TAYLOR'S UNIVERSITY

Wisdom · Integrity · Excellence

TAYLOR'S UWE DUAL AWARD PROGRAMMES
MARCH 2017 SEMESTER

ITS61604: DISTRIBUTED APPLICATION DEVELOPMENT
ASSIGNMENT 2

HAND-IN DATE: 23rd June 2017

WEIGHTAGE: 20%

STUDENT DECLARATION

We declare that:

- we understand what is meant by plagiarism; and
- the implications of plagiarism have been explained to us by our lecturer.

This project is all our work and we have acknowledged any use of the published or unpublished works of other people.

STUDENT PARTICULARS

Student Name	Student ID
ANDREW NG KHAI WEN	0321227
LEE WEI LIANG	0320883
HENG HIAN WEE	0321343

LECTURER: Dr SARFRAZ NAWAZ BROHI

TABLE OF CONTENTS

PART I

1.1 Distributed Application Development Techniques Used

1.1.1 RMI

Remote Method Invocation provides communication between Java programs remotely. Based on Oracles website, “Java Remote Method Invocation”, RMI system allows an object running in one Java Virtual Machine (JVM) to invoke methods of an object running in another Java VM. For our application, we have implement RMI when the user clicks on "About Us" button in the Login Screen, it will initiate AUClient. AUServer will create a registry service and then bind the objects in the registry service. Then, AUClient will get instance of the registry service and then client program will look up for the objects from the registry. Lastly, the information that contains the authors' name, version number and etcetera will be shown on the screen.

```
1  import java.rmi.server.UnicastRemoteObject;
2  import java.rmi.RemoteException;
3  import java.io.*;
4  import javax.swing.JOptionPane;
5
6  public class AU extends UnicastRemoteObject implements AUInterface{
7      public AU() throws RemoteException{
8
9      }
10
11
12     public String newAU() throws RemoteException{
13
14         return "Wei Liang, Andrew, Henry (WAH). All Rights Reserved. 2017.";
15     }
16
17 }
```

Code Snippet 1: About Us Server Class

```
1  import java.rmi.server.UnicastRemoteObject;
2  import java.rmi.RemoteException;
3  import java.io.*;
4  import javax.swing.JOptionPane;
5
6  public class AU extends UnicastRemoteObject implements AUInterface{
7      public AU() throws RemoteException{
8
9      }
10
11
12     public String newAU() throws RemoteException{
13
14         return "Wei Liang, Andrew, Henry (WAH). All Rights Reserved. 2017.";
15     }
16
17 }
```

Code Snippet 2: About Us Class

```
1  import java.rmi.registry.LocateRegistry;
2  import java.rmi.registry.Registry;
3  import javax.swing.JOptionPane;
4
5  public class AUClient{
6      public static void main(String[]args) throws Exception {
7          Registry registry = LocateRegistry.getRegistry("127.0.0.1");
8          AUInterface
9          AUObject = (AUInterface)registry.lookup("ToShowAU");
10         JOptionPane.showMessageDialog(null,AUObject.newAU());
11 //         System.out.printf(AUObject.newAU());
12     }
13 }
```

Code Snippet 3: About Us Client Class

```
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3
4  public interface AUInterface extends Remote{
5
6     public String newAU() throws RemoteException;
7  }
```

Code Snippet 4: About Us Interface Class

1.1.2 UDP

UDP has many benefits namely sending packages without needing the establishment of the connection. For that, we have implemented UDP in our application which is used to send audio back and forth from the client and agent. Based on Kurose, J. F.; Ross, K. W. (2010). Computer Networking: A Top-Down Approach (5th ed.). Boston, MA: Pearson Education, UDP is used to send data packets across the network without the need of a handshake process; the non-dependence on sockets. Thus, audio files can be transmitted in a fast rate which is near to real time.

```
1  public void run() {
2              try {
3
4                  stopCapture = false;
5
6                  //Get everything set up for capture
7                  audioFormat = getAudioFormat();
8                  DataLine.Info dataLineInfo = new DataLine.Info(TargetDataLine.class, audioFormat);
9                  targetDataLine = (TargetDataLine)
10                 AudioSystem.getLine(dataLineInfo);
11                 targetDataLine.open(audioFormat);
12                 targetDataLine.start();
13
14                 try{
15                     //Loop until stopCapture is set by another thread.
16                     while(!stopCapture){
17                         //Read data from the internal buffer of the data line.
18                         int cnt = targetDataLine.read(tempBuffer, 0, tempBuffer.length);
19                         if(cnt > 0){
20                             DatagramPacket outPacket = new DatagramPacket(tempBuffer, tempBuffer.length, this.ip, this.port);
21                             this.socket.send(outPacket);
22                         }
23                     }
24                 }catch (Exception e) {}
25
26             } catch (Exception e) {}
27         }
28     }
```

Code Snippet 5: Audio1 Class

1.1.3 TCP

TCP guarantees the sending of packages and if a package is not sent, we will be notified. We have used TCP for the transmission of message in our application. Based on John T. Hagen; Barry E. Mullins (2013). "TCP veto: A novel network attack and its application to SCADA protocols", there will be an acknowledgement if the message is sent and received. As the message is crucial for the communication between the client and agent, TCP will be needed to ensure that the message is sent.

```
18    //connect to server
19    public void run(){
20       try{
21          Socket socket = new Socket(ip, port);
22
23          //sends message
24          socket.getOutputStream().write(message.getBytes());
25          socket.close();
26
27       }catch(IOException e){
28          e.printStackTrace();
29       }
30    }
31
32 }
```

Code Snippet 6: Client Class

```
26    public void run(){
27
28       Socket client; //connect to the port
29
30       try{
31          //establish and keep on accepting connection until terminate
32          while((client = serverSocket.accept()) != null){
33
34                //get String from socket
35                BufferedReader read = new BufferedReader(new InputStreamReader(client.getInputStream()));
36
37                String message = read.readLine();
38                      write.writeGUI1("" +message);
39             }
40
41       }catch(IOException e){
42          e.printStackTrace();
43       }
44    }
```

Code Snippet 7: Client Server Class

```
4  public class agent extends Thread{
5     String message; //what to send
6     String host = "localhost"; // where to sent it, ipAddress
7     int port; //what port to send message to
8
9     public agent(String message, int port){
10       this.message = message;
11       this.port = port;
12    }
13
14    public void run(){
15       try{
16          //connects to the ServerSocket
17          Socket socket = new Socket(host,port);
18
19          //sends message to client as bytes
20          socket.getOutputStream().write(message.getBytes());
21          socket.close();
22
23       }catch(IOException e){
24          e.printStackTrace();
25       }
26    }
```

Code Snippet 8: Agent Class

```
26    public void run(){
27       Socket agent;
28       try{
29          while((agent = agentSocket.accept()) != null){
30 //          InputStream input = agent.getInputStream();
31             BufferedReader read = new BufferedReader(new InputStreamReader(agent.getInputStream()));
32
33             String message = read.readLine();
34             if(read!= null){
35                write.writeGUI1("Agent: " + message);
36             }
37          }
38
39       }catch(IOException e){
40          e.printStackTrace();
41       }
42    }
43    public agentServer(){
44       try{
45          agentSocket = new ServerSocket(port);
46       }catch(IOException e){
47          e.printStackTrace();
48       }
49    }
```

Code Snippet 9: Agent Server Class

1.2 System Implementation

1.2.1 User Authentication

Our chat system uses login authentication process, whereby the agent and customer will have to enter their valid username and password to enter the chat system. The system will crosscheck with the saved credential to make sure that the login details are valid for both the customer and agent respectively. If the user inputs an invalid username or password, the system will output a pop-up message informing the user about it. Cases of such that can occur:

- Wrong username entered
- Wrong password entered
- Wrong login type selected

```
int flag = 1;

        if(username.equals(strUsernameA)){
                            if(password.equals(strPasswordA)){
            flag = 0;
          }
        }
                    else if(username.equals(strUsernameA1)){
          if(password.equals(strPasswordA1)){
            flag = 0;
          }
        }
                    else if(username.equals(strUsernameA2)){
          if(password.equals(strPasswordA2)){
            flag = 0;
          }
        }
                    else if(username.equals(strUsernameA3)){
          if(password.equals(strPasswordA3)){
            flag = 0;
          }
        }

      return flag;
```

Code Snippet 10: Authentication in account class

```
public void loginAsAgent() throws IOException{
   System.out.println(account.readFromAgentFile(tfUsername.getText(), (new String(tfPassword.getPassword()))));
   if (account.readFromAgentFile(tfUsername.getText(), (new String(tfPassword.getPassword()))) == 0){
      dispose();
      new agentGUI(tfUsername.getText());
                  JOptionPane.showMessageDialog(null, "Login successful!");
   }else{
      JOptionPane.showMessageDialog(null, "Invalid username or password. Please try again");
      tfUsername.setText("");
      tfPassword.setText("");
      count++;
      if(count == 3){
         JOptionPane.showMessageDialog(null, "No more login attempts.\nSystem will now terminate!");
         System.exit(0);
         dispose();
      }
   }
}
```

Code Snippet 11: Authentication in loginGUI class

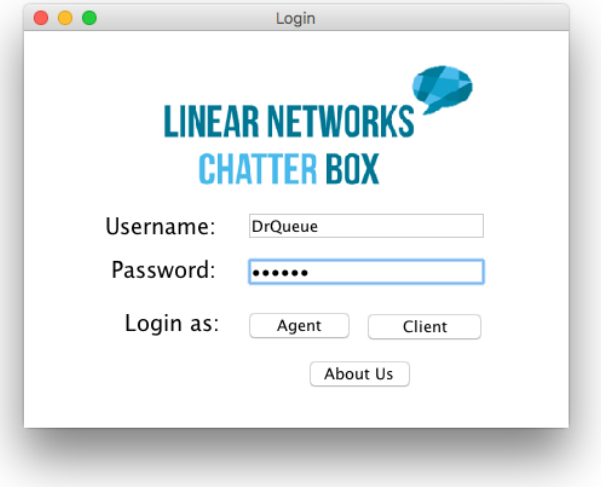

Figure 1: Login Interface with password encryption

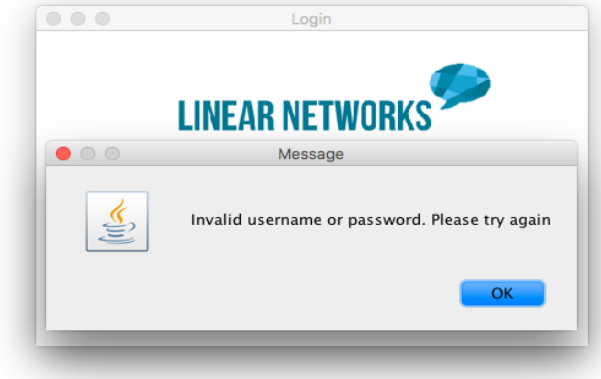

Figure 2: Pop-up box message showing invalid username/password

1.1.2 One-to-One Interaction

In this interaction, the customer is only allowed to communicate with one person which is the agent. The customer will not be able to communicate with more than one agent at a time.

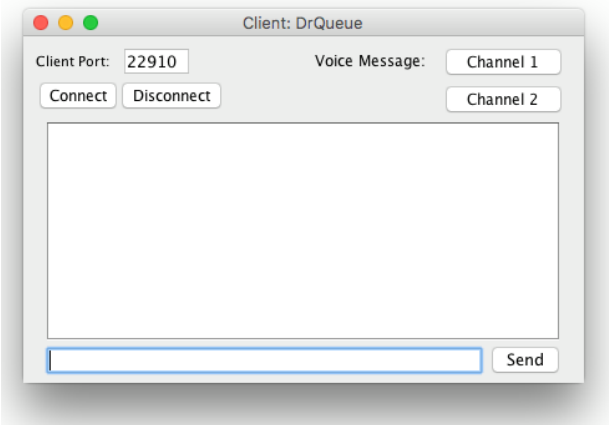

Figure 3: Customer's Chat Box

1.1.3 Many-to-One Interaction

This interaction can be seen through the agent's chat interface. The agent will be able to send a message to two customers at the same time.

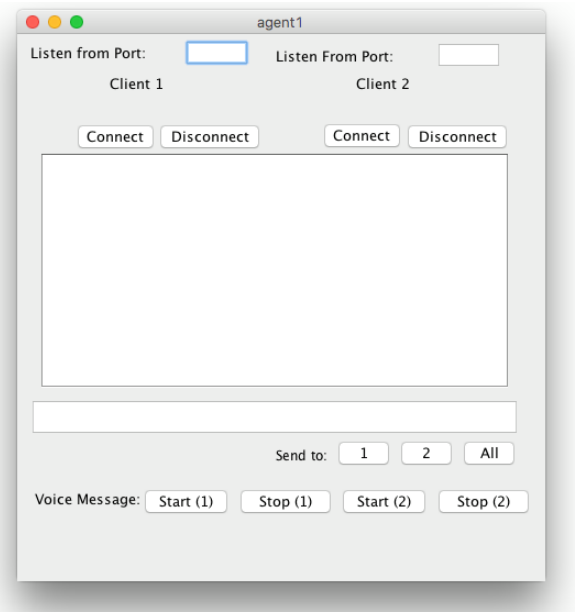

Figure 4: Agent's Chat Box

PART II

2.1 Benefits of Cloud Computing

2.1.1 Maintainability

With cloud computing, we would not need to worry about maintenance of the system as the cloud computing company will be responsible for any sort of maintenance work. Because we do not virtually own any hardware, our team can focus on maintaining applications which are related to our software besides the codes and actual program itself. Cloud computing also gives us the ability to expand our application as the user base grows without having to upgrade any hardware, software, or bandwidth manually.

2.1.2 Cost savings

In terms of production and maintenance, cloud computing allows us to save cost by not physically owning any hardware system. We do not have to purchase any extra storage (i.e. hard disk) to store our database or chat logs, nor do we need extra processor strength (RAM) to host the server. All the required things can be obtained from the cloud server. As our application deals with real time voice chat and sending messages, it will require a heavy load of storage to store all the conversations between the users. Just by paying a certain amount of fees to our service provider, we can determine how much storage we want and adjust our cost according to that need. As most things are pay-as-you-use with cloud services, this reduces the need to pay extra for redundant items that we do not need.

2.1.3 Accessibility

It is important to ensure that our chat system is available 24/7 to cater towards user's needs whenever they need it. Deploying our application on a cloud computing makes our system can run at anywhere and anytime. We do not need to keep our computers up all the time and waste our own resources to keep the application running. As our chat system is like other messaging applications like WhatsApp and Facebook Messenger, it is only sensible for us to keep the system up 24/7 so that a client would be able to communicate with an agent at any time.

2.1.4 Usability

Usually cloud services are easy to use. The whole idea of cloud services is that everything is accessible with just having an internet connection. Amazon Web Services allow users to integrate the development kit into IDEs like Eclipse for ease of deployment while providing the developers the comfort of coding with their preferred IDE. As we are developing our chat system using Java and Eclipse, this is a very sensible choice for us because it allows for easy integration with the cloud computing services and eventually easy deployment of our cloud application.

2.1.5 Security

Most cloud services nowadays provide high security. By developing our cloud application using cloud platforms, we can guarantee to have a better security than having a security of our own. This is because many cloud services have regular and robust backups for the servers, therefore we can restore our data fast enough to have our application running smoothly again. In contrast to that, if we were to run our application using our own computers as servers, there could be a major problem if any of our computers is hacked or attacked by virus, there will be no online backup for restoring our client's data. Thus, making cloud computing services one of the advantages to be used by our application.

2.2 Platform as a Service (PaaS)

A Platform as a Service (PaaS) provides a platform on which software can be developed and deployed. The chosen PaaS for our cloud application is the Amazon Web Services (AWS). As we have developed our chat application solely using Eclipse as our development engine, AWS offers compatible tool that will allow us to integrate AWS's development platform into Eclipse.

Therefore, AWS seems like the sensible choice to be used as our PaaS. The platform that AWS provides to support Java applications development is called the AWS Elastic Beanstalk and to use this platform on Eclipse, there is a toolkit called the AWS Toolkit for Eclipse which acts as a plugin that allow developers who uses Eclipse as their IDE to manage AWS resources, including Elastic Beanstalk applications and its environments, all from the comfort of within the Eclipse IDE.

However, the AWS Toolkit for Eclipse only supports projects that uses Java with Tomcat platform, not the Java SE platform which is the default for Eclipse whenever it is downloaded. In the bright side, Apache Tomcat can be easily installed into Eclipse for this to work out perfectly. Therefore, as aforementioned, AWS seems like a right fit for our application because it is very compatible with Eclipse as they provide various integration alternatives for creating and deploying Java applications using their platform.

Reference for Java Applications Development using Amazon Web Development Elastic Beanstalk ("Creating and Deploying Java Applications On AWS Elastic Beanstalk - AWS Elastic Beanstalk", 2017)

Reference for Apache Tomcat which is the alternative solution to run the AWS Toolkit on Java SE platform. (Kathy Chan, 2005. "Installing Apache Tomcat Server")

2.3 System Architecture

Our system architecture is based off a hybrid model. Our chat subsystem is the type of a peer-to-peer model which is also depicted as a client/server-to-client/server model where each user of the application act as a client and a server at the same time, whereas our audio chat subsystem is off a client-server model, where a single main audio server does all audio handling.

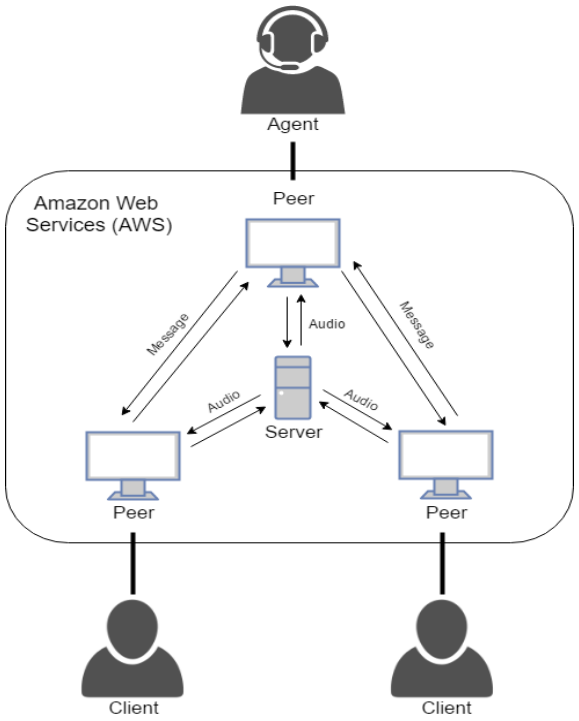

Figure 5: System Architecture

As seen from the figure, one agent will be connected to two clients at one time. After a user (can be either agent or client) login, the system will then create a server specifically for this user to handle the text transmission and reception for him/her through the TCP protocol. This part is the peer-to-peer configuration of our system, where each user acts as a client and server itself. If any user wants to communicate through audio, they will transmit and receive their audio packets through the UDP protocol which is handled independently with an audio server, which is separated from the chat servers. This is where our client-server configuration comes in place, where user acts as the client and the audio server acts as the server to transmit the audio packages.

Our application is like Skype which uses a hybrid model as well. Whenever a user log into their Skype account, this requires a central server to validate this user login procedure, this represents the client-server architecture. Once the user successfully logs into their account, their own device will act as a server itself because their Skype application is running on that person's own machine and uses its resources to carry out the user's requests, this is the peer-to-peer side of Skype application.

The above findings are referenced from (*What is Hybrid Routing Protocol (HRP)? - Definition from Techopedia - https://www.techopedia.com/definition/26311/hybrid-routing-protocol-hrp)*

2.4 Database Support

The chat logs with the agent and clients can from our application can be stored in a database. As we are using the AWS Elastic Beanstalk service, we can also integrate the Amazon Relational Database System (RDS). This will allow us to seamlessly store data gathered and modified by our application. The database can be attached to the application's environment and managed by the Elastic Beanstalk, or created and managed externally.

2.4.1 Implementation

These steps need to be followed to implement the relational database into the applications' environment:

1. Open Elastic Beanstalk console and navigate to the management the console of your environment.

2. From the management console, select Configuration.

3. Under Data Tier, choose Create a new RDS database.

4. Choose a database engine, enter a username and password, and then choose Apply.

Adding a database instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to the application through the following environment properties:

- RDS_HOSTNAME – The hostname of the DB instance.
- RDS_DB_NAME – The database name, ebdb.
- RDS_USERNAME – The username that is configured for the database.
- RDS_PASSWORD – The password that is configured for the database.
- RDS_PORT – The port on which the DB instance accepts connections. The default value varies between DB engines.

To connect to the database, we add the appropriate driver JAR to the application, load the driver class in the code, and create a connection object with the environment properties provided by Elastic Beanstalk. The driver that has to be installed is specific to the DB engine that has been chosen.

2.4.2 Configuring the database

AWS Elastic Beanstalk provides the option of running a DB instance that is integrated with the application environment but terminates when the application itself is terminated. This is unsuitable for this type of application, so we can use the alternative option provided, that is to run an independent database instance on the Amazon RDS and configure our application to connect to it on launch.

After launching the database instance and configuring security groups, we can pass the connection information (endpoint, password, etc.) to the application by using environment

properties, the same mechanism that Elastic Beanstalk uses when running a database instance in the application environment.

To launch an RDS DB instance in a default virtual private cloud,

1. Open the RDS management console.

2. Choose Launch DB Instance.

3. Choose a DB Engine and pre-set configuration.

4. Under Instance Specifications, choose a DB Instance Class. For high availability, set Multi-AZ Deployment to Yes.

5. Under Settings, type a Master Username and Master Password and note the values entered for later.

6. Choose Next.

7. For Network and Security settings, choose the following:

    - VPC – Default VPC
    - Subnet Group – default
    - Publicly Accessible – No
    - Availability Zone – No Preference
    - VPC Security Groups – Create new Security Group

8. For Database Name, type ebdb, and verify the default settings for the remaining options. Note the values of the following options:

    - Database Name
    - Database Port

9. Choose Launch DB Instance.

The next step is to modify the security group attached to the DB instance to allow inbound traffic on the appropriate port. This is the same security group that we will attach to our Elastic Beanstalk environment later, so the rule that we add will grant ingress permission to other resources in the same security group.

To configure environment properties,

1. Open the Elastic Beanstalk console.

2. Navigate to the management console for your environment.

3. Choose Configuration.

4. In the Software Configuration section, choose the gear icon .

5. In the Environment Properties section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following:

    - RDS_DB_NAME – The DB Name shown in the Amazon RDS console.
    - RDS_HOSTNAME – The Endpoint of the DB instance shown in the Amazon RDS console.
    - RDS_PORT – The Port shown in the Amazon RDS console.
    - RDS_USERNAME – The Master Username that you entered when you added the database to your environment.
    - RDS_PASSWORD – The Master Password that you entered when you added the database to your environment.

6. Choose the plus symbol (+) to add additional properties:

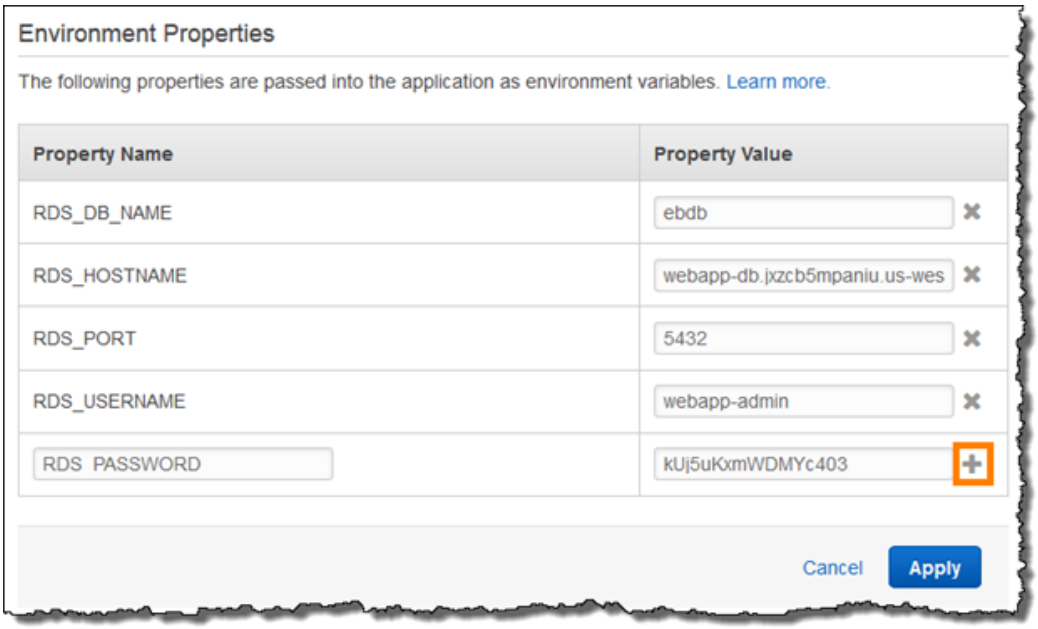

Figure 6: Configuring environment properties

7. Choose Apply.

The configuring database steps were refered from "Configuring Databases with Elastic Beanstalk - AWS Elastic Beanstalk", 2017.

2.4.3 Usage

The Amazon RDS will be used to create a database that can be managed and modified separately as an independent entity. For the purpose of our application, we will be using the MySQL DB engine and it can be accessed through the MySQL client after initiating the connection to the cloud based database. Before that, the database connection has to be initiated through the usage of the following class in Java:

```
private static Connection getRemoteConnection() {
   if (System.getenv("RDS_HOSTNAME") != null) {
   try {
      Class.forName("org.mysql.Driver");
   String dbName = System.getenv("RDS_DB_NAME");
   String userName = System.getenv("RDS_USERNAME");
   String password = System.getenv("RDS_PASSWORD");
   String hostname = System.getenv("RDS_HOSTNAME");
   String port = System.getenv("RDS_PORT");
   String jdbcUrl = "jdbc:mysql://" + hostname + ":" + port + "/" + dbName + "?user=" + userName + "&password=" + password;
   logger.trace("Getting remote connection with connection string from environment variables.");
   Connection con = DriverManager.getConnection(jdbcUrl);
   logger.info("Remote connection successful.");
   return con;
   }
   catch (ClassNotFoundException e) { logger.warn(e.toString());}
   catch (SQLException e) { logger.warn(e.toString());}
   }
return null;
  }
```

Code Snippet 12: Snippet of code that creates a connection to a PostgreSQL database.

After the database has been created, the following data along with their respective data types can be stored for future reference from the application:

Table 1: Data to be stored from the application

DATA TITLE	DATA TYPE
Client Name	VARCHAR(20)
Agent Name	VARCHAR(20)
Client IP Address	VARCHAR(10)
Agent IP Address	VARCHAR(10)
Date	DATE

Apart from the above, the database can be used to load the chat log files that have been stored as text(.txt) files in the file system and be retrieved at the management user's' discretion.

2.5 Deployment

For our application, we will be using Amazon Web Services (AWS) Toolkit for Eclipse to host our application. The steps to developing and deploying our Java application are as the below:

1. The first step is to develop the application. Our team has used Eclipse Java IDE to build and debug our application.

2. The next step would be to Install Apache Tomcat for Eclipse as the AWS Toolkit for Eclipse only supports projects that use the Java with Tomcat platform as written on their website based on *"Using The AWS Toolkit For Eclipse - AWS Elastic Beanstalk", 2017*.

3. AWS Toolkit for Eclipse will then be needed to be installed so that we can develop our Java application in an AWS environment. The AWS Toolkit for Eclipse integrates AWS Elastic Beanstalk management features with our Tomcat deployment environment to facilitate environment creation, configuration, and code deployment.

4. Once the application development is complete, using Eclipse together with the AWS Toolkit, we will then deploy our application to AWS Elastic Beanstalk where we can manage our applications in the AWS Cloud.

2.6 Conclusion

Through the duration of this assignment, our group has researched and learned more about the benefits of using cloud computing for our application. We also got to know more about the concepts and the ways cloud computing can be implemented into our application. It has also opened our eyes to see how advanced technology is today and how we can use this technology to improve and advance our applications to serve us better. The cloud computing services we use daily like Google Docs has given us the ability to work in a collaborative environment with one another without being physically present. This has led to the dramatic rise in popularity of cloud computing. The security risks initially associated with handling confidential data has been debunked since long due to the creation of advanced and sophisticated security measures and boundaries that have been kept safeguarding the data on the cloud servers.

2.6.1 Future Recommendations

For future expansion, our application can introduce login with Social Media accounts. This will bring convenience to the client as they need not register for a new account, but use their current social media account like Facebook or G-Mail to connect with our chat application instead. Another recommendation is to provide video calling features in our application. This will give a more interactive approach for the client when communicating with the agent, giving them the feeling close to an actual human interaction. Lastly, rating systems can also be implemented in our application. This can provide great feedback to the agents and if the chat or conversation went well, clients can provide their feedback through rating their chat session with the agent upon its completion.

REFERENCES

Amazon Web Services, Inc. 2017. What is AWS Elastic Beanstalk?. Retrieved from http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html

Configuring Databases with Elastic Beanstalk. (n.d.). Retrieved June 19, 2017, from http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html

Emilia Mattila. *An Analysis of Hybrid and Pure Peer-to-Peer Technologies for IP Telephony.* Retrieved from http://www.tml.tkk.fi/Publications/C/18/mattila.pdf

Kathy Chan. April 29, 2005. *Installing Apache Tomcat Server.* Retrieved from http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/InstallTomcat/InstallTomcat.html

Using Elastic Beanstalk with Amazon RDS. (n.d.). Retrieved June 16, 2017, from http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/AWSHowTo.RDS.html#rds-external-credentials

Vítor de Albuquerque Torreão. *Comparison between client-server, peer-to-peer and hybrid architectures for MMOGs.* Retrieved from http://mms.ecs.soton.ac.uk/2014/papers/3.pdf