**Appendices Q**

**Security**

As explained in the report, for this development I decided to increase the security of the application I am developing by carrying out three improvements to the system which are listed below:

* Proxy Server – User Privileges Functionality
* Login Functionality
* Forgot Password Functionality

1. **Login Functionality**

For me to implement login functionality, I decided to create a Login interface which declares two methods called, login and resetPassword(), of which I will explain the resetPassword() method later in this appendices. I then created a LoginImpl class that provides an implementation of both methods.

As you can see from Fig. 1, the LoginImpl class extends UnicastRemoteObject (enabling a LoginImpl object to be passed across the network using RMI, outlined in Iteration Cycle 3), and implements LoginInterface outlined in the appendices. Furthermore, the LoginImpl class declares a Server object (outlined in Iteration Cycle 3), which is then initiated by a parameter value supplied in the constructor.

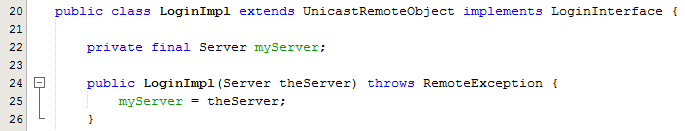


Fig. 1 – Extract from LoginImpl class

I then have to provide an implementation for each of the two methods declared within the LoginInterface interface, however I am going to go through the resetPassword implementation in the next section of this appendices. As you can see from Fig. 2, a username and password combination is passed to the login() method for the LoginImpl object, where the LoginImpl object checks to see if the Server object is not null and that the server is alive, if so then the Server.isUser() method is invoked and supplied to an if statement condition, and if the isUser method returns true, then a ServerProxy object is initialised and returned as a Server object to the Client that is invoking this method.

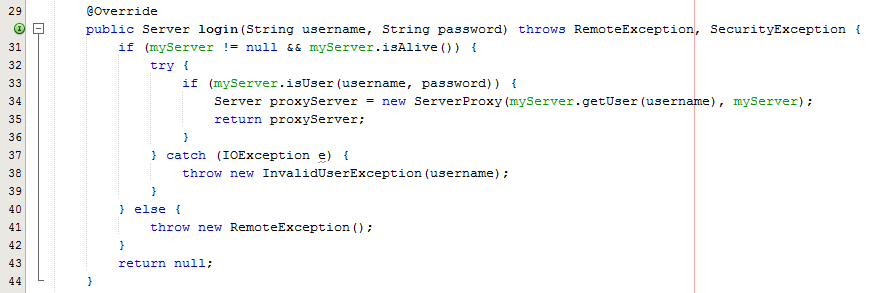


Fig. 2 – Extract from LoginImpl class – login()

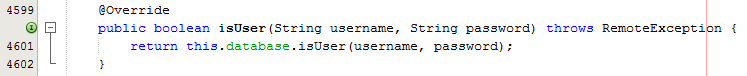


Fig. 3 – Extract from ServerImpl – isUser()

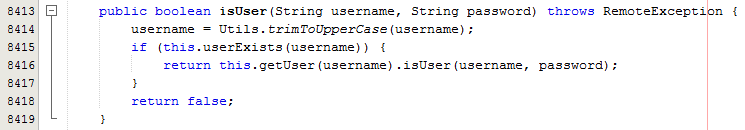


Fig. 4 – Extract from Database class – isUser()

As you can see from Fig. 3 and Fig. 4, when the LoginImpl class invokes isUser() on the ServerImpl object, the server object invokes isUser() on the Database object (outlined in Iteration cycle 2). The Database object, then invokes Utils.trimToUpperCase() and passes String username supplied as a parameter, to the method as a parameter, in which the Utils class will trim the username, removing any starting or leading spaces and converts the String to upper case, then invokes userExists() and supplies the amended String username as parameter, which checks the Database object for a user with matching username, and returns true if found and false if not. The Boolean return value is then supplied to an if statement, where the Database object gets the User, and invokes isUser() on the user object, and supplies the username and password combination, if the username and password match the user then true is return else false is returned.

The Boolean value is then passed back to the server object, and then back to the login object, where control is then passed back to the LoginImpl object, where a ServerProxy object is either created and passed back to the client or not, depending on if login was successful.

Finally, although I have shown how the LoginImpl class gives users access to a ServerProxy (going to be explained later) or denies access to the system, I am now going to explain how this is implemented within the current RMI networking setup (outlined in Iteration Cycle 3).

Firstly, as shown in Fig. 5, instead of passing the ServerImpl object (serverStub) as the parameter to the rebind method, I supply a newly created LoginImpl object, which has the serverStub passed as a constructor parameter, (as explained earlier, the ServerImpl object is then assigned to a Server object within the LoginImp class), this means that when Clients attempts to look up the server on the RMI registry, the returned object is a LoginInterface object, (as explained earlier, declares two methods which are implemented by the LoginImpl class), and allows the client to then invoke either the login or the resetPassword method, in this instance, it is the login() object that the client would want to invoke to login which returns a newly created ServerPoxy object, as shown in Fig. 6.

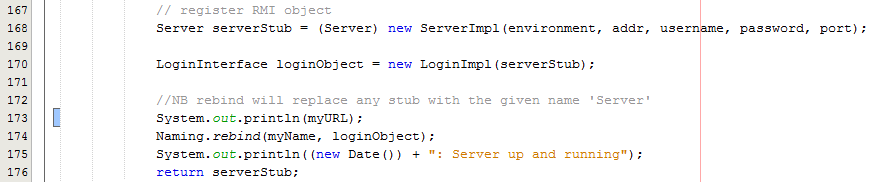


Fig. 5 – Extract from ServerImpl class

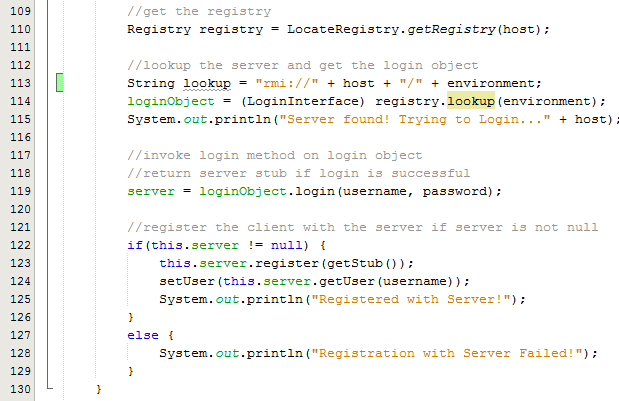


Fig. 6 – Extract from ClientImpl class

So as shown in Fig. 6, once the RMI registry for the Server has been located on the client side and lookup() is invoked on the registry then instead of the Server object being returned, a LoginInterface object is returned, allowing the client to invoke the login() method and supply the clients username and password combination supplied by the user, which will return a new ServerProxy object if the user details are correct as shown in Fig. 2, which will allow the client to then invoke any methods on the ServerProxy object to communicate with the ServerImpl object, providing the user privileges are correct, and will be explained in more detail later in the appendices, otherwise null will be returned preventing the user to accessing the server.

Once the client has the ServerProxy object it then invokes the register method, which it would normally do under the previous implementation which registers its ClientImpl object with the ServerImpl object, however it is just invoked on the ServerProxy object instead of the ServerImpl object, where the ServerProxy then deals with the server request as shown later in the appendices.

1. **Forgot Password Functionality**

Now the user can be denied access to the application if they do not have the correct username and password combination, I am now going to explain how I implemented the forgot password functionality.

For this I decided to make use of the simple mail transfer protocol (SMTP) and implement send mail functionality that will send an email to the registered email address for the user provided the user with an amended password, allowing the user to log back in to the system. Additionally, if the user has not set up the information to allow them to use the forgot password functionality then an employee of MSc Properties, with the correct user privileges, is able to amend the password of a user.

* 1. **Email Reset**

Firstly, to implement the reset password functionality through an email, I provided an implementation for the resetPassword method in LoginImpl class. As you can see from Fig. 7, the resetPassword method takes 4 parameter values, a String username, a String email, integer empRef and String answer (information assigned to the user, along with a memorable answer defined by the user), I then do the same checks as with the login method, and supply the parameter values to Server.forgotPassword() method.

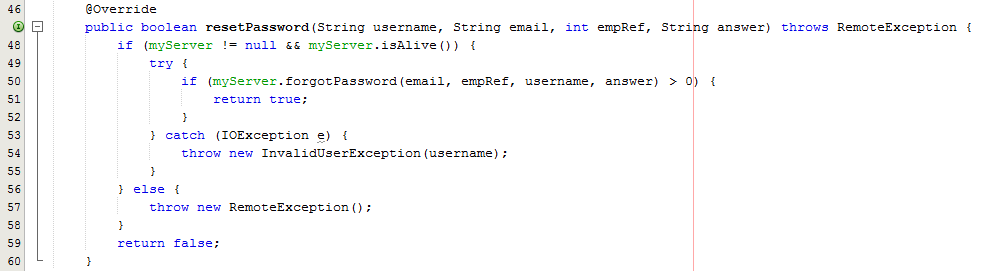


Fig. 7 – Extract from LoginImpl class – resetPassword()

As you can see from Fig. 8, I check to see if the empRef, supplied as a parameter is a valid employee ref, if so I then get the Employee object from the Database and compare the employee username and the username supplied as parameter, if this matches, I then compare the employee memorable location with the memorable location supplied as parameter, if this matches, I then compare the employee email address with the email address supplied as parameter, lastly if this matches, I create a new random String value using the BigInteger class and SecureRandom class, and invoke a local method to updateEmployeePassword, supplying the new String as the password along with the employee ref and username. I then update the employee password reset field (Boolean value), ensuring that the user has to reset the password when they next log in. I then create the content for the reset password email and supply as parameter to the SendEmail.sendEmail() method.

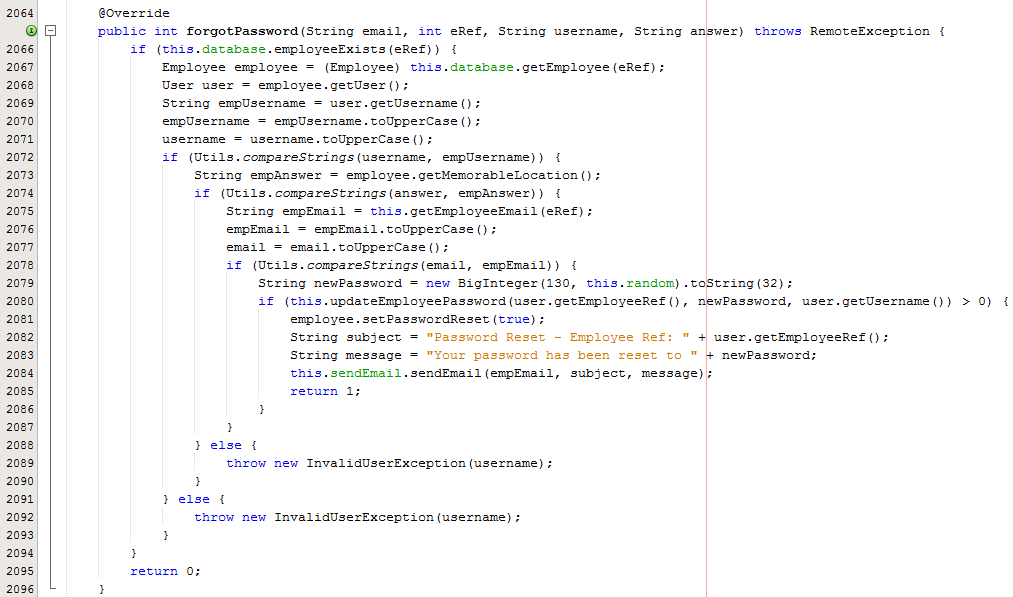


Fig. 8 – Extract from ServerImpl class – forgotPassword()

As shown above in Fig. 8, I invoke a method sendEmail() from a Send Email class, which uses the javax.mail package to make use of the JavaMail API to send an email to the user that has forgot their password. As explained, I create a SendEmail class, which declares a String from, String password, String host and Properties properties field, and takes three String values as constructor parameters, as shown in Fig. 9.

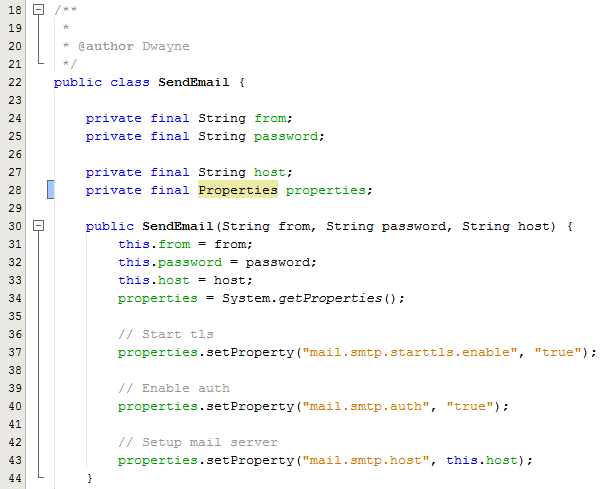


Fig. 9 – Extract from SendEmail – SendEmail field declaration and constructor

As you can see from Fig. 9, the three String values passed as constructor parameters are assigned to the from field, password field and host field, and contains the from email address, the password for that email address and the SMTP host respectively. I then assign the properties field the returning properties object, from invoking System.getProperties(). Now I have initialized the field values, I can then set the Properties object properties required to send an email. First I need to set a property which invokes the STARTTLS command, which upgrades a plain text connection, to an encrypted TLS connection, enabling my application to securely connect to a user’s email address, I then set the auth property to true, as I have got the username and password combination and am therefore going to perform password authentication when sending an email, Lastly I set the SMPT host, to the host set as a String field.

Now I have set up the System Properties to enable me to send an email, I then define a public method which allows clients of the SendEmail object to provide an email receiver email address, the email subject and email message as parameters, allowing the SendEmail object to compose a message from the parameters supplied, and send using the object properties already defined in the constructor.



Fig. 10 – Extract from SendEmail class – sendEmail()

As shown in Fig, 10, firstly, I define a Session object and initialize it with the return value of the Session.getInstance() method, where I pass the properties object and a new Authenticator object (passing the username and password as parameter to a new PasswordAuthentication instance) to, start up a new Session enabling me to connect to the host I am going to send my email from. I then create a MimeMessage object assigning the session object as parameter, and then assign the message objects settings for the from email address (converted to an internet email address from the String passed as parameter, using the InternetAddress class), the recipient, subject and message text. I then invoke Transport.send() and pass the MimeMessage object as parameter, which allows me to connect to the transport protocols defined in the message (through the Properties object defined in the constructor).

Now I have defined the class which will enable me to send emails using the SMTP, by defining a SendEmail object, and passing the from email address, email account password, and email host as parameters to the initialization of the object, and then the client of the SendMail object is then able to invoke the sendEmail object, and just provide the recipient email address, the subject, and message to be sent, allowing the client to send an email. The implementation for the SendEmail object within the ServerImpl class is shown within Fig. 8 (invoking sendEmail() method), Fig. 11 and Fig. 12.



Fig. 11 – Extract from ServerImpl class – declaration of SendEmail object



Fig. 12 – Extract from ServerImpl class – initialization of SendEmail object

* 1. **Employee Reset**

The Employee Reset functionality, will allow an employee who has the correct user privileges to update another user’s password, to allow the password to be passed to the user enabling them to log in to the system, and update their password to something different. To do this I implemented an updateEmployeePassword method, which checks to see if the employee in which the user wants to amend exists, gets the Employee object from the database, and then invoke updatePassword() method on the employee object supplying the new password as parameter, and a new ModifiedBy object, as shown in Fig. 13.

I then check to see if the employee updating the password is the same as the employee being updated (As this updatePassword can be used by an employee updating their own password), if the employees are different the employeeReset field within the employee object is updated to true, making the employee have to amend the password when they next log in next, ensuring that the user account is still secure, and no one knows the password to the account once the user has logged in and amending their own password. Finally I update the database by invoking updateUser() and updateEmployee() on the database object, ensuring that the database is up to date at all times.

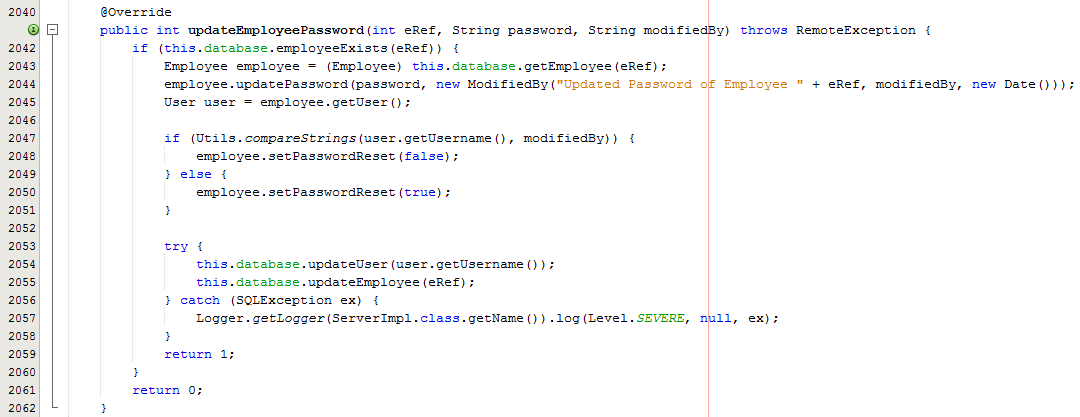


Fig. 13 – Extract from ServerImpl – updateEmployeePassword

1. **Proxy Server – User Privileges Functionality**

As explained in the report, I decided to improve security of the system, by implementing a Proxy Server which will act as a communicator between the server and client, preventing any unauthorised method calls from a client to be invoked on the server.

Firstly, to implement the Proxy Server, I created a class which similarly to ServerImpl class (outlined in Iteration Cycle 3) implements Server and extends UnicastRemoteObject, which means that I need to provide an implementation for each of the methods defined in Server interface. Additionally, I have defined a Server object and a User object as shown in Fig. 14, which are initialised by constructor parameters.

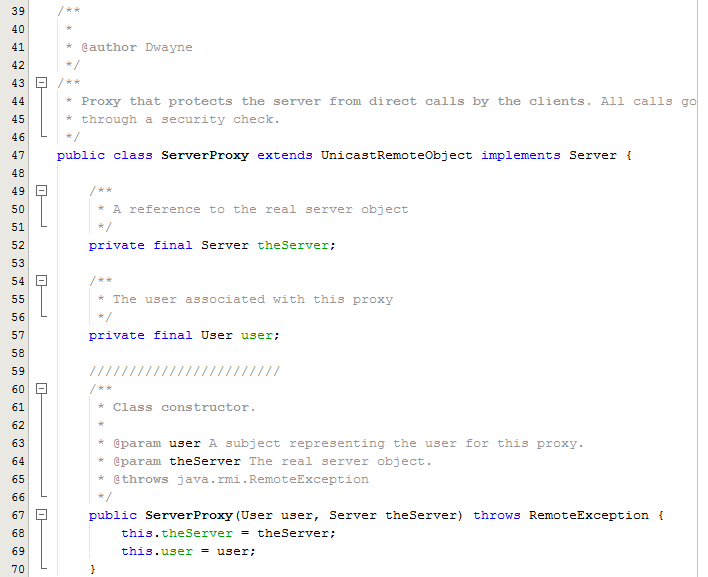


Fig. 14 – Extract from ServerProxy class

As explained previously, I have amended the RMI setup within the ServerImpl class, to bind a LoginImpl object to the registry, instead of a ServerImpl object, and once a client successfully logs in to the system, using the login() method from the Login object that bound to the registry, a ServerProxy object is created, which is passed the ServerImpl serverStub, along with the User object of the employee that has logged in.

Now the user has a serverStub object in the form of a ServerProxy, and the ServerProxy implements all methods that the Server interface declares to the client, the client can now invoke any method on the ServerImpl object, through the ServerProxy object, meaning that I can now control what users invoke what methods, stopping the client from invoking methods they do not have permission to invoke.

For the user privileges I decided to declare 8 Boolean fields within the User class, which manage the create, read, update and delete privileges for both normal data and employee data, and is assigned to an employee’s user account when a contract object is created for them, outlining their user privileges, which is shown in Fig. 15.

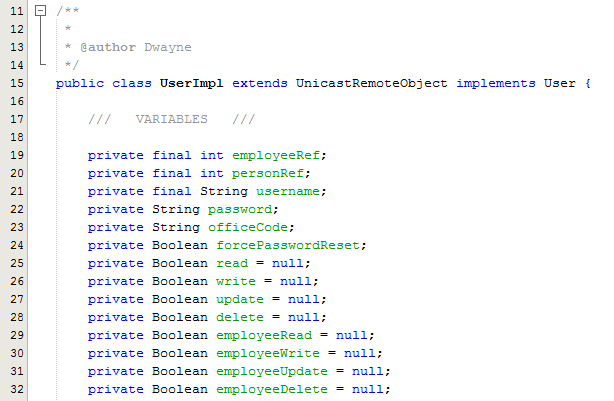


Fig. 15 – Extract from UserImpl class

So the User object will have the assigned privileges, and as shown in Fig. 6, when the client successfully logs in, the User object is assigned to the Client object, through the local setUser method, ensuring that whenever a client invokes a method the user privileges is available through the UserImpl object.

Now I have showed how the client is assigned an instance of a ServerProxy object through the LoginImpl class, I am now going to show you how I have implemented the security for each of the methods, although due to the large number of methods and the similar nature of each method, I am just going to explain one of the methods and outline the concept I have implemented.

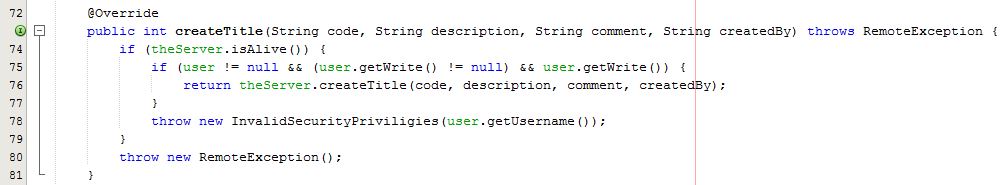


Fig. 16 – Extract from ServerProxy class

As shown in Fig. 16, the createTitle() method within the ServerProxy class checks to see if the ServerImpl object is still alive, and then I check to see if the user is initialised, and the write field within the User object is initialised, if so I then invoke getWrite() on the User object, which returns true if the user is able to write normal data, otherwise false. This enables the ServerProxy to determine if the user has the user privileges to invoke the createTitle method, and if so, then invokes the createTitle method on the ServerImplm object and returns the ServerImpl response otherwise throws the necessary exception.

From the client perspective they are still just interacting with a Server interface however, the Server has changed from being implemented by a SeverImpl object to being implemented by a ServerProxy object, and the ServerProxy manages the communication with the ServerImpl object stopping any unauthorised user access.

I follow this implementation with all of the methods outlined in the ServerProxy, substituting the create, read, update, delete, employeeCreate, employeeRead, employeeUpdate and employeeDelete user privileges for each of the different methods depending on the required user privilege, ensuring that users are unable to invoke a method on the ServerImpl object unless they have the required user privileges.