DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Higher Layer Telecommunication Systems
Content Delivery Network (CDN) - ELEC 6861

Fall 2023

PROJECT REPORT

ON

CONTENT DELIVERY NETWORK DESIGN AND IMPLEMENTATION

Course Instructor: Prof. Roch H. Glitho

Submitted by:

Mahalti Mohammed Sohail (40187593)

Ahamad Saifi (40235689)

Naga Sesha Sai Taraka Ramudu Anabathula (40233338)

Abdulkadir Garari (40258335)

Chodapuneedi Tarun Veera Raj Kiran (40230244)

Table of Contents

Abstract

The aim of the project is the deployment of a Content Delivery Network (CDN), a crucial element in the realm of modern high-bandwidth internet. The focal point of this initiative is the establishment of a CDN framework, encompassing an Origin Server and a series of surrogate servers within the CDN Provider Domain, all regulated by a central CDN controller. Executed in a localized environment with unique port numbers assigned to each server, the project utilizes Spring Boot as the backbone for its REST API infrastructure. The REST API plays a vital role in facilitating fundamental operations like GET and POST. A key innovation in this project is the adoption of the HTTP/2 protocol, surpassing the capabilities of the conventional HTTP 1.1, especially in terms of multi-streaming and enhanced security protocols including TLS 1.3 and SSL certifications. A self-signed certificate was employed for the purpose of demonstration. The architecture of the system features a novel mechanism where the CDN controller randomly selects surrogate servers. The user-end experience is showcased through a locally hosted HTML page, serving as the client interface for demonstrating the CDN's working .This report provides a thorough examination of the CDN project, explaining its structural design and workings.

Introduction

Content Delivery Networks (CDN) are Server based interconnections which enable the user to send and receive data which is stored on a server. Sometimes, due to Users being far or content not being instantly available in certain regions, some servers act as surrogate servers and fetch the data from a set of servers which contain all the data, in our case an Origin Server or a set of origin servers are called Content Provider Domain which contain all the data to be sent to the user. The intermediary servers in a CDN or the surrogate servers are called CDN Provider Domain. The requests coming from the user need to be redirected to a specific surrogate server which has minimum latency, this requires a CDN controller which is also a part of CDN Provider Domain. End-user Domain are the devices which can send and receive content from these surrogate servers. The following project is done in such a way that CDN controller randomly chooses a surrogate server to fetch data due to simplicity of the project.

The following project was done locally with varying port numbers for each server. Project was performed in Maven using spring boot REST API. REST API refers to Representation State Transfer Application Programming Interface. Which enables the user to send and receive data with packets which have commands like GET and POST. The operation is validated by postman and html page which fetches the data which in our project is a mp4 file streaming from the surrogate server. One of the requirements for this project was HTTP/2 protocol due to which security certificates are required. For the project's demo purposes a self signed certificate was created and used as an authentication to perform the send and receive operations.

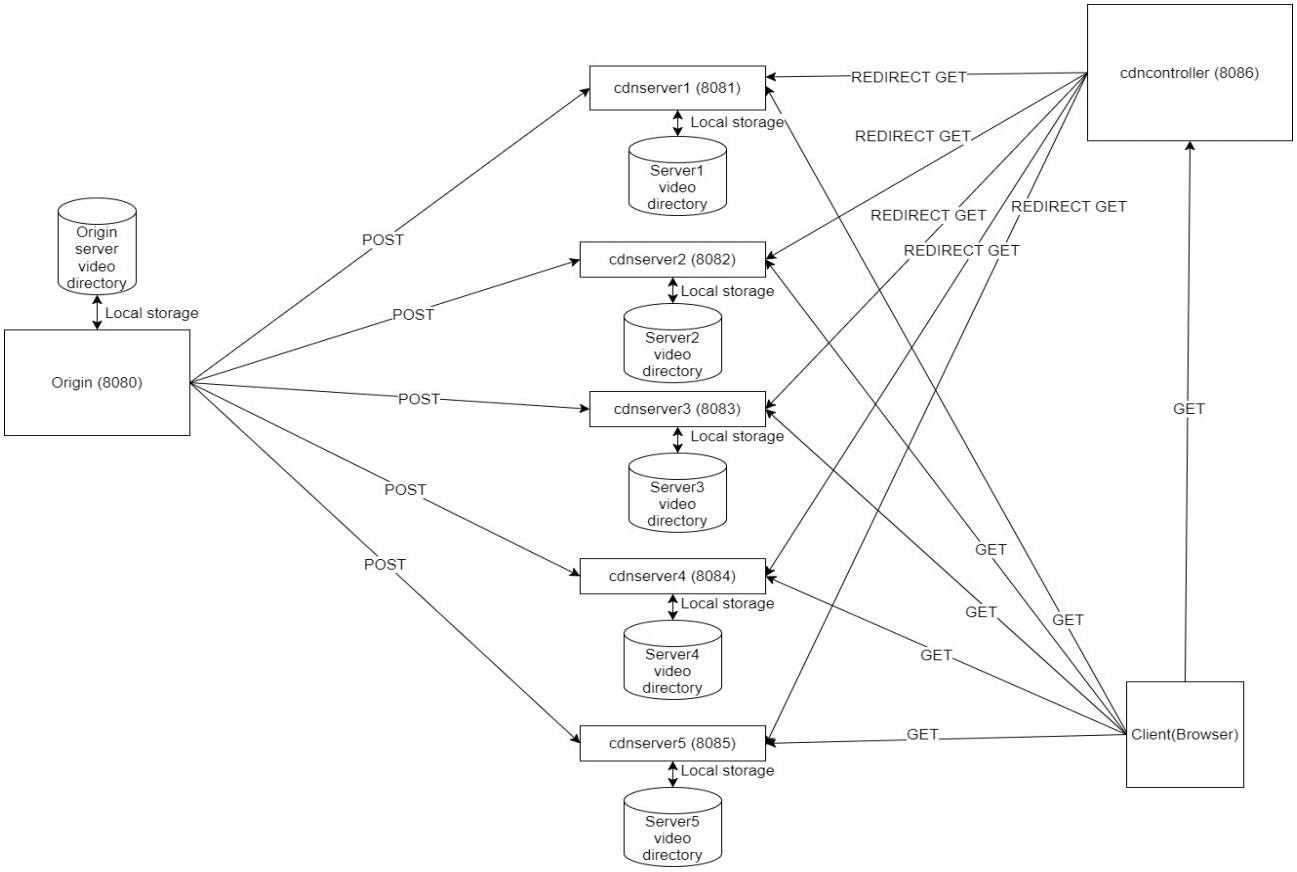

Figure 1. Content Delivery Network (CDN)

In Figure 1, The working of the Content Delivery Network is explained. The Origin server labeled with port 8080, which has a video directory in local storage. From this Origin server, there are POST requests directed to multiple CDN servers (cdnserver1 through cdnserver5), each designated with ports 8081 to 8085 respectively, and each with their own local storage and video directory. The CDN controller, operating on port 8086, is shown to redirect GET requests to these CDN servers. The flow lines indicate that the client (browser) can make GET requests to the CDN controller, which then randomly redirects these requests to one of the CDN servers.

Design Goals

1) Create an Origin server which can send data to selected surrogate server.
2) Create Replica servers capable of sending and receiving video data.
3) Create a CDN controller which redirects requests from clients.
4) Create a HTML page which displays the Video described in the API endpoint.
5) Perform data communication using HTTP/2 protocol.
6) Create a client which performs video playback for the video streamed using the API endpoint.

Technology and Software Used

Language used: Java
Java version: 19
Editor: Eclipse IDE
Server: Apache Tomcat built in Spring boot
Client: HTML
API endpoint checker: Postman

Rest API

Representational State Transfer Application Programming Interface enables user to create web services which follow REST Architecture. For this project a client-server system was chosen to perform operations in between servers. The operations which are possible in REST API are:

1. **GET**

This enables the user to fetch data from server using an URI, Uniform Resource Identifier which acts as an address for the data which the user wants to fetch.

2. **POST**

This enables the user to append new data from client using an URI, Uniform Resource Identifier which acts as an address to where the user wants to send the data.

Similarly, there are PUT, DELETE and so on, for our project only GET and POST were used to send and receive data.

Spring boot

Spring boot is a framework which runs on Tomcat server which enables to perform GET and POST using annotations which makes coding easier and it itself takes care of majority of the work. The following project takes advantage of this framework to build REST API. Using Spring boot, the tomcat server was utilized by this project.

Ports:

As the Project required was to be performed locally, to differentiate each server's URL, ports were allotted to each server. Generally, if a Servlet (part of a server) is hosted it obtains a new URL which is completely unique but when servlet is hosted locally, the IP address remains the same which is 127.0.0.1 or just localhost. For this project the ports are allocated as follows:

Table 1 Port Allocation

Port	Type of the server
8080	Origin Server
8081	Surrogate Server 1
8082	Surrogate Server 2
8083	Surrogate Server 3
8084	Surrogate Server 4

8085	Surrogate Server 5
8086	CDN Controller

Content Provider Domain

Origin Server:

In the following project, the role of Origin server is to send data to the specified replica server and to be constantly running. The Origin server sends the video through POST operation to the replica server. The Origin server asks the user for the server port which the user wants to send data to. The Port numbers range from 8081 to 8085.As per the requirements Origin server can only replicate data to the specified server but cannot do vice versa.

CDN Provider Domain

This Domain for the project is divided into 5 surrogate servers and a CDN controller in which the controller acts as an intermediary to the client and the surrogate servers. In real life scenario the surrogate servers are allocated to specific clients to minimize latency but in this project, we implement a Random select algorithm to choose any replica server randomly to serve the client. The surrogate servers must be capable of downloading and streaming data to devices, hence, GET and POST both are required in this part of the CDN.

GET:

The URI for the GET in the surrogate server is
https://localhost:{portnumber}/api/videos/{fileName}

Here “portnumber” and “fileName” are two variables. Port numbers vary from 8081 to 8085. The “filename” is name of the video which the user wants to fetch.

POST:

The URI is used by the origin server to replicate data onto to the replica server and uses to send the data. The URI is https://localhost:{portnumber}/api/videos/upload. Along with this server attached the video mp4 file so that the replica server receives it and stores it in its database.

The whole project was implemented in HTTP/2 Protocol

1) HTTP2 Protocol

Http/2 is an advanced version of HTTP 1.1 protocol which allows multi-streaming and also recommends the use of security during transfer of packets. HTTP/2 uses TLS for security and requires an SSL (Secure Socket Layer) certificate which enables encrypted communication between client and servers

2) Certificates

Generally, in real life scenario, the SSL certificates are required to have all the details pertaining to the organization, the name, the address etc. and also need to be signed and published so that the certificate is trusted by the browsers(clients). For the sake of

simplicity this project uses a Self-Signed Certificate to enable HTTP/2 as per the requirement.

3) CDN controller

The CDN controller acts an intermediary redirecting server which redirects requests coming from the clients and directs them to replica servers which in turn send them data, the CDN controller is hosted on the port 8086

End-user Domain

HTML page

Finally, the user uses a web browser to access the internet and receive data. Hence, the browser acts as a client to stream the video. The video is played on the website which is locally run but is streaming using HTTP/2 and this video can fetch the video randomly from different websites. The user fetches data from the CDN controller which redirects the data from the replica servers.

Result

The CDN project operates through a coordinated system where an Origin Server, situated on port 8080, replicates data to multiple surrogate servers across the CDN Provider Domain, each with a unique port number from 8081 to 8085. These surrogate servers, equipped to handle both GET and POST requests, store and manage the content delivery to clients. Central to the system's efficiency is the CDN controller, hosted on port 8086, which acts as an intermediary. It employs a random selection algorithm to direct client requests to a random surrogate server. The interaction between these components is governed by the HTTP/2 protocol, offering enhanced performance over the traditional HTTP 1.1. Additionally, the project integrates RESTful API services for seamless server-client communication and leverages the Spring Boot framework for server-side operations. The end-user experience is demonstrated via an HTML page that streams video content, showcasing the system’s capability to efficiently distribute high-bandwidth content, while SSL/TLS protocols, backed by a self-signed certificate, maintain security standards for HTTP\2.

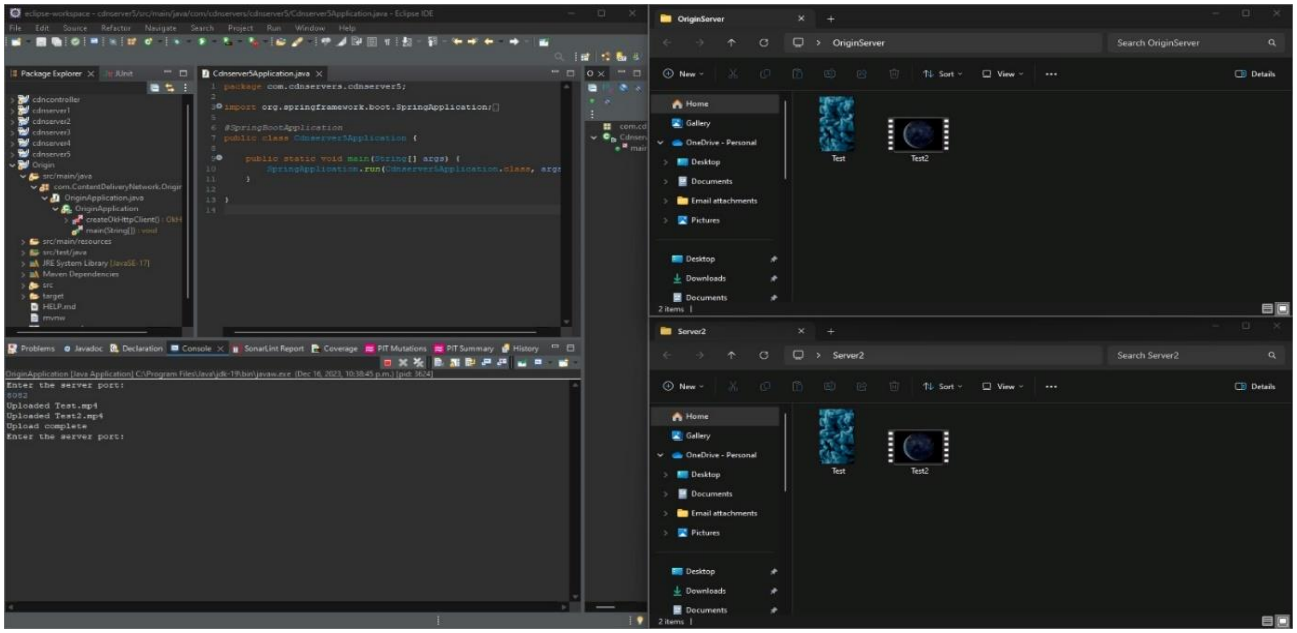

Figure 2 Origin server performing replication.

As shown in Figure 2. The developer's environment shows the replication feature of a Content Delivery Network (CDN). On the left, the Eclipse IDE reveals a project structure with an Origin server and several CDN servers, alongside a console output indicating successful video file uploads. On the right, file explorer windows display identical video files within the Origin and 'Server2' directories, signifying effective content replication from the origin to a surrogate server.

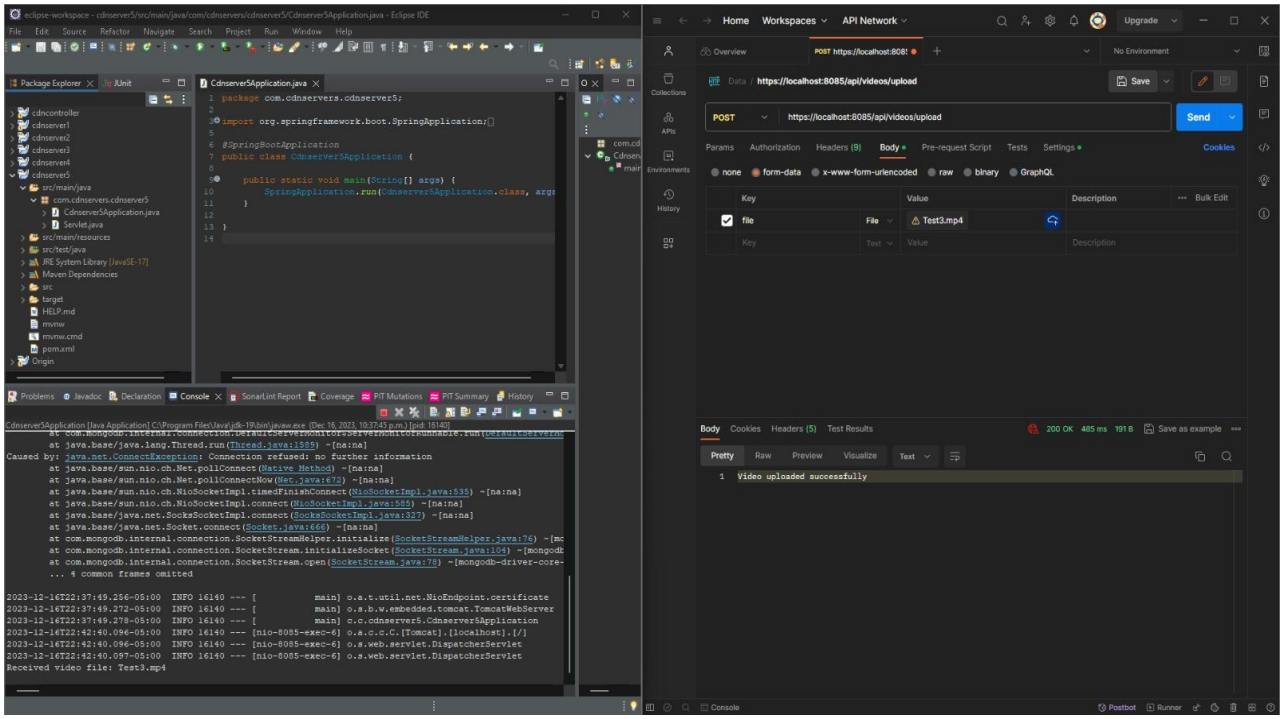

Figure 3 Working of replica server.

On the left side, the IDE (Eclipse) shows a project structure with server packages indicating a CDN server application, along with an open Java file and a console output which shows that

video file was received successfully. On the right side, Postman displays a successful POST request to a local server on port 8085 for video file upload, as evidenced by the '200 OK' status and the response message "Video uploaded successfully." This scenario suggests the process of uploading content to a surrogate server in a CDN environment, with the IDE monitoring the application's behavior and the API tool is testing the server's response to content uploads.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Video Streaming Example</title>
</head>
<body>

<h1>Video Streaming Example</h1>

<label for="videoFileName">Enter Video File Name:</label>
<input type="text" id="videoFileName" required>
<button onclick="watchVideo()">Watch Video</button>


<video width="1920" height="1080" controls id="videoPlayer">
    Your browser does not support the video tag.
</video>

<script>
function watchVideo() {
    var videoFileName = document.getElementById("videoFileName").value;
    var videoPlayer = document.getElementById("videoPlayer");
    videoPlayer.src = "https://localhost:8086/api/videos/" + videoFileName +".mp4";
}
</script>

</body>
</html>
```

Figure 4 Client code for a webpage

The following figure 4 shows, The HTML code for a webpage is designed to stream a video. It includes a video player with defined dimensions, sourcing its content from CDN controller's API endpoint. This setup indicates the use of a Content Delivery Network (CDN) to facilitate the streaming of a video file named "Test". The code is structured to be responsive and includes playback controls for an optimal viewing experience.

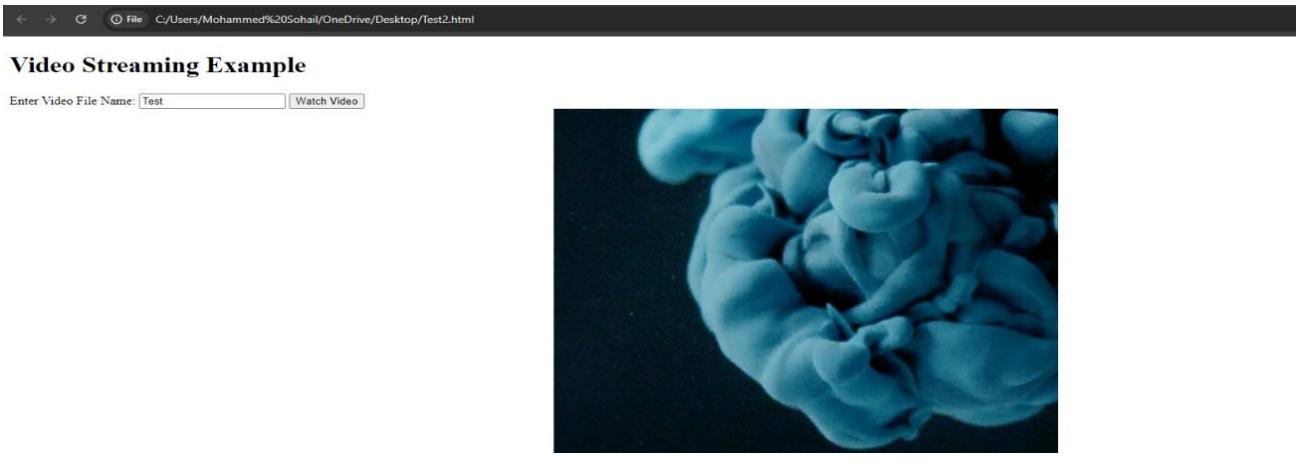

Figure 5 Working of Client

The visible content area shows video playback. This demonstrates the successful operation of a CDN system designed to facilitate efficient video streaming over the internet.

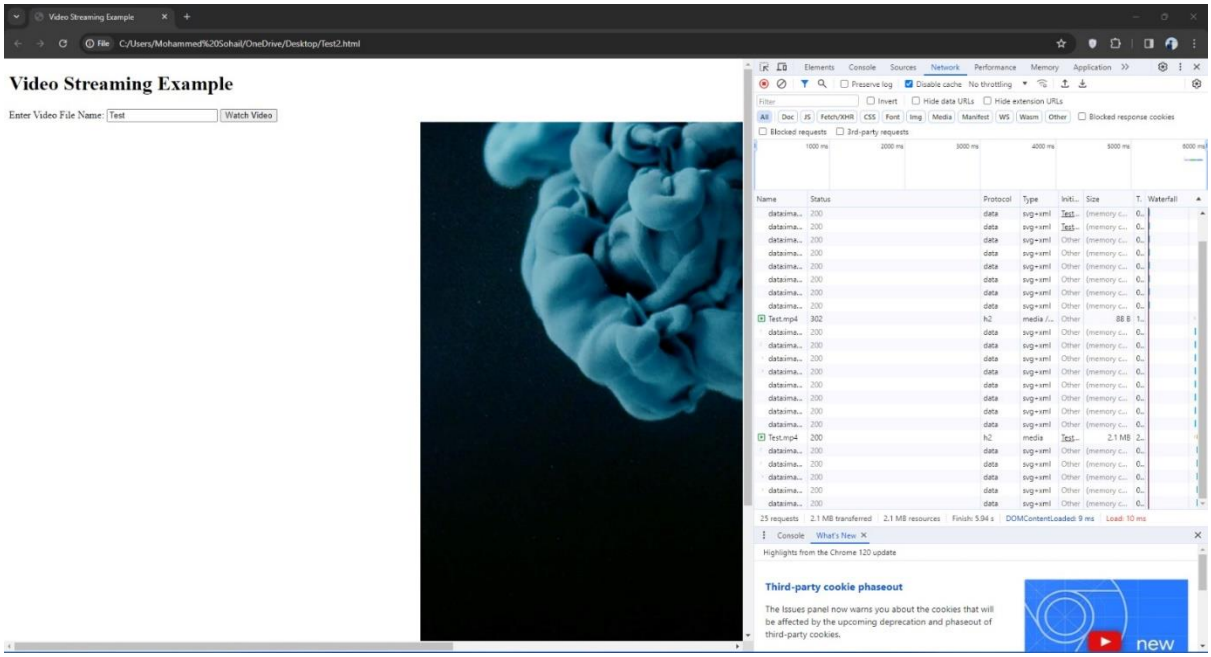

Figure 6 Client HTTP/2

The following Figure 6 displays a web browser's developer console, specifically on the Network tab, during a video streaming example. The console logs indicate multiple successful media data requests, shown by the 302 HTTP status codes, which suggest that the video content was redirected and efficiently streamed and received without errors. The protocol column for the video packet shows that HTTP\2 was used for this project by displaying "h2".

Conclusion

In conclusion, as we can see the following is Content delivery network which can stream videos to clients using HTTP/2 protocol which has many advantages as compared to HTTP 1.1. The following project shows how the Origin server sends the data to the replica servers. How the replica servers then receive and store the data. Following that whenever there is request for client to the controller the controller immediately redirects the client to one of the randomly selected servers to fetch and show the data. The following is the GitHub link to our project:

https://github.com/mohammedsohail-dev/ContentDeliveryNetworkProject

References

[1] https://docs.spring.io/spring-boot/docs/current/reference/html/

[2] Randy Connolly, Ricardo Hoar, "Fundamentals of Web Development", Pearson,2nd edition, (February 8, 2017)