SEEL Océane, FINK Jérôme

[Email address]

Programmation réseaux, web et mobiles

Evaluation 2

Table of Contents

**Code serveur trafic2**

TRAMAP2

BOOMAP9

**Code administareur serveur container17**

**Schéma relationnel BD\_COMPTA21**

**Schéma relationnel BD\_TRAFIC21**

**Trame chat UDP22**

Code du serveur trafic

1. **Partie TRAMAP :**
2. **public** **class** RunnableTraitementEntree **implements** Runnable
3. {
4. **private** Socket CSocket = **null**;
5. **private** DataInputStream dis = **null**;
6. **private** DataOutputStream dos = **null**;
7. **private** BeanBDAccess beanOracle;
9. **boolean** first = **true**;
11. **public** RunnableTraitementEntree(Socket s)
12. {
13. CSocket = s;
15. **try**
16. {
17. dis = **new** DataInputStream(**new** BufferedInputStream(CSocket.getInputStream()));
18. dos = **new** DataOutputStream(**new** BufferedOutputStream(CSocket.getOutputStream()));
19. }
20. **catch**(IOException e)
21. {
22. System.err.println("RunnableTraitement : Host non trouvÃ© : " + e);
23. }
25. beanOracle = **new** BeanBDAccess();
26. **try** {
27. beanOracle.connexionOracle("localhost", 1521, "TRAFIC", "TRAFIC", "XE");
28. } **catch** (ClassNotFoundException ex) {
29. System.err.println("Class not found " + ex.getMessage());
30. } **catch** (SQLException ex) {
31. System.err.println("SQL Exception (oracle)" + ex.getMessage());
32. } **catch** (connexionException ex) {
33. System.err.println(ex.getNumException() + " -- " + ex.getMessage());
34. }
35. }
37. @Override
38. **public** **void** run()
39. {
40. String[] parts = (ReceiveMsg()).split("#");
42. **if**(parts[0].equals("LOGIN"))
43. {
44. **if**(!login(parts))
45. **return**;
46. }
47. **else**
48. {
49. SendMsg("ERR#RequÃ¨te invalide");
50. **return**;
51. }
53. **boolean** terminer = **false**;
55. **while**(!terminer)
56. {
57. parts = ReceiveMsg().split("#");
58. **switch** (parts[0])
59. {
60. **case** "INPUT\_LORRY" :
61. inputLorry(parts);
62. **break**;
63. **case** "INPUT\_LORRY\_WITHOUT\_RESERVATION" :
64. inputLorryWithoutReserv(parts);
65. **break**;
66. **case** "LIST\_OPERATIONS" :
67. listOperation(parts);
68. **break**;
69. **case** "LOGOUT" :
70. terminer = **true**;
71. **break**;
73. **default** :
74. terminer = **true**;
75. **break**;
76. }
77. }
79. System.err.println("fin du runnable");
80. }
82. **private** **boolean** login(String[] part)
83. {
84. ResultSet rs = **null**;
86. **try**
87. {
88. rs = beanOracle.selection("PASSWORD", "UTILISATEURS", "LOGIN = '" + part[1]+"'");
89. }
90. **catch**(SQLException e){
91. System.err.println(e.getStackTrace());
92. }
94. String pwd = **null**;
96. **try** {
97. **if**(!rs.next())
98. {
99. SendMsg("ERR#Login invalide");
100. }
101. **else**
102. pwd = rs.getString("PASSWORD");
103. } **catch** (SQLException ex) {
104. System.err.println(ex.getStackTrace());
105. }
107. **if**(pwd.equals(part[2]))
108. {
109. SendMsg("ACK");
110. **return** **true**;
111. }
112. **else**
113. SendMsg("ERR#Mot de passe incorrecte");
115. **return** **false**;
116. }
118. **private** **void** inputLorry(String[] request)
119. {
121. ResultSet rs = **null**;
123. **try** {
124. rs = beanOracle.selection("ID\_CONTAINER", "CONTAINERS", "RESERVATION = '"+request[1]+"'");
125. } **catch** (SQLException ex) {
126. SendMsg("ERR#Base de donnÃ©e inaccessible");
127. System.err.println("Erreur SQL exception input lorry");
128. **return**;
129. }
131. **boolean** isResultEmpty = **true**;
132. **int** nbrElemParc = 0;
134. String[] idList =  request[2].split("@");
135. System.out.println(request[2]);
136. **try** {
137. **while**(rs.next())
138. {
139. nbrElemParc++;
140. isResultEmpty = **false**;
141. String curId = **null**;
142. String id = rs.getString("ID\_CONTAINER");
143. **boolean** invalidContainerID = **true**;
144. **for**(String s : idList)
145. {
146. System.out.println(s +"---"+id);
147. curId = s;
148. **if**(s.equals(id))
149. {
150. invalidContainerID = **false**;
151. **break**;
152. }
153. }
154. **if**(invalidContainerID)
155. {
156. SendMsg("ERR#Le container " + curId +" ne fait pas partie de la reservation" );
157. **return**;
158. }
160. **if**(nbrElemParc >= idList.length)
161. **break**;
162. }
163. } **catch** (SQLException ex) {
164. SendMsg("ERR#Base de donnÃ©e inaccessible");
165. System.err.println("Erreur SQL exception input lorry resultat");
166. **return**;
167. }
169. **if**(isResultEmpty)
170. {
171. SendMsg("ERR#Le numero de reservation demande n'existe pas");
172. **return**;
173. }
175. **try** {
176. rs = beanOracle.selection("X, Y", "PARC", "ETAT=1");
177. } **catch** (SQLException ex) {
178. SendMsg("ERR#Base de donnÃ©e inaccessible");
179. System.err.println("Erreur SQL exception input lorry" + ex.getStackTrace());
180. **return**;
181. }
183. String reponse = "ACK#";
185. **try**
186. {
187. **for**(**int** i = 0; i < idList.length ; i++)
188. {
189. **if**(rs.next())
190. {
191. reponse = reponse + idList[i] + "==>("+rs.getString("X")+";"+rs.getString("Y")+")@";
192. }
193. **else**
194. {
195. SendMsg("ERR#Erreur pas assez de places reservees");
196. **return**;
197. }
198. }
199. }
200. **catch**(SQLException ex){
201. SendMsg("ERR#Base de donnee inaccessible");
202. System.err.println("Erreur SQL exception input lorry" + ex.getStackTrace());
203. **return**;
204. }
206. SendMsg(reponse);
207. }

210. **private** **void** listOperation(String[] request)
211. {
212. String Select = "ID\_MOUVEMENT, MOUVEMENTS.ID\_CONTAINER, ID\_TRANSPORTEUR\_ENTRANT, DATE\_ARRIVEE, ID\_TRANSPORTEUR\_SORTANT, POIDS, DATE\_DEPART, DESTINATION, ID\_SOCIETE";
213. String From = "MOUVEMENTS INNER JOIN CONTAINERS ON MOUVEMENTS.ID\_CONTAINER = CONTAINERS.ID\_CONTAINER";
214. String Where = **null**;
216. **if**(request[1].equals("societe"))
217. Where = "CONTAINERS.ID\_SOCIETE = '"+request[2]+"'";
218. **if**(request[1].equals("destination"))
219. Where = "DESTINATION = '"+request[2]+"'";
220. **if**(request[1].equals("date"))
221. Where = "To\_date(DATE\_ARRIVEE, 'DD/MM/YYYY') BETWEEN To\_date('"+request[2]+"', 'DD/MM/YYYY') AND To\_date('"+request[3]+"', 'DD/MM/YYYY')";
223. //To\_date(madate, 'DD/MM/YYYY')

226. **if**(Where == **null**)
227. {
228. SendMsg("ERR#Recherche impossible sur ce critere");
229. }
231. ResultSet rs = **null**;
233. **try** {
234. rs = beanOracle.selection(Select, From, Where);
235. } **catch** (SQLException ex) {
236. SendMsg("ERR#Base de donnee inaccessible");
237. System.err.println("Erreur SQL exception input lorry" + ex.getStackTrace());
238. **return**;
239. }
241. **boolean** empty = **true**;
242. String message = "";
244. **try** {
245. **while**(rs.next())
246. {
247. empty = **false**;
248. message = message + rs.getString("ID\_MOUVEMENT") + "  ---  " + rs.getString("ID\_CONTAINER") + "  ---  " + rs.getString("ID\_TRANSPORTEUR\_ENTRANT") + "  ---  ";
249. message = message + rs.getString("DATE\_ARRIVEE") + "  ---  " + rs.getString("ID\_TRANSPORTEUR\_SORTANT") + "  ---  " + rs.getString("POIDS") + "  ---  ";
250. message = message + rs.getString("DATE\_DEPART") + "  ---  " + rs.getString("DESTINATION") + "  ---  " + rs.getString("ID\_SOCIETE")+"#";
251. }
252. } **catch** (SQLException ex) {
253. SendMsg("ERR#Base de donnee inaccessible");
254. System.err.println("Erreur SQL exception input lorry" + ex.getStackTrace());
255. **return**;
256. }
257. **if**(empty)
258. {
259. SendMsg("ERR#Aucun resultats pour la societe " + request[2]);
260. **return**;
261. }
263. SendMsg("ACK#"+message);
264. }

267. **private** **void** inputLorryWithoutReserv(String[] request)
268. {
269. ResultSet rs = **null**;
271. String[] idList =  request[2].split("@");
273. **try** {
274. rs = beanOracle.selection("X, Y", "PARC", "ETAT=0");
275. } **catch** (SQLException ex) {
276. SendMsg("ERR#Base de donnÃ©e inaccessible");
277. System.err.println("Erreur SQL exception input lorry" + ex.getStackTrace());
278. **return**;
279. }
281. String reponse = "ACK#";
282. ArrayList emplacement = **new** ArrayList();
284. **try**// on regarde si y'a assez de place et on recupere l'id de ces places.
285. {
286. **for**(**int** i = 0; i < idList.length ; i++)
287. {
288. **if**(rs.next())
289. {
290. reponse = reponse + idList[i] + "==>("+rs.getString("X")+";"+rs.getString("Y")+")@";
291. emplacement.add(rs.getString("X")+";"+rs.getString("Y"));
292. }
293. **else**
294. {
295. SendMsg("ERR#Erreur pas assez de places");
296. **return**;
297. }
298. }
299. }
300. **catch**(SQLException ex){
301. SendMsg("ERR#Base de donnÃ©e inaccessible");
302. System.err.println("Erreur SQL exception input lorry" + ex.getStackTrace());
303. **return**;
304. }
306. //On insert les containers ajoutÃ©s dans la BD et on leur met un numÃ©ro de rÃ©servation + on rÃ©serve leurs places
307. Random rand =  **new** Random();
308. **int** resID = rand.nextInt(999999);
309. **for**(**int** i = 0; i < idList.length; i++)
310. {
311. String[] coord = emplacement.get(i).toString().split(";");
312. HashMap<String, String> insertion = **new** HashMap();
313. HashMap<String, String> update = **new** HashMap();
315. insertion.put("ID\_CONTAINER", idList[i]);
316. insertion.put("RESERVATION", Integer.toString(resID));
318. update.put("ETAT", "1");
320. **try** {
321. beanOracle.ecriture("CONTAINERS", insertion);
322. beanOracle.miseAJour("PARC", update, "X="+coord[0]+" AND Y=" + coord[1]);
323. } **catch** (requeteException ex) {
324. System.err.println("Erreur d'insertion ");
325. }
326. }
328. SendMsg(reponse);
329. }
331. /\* Envoi d'un message au client \*/
332. **public** **void** SendMsg(String msg)
333. {
334. String chargeUtile = msg;
335. **int** taille = chargeUtile.length();
336. StringBuffer message = **new** StringBuffer(String.valueOf(taille) + "#" + chargeUtile);
338. **try**
339. {
340. dos.write(message.toString().getBytes());
341. dos.flush();
342. }
343. **catch**(IOException e)
344. {
345. System.err.println("RunnableTraitement : Erreur d'envoi de msg (IO) : " + e);
346. }
347. }
349. /\* RÃ©ception d'un message du client \*/
350. **public** String ReceiveMsg()
351. {
352. **byte** b;
353. StringBuffer taille = **new** StringBuffer();
354. StringBuffer message = **new** StringBuffer();
356. **try**
357. {
358. **while** ((b = dis.readByte()) != (**byte**)'#')
359. {
360. **if** (b != (**byte**)'#')
361. taille.append((**char**)b);
362. }
364. **for** (**int** i = 0; i < Integer.parseInt(taille.toString()); i++)
365. {
366. b = dis.readByte();
367. message.append((**char**)b);
368. }
369. }
370. **catch**(IOException e)
371. {
372. System.err.println("RunnableTraitement : Erreur de reception de msg (IO) : " + e);
373. }
375. **return** message.toString();
376. }
377. }
378. **Partie BOOMAP**
379. **public** **class** RunnableBOOMAP **implements** Runnable{
380. **private** Socket CSocket = **null**;
381. **private** DataInputStream dis = **null**;
382. **private** DataOutputStream dos = **null**;
383. **private** BeanBDAccess beanOracle;
385. **boolean** first = **true**;
387. **public** RunnableBOOMAP(Socket s)
388. {
389. CSocket = s;
391. **try**
392. {
393. dis = **new** DataInputStream(**new** BufferedInputStream(CSocket.getInputStream()));
394. dos = **new** DataOutputStream(**new** BufferedOutputStream(CSocket.getOutputStream()));
395. }
396. **catch**(IOException e)
397. {
398. System.err.println("RunnableTraitement : Host non trouvÃ© : " + e);
399. }
401. beanOracle = **new** BeanBDAccess();
402. **try** {
403. beanOracle.connexionOracle("localhost", 1521, "TRAFIC", "TRAFIC", "XE");
404. } **catch** (ClassNotFoundException ex) {
405. System.err.println("Class not found " + ex.getMessage());
406. } **catch** (SQLException ex) {
407. System.err.println("SQL Exception (oracle)" + ex.getMessage());
408. } **catch** (connexionException ex) {
409. System.err.println(ex.getNumException() + " -- " + ex.getMessage());
410. }
411. }
413. @Override
414. **public** **void** run()
415. {
416. String[] parts = (ReceiveMsg()).split("#");
418. **if**(parts[0].equals("LOGIN"))
419. {
420. **if**(!login(parts))
421. **return**;
422. }
423. **else**
424. {
425. SendMsg("ERR#RequÃ¨te invalide");
426. **return**;
427. }
429. **boolean** terminer = **false**;
431. **while**(!terminer)
432. {
433. parts = ReceiveMsg().split("#");
434. System.err.println(parts[0]);
435. **switch** (parts[0])
436. {
437. **case** "LOGOUT" :
438. terminer = **true**;
439. **break**;
441. **case** "GET\_XY" :
442. get\_xy(parts);
443. **break**;
445. **case** "SEND\_WEIGHT" :
446. send\_weight(parts);
447. **break**;
449. **case** "GET\_LIST" :
450. get\_list(parts);
451. **break**;
453. **case** "SIGNAL\_DEP" :
454. signal\_dep(parts);
455. **break**;
457. **default** :
458. terminer = **true**;
459. **break**;
460. }
461. }
463. **try** {
464. CSocket.close();
465. } **catch** (IOException ex) {
466. System.err.println("Erreur de close : " + ex.getStackTrace());
467. }
469. }
471. **private** **boolean** login(String[] part)
472. {
473. ResultSet rs = **null**;
475. **try**
476. {
477. rs = beanOracle.selection("PASSWORD", "UTILISATEURS", "LOGIN = '" + part[1]+"'");
478. }
479. **catch**(SQLException e){
480. System.err.println(e.getStackTrace());
481. }
483. String pwd = **null**;
485. **try** {
486. **if**(!rs.next())
487. {
488. SendMsg("ERR#Login invalide");
489. }
490. **else**
491. pwd = rs.getString("PASSWORD");
492. } **catch** (SQLException ex) {
493. System.err.println(ex.getStackTrace());
494. }
496. **if**(pwd.equals(part[2]))
497. {
498. SendMsg("ACK");
499. **return** **true**;
500. }
501. **else**
502. SendMsg("ERR#Mot de passe incorrecte");
504. **return** **false**;
505. }
507. **private** **void** get\_xy(String[] request)
508. {
509. String message = "ACK#";
510. ResultSet rs = **null**;
512. **try** {
513. rs = beanOracle.selection("X, Y", "PARC", "ETAT = 1");
514. } **catch** (SQLException ex) {
515. SendMsg("ERR#Probleme SQL");
516. System.err.println(ex.getStackTrace());
517. **return**;
518. }
519. System.err.println(request[3]);
520. String[] listContainer = request[3].split("\\@");
522. **for**(String s : listContainer)
523. {
524. System.err.println(s);
525. **try** {
526. **if**(rs.next())
527. {
528. **if**(!message.equals("ACK#"))
529. message = message + "@";
531. String[] infoContainer = s.split(";");

534. //insert societe
535. HashMap<String, String> insertSociete = **new** HashMap<>();
537. insertSociete.put("ID\_SOCIETE", request[2]);
538. **try** {
539. beanOracle.ecriture("SOCIETES", insertSociete);
540. } **catch** (requeteException ex) {
541. System.err.println("La societe existe deja");
542. }
544. //Insert transporteur
546. HashMap<String, String> insertTransporteur = **new** HashMap<>();
548. insertTransporteur.put("ID\_TRANSPORTEUR", request[1]);
549. insertTransporteur.put("ID\_SOCIETE", request[2]);
550. **try** {
551. beanOracle.ecriture("TRANSPORTEURS", insertTransporteur);
552. } **catch** (requeteException ex) {
553. System.err.println("Le transporteur existe deja");
554. }

557. //INSERTION DANS PARC
558. HashMap<String, String> updateParc = **new** HashMap();
559. updateParc.put("ETAT", "2");
560. updateParc.put("ID\_CONTAINER", infoContainer[0]);
561. Calendar cal = Calendar.getInstance();
562. SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/YYYY");
563. updateParc.put("DATE\_ARRIVEE", sdf.format(cal.getTime()));
564. updateParc.put("DESTINATION", infoContainer[1]);
566. System.err.println(infoContainer[0] + "  " + infoContainer[1] + "  " + sdf.format(cal.getTime()));
567. **try** {
568. beanOracle.miseAJour("PARC", updateParc, "X = "+rs.getString("X")+" AND Y = " + rs.getString("Y"));
569. } **catch** (requeteException ex) {
570. System.err.println("echec update parc : " + ex.getMessage());
571. }
573. //Insert mouvement
575. HashMap<String, String> insertMouvement = **new** HashMap<>();
576. Random rand = **new** Random();
577. **int** idMouvement = rand.nextInt(89892);
578. insertMouvement.put("ID\_MOUVEMENT", Integer.toString(idMouvement));
579. insertMouvement.put("ID\_CONTAINER", infoContainer[0]);
580. insertMouvement.put("ID\_TRANSPORTEUR\_ENTRANT", request[1]);
581. insertMouvement.put("DATE\_ARRIVEE", sdf.format(cal.getTime()));
582. insertMouvement.put("DESTINATION", infoContainer[1]);
584. **try** {
585. beanOracle.ecriture("MOUVEMENTS", insertMouvement);
586. } **catch** (requeteException ex) {
587. System.err.println("Le mouvement existe deja");
588. }
590. //UPDATE CONTAINERS
592. HashMap<String, String> updateContainer = **new** HashMap<>();
594. updateContainer.put("ID\_SOCIETE", request[2]);
596. **try** {
597. beanOracle.miseAJour("CONTAINERS", updateContainer, "ID\_CONTAINER = '"+infoContainer[0]+"'");
598. } **catch** (requeteException ex) {
599. System.err.println("Erreur MAJ container");
600. }
602. message = message + rs.getString("X")+";"+rs.getString("Y");
604. }
605. **else**
606. {
607. System.err.println("pas assez de resultat");
608. SendMsg("ERR#Pas assez de resultats");
609. }
610. } **catch** (SQLException ex) {
611. SendMsg("ERR#Probleme SQL");
612. System.err.println(ex.getStackTrace());
613. **return**;
614. }
615. }
617. SendMsg(message);
618. }
620. **private** **void** get\_list(String[] request)
621. {
622. ResultSet rs = **null**;
623. String transport = **null**;
625. **if**(request[2].equals("1"))
626. transport = "TRAIN";
627. **else**
628. transport = "BATEAU";
630. **try** {
631. rs = beanOracle.selection("ID\_CONTAINER, X, Y", "PARC", "TRANSPORT = '"+transport+"' AND UPPER(DESTINATION) = UPPER('"+request[3]+"')");
632. } **catch** (SQLException ex) {
633. SendMsg("ERR#Acces a la BD impossible");
634. System.err.println("erreur: " + ex.getStackTrace().toString());
635. **return**;
636. }
638. String message = "";
640. **try** {
641. **while**(rs.next())
642. {
643. **if**(!message.isEmpty())
644. message = message + "#";
646. message = message + rs.getString("ID\_CONTAINER") + "@" + rs.getString("X") + ";" + rs.getString("Y");
647. }
648. } **catch** (SQLException ex) {
649. SendMsg("ERR#Acces a la BD impossible");
650. System.err.println("test" + ex.getStackTrace().toString());
651. **return**;
652. }
654. **if**(message.isEmpty())
655. {
656. SendMsg("ERR#Aucun container pour la destination");
657. **return**;
658. }
660. SendMsg("ACK#"+message);
661. }
663. **private** **void** signal\_dep(String[] requete)
664. {
665. //Insertion transporteur :
667. //Insert transporteur
669. HashMap<String, String> insertTransporteur = **new** HashMap<>();
671. insertTransporteur.put("ID\_TRANSPORTEUR", requete[1]);
672. **try** {
673. beanOracle.ecriture("TRANSPORTEURS", insertTransporteur);
674. } **catch** (requeteException ex) {
675. System.err.println("Le transporteur existe deja");
676. }
678. **for**(String coord : requete)
679. {
680. **if**(coord.equals(requete[1]) || coord.equals(requete[0]))
681. **continue**;
683. //MAJ etat parc
684. String[] splitCoord = coord.split(";");
685. HashMap<String, String> updateParc = **new** HashMap();
686. updateParc.put("ETAT", "0");
688. **try** {
689. beanOracle.miseAJour("PARC", updateParc, "X = "+splitCoord[0]+" AND Y = " + splitCoord[1]);
690. } **catch** (requeteException ex) {
691. System.err.println("echec update parc : " + ex.getMessage());
692. }

695. //MAJ mouvement
697. //Insert mouvement
699. HashMap<String, String> insertMouvement = **new** HashMap<>();
700. Calendar cal = Calendar.getInstance();
701. SimpleDateFormat sdf = **new** SimpleDateFormat("dd/MM/YYYY");
702. insertMouvement.put("ID\_TRANSPORTEUR\_SORTANT", requete[1]);
703. insertMouvement.put("DATE\_DEPART", sdf.format(cal.getTime()));
705. String where = "ID\_CONTAINER = (SELECT ID\_CONTAINER FROM PARC WHERE X = "+splitCoord[0]+" AND Y = " + splitCoord[1]+")";
706. **try** {
707. beanOracle.miseAJour("MOUVEMENTS", insertMouvement, where);
708. } **catch** (requeteException ex) {
709. System.err.println("Le mouvement existe deja");
710. }
711. }


715. SendMsg("ACK#");
716. }
718. **private** **void** send\_weight(String[] requete)
719. {
721. **for**(String s : requete)
722. {
724. System.err.println(s);
725. **if**(s.equals("SEND\_WEIGHT"))
726. **continue**;
728. String[] contTraite = s.split(";");
729. String transport;
731. **if**(contTraite[3].equals("1"))
732. transport = "TRAIN";
733. **else**
734. transport = "BATEAU";
736. HashMap<String,String> updateMouvement = **new** HashMap<>();
737. System.err.println(contTraite[2]);
738. updateMouvement.put("POIDS", contTraite[2]);
740. **try** {
741. beanOracle.miseAJour("MOUVEMENTS", updateMouvement, "ID\_CONTAINER = '"+contTraite[0]+"'");
742. } **catch** (requeteException ex) {
743. System.err.println("Aucun mouvement trouve");
744. }
746. HashMap<String, String> updateParc = **new** HashMap<>();
747. updateParc.put("TRANSPORT", transport);
749. **try** {
750. beanOracle.miseAJour("PARC", updateParc, "ID\_CONTAINER = '"+contTraite[0]+"'");
751. } **catch** (requeteException ex) {
752. System.err.println("Aucun parc trouve");
753. }
755. }
756. SendMsg("ACK#");
757. }

760. /\* Envoi d'un message au client \*/
761. **public** **void** SendMsg(String msg)
762. {
763. String chargeUtile = msg;
764. **int** taille = chargeUtile.length();
765. StringBuffer message = **new** StringBuffer(String.valueOf(taille) + "#" + chargeUtile);
767. **try**
768. {
769. dos.write(message.toString().getBytes());
770. dos.flush();
771. }
772. **catch**(IOException e)
773. {
774. System.err.println("RunnableTraitement : Erreur d'envoi de msg (IO) : " + e);
775. }
776. }
778. /\* RÃ©ception d'un message du client \*/
779. **public** String ReceiveMsg()
780. {
781. **byte** b;
782. StringBuffer taille = **new** StringBuffer();
783. StringBuffer message = **new** StringBuffer();
785. **try**
786. {
787. **while** ((b = dis.readByte()) != (**byte**)'#')
788. {
789. **if** (b != (**byte**)'#')
790. taille.append((**char**)b);
791. }
793. **for** (**int** i = 0; i < Integer.parseInt(taille.toString()); i++)
794. {
795. b = dis.readByte();
796. message.append((**char**)b);
797. }
798. }
799. **catch**(IOException e)
800. {
801. System.err.println("RunnableTraitement : Erreur de reception de msg (IO) : " + e);
802. }
804. **return** message.toString();
805. }
806. }

Code administrateur serveur container

1. **public** **class** GUIAdmin **extends** javax.swing.JFrame {

4. **private** Socket cliSocket;
6. **private** DataInputStream dis;
7. **private** DataOutputStream dos;
9. /\*\*
10. \* Creates new form GUIAdmin
11. \*/
12. **public** GUIAdmin() {
13. initComponents();
14. }


18. **private** **void** connexionButtonActionPerformed(java.awt.event.ActionEvent evt) {
20. **if**(connexionButton.getText().equals("Connexion"))
21. {
22. **try**
23. {
24. cliSocket = **new** Socket(ipTextField.getText(), Integer.parseInt(portTextField.getText()));
25. dis = **new** DataInputStream(**new** BufferedInputStream(cliSocket.getInputStream()));
26. dos = **new** DataOutputStream(**new** BufferedOutputStream(cliSocket.getOutputStream()));
27. }
28. **catch** (IOException ex)
29. {
30. System.err.println("Erreur gui admin connexion : " + ex);
31. }
33. String message = protocoleCSA.LOGIN + "#"+LoginTextField.getText()+"#"+passwordField.getText();
35. SendMsg(message);
37. System.out.println(ReceiveMsg());
39. connexionButton.setText("DÃ©connexion");
40. }
41. **else**
42. {
43. connexionButton.setText("Connexion");
44. deconnexion();
45. }
46. }
48. **private** **void** listerButtonActionPerformed(java.awt.event.ActionEvent evt) {
49. **if**(cliSocket == **null**)
50. **return**;
52. SendMsg(protocoleCSA.LISTCLIENT + "#");
54. String str = ReceiveMsg();
56. String split[] = str.split("#");
58. listTextArea.setText("");
59. **for**(**int** i = 1; i < split.length; i++)
60. listTextArea.append(split[i] + "\n");
62. }
64. **private** **void** pauseButton1ActionPerformed(java.awt.event.ActionEvent evt) {
65. SendMsg(protocoleCSA.PAUSE + "#");
67. String str = ReceiveMsg();
68. }
70. **private** **void** continuerButtonActionPerformed(java.awt.event.ActionEvent evt) {
71. SendMsg(protocoleCSA.CONTINUER + "#");
73. String str = ReceiveMsg();
74. }
76. **private** **void** stopButtonActionPerformed(java.awt.event.ActionEvent evt) {
78. **int** sec = 0;
80. **try**
81. {
82. sec = Integer.parseInt(secondesTextField.getText());
83. }
84. **catch**(NumberFormatException nfe)
85. {
86. sec = 0;
87. }
89. **if**(sec > 1000 || sec < 0)
90. sec = 0;
92. SendMsg(protocoleCSA.STOP + "#" + sec);
94. String str = ReceiveMsg();
95. }
97. **public** **void** deconnexion()
98. {
99. SendMsg(protocoleCSA.LOGOUTCSA + "#");
100. ReceiveMsg();
101. **try**
102. {
103. dos.close();
104. dis.close();
105. cliSocket.close();
106. cliSocket = **null**;
107. System.out.println("ClientServeurBateau : Client dÃ©connectÃ©");
108. }
109. **catch**(IOException e)
110. {
111. System.err.println("ClientServeurBateau : Erreur de dÃ©connexion : " + e);
112. }
113. }

116. **public** **void** SendMsg(String chargeUtile)
117. {
118. **int** taille = chargeUtile.length();
119. String message = String.valueOf(taille) + "#" + chargeUtile;
121. **try**
122. {
123. dos.write(message.getBytes());
124. dos.flush();
125. }
126. **catch**(IOException e)
127. {
128. System.err.println("ClientServeurBateu : Erreur d'envoi de msg (IO) : " + e);
129. }
130. }
132. **public** String ReceiveMsg()
133. {
134. **byte** b;
135. StringBuffer taille = **new** StringBuffer();
136. StringBuffer message = **new** StringBuffer();
138. **try**
139. {
140. **while** ((b = dis.readByte()) != (**byte**)'#')
141. {
142. **if** (b != (**byte**)'#')
143. taille.append((**char**)b);
144. }
146. **for** (**int** i = 0; i < Integer.parseInt(taille.toString()); i++)
147. {
148. b = dis.readByte();
149. message.append((**char**)b);
150. }
151. }
152. **catch**(IOException e)
153. {
154. System.err.println("ClientServeurBateau : Erreur de reception de msg (IO) : " + e);
155. }
157. reponseLabel.setText("reponse serveur : " + message.toString()); // Ã  retirer
158. **return** message.toString();
159. }

162. /\*\*
163. \* @param args the command line arguments
164. \*/
165. **public** **static** **void** main(String args[]) {
166. /\* Set the Nimbus look and feel \*/
167. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
168. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
169. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
170. \*/
171. **try** {
172. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
173. **if** ("Nimbus".equals(info.getName())) {
174. javax.swing.UIManager.setLookAndFeel(info.getClassName());
175. **break**;
176. }
177. }
178. } **catch** (ClassNotFoundException ex) {
179. java.util.logging.Logger.getLogger(GUIAdmin.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
180. } **catch** (InstantiationException ex) {
181. java.util.logging.Logger.getLogger(GUIAdmin.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
182. } **catch** (IllegalAccessException ex) {
183. java.util.logging.Logger.getLogger(GUIAdmin.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
184. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) {
185. java.util.logging.Logger.getLogger(GUIAdmin.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex);
186. }
187. //</editor-fold>
189. /\* Create and display the form \*/
190. java.awt.EventQueue.invokeLater(**new** Runnable() {
191. **public** **void** run() {
192. **new** GUIAdmin().setVisible(**true**);
193. }
194. });
195. }
197. // Variables declaration - do not modify
198. **private** javax.swing.JTextField LoginTextField;
199. **private** javax.swing.JButton connexionButton;
200. **private** javax.swing.JButton continuerButton;
201. **private** javax.swing.JLabel ipLabel;
202. **private** javax.swing.JTextField ipTextField;
203. **private** javax.swing.JScrollPane jScrollPane1;
204. **private** javax.swing.JTextArea listTextArea;
205. **private** javax.swing.JButton listerButton;
206. **private** javax.swing.JLabel loginLabel;
207. **private** javax.swing.JPasswordField passwordField;
208. **private** javax.swing.JLabel passwordTextField;
209. **private** javax.swing.JButton pauseButton1;
210. **private** javax.swing.JLabel portLabel;
211. **private** javax.swing.JTextField portTextField;
212. **private** javax.swing.JLabel reponseLabel;
213. **private** javax.swing.JLabel secondesLabel;
214. **private** javax.swing.JTextField secondesTextField;
215. **private** javax.swing.JButton stopButton;
216. // End of variables declaration
217. }

Schéma relationnel BD\_COMPTA

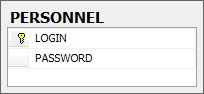
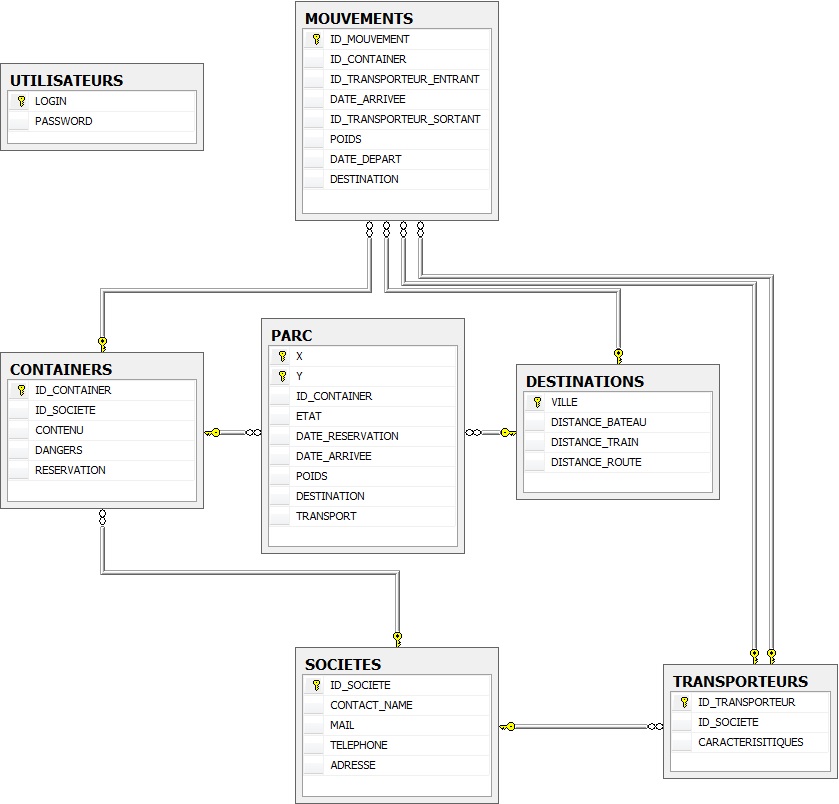
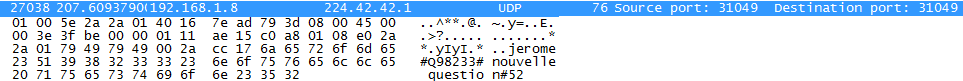


Schéma relationnel BD\_TRAFIC



Trame chat UDP

Question :



Infos :