package ca.polymtl.inf4410.tp1.server;

import java.rmi.ConnectException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

import java.security.MessageDigest;

import java.util.Arrays;

import java.util.HashMap;

import ca.polymtl.inf4410.tp1.shared.ServerInterface;

public class Server implements ServerInterface {

// Données des fichiers

private HashMap<String,byte[]> \_data;

// Données de verrou

private HashMap<String,Integer> \_lockedInfo;

// Prochain clientId disponible

private int \_nextId;

public static void main(String[] args) {

Server server = new Server();

server.run();

}

public Server() {

super();

\_nextId = 0;

\_data = new HashMap<String,byte[]>();

\_lockedInfo = new HashMap<String,Integer>();

}

private void run() {

if (System.getSecurityManager() == null) {

System.setSecurityManager(new SecurityManager());

}

try {

ServerInterface stub = (ServerInterface) UnicastRemoteObject

.exportObject(this, 0);

Registry registry = LocateRegistry.getRegistry();

registry.rebind("server", stub);

System.out.println("Server ready.");

} catch (ConnectException e) {

System.err

.println("Impossible de se connecter au registre RMI. Est-ce que rmiregistry est lancé ?");

System.err.println();

System.err.println("Erreur: " + e.getMessage());

} catch (Exception e) {

System.err.println("Erreur: " + e.getMessage());

}

}

/\*

\* Méthodes accessible par RMI.

\*/

// Méthode qui renvoie le prochain clientId disponible

public Integer generateclientid(){

\_nextId++;

return new Integer(\_nextId);

}

// Crée un fichier sur le serveur

public boolean create(String nom){

if(!\_data.containsKey(nom)){

\_data.put(nom, null);

\_lockedInfo.put(nom, null);

return true;

}

return false;

}

// Renvoie les données sur les fichiers présents ainsi que l'état des verrous

public HashMap<String,Integer> list(){

return \_lockedInfo;

}

// Renvoie les noms et contenus des fichiers. Le client s'occupe de gérer l'écriture locale

public HashMap<String, byte[]> syncLocalDir(){

return \_data;

}

// Renvoie le fichier "nom" s'il est présent sur le serveur et sa checksum est différente de celle passée en paramètre

public byte[] get(String nom, byte[] checksum){

if(\_data.containsKey(nom) && !Arrays.equals(checksum, getChecksum(nom))){

return \_data.get(nom);

}

return null;

}

// Verrouille le fichier "nom" si ce dernier est présent sur le serveur et qu'il n'est pas déjà verrouillé. Si le fichier est verrouillé avec succès, la clé TRUE est utilisée

// De plus, si la checksum est différente de celle passée en paramètre, les données du fichier sont envoyés dans la valeur de la map

// Si le fichier sur le serveur est vide, on n'écrase pas la copie locale si le client possède déjà cette derniere

public HashMap<Boolean,byte[]> lock(String nom, Integer clientid, byte[] checksum){

HashMap<Boolean,byte[]> rValue = new HashMap<Boolean,byte[]>();

if(\_data.containsKey(nom) && (\_lockedInfo.get(nom) == null || \_lockedInfo.get(nom).equals(clientid))){

// Si le fichier est présent et disponible

\_lockedInfo.replace(nom, clientid);

if(!Arrays.equals(checksum, getChecksum(nom))){

// Si le checksum est différent on renvoie le fichier

rValue.put(Boolean.TRUE,\_data.get(nom));

return rValue;

}

rValue.put(Boolean.TRUE,null);

return rValue;

}

rValue.put(Boolean.FALSE,null);

return rValue;

}

// On remplace le contenu du fichier sur le serveur par le contenu passé en paramètre et on débarre le fichier

public boolean push(String nom, byte[] contenu, Integer clientid){

if(\_data.containsKey(nom) && \_lockedInfo.get(nom) != null && \_lockedInfo.get(nom).equals(clientid)){

//Si le fichier est verrouillé par le bon utilisateur et existe sur le serveur

\_data.replace(nom, contenu);

\_lockedInfo.replace(nom, null);

return true;

}

return false;

}

// Calcul du checksum MD5 des données passées en paramètre

private byte[] getChecksum(String nom){

MessageDigest md;

try {

md = MessageDigest.getInstance("MD5");

md.update(\_data.get(nom));

return md.digest();

} catch (Exception e) {

return null;

}

}

}

package ca.polymtl.inf4410.tp1.shared;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.HashMap;

public interface ServerInterface extends Remote {

public Integer generateclientid() throws RemoteException;

public boolean create(String nom) throws RemoteException;

public HashMap<String,Integer> list() throws RemoteException;

public HashMap<String, byte[]> syncLocalDir() throws RemoteException;

public byte[] get(String nom, byte[] checksum) throws RemoteException;

public HashMap<Boolean,byte[]> lock(String nom, Integer clientid, byte[] checksum) throws RemoteException;

public boolean push(String nom, byte[] contenu, Integer clientid) throws RemoteException;

}