



**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**

**ELEC 6861, ENCS 691K  
Fall 2021**

Project Report  
On

**CONTENT DELIVERY NETWORK DESIGN AND IMPLEMENTATION**

**Course Instructor**

**PROF. ROCH H. GLITHO**

**Submitted By**

**AJAY KUMAR LAKSHMIPURA VIJAYKUMAR (40168544)  
BHAVANI SHANKAR REDDY ANNU (40168191)  
KARANJOT SINGH KOCHAR (40161109)**

## **Table of Contents**

1. Abstract
2. Introduction
3. Content Delivery Network
  - 3.1 Benefits of using a CDN
  - 3.2 How does a CDN work
4. System Architecture Design
5. Technologies Used
  - 5.1 NGINX
  - 5.2 HTTP/2
  - 5.3 Open SSL
6. Working of the Application
7. Additional Features
8. Conclusion
9. References

## 1. ABSTRACT

The aim of the project is to design and implement a Content Delivery Network (CDN) which allows the end users to stream videos from different content providers. We have developed a web application-STREAMSTER which runs on CONTENT DELIVERY NETWORK. The user can stream videos on the go. There are two major components in our application, origin sever and the replica servers. The origin server replicates the content to the replica servers in advance. When the user requests for it, the content is served by the replica servers (in round robin manner in our application). We have used NGINX webserver which supports server push and load balancing. Additionally, it comes with the support to enable HTTP/2 protocol. To do that we should generate an SSL certificate which is done in OpenSSL.

## 2. INTRODUCTION

A British scientist named Tim Berners-Lee invented World Wide Web (WWW) in 1989. Initially, it was used to share information between scientists in universities and institutions. As of January 2021, we have more than 4.66 billion active users of internet worldwide. One of the major factors while using internet is the website load time. It refers to how long it takes for the contents of the web page to fully appear on screen. The contents include images, videos, and texts. The web page load time depends on various factors such as distance between server and host, file size, number of widgets and plugins used and efficient coding.

According to research, the pages with load times between 0-2 seconds has maximum e-commerce conversion rates. The website speed can hugely affect the sales of an e-commerce site. According to a survey by google, even a one second delay will decrease page views by 11% and reduce customer satisfaction by 16%. However, the user's experience can be improved by using a content delivery network.

## 3. CONTENT DELIVERY NETWORK

A content delivery network is a group of geographically distributed servers that cache and deliver the content over the internet. With the use of CDN, the web pages with high graphic content, images and videos can load faster. A CDN is used by major sites like Amazon, Netflix, and Facebook to reduce their traffic.

### 3.1 BENEFITS OF USING A CDN

The benefits of using a CDN highly depends on the size of content of the websites. However, there are few primary benefits.

- 1. Increasing content availability and redundancy:** Since CDN is a group of distributed servers, it can handle traffic better than origin servers.
- 2. Enhancing website load times:** As the user can get the content from the nearby replica server, he or she can experience faster loading times. Eventually this will increase the number of users for any website.

3. **Decreasing bandwidth costs:** With the help of caching and optimizations provided by the CDN, the data an origin server handles is reduced thus reducing the bandwidth consumption costs.
4. **Improving Website Security:** Distributed denial-of-service (DDoS) attack is an attempt to interrupt the normal traffic of target server or service by overwhelming it. CDN can be used to protect websites from these malicious attacks.

### 3.2 HOW DOES A CDN WORK?

A user in Spain wishes to access the website of a store in New York which is hosted on a server in North America. The user experience huge load time as the request must travel all the way from Spain to North America. To prevent this CDN stores a cached version of the New York website in the replica servers. The replica servers are responsible for delivering the content close to where the user is in Spain.

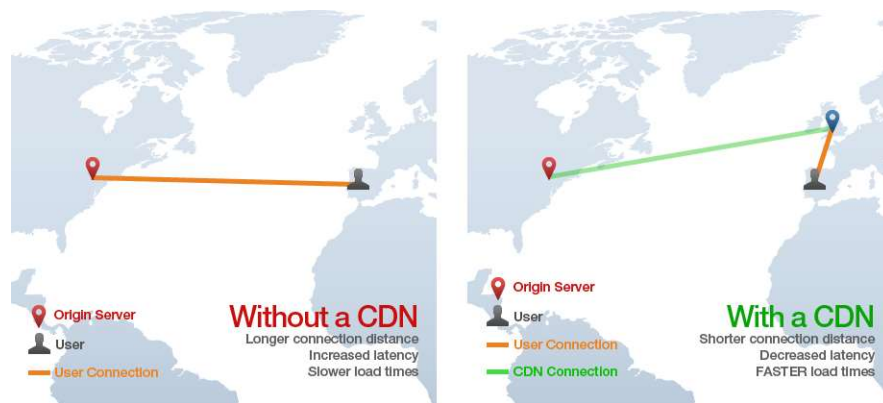


Fig. 1: With a CDN, the user can load the content faster from a replica server close to him.

## 4. SYSTEM ARCHITECTURE DESIGN

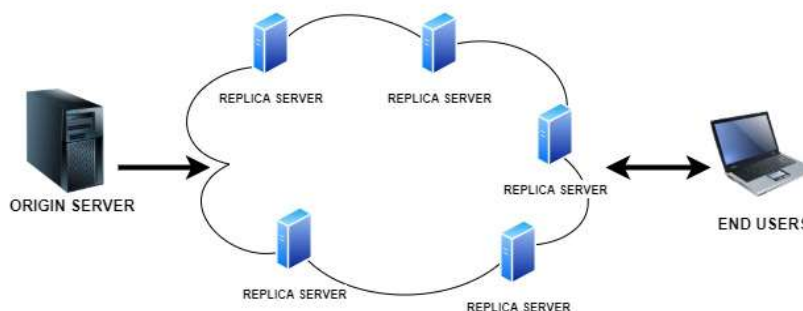


Fig. 2: System Architecture Design

The origin server replicates the content to the replica servers in advance. When the user requests for it, the content is served by the replica servers (in round robin manner in our application).

## 5. TECHNOLOGIES USED

Programming Language (Backend) – Python

Frontend – HTML, CSS, JavaScript

Framework – Django

IDE – PyCharm

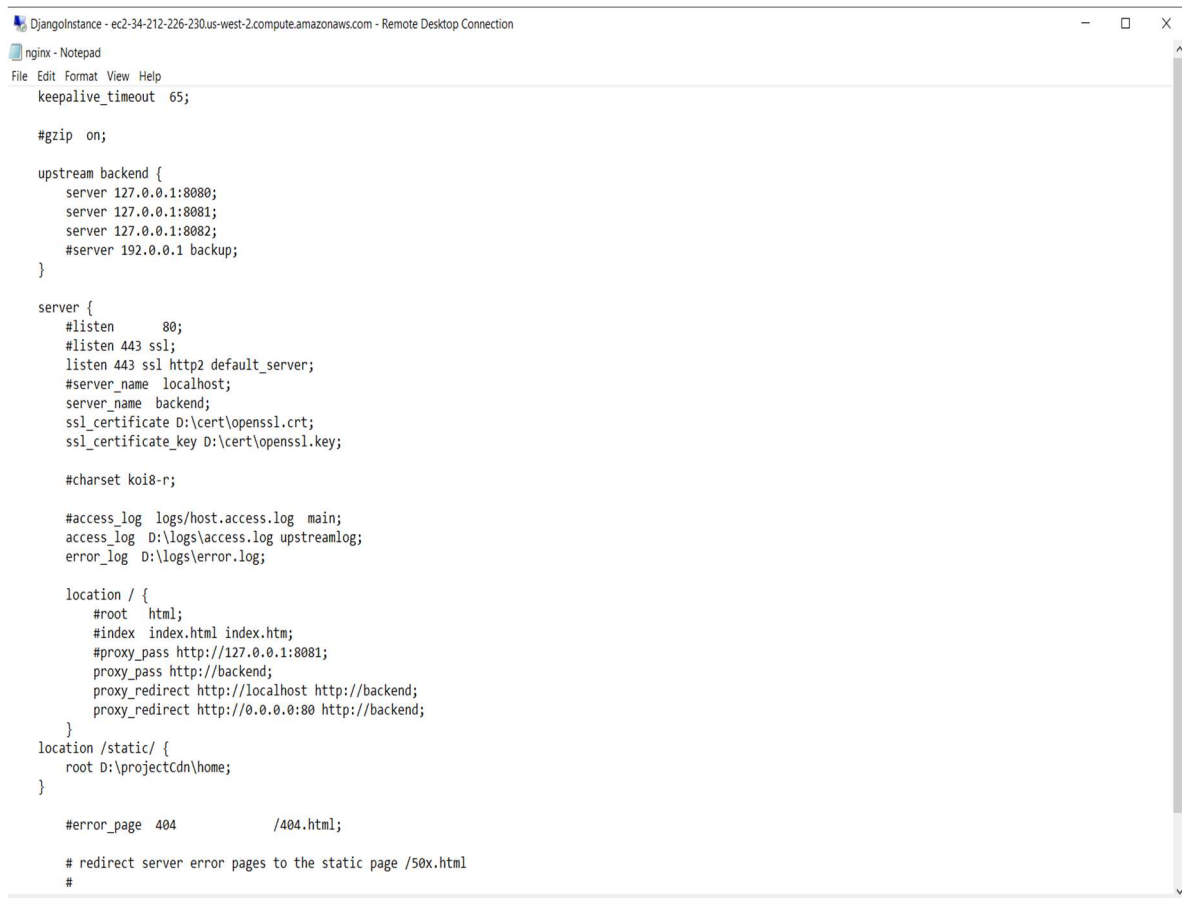
Web Server – NGINX

Application Layer Protocol – HTTP/2

Deployment – AWS EC2 Instance

### 5.1 NGINX

In our project, we have used NGINX in the backend as a web service used for replica servers. In general, NGINX is an open-source software used for web serving, reverse proxying, caching, load balancing, media streaming, and more. Due to its light-weight resource utilization and easy scalability it is widely used to aid web server for maximum performance and stability. NGINX is capable of handling higher volume of concurrent connections along with supporting various web components like streaming of various video formats.



```
keepalive_timeout 65;

gzip on;

upstream backend {
    server 127.0.0.1:8080;
    server 127.0.0.1:8081;
    server 127.0.0.1:8082;
    #server 192.0.0.1 backup;
}

server {
    #listen 80;
    #listen 443 ssl;
    listen 443 ssl http2 default_server;
    #server_name localhost;
    server_name backend;
    ssl_certificate D:\cert\openssl.crt;
    ssl_certificate_key D:\cert\openssl.key;

    #charset koi8-r;

    #access_log logs/host.access.log main;
    access_log D:\logs\access.log upstreamlog;
    error_log D:\logs\error.log;

    location / {
        #root html;
        #index index.html index.htm;
        #proxy_pass http://127.0.0.1:8081;
        proxy_pass http://backend;
        proxy_redirect http://localhost http://backend;
        proxy_redirect http://0.0.0.0:80 http://backend;
    }
    location /static/ {
        root D:\projectcdn\home;
    }

    #error_page 404 /404.html;

    # redirect server error pages to the static page /50x.html
    #
```

Fig. 3: Screenshot showing NGINX configuration

## 5.2 OpenSSL

In our project, we have used OpenSSL as a software library for applications to secure communications over computer networks. SSL (Secure Socket Layer) and TLS (Transport Layer Security) are protocols for establishing authenticated and encrypted links between networked computers. The main function of SSL is handling compression and encryption of data, after the secure connection has been established. HTTPS (Secure HTTP) is basically HTTP over SSL, and it uses port 443 to access the web content.

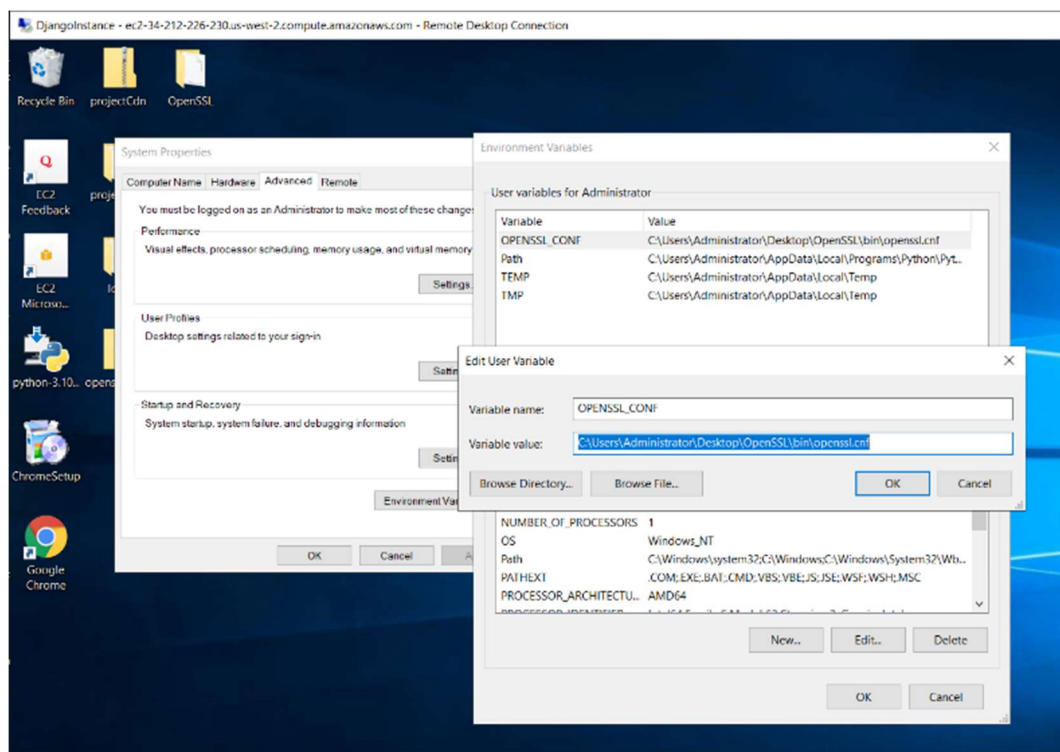


Fig. 4: Screenshot showing implementation of OpenSSL certificates

## 5.3 HTTP/2

In our project, we are implementing HTTP/2 (originally named HTTP/2.0). HTTP network protocol was majorly revised as HTTP/2 used by the World Wide Web. Stream multiplexing, Server Push which means that the server pushes data to the client even before client asks for it, and data compression of HTTP headers to make efficient use of network resources are the prominent features of HTTP/2. HTTP/2 tries to address the head-of-line blocking problem of HTTP/1 and it does at transaction level. But since, all the streams are multiplexed over single TCP connection, any packet level head-of-line blocking of the TCP stream will block all the streams that are accessed via that connection.

## 6. WORKING OF THE APPLICATION:

Our web application STREAMSTER was developed using Django framework. NGINX webserver is the heart of our application which does server push, load balancing and enables the HTTP/2 application layer protocol to our web application. Additionally, by deploying our web application on Cloud, we were able to fulfill all the design goals. Below is the screenshot of our web application- Streamster.

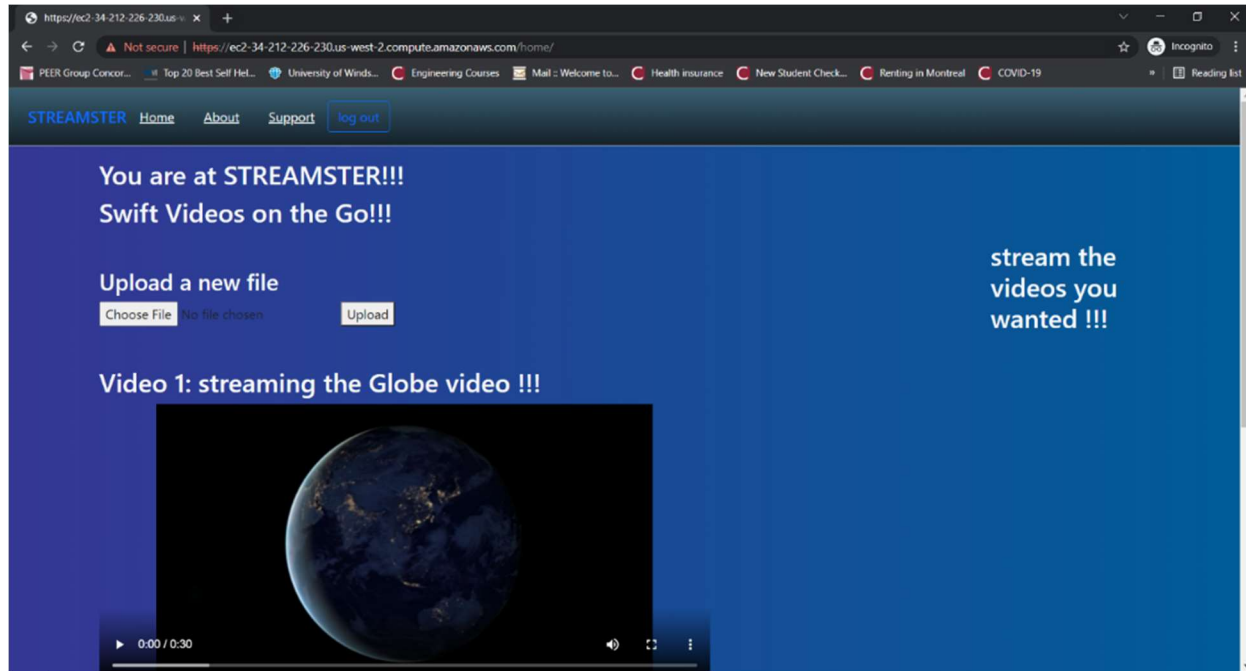


Fig. 5: Our Web application – STREAMSTER

### a. Server Push

Server Push is one of the specifications of HTTP/2 application protocol. It is a mechanism in which the server will push the content to the clients in advance, thinking that the clients might request for it. NGINX web server comes with the server push feature. By configuring the conf file of NGINX, we were able to accomplish the server push design goal. This is evident in the access log screenshot mentioned below (Fig 7).

### b. Controller (Algorithm)

The NGINX web server does the load balancing (controller), by default it uses ROUND ROBIN algorithm. It is one of the simplest algorithms to distribute the client requests to a group of servers. This is also evident in the access log screenshot in which the first request goes to the first replica server, next to another replica server, and finally the third replica server. After all the three replica servers get their client requests, again it starts from the first replica server. This mechanism is followed until the client stops requesting for the content.



### c. Redirection of end-user request by the controller

When the client requests for the content the controller that is the load balancer in the NGINX webserver can serve the request from the servers to the users. This is also evident in the access log screen shot mentioned below (Fig 7).

### d. Video Streaming at the user end

The end user can stream the video from the STREAMSTER web application without any delay. The below screenshot shows the same.



Fig. 6: Screenshot of Video Streaming

### e. Use of HTTP/2 application layer protocol

NGINX webserver supports HTTP/2 protocol. This application protocol comes with latest features such as server push and stream multiplexing. We were able to run our web application in the HTTP/2 protocol which is evident in the Fig 7.

```
DjangoInstance - ec2-34-212-226-230.us-west-2.compute.amazonaws.com - Remote Desktop Connection
access - Notepad
File Edit Format View Help
backend to: 127.0.0.1:8080 {GET / HTTP/2.0} upstream_response_time 0.265 request_time 0.265
backend to: - {GET /static/home/main.css HTTP/2.0} upstream_response_time - request_time 0.000
backend to: - {GET /static/streaming.jpg HTTP/2.0} upstream_response_time - request_time 0.312
backend to: 127.0.0.1:8081 {GET /home HTTP/2.0} upstream_response_time 0.016 request_time 0.016
backend to: 127.0.0.1:8082 {GET /home HTTP/2.0} upstream_response_time 0.157 request_time 0.157
backend to: - {GET /static/home/main.css HTTP/2.0} upstream_response_time - request_time 0.000
backend to: - {GET /static/video.mp4 HTTP/2.0} upstream_response_time - request_time 0.672
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 0.781
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 0.188
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 0.515
backend to: 127.0.0.1:8080 {GET /favicon.ico HTTP/2.0} upstream_response_time 0.359 request_time 0.438
backend to: - {GET /static/video.mp4 HTTP/2.0} upstream_response_time - request_time 1.516
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 19.594
Replica Server 1 127.0.0.1:8082 {GET / HTTP/1.1} upstream_response_time 0.016 request_time 0.016
Replica Server 3 127.0.0.1:8080 {GET /static/home/main.css HTTP/2.0} upstream_response_time 0.391 request_time 0.391
backend to: - {GET /static/streaming.jpg HTTP/2.0} upstream_response_time - request_time 0.000
backend to: 127.0.0.1:8081 {GET /favicon.ico HTTP/2.0} upstream_response_time 0.453 request_time 0.453
backend to: 127.0.0.1:8082 {GET / HTTP/2.0} upstream_response_time 0.109 request_time 0.109
backend to: - {GET /static/home/main.css HTTP/2.0} upstream_response_time - request_time 0.000
backend to: - {GET /static/streaming.jpg HTTP/2.0} upstream_response_time - request_time 0.000
backend to: 127.0.0.1:8080 {GET / HTTP/2.0} upstream_response_time 0.016 request_time 0.016
backend to: - {GET /static/home/main.css HTTP/2.0} upstream_response_time - request_time 0.000
backend to: - {GET /static/streaming.jpg HTTP/2.0} upstream_response_time - request_time 0.359
backend to: 127.0.0.1:8081 {GET /favicon.ico HTTP/2.0} upstream_response_time 0.016 request_time 0.016
backend to: 127.0.0.1:8082 {GET /home HTTP/2.0} upstream_response_time 0.015 request_time 0.015
backend to: 127.0.0.1:8080 {GET /home/ HTTP/2.0} upstream_response_time 0.016 request_time 0.016
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 0.422
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 0.187
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 0.625
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 0.234
backend to: - {GET /static/ocean.mp4 HTTP/2.0} upstream_response_time - request_time 0.234
```

Fig. 7: Screenshot showing the access log file



## f. Deployed in Cloud

Our project has been deployed in Amazon Web Services Cloud Computing Platform that is AWS EC2 Instance. One of the Amazon Web Services (AWS) clouds, which can act as a virtual server that provides scalable computing capacity. One of the main reasons to use the AWS EC2 services is that they eliminate the need to invest in hardware and hence one can develop and deploy applications faster. Instances are created from Amazon Machine Images (AMI). Below screenshot show the application running from a remote user.

The link for our deployed CDN project is:

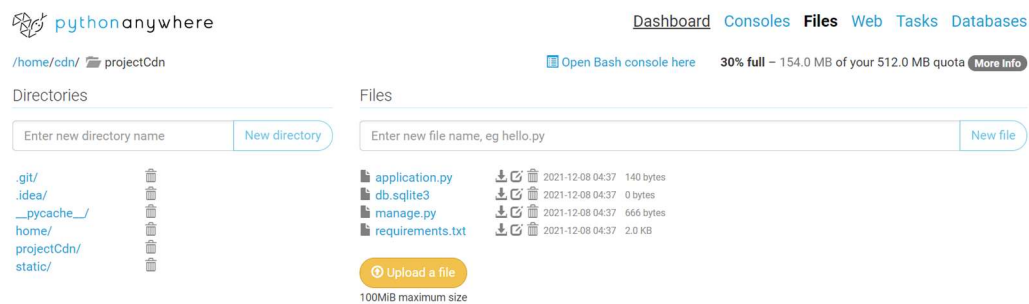
Public IPv4 address: 34.212.226.230

Public IPv4 DNS: <https://ec2-34-212-226-230.us-west-2.compute.amazonaws.com>

## 7. ADDITIONAL FEATURES

We have implemented few other additional features such as:

- a. Deployed our application in pythonanywhere- a macro-PaaS (Platform as a Service)  
Below Screen shot shows deployed application in pythonanywhere.



Link to access our application: <http://cdn.pythonanywhere.com/home/>

- b. Video Download option for offline viewing.
- c. Playback Speed Control.

## 8. CONCLUSION:

Content Delivery Network enhances website security, website load times and increases content availability. By designing and implementing CDN to our web application we were able to achieve these features. We have an origin server, replica servers and NGINX web server. The NGINX web server does server push and load balancing, thus fulfilling the design goals. Additionally, with the generation of SSL certificate the webserver makes the web application to run on HTTP/2 application layer protocol. To make our web application available to everyone we deployed it in Amazon EC2 Instance and Pythonanywhere. NGINX also supports HTTP/3 application protocol hence further this new application layer protocol can be added to our web application.

## 9. REFERENCES

1. <https://docs.djangoproject.com/en/4.0/>
2. <https://nginx.org/en/docs/>
3. <https://docs.python.org/3/>
4. <https://www.openssl.org/docs/manmaster/man1/openssl.html>
5. <https://docs.aws.amazon.com/>
6. <https://gtmetrix.com/>