2017

Réseaux et Technologie Internet

***Laurent Reynders & Océane Stasse (2301)***

# Définition et implémentation des commandes du protocole LUGAP

|  |  |  |
| --- | --- | --- |
| **commande** | **Sémantique** | **Réponse attendue** |
| Login | Requête permettant de vérifier si le login et mot de passe sont correct.  *Paramètres attendus :* login, password hashé par digest salé, time en long, random en int | OK *ou* KO + message expliquant pourquoi.  Le serveur recalcule un digest salé à l’aide du time et random de la requête et le mdp de la base de données, et le compare avec le digest salé reçu dans la requête. |
| Logout | Requête signalant la fin de la transaction entre l’application et le serveur.  *Paramètres attendus :* / | OK ou KO + message expliquant pourquoi. |
| GetFlights | Requête permettant de récupérer les vols de la journée, triés par temps de décollage croissant.  *Paramètres attendus :* / | OK + liste des vols du jour *ou* KO + message expliquant pourquoi. |
| GetLuggages | Requête récupérant la liste des bagages pour un certain vol.  *Paramètres attendus :* le vol en question | OK + liste des bagages pour le vol reçu dans la requête ou KO + message expliquant pourquoi. |
| UpdateFieldReceived | Requête de mise à jour du champ `received` dans la base de données pour un certain luggage.  *Paramètre attendus :* luggage à modifier, nouvelle valeur (O/N). | OK si au moins un row a été modifié ou KO + message expliquant pourquoi. |
| UpdateFieldLoaded | Requête de mise à jour du champ `loaded` dans la base de données pour un certain luggage.  *Paramètre attendus :* luggage à modifier, nouvelle valeur (O/N). | OK si au moins un row a été modifié ou KO + message expliquant pourquoi. |
| UpdateFieldCheckedByCustoms | Requête de mise à jour du champ `checkedbycustom` dans la base de données pour un certain luggage.  *Paramètre attendus :* luggage à modifier, nouvelle valeur (O/N). | OK si au moins un row a été modifié ou KO + message expliquant pourquoi. |
| UpdateFieldComments | Requête de mise à jour du champ `comments` dans la base de données pour un certain luggage.  *Paramètre attendus :* luggage à modifier, nouvelle remarque. | OK si au moins un row a été modifié ou KO + message expliquant pourquoi. |

## Code

### Requetes

#### Requete.java (du serveur générique)

1. package server;
2. import communicator.Communicator;
3. import database.utilities.DatabaseAccess;
4. import java.io.Serializable;
5. public interface Requete extends Serializable {
6. public Runnable createRunnable(Runnable parent, Communicator communicator, ConsoleServeur consoleServeur, DatabaseAccess databaseAccess);
7. }

#### RequeteLUGAP.java

1. package lugap.requete;
2. import communicator.Communicator;
3. import database.utilities.DatabaseAccess;
4. import java.io.Serializable;
5. import server.ConsoleServeur;
6. import server.Reponse;
7. import server.Requete;
8. public abstract class RequeteLUGAP implements Requete, Serializable {
9. protected Runnable parent;
10. protected Communicator communicator;
11. protected ConsoleServeur consoleServeur;
12. protected DatabaseAccess databaseAccess;
13. private final String requestTypeName;
14. protected final String sqlStatement;
15. protected Reponse reponse;
16. protected RequeteLUGAP(String requestTypeName, String sqlStatement) {
17. this.requestTypeName = requestTypeName;
18. this.sqlStatement = sqlStatement;
19. }@
20. Override public Runnable createRunnable(Runnable parent, Communicator communicator, ConsoleServeur consoleServeur, DatabaseAccess databaseAccess) {
21. this.parent = parent;
22. this.communicator = communicator;
23. this.consoleServeur = consoleServeur;
24. this.databaseAccess = databaseAccess;
25. return () - > {
26. doAction();
27. };
28. }
29. protected abstract void doAction();
30. protected void traceEvent(String message) {
31. consoleServeur.TraceEvenements(communicator.getSocket().getRemoteSocketAddress().toString() + "#" + message + "#" + requestTypeName + "@LUGAP");
32. }
33. public boolean requeteSucceeded() {
34. if (this.reponse == null) return false;
35. else return this.reponse.isSuccessful();
36. }
37. }

#### RequeteLUGAP\_getFlights.java

1. package lugap.requete;
2. import communicator.CommunicatorException;
3. import entities.Flight;
4. import java.io.Serializable;
5. import java.sql.ResultSet;
6. import java.sql.SQLException;
7. import java.util.ArrayList;
8. import lugap.reponse.ReponseLUGAP\_getFlights;
9. public class RequeteLUGAP\_getFlights extends RequeteLUGAP implements Serializable {
10. public RequeteLUGAP\_getFlights() {
11. super("GetFlight", "SELECT \* " + "FROM flight " + "WHERE departure=CURDATE() " + "ORDER BY takeOffTime ASC");
12. }@
13. Override protected void doAction() {
14. ArrayList < Flight > flights = new ArrayList < > ();
15. try {
16. try {
17. ResultSet resultSet = this.databaseAccess.executeQuery(this.sqlStatement);
18. System.out.println("\nToday's flights are : ");
19. while (resultSet.next()) {
20. Flight flight = new Flight(resultSet.getInt("fk\_idairplane"), resultSet.getString("fk\_idairline"), resultSet.getDate("departure").toLocalDate(), resultSet.getString("destination"), resultSet.getTime("takeOffTime").toLocalTime(), resultSet.getTime("scheduledLanding").toLocalTime());
21. flights.add(flight);
22. System.out.println("    Flight n°" + flights.size() + " : " + flight.toString());
23. }
24. System.out.println("-------");
25. this.reponse = ReponseLUGAP\_getFlights.OK(flights);
26. traceEvent("GetFlight OK (flights found : " + flights.size() + ")");
27. } catch (SQLException ex) {
28. this.reponse = ReponseLUGAP\_getFlights.KO("Erreur Serveur (SQL)");
29. traceEvent("Erreur SQL/BDD : " + ex.getMessage());
30. }
31. this.communicator.SendSerializable(this.reponse);
32. } catch (CommunicatorException ex) {
33. traceEvent(ex.getMessage());
34. }
35. }
36. }

#### RequeteLUGAP\_getLuggages.java

1. package lugap.requete;
2. import communicator.CommunicatorException;
3. import entities.Flight;
4. import entities.Luggage;
5. import java.io.Serializable;
6. import java.sql.PreparedStatement;
7. import java.sql.ResultSet;
8. import java.sql.SQLException;
9. import java.util.ArrayList;
10. import java.util.HashMap;
11. import java.util.Map;
12. import lugap.reponse.ReponseLUGAP\_getLuggages;
13. public class RequeteLUGAP\_getLuggages extends RequeteLUGAP implements Serializable {
14. private final Flight flight;
15. public RequeteLUGAP\_getLuggages(Flight flight) {
16. super("getLuggages", "SELECT passenger.firstname, passenger.lastname, luggage.fk\_idticket, luggage.idluggage, " + "luggage.weight, luggage.isluggage, luggage.received, luggage.loaded, " + "luggage.checkedbycustom, luggage.comments " + "FROM luggage INNER JOIN ticket " + "ON luggage.fk\_idairplane = ticket.fk\_idairplane " + "AND luggage.fk\_idairline = ticket.fk\_idairline " + "AND luggage.fk\_departure = ticket.fk\_departure " + "AND luggage.fk\_destination = ticket.fk\_destination " + "AND luggage.fk\_idticket = ticket.idticket " + "INNER JOIN passenger " + "ON ticket.fk\_idpassenger = passenger.idpassenger " + "WHERE luggage.fk\_idairplane = ? " + "AND luggage.fk\_idairline = ? " + "AND luggage.fk\_departure = ? " + "AND luggage.fk\_destination = ? " + "FOR UPDATE");
17. this.flight = flight;
18. }@
19. Override protected void doAction() {
20. ArrayList < Luggage > luggages = new ArrayList < > ();
21. Map < Integer, Object > statementMap = new HashMap < > ();
22. statementMap.put(1, this.flight.getIdAirplane());
23. statementMap.put(2, this.flight.getIdAirline());
24. statementMap.put(3, this.flight.getDepartureDate());
25. statementMap.put(4, this.flight.getDestination());
26. try {
27. try {
28. PreparedStatement preparedStatement = this.databaseAccess.getPreparedStatement(sqlStatement);
29. preparedStatement.setQueryTimeout(5);
30. ResultSet resultSet = this.databaseAccess.executeQuery(preparedStatement, statementMap);
31. System.out.println("\nFlight [" + this.flight.toString() + "]'s luggages : ");
32. while (resultSet.next()) {
33. Luggage luggage = new Luggage(this.flight, resultSet.getString("firstname"), resultSet.getString("lastname"), resultSet.getInt("fk\_idticket"), resultSet.getInt("idluggage"), resultSet.getFloat("weight"), resultSet.getBoolean("isluggage"), resultSet.getString("received").charAt(0), resultSet.getString("loaded").charAt(0), resultSet.getString("checkedbycustom").charAt(0), resultSet.getString("comments"));
34. luggages.add(luggage);
35. System.out.println("    luggage n°" + luggages.size() + " : " + luggage.toString());
36. }
37. System.out.println("-------");
38. this.reponse = ReponseLUGAP\_getLuggages.OK(luggages);
39. traceEvent("GetLuggages OK (luggages found : " + luggages.size() + ")");
40. } catch (SQLException ex) {
41. this.reponse = ReponseLUGAP\_getLuggages.KO("Erreur Serveur (SQL) [" + ex.getMessage() + "]");
42. traceEvent("Erreur SQL/BDD : " + ex.getMessage());
43. }
44. this.communicator.SendSerializable(this.reponse);
45. } catch (CommunicatorException ex) {
46. traceEvent(ex.getMessage());
47. }
48. }
49. public Flight getFlight() {
50. return flight;
51. }
52. }

#### RequeteLUGAP\_login.java

1. package lugap.requete;
2. import communicator.CommunicatorException;
3. import encryption.Encrypt;
4. import encryption.EncryptionException;
5. import java.io.Serializable;
6. import java.sql.ResultSet;
7. import java.sql.SQLException;
8. import java.util.Arrays;
9. import java.util.HashMap;
10. import java.util.Map;
11. import lugap.reponse.ReponseLUGAP\_login;
12. public class RequeteLUGAP\_login extends RequeteLUGAP implements Serializable {
13. private final String login;
14. private final byte[] digestedPassword;
15. private final long time;
16. private final double rand;
17. public RequeteLUGAP\_login(String login, byte[] digestedPassword, long time, double rand) {
18. super("login", "SELECT login, password " + "FROM agent INNER JOIN job " + "ON fk\_idjob = idjob " + "WHERE job.name='Bagagiste'" + "AND agent.login = ?");
19. this.login = login;
20. this.digestedPassword = digestedPassword;
21. this.time = time;
22. this.rand = rand;
23. }@
24. Override protected void doAction() {
25. Map < Integer, Object > statementMap = new HashMap < > ();
26. statementMap.put(1, this.login);
27. try {
28. try {
29. ResultSet resultSet = this.databaseAccess.executeQuery(this.sqlStatement, statementMap);
30. if (!resultSet.next() || !Arrays.equals(this.digestedPassword, Encrypt.saltDigest(resultSet.getString("password"), this.time, this.rand))) {
31. this.reponse = ReponseLUGAP\_login.KO("Wrong login/password");
32. traceEvent("Wrong login/password (login:" + this.login + ")");
33. } else {
34. this.reponse = ReponseLUGAP\_login.OK();
35. traceEvent("Login OK (login: " + this.login + ")");
36. }
37. } catch (SQLException ex) {
38. this.reponse = ReponseLUGAP\_login.KO("Erreur Serveur (SQL)");
39. traceEvent("Erreur SQL/BDD : " + ex.getMessage());
40. } catch (EncryptionException ex) {
41. this.reponse = ReponseLUGAP\_login.KO("Erreur Serveur (Encrypt)");
42. traceEvent("Erreur d'encrypt : " + ex.getMessage());
43. }
44. this.communicator.SendSerializable(this.reponse);
45. } catch (CommunicatorException ex) {
46. traceEvent("Login : " + ex.getMessage());
47. }
48. }
49. }

#### RequeteLUGAP\_logout.java

1. package lugap.requete;
2. import communicator.CommunicatorException;
3. import java.io.Serializable;
4. import lugap.reponse.ReponseLUGAP\_logout;
5. public class RequeteLUGAP\_logout extends RequeteLUGAP implements Serializable {
6. public RequeteLUGAP\_logout() {
7. super("logout", "");
8. }@
9. Override protected void doAction() {
10. try {
11. this.reponse = ReponseLUGAP\_logout.OK();
12. traceEvent("Logout OK");
13. this.communicator.SendSerializable(this.reponse);
14. } catch (CommunicatorException ex) {
15. traceEvent("" + ex.getMessage());
16. }
17. }
18. }

#### RequeteLUGAP\_updateFieldReceived.java

1. package lugap.requete;
2. import communicator.CommunicatorException;
3. import entities.Luggage;
4. import java.io.Serializable;
5. import java.sql.SQLException;
6. import java.util.HashMap;
7. import java.util.Map;
8. import lugap.reponse.ReponseLUGAP\_updateField;
9. public class RequeteLUGAP\_updateFieldReceived extends RequeteLUGAP implements Serializable {
10. private final Luggage luggage;
11. private final char newValue;
12. public RequeteLUGAP\_updateFieldReceived(Luggage luggage, char newValue) {
13. super("updateFieldReceived", "UPDATE luggage " + "SET received=? " + "WHERE fk\_idairplane = ? " + "AND fk\_idairline = ? " + "AND fk\_departure = ? " + "AND fk\_destination = ? " + "AND fk\_idticket = ? " + "AND idluggage = ?");
14. this.luggage = luggage;
15. this.newValue = newValue;
16. }@
17. Override protected void doAction() {
18. Map < Integer, Object > statementMap = new HashMap < > ();
19. statementMap.put(1, String.valueOf(this.newValue));
20. statementMap.put(2, this.luggage.getFlight().getIdAirplane());
21. statementMap.put(3, this.luggage.getFlight().getIdAirline());
22. statementMap.put(4, this.luggage.getFlight().getDepartureDate());
23. statementMap.put(5, this.luggage.getFlight().getDestination());
24. statementMap.put(6, this.luggage.getIdTicket());
25. statementMap.put(7, this.luggage.getIdLuggage());
26. try {
27. try {
28. int result = this.databaseAccess.executeUpdate(this.sqlStatement, statementMap);
29. if (result == 0) {
30. this.reponse = ReponseLUGAP\_updateField.KO("No row updated in database");
31. traceEvent("No row updated in database (luggage:" + this.luggage.toString() + ")");
32. } else {
33. this.reponse = ReponseLUGAP\_updateField.OK();
34. traceEvent("Field 'received' updated to '" + this.newValue + "' (luggage: " + this.luggage.toString() + ")");
35. }
36. } catch (SQLException ex) {
37. this.reponse = ReponseLUGAP\_updateField.KO("Erreur Serveur (SQL)");
38. traceEvent("Erreur SQL/BDD : " + ex.getMessage());
39. } catch (InterruptedException ex) {
40. this.reponse = ReponseLUGAP\_updateField.KO("Erreur Serveur (Interrupted)");
41. traceEvent("Erreur (Interrupted) : " + ex.getMessage());
42. }
43. this.communicator.SendSerializable(this.reponse);
44. } catch (CommunicatorException ex) {
45. traceEvent("UpdateFieldReceived : " + ex.getMessage());
46. }
47. }
48. }

#### RequeteLUGAP\_updateFieldLoaded.java

1. package lugap.requete;
2. import communicator.CommunicatorException;
3. import entities.Luggage;
4. import java.io.Serializable;
5. import java.sql.SQLException;
6. import java.util.HashMap;
7. import java.util.Map;
8. import lugap.reponse.ReponseLUGAP\_updateField;
9. public class RequeteLUGAP\_updateFieldLoaded extends RequeteLUGAP implements Serializable {
10. private final Luggage luggage;
11. private final char newValue;
12. public RequeteLUGAP\_updateFieldLoaded(Luggage luggage, char newValue) {
13. super("updateFieldReceived", "UPDATE luggage " + "SET loaded=? " + "WHERE fk\_idairplane = ? " + "AND fk\_idairline = ? " + "AND fk\_departure = ? " + "AND fk\_destination = ? " + "AND fk\_idticket = ? " + "AND idluggage = ?");
14. this.luggage = luggage;
15. this.newValue = newValue;
16. }@
17. Override protected void doAction() {
18. Map < Integer, Object > statementMap = new HashMap < > ();
19. statementMap.put(1, String.valueOf(this.newValue));
20. statementMap.put(2, this.luggage.getFlight().getIdAirplane());
21. statementMap.put(3, this.luggage.getFlight().getIdAirline());
22. statementMap.put(4, this.luggage.getFlight().getDepartureDate());
23. statementMap.put(5, this.luggage.getFlight().getDestination());
24. statementMap.put(6, this.luggage.getIdTicket());
25. statementMap.put(7, this.luggage.getIdLuggage());
26. try {
27. try {
28. int result = this.databaseAccess.executeUpdate(this.sqlStatement, statementMap);
29. if (result == 0) {
30. this.reponse = ReponseLUGAP\_updateField.KO("No row updated in database");
31. traceEvent("No row updated in database (luggage:" + this.luggage.toString() + ")");
32. } else {
33. this.reponse = ReponseLUGAP\_updateField.OK();
34. traceEvent("Field 'loaded' updated to '" + this.newValue + "' (luggage: " + this.luggage.toString() + ")");
35. }
36. } catch (SQLException ex) {
37. this.reponse = ReponseLUGAP\_updateField.KO("Erreur Serveur (SQL)");
38. traceEvent("Erreur SQL/BDD : " + ex.getMessage());
39. } catch (InterruptedException ex) {
40. this.reponse = ReponseLUGAP\_updateField.KO("Erreur Serveur (Interrupted)");
41. traceEvent("Erreur (Interrupted) : " + ex.getMessage());
42. }
43. this.communicator.SendSerializable(this.reponse);
44. } catch (CommunicatorException ex) {
45. traceEvent("UpdateFieldLoaded : " + ex.getMessage());
46. }
47. }
48. }

#### RequeteLUGAP\_updateFieldCheckedByCustoms.java

1. package lugap.requete;
2. import communicator.CommunicatorException;
3. import entities.Luggage;
4. import java.io.Serializable;
5. import java.sql.SQLException;
6. import java.util.HashMap;
7. import java.util.Map;
8. import lugap.reponse.ReponseLUGAP\_updateField;
9. public class RequeteLUGAP\_updateFieldCheckedByCustoms extends RequeteLUGAP implements Serializable {
10. private final Luggage luggage;
11. private final char newValue;
12. public RequeteLUGAP\_updateFieldCheckedByCustoms(Luggage luggage, char newValue) {
13. super("updateFieldReceived", "UPDATE luggage " + "SET checkedbycustom=? " + "WHERE fk\_idairplane = ? " + "AND fk\_idairline = ? " + "AND fk\_departure = ? " + "AND fk\_destination = ? " + "AND fk\_idticket = ? " + "AND idluggage = ?");
14. this.luggage = luggage;
15. this.newValue = newValue;
16. }@
17. Override protected void doAction() {
18. Map < Integer, Object > statementMap = new HashMap < > ();
19. statementMap.put(1, String.valueOf(this.newValue));
20. statementMap.put(2, this.luggage.getFlight().getIdAirplane());
21. statementMap.put(3, this.luggage.getFlight().getIdAirline());
22. statementMap.put(4, this.luggage.getFlight().getDepartureDate());
23. statementMap.put(5, this.luggage.getFlight().getDestination());
24. statementMap.put(6, this.luggage.getIdTicket());
25. statementMap.put(7, this.luggage.getIdLuggage());
26. try {
27. try {
28. int result = this.databaseAccess.executeUpdate(this.sqlStatement, statementMap);
29. if (result == 0) {
30. this.reponse = ReponseLUGAP\_updateField.KO("No row updated in database");
31. traceEvent("No row updated in database (luggage:" + this.luggage.toString() + ")");
32. } else {
33. this.reponse = ReponseLUGAP\_updateField.OK();
34. traceEvent("Field 'checkedbycustom' updated to '" + this.newValue + "' (luggage: " + this.luggage.toString() + ")");
35. }
36. } catch (SQLException ex) {
37. this.reponse = ReponseLUGAP\_updateField.KO("Erreur Serveur (SQL)");
38. traceEvent("Erreur SQL/BDD : " + ex.getMessage());
39. } catch (InterruptedException ex) {
40. this.reponse = ReponseLUGAP\_updateField.KO("Erreur Serveur (Interrupted)");
41. traceEvent("Erreur (Interrupted) : " + ex.getMessage());
42. }
43. this.communicator.SendSerializable(this.reponse);
44. } catch (CommunicatorException ex) {
45. traceEvent("UpdateFieldCheckedByCustoms : " + ex.getMessage());
46. }
47. }
48. }

#### RequeteLUGAP\_updateFieldComments.java

1. package lugap.requete;
2. import communicator.CommunicatorException;
3. import entities.Luggage;
4. import java.io.Serializable;
5. import java.sql.SQLException;
6. import java.util.HashMap;
7. import java.util.Map;
8. import lugap.reponse.ReponseLUGAP\_updateField;
9. public class RequeteLUGAP\_updateFieldComments extends RequeteLUGAP implements Serializable {
10. private final Luggage luggage;
11. private final String newValue;
12. public RequeteLUGAP\_updateFieldComments(Luggage luggage, String newValue) {
13. super("updateFieldReceived", "UPDATE luggage " + "SET comments=? " + "WHERE fk\_idairplane = ? " + "AND fk\_idairline = ? " + "AND fk\_departure = ? " + "AND fk\_destination = ? " + "AND fk\_idticket = ? " + "AND idluggage = ?");
14. this.luggage = luggage;
15. this.newValue = newValue;
16. }@
17. Override protected void doAction() {
18. Map < Integer, Object > statementMap = new HashMap < > ();
19. statementMap.put(1, this.newValue);
20. statementMap.put(2, this.luggage.getFlight().getIdAirplane());
21. statementMap.put(3, this.luggage.getFlight().getIdAirline());
22. statementMap.put(4, this.luggage.getFlight().getDepartureDate());
23. statementMap.put(5, this.luggage.getFlight().getDestination());
24. statementMap.put(6, this.luggage.getIdTicket());
25. statementMap.put(7, this.luggage.getIdLuggage());
26. try {
27. try {
28. int result = this.databaseAccess.executeUpdate(this.sqlStatement, statementMap);
29. if (result == 0) {
30. this.reponse = ReponseLUGAP\_updateField.KO("No row updated in database");
31. traceEvent("No row updated in database (luggage:" + this.luggage.toString() + ")");
32. } else {
33. this.reponse = ReponseLUGAP\_updateField.OK();
34. traceEvent("Field 'comments' updated to '" + this.newValue + "' (luggage: " + this.luggage.toString() + ")");
35. }
36. } catch (SQLException ex) {
37. this.reponse = ReponseLUGAP\_updateField.KO("Erreur Serveur (SQL)");
38. traceEvent("Erreur SQL/BDD : " + ex.getMessage());
39. } catch (InterruptedException ex) {
40. this.reponse = ReponseLUGAP\_updateField.KO("Erreur Serveur (Interrupted)");
41. traceEvent("Erreur (Interrupted) : " + ex.getMessage());
42. }
43. this.communicator.SendSerializable(this.reponse);
44. } catch (CommunicatorException ex) {
45. traceEvent("UpdateFieldComments : " + ex.getMessage());
46. }
47. }
48. }

### Réponses

#### Reponse.java (du serveur générique)

1. package server;
2. import java.io.Serializable;
3. public abstract class Reponse implements Serializable {
4. private final String message;
5. private final boolean successful;
6. protected Reponse(String message, boolean successful) {
7. this.message = message;
8. this.successful = successful;
9. } //<editor-fold defaultstate="collapsed" desc="Getter & Setter">
10. public String getMessage() {
11. return this.message;
12. }
13. public boolean isSuccessful() {
14. return this.successful;
15. } //</editor-fold>
16. }

#### ReponseLUGAP\_login.java

1. package lugap.reponse;
2. import java.io.Serializable;
3. import server.Reponse;
4. public class ReponseLUGAP\_login extends Reponse implements Serializable {
5. private ReponseLUGAP\_login(String message, boolean successful) {
6. super(message, successful);
7. }
8. public static ReponseLUGAP\_login OK() {
9. return new ReponseLUGAP\_login("", true);
10. }
11. public static ReponseLUGAP\_login KO(String message) {
12. return new ReponseLUGAP\_login(message, false);
13. }
14. }

#### ReponseLUGAP\_logout.java

1. package lugap.reponse;
2. import java.io.Serializable;
3. import server.Reponse;
4. public class ReponseLUGAP\_logout extends Reponse implements Serializable {
5. private ReponseLUGAP\_logout(String message, boolean successful) {
6. super(message, successful);
7. }
8. public static ReponseLUGAP\_logout OK() {
9. return new ReponseLUGAP\_logout("", true);
10. }
11. public static ReponseLUGAP\_logout KO(String message) {
12. return new ReponseLUGAP\_logout(message, false);
13. }
14. }

#### ReponseLUGAP\_getFlights.java

1. package lugap.reponse;
2. import entities.Flight;
3. import java.io.Serializable;
4. import java.util.ArrayList;
5. import server.Reponse;
6. public class ReponseLUGAP\_getFlights extends Reponse implements Serializable {
7. ArrayList < Flight > flights;
8. private ReponseLUGAP\_getFlights(String message, boolean successful, ArrayList < Flight > flights) {
9. super(message, successful);
10. this.flights = flights;
11. }
12. public static ReponseLUGAP\_getFlights OK(ArrayList < Flight > flights) {
13. return new ReponseLUGAP\_getFlights("", true, flights);
14. }
15. public static ReponseLUGAP\_getFlights KO(String message) {
16. return new ReponseLUGAP\_getFlights(message, false, null);
17. }
18. public ArrayList < Flight > getFlights() {
19. return flights;
20. }
21. }

#### ReponseLUGAP\_getLuggages.java

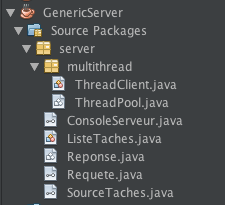
1. package lugap.reponse;
2. import entities.Luggage;
3. import java.io.Serializable;
4. import java.util.ArrayList;
5. import server.Reponse;
6. public class ReponseLUGAP\_getLuggages extends Reponse implements Serializable {
7. ArrayList < Luggage > luggages;
8. private ReponseLUGAP\_getLuggages(String message, boolean successful, ArrayList < Luggage > luggages) {
9. super(message, successful);
10. this.luggages = luggages;
11. }
12. public static ReponseLUGAP\_getLuggages OK(ArrayList < Luggage > luggages) {
13. return new ReponseLUGAP\_getLuggages("", true, luggages);
14. }
15. public static ReponseLUGAP\_getLuggages KO(String message) {
16. return new ReponseLUGAP\_getLuggages(message, false, null);
17. }
18. public ArrayList < Luggage > getLuggages() {
19. return luggages;
20. }
21. }

#### ReponseLUGAP\_updateField.java

1. package lugap.reponse;
2. import java.io.Serializable;
3. import server.Reponse;
4. public class ReponseLUGAP\_updateField extends Reponse implements Serializable {
5. private ReponseLUGAP\_updateField(String message, boolean successful) {
6. super(message, successful);
7. }
8. public static ReponseLUGAP\_updateField OK() {
9. return new ReponseLUGAP\_updateField("", true);
10. }
11. public static ReponseLUGAP\_updateField KO(String message) {
12. return new ReponseLUGAP\_updateField(message, false);
13. }
14. }

# Code Java du serveur Serveur\_Bagages

Certaines classes n’ont pas été inséré dans cette partie car elles se trouvent dans le Syllabus « Java II – Programmation avancée des applications classiques et cryptographie » et n’ont pas été modifiée. Ces classes sont :



## JFrameServeurBagages.java

1. package serveur\_bagages;
2. import database.utilities.DatabaseAccess;
3. import java.awt.Component;
4. import java.io.FileInputStream;
5. import java.io.FileNotFoundException;
6. import java.io.FileOutputStream;
7. import java.io.IOException;
8. import java.net.ServerSocket;
9. import java.sql.SQLException;
10. import java.util.Properties;
11. import java.util.StringTokenizer;
12. import java.util.Vector;
13. import java.util.logging.Level;
14. import java.util.logging.Logger;
15. import javax.swing.JLabel;
16. import javax.swing.JScrollBar;
17. import javax.swing.JTable;
18. import javax.swing.JTextArea;
19. import javax.swing.table.DefaultTableCellRenderer;
20. import javax.swing.table.DefaultTableModel;
21. import javax.swing.table.TableCellRenderer;
22. import server.ConsoleServeur;
23. import server.ListeTaches;
24. public class JFrameServeurBagages extends javax.swing.JFrame implements ConsoleServeur {
25. private int ligneConsoleServeur = 1;
26. private final String propertyFilePath = "../serveurBagage.conf";
27. private final Properties propertyFile = new Properties();
28. private ThreadPoolLUGAP poolThread;
29. private int NB\_THREAD\_LUGAP;
30. private boolean isRunning = false;
31. private String PORT\_BAGAGES;
32. private String PORT\_CHECKIN;
33. private String DB\_IP\_ADDRESS;
34. private String DB\_PORT;
35. private String DB\_SCHEMA;
36. private String DB\_LOGIN;
37. private String DB\_PASSWORD;
38. private DatabaseAccess databaseAccess;
39. private final DefaultTableModel defaultTableModel = new DefaultTableModel(); /\*\* Creates new form JFrameServeurBagage \*/
40. public JFrameServeurBagages() {
41. initComponents();
42. defaultTableModel.addColumn("Ligne");
43. defaultTableModel.addColumn("Origine");
44. defaultTableModel.addColumn("Message");
45. defaultTableModel.addColumn("Lieu");
46. jTableConsole.setModel(this.defaultTableModel); //On change la taille des colonnes et l'alignement du texte
47. jTableConsole.getColumnModel().getColumn(0).setMaxWidth(40);
48. DefaultTableCellRenderer rightRenderer = new DefaultTableCellRenderer();
49. rightRenderer.setHorizontalAlignment(JLabel.RIGHT);
50. jTableConsole.getColumnModel().getColumn(0).setCellRenderer(rightRenderer);
51. jTableConsole.getColumnModel().getColumn(1).setMaxWidth(250);
52. jTableConsole.getColumnModel().getColumn(1).setPreferredWidth(120);
53. jTableConsole.getColumnModel().getColumn(2).setCellRenderer(new WordWrapCellRenderer());
54. jTableConsole.getColumnModel().getColumn(3).setMaxWidth(250);
55. jTableConsole.getColumnModel().getColumn(3).setPreferredWidth(115);
56. loadPropertyFile();
57. this.jTextFieldPort.setText(this.PORT\_BAGAGES);
58. }
59. /\*\* This method is called from within the constructor to     \* initialize the form.     \* WARNING: Do NOT modify this code. The content of this method is     \* always regenerated by the Form Editor.     \*/
60. @
61. SuppressWarnings("unchecked") // <editor-fold defaultstate="collapsed" desc="Generated Code">
62. private void initComponents() { //[...]
63. } // </editor-fold>
64. private void jButtonStartStopActionPerformed(java.awt.event.ActionEvent evt) {
65. try {
66. if (!isRunning) {
67. TraceEvenements("serveur#Démarrage du serveur...#main");
68. this.jButtonStartStop.setText("Stop");
69. TraceEvenements("serveur#acquisition du port#main");
70. String port = this.jTextFieldPort.getText();
71. if (!this.PORT\_BAGAGES.equals(port)) setPORT\_BAGAGES(port);
72. TraceEvenements("serveur#Lancement des threads#main");
73. int portInt = Integer.valueOf(this.PORT\_BAGAGES);
74. this.poolThread = new ThreadPoolLUGAP(this.NB\_THREAD\_LUGAP, new ListeTaches(), this, portInt, this.DB\_IP\_ADDRESS, this.DB\_PORT, this.DB\_SCHEMA, this.DB\_LOGIN, this.DB\_PASSWORD);
75. new ServerSocket(portInt).close(); //On check si la socket est libre ou non
76. this.poolThread.start();
77. this.isRunning = true;
78. TraceEvenements("serveur#Threads Lancés#main");
79. } else {
80. TraceEvenements("serveur#Extinction du serveur...#main");
81. this.jButtonStartStop.setText("Start");
82. this.poolThread.doStop();
83. this.isRunning = false;
84. }
85. } catch (IOException ex) {
86. TraceEvenements("serveur#ERREUR : " + ex.getMessage() + "#main");
87. this.jButtonStartStop.setText("Start");
88. this.poolThread.doStop();
89. this.isRunning = false;
90. }
91. }
92. private void buttonClearActionPerformed(java.awt.event.ActionEvent evt) {
93. ((DefaultTableModel) this.jTableConsole.getModel()).setRowCount(0);
94. }
95. /\*\*     \* @param args the command line arguments     \*/
96. public static void main(String args[]) { /\* Set the Nimbus look and feel \*/ //[...]
97. /\* Create and display the form \*/
98. java.awt.EventQueue.invokeLater(() - > {
99. new JFrameServeurBagages().setVisible(true);
100. });
101. } // Variables declaration - do not modify
102. private javax.swing.JButton buttonClear;
103. private javax.swing.JButton jButtonStartStop;
104. private javax.swing.JLabel jLabelPort;
105. private javax.swing.JScrollPane jScrollPane1;
106. private javax.swing.JSeparator jSeparator1;
107. private javax.swing.JTable jTableConsole;
108. private javax.swing.JTextField jTextFieldPort;
109. private javax.swing.JLabel labelServerStatus; // End of variables declaration
110. @
111. Override public void TraceEvenements(String commentaire) {
112. Vector ligne = new Vector();
113. ligne.add(Integer.toString(ligneConsoleServeur++));
114. StringTokenizer parser = new StringTokenizer(commentaire, "#");
115. while (parser.hasMoreTokens()) ligne.add(parser.nextToken());
116. defaultTableModel.addRow(ligne);
117. jScrollPane1.validate();
118. JScrollBar bar = this.jScrollPane1.getVerticalScrollBar();
119. bar.setValue(bar.getMaximum());
120. }
121. private void setPORT\_BAGAGES(String newPort) {
122. this.PORT\_BAGAGES = newPort;
123. this.propertyFile.setProperty("PORT\_BAGAGES", newPort);
124. try (FileOutputStream fileOutputStream = new FileOutputStream(this.propertyFilePath)) {
125. this.propertyFile.store(fileOutputStream, null);
126. } catch (IOException ex) {
127. Logger.getLogger(JFrameServeurBagages.class.getName()).log(Level.SEVERE, null, ex);
128. System.exit(1);
129. }
130. }
131. private void loadPropertyFile() {
132. try (FileInputStream fileInputStream = new FileInputStream(this.propertyFilePath)) {
133. this.propertyFile.load(fileInputStream);
134. } catch (FileNotFoundException ex) {
135. try {
136. propertyFile.setProperty("THREAD\_NB\_LUGAP", "10");
137. propertyFile.setProperty("PORT\_BAGAGES", "26010");
138. propertyFile.setProperty("PORT\_CHECKIN", "26020");
139. propertyFile.setProperty("DB\_IP\_ADDRESS", "127.0.0.1");
140. propertyFile.setProperty("DB\_PORT", "3306");
141. propertyFile.setProperty("DB\_SCHEMA", "BD\_AIRPORT");
142. propertyFile.setProperty("DB\_LOGIN", "LaboReseaux");
143. propertyFile.setProperty("DB\_PASSWORD", "mysql");
144. try (FileOutputStream fileOutputStream = new FileOutputStream(this.propertyFilePath)) {
145. propertyFile.store(fileOutputStream, null);
146. }
147. } catch (IOException ex1) {
148. Logger.getLogger(JFrameServeurBagages.class.getName()).log(Level.SEVERE, null, ex1);
149. }
150. } catch (IOException ex) {
151. Logger.getLogger(JFrameServeurBagages.class.getName()).log(Level.SEVERE, null, ex);
152. System.exit(1);
153. }
154. this.NB\_THREAD\_LUGAP = Integer.valueOf(propertyFile.getProperty("THREAD\_NB\_LUGAP", "10"));
155. this.PORT\_BAGAGES = propertyFile.getProperty("PORT\_BAGAGES", "26010");
156. this.PORT\_CHECKIN = propertyFile.getProperty("PORT\_CHECKIN", "26020");
157. this.DB\_IP\_ADDRESS = propertyFile.getProperty("DB\_IP\_ADDRESS", "127.0.0.1");
158. this.DB\_PORT = propertyFile.getProperty("DB\_PORT", "3306");
159. this.DB\_SCHEMA = propertyFile.getProperty("DB\_SCHEMA", "BD\_AIRPORT");
160. this.DB\_LOGIN = propertyFile.getProperty("DB\_LOGIN", "LaboReseaux");
161. this.DB\_PASSWORD = propertyFile.getProperty("DB\_PASSWORD", "mysql");
162. }
163. static class WordWrapCellRenderer extends JTextArea implements TableCellRenderer {
164. WordWrapCellRenderer() {
165. setLineWrap(true);
166. setWrapStyleWord(true);
167. setOpaque(true);
168. }@
169. Override public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
170. setText(value.toString());
171. setSize(table.getColumnModel().getColumn(column).getWidth(), getPreferredSize().height);
172. setOpaque(true);
173. setBorder(null);
174. setForeground(table.getForeground());
175. setBackground(table.getBackground());
176. if (table.getRowHeight(row) != getPreferredSize().height) {
177. table.setRowHeight(row, getPreferredSize().height);
178. }
179. return this;
180. }
181. }
182. }

## ThreadPool.java

1. package server.multithread;
2. import communicator.CommunicatorException;
3. import java.io.IOException;
4. import java.net.ServerSocket;
5. import java.net.Socket;
6. import java.util.ArrayList;
7. import java.util.logging.Level;
8. import java.util.logging.Logger;
9. import server.ConsoleServeur;
10. import server.SourceTaches;
11. public abstract class ThreadPool extends Thread {
12. protected final int listeningPort;
13. protected final SourceTaches tachesAExecuter;
14. protected final ConsoleServeur guiApplication;
15. protected ServerSocket serverSocket = null;
16. protected final int nbThreads;
17. private final ArrayList < ThreadClient > threadClients;
18. public ThreadPool(int nbThreads, SourceTaches tachesAExecuter, ConsoleServeur guiApplication, int port) {
19. this.listeningPort = port;
20. this.tachesAExecuter = tachesAExecuter;
21. this.guiApplication = guiApplication;
22. this.nbThreads = nbThreads;
23. this.threadClients = new ArrayList < > ();
24. }@
25. Override public void run() {
26. try {
27. this.serverSocket = new ServerSocket(this.listeningPort);
28. } catch (IOException ex) {
29. Logger.getLogger(ThreadPool.class.getName()).log(Level.SEVERE, null, ex);
30. System.exit(1);
31. } //Démarrage du pool de threads
32. for (int i = 0; i < this.nbThreads; i++) {
33. ThreadClient client = new ThreadClient(tachesAExecuter, "Thread du pool n°" + String.valueOf(i));
34. client.start();
35. this.threadClients.add(client);
36. } //Mise en attente du serveur
37. Socket socket;
38. while (!isInterrupted()) {
39. try {
40. socket = serverSocket.accept();
41. this.guiApplication.TraceEvenements("serveur#Accepted an incoming connection#ThreadPoolLUGAP");
42. tachesAExecuter.recordTache(getProtocolRunnable(socket));
43. } catch (IOException ex) {
44. System.err.println("Erreur d'accept ! ? [" + ex.getMessage() + "]");
45. } catch (CommunicatorException ex) {
46. System.err.println("Erreur de communication ! ? [" + ex.getMessage() + "]");
47. }
48. }
49. }
50. public void doStop() {
51. this.guiApplication.TraceEvenements("serveur#Arret du serveur#ThreadPoolLUGAP");
52. try {
53. if (this.serverSocket != null) this.serverSocket.close();
54. this.threadClients.forEach(threadClient - > {
55. threadClient.interrupt();
56. });
57. } catch (IOException ex) {
58. this.guiApplication.TraceEvenements("serveur#" + ex.getMessage() + "#ThreadPoolLUGAP");
59. }
60. this.interrupt();
61. }
62. protected abstract Runnable getProtocolRunnable(Socket socket) throws CommunicatorException;
63. }

## ThreadPoolLUGAP.java

1. package serveur\_bagages;
2. import communicator.Communicator;
3. import communicator.CommunicatorException;
4. import database.utilities.DatabaseAccess;
5. import java.net.Socket;
6. import server.ConsoleServeur;
7. import server.SourceTaches;
8. import server.multithread.ThreadPool;
9. public class ThreadPoolLUGAP extends ThreadPool {
10. private final String DBip;
11. private final String DBport;
12. private final String DBSID;
13. private final String DBschema;
14. private final String DBpassword;
15. public ThreadPoolLUGAP(int nbThreads, SourceTaches tachesAExecuter, ConsoleServeur guiApplication, int port, String ip, String DBport, String SID, String schema, String password) {
16. super(nbThreads, tachesAExecuter, guiApplication, port);
17. this.DBip = ip;
18. this.DBport = DBport;
19. this.DBSID = SID;
20. this.DBschema = schema;
21. this.DBpassword = password;
22. }@
23. Override protected Runnable getProtocolRunnable(Socket socket) throws CommunicatorException {
24. return new RunnableLUGAP(this, this.guiApplication, new Communicator(socket), new DatabaseAccess(DatabaseAccess.databaseType.MYSQL, this.DBip, this.DBport, this.DBSID, this.DBschema, this.DBpassword));
25. }
26. }

## RunnableLUGAP.java

1. package serveur\_bagages;
2. import communicator.Communicator;
3. import communicator.CommunicatorException;
4. import database.utilities.DatabaseAccess;
5. import java.sql.SQLException;
6. import lugap.reponse.ReponseLUGAP\_getFlights;
7. import lugap.reponse.ReponseLUGAP\_getLuggages;
8. import lugap.reponse.ReponseLUGAP\_login;
9. import lugap.reponse.ReponseLUGAP\_updateField;
10. import lugap.requete.RequeteLUGAP;
11. import lugap.requete.RequeteLUGAP\_getFlights;
12. import lugap.requete.RequeteLUGAP\_getLuggages;
13. import lugap.requete.RequeteLUGAP\_login;
14. import lugap.requete.RequeteLUGAP\_logout;
15. import lugap.requete.RequeteLUGAP\_updateFieldCheckedByCustoms;
16. import lugap.requete.RequeteLUGAP\_updateFieldComments;
17. import lugap.requete.RequeteLUGAP\_updateFieldLoaded;
18. import lugap.requete.RequeteLUGAP\_updateFieldReceived;
19. import server.ConsoleServeur;
20. public class RunnableLUGAP implements Runnable {
21. private final ThreadPoolLUGAP parent;
22. private final ConsoleServeur guiApplication;
23. private final Communicator communicator;
24. private final DatabaseAccess databaseAccess;
25. private Class previousRequete;
26. private boolean endOfTransaction;
27. private boolean clientConnected;
28. RunnableLUGAP(ThreadPoolLUGAP parent, ConsoleServeur guiApplication, Communicator communicator, DatabaseAccess databaseAccess) throws CommunicatorException {
29. this.parent = parent;
30. this.guiApplication = guiApplication;
31. this.communicator = communicator;
32. this.databaseAccess = databaseAccess;
33. }@
34. Override public void run() {
35. boolean failedToConnect;
36. this.endOfTransaction = false;
37. try {
38. this.databaseAccess.connect();
39. failedToConnect = false;
40. } catch (ClassNotFoundException | SQLException ex) {
41. failedToConnect = true;
42. }
43. try {
44. while (!this.endOfTransaction) {
45. RequeteLUGAP req = this.communicator.receiveSerializable(RequeteLUGAP.class);
46. System.out.println("Received request type : " + req.getClass().toString());
47. if (failedToConnect) {
48. this.communicator.SendSerializable(ReponseLUGAP\_login.KO("Server could not connect to the database."));
49. endTransaction();
50. } else {
51. Runnable runnable = req.createRunnable(this.parent, this.communicator, this.guiApplication, this.databaseAccess);
52. if (req instanceof RequeteLUGAP\_login) {
53. runnable.run();
54. if (req.requeteSucceeded()) {
55. setClientIsConnected();
56. previousRequete = RequeteLUGAP\_login.class;
57. }
58. } else if (req instanceof RequeteLUGAP\_logout) {
59. runnable.run();
60. endTransaction();
61. previousRequete = null;
62. } else if (req instanceof RequeteLUGAP\_getFlights) {
63. if (!isClientConnected()) this.communicator.SendSerializable(ReponseLUGAP\_getFlights.KO("Please connect first!"));
64. else {
65. runnable.run();
66. previousRequete = RequeteLUGAP\_getFlights.class;
67. }
68. } else if (req instanceof RequeteLUGAP\_getLuggages) {
69. if (!isClientConnected() || !previousRequete.equals(RequeteLUGAP\_getFlights.class)) this.communicator.SendSerializable(ReponseLUGAP\_getLuggages.KO("Please do a getFlights first!"));
70. else {
71. runnable.run();
72. previousRequete = RequeteLUGAP\_getLuggages.class;
73. }
74. } else if (req instanceof RequeteLUGAP\_updateFieldReceived || req instanceof RequeteLUGAP\_updateFieldLoaded || req instanceof RequeteLUGAP\_updateFieldCheckedByCustoms || req instanceof RequeteLUGAP\_updateFieldComments) {
75. if (!isClientConnected() || !previousRequete.equals(RequeteLUGAP\_getLuggages.class)) this.communicator.SendSerializable(ReponseLUGAP\_updateField.KO("Please do a getLuggages first!"));
76. else runnable.run();
77. }
78. }
79. }
80. this.databaseAccess.commit();
81. this.guiApplication.TraceEvenements(this.communicator.getSocket().getRemoteSocketAddress().toString() + "#Commit successful" + "#LUGAPRunnable");
82. this.communicator.close();
83. } catch (CommunicatorException | SQLException ex) {
84. this.guiApplication.TraceEvenements(this.communicator.getSocket().getRemoteSocketAddress().toString() + "#" + ex.getMessage() + "#LUGAPRunnable");
85. try {
86. this.databaseAccess.rollback();
87. this.guiApplication.TraceEvenements(this.communicator.getSocket().getRemoteSocketAddress().toString() + "#Rollback successful" + "#LUGAPRunnable");
88. } catch (SQLException ex1) {
89. this.guiApplication.TraceEvenements(this.communicator.getSocket().getRemoteSocketAddress().toString() + "#" + ex.getMessage() + "#LUGAPRunnable");
90. }
91. }
92. }
93. private void endTransaction() {
94. this.endOfTransaction = true;
95. }
96. private boolean isClientConnected() {
97. return this.clientConnected;
98. }
99. private void setClientIsConnected() {
100. this.clientConnected = true;
101. }
102. }