Djillali Liabes University of Sidi Bel Abbes
Faculty of Exact Sciences
Department of Computer Sciences

Module : Réseaux et Systèmes Répartis
1st year of Master's degeree in
Networks,Information Systems & Security(RSSI)
2021/2022

Application java RMI pour une gestion simplifiée d'un dictionnaire

Students:
Hadjazi M.Hisham
AMOUR Wassim Malik
Group: 01/RSSI

Instructors:
Dr. MEHADJI Djamil

A paper submitted in fulfilment of the requirements for the
TP-03

April 21, 2022

Contents

Chapter 1

Application java RMI pour une gestion simplifiée d'un dictionnaire

1.1 Introduction

Java Remote Method Invocation (Java RMI) enables you to create distributed Java technology-based applications that can communicate with other such applications. Methods of remote Java objects can be run from other Java virtual machines (JVMs), possibly on different hosts.

RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting object-oriented polymorphism. The RMI registry is a lookup service for ports.

1.1.1 The RMI implementation

Java Remote Method Invocation (RMI) provides a simple mechanism for distributed Java programming. The RMI implementation consists of three abstraction layers.

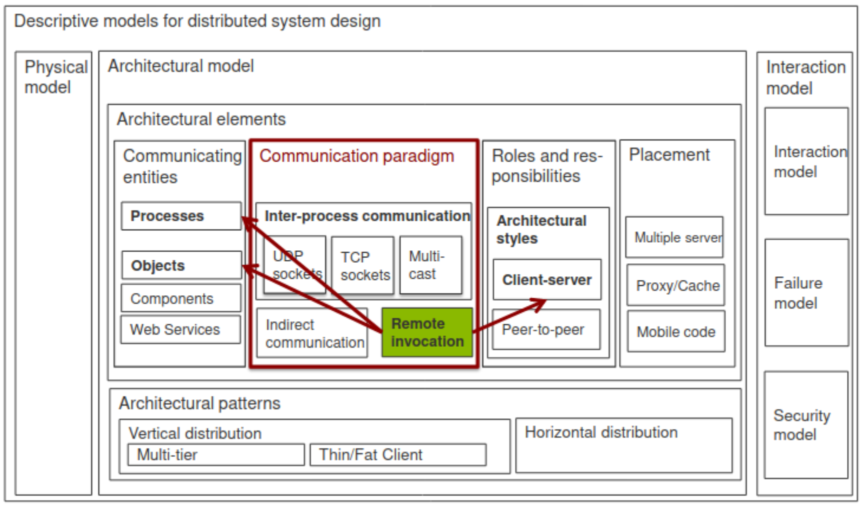


FIGURE 1.1: RMI.

These abstraction layers are:

1. The Stub and Skeleton layer, which intercepts method calls made by the client to the interface reference variable and redirects these calls to a remote RMI service.

2. The Remote Reference layer understands how to interpret and manage references made from clients to the remote service objects.

3. The bottom layer is the Transport layer, which is based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

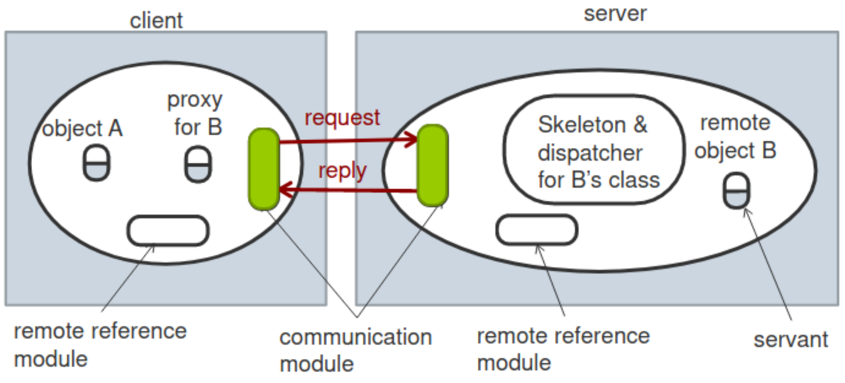


FIGURE 1.2: RMI Implementation.

On top of the TCP/IP layer, RMI uses a wire-level protocol called Java Remote Method Protocol (JRMP), which works like this:

1. Objects that require remote behavior should extend the RemoteObject class, typically through the UnicastRemoteObject subclass.
   (a) The UnicastRemoteObject subclass exports the remote object to make it available for servicing incoming RMI calls.
   (b) Exporting the remote object creates a new server socket, which is bound to a port number.
   (c) A thread is also created that listens for connections on that socket. The server is registered with a registry.
   (d) A client obtains details of connecting to the server from the registry.
   (e) Using the information from the registry, which includes the hostname and the port details of the server's listening socket, the client connects to the server.

2. When the client issues a remote method invocation to the server, it creates a TCPConnection object, which opens a socket to the server on the port specified and sends the RMI header information and the marshalled arguments through this connection using the StreamRemoteCall class.

3. On the server side:
   (a) When a client connects to the server socket, a new thread is assigned to deal with the incoming call. The original thread can continue listening to the original socket so that additional calls from other clients can be made.

   (b) The server reads the header information and creates a RemoteCall object of its own to deal with unmarshalling the RMI arguments from the socket.
   (c) The serviceCall() method of the Transport class services the incoming call by dispatching it.
   (d) The dispatch() method calls the appropriate method on the object and pushes the result back down the wire.
   (e) If the server object throws an exception, the server catches it and marshals it down the wire instead of the return value.

4. Back on the client side:
   (a) The return value of the RMI is unmarshalled and returned from the stub back to the client code itself.
   (b) If an exception is thrown from the server, that is unmarshalled and thrown from the stub.

1.1.2 What does the remote reference module do ?

It is responsible for translating between local and remote object references and for creating remote object references. The remote reference module holds a remote object table that records the correspondence between local object references in that process and remote object references (which are system wide).

Table includes:

1. An entry (in the table at server) for all remote objects held by the process
2. An entry (in the table at client) for each local proxy

1.1.3 Generation of classes for proxies, dispatcher and skeleton

Classes for proxies, dispatcher and skeleton are generated automatically by an interface compiler.

In Java RMI :

1. Set of methods offered by a remote object is defined as a Java interface that is implemented within the class of the remote object
2. Java RMI compiler generates the proxy, dispatcher and skeleton classes from the class remote object

1.1.4 Dynamic invocation: An alternative to proxies

Dynamic invocation gives the client access to a generic representation of a remote invocation. In order to make a dynamic invocation not only information (e.g., name) about the interface of the remote object are included in the remote object reference. Additionally the names of the methods and the types of the argument are required.

When is it useful?

In applications, where some of the interfaces of the remote objects cannot be predicted at design time.

1.1.5 Server and client programs

Server program

1. Contains classes for the dispatcher and skeletons, together with the implementations of the classes of all of the servants
2. Contains a initialization section (responsible for creating and initializing at least one of the servants to be hosted by the server)
3. Generally allocates a separate thread for the execution of each remote invocation -> designer of the remote object implementation must allow concurrent executions

Client program

1. Contain the classes of the proxies for all of the remote objects that it will invoke
2. Require a means of obtaining a remote object reference for at least one of the remote objects held by the server -> **binder**

1.1.6 Factory methods

What is a "factory" and why would you want to use one? A factory, in this context, is a piece of software that implements one of the "factory" design patterns. In general, a factory implementation is useful when you need one object to control the creation of and/or access to other objects.By using a factory in Java Remote Method Invocation (Java RMI), you can reduce the number of objects that you need to register with the Java RMI registry.

Servants are created either in the initialization section or in methods in a remote interface designed for that purpose

- **Factory method:** used to refer to a method that creates servants
- **Factory object:** object with factory methods

How Does a Factory Work in Java RMI?

Just like any other Java RMI program, there are a few basic players: a server that produces one or more remote objects, each of which implements a remote interface; a client that accesses a name server (the rmiregistry) to get a reference to one of the remote objects; and the rmiregistry, which facilitates the client's initial contact with the server.

For the picture below and the steps that follow, you may make the following assumptions:

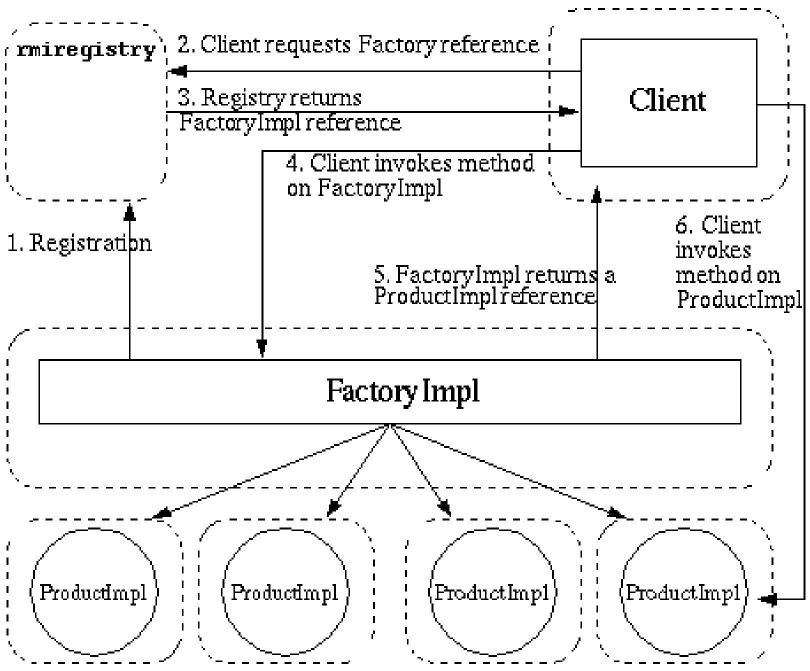

rmiregistry
2. Client requests Factory reference
3. Registry returns FactoryImpl reference
4. Client invokes method on FactoryImpl
Client
1. Registration
5. FactoryImpl returns a ProductImpl reference
6. Client invokes method on ProductImpl
Factory Impl
ProductImpl
ProductImpl
ProductImpl
ProductImpl

FIGURE 1.3: RMI Factory pattern.

1. There are two remote interfaces that the client understands, Factory and Product
2. The FactoryImpl implements the Factory interface and the ProductImpl implements the Product interface
3. The FactoryImpl registers, or is registered, with the rmiregistry
4. The client requests a reference to a Factory
5. The rmiregistry returns a remote reference to a FactoryImpl
6. The client invokes a remote method on the FactoryImpl to obtain a remote reference to a ProductImpl
7. The FactoryImpl returns a remote reference to an existing ProductImpl or to one that it just created, based on the client request
8. The client invokes a remote method on the ProductImpl

1.2 Implementation of the Dictionary

I have decided to use a mixture of ArrayList and a database to store the words and their meaning. ArrayList for temporary base as an object and in the database as a permanent base.

HyperSQL

HSQLDB - 100% Java Database

FIGURE 1.4: HyperSQL.

My choice of DBMS is **HyperSQL database management system** as it has 3 advantages First it can be **embedded** in my application, and second it is **lightweight and fast** for simple **CRUD** applications, and third it is fully **integrated for JAVA**.

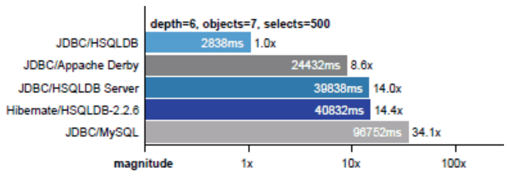


FIGURE 1.5: HyperSQL performance.

1.2.1 A set of operations can be defined on the dictionary

Add a word and its definition.

In Interface

```
void save(String word, String def) throws RemoteException;
```

In Client

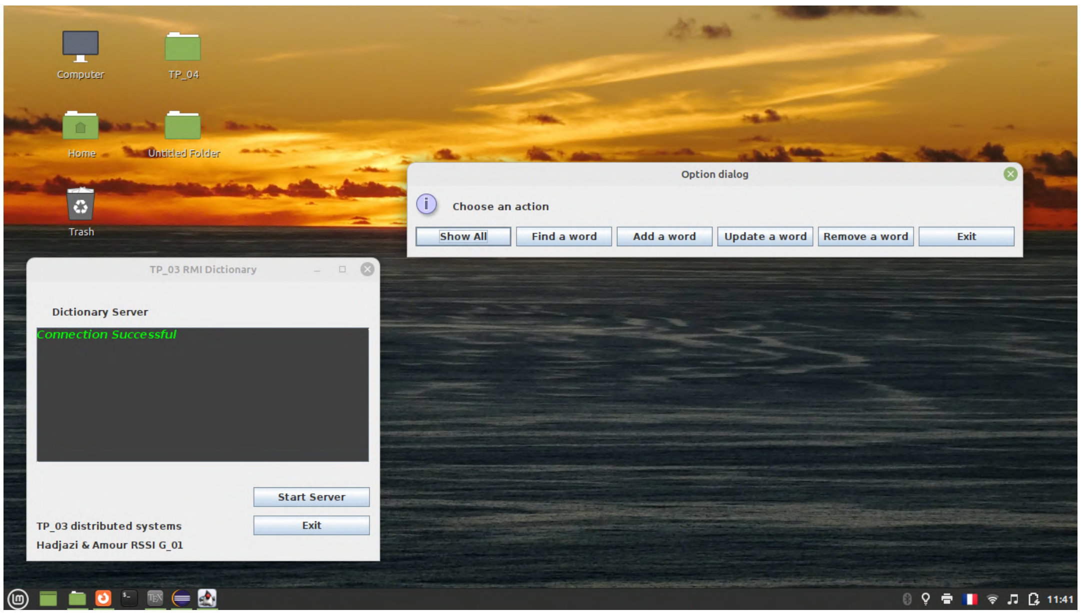
```
 String name = "Words";
            String wordd = JOptionPane.showInputDialog("Enter the
                word you want to add");
            String meann = JOptionPane.showInputDialog("Enter its
                meaning");
            comp.save(wordd, meann);
```

In Factory

```
@Override
    public void save(String word, String def) throws
        RemoteException {
         Word wordd = database.save(word, def);
         database.addWords(wordd);
    }
```

In Dictionary DB

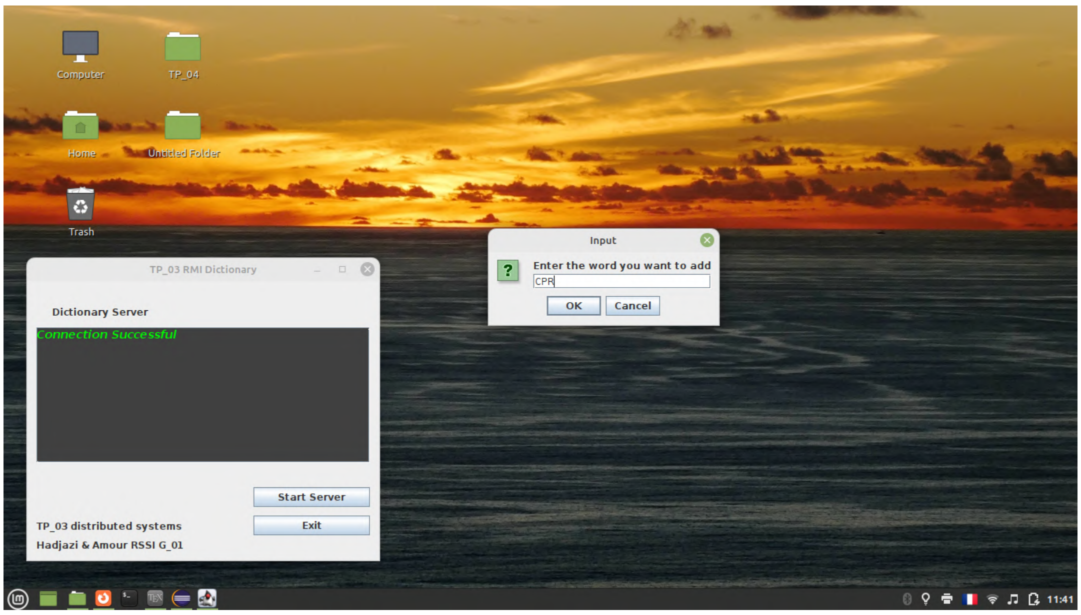
```
public Word save(String wword,String defofword) {
         Word c = new Word(wword, defofword);
         return c;
    }

    public void addWords(Word word) {
        try (PreparedStatement st = conn.prepareStatement("INSERT
            INTO dictionary (wword, defofword) VALUES (?, ?);")) {
            st.setString(1, word.getwword());
            st.setString(2, word.getdefofword());
            st.addBatch();
            st.executeBatch();
        }catch(SQLException ex) {
            System.out.println("Word Already Exist!");
        }
    }
```



Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.6: Add a word 1.



Dictionary Server

Connection Successful

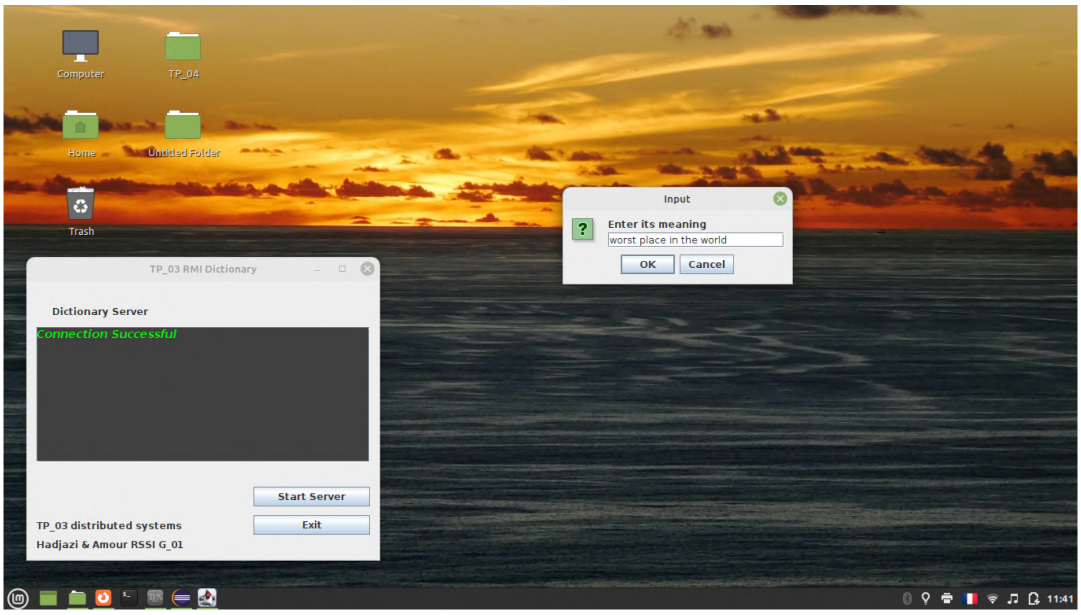
TP_03 distributed systems

Hadjazi & Amour RSSI G_01

Enter the word you want to add

CPR

FIGURE 1.7: Add a word 2.



Dictionary Server

Connection Successful

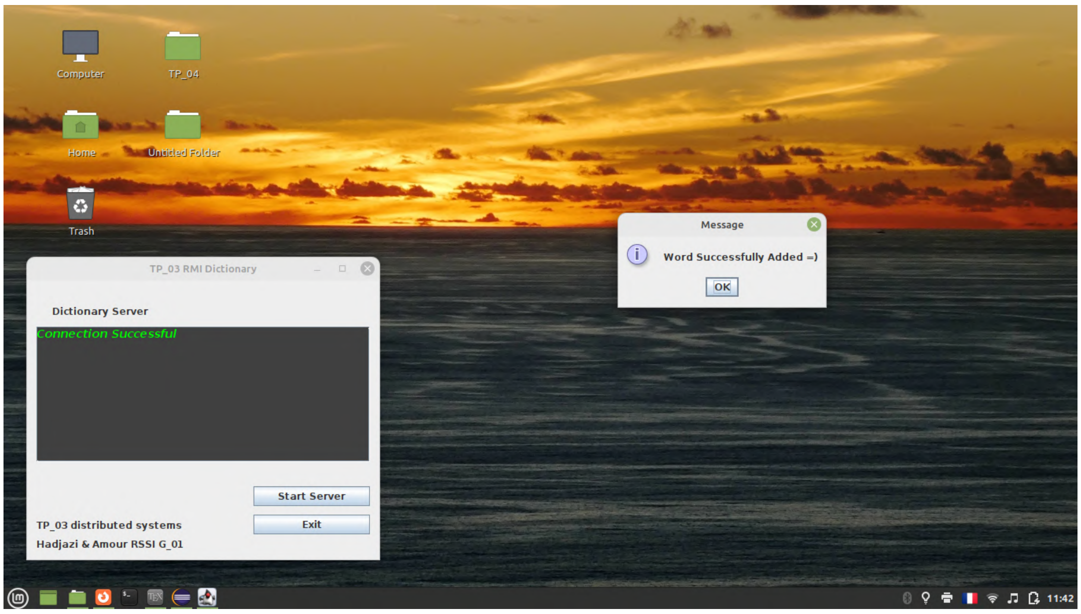
TP_03 distributed systems

Hadjazi & Amour RSSI G_01

Enter its meaning

worst place in the world

FIGURE 1.8: Add a word 3.



Message

Word Successfully Added =)

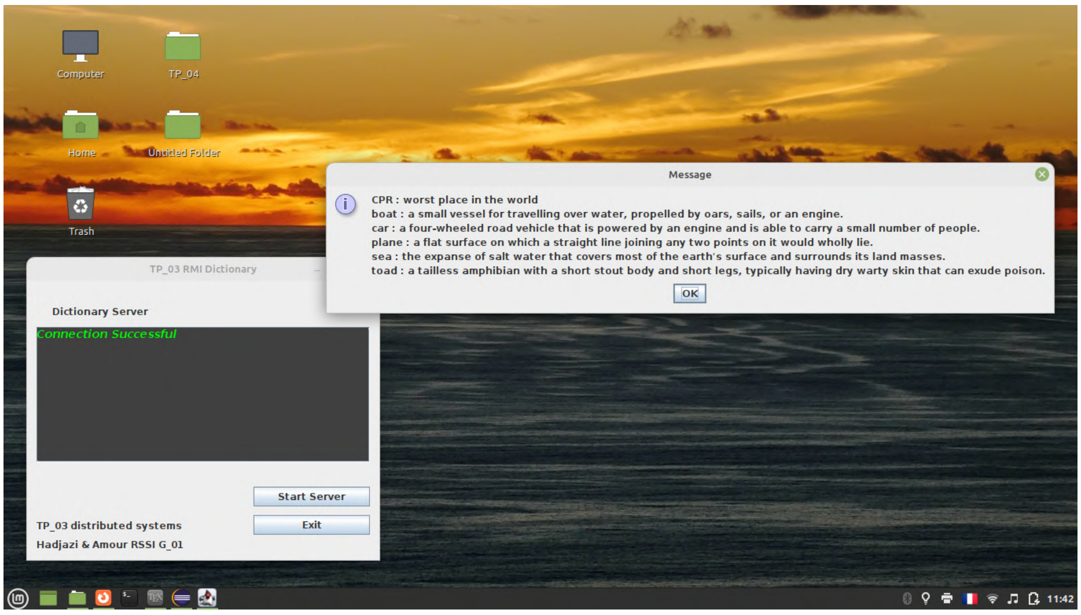
OK

Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.9: Add a word 4.



Message

CPR : worst place in the world
boat : a small vessel for travelling over water, propelled by oars, sails, or an engine.
car : a four-wheeled road vehicle that is powered by an engine and is able to carry a small number of people.
plane : a flat surface on which a straight line joining any two points on it would wholly lie.
sea : the expanse of salt water that covers most of the earth's surface and surrounds its land masses.
toad : a tailless amphibian with a short stout body and short legs, typically having dry warty skin that can exude poison.

OK

Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.10: Add a word 5.

Look up the definition of a given word.

In Interface

```java
String lookup(String keyword)throws RemoteException;
```

In Client

```java
String code = JOptionPane.showInputDialog("Type the word you are
    looking for ?");
String resultLookup = comp.lookup(code);
    JOptionPane.showMessageDialog(null, "Word :"
            + code + "\n" + "Means : "
            + resultLookup,
            comp.lookup(code), JOptionPane.INFORMATION_MESSAGE);
```

In Factory

```
 @Override
    public String lookup(String keyword) throws RemoteException {
        List<String> result = null;
        try{
            result = database.lookup(keyword);
        }catch(SQLException ex){
            ex.printStackTrace();
        }
        String res = result.get(0);
        return res;
    }
```

In Dictionary DB

```
public List<String> lookup(String parameter) throws SQLException {
        List<String> result = new ArrayList<>();
        PreparedStatement st = conn.prepareStatement("SELECT * from
            dictionary WHERE wword LIKE ? OR defofword LIKE ?;");
        st.setString(1, '%'+parameter+'%');
        st.setString(2, '%'+parameter+'%');
        ResultSet rs = st.executeQuery();
        try{
            while(rs.next()) {
                final String wword = rs.getString("wword");
                final String defofword = rs.getString("defofword");
                result.add(wword + " : " + defofword);
            }
        }catch(Exception e) {
            e.printStackTrace();
        }
        return result;
    }
```

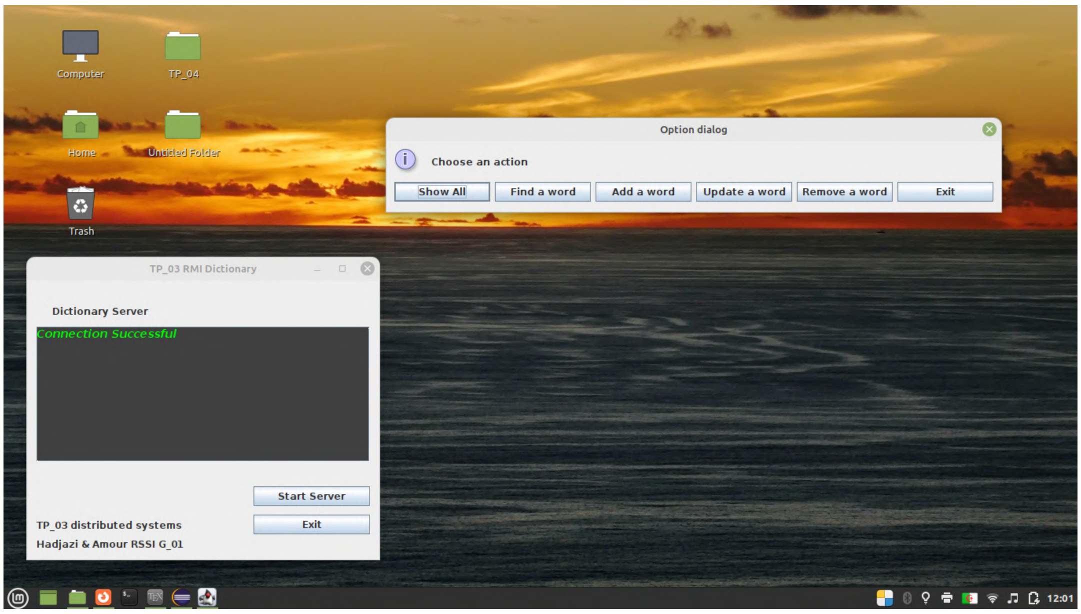


FIGURE 1.11: Find a word 1.

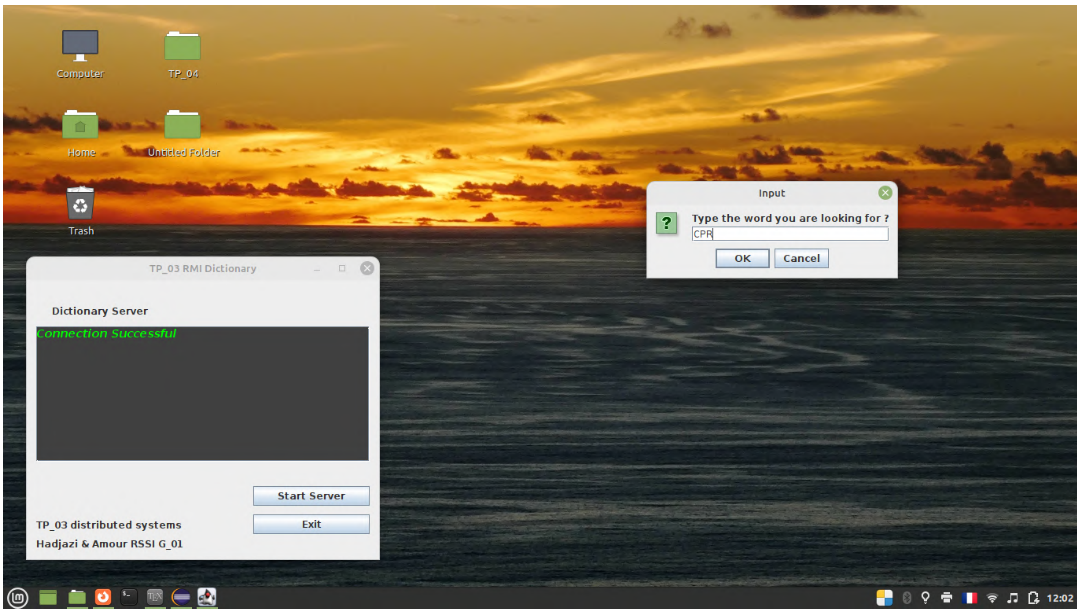
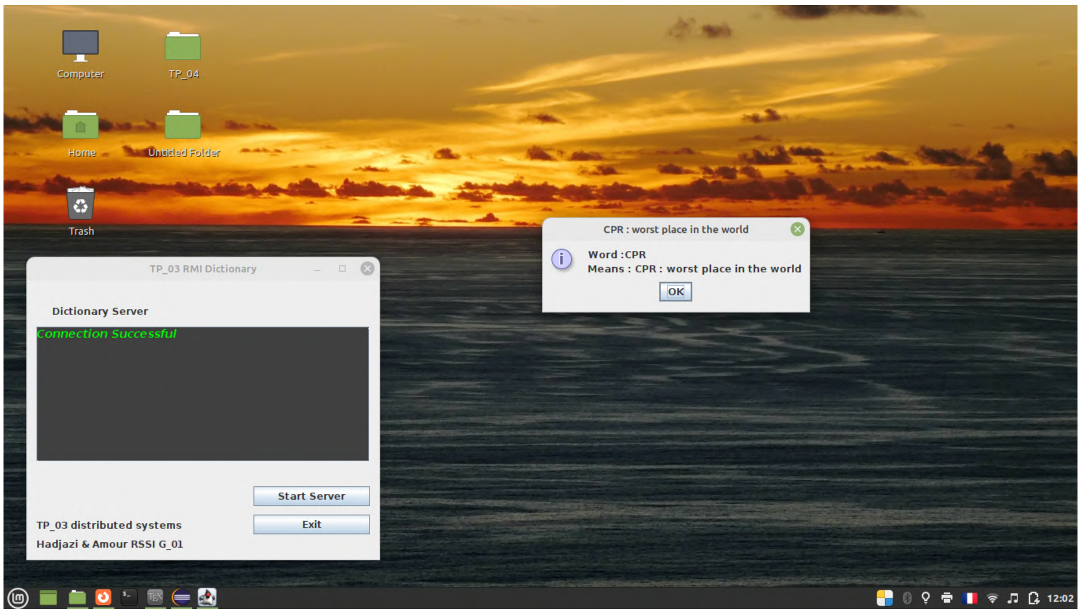


FIGURE 1.12: Find a word 2.



CPR : worst place in the world

Word :CPR
Means : CPR : worst place in the world

Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.13: Find a word 3.

Delete a dictionary entry.

In Interface

```java
void removeWord(String word) throws RemoteException;
```

In Client

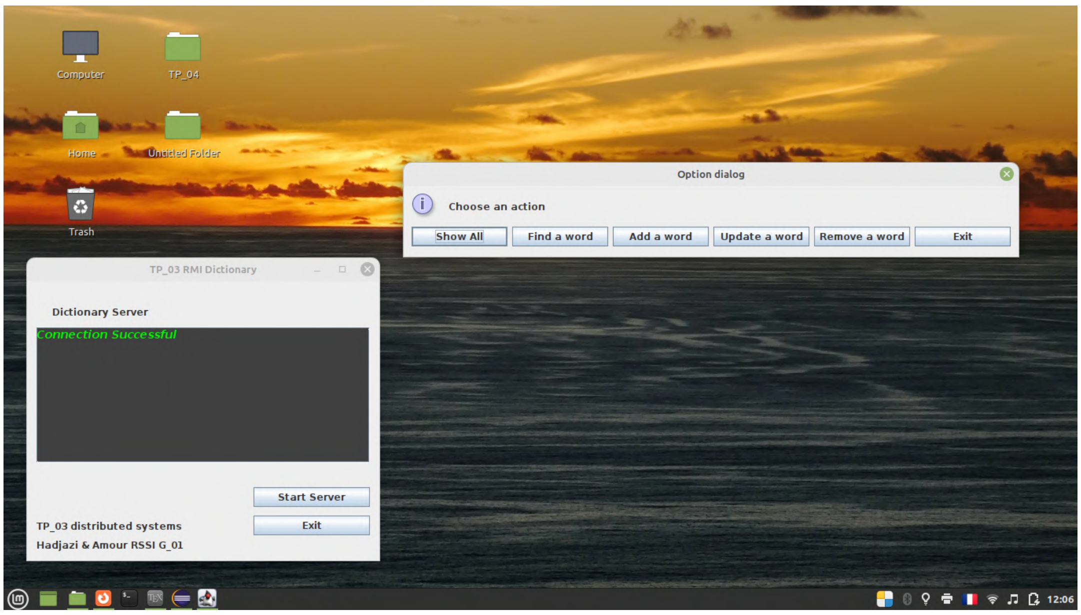
```java
String wordd = JOptionPane.showInputDialog("Enter the word you want
     to delete");
comp.removeWord(wordd);
```

In Factory

```java
@Override
    public void removeWord(String word) throws RemoteException {

        database.deleteWords(word);

    }
```

In Dictionary DB

```java
    public void deleteWords(String word) {
        try (PreparedStatement st = conn.prepareStatement("DELETE
            FROM dictionary WHERE wword ='"+word+"';")) {

            st.execute();
        }catch(SQLException ex) {
            System.out.println("ERROR Deleting!");
        }
    }
```



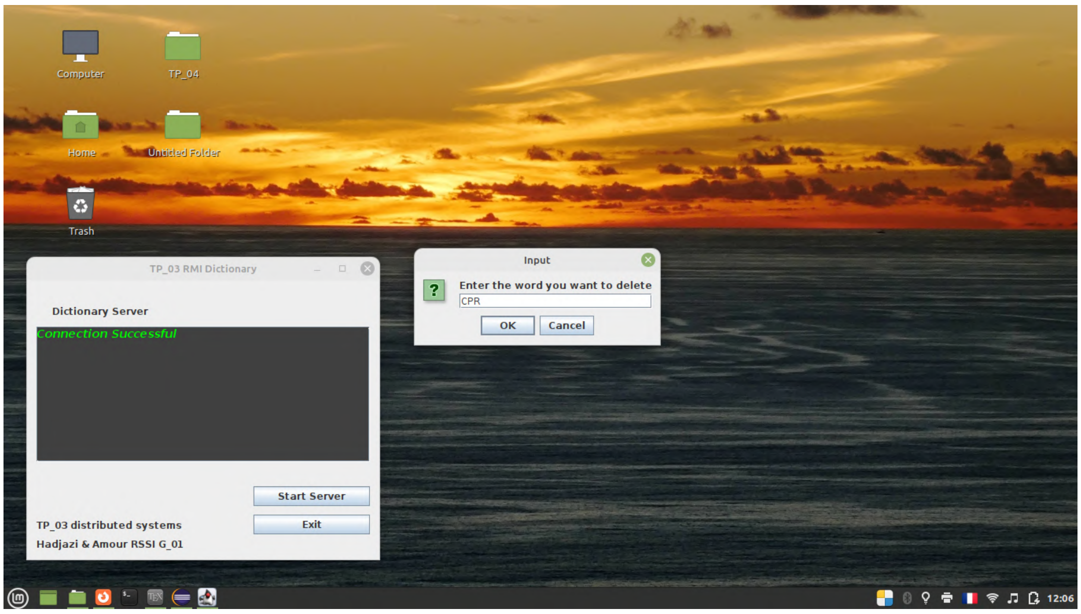
Choose an action

Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.14: Delete a word 1.



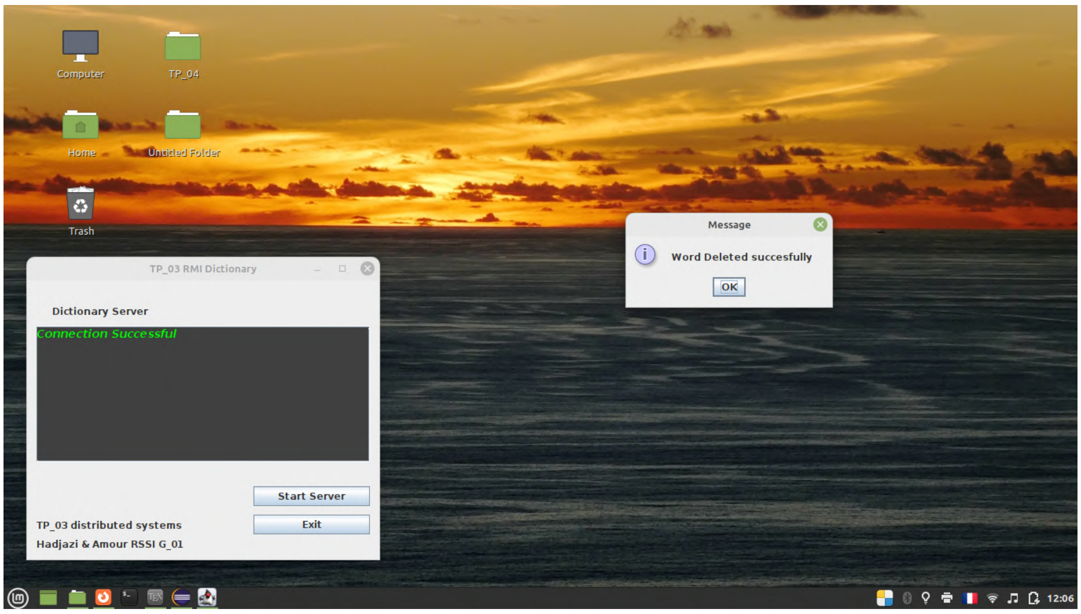
Enter the word you want to delete
CPR

Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.15: Delete a word 2.



Word Deleted succesfully

Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.16: Delete a word 3.

Modify the definition of a given word.

In Interface

```java
void replaceWord(String word, String def) throws RemoteException;
```

In Client

```java
String wordd = JOptionPane.showInputDialog("Enter the word you want to update");
String meann = JOptionPane.showInputDialog("Enter the new meaning : ");
comp.replaceWord(wordd,meann);
```

In Factory

```java
    @Override
    public void replaceWord(String word, String def) throws RemoteException {

         database.updateWords(word, def);

    }
```

In Dictionary DB

```java
public void updateWords(String word, String def) {
        try (PreparedStatement st = conn.prepareStatement("UPDATE dictionary SET wword = ?, defofword = ? WHERE wword = ?;")) {
            st.setString(1,word);
            st.setString(2,def);
            st.setString(3,word);
            st.addBatch();
```

```
                st.executeBatch();
            }catch(SQLException ex) {
                System.out.println("Error Updating!");
            }
        }
```

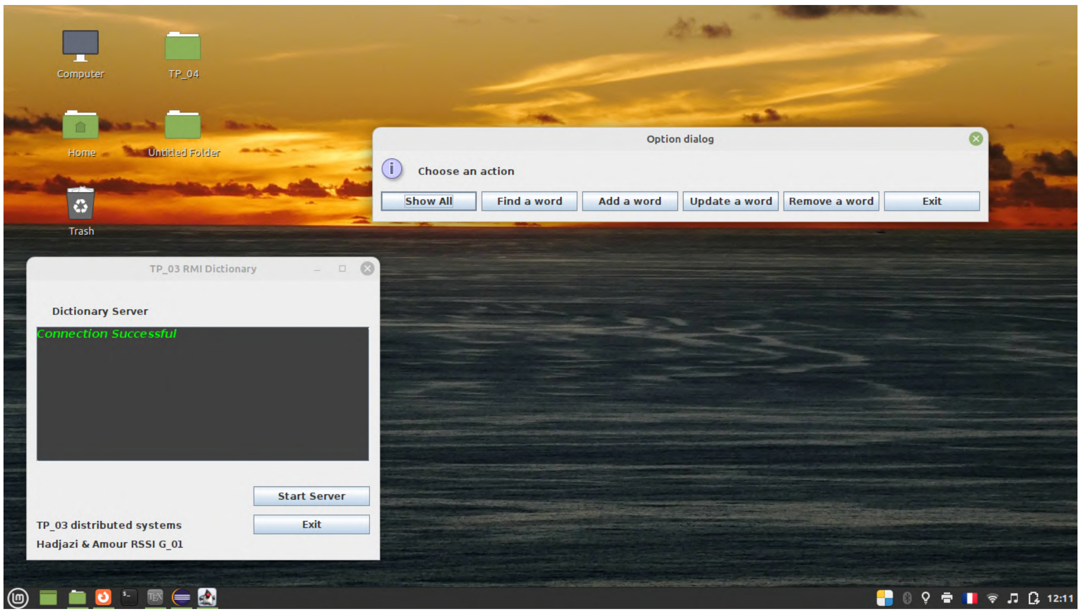


FIGURE 1.17: Update a word 1.

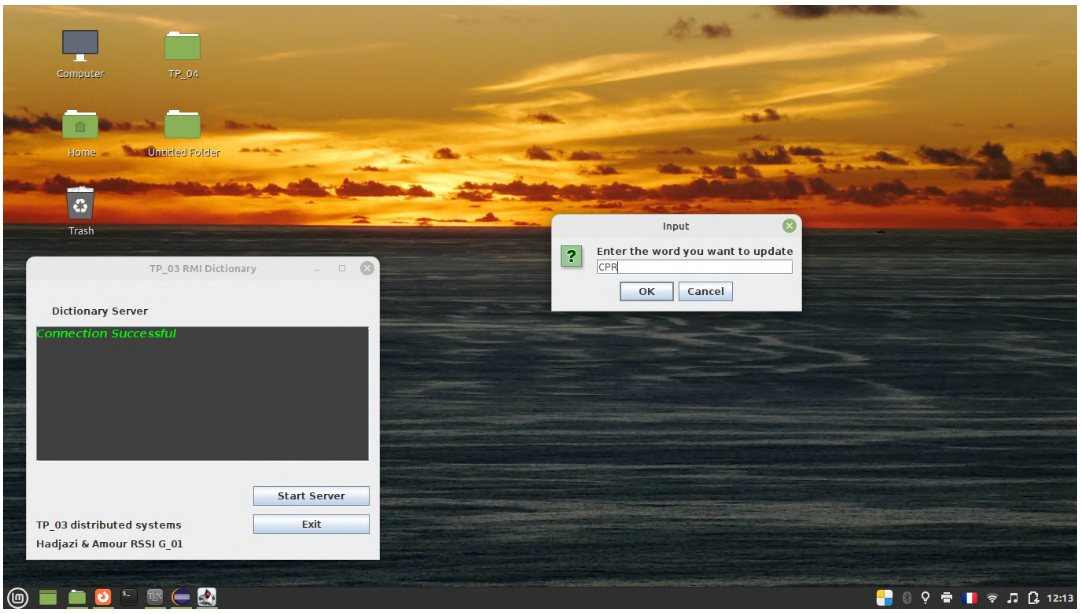
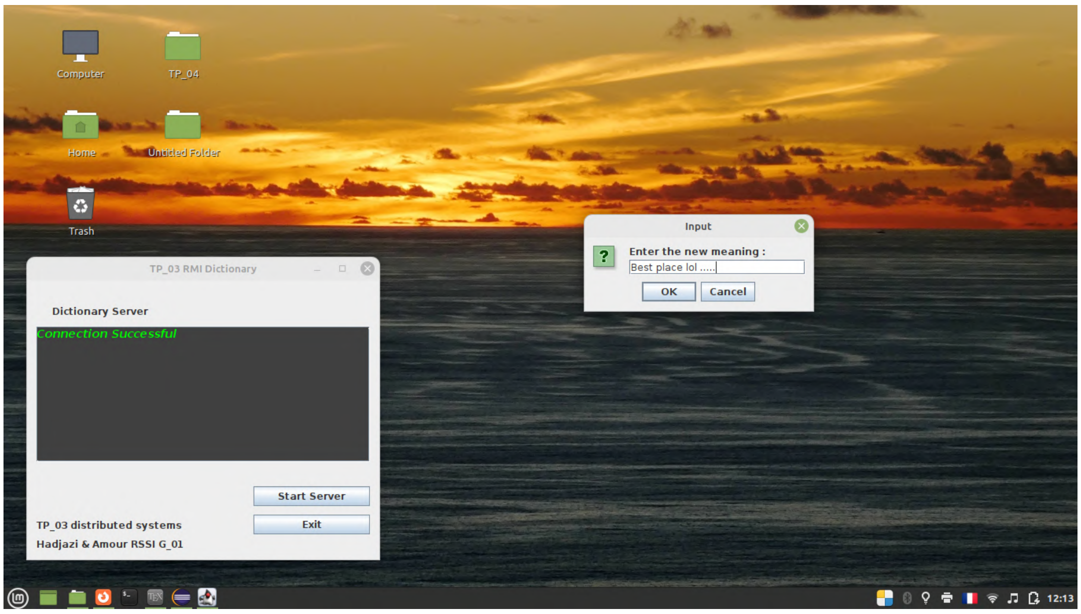


FIGURE 1.18: Update a word 2.



Dictionary Server

Connection Successful

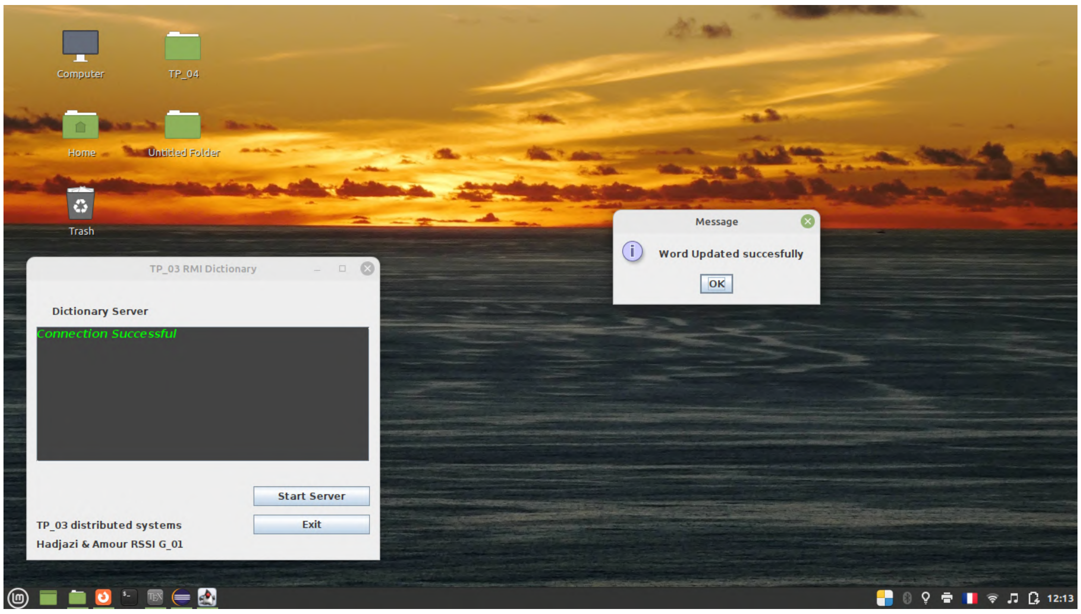
TP_03 distributed systems

Hadjazi & Amour RSSI G_01

Enter the new meaning :

Best place lol .....

FIGURE 1.19: Update a word 3.



Dictionary Server

Connection Successful

TP_03 distributed systems

Hadjazi & Amour RSSI G_01

Word Updated succesfully

FIGURE 1.20: Update a word 4.

View dictionary content.

In Interface

```
List<String> list() throws RemoteException ;
```

In Client

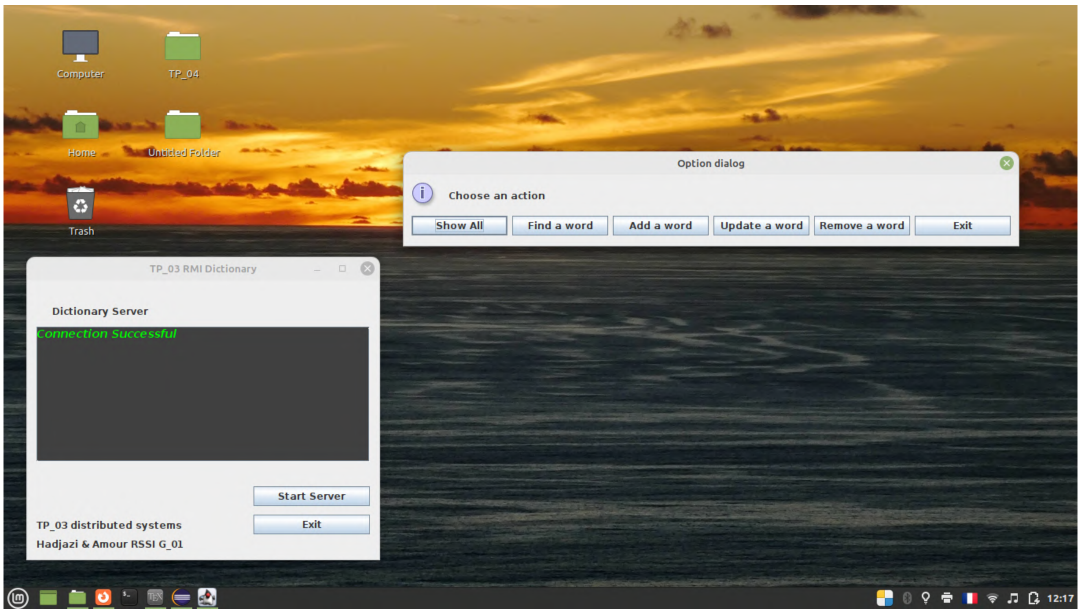
```
List<String> resultList = comp.list();

        StringBuilder message = new StringBuilder();
        resultList.forEach( x -> {
                message.append(x.toString() + "\n"); });
```

In Factory

```
    @Override
    public List<String> list() throws RemoteException {
        List<String> result = null;
        try{
            result = database.list();
        }catch(SQLException ex){
            ex.printStackTrace();
        }
        return result;
    }
```

In Dictionary DB

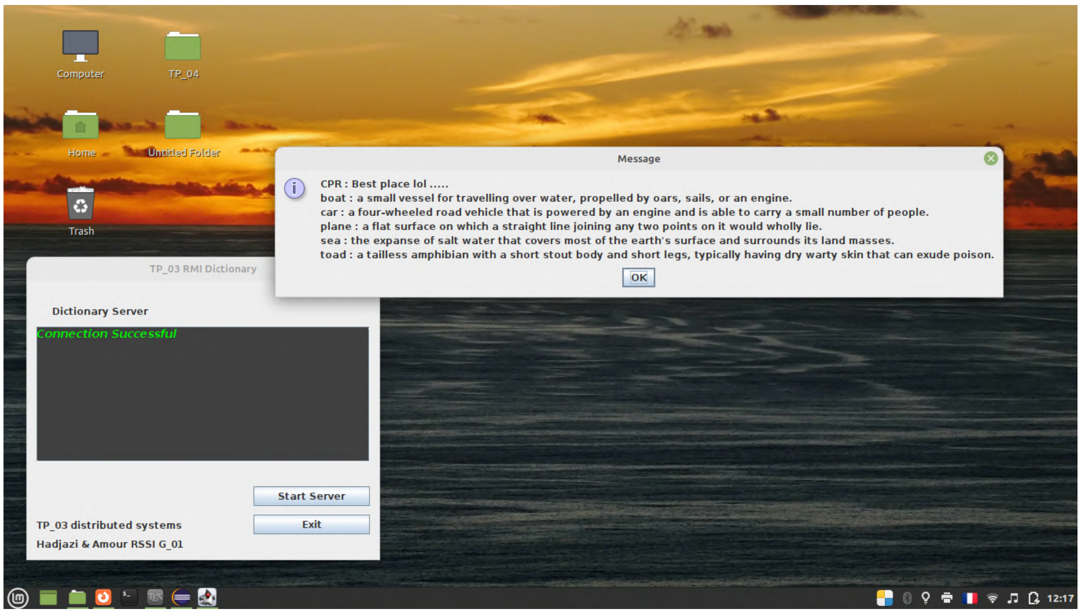
```
    public List<String> list() throws SQLException {
        List<String> result = new ArrayList<>();
        try(Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery("SELECT * from dictionary;")
            ) {
            while(rs.next()) {
                final String wword = rs.getString("wword");
                final String defofword = rs.getString("defofword");
                result.add(wword + " : " + defofword);
            }
        }
        return result;
    }
```



Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.21: View Dictionary Content 1.



Message

CPR : Best place lol .....
boat : a small vessel for travelling over water, propelled by oars, sails, or an engine.
car : a four-wheeled road vehicle that is powered by an engine and is able to carry a small number of people.
plane : a flat surface on which a straight line joining any two points on it would wholly lie.
sea : the expanse of salt water that covers most of the earth's surface and surrounds its land masses.
toad : a tailless amphibian with a short stout body and short legs, typically having dry warty skin that can exude poison.

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.22: View Dictionary Content 2.

1.2.2 Propose a java RMI implementation, with a dictionary that can be shared by all customers. (Defining a dictionary class implements Serializable).

Serializing Word class

```java
public class Word implements Serializable {
    private static final long serialVersionUID = 12351313553L;
    private String wword;
    private String defofword;

    public Word(String wword, String defofword) {
        this.wword = wword;
        this.defofword = defofword;
    }
```

1.2.3 Modify the implementation so that each client can have its own dictionary. (The dictionary class becomes a remote object with the use of the principle of the factory objects)

Serializing FaactoryImp class

```java
public class FaactoryImp implements IntDictionary, Serializable {

    private static final long serialVersionUID =
       1462043410155727587L;
    private static String user = "sa";
    private static String password = "";
    private static String url = "jdbc:hsqldb:mem:.";
    private static Dictionary database;

    public FaactoryImp() throws SQLException {
        this.database = new Dictionary(user, password, url);
    }
```

1.2.4 Bonus: access to the dictionary can be done by password for customers.

In Interface

```
public boolean authenticate(String userName, String password)
    throws RemoteException ;
```

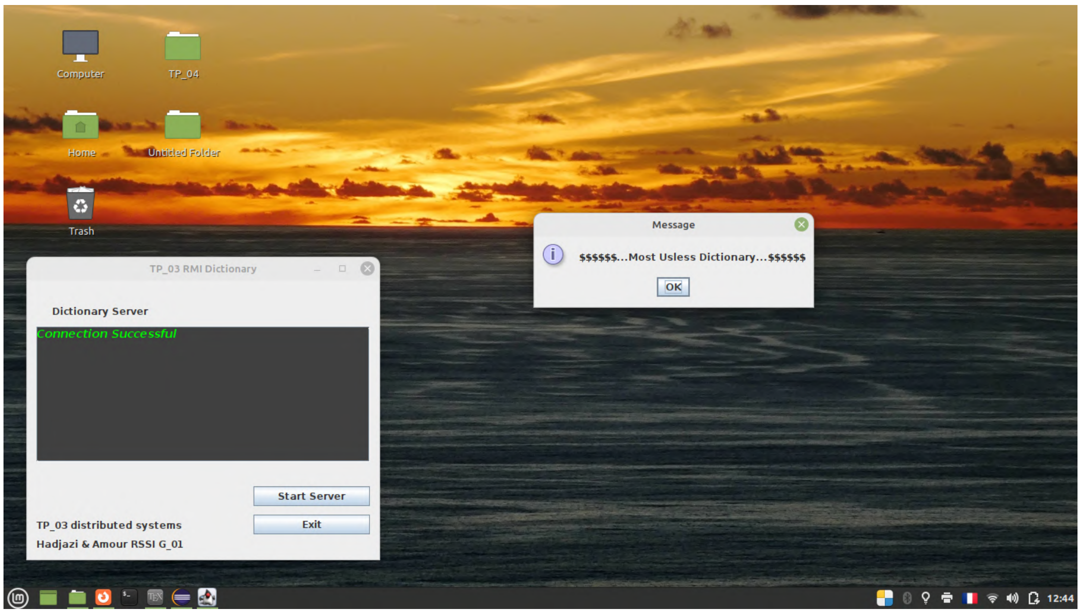
In Client

```
            // Invoking the Method
            boolean status = comp.authenticate(userName, password);

            if(status) {
            System.out.println("You are an authorized user...");
            JOptionPane.showMessageDialog(null, "WELCOME "+userName
                +"\nYou are an authorized user...");

            ...........................


            } else {

                System.out.println("Unauthorized Login Attempt");
                JOptionPane.showMessageDialog(null, "KICKED OUT
                    !!!!\nUnauthorized Login Attempt");

            }
```

In Factory

```
    @Override
    public boolean authenticate(String userName, String password)
            throws RemoteException {


        if ((userName != null && !userName.isEmpty())
                && (password != null && !password.isEmpty())) {

            if(((userName.equalsIgnoreCase("admin"))
                    && (password.equalsIgnoreCase("admin")))

                ||

                ((userName.equalsIgnoreCase("user1"))
                        && (password.equalsIgnoreCase("pass1")))

                ||

                ((userName.equalsIgnoreCase("user2"))
                                    && (password.equalsIgnoreCase("pass2")))


                               ) {

                        return true;
                  }
            }
            return false;
      }
```

Authorized Login



Message

$$$$$$...Most Usless Dictionary...$$$$$$

TP_03 RMI Dictionary

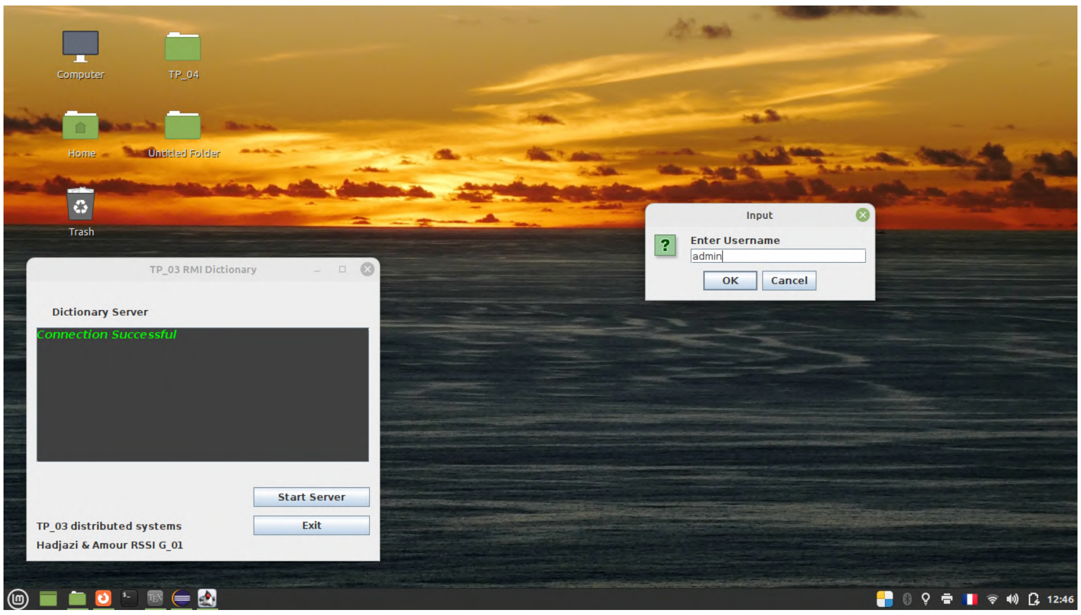
Dictionary Server

Connection Successful

TP_03 distributed systems

Hadjazi & Amour RSSI G_01

FIGURE 1.23: Login 1.



Input

Enter Username

admin

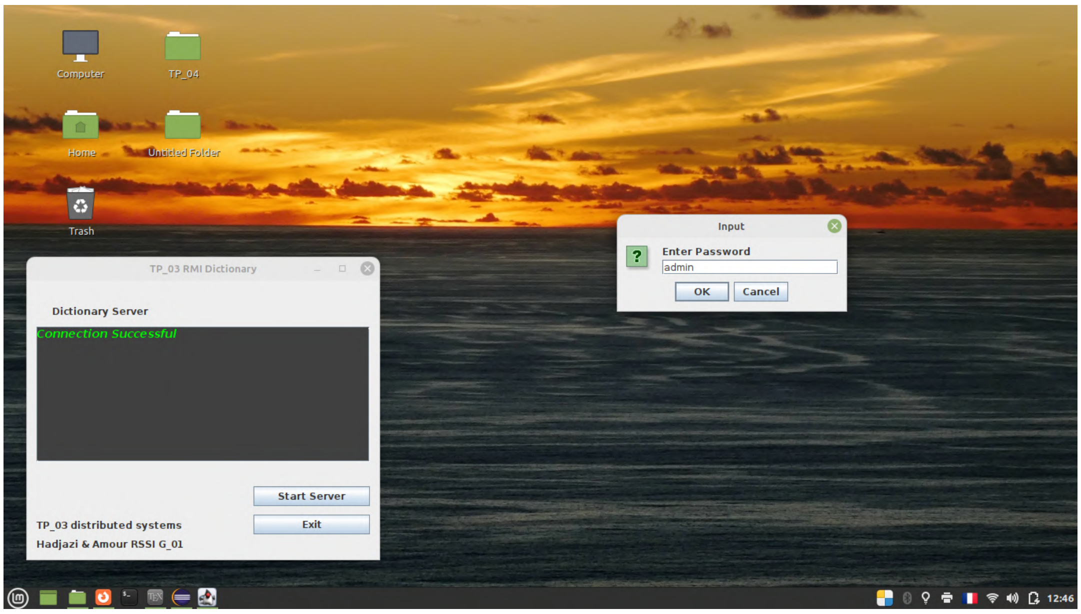
Dictionary Server

```
Connection Successful
```

TP_03 distributed systems

Hadjazi & Amour RSSI G_01

FIGURE 1.24: Login 2.



Input

Enter Password

admin

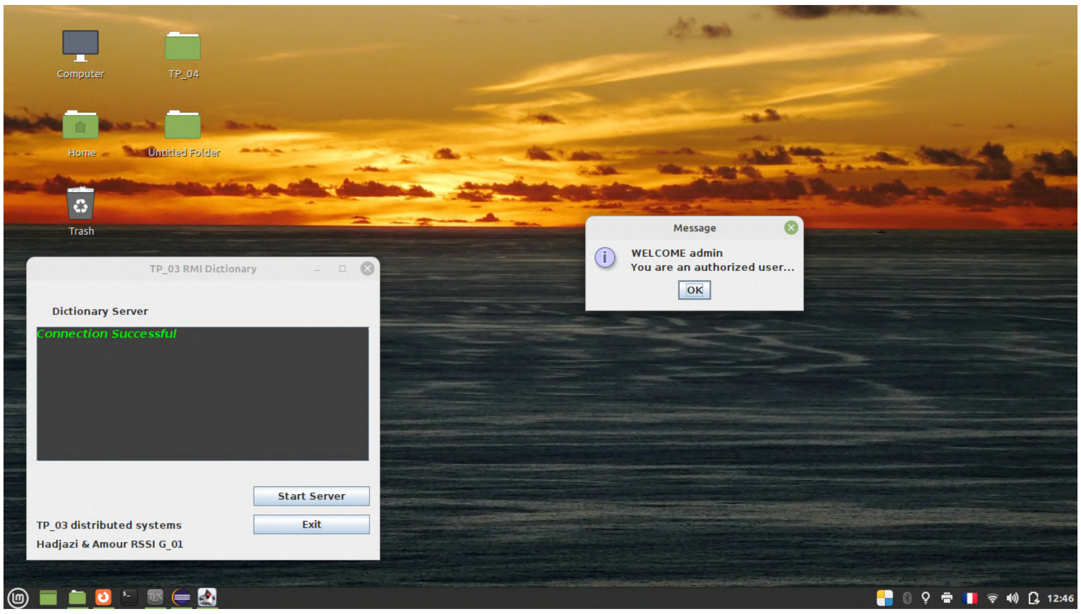
Dictionary Server

```
Connection Successful
```

TP_03 distributed systems

Hadjazi & Amour RSSI G_01

FIGURE 1.25: Login 3.



Message

WELCOME admin
You are an authorized user...

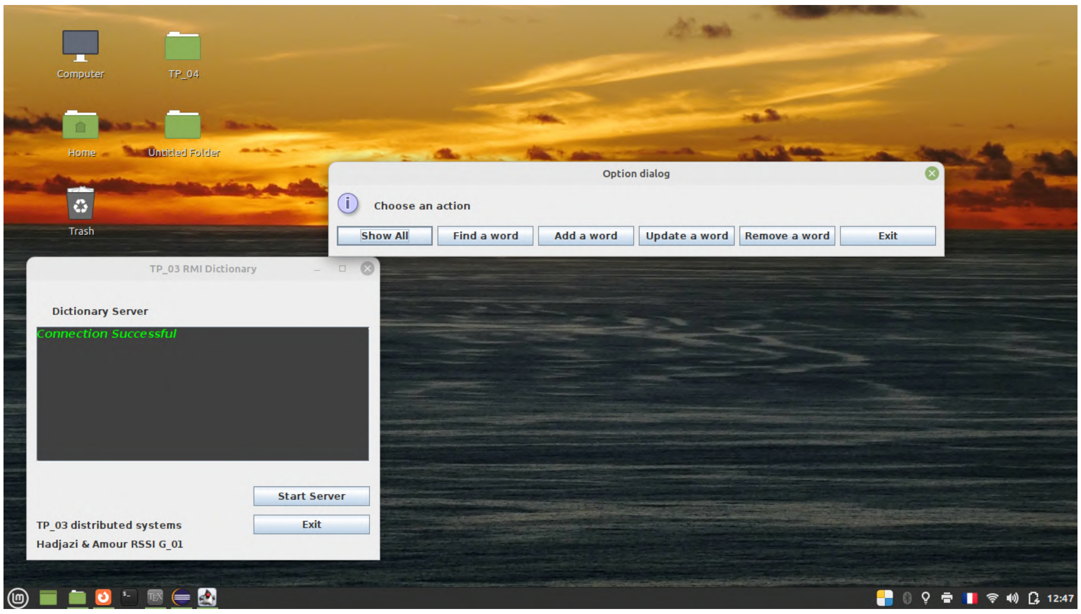
TP_03 RMI Dictionary

Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.26: Login 4.



Option dialog

Choose an action

TP_03 RMI Dictionary

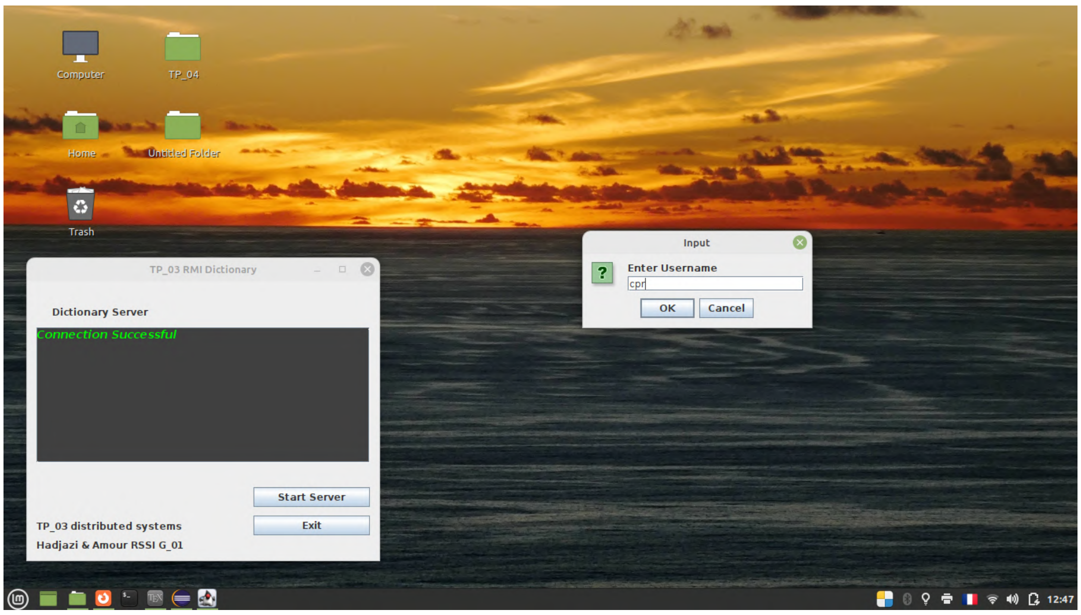
Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

FIGURE 1.27: Login 5.

Unauthorized Login



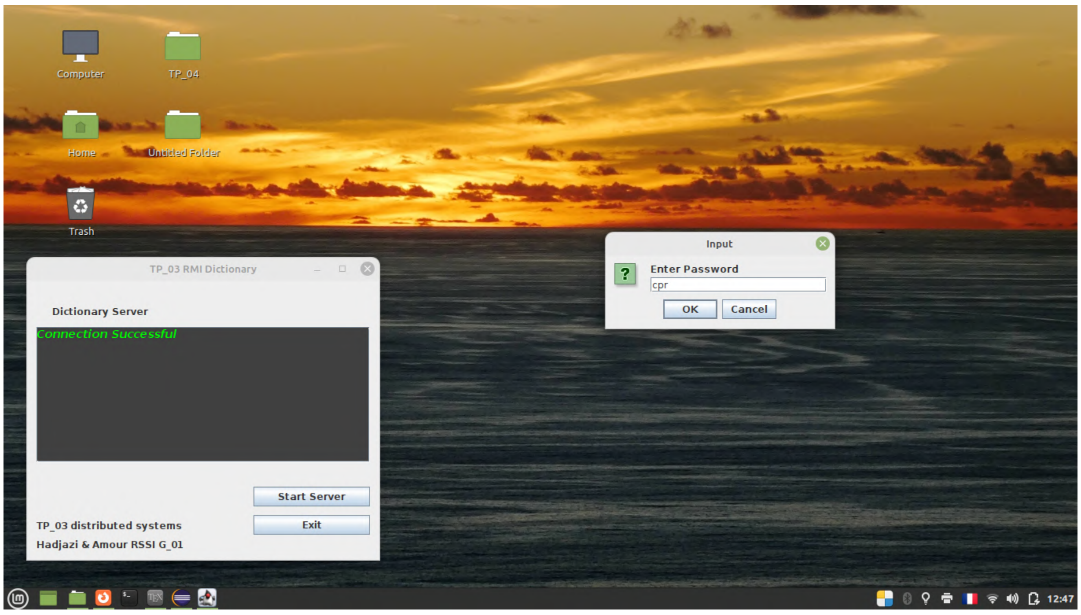
Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

Enter Username
cpr

FIGURE 1.28: Resected Login 1.



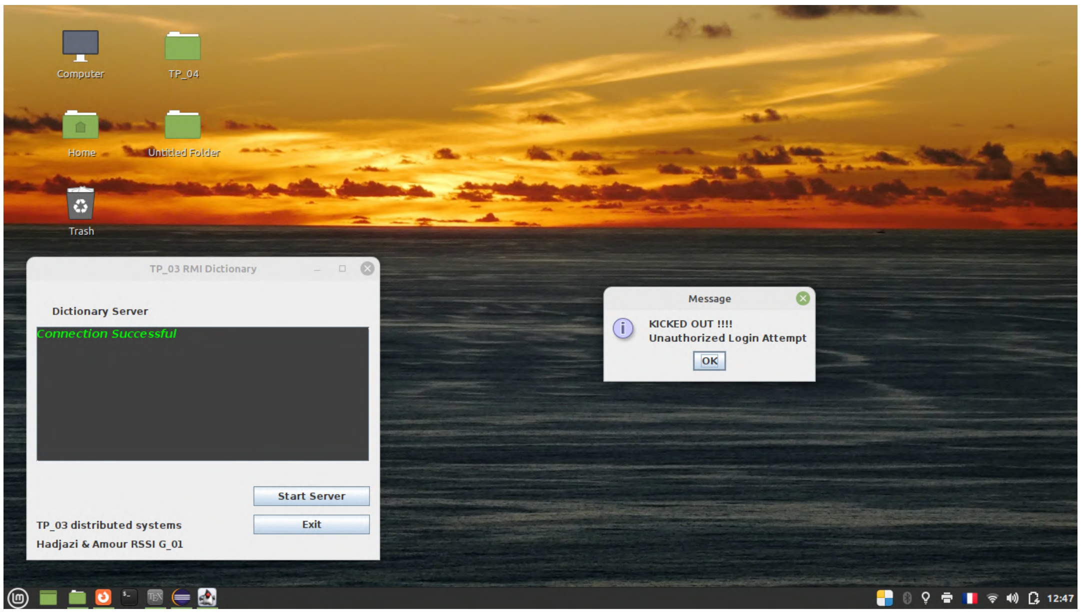
Dictionary Server

Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

Enter Password
cpr

FIGURE 1.29: Resected Login 2.



Connection Successful

TP_03 distributed systems
Hadjazi & Amour RSSI G_01

KICKED OUT !!!!
Unauthorized Login Attempt

FIGURE 1.30: Resected Login 3.

Appendix A

Appendix A

A.1 IntDictionary

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface IntDictionary extends Remote{

    void save(String word, String def) throws RemoteException;

    String lookup(String keyword)throws RemoteException;

    List<String> list() throws RemoteException ;

    void removeWord(String word) throws RemoteException;

    void replaceWord(String word, String def) throws
        RemoteException;

    public boolean authenticate(String userName, String password)
        throws RemoteException ;

}
```

A.2 Word

```
import java.io.Serializable;
import java.util.Objects;

public class Word implements Serializable{
    private static final long serialVersionUID = 12351313553L;
    private String wword;
    private String defofword;

    public Word(String wword, String defofword) {
        this.wword = wword;
        this.defofword = defofword;
    }

    public String getwword() {
        return wword;
    }
```

```java

    public void setwword(String wword) {
        this.wword = wword;
    }

    public String getdefofword() {
        return defofword;
    }

    public void setdefofword(String defofword) {
        this.defofword = defofword;
    }

    @Override
    public String toString() {
        return "Word{" +
                "wword='" + wword + '\'' +
                ", defofword='" + defofword + '\'' +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Word word = (Word) o;
        return Objects.equals(wword, word.wword) &&
                Objects.equals(defofword, word.defofword);
    }

    @Override
    public int hashCode() {
        return Objects.hash(wword, defofword);
    }
}
```

A.3 Dictionary

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class Dictionary implements AutoCloseable{

    private static String user = "sa";
    private static String password = "";
    private static String url = "jdbc:hsqldb:mem:.";
    private static Connection conn;



    public Dictionary(String user,String password,String url) throws SQLException {
        this.conn = DriverManager.getConnection(url,user,password);
        try {
                createTables(conn);
                addWords(new Word("plane", "a flat surface on which a straight line joining any two points on it would wholly lie."));
                addWords(new Word("car", "a four-wheeled road vehicle that is powered by an engine and is able to carry a small number of people."));
                addWords(new Word("boat", "a small vessel for travelling over water, propelled by oars, sails, or an engine."));
                addWords(new Word("sea", "the expanse of salt water that covers most of the earth's surface and surrounds its land masses."));
                addWords(new Word("toad", "a tailless amphibian with a short stout body and short legs, typically having dry warty skin that can exude poison."));
                printWords();
            } catch (SQLException e) {
                e.printStackTrace();
            }
    }

    @Override
    public void close() throws Exception {
        conn.close();
    }

    public Word save(String wword,String defofword) {
        Word c = new Word(wword, defofword);
        return c;
    }

    private void createTables(Connection conn) throws SQLException {
        try(Statement st = conn.createStatement()) {
            //st.executeUpdate("DROP TABLE IF EXISTS dictionary;");
            st.executeUpdate("CREATE TABLE dictionary (wword VARCHAR(80) PRIMARY KEY, defofword VARCHAR(200));");
            System.out.println("Tables created");
        }
    }

    public void addWords(Word word) {
        try (PreparedStatement st = conn.prepareStatement("INSERT INTO dictionary (wword, defofword) VALUES (?, ?);")) {
            st.setString(1, word.getwword());
            st.setString(2, word.getdefofword());
            st.addBatch();
            st.executeBatch();
        }catch(SQLException ex) {
            System.out.println("Word Already Exist!");
        }
    }

    public void updateWords(String word, String def) {
        try (PreparedStatement st = conn.prepareStatement("UPDATE
            dictionary SET wword = ?, defofword = ? WHERE wword = ?;
            ")) {
            st.setString(1,word);
            st.setString(2,def);
            st.setString(3,word);
            st.addBatch();
            st.executeBatch();
        }catch(SQLException ex) {
            System.out.println("Error Updating!");
        }
    }

    public void deleteWords(String word) {
        try (PreparedStatement st = conn.prepareStatement("DELETE
            FROM dictionary WHERE wword ='"+word+"';")) {

            st.execute();
        }catch(SQLException ex) {
            System.out.println("ERROR Deleting!");
        }
    }

    public void printWords() throws SQLException {
        try(Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery("SELECT * from dictionary;")
            ) {
            while(rs.next()) {
                final String wword = rs.getString("wword");
                final String defofword = rs.getString("defofword");
                System.out.println(wword + " : " + defofword);
            }
        }
    }

    public List<String> list() throws SQLException {
        List<String> result = new ArrayList<>();
        try(Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery("SELECT * from dictionary;")
            ) {
            while(rs.next()) {
                final String wword = rs.getString("wword");
                final String defofword = rs.getString("defofword");
                result.add(wword + " : " + defofword);
            }
        }
        return result;
    }

    public List<String> lookup(String parameter) throws
       SQLException {
        List<String> result = new ArrayList<>();
        PreparedStatement st = conn.prepareStatement("SELECT * from
            dictionary WHERE wword LIKE ? OR defofword LIKE ?;");
        st.setString(1, '%'+parameter+'%');
        st.setString(2, '%'+parameter+'%');
        ResultSet rs = st.executeQuery();
        try{
            while(rs.next()) {
                final String wword = rs.getString("wword");
                final String defofword = rs.getString("defofword");
                result.add(wword + " : " + defofword);
            }
        }catch(Exception e) {
            e.printStackTrace();
        }
        return result;
    }

}
```

A.4 FaactoryImp

```
import java.io.Serializable;
import java.rmi.RemoteException;
import java.sql.SQLException;
import java.util.List;

public class FaactoryImp implements IntDictionary, Serializable {

    private static final long serialVersionUID =
       1462043410155727587L;
    private static String user = "sa";
    private static String password = "";
    private static String url = "jdbc:hsqldb:mem:.";
    private static Dictionary database;

    public FaactoryImp() throws SQLException {
        this.database = new Dictionary(user, password, url);
    }

    @Override
    public void save(String word, String def) throws
       RemoteException {
        Word wordd = database.save(word, def);
        database.addWords(wordd);
    }

    @Override
    public String lookup(String keyword) throws RemoteException {
        List<String> result = null;
        try{
            result = database.lookup(keyword);
        }catch(SQLException ex){
            ex.printStackTrace();
        }
        String res = result.get(0);
```

```java
        return res;
    }

    @Override
    public List<String> list() throws RemoteException {
        List<String> result = null;
        try{
            result = database.list();
        }catch(SQLException ex){
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public boolean authenticate(String userName, String password)
            throws RemoteException {


        if ((userName != null && !userName.isEmpty())
                && (password != null && !password.isEmpty())) {

            if((userName.equalsIgnoreCase("admin"))
                    && (password.equalsIgnoreCase("admin"))) {

                return true;
            }
        }
        return false;
    }



    @Override
    public void removeWord(String word) throws RemoteException {

        database.deleteWords(word);


    }

    @Override
    public void replaceWord(String word, String def) throws
       RemoteException {

        database.updateWords(word, def);

    }


}
```

A.5 ServerGUI

```java
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingWorker;
import javax.swing.JLabel;
import java.awt.Color;
import java.awt.Font;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.awt.event.ActionEvent;

public class ServerGUI {

    private JFrame frmTp;
    static JTextArea textArea;

    /**
     * Launch the application.
     */
    public static void main(String[] args) throws RemoteException{
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    ServerGUI window = new ServerGUI();
                    window.frmTp.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public ServerGUI() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frmTp = new JFrame();
        frmTp.setTitle("TP_03 RMI Dictionary");
        frmTp.setBounds(100, 100, 450, 386);
        frmTp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmTp.getContentPane().setLayout(null);

        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setBounds(12, 59, 424, 172);
        frmTp.getContentPane().add(scrollPane);

        textArea = new JTextArea();
        textArea.setForeground(Color.GREEN);
        textArea.setFont(new Font("Dialog", Font.BOLD | Font.ITALIC
            , 14));
        textArea.setBackground(Color.DARK_GRAY);
        scrollPane.setViewportView(textArea);

        JLabel lblDictionaryServer = new JLabel("Dictionary Server"
            );
        lblDictionaryServer.setBounds(32, 32, 148, 15);
        frmTp.getContentPane().add(lblDictionaryServer);

        JButton btnStartServer = new JButton("Start Server");
        btnStartServer.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {


                new SwingWorker() {

                    @Override
                    protected Object doInBackground() throws
                        Exception {

                        new EServer().start();
                        return null;

                    }

                }.execute();






            }
        });
        btnStartServer.setBounds(288, 262, 148, 25);
        frmTp.getContentPane().add(btnStartServer);

        JButton btnExit = new JButton("Exit");
        btnExit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                textArea.append("Good Bye\nExiting....");
                Thread.sleep(500);
                System.exit(0);
                }catch(Exception ex) {
                    textArea.append("Error : " + ex);
                }
            }
        });
        btnExit.setBounds(288, 298, 148, 25);
        frmTp.getContentPane().add(btnExit);

        JLabel lblTp = new JLabel("TP_03 distributed systems");
        lblTp.setBounds(12, 304, 206, 15);
        frmTp.getContentPane().add(lblTp);

        JLabel lblHadjaziAmour = new JLabel("Hadjazi & Amour RSSI
            G_01");
        lblHadjaziAmour.setBounds(12, 328, 206, 15);
        frmTp.getContentPane().add(lblHadjaziAmour);
    }






    static class EServer extends Thread {



        public EServer() throws RemoteException {

        }

        public void run() {


            try {

                String name = "Words";
                IntDictionary engine = new FaactoryImp();
                IntDictionary stub = (IntDictionary)
                    UnicastRemoteObject.exportObject(engine, 0);


                Registry registry = LocateRegistry.createRegistry
                    (1888);
                registry.rebind(name, stub);
                System.out.println("Connection Successful");
                textArea.append("Connection Successful");

            } catch (Exception e) {

                System.err.println("ERROR connecting: " + e);
                textArea.append("ERROR connecting: " + e);
                e.printStackTrace();
            }



        }
    }
}
```

A.6 ClientGUI

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import javax.swing.JOptionPane;
import java.util.List;
import java.util.NoSuchElementException;

public class ClientGUI {

    public static void main(String[] args) {


        try {
            String name = "Words";
            Registry registry = LocateRegistry.getRegistry(1888);
            IntDictionary comp = (IntDictionary) registry.lookup(
               name);






            JOptionPane.showMessageDialog(null, "$$$$$$...Most
               Usless Dictionary...$$$$$$");
            String userName = JOptionPane.showInputDialog("Enter
               Username");
            String password = JOptionPane.showInputDialog("Enter
               Password");

            // Invoking the Method
            boolean status = comp.authenticate(userName, password);

            if(status) {

                System.out.println("You are an authorized user...")
                   ;
                JOptionPane.showMessageDialog(null, "WELCOME "+
                   userName+"\nYou are an authorized user...");




                 boolean findMore;
                    do{
                        String[] options = {"Show All", "Find a
                           word", "Add a word","Update a word","
                           Remove a word", "Exit"};

                        int choice = JOptionPane.showOptionDialog(
                           null, "Choose an action", "Option dialog
                           ",
                                    JOptionPane.DEFAULT_OPTION,
                                    JOptionPane.INFORMATION_MESSAGE
                                       ,
                                    null, options, options[0]);

                        switch(choice){
                      //Show all words
                            case 0:{
```

```java
                List<String> resultList = comp.list();

                StringBuilder message = new StringBuilder();
                resultList.forEach( x -> {
                    message.append(x.toString() + "\n");
                });
                JOptionPane.showMessageDialog(null, new String(message));




                break;
            }
            // Find a word
            case 1: {
                String code = JOptionPane.showInputDialog("Type the word you are looking for ?");
                try {
                    String resultLookup = comp.lookup(code);


                    JOptionPane.showMessageDialog(null, "Word :"
                            + code + "\n" + "Means : "
                            + resultLookup,
                    comp.lookup(code), JOptionPane.INFORMATION_MESSAGE);

                } catch (NoSuchElementException ex) {
                    JOptionPane.showMessageDialog(null, "Word Not found");
                }
                break;
            }
            // Add a word
            case 2: {
                String wordd = JOptionPane.showInputDialog("Enter the word you want to add");
                String meann = JOptionPane.showInputDialog("Enter its meaning");
                try {
                    comp.save(wordd, meann);


                    JOptionPane.showMessageDialog(null, "Word Successfully Added =)");
```

```

                                } catch (NoSuchElementException ex) {

                                    JOptionPane.showMessageDialog(null, "World already exist !!!");
                                }
                                break;
                            }
                            //Update Function
                            case 3: {
                                String wordd = JOptionPane.showInputDialog("Enter the word you want to update");
                                String meann = JOptionPane.showInputDialog("Enter the new meaning : ");
                                try {

                                    comp.replaceWord(wordd,meann);

                                    JOptionPane.showMessageDialog(null, "Word Updated succesfully");

                                } catch (NoSuchElementException ex) {
                                    JOptionPane.showMessageDialog(null, "Word Not found");
                                }
                                break;
                            }
                            //Delete Function
                            case 4: {
                                String wordd = JOptionPane.showInputDialog("Enter the word you want to delete");

                                try {


                                    comp.removeWord(wordd);



                                    JOptionPane.showMessageDialog(null, "Word Deleted succesfully");

                                } catch (NoSuchElementException ex) {
                                    JOptionPane.showMessageDialog(null, "Word Not found");
                                }
                                break;
                            }
                            case 5: {

                                JOptionPane.showMessageDialog(null,
                                    "Good bye =)\nThnak you");
                                System.exit(0);

                                break;
                            }
                            default:
                                JOptionPane.showMessageDialog(null,
                                    "Good bye =)\nThnak you");
                                System.exit(0);

                                break;
                        }
                        findMore = (JOptionPane.showConfirmDialog(
                           null, "Do you want to exit?","Exit",
                                    JOptionPane.YES_NO_OPTION) ==
                                       JOptionPane.NO_OPTION);

                    }while(findMore);
            } else {

                System.out.println("Unauthorized Login Attempt");
                JOptionPane.showMessageDialog(null, "KICKED OUT
                   !!!!\nUnauthorized Login Attempt");

            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```