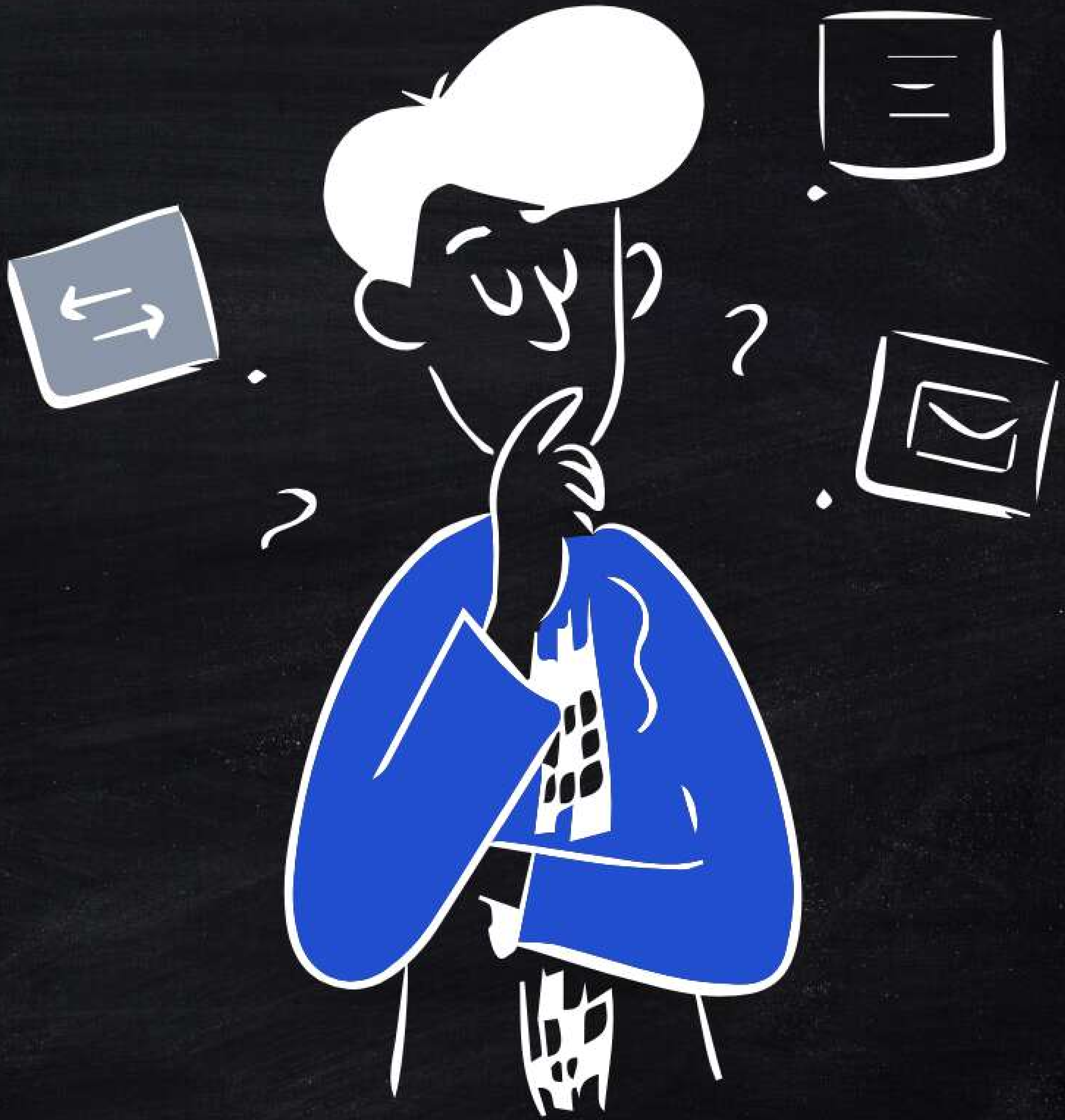
Efficient File Compression Using Huffman Encoding

Discover how Huffman encoding revolutionizes data storage efficiency through advanced file compression techniques for better performance.

Ashish Gupta , Aditya Kimothi , Sameer Lohani

PROBLEM STATEMENT

Large files consume excessive storage , affecting efficiency and increasing costs.
Existing compression tools are often closed-source and not suited for learning purposes.
We aim to develop a simple, open-source file compression tool.



OBJECTIVES:

- The growing size of digital files poses challenges in storage and data transfer. There is a need for an efficient, lossless method to reduce file sizes while maintaining data integrity and accessibility.

- Proprietary compression tools limit accessibility and customization for learners and developers. There is a need for an open-source solution that is lightweight, transparent, and educational, enabling efficient file compression and better understanding of compression algorithms.

- Many existing compression tools have complex interfaces that hinder ease of use. There is a need for a minimal, intuitive UI that allows users to compress and decompress files quickly without unnecessary features or confusion.

WORKFLOW

1. Upload File

Initiate the process by selecting the file to be compressed.

2. Huffman Encoding (C++)

Apply Huffman encoding algorithm in C++ to compress the uploaded file.

3. Download Compressed File

Once encoding is complete, download the newly compressed file.

INNOVATION

Educational transparency of the full compression pipeline

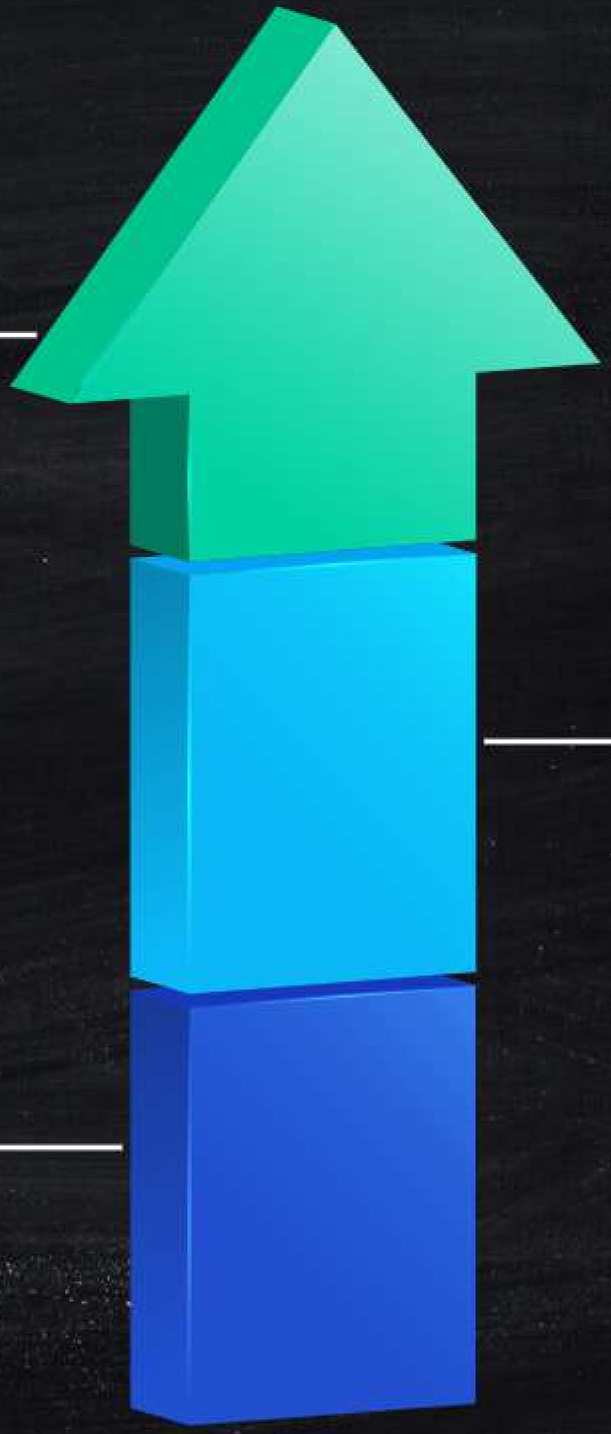
Flexibility to extend to more complex formats and use cases

Open Source

File Compression System Architecture

- **Frontend File Upload Form**
  The interface where users can upload files for compression.
- **Node.js Server**
  Handles routing and serves frontend and backend requests efficiently.
- **C++ Engine for Compression**
  Performs the core logic for compressing and decompressing files.
- **File Storage System**
  Temporarily stores input and output files during processing.

Technical Progress



UPCOMING WORK

The focus for the coming days will be on completing frontend-backend integration, performing extensive testing on various file types, and finalizing documentation. With continued progress, we expect to complete the project within the remaining timeline.

CURRENT SITUATION

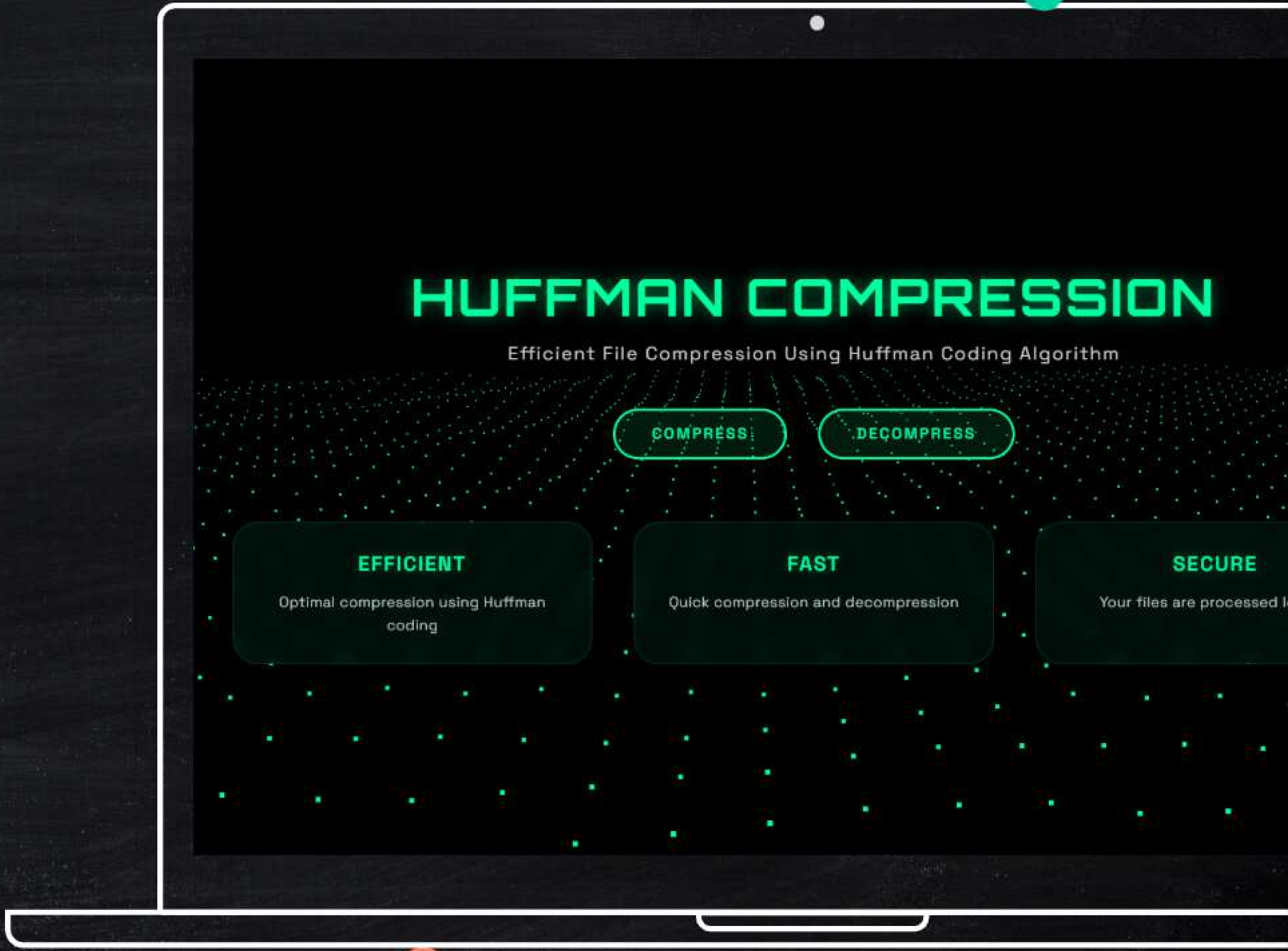
As of now, a significant portion of the project has been completed. The core C++ compression and decompression algorithms using Huffman encoding have been successfully implemented and tested with sample files. Additionally, the web interface — including Home, Compress, and Decompress pages — has been designed and developed, meeting our planned milestones for frontend design.

CURRENT PROBLEMS

Currently, the project is slightly behind schedule in terms of integration. The connection between the frontend interface and the backend C++ engine is still in progress, as handling file I/O between Node.js and native C++ requires careful execution and testing. We are prioritizing this task now to stay on track for final testing.

A User opens the website

The user has to click on compress or decompress as per their requirement and this will redirect them to their desired algorithm page .



HUFFMAN COMPRESSION

Efficient File Compression Using Huffman Coding Algorithm

EFFICIENT

Optimal compression using Huffman coding

FAST

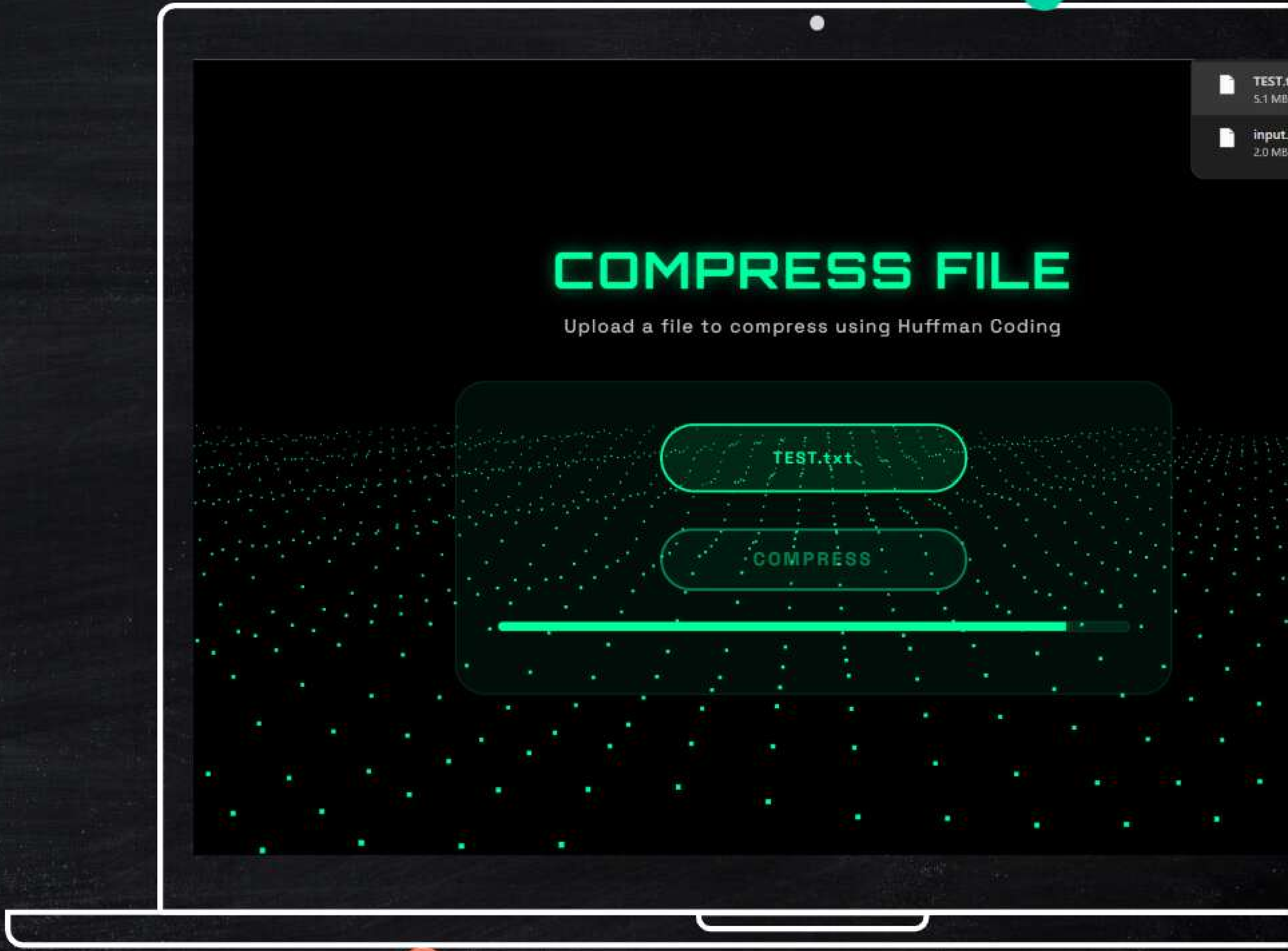
Quick compression and decompression

SECURE

Your files are processed l

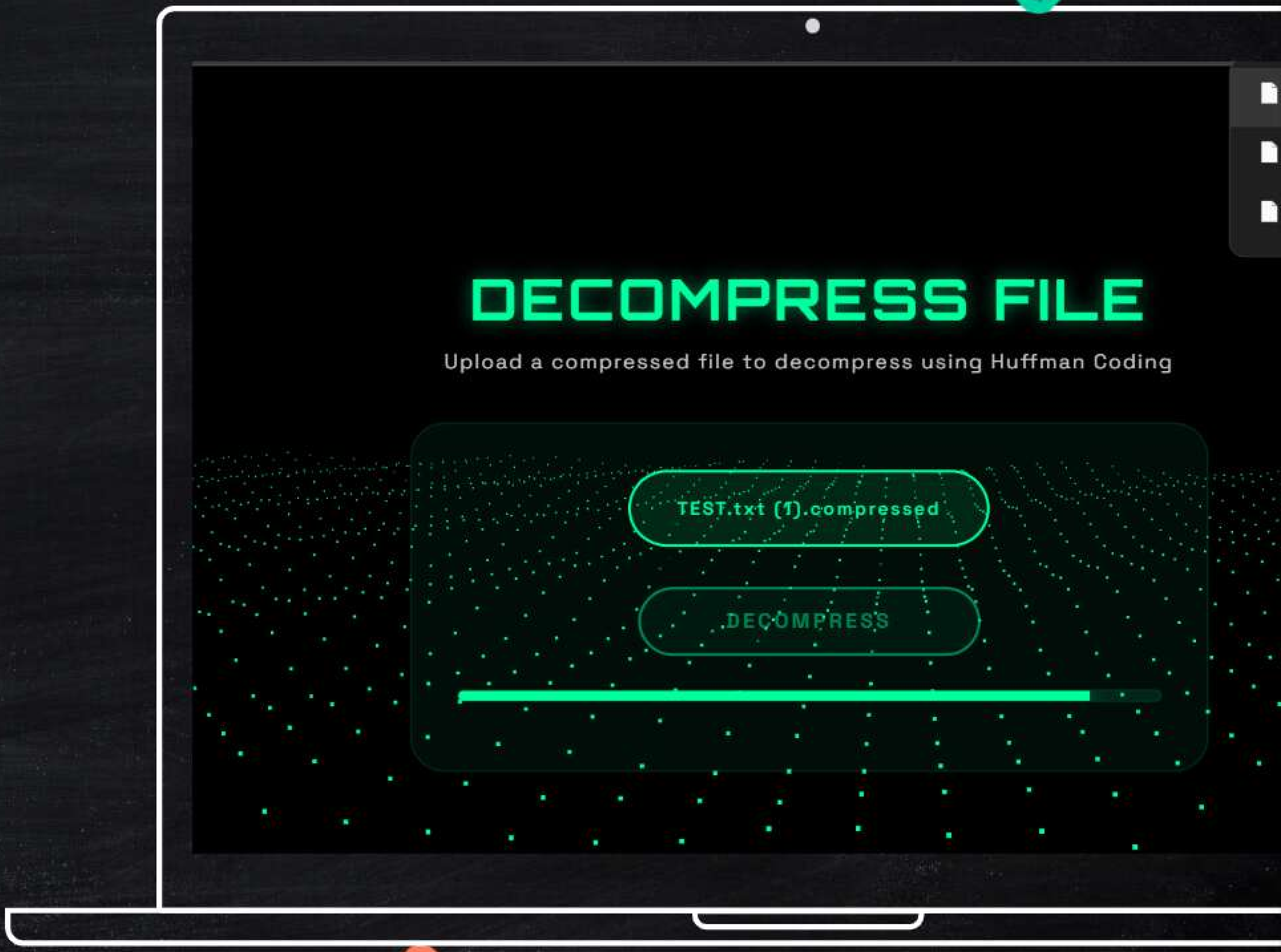
IF User Clicks on Compress

If the user clicks on compress then user will be redirected to the compression page where the user has to upload the file.



After Compression User Will Click on Decompress

The binary compressed file will be converted to decompressed txt file


DECOMPRESS FILE

Upload a compressed file to decompress using Huffman Coding

Future Scopes of Huffman Encoding

Enhancements and Expansions

Support for Multiple File Formats

Extend the tool to handle various formats including images, PDFs, and multimedia files. Implement logic to determine optimal strategies for different file types, possibly combining Huffman with other algorithms.

Real-Time Compression Statistics

Provide users with detailed feedback such as compression ratio, space saved, and processing time. Visual analytics (charts/graphs) can enhance understanding of performance.

Enhanced User Interface

Add features like drag-and-drop file upload, progress bars, and multilingual support to improve interactivity and accessibility. Consider integrating dark mode or customizable themes for better user experience.

Cloud Integration

Enable compression and decompression directly from cloud storage services, facilitating secure cloud-based compression for low-storage devices.

Mobile Application Version

Develop a lightweight app for Android/iOS to address mobile file management and compression needs.

Integration of Other Compression Algorithms

Introduce a modular backend to support alternative algorithms, allowing users to choose the most suitable one based on their requirements.

Multi-File and Folder Compression

Allow batch processing or folder compression, with ZIP-style packaging and extraction features.

Security and Encryption

Implement encryption options during compression to ensure file security, particularly for sensitive data, and add user authentication for secure usage.

Web Hosting and API Service

Host the application publicly with a REST API for third-party integration, considering an open-source SaaS model.

Performance Optimization

Further optimize the C++ implementation for large files using multithreading or memory mapping, and explore parallelism or GPU acceleration for faster processing.

Team Roles and Responsibilities

Ashish Gupta

Team Lead

Oversees Backend C++ Development, Frontend Development, and Project Coordination.

Aditya Kimothi

Team Member

Focuses on Web Integration and Frontend Development.

Sameer Lohani

Team Member

Handles Backend C++ Development, Testing, and Documentation.

Role-wise Contributions

Design and algorithm implementation

Algorithm For Compression

Completed By: Ashish Gupta

Algorithm For Decompression

Completed By: Sameer Lohani

Role-wise Contributions

Web Pages

Home Page

Completed By: Sameer Lohani

For Compression

Completed By: Aditya Kimothi

For Decompression

Completed By: Ashish Gupta

Role-wise Contributions

Design and algorithm implementation

Algorithm For Compression

Completed By: Ashish Gupta

Algorithm For Decompression

Completed By: Sameer Lohani

Role-wise Contributions

Ashish Gupta

Team Lead
Oversees Backend C++ Development, Frontend Development, and Project Coordination.

Aditya Kimothi

Team Member
Focuses on Web Integration and Frontend Development.

Sameer Lohani

Team Member
Handles Backend C++ Development, Testing, and Documentation.

Thanks for Being Here with Us

We appreciate your support

TEAM BYTESQUASHERS