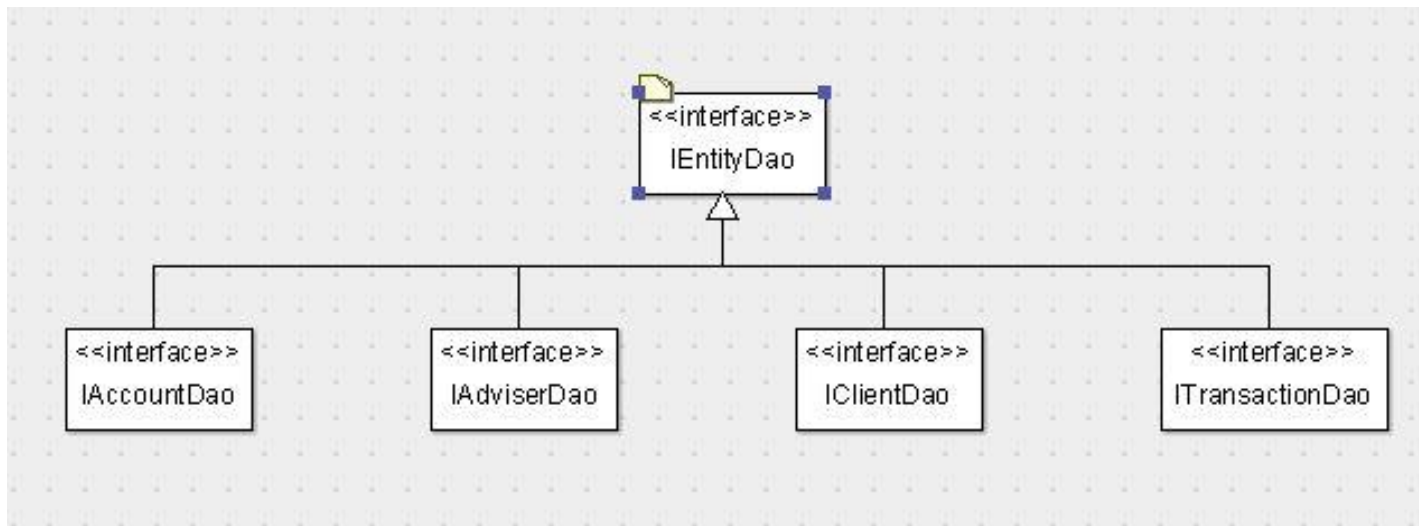
+ Options

| ← T →                    |  |  |   | idClient | address                                  | email                   | firstName     | lastName  | adviser_id | currentAccount_id | savingAccount_id |
|--------------------------|--|--|---|----------|--|-------------------------|---------------|-----------|------------|-------------------|------------------|
| <input type="checkbox"/> |  Modifier   |  Copier   |  Effacer   | 1        | 17 rue des oliviers 75001 Paris          | paul.leroy@gmail.com    | Leroy         | Paul      | 1          | 1                 | 2                |
| <input type="checkbox"/> |  Modifier   |  Copier   |  Effacer   | 2        | 1 rue des rateaux paris                  | bobby@marley            | marley        | bobby     | 1          | 3                 | 4                |
| <input type="checkbox"/> |  Modifier   |  Copier   |  Effacer   | 3        | 1 rue du paradis 75010 Paris             | paula.lis@gmail.com     | Lis           | Paula     | 2          | 5                 | 6                |
| <input type="checkbox"/> |  Modifier   |  Copier   |  Effacer   | 4        | 10 rue des nénuphars paris               | bob@marley              | marley        | bob       | 2          | 7                 | NULL             |
| <input type="checkbox"/> |  Modifier   |  Copier   |  Effacer   | 5        | Grande Bretagne                          | harry.potter@gmail.com  | Potter        | Harry     | 2          | 8                 | NULL             |
| <input type="checkbox"/> |  Modifier |  Copier |  Effacer | 6        | 5 avenue des champs Elysées 75007 Paris  | jean.dujardin@gmail.com | Dujardin      | Jean      | 3          | 9                 | 10               |
| <input type="checkbox"/> |  Modifier |  Copier |  Effacer | 7        | Alexandrie                               | a.legrand@gmail.com     | Le grand      | Alexandre | 3          | 11                | 12               |
| <input type="checkbox"/> |  Modifier |  Copier |  Effacer | 8        | Gaulle                                   | r.lion.r@gmail.com      | Coeur de Lion | Richard   | 3          | 13                | NULL             |
| <input type="checkbox"/> |  Modifier |  Copier |  Effacer | 9        | 86 boulevard des batignolles 75007 Paris | a.lamy@gmail.com        | Lamy          | Alexandra | 4          | 14                | NULL             |
| <input type="checkbox"/> |  Modifier |  Copier |  Effacer | 10       | Italia                                   | ricuu.r@gmail.com       | Ricci         | Riccu     | 4          | 15                | NULL             |

## ■ Fonction 3 : Gérer les clients existants (couche persistance)

Choix de méthodes génériques :

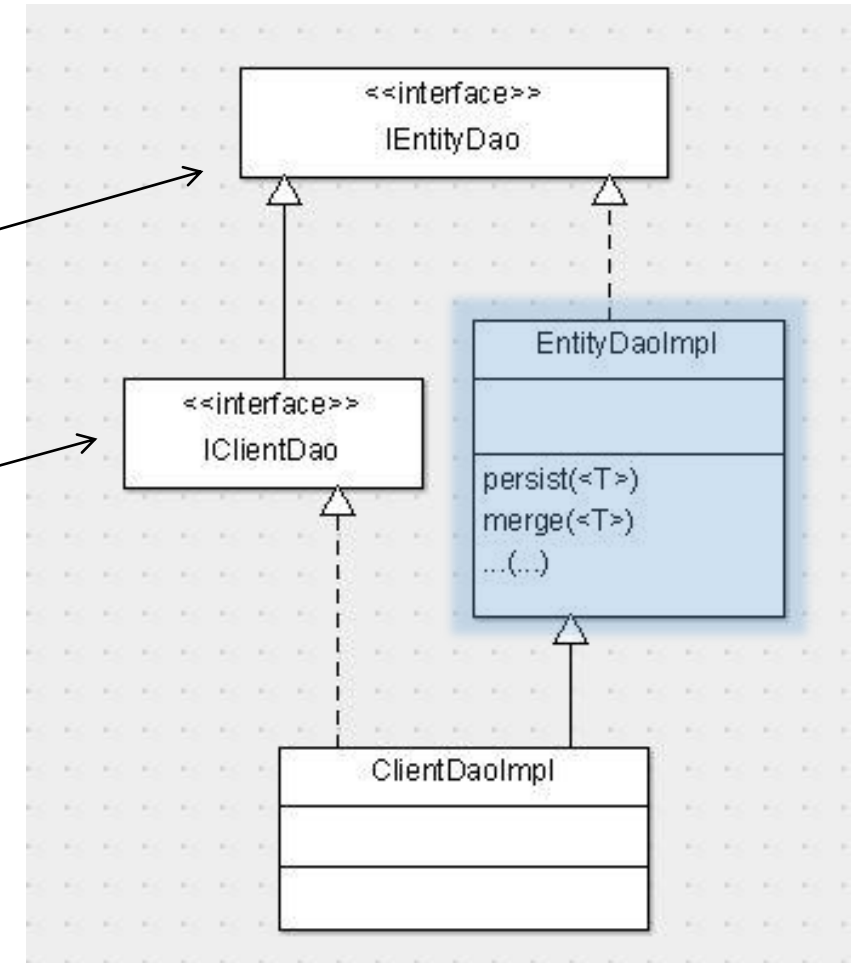
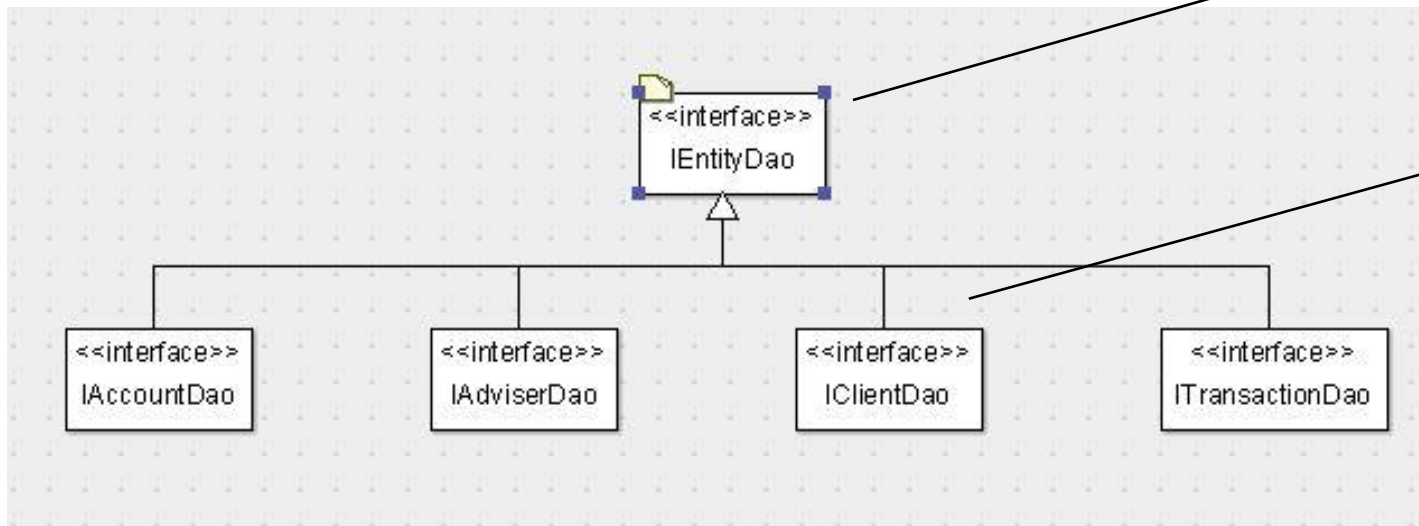
- Code réutilisable pour tous types d'objets



## ■ Fonction 3 : Gérer les clients existants (couche persistance)

Choix de méthodes génériques :

- Code réutilisable pour tous types d'objets
- Maintenance facilitée

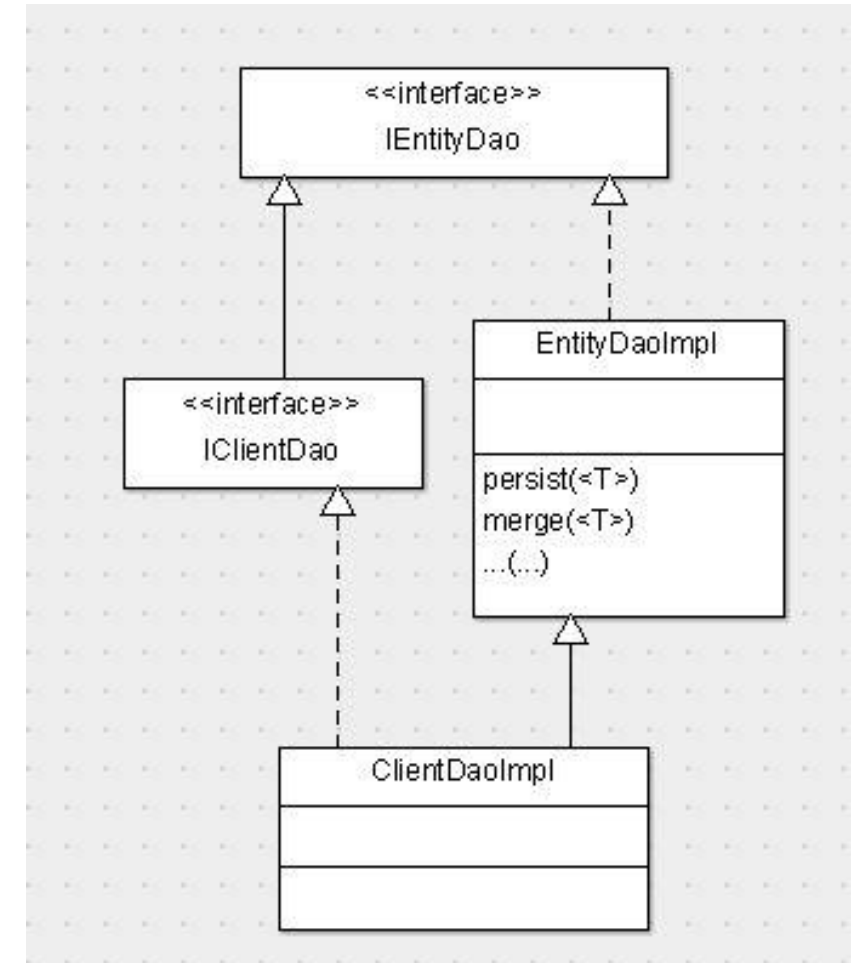


## ■ Fonction 3 : Gérer les clients existants (couche persistance)

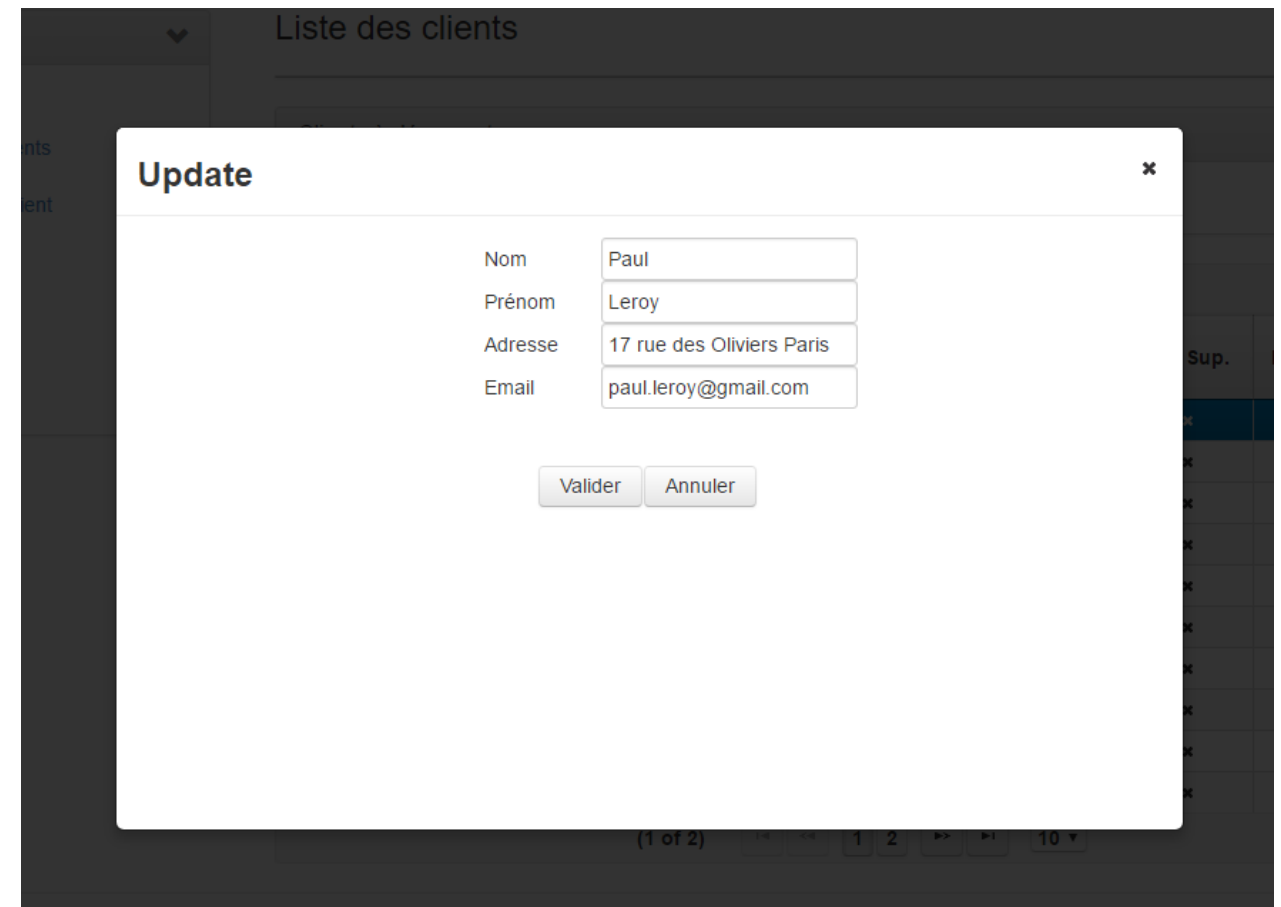
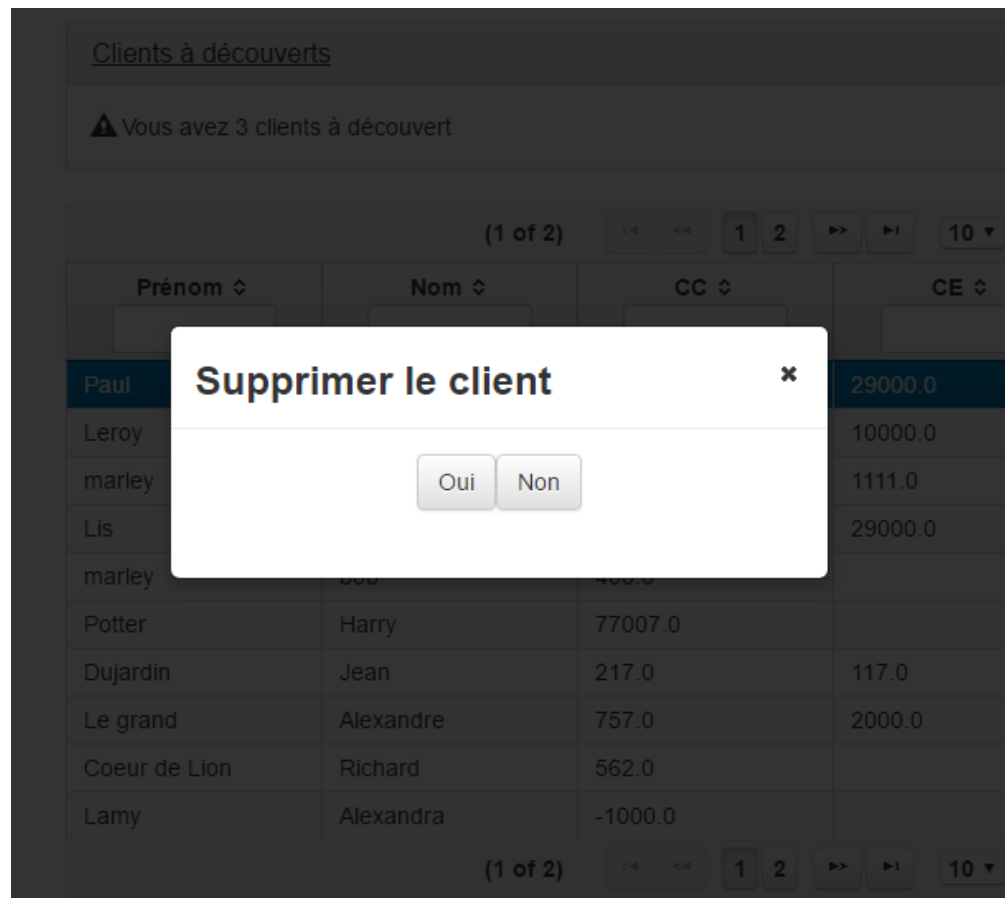
Choix de méthodes génériques :

- Code réutilisable pour tous types d'objets
- Maintenance facilitée

```
public class EntityDaoImpl<E> implements IEntityDao<E> {  
  
    @Transactional  
    public void persist(E e) throws HibernateException {  
        getEntityManager().persist(e);  
    }  
}
```



## ■ Fonction 3 : Gérer les clients existants



## ■ Fonction 3 : Gérer les clients existants (couche service)

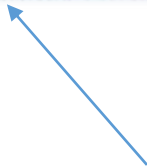
### Couche de persistance

```
public class EntityDaoImpl<E> implements IEntityDao<E> {  
  
    @Transactional  
    public void persist(E e) throws HibernateException {  
        getEntityManager().persist(e);  
    }  
}
```

### Couche service

```
@Service("serviceClient")  
public class ServiceClient implements IClientService {  
  
    @Autowired  
    IClientDao clientDao;  
  
    @Override  
    public void persist(Client client) throws Exception {  
        clientDao.persist(client);  
    }  
}
```

Méthode de la couche de  
persistance



## ■ Fonction 4 : virement

- Besoin client (conseiller) :
  - Effectuer un virement de compte à compte, interne à la banque
- Solution retenue
  - Création d'une page dédiée qui possède deux listes avec des champs de saisie

ProxiBanque

Logout

Liste des Clients  
Ajouter un client  
Virement  
Bilan financier

### Virement

| Prénom | Nom    | Solde | Sélectionner |
|--------|--------|-------|--------------|
| Paul   | Leroy  | 400   | (o)          |
| Lisa   | Paul   | 55    | (o)          |
| Bob    | Marley | 500   | @            |

| Prénom | Nom   | Solde | Sélectionner |
|--------|-------|-------|--------------|
| Paul   | Leroy | 400   | (o)          |
| Lisa   | Paul  | 55    | (o)          |

Compte à débiter : Bob Marley

Compte à créditer : Lisa Paul

50 €

Ok Effacer

VIREMENT  
Etes-vous sûr ?  
Oui Non



# PROXIBANQUE : PRESENTATION DE L'APPLICATION

Besoin client

Equipe – Plan

Conception

Fonctionnalités (4/5)

Bilan

Conclusion

ProxiBanque

Logout

Menu



Actions clients

Liste des clients

Ajouter un client

Transactions

Virement

Bilan

Découverts

## Virement

### Compte à débiter

| Nom ↕                | Prénom ↕             | Solde    | Débit |
|----------------------|----------------------|----------|-------|
| <input type="text"/> | <input type="text"/> |          |       |
| Homme                | Jean                 | 133508.0 | +     |
| Head                 | Radio                | 108780.0 | +     |
| Aurélie              | Po                   | 3654.0   | +     |
| Ghania               | B                    | 2100.0   | +     |
| Moche                | Poule                | 123.0    | +     |
| Kevin                | Coco                 | 1230.0   | +     |

### Compte à Créditer

| Nom ↕                            | Prénom ↕             | Solde | Crédit |
|----------------------------------|----------------------|-------|--------|
| <input type="text"/>             | <input type="text"/> |       |        |
| Sélectionner un compte à débiter |                      |       |        |

## Synthèse

| Comptes sélectionnés |                   |       |                       |
|----------------------|-------------------|-------|-----------------------|
|                      | Compte à débiter  | Solde |                       |
|                      | Compte à créditer | Solde |                       |
|                      | 0,0               | €     | Valider Reset Refresh |

### Compte à débiter

| Nom ↕                | Prénom ↕             | Solde   | Débit |
|----------------------|----------------------|---------|-------|
| <input type="text"/> | <input type="text"/> |         |       |
| Lis                  | Paula                | 49700.0 | +     |
| marley               | bob                  | 400.0   | +     |
| Potter               | Harry                | 77007.0 | +     |
| Dujardin             | Jean                 | 1317.0  | +     |
| Le grand             | Alexandre            | 757.0   | +     |
| Coeur de Lion        | Richard              | 512.0   | +     |
| Lamy                 | Alexandra            | -99.0   | +     |
| Ricci                | Riccu                | -2.0    | +     |

### Compte à Créditer

| Nom ↕                | Prénom ↕             | Solde   | Crédit |
|----------------------|----------------------|---------|--------|
| <input type="text"/> | <input type="text"/> |         |        |
| bob                  | marley               | 400.0   | +      |
| Harry                | Potter               | 77007.0 | +      |
| Jean                 | Dujardin             | 1317.0  | +      |
| Alexandre            | Le grand             | 757.0   | +      |
| Richard              | Coeur de Lion        | 512.0   | +      |
| Alexandra            | Lamy                 | -99.0   | +      |
| Riccu                | Ricci                | -2.0    | +      |

## Synthèse

Aurélie POTIER – Samuel BOUCHET  
Ghania BOUZEMAME - Maëva SOULABAILLE

## ■ Fonction 4 : virement

- Méthode de virement
  - Appel sur la page xhtml
  - Transfert de l'appel par le contrôleur
  - Action de la méthode de virement

```
<b:buttonGroup>
  <b:commandButton id="btn_add" value="Valider" update=":form1"
    resetValues="montant" action="#{virementController.transfer}" />
  <b:commandButton id="btn_reset" value="Reset" type="reset" />
  <b:commandButton id="btn_refresh" value="Refresh"
    action="#{virementController.refreshList}" />
</b:buttonGroup>
```

xhtml

```
public void transfer() {

    try {
        accountService.transfer(selectedClientDebit.getCurrentAccount(), selectedCli
            montant);
        notificationSuccess("Virement Effectué");
        refreshList();
    } catch (Exception e) {
        notificationError(e, "suppression Client");
    }
}
```

contrôleur

```
@Override
public String transfer(Account accountToWithdraw, Account accountToCredit, double sum) {

    DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");
    Date date = new Date();

    if (sum <= 0.0) {

        throw new RuntimeException("Erreur : La somme est inférieure ou égale à 0");
    } else if (accountToWithdraw == accountToCredit) {
        throw new RuntimeException("Erreur : memes comptes ");
    } else if (sum >= accountToWithdraw.getBalance()-authorizedOverdraft) {
        throw new RuntimeException("Erreur : Somme supérieure au montant de votre compte en banque !");
    } else {
        accountToWithdraw.setBalance(-sum);
        accountToCredit.setBalance(sum);
        try {
            accountDao.merge(accountToWithdraw);
            accountDao.merge(accountToCredit);

            Transaction transaction = new Transaction();
            transaction.setDate(date);
            transaction.setAccountToWithdrawId(accountToWithdraw.getId());
            transaction.setAccountToCreditId(accountToCredit.getId());
            transaction.setAmount(sum);
            transactionDao.persist(transaction);

            return "Virement effectué";
        } catch (Exception e) {
            return "probleme Exception dans virement dans serviceAccount";
        }
    }
}
```

Méthode de la  
couche service

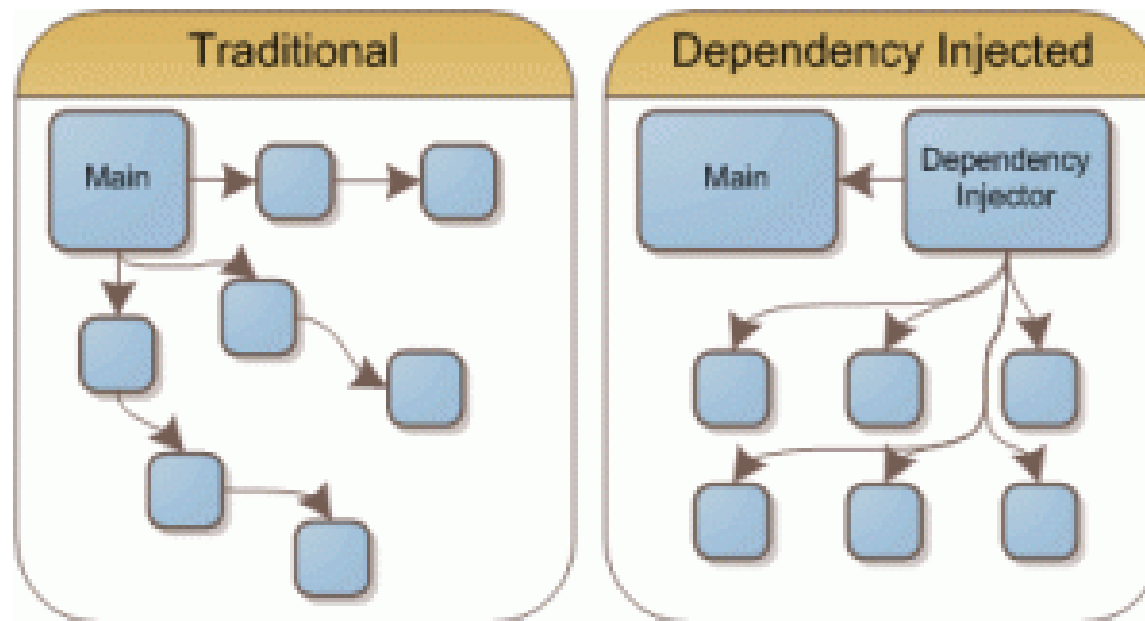
## ■ Fonction 4 : virement

- Technologie mise en œuvre : PrimeFaces
  - Utilisation de la librairie (appel de composants)

```
<h:outputText value="Compte courant"
    style="padding-left: 30px; padding-right: 30px;" />
<p:inputText type="number" style="text-align: right"
    id="currentAccount" value="#{clientController.balanceCurrent}"
    label="Compte Courant" required="true">
    <f:validateLongRange minimum="1" />
</p:inputText>
<p:message for="currentAccount"></p:message>
```

## ■ Fonction 4 : virement

- Technologie mise en œuvre : SPRING & Injection de dépendances
  - Inversion de contrôle
  - Modularité



## ■ **Fonction 4 : virement**

- Technologie mise en œuvre : SPRING
  - Définition
    - Framework
    - Conteneur léger
    - Injection de dépendance
    - Programmation orienté aspect
  - Avantages
    - Réutilisabilité
    - Testabilité
    - Productivité
    - Gestion des injections des composants

## ■ Fonction 4 : virement

- Technologie mise en œuvre : SPRING & Injection de dépendances
  - Mise en place

```
@Configuration
@EnableJpaRepositories(basePackages = { "org.proxib" })
@EnableTransactionManagement
@PropertySource("classpath:application.properties")
@EnableAspectJAutoProxy
@ComponentScan(basePackages = { "org.proxib" })
public class ApplicationConfig {
```

```
    @Configuration
    @ComponentScan(basePackages = { "org.proxib" })
    @Component(value = "virementController")
    @Autowired
```

```
    @Component(value = "virementController")
    @SessionScoped
    public class VirementController implements Serializable {

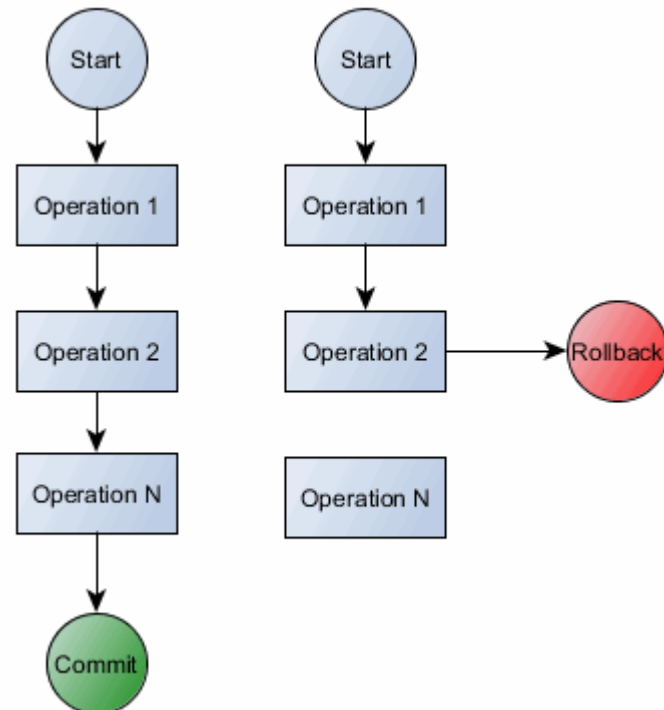
        private static final long serialVersionUID = 1L;

        private static Logger LOGGER = LoggerFactory.getLogger(ClientController.class);

        @Autowired
        IClientService clientService;
```

## ■ Fonction 4 : virement

- Technologie mise en œuvre : SPRING Transaction Management
  - @Transactional



```

@Configuration
@EnableJpaRepositories(basePackages = { "org.proxib" })
@EnableTransactionManagement
@PropertySource("classpath:application.properties")
@EnableAspectJAutoProxy
@ComponentScan(basePackages = { "org.proxib" })
public class ApplicationConfig {

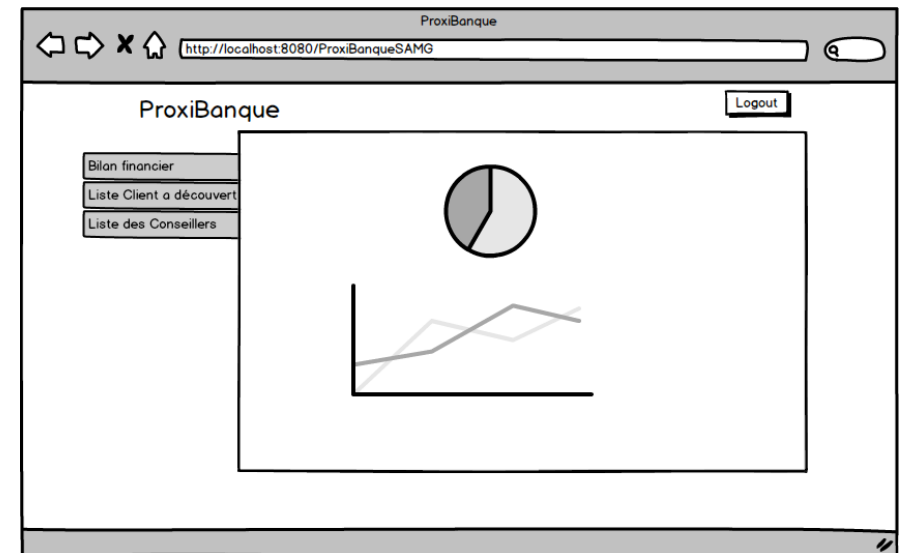
    public class EntityDaoImpl<E> implements IEntityDao<E> {

        @PersistenceContext(unitName = "persistenceUnit")
        protected EntityManager entityManager;

        @Transactional
        public void persist(E e) throws HibernateException {
            entityManager.persist(e);
        }
    }
}
    
```

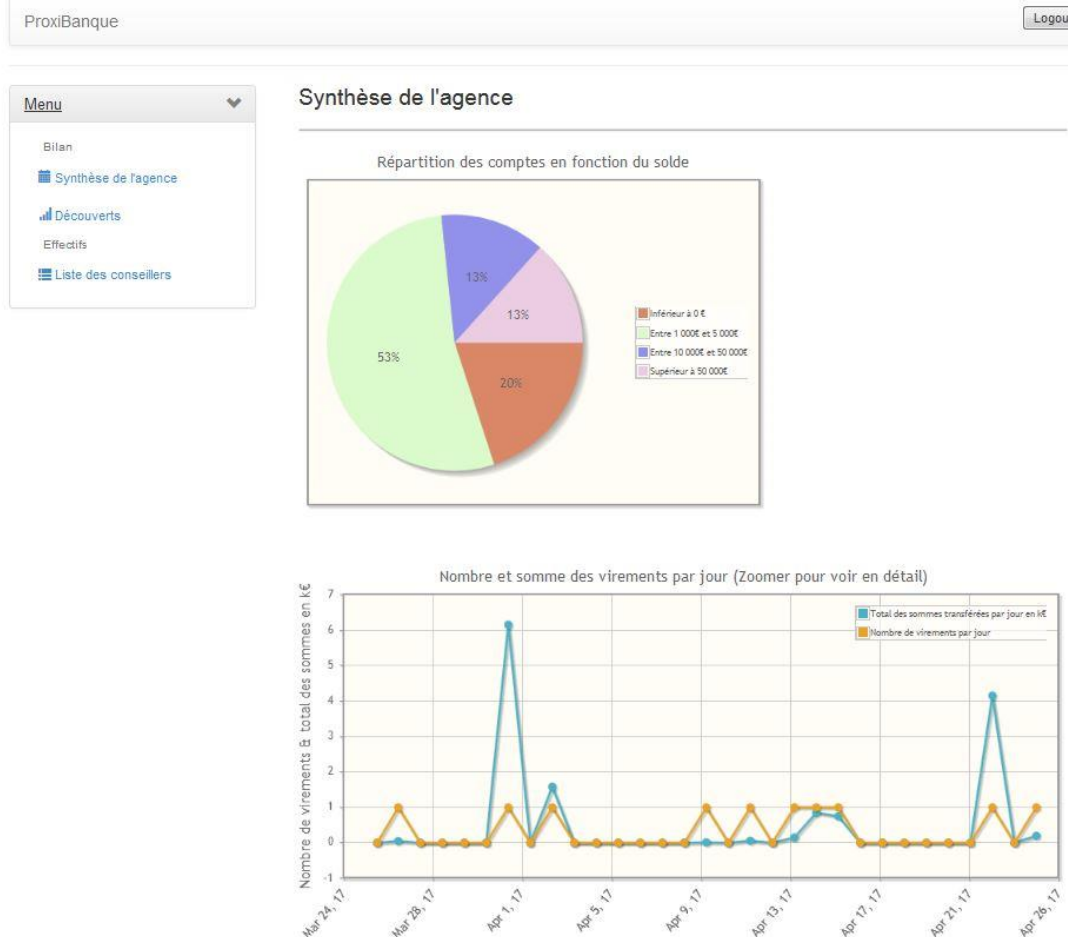
## ■ Fonction 5 : suivi des transactions

- Besoin client (directeur) :
  - Visualiser graphiquement un historique des transactions
- Solution retenue
  - Enregistrer chaque transaction en base de données
  - L'inscrire dans un fichier de log
  - Afficher graphiquement l'historique





## ■ Fonction 5 : suivi des transactions



## ■ Fonction 5 : suivi des transactions

Nature de la transaction : Virement compte à compte

Date : 2017-04-24 08:52:26.0

Montant : 70.0€

Compte débité n° : 32769

Compte crédité n° : 32768

Nature de la transaction : Virement compte à compte

Date : 2017-04-24 15:33:24.0

Montant : 700.0€

Compte débité n° : 54554

Compte crédité n° : 75456

Nature de la transaction : Virement compte à compte

Date : 2017-04-24 15:58:19.0

Montant : 250.0€

Compte débité n° : 65540

Compte crédité n° : 65542

```
@Configuration
@EnableJpaRepositories(basePackages = { "org.proxib" })
@EnableTransactionManagement
@PropertySource("classpath:application.properties")
@EnableAspectJAutoProxy
@ComponentScan(basePackages = { "org.proxib" })
public class ApplicationConfig {
```

```
indAll().size() - 1).getDate());
```

```
ize() - 1).getAmount() + "€");
```

```
vId());
```

```
ize() - 1).getAccountToCreditId()
```

## ■ Bilan

| Fonctionnel   | Modulaire  | Securisé  | Maintenable   |
|---|--|---|---|
|  |  |  |  |

## ■ Perspectives

- Sécurité
- Création compte
- Virement compte épargne
- Webservice