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

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.SecureRandom;
import java.security.Security;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Arrays;
import java.util.Random;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;


/**
 *
 * @author rapha
 */
public class IdentityServer {


    public static void main(String[] args)
            throws IOException, ClassNotFoundException, InstantiationException,
            IllegalAccessException, SQLException
    {
        Security.addProvider(
            new org.bouncycastle.jce.provider.BouncyCastleProvider()
        );

        ServerSocket server_sock = new ServerSocket(Config.IDENTITY_PORT);

        Class.forName("com.mysql.jdbc.Driver").newInstance();
        String url = "jdbc:mysql://"+Config.MYSQL_HOST+"/frontier";
        Connection con = DriverManager.getConnection(url, "ferryinpres", "pass");

        ExecutorService pool = Executors.newFixedThreadPool(12);

        for (;;) {
            System.out.println("En attente d'un nouveau client");
            Socket sock = server_sock.accept();
            System.out.println("Nouveau client");

            pool.execute(new ServerThread(sock, con));
        }
    }
    private Connection _conn;
}
```

```java
class ServerThread implements Runnable {
    private Socket _sock;
    private Connection _con;

    public ServerThread(Socket sock, Connection con)
    {
        this._sock = sock;
        this._con = con;
    }

    @Override
    public void run()
    {
        try {
            ObjectInputStream in = new ObjectInputStream(
                this._sock.getInputStream()
            );
            ObjectOutputStream out = new ObjectOutputStream(
                this._sock.getOutputStream()
            );

            SecretKey sessionKey = this.keyExchange(in, out);

            // Crée les instances de cryptage et de décryptage
            Cipher cryptor = Cipher.getInstance("DES/ECB/PKCS5Padding");
            cryptor.init(Cipher.ENCRYPT_MODE, sessionKey);
            Cipher decryptor = Cipher.getInstance("DES/ECB/PKCS5Padding");
            decryptor.init(Cipher.DECRYPT_MODE, sessionKey);

            login(in, out, cryptor, decryptor);
        } catch (Exception e) {
            System.err.println("Fermeture de la connexion: ");
            e.printStackTrace();
        }

        try {
            this._sock.close();
        } catch (IOException e) { }
    }

    public SecretKey keyExchange(ObjectInputStream in, ObjectOutputStream out)
            throws IOException, NoSuchAlgorithmException,
            NoSuchPaddingException, InvalidKeyException,
            ClassNotFoundException, NoSuchProviderException,
            IllegalBlockSizeException, BadPaddingException
    {

        // Recoit la clé publique
        KeyExchangeClient query = (KeyExchangeClient) in.readObject();

        // Génère la clé de session
        KeyGenerator gen = KeyGenerator.getInstance("DES");
        gen.init(new SecureRandom());
        SecretKey sessionKey = gen.generateKey();

        // Envoie le clé de session
        //Cipher rsaCryptor = Cipher.getInstance("RSA/ECB/PKCS#1");
        Cipher rsaCryptor = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        rsaCryptor.init(Cipher.ENCRYPT_MODE, query.getPublicKey());
        out.writeObject(
            new KeyExchangeServer(
                rsaCryptor.doFinal(sessionKey.getEncoded())
            )
        );
        out.flush();

        return sessionKey;
    }
```

```
    private void login(ObjectInputStream in, ObjectOutputStream out,
            Cipher cryptor, Cipher decryptor) throws IOException,
            IllegalBlockSizeException, BadPaddingException,
            ClassNotFoundException, SQLException, NoSuchAlgorithmException
    {
        // Génère et envoie le sel
        int serverSalt = (new Random()).nextInt();

        byte[] d = Utils.cryptObject(new LoginServer(serverSalt), cryptor);
        System.out.println("Length: "+d.length);
        out.writeObject(
            d
        );
        out.flush();
        System.out.println("Salt: "+ serverSalt);

        // Attends les informations d'authentification
        LoginClient response = (LoginClient) Utils.decryptObject(
            (byte[]) in.readObject(), decryptor
        );

        // Extrait le mot de passe de la base de données
        PreparedStatement instruc = this._con.prepareStatement(
            "SELECT mot_de_passe " +
            "FROM agent " +
            "WHERE nom = ?"
        );
        instruc.setString(1, response.getName());
        ResultSet rs = instruc.executeQuery();
        byte[] hashedPassword;
        if (rs.next()) {
            String password = rs.getString("mot_de_passe");
            hashedPassword = Utils.hashPassword(
                password, response.getClient_salt(), serverSalt
            );
        } else {
            hashedPassword = null;
        }

        if (hashedPassword != null
            && Arrays.equals(hashedPassword, response.getPassword_hashed())) {
            // Mot de passe valide
            out.writeObject(Utils.cryptObject(new Ack(), cryptor));
            out.flush();

            verifId(in, out, cryptor, decryptor);
        } else {
            // Mot de passe invalide
            out.writeObject(Utils.cryptObject(new Fail(), cryptor));
            out.flush();
        }
    }

    private void verifId(ObjectInputStream in, ObjectOutputStream out,
            Cipher cryptor, Cipher decryptor) throws IOException,
            ClassNotFoundException, IllegalBlockSizeException,
            BadPaddingException, SQLException
    {
        for (;;) {
            VerifId query = (VerifId) Utils.decryptObject(
                (byte[]) in.readObject(), decryptor
            );

            // Vérifie si la personne est valide
            PreparedStatement instruc = this._con.prepareStatement(
                "SELECT COUNT(*) AS existe " +
                "FROM voyageur " +
```

```
                    "WHERE id_national = ? AND nom = ? AND prenom = ?"
                );
                instruc.setInt(1, query.getClientNationalId());
                instruc.setString(2, query.getClientName());
                instruc.setString(3, query.getClientSurname());
                ResultSet rs = instruc.executeQuery();
                rs.next();

                if (rs.getInt("existe") != 0) {
                    out.writeObject(Utils.cryptObject(new Ack(), cryptor));
                } else {
                    out.writeObject(Utils.cryptObject(new Fail(), cryptor));
                }
                out.flush();
            }
        }
}
```