```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package predictif.service;

import java.security.MessageDigest;
import java.util.ArrayList;
import java.util.GregorianCalendar;
import java.util.List;
import javax.persistence.EntityTransaction;
import javax.persistence.NoResultException;
import predictif.dao.ClientDao;
import predictif.dao.EmployeDao;
import predictif.dao.HoroscopeDao;
import predictif.dao.MediumDao;
import predictif.dao.PredictionDao;
import predictif.dao.SigneAstrologiqueDao;
import predictif.modele.Client;
import predictif.modele.Employe;
import predictif.modele.Horoscope;
import predictif.modele.Medium;
import predictif.modele.Predictions.AmourPrediction;
import predictif.modele.Predictions.Prediction;
import predictif.modele.Predictions.SantePrediction;
import predictif.modele.Predictions.TravailPrediction;
import predictif.modele.SigneAstrologique;
import predictif.Util.JpaUtil;

/**
 *
 * @author Administrateur
 */
public class Service
{

    public enum PredictionType
    {

        TRAVAIL, SANTE, AMOUR, TOUS
    };
    protected ClientDao clientDao;
    protected HoroscopeDao horoscopeDao;
    protected SigneAstrologiqueDao signeDao;
    protected EmployeDao employeDao;
    protected MediumDao mediumDao;
    protected PredictionDao predictionDao;

    public Service()
    {
        clientDao = ClientDao.getInstance();
        horoscopeDao = HoroscopeDao.getInstance();
        signeDao = SigneAstrologiqueDao.getInstance();
        employeDao = EmployeDao.getInstance();
        mediumDao = MediumDao.getInstance();
        predictionDao = PredictionDao.getInstance();
    }

    /**
     * Permet de créer un nouveau client en BD à partir des paramètres passés.
     * L'attribut referent du client est valorisé via la méthode findMinReferent()
     * @see findMinReferent()
     * @param nom
     * @param prenom
     * @param adresse
     * @param email
     * @param tel
     * @param dateNaissance
     * @param mediums la liste des mediums favoris sous forme d'une <code>List</code>
```

```
     * de<code> Medium </code>
     */
    public boolean createClient(String nom, String prenom, String adresse, String email,
String tel, GregorianCalendar dateNaissance, List<Medium> mediums)
    {
        boolean status = false;

        Employe referent = findMinReferent();

        Client leClient = new Client(nom, prenom, adresse, email, tel, dateNaissance,
calculateSigne(dateNaissance), mediums, referent);
        referent.getClients().add(leClient);

        EntityTransaction tx = null;

        JpaUtil.openEntityManager();

        try
        {
            tx = JpaUtil.getEntityManagerTransaction();
            tx.begin();
            employeDao.update(referent);
            tx.commit();
            status = true;
        }
        catch (Exception e)
        {
            if (tx != null && tx.isActive())
            {
                tx.rollback();
            }
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return status;
        }
    }

    /**
     * Permet de créer un nouveau client en BD à partir des paramètres passés. Ne devrait pas
     * être utilisé dans l'application finale mais juste pour remplir la BD.
     * @param nom
     * @param prenom
     * @param adresse
     * @param email
     * @param tel
     * @param dateNaissance
     * @param mediums
     * @param referent
     */
    public boolean createClient(String nom, String prenom, String adresse, String email,
String tel, GregorianCalendar dateNaissance, List<Medium> mediums, Employe referent)
    {
        boolean status = false;
        Client leClient = new Client(nom, prenom, adresse, email, tel, dateNaissance,
calculateSigne(dateNaissance), mediums, referent);
        referent.addClient(leClient);

        EntityTransaction tx = null;

        JpaUtil.openEntityManager();

        try
        {
            tx = JpaUtil.getEntityManagerTransaction();
            tx.begin();
            employeDao.update(referent);
            tx.commit();
```

```
            status = true;
        }
        catch (Exception e)
        {
            if (tx != null && tx.isActive())
            {
                tx.rollback();
            }
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return status;
        }
    }

    /**
     * Permet de mettre à jour l'état interne du client présent en BD
     * @param client le client que l'on souhaite mettre à jour
     * @return true si la mise à jour s'est bien passée, faux sinon
     */
    public boolean updateClient(Client client)
    {
        boolean status = false;

        Client autreVersionClient = retrieveClient(client.getNumClient());

        if (client.isBirthModified(autreVersionClient))
        {
            client.setSigneAstrologique(calculateSigne(client.getDateNaissance()));
        }

        JpaUtil.openEntityManager();
        EntityTransaction tx = null;

        try
        {
            tx = JpaUtil.getEntityManagerTransaction();
            tx.begin();
            clientDao.update(client);
            tx.commit();
            status = true;
        }
        catch (Exception e)
        {
            System.err.println("Erreur rencontrée à updateClient : " + e.toString());
            if (tx != null && tx.isActive())
            {
                tx.rollback();
            }
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return status;
        }
    }

    /**
     * Méthode permettant de supprimer un client du moyen de persistance
     * @param client le client que l'on veut supprimer de la bd
     * @return true si la suppression s'est bien passée, faux sinon
     */
    public boolean deleteClient(Client client)
    {
        boolean status = false;
        JpaUtil.openEntityManager();
        EntityTransaction tx = null;
```

```
        try
        {
            tx = JpaUtil.getEntityManagerTransaction();
            tx.begin();

            clientDao.deleteClient(client);
            tx.commit();
            status = true;
        }
        catch (Exception e)
        {
            System.err.println("Erreur rencontrée au deleteClient : " + e.toString());
            if (tx != null && tx.isActive())
            {
                tx.rollback();
            }
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return status;
        }
    }

    /**
     * Méthode permettant de retrouver l'employé possédant le minimum de clients
     * @return
     */
    private Employe findMinReferent()
    {
        Employe referent = null;

        JpaUtil.openEntityManager();

        try
        {
            referent = employeDao.findMinusEmploye();
        }
        catch (Exception e)
        {
            System.err.println("Erreur rencontrée au findMinReferent : " + e.toString());
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return referent;
        }
    }

    /**
     * Méthode permettant de récuperer le signe astrologique correspondant à une
     * date de naissance.
     * Utile pour maintenir l'IHM à jour en cas de changement de date de naissance
     * Utile également lors de la création d'un nouveau client
     * @param dateNaissance une date de naissance
     * @return le <code>SigneAstrologique</code> correspondant à la date de naissance
     */
    public SigneAstrologique calculateSigne(GregorianCalendar dateNaissance)
    {
        SigneAstrologique signe = null;
        JpaUtil.openEntityManager();

        try
        {
            signe = signeDao.retrieve(dateNaissance);
        }
        catch (Exception e)
        {
            System.err.println("Erreur rencontrée au calculateSigne : " + e.toString());
```

```
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return signe;
        }
    }

    /**
     * Méthode permettant de récupérer tous les clients présent
     * dans le moyen de persistance
     * @return la liste de clients sous la forme d'une <code>List<Client></code> la valeur de
     * retour peut valoir null si une erreur s'est passée sinon contient au minimum une liste
     * vide.
     */
    public List<Client> getAllClients()
    {
        JpaUtil.openEntityManager();

        List<Client> clients = null;

        try
        {
            clients = clientDao.findAllClient();
        }
        catch (Exception e)
        {
            System.err.println("Erreur getAllClients:" + e.getMessage());
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return clients;
        }
    }

    /**
     * Méthode permettant de retrouver une occurence de <code>Client</code>
     * à partir de son identifiant
     * @param num l'identifiant sous la forme d'un int
     * @return le client <code>Client</code>
     */
    public Client retrieveClient(int num)
    {
        Client client = null;
        JpaUtil.openEntityManager();
        try
        {
            client = clientDao.retrieveClient(num);
        }
        catch (Exception e)
        {
            System.err.println("Erreur rencontrée au retrieveClient : " + e.toString());
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return client;
        }
    }

    public boolean createHoroscope(AmourPrediction amour, TravailPrediction travail,
SantePrediction sante, Medium mediumChoisi, Client leClient)
    {
        boolean status = false;
        Horoscope horo = new Horoscope(mediumChoisi, amour, travail, sante, leClient);
        leClient.addHoroscope(horo);
        JpaUtil.openEntityManager();
```

```
        EntityTransaction tx = null;

        try
        {
            tx = JpaUtil.getEntityManagerTransaction();
            tx.begin();
            clientDao.update(leClient);
            tx.commit();
            status = true;
        }
        catch (Exception e)
        {
            System.err.println("Erreur rencontrée au createHoroscope : " + e.toString());
            if (tx != null && tx.isActive())
            {
                tx.rollback();
            }
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return status;
        }
    }

    /**
     * Méthode permettant de vérifier l'identité de l'employé. Il pourrait etre
     * judicieux d'appeller cette méthode au travers d'une autre classe Service
     * beaucoup plus proche de l'IHM web et contenant un boolean connected permettant
     * de contrôler les méthodes pouvant être appelés.
     * @param codeEmploye le code de l'employé
     * @param passwd hashed
     */
    public boolean connectEmploye(int codeEmploye, String passwd)
    {
        boolean status = false;
        JpaUtil.openEntityManager();
        try
        {
            MessageDigest messageDigest = MessageDigest.getInstance("MD5");
            byte[] mdp = messageDigest.digest(passwd.getBytes());
            Employe employe = employeDao.findEmploye(codeEmploye);
            if (MessageDigest.isEqual(mdp, employe.getPassword()))
            {
                status = true;
                //Connexion Ok, le booleen status pourrait valoriser un booléen
                //une variable instance de classe afin de contrôler les actions
                //disponibles
            }
        }
        catch (NoResultException e)
        {
//            System.err.println("Erreur connexion, employe/mdp inexistant : " +
e.getMessage());
        }
        catch (Exception e)
        {
            System.err.println("Erreur technique au connectEmploye : " + e.getMessage());
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return status;
        }
    }

    /**
     * Méthode permettant de récuperer toutes les prédictions d'un type
     * particulier. Types acceptés:
```

```
     * <ul>
     *  <li>Service.PredictionType.AMOUR</li>
     *  <li>Service.PredictionType.TRAVAIL</li>
     *  <li>Service.PredictionType.SANTE</li>
     *  <li>Service.PredictionType.TOUS permet de récuperer tous les types
     * précédemment cités</li>
     * </ul>
     * @param type le type de prédiction souhaité, variable de type enum
     * @return la <code>liste de <code>Prediction</code>
     */
    public List<Prediction> getPrediction(PredictionType type)
    {
        List<Prediction> predictions = null;

        JpaUtil.openEntityManager();
        try
        {
            switch (type)
            {
                case AMOUR:
                    predictions = predictionDao.getAllPredictionAmour();
                    break;
                case TRAVAIL:
                    predictions = predictionDao.getAllPredictionTravail();
                    break;
                case SANTE:
                    predictions = predictionDao.getAllPredictionSante();
                    break;
                case TOUS:
                    predictions = predictionDao.getllAllPredictions();
                    break;
                default:
                    ;
            }
        }
        catch (Exception e)
        {
            System.err.println("Erreur dans le getPrediction " + e.getMessage());
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return predictions;
        }
    }

    /**
     * Permet de récupérer l'ensemble des mediums présent en BD sous la forme
     * d'une liste
     * @return la <code> List Medium</code>
     */
    public List<Medium> getAllMediums()
    {
        List<Medium> mediums = null;
        JpaUtil.openEntityManager();

        try
        {
            mediums = mediumDao.findAll();
        }
        catch (Exception e)
        {
            System.err.println("erreur getAllMediums service : " + e.getMessage());
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return mediums;
        }
```

```
    }

    /**
     * Permet de récuperer un horoscope selon son identifiant. Le formatage de son
     * état interne peut être facilement obtenu en utilisant la méthode toString()
     * de cet objet.
     * @param num son identifiant unique
     * @return l'horoscope en question ou null si l'identifiant n'existe pas
     */
    public Horoscope getHoroscope(int num)
    {
        Horoscope horo = null;

        JpaUtil.openEntityManager();

        try
        {
            horo = horoscopeDao.getHoroscope(num);
        }
        catch (Exception e)
        {
            System.err.println("Erreur dans getDetailsHoroscope(int) : "+e.getMessage());
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return horo;
        }

    }

    /**
     * Permet de récuperer une <code>List</code> d'<code>Horoscope</code> selon la
     * date d'insertion dans la base de données (l'insertion se faisant juste après
     * sa création).
     * @param dateInsertion la date pour laquelle on souhaite les horoscopes
     * @return une <code>List</code> contenant les entités ou vide si aucun horoscope
     * n'existe pour la date demandée
     */
    public List<Horoscope> getHoroscope(GregorianCalendar dateInsertion)
    {
        List<Horoscope> horos = new ArrayList<Horoscope>();

        JpaUtil.openEntityManager();

        try
        {
            horos = horoscopeDao.getHoroscopes(dateInsertion);
        }
        catch(Exception e)
        {
            System.err.println("Erreur dans getDetailsHoroscope(Calendar) :
"+e.getMessage());
        }
        finally
        {
            JpaUtil.closeEntityManager();
            return horos;
        }
    }

}
```