

National College of Ireland

**Distributed Systems (BSHCSDE4)**

Project Report

BSc (Honours) in Computing

Software Development

Academic Year 2020/2021

Student Name: Daniel Costel Neagu

Student Number: x17128463

Student Email: [x17124863@student.ncirl.ie](mailto:x17124863@student.ncirl.ie)

Contents

[**Distributed Systems (BSHCSDE4)** 1](#_Toc67862043)

[Overview 2](#_Toc67862044)

[Service Definition 2](#_Toc67862045)

[Service Implementation 3](#_Toc67862046)

[Naming Services 3](#_Toc67862047)

[Remote Error Handling 4](#_Toc67862048)

[Client - Graphical User Interface 5](#_Toc67862049)

[GitHub 5](#_Toc67862050)

[References 6](#_Toc67862051)

# Overview

In this report, I present the scenario and services I have chosen to design and developed a Java Maven application that is using a collection of independent services located on different devices that share resources with each other over the network to be able to achieve a common goal. All these services are called distributed systems and they will appear to the end-user like he is interacting with only one device. This will simulate the operations of a Smart Automated Cloud Management Environment.

# Service Definition

For my Smart Automated Cloud Management Environment system to be able to work properly first it needs to call methods on each other to synchronise easily, in this way it will give the end-user an appearance of interacting with only one device. For this to be achieved I will use the gRPC, which is an open-source remote procedure call and was developed in 2015 by Google at the beginning to be able to connect their microservices in their Datacentres (gRPC, 2021).

The service definition is implemented in the proto files for each service implementation that can be found in the resources directory of my project along with its afferent type of RPC invocation.

In the userService.proto file we can find two simple RPC invocations, one for userLogin and the second one for userLogout services:

rpc login(LoginRequest) returns (LoginResponse){}

rpc logout(Empty) returns (LogoutResponse){}

In the dataBaseService.proto file we can find server-striming RPC streaming invocation:

rpc dataBase\_do(RequestMessage) returns (stream ResponseMessage) {}

In the VMService.proto file we can find all four types of RPC invocations, simple RPC, serverside streaming RPC, client-side streaming RPC and the bidirectional streaming RPC:

rpc vmServices\_do(RequestMessage) returns (ResponseMessage) {}

rpc vmServices\_broadcast\_status(RequestStatus) returns (stream ResponseStatus) {}

rpc vmServices\_deploy\_code(stream RequestDeployCode) returns (stream ResponseDeployCode) {}

rpc vmServices\_deploy\_database(stream RequestDeployDatabase) returns (ResponseDeployDatabase) {}

# Service Implementation

The first service used and implemented in my Smart Automated Cloud Management Environment project is the User service. This service is allowing the user that is interacting with the system to be authenticated and to get logged in by providing a username and a password, by performing the Login action. The user can be logged out of the system too by performing the Logout action on the server.

The second service provided to the user on my Smart Automated Cloud Management Environment is a Database Service where the user can retrieve a file and analyse how many words that selected file contains. At the same time, the user can update with values the database by row and column.

The third service provided by my Smart Automated Cloud Management Environment is a VM service where the user can choose to see the Compute, Storage and General Purpose services available to him.

# Naming Services

Another way to allow my devices to efficiently connect is by making them directly locate and communicate with other components within my system in the local area network. In this matter, I will make use of a Java implementation called jmDNS library to send and receive packets over the network using the multicasting protocol that uses the service registration and discovery of the devices on the local area network (Laurent Doguin, 2015).

The use of jmDNS in my project is implemented on the server-side for services registration of each service implementation (jmdns.org 2020).

On the Database Server the output of jmDNS service looks like this in the console:

Service added: [ServiceInfoImpl@1208825205 name: 'database.\_http.\_tcp.local.' address: '(null):9093' status: 'NO DNS state: probing 1 task: null', has NO data empty]

Starting the Database Server with Port:9093

On the userService Server the output of jmDNS service looks like this in the console:

Service added: [ServiceInfoImpl@1125964210 name: 'userServer.\_http.\_tcp.local.' address: '(null):9092' status: 'NO DNS state: probing 1 task: null', has NO data empty]

Starting the UserService Server with Port:9092

On the VMService Server the output of jmDNS service looks like this in the console:

Service added: [ServiceInfoImpl@1787189503 name: 'vmService.\_http.\_tcp.local.' address: '(null):9091' status: 'NO DNS state: probing 1 task: null', has NO data empty]

Starting the VM Server with Port:9091

The discovery (jmDNS Listener) is implemented on the client-side which for this project is the GUI service (jmdns.org 2020).

# Remote Error Handling

Error handling is implemented along with the project code on the server-side for each service implementation of my Smart Automated Cloud Management Environment project. The remote error handling helps me as a developer to find a problem with the developed code when the application cannot execute properly the written code. To be able to anticipate these problems we can write code that accommodates these situations.

One method of dealing with errors in my code is by printing a message in the console terminal of the server with an error code just like one of the errors handling responses implemented in the userService server on my application where a user is being verified to see if the user is logged in first before logged out action is performed:

response.setResponseCode(99).setResponseMessage(

"... Sorry Logout Failed, user not logged in: " + userName);.

Another way that I deal with errors in my project code is by throw exceptions clause and using try-catch blocks and using finally Bloks. These two methods are being used as an example in my project in the Database Service implementation before returning a file, where a file is chosen first by the user and then is being inspected where a word count is being performed:

public int getFile(final BufferedReader reader) {

int file = 0;

try {

String line;

while ((line = reader.readLine()) != null) {

file += line.trim().split(" ").length;

}

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

if (reader != null) {

reader.close();

}

} catch (IOException e) {

e.printStackTrace();

}

}

return file;

# Client - Graphical User Interface

The user can access all these services via the Graphical User Interface (GUI). The GUI for this project was made using WinowBuilder, which is a plug-in tool for Eclipse IDE and was developed using java language. The GUI is also the client for each service implementation of my project that allows the user to view control and invoke the services on the server-side of the application.

# GitHub

My project is located on GitHub and the link for it is <https://github.com/DanielCostelNeagu/Distributed-Systems>. The project was regularly committed on GitHub after each major code implementation or revision. I believe this is a good practice to do so because at times I had to review previous code versions.

# References

Laurent Doguin (2015) *Let your Devices talk to each other*. [Online] Available at: https://blog.couchbase.com/let-your-devices-talk-to-each-other-p2p/ [Accessed 25 March 2021].

gRPC.io (2021) *A high performance, open source universal RPC framework*. [Online] Available at: https://grpc.io/ [Accessed 25 March 2021].

jmdns.org (2020) *Official home of the jmDNS library.* [Online] Available at: <https://github.com/jmdns/jmdns> [Accessed 28 March 2021].