# Image compressor WebApp

*B. TECH SEM – VI Cloud Computing Lab Project*
*Dept. of Computer Science & Engineering*

By

| | |
|---|---|
| **Vipul Chaudhary** | **20BCP055** |
| **Rushi Thakkar** | **20BCP045** |
| **Shrey Shah** | **20BCP042** |

**Under the Supervision Of**

**Dr. Shivangi Surati**

**SCHOOL OF TECHNOLOGY**
**PANDIT DEENDAYAL ENERGY UNIVERSITY**
**GANDHINAGAR, GUJARAT, INDIA**
**May 2023**

# ABSTRACT

The main objective of this project is to provide users with an easy-to-use tool to reduce the size of their digital images, making them more manageable for storage, transfer and sharing purposes. This app will feature a user-friendly interface and allow users to upload and compress their images with a single click. The app will utilize cloud infrastructure for image storage and processing, making it possible to process large image files in a timely and efficient manner. The resulting compressed images will retain the high-quality visual appearance while being significantly smaller in size, making it possible to store, transfer and share more images.

The "Image Compressor App" is a cloud-based application that will be created using the JavaScript programming language. The user uploads an image, which we then process and compress on the server before sending back to the user so he may download it. Users of the software will also be able to change the compression level, allowing them to compromise image quality in favor of smaller file sizes. By doing so, users can choose the level of compression that will best suit their requirements and achieve the ideal balance between image quality and file size.

# INDEX

# INTRODUCTION

In today's fast-paced world, web applications have become an integral part of our daily lives. With the advent of high-speed internet and cloud computing, users now demand applications that can process and deliver data in real-time. One such application that is gaining popularity is image compression. With the increasing size of digital images, compressing them has become essential to ensure fast and efficient data transmission over the internet.

To meet this growing demand, we are developing a web application that will allow users to compress their images on the fly. The application will use a compressing algorithm that runs on an AWS EC2 instance server, which will ensure fast and reliable processing of images. Users will be able to upload their images to the application, which will then be compressed using the algorithm and returned to the user in a compressed format. This application will be a valuable tool for anyone who needs to compress images quickly and efficiently, without the need for specialized software or technical skills.

# METHODOLOGY

1. **Choosing algorithm**

   We go through many algorithms and explored various techniques on how to utilize the cloud for image compressing, at last we get one way to do that image compression and we choose the algorithm 'sharp' provided by npm in NodeJS to compress your image. It is feasible algorithm that can be deployed on cloud and we can utilize it for our compression application.

2. **Choosing technology for frontend interface**

   There are several reasons why choosing a frontend technology like ReactJS for a compression image app developed using Node.js might be beneficial:

   - **User experience**: Frontend technologies like ReactJS can help create a better user experience for the compression image app. ReactJS is known for its fast-rendering speed, which can help the app load quickly and respond faster to user interactions. This can lead to a smoother and more responsive user experience.
   - **Easy to build interfaces**: Frontend technologies like ReactJS make it easier to build user interfaces for the compression image app. ReactJS uses a component-based architecture, which allows developers to build reusable UI components that can be easily combined and reused throughout the app. This can lead to faster development times and more maintainable code.
   - **Better performance**: Using a frontend technology like ReactJS can improve the performance of the compression image app. ReactJS uses a virtual DOM, which allows it to update only the parts of the UI that have changed, instead of updating the entire UI. This can result in faster rendering times and better overall performance.
   - **Ecosystem and community support**: ReactJS has a large and active community, which means that there are plenty of resources available for developers to learn from and use. Additionally, ReactJS has a large ecosystem of third-party libraries and tools that can help developers build better compression image apps more easily.

3. **Choosing cloud service for algorithm hosting**

   There are several reasons why Amazon EC2 can be a good choice for hosting a NodeJS image compression algorithm:

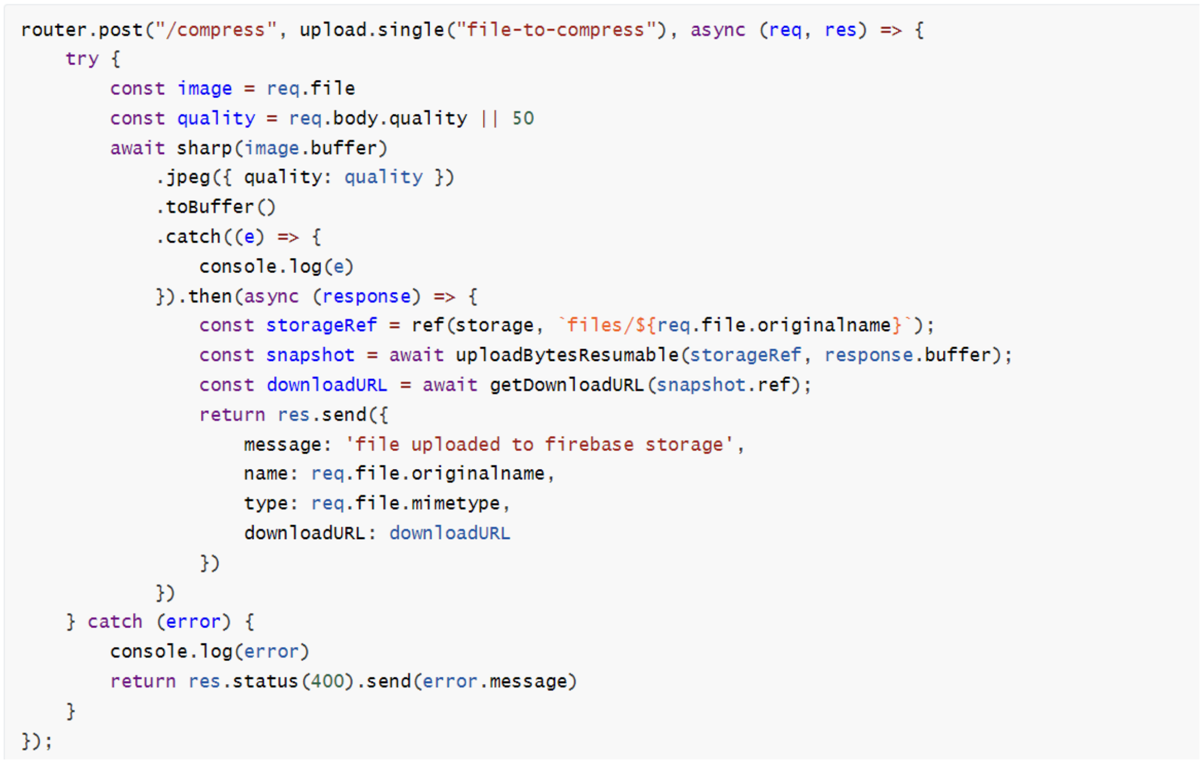
   1. Scalability: EC2 provides the ability to easily scale up or down the computing resources as per the requirements of the application. This is particularly important when dealing with image processing as it can be resource-intensive.
   2. Flexibility: EC2 offers a wide range of instance types that can be customized to suit specific needs, including memory, CPU, and storage. This allows you to choose an instance type that best suits your requirements.

3. Security: EC2 provides a secure environment to host your application, including options for network isolation and encryption of data in transit and at rest.
4. Integration with other AWS services: EC2 can be easily integrated with other AWS services such as Amazon S3 for storing and retrieving images, Amazon CloudFront for content delivery, and Amazon Elastic Load Balancer for load balancing.
5. Cost-effectiveness: EC2 offers a pay-as-you-go pricing model, which means you only pay for the computing resources you use. This can be particularly cost-effective for applications with variable traffic and resource needs.

# IMPLEMENTATION

## Created an NodeJS Application for image compression

We need to create one express (NodeJs) API to communicate with the image compression algorithm using any frontend interface. So we created one express API that take image as input and will return the compressed image in return to the user.



```
router.post("/compress", upload.single("file-to-compress"), async (req, res) => {
    try {
        const image = req.file
        const quality = req.body.quality || 50
        await sharp(image.buffer)
            .jpeg({ quality: quality })
            .toBuffer()
            .catch((e) => {
                console.log(e)
            }).then(async (response) => {
                const storageRef = ref(storage, `files/${req.file.originalname}`);
                const snapshot = await uploadBytesResumable(storageRef, response.buffer);
                const downloadURL = await getDownloadURL(snapshot.ref);
                return res.send({
                    message: 'file uploaded to firebase storage',
                    name: req.file.originalname,
                    type: req.file.mimetype,
                    downloadURL: downloadURL
                })
            })
    } catch (error) {
        console.log(error)
        return res.status(400).send(error.message)
    }
});
```

This API has one endpoint of /compress that will take image as input from request body.

Now once the image is compressed the image will be stored into cloud storage and public link will be return as response and user will be able to download it

## Creating user interface using react

First, we created one simple react app using ‘npx create-react-app client’ command

Then we added the required components for compressing image such as image input form, image quality input box and one button to download the image.

This “Ready to compress” message will be changed to “Network error” if we have any issue with our cloud service, that will happen one in a lakh time but we need to inform user that our algorithm is currently down mode, you may try letter.

Here once user select the image, then he can add his compression ration between 0 to 100 then request will be made to cloud for image compression and once the compression is done user will be able to see the download the image.

It will take very less time to compress the image.

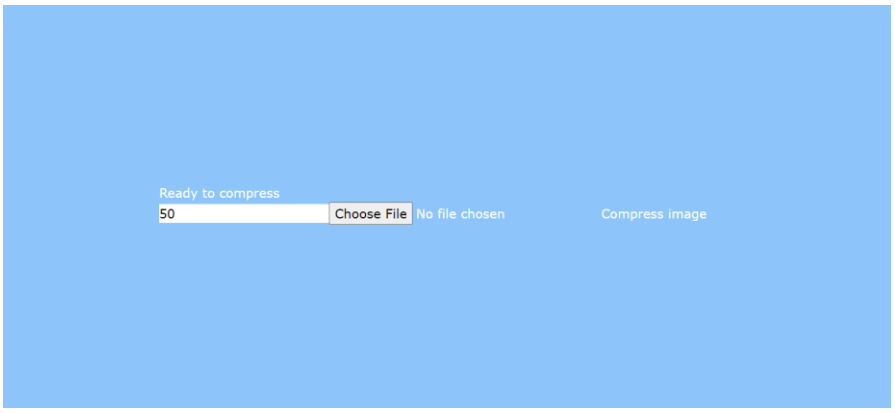


*Figure 1 Application interface*

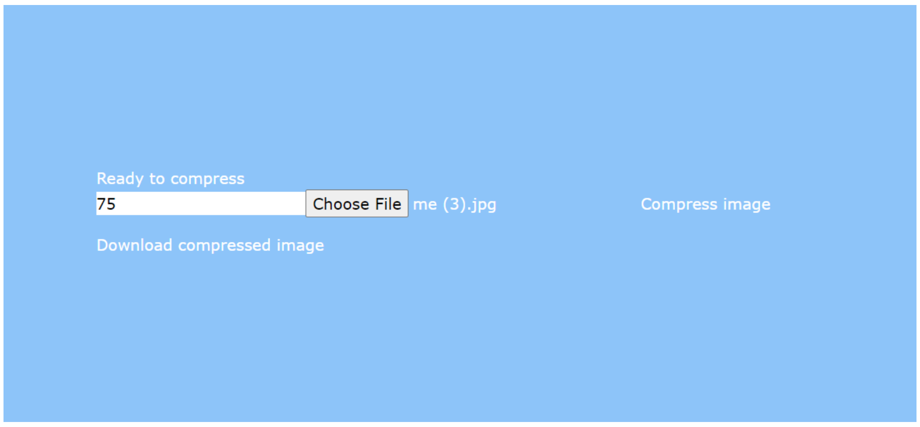


*Figure 2 Image compressed*

At this time, we are running our algorithm on local machine

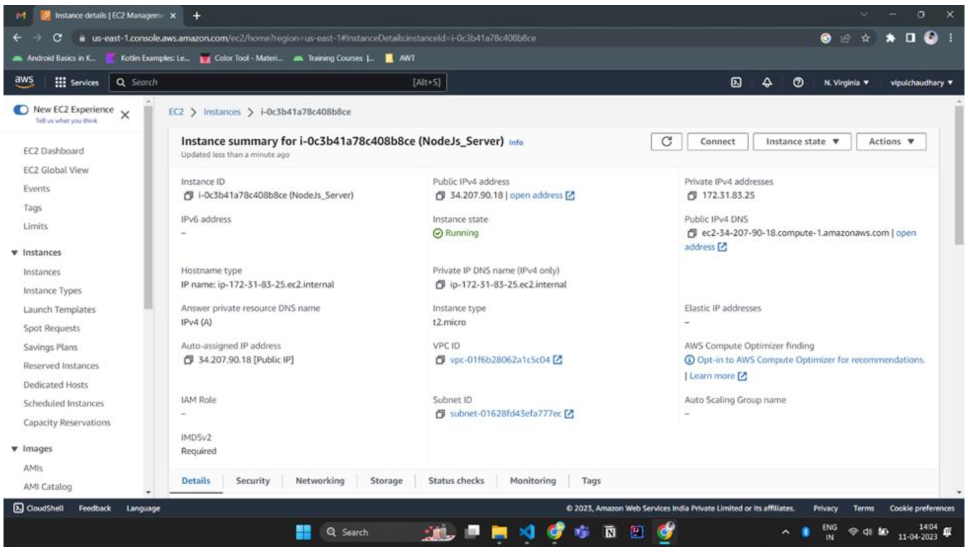
Let's now host it onto AWS EC2 machine

## <u>Hosting Algorithm on AWS</u>

Created AWS EC2 Instance.

Keeping all settings default, just allowing SSH access at port 22, that will be useful for algorithm compression settings.

Downloaded. pem key

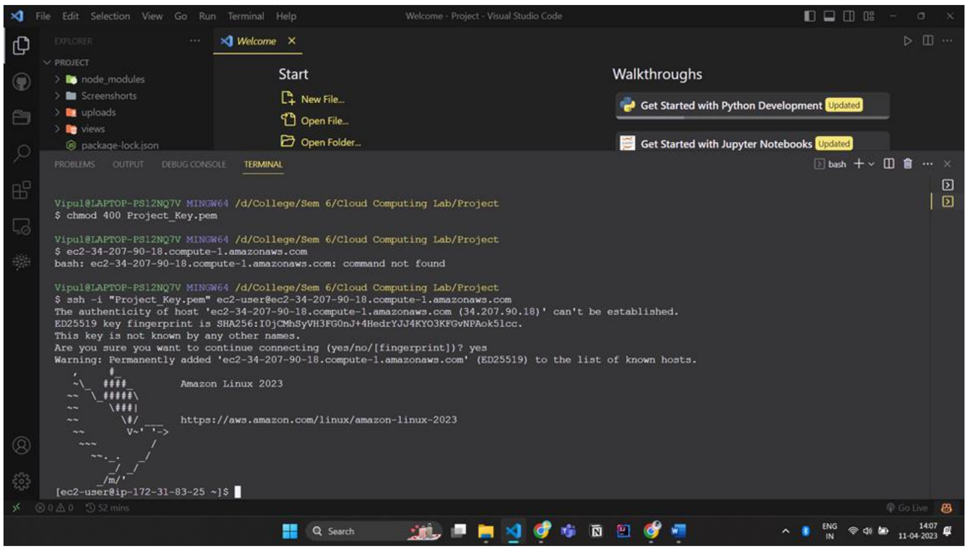


Instance summary for i-0c3b41a78c408b8ce (NodeJs_Server)
Updated less than a minute ago

| Field | Value |
|---|---|
| Instance ID | i-0c3b41a78c408b8ce (NodeJs_Server) |
| Public IPv4 address | 34.207.90.18 \| open address |
| Private IPv4 addresses | 172.31.83.25 |
| IPv6 address | – |
| Instance state | Running |
| Public IPv4 DNS | ec2-34-207-90-18.compute-1.amazonaws.com \| open address |
| Hostname type | IP name: ip-172-31-83-25.ec2.internal |
| Private IP DNS name (IPv4 only) | ip-172-31-83-25.ec2.internal |
| Answer private resource DNS name | IPv4 (A) |
| Instance type | t2.micro |
| Elastic IP addresses | – |
| Auto-assigned IP address | 34.207.90.18 [Public IP] |
| VPC ID | vpc-01f6b28062a1c5c04 |
| AWS Compute Optimizer finding | Opt-in to AWS Compute Optimizer for recommendations. \| Learn more |
| IAM Role | – |
| Subnet ID | subnet-01628fd43efa777ec |
| Auto Scaling Group name | – |
| IMDSv2 | Required |

Figure 3 AWS EC2 instance dashboard

Connected using GitBash tool



```
Vipul@LAPTOP-PS12NQ7V MINGW64 /d/College/Sem 6/Cloud Computing Lab/Project
$ chmod 400 Project_Key.pem

Vipul@LAPTOP-PS12NQ7V MINGW64 /d/College/Sem 6/Cloud Computing Lab/Project
$ ec2-34-207-90-18.compute-1.amazonaws.com
bash: ec2-34-207-90-18.compute-1.amazonaws.com: command not found

Vipul@LAPTOP-PS12NQ7V MINGW64 /d/College/Sem 6/Cloud Computing Lab/Project
$ ssh -i "Project_Key.pem" ec2-user@ec2-34-207-90-18.compute-1.amazonaws.com
The authenticity of host 'ec2-34-207-90-18.compute-1.amazonaws.com (34.207.90.18)' can't be established.
ED25519 key fingerprint is SHA256:I0jCMhSyVH3FG0nJ+4HedrYJJ4KYO3KFGvNPAok5lcc.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-34-207-90-18.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-83-25 ~]$
```

Figure 4 Connecting using SSH to the machine

Create workspace folder at /home/ec2-user

Installed git

```
yum install git
```

Installed Nodejs utilizing this docs https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh |
bash
. ~/.nvm/nvm.sh
nvm install --lts
```

Uploaded codebase to GitHub

```
git init
git add .
git commit -m "initial commit'
git remote add origin git@github.com:vipulchaudhary16/Image-compress-
API.git
git push --set-upstream origin master
```

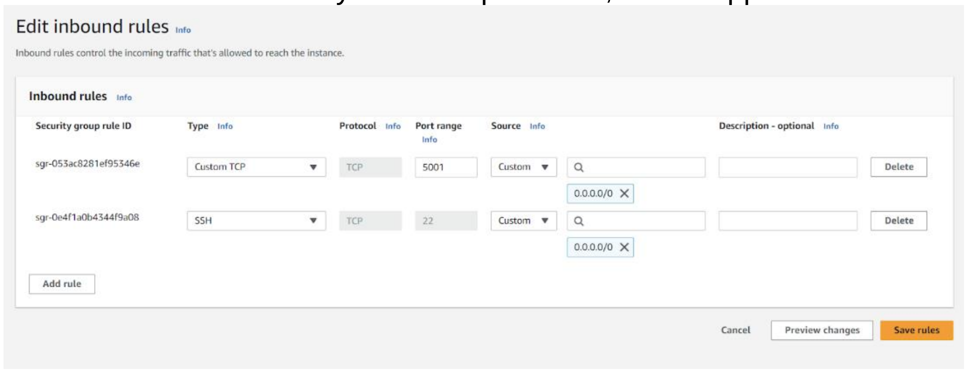
Clone code into machine on AWS

```
git clone https://github.com/vipulchaudhary16/Image-compress-API
```

Installed node modules

```
npm i
```

Set inbound rule to access from anywhere for port 5001, as our app will be run on this port



Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules Info

| Security group rule ID | Type Info | Protocol Info | Port range Info | Source Info | | Description - optional Info |
|---|---|---|---|---|---|---|
| sgr-053ac8281ef95346e | Custom TCP | TCP | 5001 | Custom | 0.0.0.0/0 | |
| sgr-0e4f1a0b4344f9a08 | SSH | TCP | 22 | Custom | 0.0.0.0/0 | |

*Figure 5 setting inbound rule for access to the port*

Just go to [http://public_ip_address:](http://public_ip_address:)5001 and your app is running there

Now to keep running our API all time we need to use pm2

```
pm2 start App.js
```

Now we have got public IP address of machine which will be helpful for API calls

Let's connect our front-end with this AWS instance

```
const ALGORITHM_HOST = "http://34.202.162.4:5001"
  const handleSubmit = (e) => {
    e.preventDefault()
    if (quality > 100 || quality < 0) {
      alert("Quality must be between 0 and 100")
      return
    }
    const formData = new FormData();
    const imagefile = document.querySelector('#file-to-compress');
    try {
      setIsCompressing(true)
      formData.append("file-to-compress", imagefile.files[0]);
      axios.post(ALGORITHM_HOST + '/file/compress', formData, {
        headers: {
          'Content-Type': 'multipart/form-data'
        }
      }).then((res) => {
        setDownloadLink(res.data.downloadURL)
        setIsCompressing(false)
      })
    } catch (error) {
      console.log(error)
      setIsCompressing(false)
    }
  }
```

This IP address varies every time we reboot the machine, Now request will be made to the ec2 instance for image compression.

## CONCLUSION

We utilized the Sharp algorithm to achieve efficient image compression, and provided a user-friendly interface for users to easily compress their image files. Through our evaluation, we demonstrated the effectiveness of our app in terms of compression ratio, file size reduction, and image quality. We believe that our app can be useful for individuals and businesses seeking to optimize their image files and improve their website or application performance. We also recognize the potential for further improvements, such as supporting additional file formats and integrating other compression algorithms. our successful development of the image compression app using Node.js and React.js, we also utilized AWS EC2, a cloud computing service, to host and deploy our application. By leveraging the power and scalability of cloud services, we were able to optimize the performance of our app and ensure its availability to users at all times. Overall, the integration of AWS EC2 added an additional layer of reliability and efficiency to our project.