Seel Océane Fink Jérôme

tECHNOLOGIE DE   
l’e-commerce

Evaluation 3

# Code Serveur Compta

**Création de la connexion SSL**

1. **public** **class** Serveur\_SAMOP **extends** Thread{
3. **private** **final** **int** port;
4. **private** **final** SourceTaches tachesAExecuter;
5. **private** SSLServerSocket SSocket = **null**;
6. **private** **final** **int** nbrThreads;
8. **public** Serveur\_SAMOP(**int** p, SourceTaches st, **int** nt)
9. {
10. port = p;
11. tachesAExecuter = st;
12. nbrThreads = nt;
13. }
15. @Override
16. **public** **void** run()
17. {
18. **try**
19. {
20. //Ouverture d'une socket SSL
21. KeyStore ServerKs = KeyStore.getInstance("JKS");
22. FileInputStream fichierKeyStore  = **new** FileInputStream("keystores/serveur\_ssl\_KS");
24. **char**[] password\_keystore = "sslserveur".toCharArray();
26. ServerKs.load(fichierKeyStore, password\_keystore);
28. //Recuperation du contexte SSL
29. SSLContext SllC = SSLContext.getInstance("SSLv3");
31. KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
32. kmf.init(ServerKs, password\_keystore);
34. TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
35. tmf.init(ServerKs);
37. //Init du contexte SSL
38. SllC.init(kmf.getKeyManagers(), tmf.getTrustManagers(), **null**);
40. SSLServerSocketFactory socketfactory = SllC.getServerSocketFactory();
42. SSocket = (SSLServerSocket) socketfactory.createServerSocket(port);
44. }
45. **catch**(IOException e)
46. {
47. System.err.println("Serveur\_SAMOP : Erreur de la creation de socket  : " + e);
48. } **catch** (KeyStoreException | NoSuchAlgorithmException | CertificateException ex) {
49. Logger.getLogger(Serveur\_SAMOP.**class**.getName()).log(Level.SEVERE, **null**, ex);
50. } **catch** (UnrecoverableKeyException ex) {
51. Logger.getLogger(Serveur\_SAMOP.**class**.getName()).log(Level.SEVERE, **null**, ex);
52. } **catch** (KeyManagementException ex) {
53. Logger.getLogger(Serveur\_SAMOP.**class**.getName()).log(Level.SEVERE, **null**, ex);
54. }
56. **for**(**int** i = 0; i < nbrThreads; i++)
57. {
58. ThreadTraitement tt = **new** ThreadTraitement(tachesAExecuter);
59. tt.start();
60. }
62. Socket CSocket = **null**;
64. **while**(!isInterrupted())
65. {
66. **try**
67. {
68. System.out.println("Serveur\_SAMOP : Attente d'un client sur le port " + port);
69. CSocket = SSocket.accept();
70. System.out.println("Serveur\_SAMOP : Client dispo");
71. }
72. **catch**(IOException e)
73. {
74. System.err.println("Serveur\_SAMOP : Erreur d'accept : " + e);
75. }
77. tachesAExecuter.recordTache(**new** Runnable\_SAMOP(CSocket));
78. }
79. }
80. }

**Implémentation du protocole**

1. **public** **class** Runnable\_SAMOP **implements** Runnable{
3. **private** Socket CSocket = **null**;
4. **private** DataInputStream dis = **null**;
5. **private** DataOutputStream dos = **null**;
6. **private** BeanBDAccess beanOracle;

9. **public** Runnable\_SAMOP(Socket s)
10. {
11. CSocket = s;
13. **try**
14. {
15. dis = **new** DataInputStream(**new** BufferedInputStream(CSocket.getInputStream()));
16. dos = **new** DataOutputStream(**new** BufferedOutputStream(CSocket.getOutputStream()));
17. }
18. **catch**(IOException e)
19. {
20. System.err.println("Runnable\_SAMOP : Host non trouvÃ© : " + e);
21. }
23. beanOracle = **new** BeanBDAccess();
24. **try**
25. {
26. beanOracle.connexionOracle("localhost", 1521, "COMPTA", "COMPTA", "XE");
27. }
28. **catch** (ClassNotFoundException ex)
29. {
30. System.err.println("Runnable\_SAMOP : Class not found " + ex.getMessage());
31. }
32. **catch** (SQLException ex)
33. {
34. System.err.println("Runnable\_SAMOP : SQL Exception (Oracle)" + ex.getMessage());
35. }
36. **catch** (connexionException ex)
37. {
38. System.err.println("Runnable\_SAMOP : " + ex.getNumException() + " -- " + ex.getMessage());
39. }
40. }
42. @Override
43. **public** **void** run() {
45. String[] parts = (ReceiveMsg()).split("#");
47. **if**(parts[0].equals("LOGIN\_SSL"))
48. {
49. **if**(!login(parts))
50. **return**;
51. }
52. **else**
53. {
54. SendMsg("ERR#Requete invalide");
55. **return**;
56. }
58. **boolean** terminer = **false**;
60. **while**(!terminer)
61. {
62. parts = ReceiveMsg().split("#");
63. **switch** (parts[0])
64. {
65. **case** "LAUNCH\_PAYEMENT" :
66. launchPayement(parts);
67. **break**;
69. **case** "LAUNCH\_PAYEMENTS" :
70. launchPayements();
71. **break**;
73. **case** "ASK\_PAYEMENTS" :
74. askPayements(parts);
75. **break**;
77. **default** :
78. terminer = **true**;
79. **break**;
80. }
81. }
83. System.err.println("Runnable\_SAMOP : Fin du runnable");
85. **try**
86. {
87. CSocket.close();
88. }
89. **catch** (IOException ex)
90. {
91. System.err.println("Runnable\_SAMOP : Erreur de close : " + ex.getMessage());
92. }
93. }

96. **private** **boolean** login(String[] requete)
97. {
98. String login = requete[1];
99. String password = requete[2];
101. **try** {
102. //Recuperation des infos dans la BD
103. ResultSet rs = beanOracle.selection("PASSWORD, FONCTION", "PERSONNEL", "LOGIN = '" + login + "'");
105. **if**(!rs.next())
106. {
107. SendMsg("ERR#Login incorrecte");
108. **return** **false**;
109. }
111. String pwdBD = rs.getString("PASSWORD");
112. String fonction = rs.getString("FONCTION");
114. **if**(!fonction.equalsIgnoreCase("chef-comptable"))
115. {
116. SendMsg("ERR#Seul les chefs comptable ont acces Ã  cette application");
117. **return** **false**;
118. }
120. **if**(!password.equals(pwdBD))
121. {
122. SendMsg("ERR#Mot de pass invalide");
123. **return** **false**;
124. }
126. SendMsg("OK");
127. **return** **true**;
129. } **catch** (SQLException ex) {
130. SendMsg("ERR#Base de donnee inaccessible. Reessaye plus tard");
131. **return** **false**;
132. }
133. }



138. **private** **void** launchPayement(String[] requete)
139. {
140. String login = requete[1];
142. ResultSet rs;
143. **try** {
144. rs = beanOracle.selection("\*", "PERSONNEL", "LOGIN = '"+login+"'");
146. **if**(!rs.next())
147. {
148. SendMsg("ERR#Aucun employe pour ce login");
149. **return**;
150. }
152. String where = "LOGIN = '"+login+"' AND FLAG\_FICHE\_ENVOYEE = 0 AND FLAG\_VALIDE = 1";
153. rs = beanOracle.selection("\*", "SALAIRES", where);
154. String resultat = "";
156. **while**(rs.next())
157. {
158. resultat += rs.getString("LOGIN") + "  /  ";
159. resultat += rs.getString("MONTANT\_BRUT")+ "  /  ";
160. resultat += rs.getString("MOIS\_ANNEE")+ "#";
161. }
163. **if**(resultat.isEmpty())
164. {
165. SendMsg("ERR#Aucun payements a effectuer pour ce login");
166. **return**;
167. }
169. //MAJ des salaires
171. HashMap update = **new** HashMap();
173. update.put("FLAG\_FICHE\_ENVOYEE", 1);
174. update.put("FLAG\_SALAIRE\_VERSE", 1);
176. beanOracle.miseAJour("SALAIRES", update, where);
178. SendMsg("OK#"+resultat);
180. } **catch** (SQLException | requeteException ex) {
181. ex.printStackTrace();
182. SendMsg("ERR#Erreur BD");
183. }
184. }

187. **private** **void** launchPayements()
188. {
189. ResultSet rs;
190. **try** {
191. String where = "FLAG\_FICHE\_ENVOYEE = 0 AND FLAG\_VALIDE = 1";
192. rs = beanOracle.selection("\*", "SALAIRES", where);
193. String resultat = "";
195. **while**(rs.next())
196. {
197. resultat += rs.getString("LOGIN") + "  /  ";
198. resultat += rs.getString("MONTANT\_BRUT")+ "  /  ";
199. resultat += rs.getString("MOIS\_ANNEE")+ "#";
200. }
202. **if**(resultat.isEmpty())
203. {
204. SendMsg("ERR#Aucun payements a effectuer pour ce login");
205. **return**;
206. }
208. //MAJ des salaires
210. HashMap update = **new** HashMap();
212. update.put("FLAG\_FICHE\_ENVOYEE", 1);
213. update.put("FLAG\_SALAIRE\_VERSE", 1);
215. beanOracle.miseAJour("SALAIRES", update, where);
217. SendMsg("OK#"+resultat);
219. } **catch** (SQLException | requeteException ex) {
220. ex.printStackTrace();
221. SendMsg("ERR#Erreur BD");
222. }
223. }

226. **private** **void** askPayements(String[] requete)
227. {
228. //On va recuperer toutes les fiches de salaire pour une date
229. String where = "MOIS\_ANNEE = '"+requete[1]+"' AND FLAG\_FICHE\_ENVOYEE = 1";
231. **try** {
232. ResultSet rs = beanOracle.selection("\*", "SALAIRES", where);
234. String listResultat = "";
235. **while**(rs.next())
236. {
237. listResultat += rs.getString("LOGIN") + "   /   ";
238. listResultat += rs.getString("MONTANT\_BRUT")+"#";
239. }
241. **if**(listResultat.isEmpty())
242. {
243. SendMsg("ERR#Aucun salaire verse ce mois");
244. **return**;
245. }
247. SendMsg("OK#"+listResultat);
249. } **catch** (SQLException ex) {
250. ex.printStackTrace();
251. SendMsg("ERR#Base de donnees hors ligne");
252. }
253. }

256. /\* Envoi d'un message au client \*/
257. **public** **void** SendMsg(String msg)
258. {
259. String chargeUtile = msg;
260. **int** taille = chargeUtile.length();
261. StringBuffer message = **new** StringBuffer(String.valueOf(taille) + "#" + chargeUtile);
263. **try**
264. {
265. dos.write(message.toString().getBytes());
266. dos.flush();
267. }
268. **catch**(IOException e)
269. {
270. System.err.println("Runnable\_SAMOP : Erreur d'envoi de msg (IO) : " + e);
271. }
272. }

275. /\* RÃ©ception d'un message du client \*/
276. **public** String ReceiveMsg()
277. {
278. **byte** b;
279. StringBuffer taille = **new** StringBuffer();
280. StringBuffer message = **new** StringBuffer();
282. **try**
283. {
284. **while** ((b = dis.readByte()) != (**byte**)'#')
285. {
286. **if** (b != (**byte**)'#')
287. taille.append((**char**)b);
288. }
290. **for** (**int** i = 0; i < Integer.parseInt(taille.toString()); i++)
291. {
292. b = dis.readByte();
293. message.append((**char**)b);
294. }
295. }
296. **catch**(IOException e)
297. {
298. System.err.println("Runnable\_SAMOP : Erreur de reception de msg (IO) : " + e);
299. }
301. **return** message.toString();
302. }
304. }