

---

# SLIITLIFY

## Content Delivery Network

Distributed Computing – SE5090

<b>Weerakoon W.M.P.P.</b>	<b>MS21910968</b>
---------------------------	-------------------

# Table of Contents

<b>Table of Contents .....</b>	<b>ii</b>
<b>1. Introduction .....</b>	<b>1</b>
1.1 Purpose.....	1
1.2 Document Conventions.....	1
1.3 References.....	1
<b>2. Overall Description .....</b>	<b>2</b>
2.1 Project Perspective.....	2
2.2 Outline Approach.....	2
2.3 User Classes and Characteristics .....	3
2.4 Solution Design.....	3
2.4.1 Problem .....	3
2.4.2 Provided solution.....	4
2.5 Assumptions and Dependencies .....	5
<b>3. System Features .....</b>	<b>5</b>
3.1 System Features .....	5
3.1.1 Microservices .....	5
3.1.2 Algorithms.....	6
<b>4. Failure Scenarios &amp; Fault Tolerance. ....</b>	<b>6</b>
4.1 Failure scenarios .....	6
4.2 Fault Tolerance .....	6
<b>5. Other Requirements .....</b>	<b>7</b>
<b>Appendix B: Analysis Models .....</b>	<b>7</b>
	.....7
	....7

## Use case selection method

$$2 + 1 + 9 + 1 + 0 + 9 + 6 + 8 = 36$$

$$3 + 6 = 9$$

$$9 / 2 = 4 \Rightarrow \text{reminder } 1$$

**Selected Use case is use case1**

# 1. Introduction

## 1.1 Purpose

This is “SLIITLIFY” repost contains all the information about the project. The purpose of this document is to provide a consistent and complete description of the CDN and the developed solution and demonstrate the knowledge and expertise of distributed design patterns and algorithms.

## 1.2 Document Conventions

Acronym	Definition
CDN	Content Delivery Network
User, He/She	Someone who interacts with this system
UI	User Interface

## 1.3 References

*T. Iwata, T. Abe, K. Ueda and H. Sunaga, "A DRM system suitable for P2P content delivery and the study on its implementation," 9th Asia-Pacific Conference on Communications (IEEE Cat. No.03EX732), 2003, pp. 806-811 Vol.2, doi: 10.1109/APCC.2003.1274471.*

*J. Wijekoon, S. Ishida, E. Harahap and H. Nishi, "Service-Oriented Router-Based CDN System: An SoR-Based CDN Infrastructure Implementation on a Real Network Environment," 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops, 2013, pp. 742-747, doi: 10.1109/COMPSACW.2013.130.*

*M. Sung and C. Han, "A study on architecture of CDN(Content Delivering Network) with content re-distribution function," 2009 11th International Conference on Advanced Communication Technology, 2009, pp. 772-777.*

*KyoJin Hwang and DougYoung Suh, "Reducing perceptible IPTV zapping delay using CDN cache server," 2013 International Conference on ICT Convergence (ICTC), 2013, pp. 738-739, doi: 10.1109/ICTC.2013.6675467.*

## 2. Overall Description

### 2.1 Project Perspective

This section will give an overview of the whole system. The system will be explained in its context to show how the system interacts with the user and introduce the basic functionality of it. It will also describe all the phases of the system, problems and solutions, what functionality is available and explanation of each functionality. At last, the constraints and assumptions for the system will be presented.

### 2.2 Outline Approach.

This system will consist of two parts: the main web application and six servers. The web application mainly communicates with the user, and it is working as the client application. All the backend communications and data handling cover by the servers. Mainly this application contains six servers that are geographically separated.

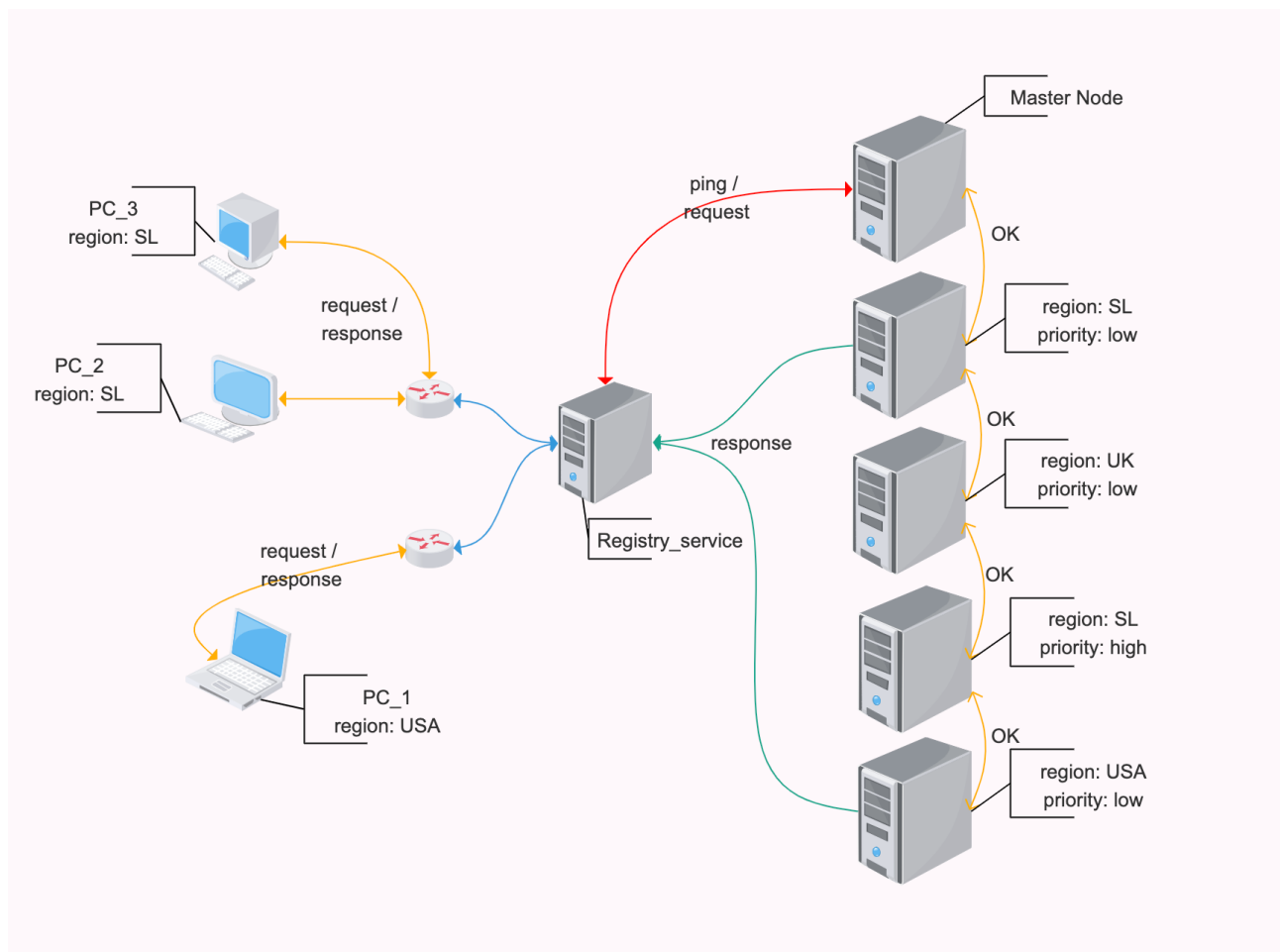


Figure 1: basic approach

The web application will be dealing with the user, user can upload their files and download files which are already cached in the folder inside each server. According to the user's perspective he/she can decide what type of action needs to be done (upload or download). If the user needs to download a file through the system, I have enabled the search option to search files. If a file available user can download a file to the device.

All the backend parts will be handled by the microservices. This application mainly used six microservices that communicate with each other. I introduce them as the nodes. These nodes have separate unique port address to introduce them to each other (currently these are running on local ports), such as 6060, 6061, 6062, 6063, 6064, and 6065. Each microservice has few attributes including name, priority, location configuration, and a few other.

The application has some restrictions about the,

all the data stored in the local machine due to server unavailability. Each service has a separate folder to store data. According to the user's region response node will be selected, then the node will provide a list of files available in the folder.

## 2.3 User Classes and Characteristics

There is only one type of user that interacts with the system: users of the application, restaurant owners, and administrators. The system will be working according to the user's requests.

## 2.4 Solution Design

### 2.4.1 Problem

The requested requirement is a content delivery system. Before developing the solution, I have investigated lots of available solutions on the internet. I saw lots of CDNs available, and they are providing more advanced services to the share files through the network. Few of real-world top CDNs are as follows,

- MaxCDN.
- Amazon CloudFront.
- CloudFlare.
- Incapsula.

The content delivery network (CDN) is a combination of a geographically distributed group of servers. They are working together to archive a common goal which is providing the fastest delivery for their contents through the internet. The CDN allows assets sharing, web pages, contents, and resources loading and handling web traffic of the site.

The requested solution for this assignment is to create a CDN to provide access to static files and content by the nearest location. There are geographically separated nodes that need to be created and they should be communicated with each other. Nodes can elect a new master node. The master node has the responsibility of handling user requests and cache copy of files between nodes.

## 2.4.2 Provided solution

- **Architecture of the system.**

This system has six separate nodes and one client-side web application.

All the separate nodes are running as separate microservices. Each microservices has a configuration file named config\_<node\_number>.json file. This file has the configuration properties of each node. Such as “node\_name”, “node\_priority”, “node\_region” and etc.

When the user requests a service, first that request reaches the registry service. Registry service is a separate microservice that is responsible for check the master node is available or not. If the master node available registry service will inform the master node to continue the user request. If the master node not available registry node requests all nodes to start the leader election process.

Once the leader election process started each node communicate among themselves to decide which of them will get into the "new leader" state. Once the leader elected it will handle the user requests.

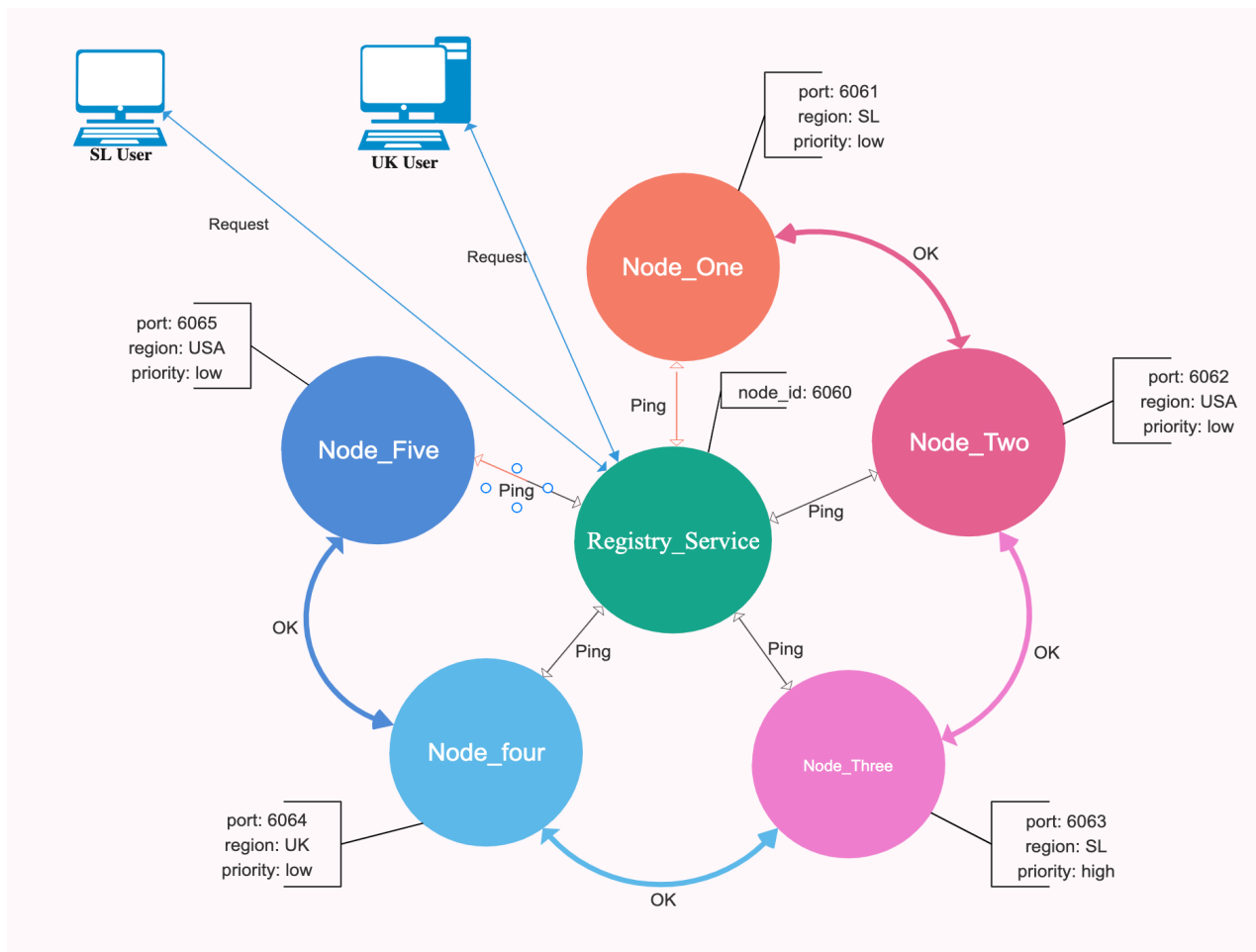


Figure 2 : high level architecture diagram

Now master node starts the identify the shortest distance by using the region. Each node shares their region which is already stored in config\_<node\_number>.json. When the master node collects all the regions from active nodes it will start another process to identify the distance. According to the type of the request by user (upload file or download file) master node will perform the actions available in the nearest node.

## 2.5 Assumptions and Dependencies

- The complete application was developed based on the assumptions. I assume all the microservices running on the local machine as the servers. Each node running on separate ports, and the ports are defined by myself. I assume all the microservices are working as geographically separated servers as needed.
- To implement the complete application, I use Eclipse and NetBeans ides, and the application tested on Windows 10 operating system.
- Leader Election algorithm (bully algorithm) and shortest distance (Paxos Algorithm), I read few articles related to get an idea about those algorithms and I have implemented them in my own way.

## 3. System Features

This section contains all of the functional and quality requirements of the system. It gives a detailed description of the system and all its features.

### 3.1 System Features

#### 3.1.1 Microservices

Each node of a system is a Microservice. All microservices are programmed by Eclipse Ide using spring boot technology. I have configured a port for each node, e.g.: 6060, 6061 .... 6065. All the nodes running on localhost (127.0.0.1).

- **Registry service. (Port: 6060)**

Registry service is the main service. Using registry service system check availability of Master node. This sends a ping request to the master node (after leader election, service stored the port id of the master node.). If the master node not available, it executes the “startElection()” methods to send all other nodes to notify the master node unavailable.

Also, this service directly interacts with the client-side of the application. When a user requests a service (upload or download file) request came to this service, according to the type of the request and the region of the user this service share information with the master node.

- **Nodes. (Ports: 6061 to 6065)**

All nodes include a config file, this config file has its configuration details.

e.g.: Node id: 6061, Priority: “High”, Region: “SL”.

Config file responsible to provide data for the node.

Each node can communicate along with them to check the availability of the nodes. If a node available, it will return an “OK” message as a response. It will collect by the requested node. If a response message is not returned from the expected node, it will be marked as not available.

All the nodes can share their regions and priorities with the master node. Then master node executes the “**selectNearByNode()**” method in the registry node. It will select the nearest node and returned it to the master node. Then master node will request the service from the nearest node.

- **User Interfaces.**

The web application consists of user interfaces. All the required data will be displayed here.

UI includes a search option to search files. When the user selected the requested file, it can be downloaded. If he/she needs to upload a file there is a separate section for that. According to the selection of the user information in the terminal section will be changed.

### 3.1.2 Algorithms

- **Leader Election Algorithm.**

The leader election is an important problem in a distributed system as data is distributed among different node which is geographically separated. Bully Algorithm is one of the most famous algorithms to select master. It was presented by “Gracia Molina” in 1982. Using bully algorithm information can be exchanged between nodes by transmitting messages to one another until an agreement is reached. Once a decision is made, a node is elected as the leader and all the other nodes will acknowledge the role of that node as the leader.

- **Paxos Algorithm**

Paxos is an algorithm for choosing a single value among multiple ones. All the nodes will get the same chosen value and the chosen value is indeed requested by a client. Here this algorithm helps to find the node which is available in the short distance of the user.

## 4. Failure Scenarios & Fault Tolerance.

Fault tolerance is if the system has a fault, what is the impact of that fault to the system. It measures how much tolerance has that specific problem.

### 4.1 Failure scenarios

A microservice or multiple microservices goes down.  
A microservice instance getting slow.

### 4.2 Fault Tolerance

- A microservice or multiple microservices goes down – As a solution for this, I run multiple instances in the same region.
- A microservice instance getting slow – using threads and setting the timeout to the spring RestTemplate can be solved this problem or use circuit breaker pattern. Assume one microservice is getting slow then other microservices temporarily stop sending



requests to that service. Once the problem is solved the microservices others will continue their requests.

## 5. Other Requirements

## Appendix B: Analysis Models

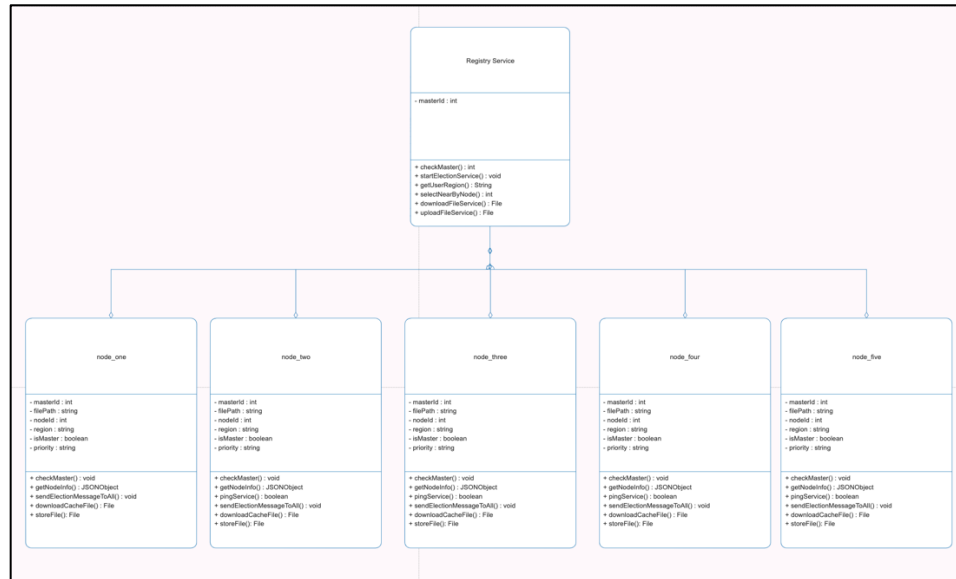


Figure 3: class diagram

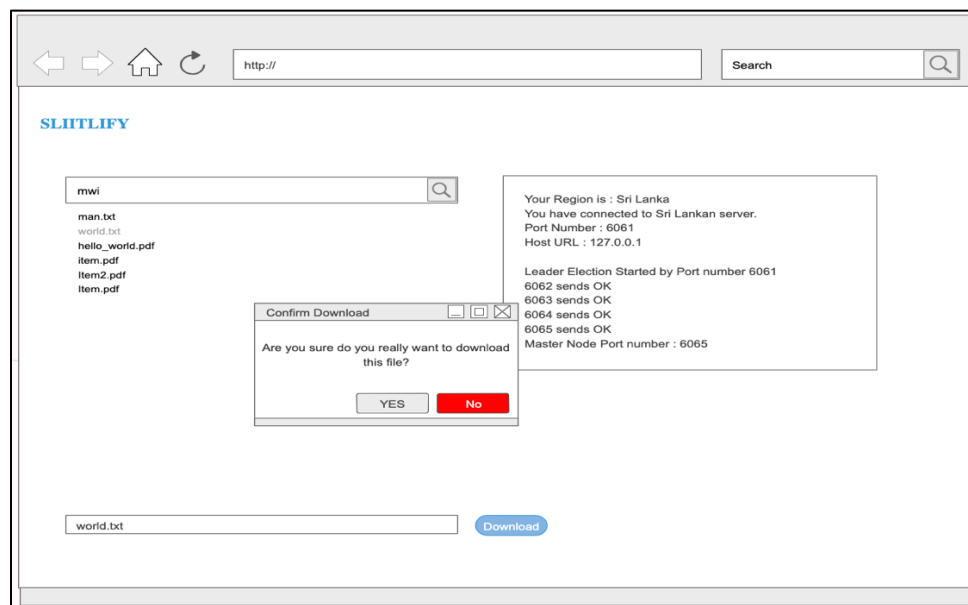


Figure 3: sample UI diagram

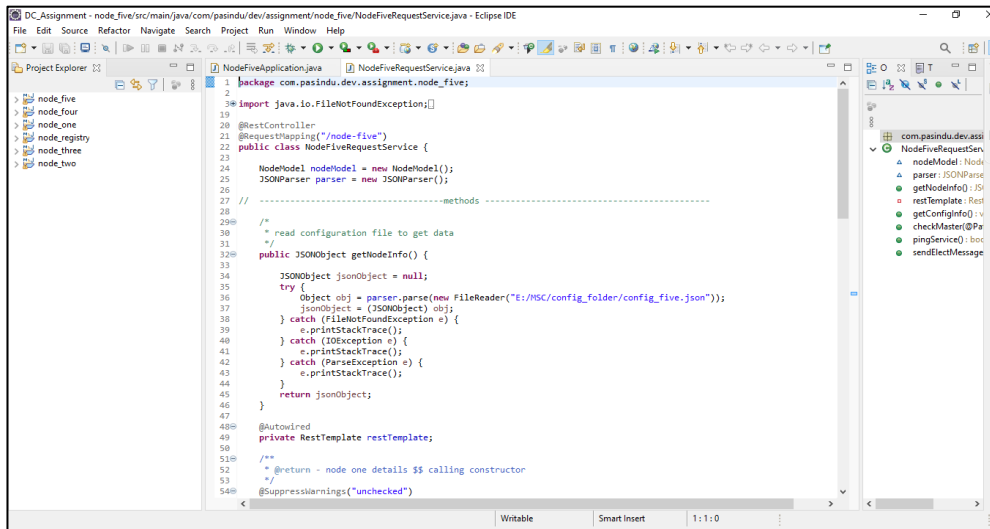


Figure 4 : Coding samples with folder structure



Figure 5 : Send election message to all node method

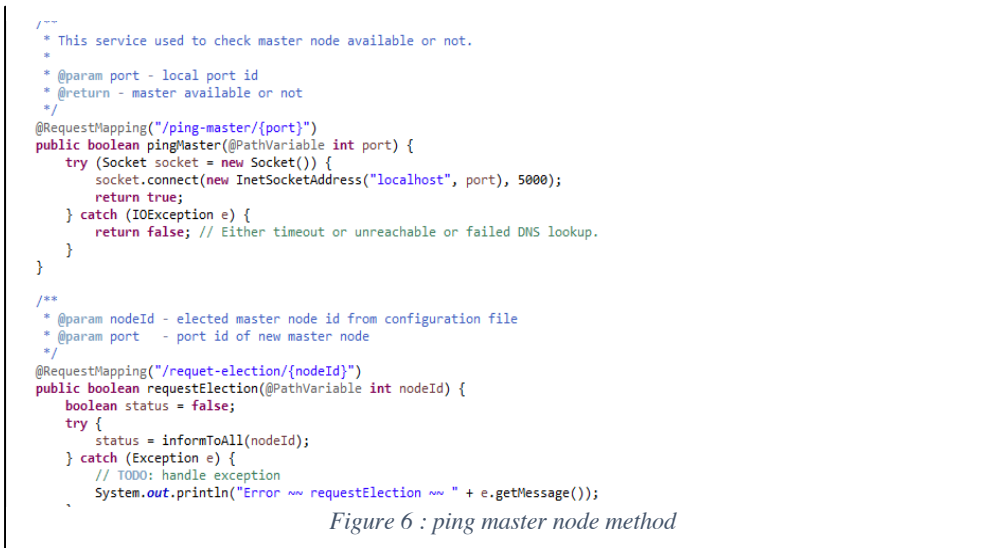
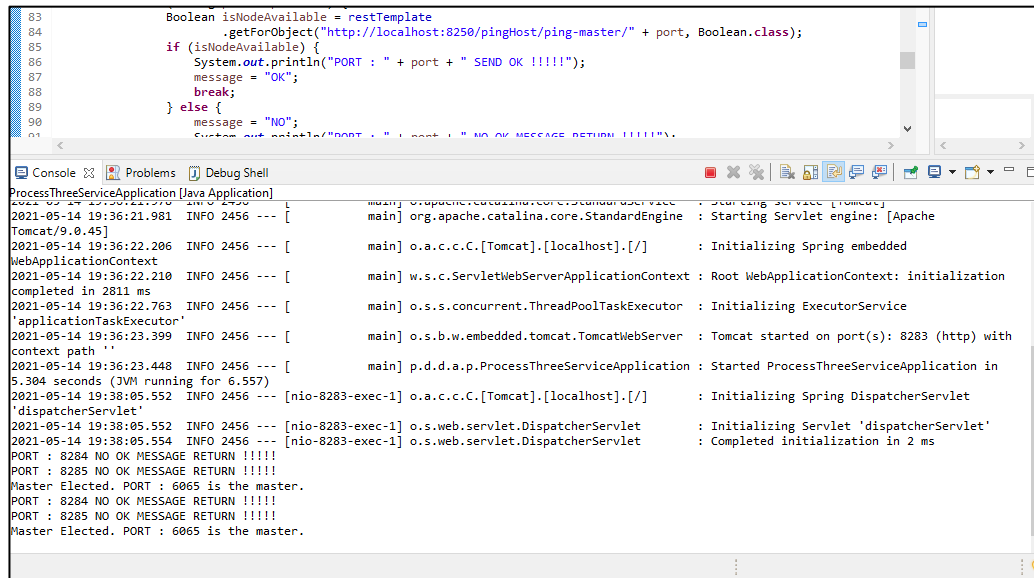


Figure 6 : ping master node method



The screenshot displays the Eclipse IDE interface. The top editor shows a Java class with a method `isNodeAvailable` that checks for a message on a specific port (8250) and returns a Boolean value. The console window below shows the execution output, including the starting of the Servlet engine, Spring embedded initialization, and the election of a master node (6065) from a group of nodes (8284, 8285, 6065).

```

83      Boolean isNodeAvailable = restTemplate
84          .getForObject("http://localhost:8250/pingHost/ping-master/" + port, Boolean.class);
85      if (isNodeAvailable) {
86          System.out.println("PORT : " + port + " SEND OK !!!!!");
87          message = "OK";
88          break;
89      } else {
90          message = "NO";
91          System.out.println("PORT : " + port + " NO OK MESSAGE RETURN !!!!!");
92      }

```

Console Output:

```

2021-05-14 19:36:21.981 INFO 2456 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.45]
2021-05-14 19:36:22.206 INFO 2456 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-05-14 19:36:22.210 INFO 2456 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2811 ms
2021-05-14 19:36:22.763 INFO 2456 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-05-14 19:36:23.399 INFO 2456 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8283 (http) with context path ''
2021-05-14 19:36:23.448 INFO 2456 --- [main] p.d.d.a.p.ProcessThreeServiceApplication : Started ProcessThreeServiceApplication in 5.304 seconds (JVM running for 6.557)
2021-05-14 19:38:05.552 INFO 2456 --- [nio-8283-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-05-14 19:38:05.552 INFO 2456 --- [nio-8283-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2021-05-14 19:38:05.554 INFO 2456 --- [nio-8283-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
PORT : 8284 NO OK MESSAGE RETURN !!!!!
PORT : 8285 NO OK MESSAGE RETURN !!!!!
Master Elected. PORT : 6065 is the master.
PORT : 8284 NO OK MESSAGE RETURN !!!!!
PORT : 8285 NO OK MESSAGE RETURN !!!!!
Master Elected. PORT : 6065 is the master.

```

Figure 7 : Sample result, Eclipse Ide