"VIDEO COMPRESSION TOOL USING FFMPEG"

A

MAJOR PROJECT REPORT

Submitted to

partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS & TELECOMMUNICATION ENGINEERING

Submitted By

Digya Pratap Singh **(0201EC211031)**
Diva Gupta **(0201EC211032)**
Diya Mehra **(0201EC211034)**
Gagan Patel **(0201EC211035)**

Under the Guidance of
Prof. Namrata Sahayam
(Asst. Professor)

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING
JABALPUR ENGINEERING COLLEGE, JABALPUR (M.P.)
(Established in 1947 as Government Engineering College, Jabalpur)
(Declared Autonomous by Government of Madhya Pradesh and RGPV, Bhopal)

SESSION: 2021-2025

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

CANDIDATE'S DECLARATION

We hereby declare that the work, which is being presented in this Major Project Report, entitled ***"Video Compression Tool Using FFMPEG"*** *in partial fulfillment for the award of* ***Bachelor of Technology*** *degree in* ***Electronics and Telecommunication Engineering****, submitted in the Department of Electronics and Telecommunication Engineering, Jabalpur Engineering College, Jabalpur, is an authentic record of our own work carried out under the guidance of* ***Prof. Namrata Sahayam, Asst. Professor, Electronics and Telecommunication Engineering, Jabalpur Engineering College, Jabalpur.***

We have not submitted the matter embodied in this report for award of any other degree or diploma.

Digya Pratap Singh	***(0201EC211031)***
Diva Gupta	***(0201EC211032)***
Diya Mehra	***(0201EC211034)***
Gagan Patel	***(0201EC211035)***

Date:
Place:

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that Major Project Report entitled ***"Video Compression Tool Using FFMPEG"****submitted by* ***Digya Pratap Singh(0201EC211031), Diva Gupta (0201EC211032), Diya Mehra (0201EC211034), Gagan Patel (0201EC211035),*** *have been carried out under my guidance and supervision. This report is forwarded for submission towards the partial fulfillment for the award of* ***Bachelor of Technology*** *degree in* ***Electronics and Telecommunication Engineering.***

Forwarded to:

Head Of Department
(ECE , JEC)

Supervisor:

Prof. Namrata Sahayam
(Asst. Professor)

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

CERTIFICATE OF APPROVAL

This is to certify that Major Project Report entitled ***"Video Compression Tool Using FFMPEG"*** *submitted by* ***Digya Pratap Singh*** *(**0201EC211031**),* ***Diva Gupta*** *(**0201EC211032**),* ***Diya Mehra (0201EC211034)***, ***Gagan Patel*** *(**0201EC211035**),* *is accepted towards partial fulfillment for the award of* ***Bachelor of Technology*** *degree in* ***Electronics and Telecommunication Engineering.***

INTERNAL EXAMINER
DATE:_____

EXTERNAL EXAMINER
DATE: _____

ACKNOWLEDGEMENT

It is a matter of extreme honor and privilege for us to offer our grateful acknowledgement to our respected guide ***Prof. Namrata Sahayam, Asst. Professor Electronics and Telecommunication Engineering, Jabalpur Engineering College, Jabalpur*** *to give us a chance to work under her guidance and supervision. We also grateful for all kinds of support and inspiration, constant encouragement, sincere criticism and valuable guidance which were given by her throughout the investigation and preparation of this thesis.*

We would also like to take this opportunity to present our sincere regards to ***Dr. Bhavana Jharia (HOD)****,* ***Dr. A. K. Buchke, Dr. Agya Mishra, Dr.Kanchan Cecil, Prof. Sunil Kumar Singh, Prof. Devendra Kumar Meda and all technical staff of the Department of Electronics and Telecommunication Engineering,*** *for their valuable advice and assistance during our studies at Jabalpur Engineering College Jabalpur, and their support.*

We want to express our sincere gratitude to ***Dr. Rajeev Chandak , Principal Jabalpur Engineering College, Jabalpur****, for his valuable support and facilities provided to us to carry out this work at Jabalpur Engineering College, Jabalpur (M.P.).*

We also sincerely thank our guest faculties for their valuable support during the project preparation, which greatly contributed to the smooth execution of the work.

Above all our deepest love and gratitude is devoted to our parents. They are always a source of love, support and encouragement.

Last but not least, we also thankful to Almighty God for giving us the patience and the strength for guiding us make choices in life and helping us in such mysterious way that is impossible for us to explain.

Digya Pratap Singh
Diva Gupta
Diya Mehra
Gagan Patel

TABLE OF CONTENT

Certificates	i – iv
Acknowledgement	v
Abstract	vi
List of Figures & Table	vii
Abbreviations Used	viii
Chapter – 1 Introduction	1 - 2
1.1 Overview	
1.2 Problem Statement	1
1.3 Objectives of the Project	1
1.4 Scope of the Project	2
1.5 Summary	2
Chapter – 2 Review	3 - 4
2.1 Overview of Video Compression Techniques	
2.1.1 Lossless Vs. Lossy Compression	3
2.2 Traditional Video Compression Standards	3
2.2.1 H.264 (Advanced Video Coding – AVC)	3
2.2.2 H.265 (High Efficiency Video Coding – HEVC)	4
2.2.3 MPEG-4 and MPEG-2	4
2.2.4 VP9 (Developed by Google)	4
2.3 Conclusion	4
Chapter – 3 Methodology	5 – 13
3.1 Introduction to FFmpeg	
3.1.1 How FFmpeg Works	5
3.1.2 Comparative Analysis: FFmpeg Vs. Traditional Video Compression Tools	5
3.2 Implementation	5
3.2.1 System Architecture	5
a. Frontend Technology: React	6
b. Backend Technology: Node.js	7
c. Cloud Storage Solution: Amazon Web Services - S3 Bucket	8
3.3 Working	10
3.4 Algorithm Used	11
3.5 FFmpeg Installation	11
3.6 Required Dependencies	11
3.7 FFmpeg Compression Command	12
3.8 Conclusion	13
Chapter – 4 Result and Testing	14– 22
4.1 Compression Performance Analysis	14
4.2 Backend Testing Using Postman	17
4.3 Frontend Testing Using Web Browser	18
Chapter – 5 Advantages, Disadvantages & Limitations	23– 24
5.1 Advantages	23
5.2 Disadvantages	23
5.3 Limitations	24
Chapter – 6 Conclusion	25
Reference	

ABSTRACT

Video compression is a fundamental technology in modern multimedia systems, enabling efficient storage, transmission, and processing of video content. By reducing file sizes while preserving visual quality, video compression optimizes bandwidth usage and enhances the performance of applications such as video streaming, telemedicine, and surveillance. Various compression techniques, including inter-frame and intra-frame coding, are employed to achieve high compression ratios while minimizing perceptible quality loss. Tools like FFmpeg, which utilize advanced encoding algorithms, play a crucial role in implementing efficient video compression solutions.

In this project, we developed a software-based Video Compression Tool that utilizes FFmpeg for efficient video encoding and compression. The backend is built using Node.js, while the frontend is developed using React.js for an intuitive user interface. Compressed video files are stored securely in AWS S3, ensuring scalability and accessibility. This project demonstrates the practical application of video compression in real-world scenarios, offering a robust and scalable solution for multimedia processing and storage optimization.

LIST OF FIGURES AND TABLE

Fig 2.1	Compression
Fig 3.1	Client-Server Architecture
Fig 3.2	React Characteristics
Fig 3.3	Node.js Characteristics
Fig 3.4	S3 Storage Code Overview
Table 4.1	Compression Analysis Table

LIST OF SCREENSHOTS

ss. 3.1	AWS S3 Client Configuration
ss. 3.2	File Uploading to S3 Bucket (Code)
ss. 3.3	Generating Pre-signed URL and Returning Response (Backend API)
ss. 4.1	AWS S3 Bucket Listing in the AWS
ss. 4.2	S3 Bucket Folder Structure “compressed”
ss. 4.3	Compressed video files
ss. 4.4	Postman - Sending Video Upload Request to Backend API
ss. 4.5	Backend API Response with Pre-signed
ss. 4.6	Frontend - Clean View of the Video Compression Tool
ss. 4.7	Frontend - File Selection for video upload
ss. 4.8	100% Upload Progress Bar(frontend)
ss. 4. 9	Frontend - Compression Process Progress
ss. 4.10	Compression Completion and Download.

ABBREVIATIONS USED

FFmpeg	Fast Forward Moving Picture Experts Group
API	Application Programming Interface
AWS	Amazon Web Services
GPU	Graphics Processing Unit
CPU	Central Processing Unit
MP4	MPEG-4 (Moving Picture Experts Group 4)
AVI	Audio Video Interleave
H.264	Advanced Video Coding (AVC)
HEVC	High-Efficiency Video Coding
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
RTMP	Real-Time Messaging Protocol
HTTP	Hypertext Transfer Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
CRF	Constant Rate Factor
AAC	Advanced Audio Codec
CABAC	Context-Adaptive Binary Arithmetic Coding
VLC	Video LAN Client
FPS	Frames per Second
DCT	Discrete Cosine Transform
SS	Screenshot

2

1. INTRODUCTION

1.1 Overview

With the rapid advancement in digital media and communication technologies, video content has become a dominant form of information exchange across various domains, including telecommunications, medical imaging, satellite communication, and surveillance systems. However, uncompressed video files require high storage space and bandwidth, making them impractical for real-world applications where efficient data transmission and storage are crucial [1].

In Electronics and Communication Engineering (ECE), video compression plays a vital role in ensuring seamless multimedia transmission over constrained networks. Applications such as real-time video streaming, wireless communication, remote sensing, and IoT-based multimedia systems demand optimized video data to function effectively. Video compression algorithms and techniques help reduce the size of video files while preserving visual quality, making them essential for enhancing system performance in bandwidth-limited environments [1].

1.2 Problem Statement

Traditional video storage and transmission methods suffer from several challenges:

- Large file sizes that require significant storage space [2].
- High bandwidth consumption, leading to transmission delays in real-time applications [2].
- Loss of quality in low-bit-rate streaming, affecting user experience [2].
- Hardware limitations in embedded systems, restricting efficient video processing [2].

To address these challenges, this project presents a Video Compression Tool, designed to reduce file sizes without compromising visual quality. The tool is built using React.js for the frontend and Node.js for the backend, with FFmpeg, a powerful multimedia framework, handling the core compression tasks [2].

1.3 Objectives of the Project

The primary objectives of this project are:

- To develop an efficient video compression tool that minimizes file size while maintaining high-quality output.
- To enable optimized storage and transmission for multimedia applications.
- To integrate industry-standard FFmpeg compression techniques for high-performance encoding.
- To provide an intuitive and user-friendly interface for seamless video processing.

1.4 Scope of the Project

The primary goal is to reduce video file sizes while maintaining acceptable quality for efficient

storage and transmission. However, due to hardware limitations (Intel Core i3 processor) and the absence of high performance computing resources, the project operates with certain constraints.

The key scope of this project includes:

- Basic Video Compression– Supports standard video compression formats such as H.264.
- Small to Medium-Sized Videos– The tool is suitable for compressing low to moderate resolution videos, but large-sized videos may cause performance issues.
- Local Testing Environment– Designed primarily for local execution rather that large-scale cloud deployments, making it more of a proof-of-concept than an enterprise solution.
- Processing Speed– Due to hardware constraints, compression may take longer compared to dedicated video processing systems. Real-time or near-real-time compression is not achievable in the current setup.

1.5 Summary

This chapter provided an introduction to the project and outlining the objectives of the developer tool. The following chapters will explore the project in more detail.

Chapter 2 presents a review of existing video compression techniques, related technologies and previous research in the field.

Chapter 3 explains the methodology, including the technical implementation, tools, and algorithms used.

Chapter 4 discusses the result obtained from testing the compression tool, along with performance analysis.

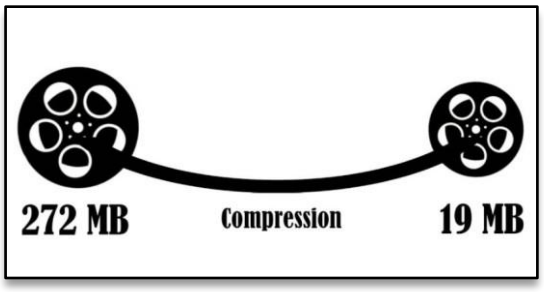
Chapter 5 highlights the advantages, disadvantages, and various application of the tool.

Finally, Chapter 6 concludes the report and explores potential future improvements to enhance the tool’s efficiency and scalability.

2. REVIEW

2.1 Overview of Video Compression Techniques

Video compression is a critical aspect of multimedia processing, enabling efficient storage and transmission of video data while maintaining acceptable visual quality. Compression techniques are categorized into lossy and lossless compression methods, with lossy compression being the preferred approach for reducing file size significantly while preserving perceptual quality. Over the years, multiple video compression standards have been developed, each optimized for specific applications, including broadcasting, streaming, video conferencing, and surveillance [3].



(fig 2.1 Compression)

2.1.1 Lossless vs. Lossy Compression

1. **Lossless Compression**
    - Retains all original data without degradation in quality.
    - Common techniques include Run-length Encoding (RLE), Huffman Coding, and Lempel-Ziv-Welch (LZW).
    - Used in application where quality preservation is critical, such as medical imaging and archival storage.
    - Results in limited compression efficiency (typically 2:1 or 3:1 compression ratios).

2. **Lossy Compression**
    - Removes redundant and less noticeable data to achieve higher compression ratios.
    - Relies on perceptual coding techniques that exploit human visual system limitations.
    - Commonly used for video streaming, broadcasting, and multimedia applications.
    - Includes standards like H.264, H.265 (HEVC), and VP9 [4].

2.2 Traditional Video Compression Standards

2.2.1 H.264 (Advanced Video Coding – AVC)

- Developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG).
- One of the most widely used video compression standards, offering up to 50% better compression than MPEG-2.
- Features include:
    - Block-based motion compensation to reduce temporal redundancy.

    - Entropy coding techniques like Context-Adaptive Binary Arithmetic Coding (CABAC) for higher efficiency.
    - Profiles and levels optimized for diverse applications (Baseline, Main, High Profile).
- Common Use Cases: Blu-ray discs, YouTube, mobile video streaming, and digital TV broadcasting [5].

2.2.2 H.265 (High Efficiency Video Coding – HEVC)

- Successor to H.264, developed to achieve better compression efficiency, reducing file size by 50% compared to H.264.
- Features include:
    - Larger macroblocks (Coding Tree Units – CTUs) for improved compression.
    - **Advanced motion compensation techniques** like asymmetric motion partitioning.
    - Higher bit depth support (10-bit and beyond) for enhanced color reproduction.
- Common Use Cases: 4K/8K Ultra HD streaming, video conferencing, and cloud gaming [6].

2.2.3 MPEG-4 and MPEG-2

- **MPEG-4:** Designed for internet video streaming and mobile applications. It introduced **object-based coding** and improved compression over MPEG-2.
- **MPEG-**2: Earlier compression standard used in DVDs, digital TV broadcasting, and satellite communications but now mostly replaced by H.264 and H.265 [7].

2.2.4 VP9 (Developed by Google)

- Open-source alternative to H.265, optimized for low-bandwidth streaming applications.
- Used by YouTube and WebRTC-based video conferencing systems [7]**.**

2.3 Conclusion

From the analysis of existing video compression techniques, it is evident that H.264, H.265, and VP9 are the most widely used standards for modern video applications. However, these traditional compression techniques have certain limitations, such as restricted codec support in proprietary software, and inefficiencies in real-time processing.

By leveraging FFmpeg, this project aims to overcome the limitations of traditional compression tools, enabling improved compression efficiency, better compatibility across platforms, and enhanced processing speed, making it a practical choice for applications in many engineering branches.

3. METHODOLOGY

3.1 Introduction to FFmpeg

FFmpeg, which stands for ***Fast Forward Moving Picture Experts Group,*** is an open-source multimedia framework widely used for video and audio processing. It provides a comprehensive suite of libraries and tools that enable video encoding, decoding, transcoding, streaming, and compression. Due to its versatility and extensive codec support, FFmpeg has become the preferred choice for handling multimedia data efficiently [8].

3.1.1 How FFmpeg Works

FFmpeg operates through a command-line interface (CLI) that processes multimedia files using a set of libraries, including:

- **libavcodec**– Handles video and audio encoding/decoding.
- **libavformat**– Manages multimedia container formats such as MP4, MKV, and AVI.
- **libswscale**– Performs image scaling and pixel format conversions.
- **libavfilter**– Provides video and audio filtering capabilities for processing effects.

FFmpeg processes videos by reading the input file, applying the necessary compression techniques, and encoding the output in the desired format. The encoding process includes bitrate control, resolution adjustments, and format conversion, ensuring optimized file sizes without significant quality loss [8].

3.1.2 Comparative Analysis: FFmpeg vs Traditional Video Compression Tools

Traditional video compression tools, including standalone software like HandBrake and proprietary solutions, have various limitations that FFmpeg effectively addresses [9]:

- Limited Codec Support
- Computational Inefficiency
- Lack of Customization
- Real-Time Processing
- Open-Source and Free

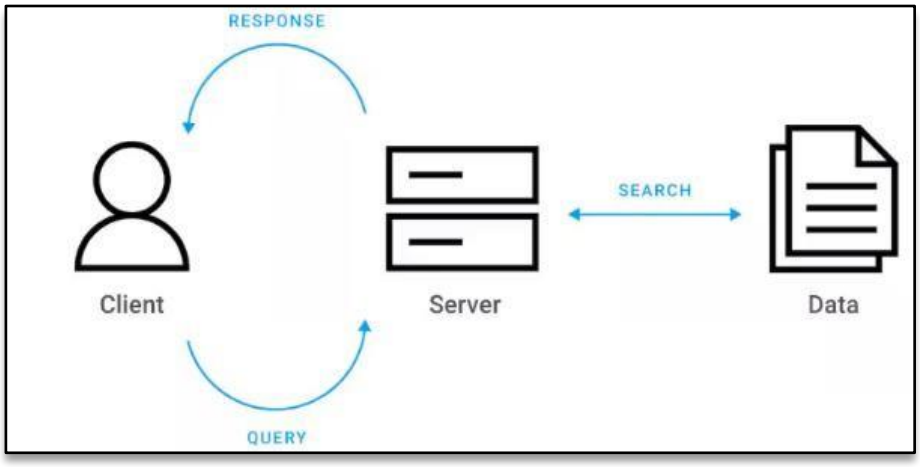
3.2 Implementation

The methodology for implementing this video compression tool using FFmpeg follows a structured approach.

3.2.1 System Architecture

The system is built using a client-server architecture with multiple integrated components ensuring smooth video compression, storage, and retrieval. The core components as follow:

a. Frontend Technology: React
b. Backend Technology: Node.js
c. Storage: Amazon Web Services (AWS) S3



(fig 3.1 Client-Server Architecture)

❖ **Client server architecture**

A distributed computing model where clients request services, and servers provide responses over a network [10].

1. Components:
    - **Client**: Requests data or services (e.g., web browser, mobile app).The server processes the request and sends back the required data or response.
    - **Server**: Processes requests and returns results (e.g., web server, database server)

2. Communication Flow:
    - Clients send requests to the server using HTTP, TCP, or other protocols.
    - The server processes the request and sends back the required data or response.

a. Frontend Technology: React

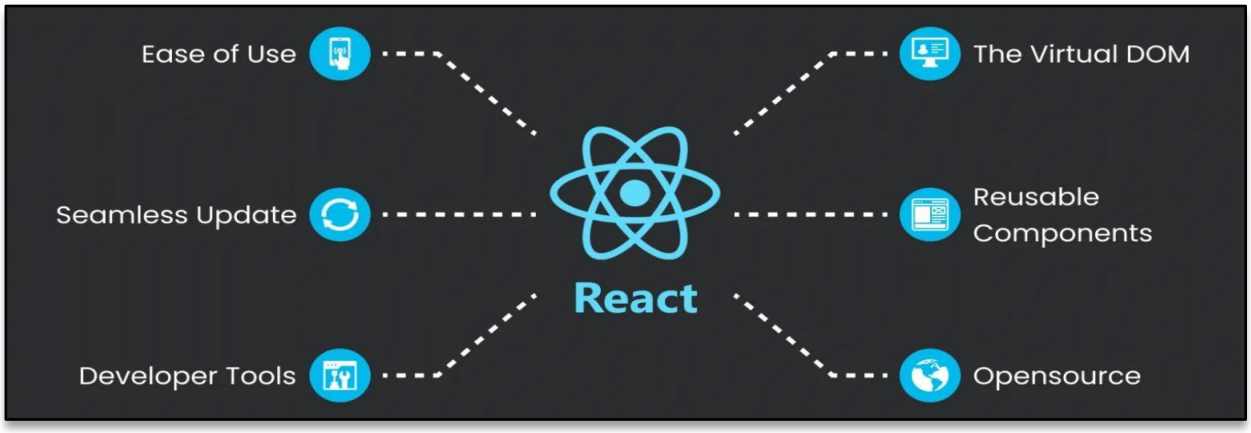
❖ **Overview**

React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

Developed by Facebook, it enables creation of complex, interactive web applications with component-based architecture [11].

❖ **Core React Concepts**

1. Component-Based Architecture
2. Virtual DOM
3. Unidirectional Data Flow

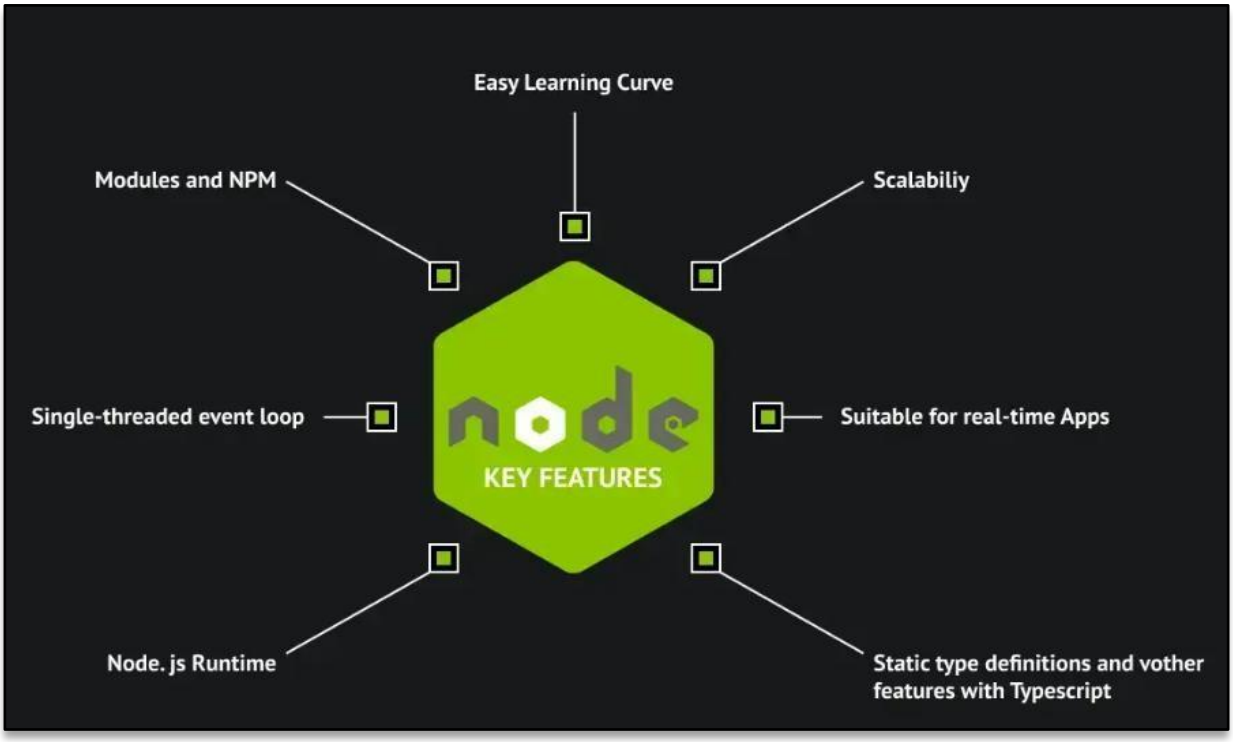


(fig 3.2 React Characteristics)

b. Backend Technology: Node.js

❖ **Overview**

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside a web browser. Built on Chrome's V8 JavaScript engine, it enables server-side and networking applications [12].



(fig 3.3 Node.js Characteristics)

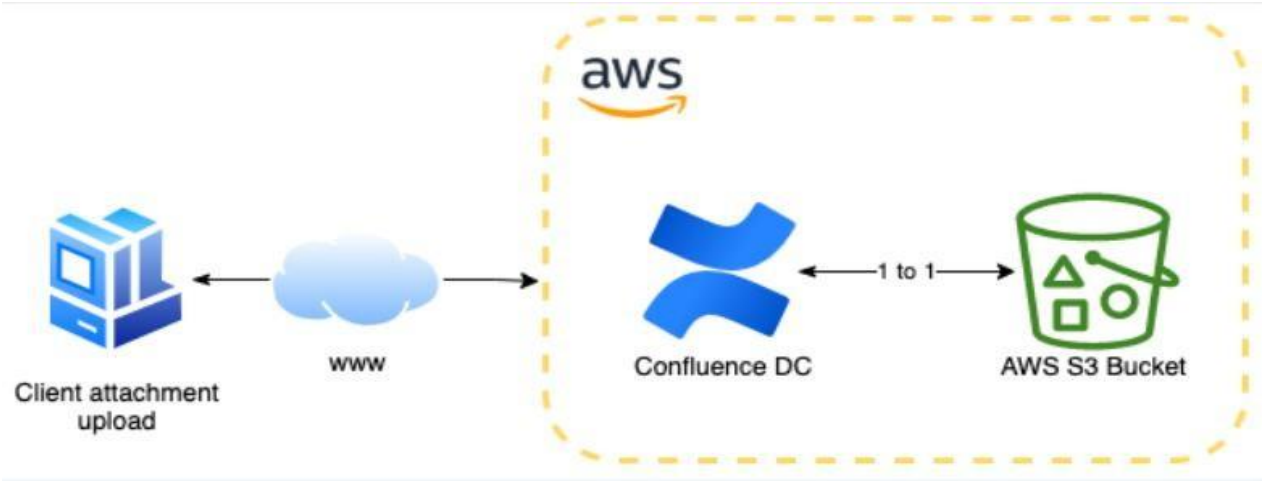
❖ **Core Characteristics**

1. Event-Driven Architecture
2. Module System
3. Performance Advantages

c. Cloud Storage Solution: Amazon Web Services (AWS)

❖ **Introduction to AWS S3**

Amazon Simple Storage Service (S3) is a highly scalable, secure, and durable object storage service provided by Amazon Web Services. It is designed to store and retrieve any amount of data from anywhere on the web, making it an ideal solution for cloud storage needs [13].



(fig 3.4 AWS S3 Bucket Flow)

❖ **Key Feature of AWS S3**

1. Scalability
2. Durability and Reliability
3. Security Features
4. Performance Optimization

❖ **S3 Implementation in Video Compression Tool Project**

In our project, S3 serves as the primary storage solution for compressed video files:

- Secure storage of compressed video files
- Generation of pre-signed URLs for temporary, secure access
- Automatic file management and cleanup
- Scalable storage infrastructure

```
// AWS S3 Configuration
const s3 = new AWS.S3({
  accessKeyId: process.env.AWS_ACCESS_KEY,
  secretAccessKey: process.env.AWS_SECRET_KEY,
  region: process.env.AWS_REGION
});
```

(Screenshot 3.1 S3 Bucket Object)

❖ **Code Implementation Highlights**

The code is creating a new instance of the AWS S3 client with the following configuration:

- **const s3 = new AWS.S3({...})** - This line declares a constant variable named 's3' and initializes it with a new instance of the AWS S3 client.

Inside the configuration object, three key parameters are being set:

- accessKeyId: process.env.AWS_ACCESS_KEY - This sets the AWS access key ID using an environment variable
- secretAccessKey: process.env.AWS_SECRET_KEY - This sets the AWS secret access key using an environment variable
- region: process.env.AWS_REGION - This sets the AWS region using an environment variable

```
// Upload to S3
const fileContent = fs.readFileSync(outputFile);
const fileKey = `compressed/${outputFile}`;
const uploadParams = {
  Bucket: process.env.AWS_S3_BUCKET,
  Key: fileKey,
  Body: fileContent,
  ContentType: 'video/mp4'
};
const s3Response = await s3.upload(uploadParams).promise();
```

(Screenshot 3.2 File Uploading Code for S3 Bucket)

- const fileContent = fs.readFileSync(outputFile); - This line reads the contents of a file (referenced by the variable outputFile) synchronously using the Node.js file system (fs) module.
- const fileKey = 'compressed/${outputFile}; - This creates a storage path/key for the file in S3, placing it in a "compressed" folder with the original filename.
- const uploadParams = {...} - This object defines the parameters for the S3 upload:
  - Bucket: Specifies the S3 bucket name from an environment variable
  - Key: The path/filename in S3 where the file will be stored

- Body: The actual file content to be uploaded
- ContentType: Set to 'video/mp4', indicating this is an MP4 video file

- const s3Response = await s3.upload(uploadParams).promise(); - This line uploads the file to S3 using the previously configured S3 client. The await keyword indicates this is an asynchronous operation, and promise(s) ensures it returns a Promise for proper async handling.

```
const presignedUrl = s3.getSignedUrl('getObject', {
  Bucket: process.env.AWS_S3_BUCKET,
  Key: fileKey,
  Expires: 60 * 60
});

res.json({
  message: 'Compression successful',
  presignedUrl
});
```

(Screenshot 3.3 Generating Pre signed URL & return response)

- const presignedUrl = s3.getSignedUrl('getObject', {...}) - This generates a temporary URL that allows access to the file in S3:
  - 'getObject' specifies the operation type (downloading the file)
  - Parameters include:
    - Bucket: The S3 bucket name (from environment variable)
    - Key: The file path/key in S3
    - Expires: Set to 3600 seconds (60 * 60 = 1 hour), after which the URL becomes invalid
- res.json({...}) - This sends a JSON response back to the client with:
  - message: A success message indicating compression was successful
  - presignedUrl: The temporary URL that allows the client to access the uploaded file

3.3 Working

1. **Video Upload:**
   - The user selects a video and uploads it using the React frontend.
   - The file is sent to the backend via an API endpoint.
2. **Processing & Compression:**
   - The backend receives the file and stores it temporarily.
   - FFmpeg is initiated for video compression.
   - The H.265 codec (libx265) is used for high-efficiency compression with CRF set to 28.
3. **Real-Time Progress Updates:**
   - WebSockets send compression progress updates to the frontend.

4. **Finalizing Compression:**
    - The compressed video is stored in an AWS S3 bucket.
    - A pre-signed URL is generated for secure file access.
    - The original and compressed files are deleted from the server to optimize storage.
5. **User Access:**
    - The frontend displays a download link for the compressed video.
    - The user can download the file directly from AWS S3.

3.4 Algorithms Used

3.4.1 Encoding Techniques

❖ **H.265 (HEVC - High Efficiency Video Coding):**

- Chosen for its advanced compression efficiency compared to H.264.
- Reduces file size significantly while preserving visual quality.

❖ **Constant Rate Factor (CRF):**

- Used for variable bitrate encoding.
- Maintains a balance between compression and quality.

3.5 FFmpeg Installation

These commands are used in a Linux environment to install and verify the FFmpeg multimedia framework:

- To install FFmpeg on Linux:

sudo apt update
sudo apt install ffmpeg

✓ sudo apt update: refreshes the package lists to get information about the newest versions of packages
✓ sudo apt install ffmpeg: installs the FFmpeg software package

- To verify installation:

ffmpeg -version

✓ ffmpeg -version: verifies the installation by displaying the version information of FFmpeg

3.6 Required Dependencies

- Backend

npm install express multer aws-sdk cors dotenv uuid ws

✓ express: Web application framework for creating API endpoints
✓ multer: Middleware for handling multipart/form-data (file uploads)
✓ aws-sdk: AWS SDK for JavaScript to interact with AWS services
✓ cors: Middleware for enabling Cross-Origin Resource Sharing
✓ dotenv: Loads environment variables from a .env file
✓ uuid: For generating unique identifiers

- ✓ ws: WebSocket implementation for real-time communication

- Dev Dependencies

npm install –save-dev nodemon

- ✓ nodemon: Tool that automatically restarts the server when file changes are detected

- Frontend

npm install react axios react-hot-toast

- ✓ react: JavaScript library for building user interfaces
- ✓ axios: Promise-based HTTP client for making requests
- ✓ react-hot-toast: Lightweight notification library for React

3.7 Ffmpeg Compression Command

This JavaScript code uses Node.js's child process functionality to execute FFmpeg for video compression:

- ✓ spawn('ffmpeg', [...]) creates a new FFmpeg process with specific command-line arguments

```
const ffmpeg = spawn('ffmpeg', [
        '-i', inputFile,          // Input video file
        '-c:v', 'libx265',        // Use H.265 (HEVC) codec
        '-preset', 'slow',        // Optimize for better compression
        '-crf', '28',             // Adjust quality and compression balance
        '-c:a', 'aac',            // Use AAC audio codec
        '-b:a', '128k',           // Set audio bitrate to 128 kbps
        '-movflags', 'faststart', // Optimize for progressive streaming
        '-vf', 'scale=-2:1080',   // Resize video to 1080p
        outputFile
]);
```

The array contains FFmpeg command parameters:

- ✓ `-i, inputFile`: Specifies the input video file path
- ✓ `-c:v, 'libx265'`: Uses the H.265 (HEVC) video codec, which provides better compression than older codecs like H.264
- ✓ `-preset, 'slow'`: Sets the encoding speed to "slow", which produces better compression at the cost of longer processing time
- ✓ `-crf, '28'`: Sets the Constant Rate Factor to 28, balancing quality and file size (lower values = higher quality, larger files)
- ✓ `-c:a, 'aac'`: Uses the AAC audio codec, which is widely supported
- ✓ `-b:a, '128k'`: Sets the audio bitrate to 128 kbps for good audio quality with reasonable file size
- ✓ `-movflags, 'faststart'`: Optimizes the output file for progressive streaming (allows playback before the entire file is downloaded)

- ✓ `-vf, 'scale=-2:1080'`: Resizes the video to 1080p height while maintaining aspect ratio (-2 ensures even dimensions)
- ✓ `outputFile`: Specifies the destination file path

3.8 Conclusion

1. FFmpeg stands out as a powerful, open-source multimedia framework that provides efficient video and audio processing with extensive codec support, making it a preferred choice for video compression tasks.
2. The integration of FFmpeg in a client-server architecture, combined with technologies like React, Node.js, and AWS S3, ensures an efficient, scalable, and secure video compression solution.
3. By utilizing advanced encoding techniques such as H.265 and CRF, the system achieves significant file size reduction while maintaining high video quality, optimizing both storage and bandwidth usage.
4. The system's use of real-time progress updates and secure file management through AWS S3 enhances the overall user experience, making it a seamless and reliable solution for video compression and retrieval.

4. RESULT AND TESTING

The testing phase is essential in validating the efficiency and accuracy of the video compression tool. This section presents the compression performance results, and provides an overview of the testing methodology used to ensure the system meets the required standards. The results are backed by **file size comparisons, and** backend testing using Postman. Furthermore, screenshots of the frontend, backend, and API responses are included to illustrate the system's functional integrity.

4.1 Compression Performance Analysis

The effectiveness of a video compression system is primarily determined by the extent to which it reduces file size while maintaining visual quality. In this project, compression results were evaluated using a set of test videos of varying resolutions (e.g., 360p, 720p, 1080p, 4K).

Compression Ratio: In video encoding refers to the proportion by which the original video file size is reduced while maintaining an acceptable level of quality [13].
A key metric in video compression is the **compression ratio**, which is calculated as:

$$Compression\ Ratio = Original\ File\ Size \div Compressed\ File\ Size$$

Video Resolution	Original Size (MB)	Compressed Size (MB)	Compression Ratio
360p	4.88 MB	2.14 MB	2.28:1
720p (HD)	7.69 MB	4.00 MB	1.92:1
1080p (Full HD)	10.0 MB	6.90 MB	1.45:1
1080p (Full HD)	20.0 MB	13.2 MB	1.52:1

(Table 4.1)

From the results, it is evident that the video compression system significantly reduces file sizes while maintaining acceptable quality levels.

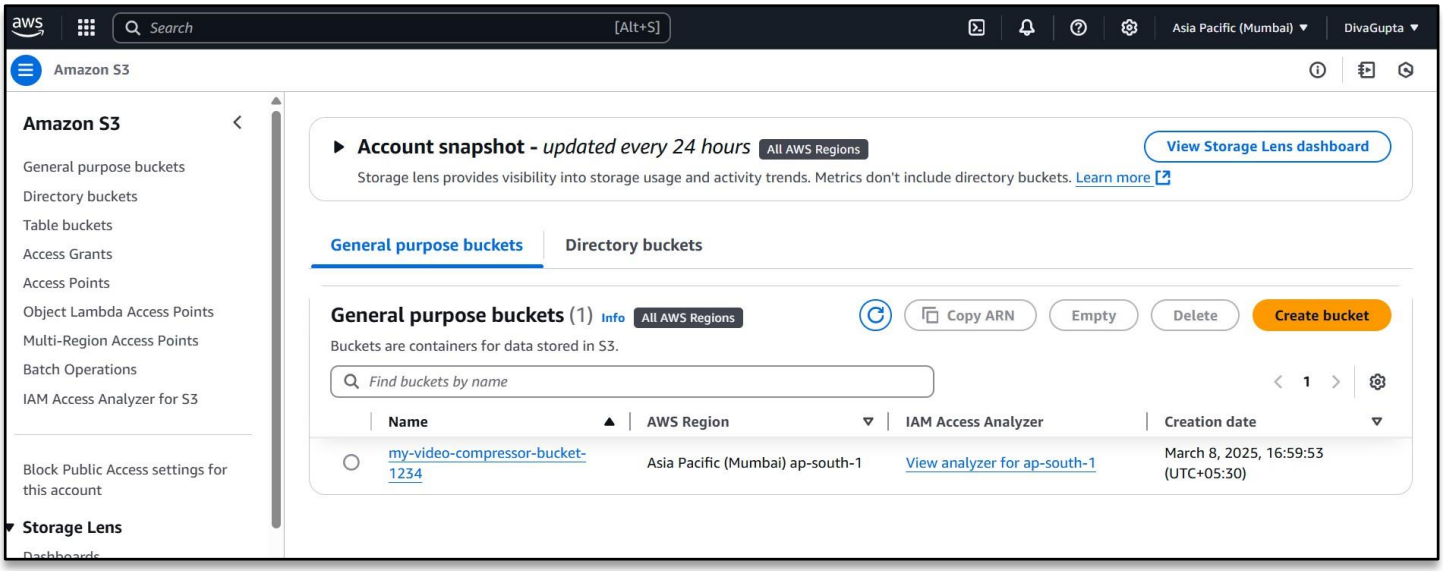
Compressed files save into AWS storage.

my-video-compressor-bucket-1234.s3.ap-south-1.amazonaws.com/compressed/

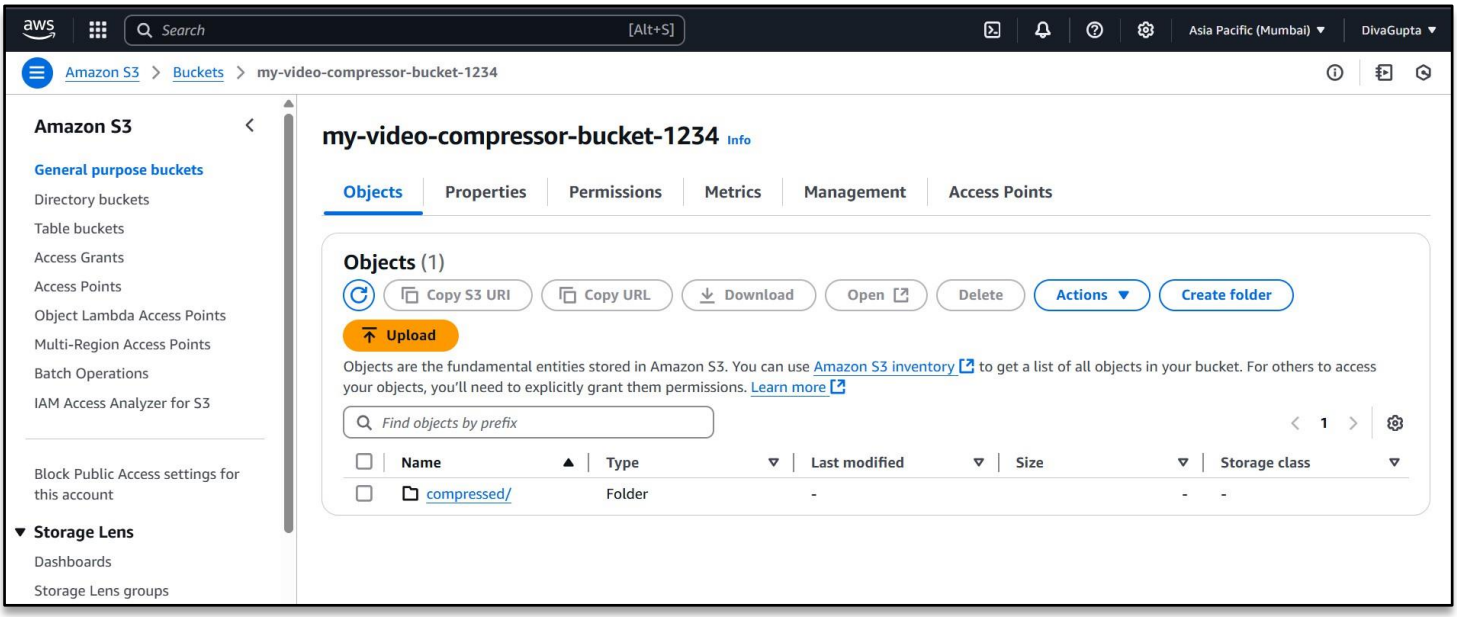
This above url pattern indicates:

- *Your bucket name: "my-video-compressor-bucket-1234"*
- *Region: "ap-south-1" (Asia Pacific Mumbai)*
- *Folder path: "compressed/"*
- *This URL structure facilitates secure access to your processed videos through presigned URLs*

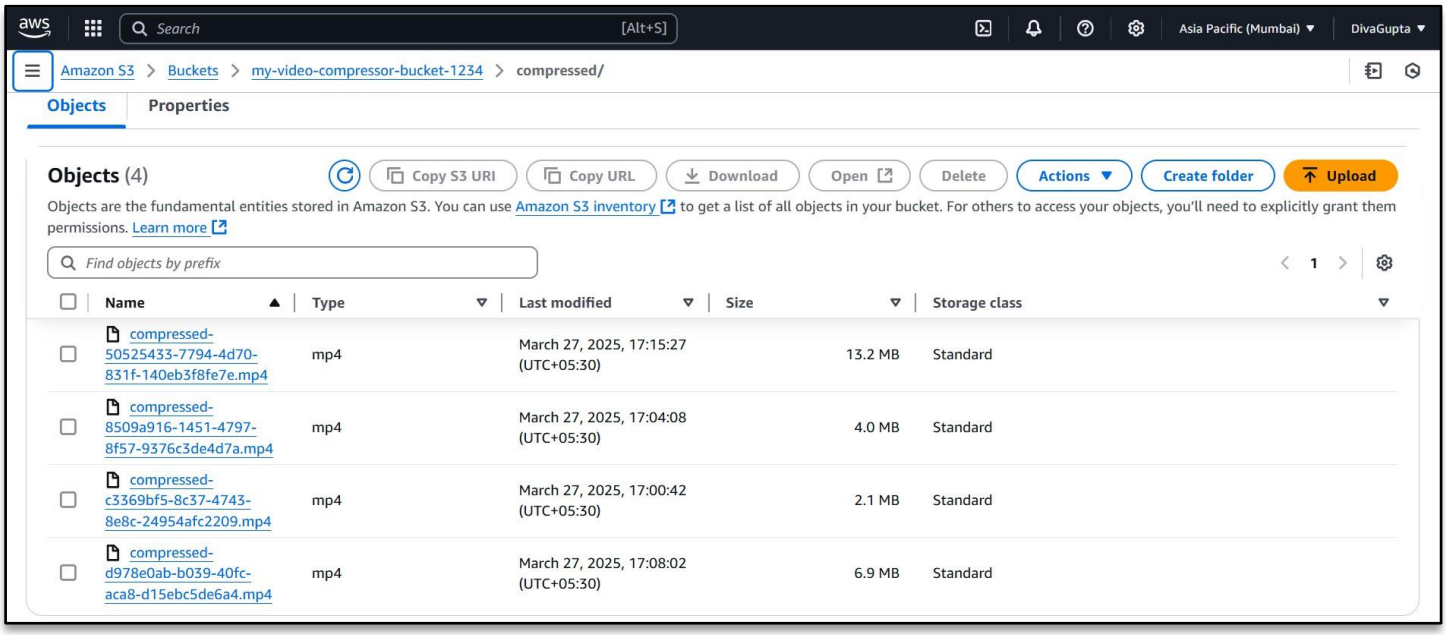
(Screen shot 4.1 - This shows the main S3 console with your "my-video-compressor-bucket-1234" bucket listed among general-purpose buckets. The bucket is hosted in the Asia Pacific (Mumbai) region, created on March 8, 2025, and includes Storage Lens monitoring capabilities.)



(Screen shot 4.2 - This displays the bucket structure, showing the "compressed/" folder which serves as your organized storage location for processed videos. The interface provides various management options including upload, download, copy URL functionalities, and permission settings.

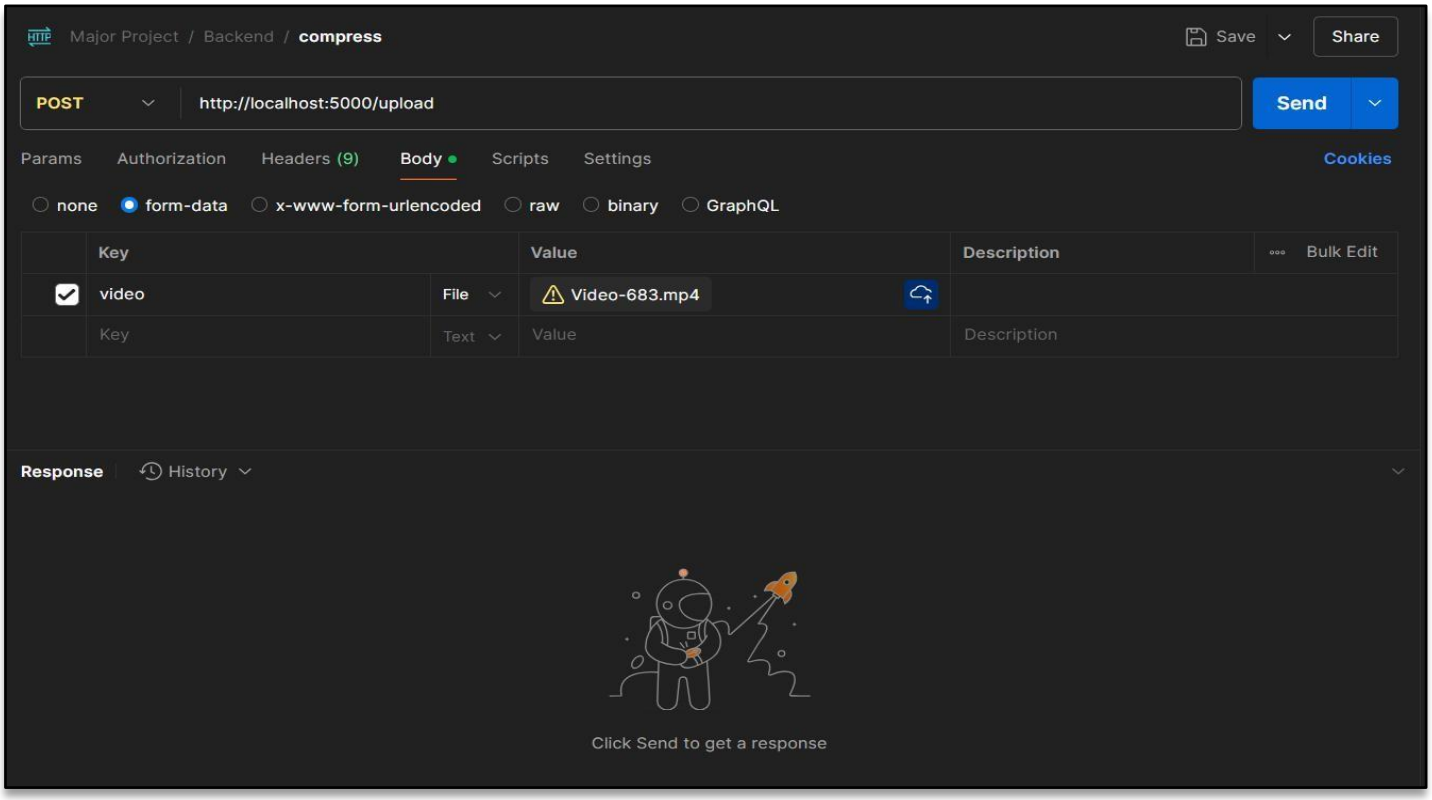


(Screen shot 4.3 - This screenshot reveals the contents of the "compressed/" folder, showing four MP4 files that have been successfully compressed and stored. The file sizes range from 2.1 MB to 13.2 MB, demonstrating effective compression.)

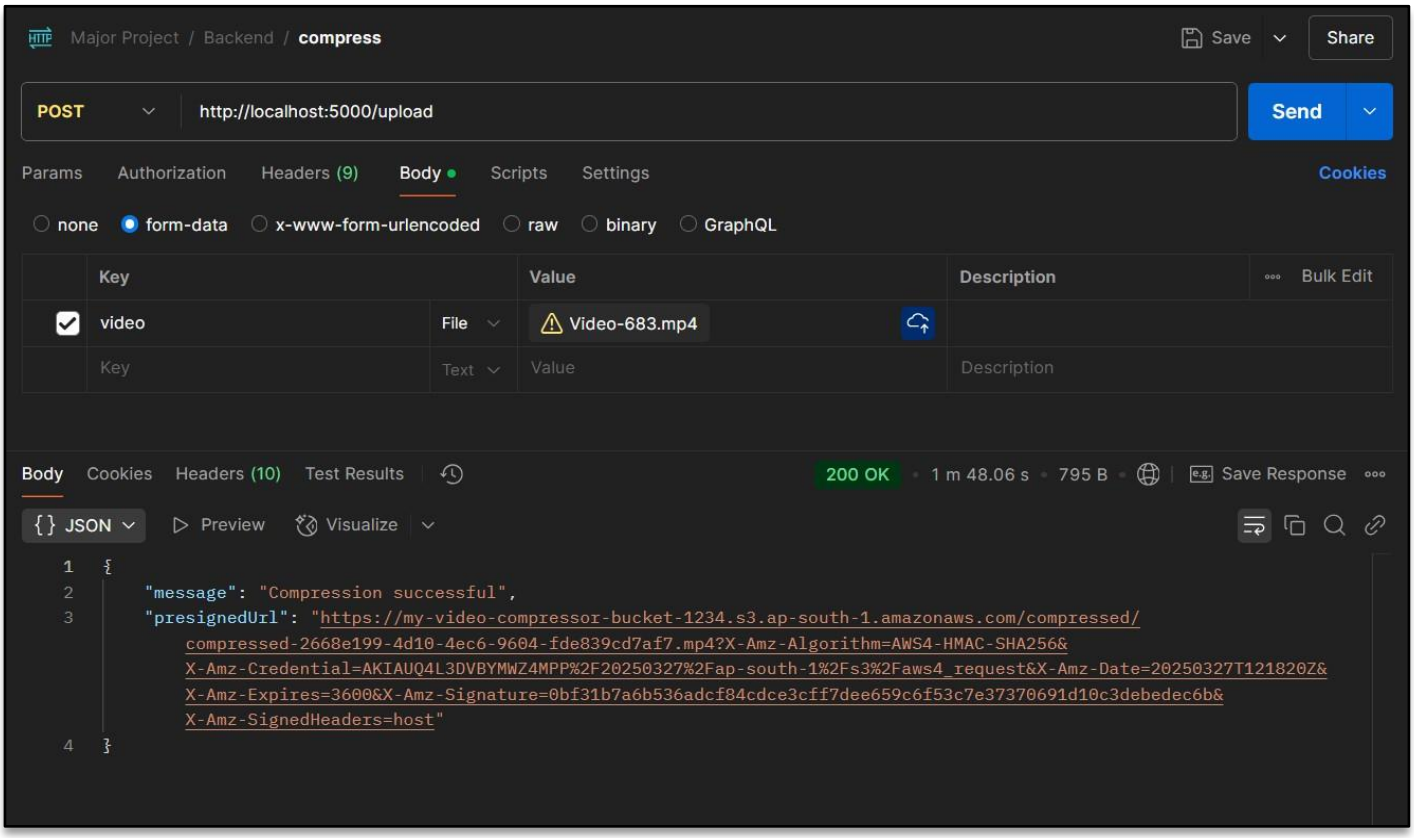


4.2 Backend Testing Using Postman

(Screen shot 4.4 - Shows a POST request to your local development with form-data content type.)



Screenshot 4.5- Show the result of backend API)

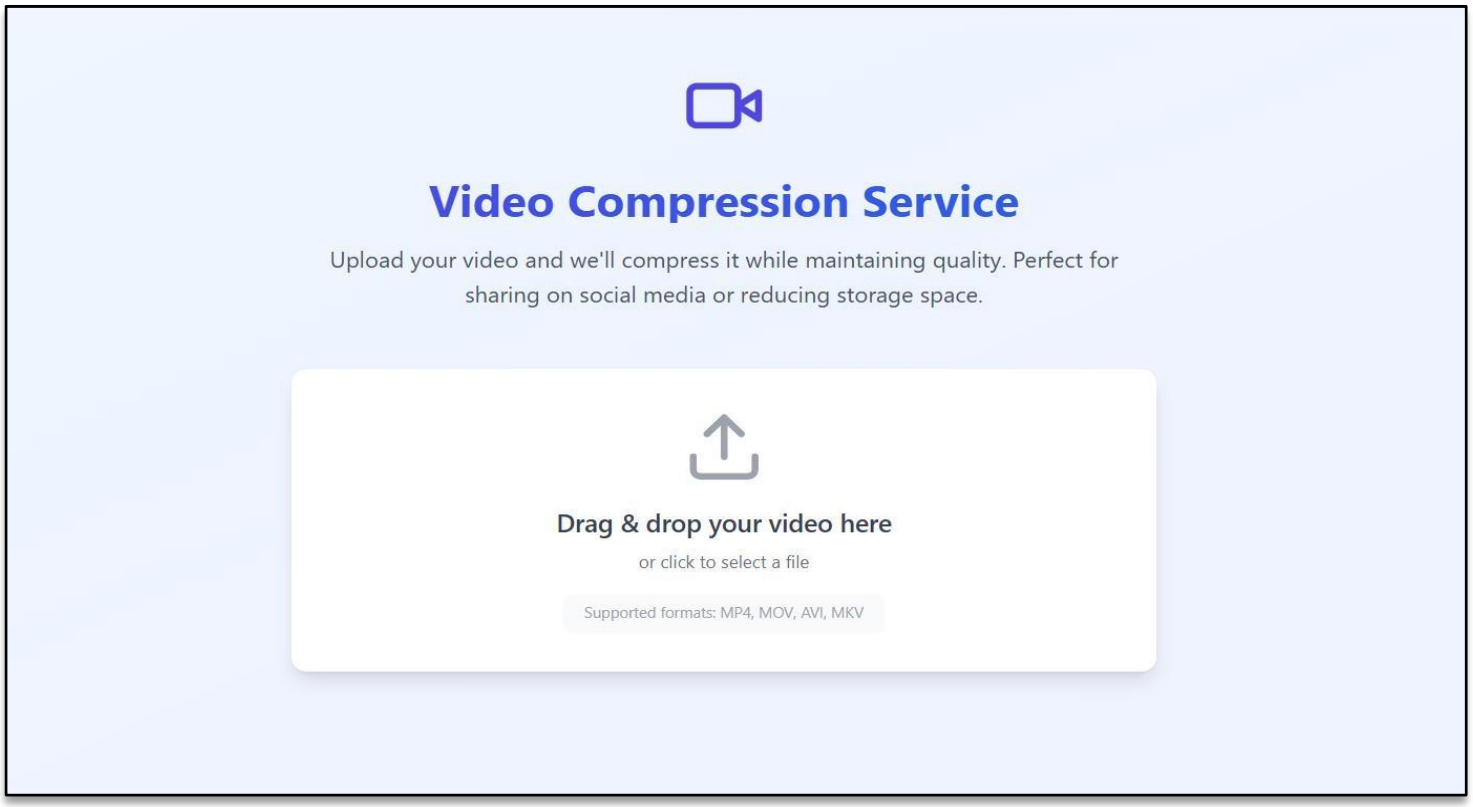


Displays a successful response (200 OK) with a 1m 48.06s processing time and 795 B response size. The JSON response includes:

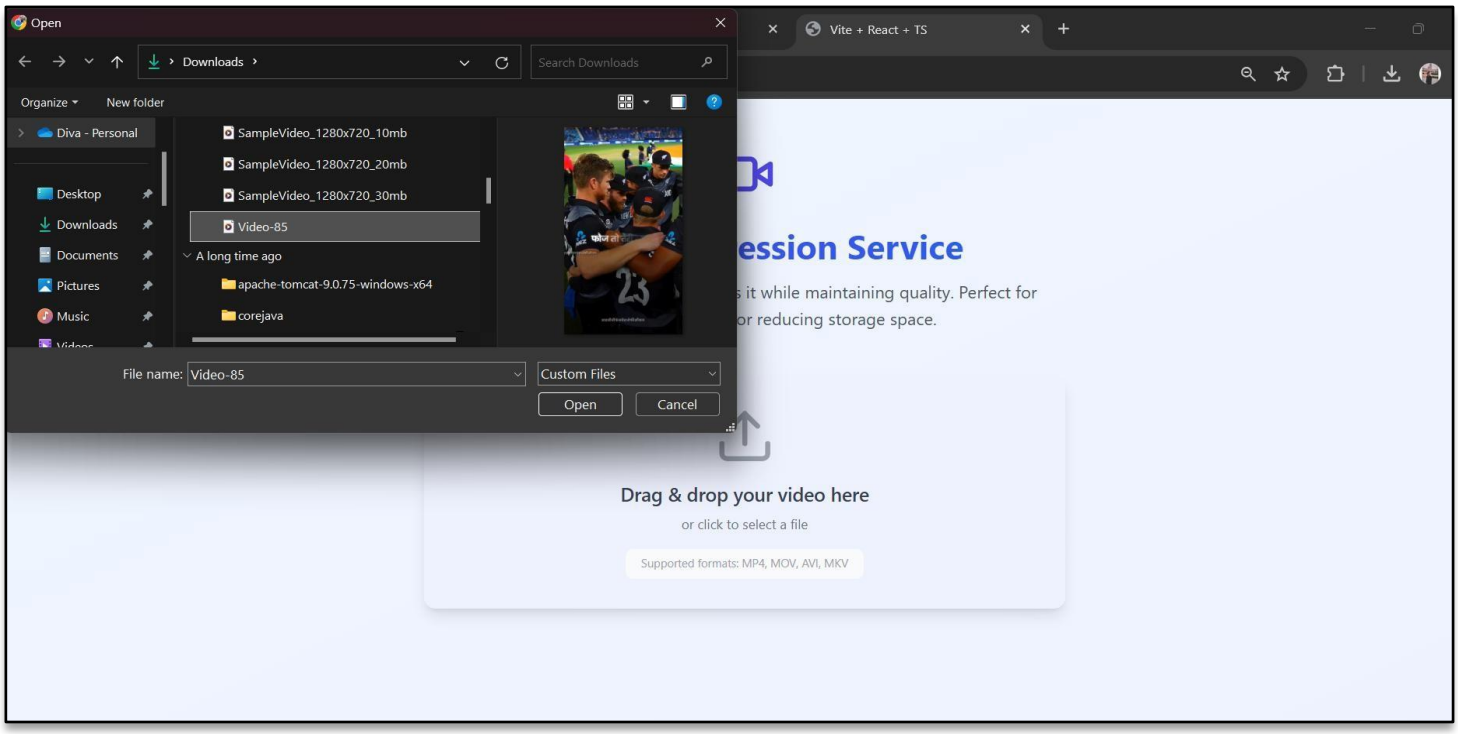
- A "message" confirming "Compression successful"
- A "presignedUrl" pointing to your S3 bucket, containing AWS authentication parameters and request signatures
- The presigned URL is time-limited (3600 seconds validity) for secure access

4.3 Frontend Testing Using Chrome Browser

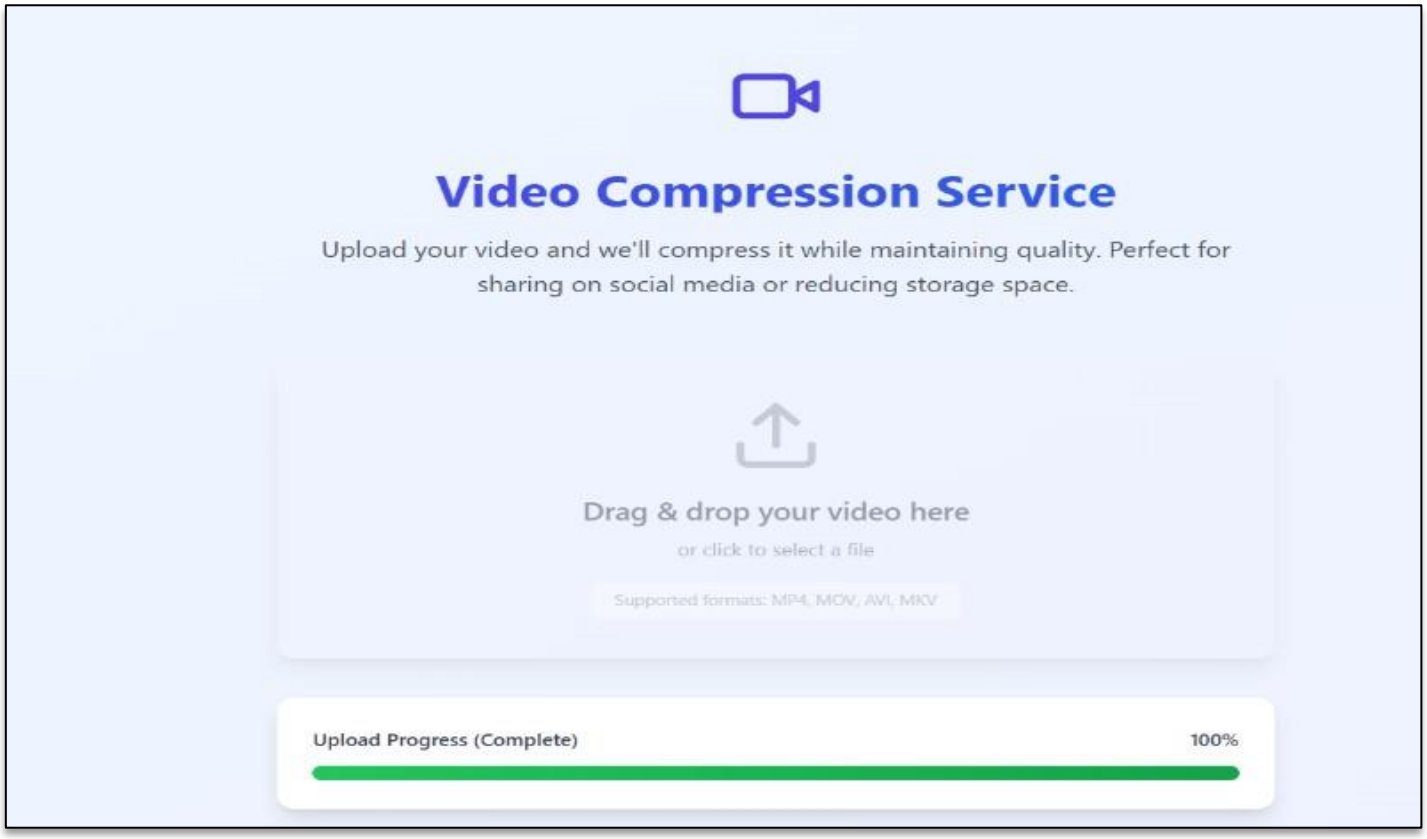
(Screenshot 4.6- Show clean frontend view of video compression tool)



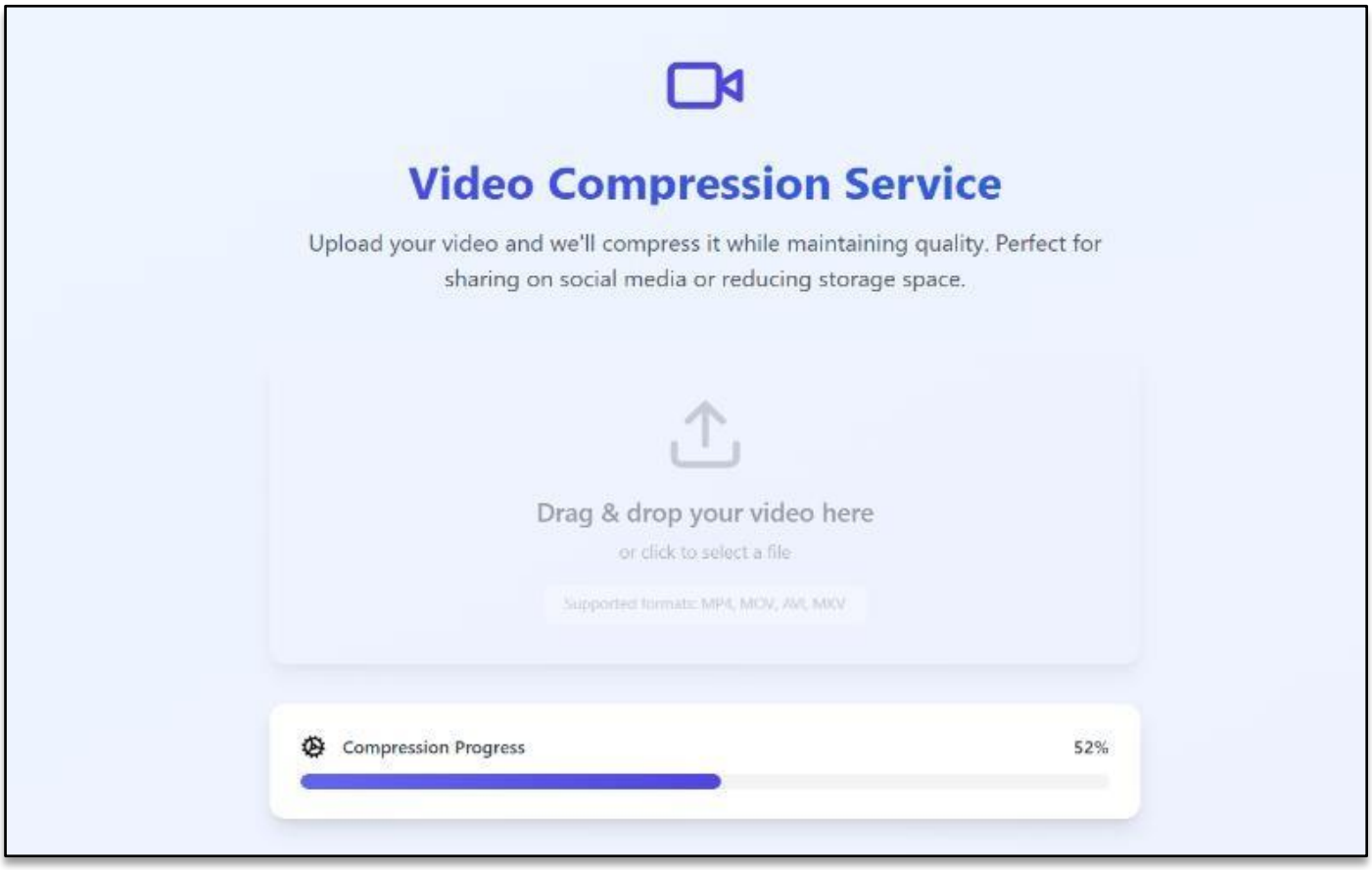
(Screenshot 4.7- Displays the file selection dialog integrated with the system's file browser, allowing users to navigate to and select videos for compression)



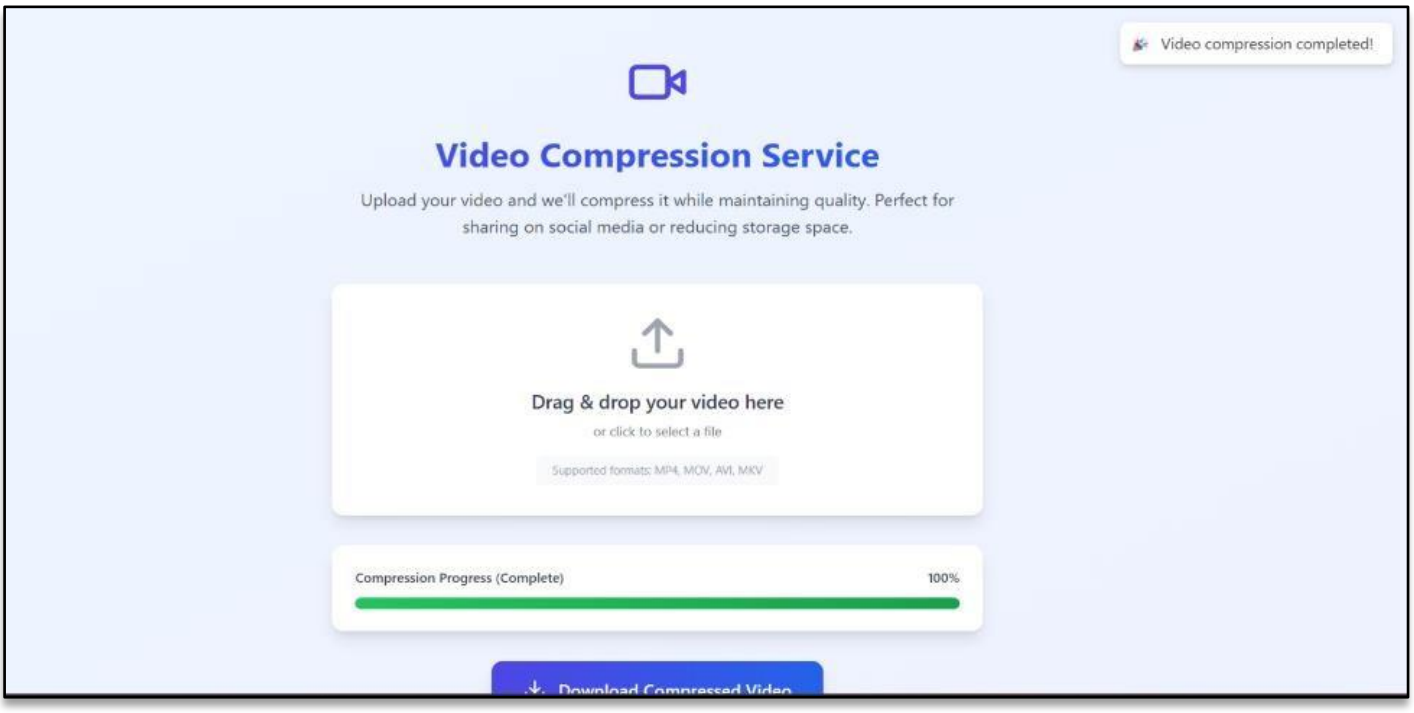
(Screenshot 4.8-Shows the user interface with a 100% complete upload progress bar, providing visual feedback about the file transfer stage.)



(Screenshot 4.9- Displays the compression process at 52% completion with a purple progress bar, indicating real- time feedback during the FFmpeg processing stage.)



(Screenshot 4.10- 100% compression has been done.)



Shows the final state with:

- A "Video compression completed!" success message
- A 100% complete progress bar
- A prominent "Download Compressed Video" button for retrieving the processed file

Through rigorous testing, this project has proven to be efficient, scalable, and user-friendly. The compression performance results indicate that the system achieves high compression ratios without significant loss of quality. Testing via Postman for backend API and functional testing on the frontend has validated that the application is robust and reliable.

5. ADVANTAGES, DISADVANTAGES AND LIMITATIONS

5.1 Advantages

- Efficient Video Compression:
    - The product successfully reduces video file sizes while maintaining reasonable quality.
    - Compressed videos take up less storage and require lower bandwidth for transmission.
- Functional on a Low-End System:
    - The tool runs efficiently on an i3-core CPU, making it accessible for students and small-scale users.
    - No requirement for high-end GPUs or expensive hardware to perform basic compression tasks.
- User-Friendly Interface:
    - The tool has a simple frontend, making it easy for users to upload and compress videos.
    - Users can interact with the backend using Postman API testing or through the frontend interface.
- Backend Flexibility:
    - The backend is tested and verified for proper functionality using Postman.
    - The system can be integrated with cloud-based hosting in the future, ensuring scalability.
- Potential for Future Enhancements:
    - The product is containerized using Docker, allowing for seamless deployment.
    - Plans to host the frontend and backend on AWS EC2 will make it globally accessible.
    - Features like user-controlled compression cancellation can enhance usability.

5.2 Disadvantages

- Limited Processing Power:
    - As the current system is running on a low-end i3-core CPU, the compression speed is slower compared to professional tools.
    - Processing large video files may take significantly more time.
- Limited Support for High-Resolution Videos:
    - While the tool efficiently handles small to medium-sized videos, 4K or higher-resolution videos may cause delays or crashes.
- No Real-Time Compression
    - The current implementation does not support real-time video compression for live-streaming applications.

    - A future update with hardware acceleration (e.g., GPU encoding) can improve real-time processing.
- Lack of Advanced Compression Features:
    - The tool currently compresses videos based on predefined parameters, lacking customizable bitrate control or adaptive compression options.
    - Future iterations can include AI-based adaptive compression to enhance efficiency.

5.3 Limitations

- High Processing Time for Large Files
    - **Limitation:** Compressing high-resolution videos (1080p, 4K) takes longer due to CPU constraints.
    - **Solution:** Implement hardware acceleration using GPUs (e.g., NVIDIA NVENC, Intel Quick Sync) or shift processing to cloud-based services.
- **Dependency on Local Hardware**
    - **Limitation:** The tool currently relies on local CPU processing, which limits scalability.
    - **Solution:** Using cloud-based processing (AWS Lambda, Google Cloud Functions) can offload video compression to powerful remote servers.
- **Storage and Bandwidth Constraints**
    - **Limitation:** Compressed videos still require local storage, and transferring large files consumes bandwidth.
    - **Solution:** Integrate cloud storage (AWS S3, Google Drive) to store and manage compressed files online.
- **No Real-Time Video Processing**
    - **Limitation:** The current system does not support live-streaming video compression.
    - **Solution:** Implement real-time encoding techniques (FFmpeg with HLS streaming) and use dedicated encoders for low-latency compression.

By integrating these solutions, video compression can be optimized for various applications, ensuring faster and more efficient video processing.

6. Conclusion & Future Scope

This project successfully demonstrates an efficient video compression system tailored for small to medium-scale use. By leveraging optimized compression techniques, the tool significantly reduces video file sizes while maintaining quality, making it practical for storage optimization and bandwidth savings. Despite being built on a low-end i3-core CPU, the system delivers functional compression, tested rigorously using Postman for backend validation and a working frontend interface. The project highlights the potential of accessible, cost-effective video compression solutions that cater to students, developers, and small businesses.

Looking ahead, several enhancements can further refine this system. AI-driven adaptive compression will intelligently adjust bitrate and resolution based on video complexity, improving efficiency without sacrificing quality. User-controlled compression cancellation will provide greater flexibility, allowing users to stop the process when necessary. Additionally, containerizing the application with Docker and deploying it on AWS EC2 will ensure scalability and accessibility to a broader audience. By integrating these advancements, this project can evolve into a robust, high-performance video processing solution with real-world applications across various industries.

Reference

[1] Springer Nature, “Visual Data Compression: The Ever-Changing Interdisciplinary Field for Efficient and Sustainable Storage and Transmission,” *Springer Nature Communities*, 2024. [Online]. Available: https://communities.springernature.com/posts/visual-data-compression-the-ever-changing-interdisciplinary-field-for-efficient-and-sustainable-storage-and-transmission.

[2] Network Innovations, “Four Video Compression Challenges and How Specta Integra Simplifies Them,” *Network Innovations Blog*, 2024. [Online].
Available: https://blog.networkinnovations.com/four-video-compression-challenges-and-how-specta-integra- simplifies-them.

[3] ImageKit, "A Detailed Overview Of Popular Video Compression Techniques," [Online].
Available: https://imagekit.io/blog/video-compression-techniques/.

[4] Wikipedia, "Data compression," [Online].
Available: https://en.wikipedia.org/wiki/Data_compression.

[5] BoxCast, "HEVC (H.265) vs. AVC (H.264): What's the Difference?," [Online].
Available: https://www.boxcast.com/blog/hevc-h.265-vs.-h.264-avc-whats-the-difference.

[6] Z3 Technology, "Is H.265 Better Than H.264?," [Online]
Available: https://z3technology.com/news/h265-better-h264/.

[7] EE Times, "An Overview of Video Compression Algorithms," [Online].
Available: https://www.eetimes.com/an-overview-of-video-compression-algorithms/.

[8] FFmpeg Documentation, *FFmpeg: A Complete, Cross-Platform Solution to Record, Convert, and Stream Audio and Video*, [Online].
Available: https://ffmpeg.org/documentation.html

[9] R. Patel and A. Gupta, "Comparative Study of Traditional Video Compression Techniques and FFmpeg-based Optimization," in *Proc. Int. Conf. Multimedia Comput. Process.*, 2021.

[10] Abirami N Natesan, "A Detailed Study of Client-Server and its Architecture," in *Proc. Int. Conf. Multimedia Comput. Process.*, 2019. [online]
Available: https://www.researchgate.net/publication/337469571_A_Detailed_Study_of_Client-Server_and_its_Architecture.

[11] Meta (Facebook), *React Documentation: A JavaScript Library for Building User Interfaces*, [Online]. Available: https://react.dev/.

[12] OpenJS Foundation, *Node.js Documentation: JavaScript Runtime Built on Chrome’s V8 Engine*, [Online]. Available: https://nodejs.org/en/docs.

[13] Amazon Web Services, *Amazon Simple Storage Service (S3) Overview*, [Online]. Available: https://docs.aws.amazon.com/s3/.

[14] A. Bovik, *Handbook of Image and Video Processing*, 2nd ed., Academic Press, 2005.