```java
package com.nxkundu.client.service;

import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.lang.reflect.Type;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.Date;
import java.util.HashMap;
import java.util.UUID;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.ConcurrentMap;

import javax.imageio.ImageIO;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import com.nxkundu.server.bo.Client;
import com.nxkundu.server.bo.DataPacket;
import com.nxkundu.server.bo.Server;

/**
 *
 * @author nxkundu
 *
 * @email nxk161830@utdallas.edu
 * @name Nirmallya Kundu
 *
 * ClientService - This service is initialized on the Client Side
 * When the user opens the Chat Application
 * This is a singleton class
 *
```

```
38 * These are the below methods:
39 *
40 * 1> getInstance()
41 * - ClientService() - singleton class
42 * This Service is initialized at each individual Client Side
43 * When the client access to the Client Chat Application
44 *
45 * 2> login()
46 * - This method is called by the UI Chat Application
47 * when the client wants to Login
48 * This basically sends the LOGIN DataPacket to the
49 * server, requesting to login with username and password
50 *
51 * 3> signup()
52 * - This method is called by the UI Chat Application
53 * when the client wants to Signup for the first Time
54 * This basically sends the SIGNUP DataPacket to the
55 * server, requesting to SIGNUP AND LOGIN with username and password
56 *
57 * 4> logout()
58 * - This method is called by the UI Chat Application
59 * when the client wants to Logout
60 * This basically sends the Logout DataPacket to the
61 * server, requesting to Logout with username and password
62 *
63 * 5> sendPacket()
64 * - This method creates the DataPacket
65 * by the parameters of the DataPacket and
66 * send the DataPacket to the method sendPacket(DataPacket dataPacket)
67 * which handles the sending of the DataPacket to the server
68 *
69 * 6> resendDataPacketIfNoACKReceived()
70 * - This method runs the
71 * thread threadResendDataPacketIfNoACKReceived
72 * which Resends the data for which no ACK is received
73 * from the server after a predefined amount of time
74 *
```

```
 75 * 7> addReceivedDataPacket()
 76 * - As soon as the client receives any DataPacket
 77 * After reading the FROM_CLIENT in the DataPacket
 78 * the DataPacket is added to the queue of the respective queue
 79 * which is in-turn stored in the mapClientReceivedDataPacket
 80 *
 81 * 8> recievePacketUDP()
 82 * - This method runs the thread threadReceivePacketUDP
 83 * and continuously receives UDP DataPacket from the server and process
 84 * them to find the content of the packet and perform the necessary action
 85 *
 86 * 9> processReceivedDatagramPacket()
 87 * - This method takes the action on the
 88 * received DataPacket based on the action field in the DataPacket
 89 *
 90 * 10> sendPacketOnlineStatus()
 91 * - This method runs the thread threadOnlineStatus and
 92 * continuously send Online DataPacket to the server to notify that the client is ONLINE.
 93 * When the server does not receive the  Online DataPacketfrom a client for more than 3 cycle,
 94 * the server assumes that the client is OFFLINE
 95 *
 96 * 11> sendPacketByUDP()
 97 *  - This methods sends the DataPacket to the
 98 * server based on UDP DatagramPacket
 99 *
100 */
101 public class ClientService implements Runnable{
102
103     /**
104      * Variable declarations
105      */
106
107     /*
108      *  private Server server - holds the information about the server
109      *  this is used when the client on the client side
110      *  connects to the server
111      *
```

```
112     * private Client client - holds the client information
113     *
114     * public static boolean isLoggedIn - This variable is changed to true when the server
115     * responds with Successful Logged In message
116     *
117     * private Thread threadService - This is the main thread which runs on the Client Side
118     *
119     * private Thread threadReceivePacketUDP - This thread continuously receives
120     * UDP DataPacket from the server and process them to find the content of the
121     * packet and perform the necessary action
122     *
123     * private Thread threadOnlineStatus - THis thread continuously send Online DataPacket to the server
124     * to notify that the client is ONLINE. When the server does not receive the  Online DataPacket
125     * from a client for more than 3 cycle, the server assumes that the client is OFFLINE
126     *
127     * private Thread threadResendDataPacketIfNoACKReceived - This thread Resends the data for which no ACK
128     * is received from the server after a predefined amount of time
129     *
130     * private ConcurrentMap<String, Client> mapAllClients - This map stores the list of all clients
131     * received from the server, this is basically from where we receive the ONLINE clients and OFFLINE clients
132     *
133     * private ConcurrentMap<UUID, DataPacket> mapSentDataPacket - This map stores all the DataPackets
134     * that was sent to the server, and when the client receives the ACK for the DataPacket,
135     * the respective DataPacket is removed from the map
136     *
137     * private ConcurrentMap<UUID, DataPacket> mapReceivedDataPacket - This map stores all the DataPackets
138     * received from the server so that the client can send ACK to the server that it has
139     * successfully received the DataPacket
140     *
141     * private ConcurrentLinkedQueue<DataPacket> qSignupLoginLogoutDataPacket - This queue stores the
142     * LOGIN and LOGOUT DataPacket that the client sends to the server while login and logout respectively
143     *
144     * private ConcurrentMap<String, ConcurrentLinkedQueue<DataPacket>> mapClientReceivedDataPacket - This map
145     * stores the queue of DataPacket received from the server for each individual client
146     *
147     * private static ClientService clientService - this is used to make the ClientService class a singleton class.
148     *
```

```
149      *
150      */
151     private Server server;
152     private Client client;
153     public static boolean isLoggedIn;
154 
155     private Thread threadService;
156     private Thread threadReceivePacketUDP;
157     private Thread threadOnlineStatus;
158     private Thread threadResendDataPacketIfNoACKReceived;
159 
160     private ConcurrentMap<String, Client> mapAllClients;
161 
162     private ConcurrentMap<UUID, DataPacket> mapSentDataPacket;
163     private ConcurrentMap<UUID, DataPacket> mapReceivedDataPacket;
164 
165     private ConcurrentLinkedQueue<DataPacket> qSignupLoginLogoutDataPacket;
166 
167     private ConcurrentMap<String, ConcurrentLinkedQueue<DataPacket>> mapClientReceivedDataPacket;
168 
169     private static ClientService clientService;
170 
171 
172     /***************************** Constructors *****************************/
173 
174     private ClientService() {
175 
176         super();
177 
178         isLoggedIn = false;
179 
180         mapClientReceivedDataPacket = new ConcurrentHashMap<>();
181 
182         mapSentDataPacket = new ConcurrentHashMap<>();
183         mapReceivedDataPacket = new ConcurrentHashMap<>();
184 
185         qSignupLoginLogoutDataPacket = new ConcurrentLinkedQueue<>();
```

```
186
187         try {
188
189             /*
190              * server - gives a reference to the server object
191              *
192              * server.connectToServer() - connects to the server for the first time
193              * to send LOGIN DataPackets and the other following DataPackets
194              *
195              */
196             server = Server.getInstance();
197             server.connectToServer();
198
199         }
200         catch(SocketException e) {
201
202             e.printStackTrace();
203         }
204         catch (UnknownHostException e) {
205
206             e.printStackTrace();
207         }
208         catch (IOException e) {
209
210             e.printStackTrace();
211         }
212
213         threadService = new Thread(this, "ClientStart");
214         threadService.start();
215     }
216
217     /****************************** Object Methods ******************************/
218
219     /**
220      * ClientService() - singleton class
221      * This Service is initialized at each individual Client Side
222      * When the client access to the Client Chat Application
```

```
223     */
224    public static ClientService getInstance() {
225
226        if(clientService == null) {
227            clientService = new ClientService();
228        }
229
230        isLoggedIn = true;
231        return clientService;
232    }
233
234    @Override
235    public void run() {
236
237        /*
238         * recievePacketUDP - This method runs the thread threadReceivePacketUDP
239         * and continuously receives UDP DataPacket from the server and process
240         * them to find the content of the packet and perform the necessary action
241         */
242        recievePacketUDP();
243
244        /*
245         * sendPacketOnlineStatus() - This method runs the thread threadOnlineStatus and
246         * continuously send Online DataPacket to the server to notify that the client is ONLINE.
247         * When the server does not receive the  Online DataPacketfrom a client for more than 3 cycle,
248         * the server assumes that the client is OFFLINE
249         */
250        sendPacketOnlineStatus();
251
252        /*
253         * resendDataPacketIfNoACKReceived() - This method runs the
254         * thread threadResendDataPacketIfNoACKReceived
255         * which Resends the data for which no ACK is received
256         * from the server after a predefined amount of time
257         */
258        resendDataPacketIfNoACKReceived();
259
```

```
260     }
261
262     /**
263      * This method is called by the UI Chat Application
264      * when the client wants to Login
265      * This basically sends the LOGIN DataPacket to the
266      * server, requesting to login with username and password
267      *
268      * @param userName
269      * @param password
270      */
271     public void login(String userName, String password) {
272
273         try {
274
275             client = new Client(userName, password);
276             DataPacket dataPacket = new DataPacket(client, DataPacket.ACTION_TYPE_LOGIN);
277
278             sendPacket(dataPacket);
279
280         }
281         catch (IOException e) {
282
283             e.printStackTrace();
284         }
285     }
286
287     /**
288      * This method is called by the UI Chat Application
289      * when the client wants to Signup for the first Time
290      * This basically sends the SIGNUP DataPacket to the
291      * server, requesting to SIGNUP AND LOGIN with username and password
292      *
293      * @param userName
294      * @param password
295      */
296     public void signup(String userName, String password) {
```

```
297
298         try {
299
300             client = new Client(userName, password);
301             DataPacket dataPacket = new DataPacket(client, DataPacket.ACTION_TYPE_SIGNUP);
302
303             sendPacket(dataPacket);
304
305         }
306         catch (IOException e) {
307
308             e.printStackTrace();
309         }
310     }
311
312     /**
313      * This method is called by the UI Chat Application
314      * when the client wants to Logout
315      * This basically sends the Logout DataPacket to the
316      * server, requesting to Logout with username and password
317      *
318      */
319     public void logout() {
320
321         try {
322
323             DataPacket dataPacket = new DataPacket(client, DataPacket.ACTION_TYPE_LOGOUT);
324
325             sendPacket(dataPacket);
326
327         }
328         catch (IOException e) {
329
330             e.printStackTrace();
331         }
332
333         isLoggedIn = false;
```

```
334     }
335
336     /**
337      * sendPacket() - This method creates the DataPacket
338      * by the parameters of the DataPacket and
339      * send the DataPacket to the method sendPacket(DataPacket dataPacket)
340      * which handles the sending of the DataPacket to the server
341      *
342      * @param messageType
343      * @param message
344      * @param toClientUserName
345      */
346     public void sendPacket(String messageType, String message, String toClientUserName) {
347
348         try {
349
350             DataPacket dataPacket = new DataPacket(client, DataPacket.ACTION_TYPE_MESSAGE);
351             dataPacket.setMessage(message);
352
353             boolean isValid = false;
354             if(DataPacket.MESSAGE_TYPE_MESSAGE.equalsIgnoreCase(messageType)) {
355
356                 dataPacket.setMessageType(DataPacket.MESSAGE_TYPE_MESSAGE);
357                 Client toClient = new Client(toClientUserName);
358                 dataPacket.setToClient(toClient);
359
360                 isValid = true;
361             }
362             else if(DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE.equalsIgnoreCase(messageType)) {
363
364                 dataPacket.setMessageType(DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE);
365                 isValid = true;
366             }
367             else if(DataPacket.MESSAGE_TYPE_IMAGE_MESSAGE.equalsIgnoreCase(messageType)) {
368
369                 dataPacket.setMessageType(DataPacket.MESSAGE_TYPE_IMAGE_MESSAGE);
370                 System.out.println(message);
```

```
371                 System.out.println(new File(message).exists());
372                 BufferedImage bufferedImage = ImageIO.read(new File(message));
373                 ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
374                 ImageIO.write(bufferedImage, "jpg", byteArrayOutputStream);
375                 byteArrayOutputStream.flush();
376                 dataPacket.setByteImage(byteArrayOutputStream.toByteArray());
377
378                 Client toClient = new Client(toClientUserName);
379                 dataPacket.setToClient(toClient);
380
381                 isValid = true;
382             }
383
384
385             if(isValid) {
386
387                 //addClientSendReceiveDataPacket(dataPacket);
388
389                 sendPacket(dataPacket);
390             }
391
392         }
393         catch (IOException e) {
394
395             e.printStackTrace();
396         }
397
398     }
399
400     /**
401      * recievePacketUDP - This method runs the thread threadReceivePacketUDP
402      * and continuously receives UDP DataPacket from the server and process
403      * them to find the content of the packet and perform the necessary action
404      */
405     public void recievePacketUDP() {
406
407         threadReceivePacketUDP = new Thread("RecievePacketUDP"){
```

```
408
409             @Override
410             public void run() {
411
412                 while(true) {
413
414                     if(isLoggedIn) {
415
416                         byte[] data = new byte[1024*60];
417                         DatagramPacket datagramPacket = new DatagramPacket(data, data.length);
418
419                         try {
420
421                             server.getDatagramSocket().receive(datagramPacket);
422
423                             String received = new String(datagramPacket.getData(), 0, datagramPacket.getLength());
424                             DataPacket dataPacket = new Gson().fromJson(received, DataPacket.class);
425                             System.out.println(dataPacket);
426
427                             processReceivedDatagramPacket(dataPacket);
428
429                         }
430                         catch (IOException e) {
431
432                             e.printStackTrace();
433                         }
434                     }
435
436                     try {
437
438                         Thread.sleep(500);
439                     }
440                     catch(Exception e) {
441
442                         e.printStackTrace();
443                     }
444                 }
```

```
445             }
446         };
447 
448         threadReceivePacketUDP.start();
449     }
450 
451     /**
452      * As soon as the client receives any DataPacket
453      * After reading the FROM_CLIENT in the DataPacket
454      * the DataPacket is added to the queue of the respective queue
455      * which is in-turn stored in the mapClientReceivedDataPacket
456      *
457      * @param dataPacket
458      */
459     private void addReceivedDataPacket(DataPacket dataPacket) {
460 
461         ConcurrentLinkedQueue<DataPacket> qClientReceived = null;
462         if(mapClientReceivedDataPacket.containsKey(dataPacket.getFromClient().getUserName())) {
463             qClientReceived = mapClientReceivedDataPacket.get(dataPacket.getFromClient().getUserName());
464         }
465         else {
466             qClientReceived = new ConcurrentLinkedQueue<>();
467         }
468 
469         qClientReceived.add(dataPacket);
470         mapClientReceivedDataPacket.put(dataPacket.getFromClient().getUserName(), qClientReceived);
471 
472     }
473 
474     /**
475      * resendDataPacketIfNoACKReceived() - This method runs the
476      * thread threadResendDataPacketIfNoACKReceived
477      * which Resends the data for which no ACK is received
478      * from the server after a predefined amount of time
479      */
480     public void resendDataPacketIfNoACKReceived() {
481 
```

```
482         threadResendDataPacketIfNoACKReceived = new Thread("ResendDataPacketIfNoACKReceived"){
483 
484             @Override
485             public void run() {
486 
487                 while(true) {
488 
489                     if(isLoggedIn) {
490 
491                         try {
492 
493                             if(mapSentDataPacket.size() > 0) {
494 
495                                 for(UUID sentDataPacketId : mapSentDataPacket.keySet()) {
496 
497                                     DataPacket sentDataPacket = mapSentDataPacket.get(sentDataPacketId);
498 
499                                     if(sentDataPacket.getTimestamp() - new Date().getTime() > 5000) {
500 
501                                         sentDataPacket.setTimestamp(new Date().getTime());
502                                         sentDataPacket.incrementTimesResentDataPacket();
503                                         mapSentDataPacket.put(sentDataPacketId, sentDataPacket);
504 
505                                         sendPacket(sentDataPacket);
506                                     }
507                                 }
508                             }
509                         }
510                         catch (IOException e) {
511 
512                             e.printStackTrace();
513                         }
514                     }
515 
516                     try {
517 
518                         Thread.sleep(500);
```

```
519                     }
520                     catch(Exception e) {
521 
522                         e.printStackTrace();
523                     }
524 
525                 }
526             }
527         };
528 
529         threadResendDataPacketIfNoACKReceived.start();
530     }
531 
532     /**
533      * processReceivedDatagramPacket() - This method takes the action on the
534      * received DataPacket based on the action field in the DataPacket
535      *
536      * @param dataPacket
537      */
538     private void processReceivedDatagramPacket(DataPacket dataPacket) {
539 
540         switch(dataPacket.getAction()) {
541 
542         case DataPacket.ACTION_TYPE_LOGIN_SUCCESS:
543             System.out.println("Received Login Packet Success!");
544             isLoggedIn = true;
545             qSignupLoginLogoutDataPacket.add(dataPacket);
546             break;
547 
548         case DataPacket.ACTION_TYPE_LOGIN_FAILED:
549             System.out.println("Received Login Packet Failed!");
550             isLoggedIn = false;
551             qSignupLoginLogoutDataPacket.add(dataPacket);
552             break;
553 
554         case DataPacket.ACTION_TYPE_SIGNUP_FAILED:
555             System.out.println("Received Signup Packet Failed!");
```

```
556             isLoggedIn = false;
557             qSignupLoginLogoutDataPacket.add(dataPacket);
558             break;
559 
560         case DataPacket.ACTION_TYPE_ONLINE:
561 
562             Type type = new TypeToken<HashMap<String, Client>>(){}.getType();
563             mapAllClients =  new ConcurrentHashMap<>(new Gson().fromJson(dataPacket.getMessage(), type));
564 
565             break;
566 
567         case DataPacket.ACTION_TYPE_ACK:
568 
569             UUID dataPacketACKId = UUID.fromString(dataPacket.getMessage());
570 
571             if(mapSentDataPacket.containsKey(dataPacketACKId)) {
572 
573                 mapSentDataPacket.remove(dataPacketACKId);
574             }
575             else {
576 
577                 //Not Possible
578             }
579 
580             break;
581 
582 
583         case DataPacket.ACTION_TYPE_MESSAGE:
584 
585             DataPacket dataPacketACK = new DataPacket(dataPacket.getFromClient(), DataPacket.ACTION_TYPE_ACK);
586             dataPacketACK.setMessage(dataPacket.getId().toString());
587 
588             if(mapReceivedDataPacket.containsKey(dataPacket.getId())) {
589 
590                 try {
591 
592                     sendPacket(dataPacketACK);
```

```
593
594                 }
595                 catch (IOException e) {
596                     e.printStackTrace();
597                 }
598                 break;
599             }
600
601             mapReceivedDataPacket.put(dataPacket.getId(), dataPacket);
602             System.out.println(dataPacket.getFromClient().getUserName() + " => " + dataPacket.getMessage());
603
604             switch (dataPacket.getMessageType()) {
605
606             case DataPacket.MESSAGE_TYPE_MESSAGE:
607
608                 addReceivedDataPacket(dataPacket);
609                 break;
610
611             case DataPacket.MESSAGE_TYPE_BROADCAST_MESSAGE:
612
613                 addReceivedDataPacket(dataPacket);
614                 break;
615
616             case DataPacket.MESSAGE_TYPE_IMAGE_MESSAGE:
617
618                 addReceivedDataPacket(dataPacket);
619                 break;
620
621             }
622
623             try {
624
625                 sendPacket(dataPacketACK);
626
627             }
628             catch (IOException e) {
629                 e.printStackTrace();
```

```
630             }
631 
632             break;
633         }
634 
635     }
636 
637     /**
638      * sendPacketOnlineStatus() - This method runs the thread threadOnlineStatus and
639      * continuously send Online DataPacket to the server to notify that the client is ONLINE.
640      * When the server does not receive the  Online DataPacketfrom a client for more than 3 cycle,
641      * the server assumes that the client is OFFLINE
642      */
643     private void sendPacketOnlineStatus() {
644 
645         threadOnlineStatus = new Thread("SendOnlineStatus"){
646 
647             @Override
648             public void run() {
649 
650                 while(true) {
651 
652                     if(isLoggedIn) {
653 
654                         try {
655 
656                             DataPacket dataPacket = new DataPacket(client, DataPacket.ACTION_TYPE_ONLINE);
657 
658                             sendPacket(dataPacket);
659 
660                         }
661                         catch (IOException e) {
662 
663                             e.printStackTrace();
664                         }
665                     }
666                     try {
```

```
667
668                     Thread.sleep(4000);
669                 }
670                 catch(Exception e) {
671
672                     e.printStackTrace();
673                 }
674             }
675         }
676     };
677
678     threadOnlineStatus.start();
679 }
680
681 /**
682  * sendPacketByUDP() - This methods sends the DataPacket to the
683  * server based on UDP DatagramPacket
684  *
685  * @param dataPacket
686  * @throws IOException
687  */
688 public void sendPacketByUDP(DataPacket dataPacket) throws IOException {
689
690     InetAddress inetAddress = server.getInetAddress();
691     int port = server.getPort();
692     byte[] data = dataPacket.toJSON().getBytes();
693     DatagramPacket datagramPacket = new DatagramPacket(data, data.length, inetAddress, port);
694
695     server.getDatagramSocket().send(datagramPacket);
696 }
697
698 /**
699  * sendPacket() - This method decides on
700  * which method to use to send the DataPacket to the server
701  *
702  * @param dataPacket
703  * @throws IOException
```

```
704      */
705     public void sendPacket(DataPacket dataPacket) throws IOException {
706 
707         if(dataPacket.getAction().equals(DataPacket.ACTION_TYPE_MESSAGE)) {
708             mapSentDataPacket.put(dataPacket.getId(), dataPacket);
709         }
710 
711         sendPacketByUDP(dataPacket);
712 
713     }
714 
715     /***************************** Getters Setters *****************************/
716 
717     public Server getServer() {
718         return server;
719     }
720 
721     public void setServer(Server server) {
722         this.server = server;
723     }
724 
725     public Client getClient() {
726         return client;
727     }
728 
729     public void setClient(Client client) {
730         this.client = client;
731     }
732 
733     public static ClientService getClientService() {
734         return clientService;
735     }
736 
737     public static void setClientService(ClientService clientService) {
738         ClientService.clientService = clientService;
739     }
740 
```

```
741     public ConcurrentMap<String, Client> getMapAllClients() {
742         return mapAllClients;
743     }
744 
745     public ConcurrentMap<UUID, DataPacket> getMapSentDataPacket() {
746         return mapSentDataPacket;
747     }
748 
749     public void setMapSentDataPacket(ConcurrentMap<UUID, DataPacket> mapSentDataPacket) {
750         this.mapSentDataPacket = mapSentDataPacket;
751     }
752 
753     public ConcurrentMap<UUID, DataPacket> getMapReceivedDataPacket() {
754         return mapReceivedDataPacket;
755     }
756 
757     public void setMapReceivedDataPacket(ConcurrentMap<UUID, DataPacket> mapReceivedDataPacket) {
758         this.mapReceivedDataPacket = mapReceivedDataPacket;
759     }
760 
761     public ConcurrentMap<String, ConcurrentLinkedQueue<DataPacket>> getMapClientReceivedDataPacket() {
762         return mapClientReceivedDataPacket;
763     }
764 
765     public void setMapClientReceivedDataPacket(
766             ConcurrentMap<String, ConcurrentLinkedQueue<DataPacket>> mapClientReceivedDataPacket) {
767         this.mapClientReceivedDataPacket = mapClientReceivedDataPacket;
768     }
769 
770     public void setMapAllClients(ConcurrentMap<String, Client> mapAllClients) {
771         this.mapAllClients = mapAllClients;
772     }
773 
774     public ConcurrentLinkedQueue<DataPacket> getqSignupLoginLogoutDataPacket() {
775         return qSignupLoginLogoutDataPacket;
776     }
777 
```

```
778     public void setqSignupLoginLogoutDataPacket(ConcurrentLinkedQueue<DataPacket> qSignupLoginLogoutDataPacket) {
779         this.qSignupLoginLogoutDataPacket = qSignupLoginLogoutDataPacket;
780     }
781 
782 
783 }
784 
```