guide-programmation

Table des matières

Introduction

Ce document présente la conception de l'application, c'est à dire les cas d'utilisation pour les différents acteurs. Il présente aussi
l'architecture logicielle qu'on a implémenté pour satisfaire aux besoins des différentes couches applicatives.

La partie "Persistance" présente comment l'application accède aux données et comment la persistance est configurée.

La partie traitements "Traitements métier" présente la partie métier de l'application et comment la base de données est initialisée dès le déploiement.

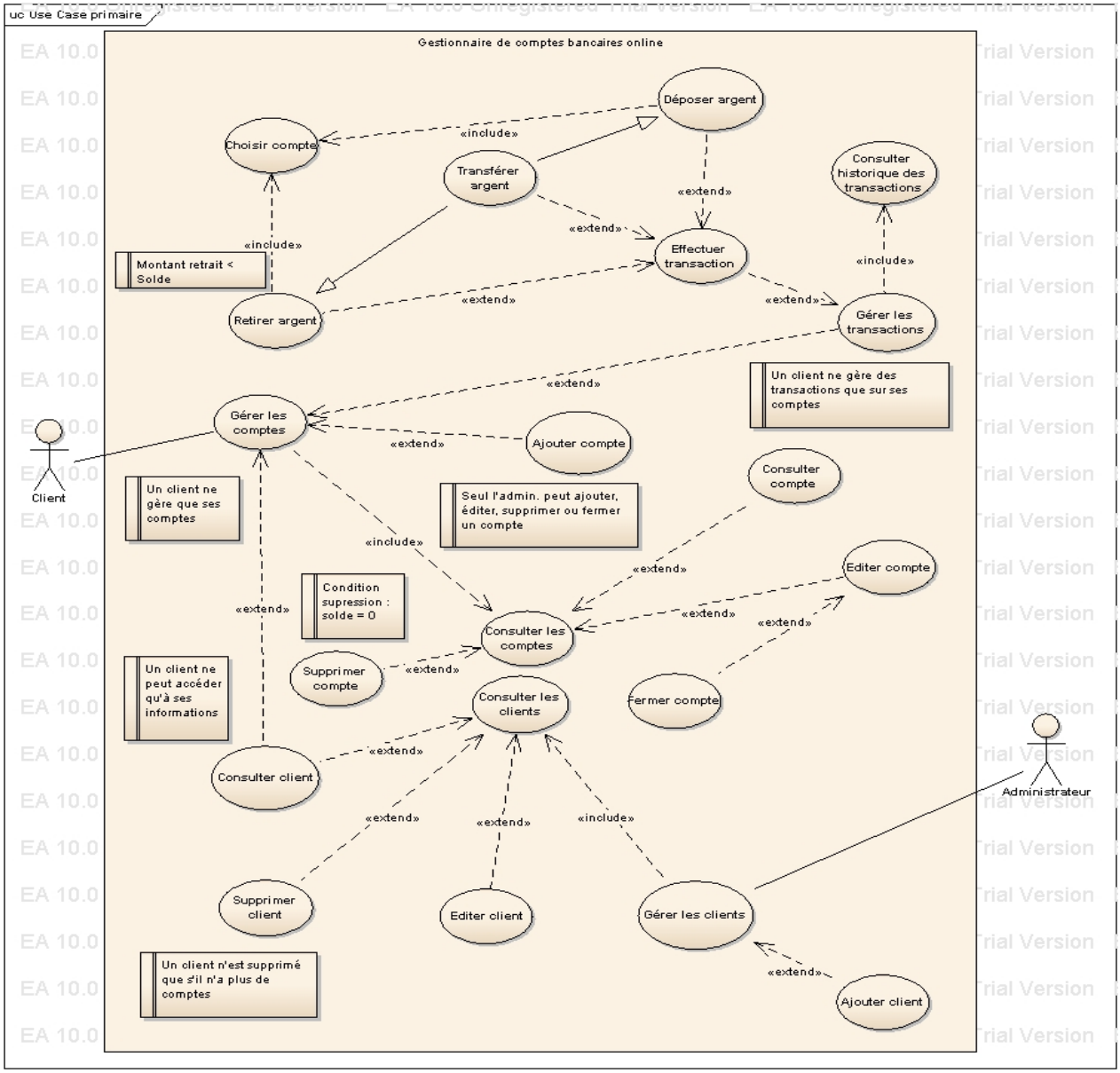
La "Présentation" présente comment les interfaces utilisateurs sont implémentées , comment elle accède à la partie métier, et comment la validation et les messages d'information et d'erreur sont traités et l'internationalisation.

Conception

Dans cette partie on présentera une petite conception définie à partir de l'énoncé du projet. Cela permettra la compréhension
de la logique de l'application. Le modèle objet servira aussi pour la partie persistance (les classes entités avec les annotations JPA)
donc pour la partie des traitements métier (les EJB session).

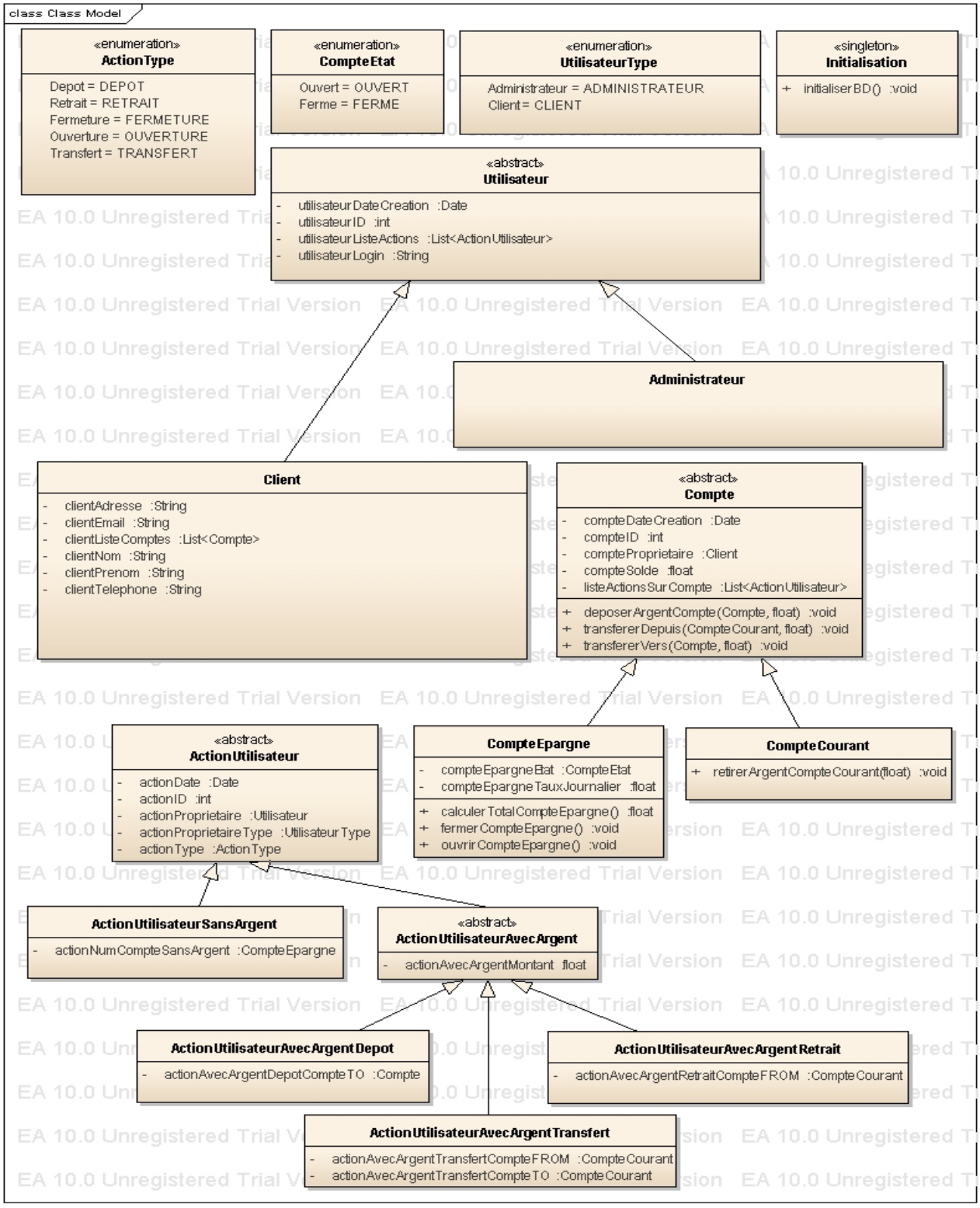
Cas d'utilisation

Voici un diagramme qui représente les fonctionnalités de l'application, les acteurs et leurs rôles.



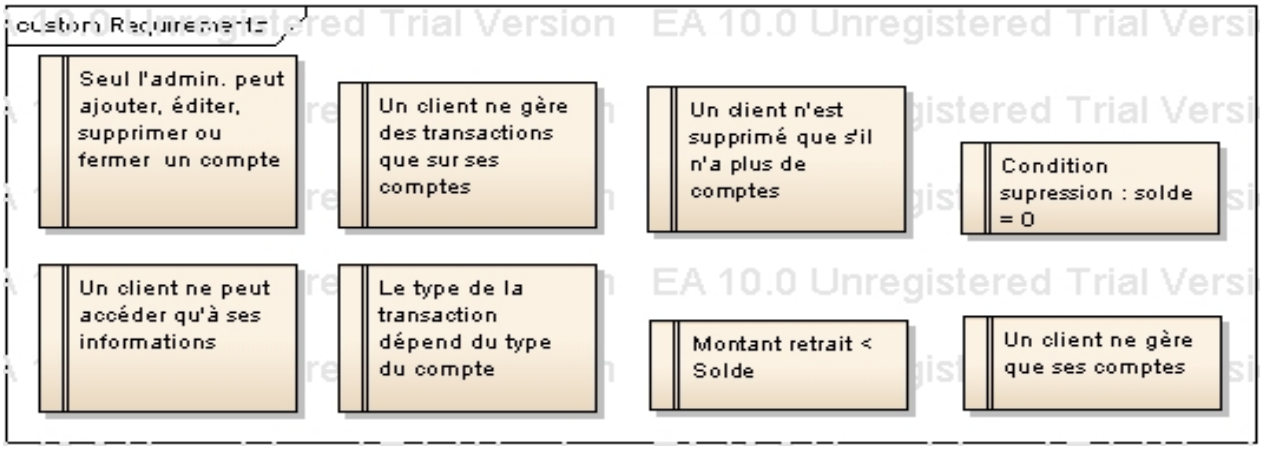
Modélisation objet

Le diagramme de classe suivant représente la modélisation métier UML de l'application.



Exigences de base

Ci dessous les principales exigences demandées. Nous les avons extraites de l'énoncé du projet. Celle ci concerne essentiellement
fonctionnalités de l'application.



Architecture logicielle et technique

Cette partie montre comment les fonctionnalités de l'application sont implémentées.
La partie persistance donne une description de comment l'application accède aux données.
La partie traitements métier décrit comment l'application utilise EJB pour effectuer les traitements métier et accéder aux données.
La partie présentation décrit comment l'application interagit avec les utilisateurs et comment elle accède à la partie traitements métier.

Persistance

JPA : configuration avec annotations

1. Stratégie d'héritage utilisée : une seule table par hiérarchie de classe + discriminateur

Exemple pour la Classe Utilisateur et Client :

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)

public abstract class Utilisateur implements Serializable {

@Entity
@DiscriminatorValue("CLI")

public class Client extends Utilisateur implements Serializable {
```

2. Stratégie de génération des clés : C'est le fournisseur de persistance qui s'en charge

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int utilisateurID;
```

3. Gestion des relations one-to-one, one-to-many et many-to-one

//BUG

Java DB et ressources GlassFish

1. Configuration de la persistance (fichier persistence.xml)

```
<persistence-unit name="djbrabatbPU" transaction-type="JTA">
    <jta-data-source>jdbc/djbrabatb</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
        <property name="javax.persistence.schema-generation.database.action" value="create"/>
    </properties>
</persistence-unit>
```

2. Ressource JDBC et pool de connexions (glassfish-ressources.xml)

```
<resources>
    <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread="false" connection-creation-retry-attempts="0"
        <property name="serverName" value="localhost"/>
        <property name="portNumber" value="1527"/>
        <property name="databaseName" value="djbrabatb"/>
        <property name="User" value="root"/>
        <property name="Password" value="root"/>
        <property name="URL" value="jdbc:derby://localhost:1527/djbrabatb"/>
        <property name="driverClass" value="org.apache.derby.jdbc.ClientDriver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/djbrabatb" object-type="user" pool-name="derby_net_djbrabatb_rootPool"/>
</resources>
```


Traitements métier

Design pattern Abstract Façade : EJBs métier et opérations CRUD

Les EJB sont utilisés pour implémenter les opérations métier et les accès à la base de données.

```
protected abstract EntityManager getEntityManager();

public void create(T entity) {
    getEntityManager().persist(entity);
}

public void edit(T entity) {
    getEntityManager().merge(entity);
}

public void remove(T entity) {
    getEntityManager().remove(getEntityManager().merge(entity));
}

public T find(Object id) {
    return getEntityManager().find(entityClass, id);
}
```


Design pattern Singleton : EJB Initialisation BD

Pour initialiser la base, on utilise un EJB singleton, qui va être instancié une fois au déploiement de l'application. Et va exécuter la méthode
initialiserDB() annotée par @PostConstruct, c'est à dire après que l'injection de dépendence soit faite.

```
@Singleton
@Startup
public class InitialisationBD {

@PostConstruct
public void initialiserBD() {
```


Présentation

Templates et templateClients

On a utilisé les facelets / facelets templates clients pour factoriser des traitements liés à plusieurs pages.


```
<ui:composition template="./../templates/templateClient.xhtml">

    <ui:define name="content">
```

Backing beans

Les backings beans sont utilisés dans les pages xhtml via des expressions EL pour accéder à des propriétés ou appeler des méthodes
.On peut injecter des EJB pour accéder aux données de la base et effectuer des traitements métier.

-Injection d'un EJB

```
/**
 * LoginManagedBean contient le client ou l'administrateur connecté (administrateurLoged/clientLoged)
 */
@Named("loginManagedBean")
@SessionScoped
public class LoginManagedBean implements Serializable {

    /**
     *
     */
    @EJB
    private AdministrateurFacade administrateurFacade;
```

-Injection d'un autre backing bean

```
@Inject
LoginManagedBean loginManagedBean;
```

```
/**
 * tous les clients
 */
List<Client> tousLesClients;
```


Validation

On a utilisé une validation dans les backing beans pour une validation coté serveur, et la validation par des tags dans le xhtml pour la validation coté client.

-validation dans les beans

```
/**
 * login de l'utilisateur voulant se connecter
 */
@Size(min = 5, max = 25, message = "Entrez votre login (Taille : 5-25 caracatères)")
private String requesterUtilisateurLogin;
```

-validation dans le xhtml

```
<p:outputLabel for="loginInputText" value="Entrez votre Login : "/>
<p:inputText id="loginInputText"
             label="loginInputText"
             value="#{loginManagedBean.requesterUtilisateurLogin}"
             required="true" />
<h:message for="loginInputText"/>
```


Injection de contexte et de dépendances

La CDI permet d'injecter des beans dans d'autres bean? par exemple on injecte un bean session dans un bean requete.

```
/**
 *
 */
@Inject
LoginManagedBean loginManagedBean;
```

et injecter les EJB pour effectuer les traitements métier => c'est le conteneur d'EJB qui fournit les instances.

```
/**
 *
 */
@EJB
private ClientFacade clientFacade;
```


Messages d'erreur et d'information

On ajoute des messages d'erreur mais aussi des messages d'informations en cas de succès.
On peut aussi utiliser la classe flash pour afficher des messages de succès après une redirection

```
FacesContext.getCurrentInstance().getExternalContext().getFlash().setKeepMessages(true);
FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO, "Bonjour "
```


Internationalisation

Un backing bean de portée session est chargé de modifier la Locale de l'application dans le cas où l'utilisateur change de langue.
On utilise deux fichier properties pour contenir les chaînes de caractères traduites. Et la configuration dans faces-config.xml pour définir
la Locale par défaut et les chemins des fichier.

-Beans de portée session

-événement écouteur pour le clic sur le changement de langue

```
public void changeLocale(ValueChangeEvent e){

    if(e.getNewValue().toString().equalsIgnoreCase(Locale.ENGLISH.toString())){
    FacesContext.getCurrentInstance().getViewRoot().setLocale(new Locale(Locale.ENGLISH.toString()));
    }
```

- la configuration de l'i18n

```
<locale-config>
    <default-locale>fr</default-locale>
    <supported-locale>en</supported-locale>
</locale-config>
<resource-bundle>
    <base-name>/messages_fr_FR</base-name>
    <var>msg</var>
</resource-bundle>
```


Sécurité

La sécurité de l'application repose sur le fait d'entrer son login sans le password comme demandé. Le login est unique. On ne travaille pas non plus avec les paramètres de vue donc les informations ne circulent pas dans les URL puisq'on utilise des EJB sessions pour stocker certaines informations.