Chandigarh Engineering College Jhanjeri
Mohali-140307
Department of Artificial Intelligence and Data Sciences

PROJECT REPORT
ON

Design of a Multithreaded Application

Final Project

Department of Artificial Intelligence & Data Science
CHANDIGARH ENGINEERING COLLEGE JHANJERI, MOHALI

In partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Artificial Intelligence & Data Science

SUBMITTED BY:	**Under the Guidance of**
Viren, Vishal, Yanvi, Yogesh, Anchal	Dr. Kunal Gagneja
2330792, 2330793, 2330794, 2330795, 2330241	Assistant Professor, CEC

MAY, 2025

Affiliated to I.K Gujral Punjab Technical University, Jalandhar
(Batch: 2023-2027)

Chandigarh Engineering College Jhanjeri
Mohali-140307
Department of Artificial Intelligence and Data Sciences

DECLARATION

Hereby, Viren, Vishal, Yanvi, Yogesh, Anchal, declare that the report of the project entitled " Design of a multithreaded application" has not presented as a part of any other academic work to get my degree or certificate except Chandigarh Engineering College Jhanjeri, Mohali, affiliated to I.K. Gujral Punjab Technical University, Jalandhar, for the fulfillment of the requirements for the degree of B.Tech in Artificial Intelligence & Data Science .

Viren, Vishal, Yanvi, Yogesh, Anchal	**Mr. Kunal Gagneja**
2330792, 2330793, 2330794, 2330795, 2330241	Assistant Professor, CEC
Semester: 4th	

Signature of the Head of Department

ACKNOWLEDGEMENT

It gives us great pleasure to deliver this report on Project, which we worked on for our B.Tech in Artificial Intelligence & Data Science 2nd year, which was titled "Design of a multithreaded application “. We are grateful to my university for presenting me with such a wonderful and challenging opportunity. We also want to convey my sincere gratitude to all coordinators for their unfailing support and encouragement.

We are extremely thankful to the HOD Dr. Gurpreet Singh and Project Mentor Dr. Kunal Gagneja of Artificial Intelligence & Data Science at Chandigarh Engineering College Jhanjeri, Mohali (Punjab) for valuable suggestions and the heartiest cooperation.

We are also grateful to the management of the institute and Dr. Avinash (Director Engineering) for giving us the chance to acquire the information. We also appreciate all of my faculty members, who have instructed me throughout my degree.

Viren, Vishal, Yanvi, Yogesh, Anchal

TABLE OF CONTENTS

Abstract

The **Word Count File Processor Web Application** is a lightweight, browser-based tool designed to analyse the content of .txt files and display the total word count for each uploaded file. Built using **HTML**, **CSS**, and **vanilla JavaScript**, the application allows users to upload multiple text files simultaneously and processes them asynchronously within the browser environment.

Although JavaScript is single-threaded by nature, the project leverages **asynchronous file handling (FileReader API)** and **UI-level concurrency (setInterval-based progress bars)** to simulate a **multithreaded experience**. Each file's word count is processed in the background without blocking the main thread, and a dynamic progress bar is displayed for visual feedback. This creates the illusion of parallel processing and significantly enhances the user experience.

The application's frontend is styled for clarity and responsiveness, making it suitable for educational, analytical, and lightweight data-processing use cases. It does not require any backend or server deployment, ensuring easy portability and accessibility across devices.

In conclusion, this project demonstrates how core web technologies can be combined to build an efficient and user-friendly tool that simulates multithreaded behaviour in a single-threaded environment.

List of Figures

Chapter-1:
INTRODUCTION

In today's digital landscape, handling analyzing text-based data efficiently is a critical requirement across domains such as education, content management, and data processing. Web-based tools that offer real-time processing capabilities have become increasingly popular due to their accessibility and ease of use.

This project, titled **"Word Count File Processor Web App",** is developed using fundamental web technologies — HTML, CSS, and JavaScript. It allows users to upload multiple .txt files directly from their local system and instantly view the word count for each file.

The application simulates multithreaded behavior using JavaScript's asynchronous features and includes a progress bar for each file, creating a smooth and responsive user experience.

The following sections provide a deeper understanding of the technologies and methodologies used in the application.

1.1 Understanding Client-Side File Processing:

Client-side file processing involves handling user-uploaded data entirely within the browser, without transferring it to a remote server. This improves privacy, performance, and speed.

In this project:

- The **FileReader API** is used to read `.txt` files directly in the browser.
- File content is parsed, and a **word count** is calculated using simple string manipulation and regular expressions.
- This approach ensures that all operations are **lightweight**, **secure**, and **instant**.

Advantages of client-side processing:

- No need for internet or backend server
- Immediate results
- Complete control over the user’s data (privacy)

1.2 Multithreading in JS:

JavaScript is inherently single-threaded, meaning it executes one command at a time. However, web applications can simulate **parallel processing** through:

- **Asynchronous operations** (e.g., FileReader.onload)
- **Timers like setTimeout() and setInterval()**
- **Visual elements** like progress bars to indicate processing activity

In this app:

- Each file is read using FileReader, which runs asynchronously.
- A progress bar is animated using setInterval() during the file processing.
- Once reading is complete, the bar is removed and word count is displayed.

This gives users the **illusion of multithreading**, where multiple tasks appear to be handled at the same time — enhancing interactivity and performance.

Chapter-2
REVIEW OF LITERATURE

Text processing and browser-based applications have evolved significantly with the advancement of web technologies. The ability to perform computations like word counting within a web browser, without the need for backend support, represents a shift towards more decentralized and user-friendly tools. This section reviews existing literature and technologies relevant to the development of this project.

2.1 Previous Work on Text File Processing

Text file processing has traditionally been implemented in programming languages like Python, Java, and C++, where large files are read from storage and parsed line-by-line to count words, characters, or patterns. These applications often depend on disk I/O and require compilation or runtime environments.
In contrast, web-based solutions using JavaScript are relatively recent. Libraries like **PapaParse** (for CSV) and frameworks like **ElectronJS** have enabled client-side or hybrid solutions, but they often rely on external dependencies or backend APIs. However, simple operations like word counting can be efficiently achieved using native browser capabilities — a technique explored in this project.

2.2 Client-Side Processing and Asynchronous JavaScript:

With the advent of HTML5 and modern JavaScript APIs, client-side processing has become feasible and effective. Instead of sending user data to a server, operations such as file reading, data parsing, and even limited computations can now be handled entirely within the browser.

Asynchronous JavaScript enables such applications to remain responsive, even while multiple tasks (such as reading large files) are in progress. This approach avoids the blocking of the main UI thread and provides a seamless user experience.

2.2.1 FileReader API and Event-Driven Execution:

The FileReader API is a standard browser feature used to asynchronously read the contents of files selected via an <input> element. It operates through an **event-driven model**, where actions such as onload, onerror, and onprogress are triggered during the file reading process.

In this project:

- readAsText(file) is used to extract raw text from .txt files.
- The onload event is handled to process the text and compute word counts.
- A progress bar is animated in parallel using setInterval, giving the impression of live background processing.

This combination of event-driven execution and UI feedback creates an experience similar to **multithreading**, despite JavaScript being single-threaded at its core.

3. Problem Definition and Objectives:

3.1 Problem Definition:

In the digital age, individuals and organizations frequently handle large volumes of textual data in the form of .txt files. A basic yet essential requirement is the ability to determine word counts in these files for analysis, content validation, or academic purposes. Existing solutions for this often involve downloading software or writing scripts in programming languages like Python or Java, which may not be accessible to non-technical users.

Hence, there exists a need for a simple, browser-based application that can quickly process multiple text files and provide accurate word counts without the need for any installations or backend support.

3.2 Project Objectives:

The primary objective of this project is to develop a client-side web application using HTML, CSS, and JavaScript that can:

- Allow users to upload multiple .txt files
- Process each file asynchronously and efficiently
- Count the number of words in each file
- Display the results with responsive and user-friendly UI
- Simulate multithreading using progress bars and asynchronous behavior
- Ensure ease of use, privacy (no data sent to server), and cross-platform compatibility

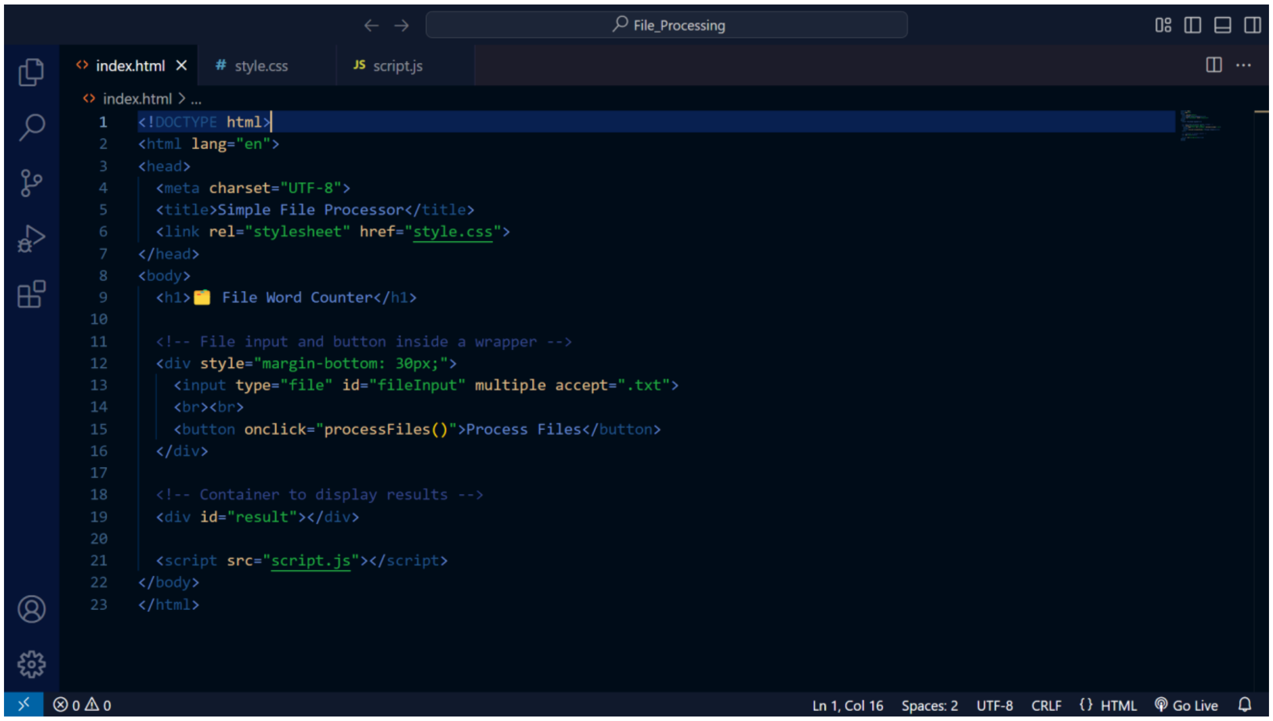
4. Design and Implementation:

4.1 Application Design:

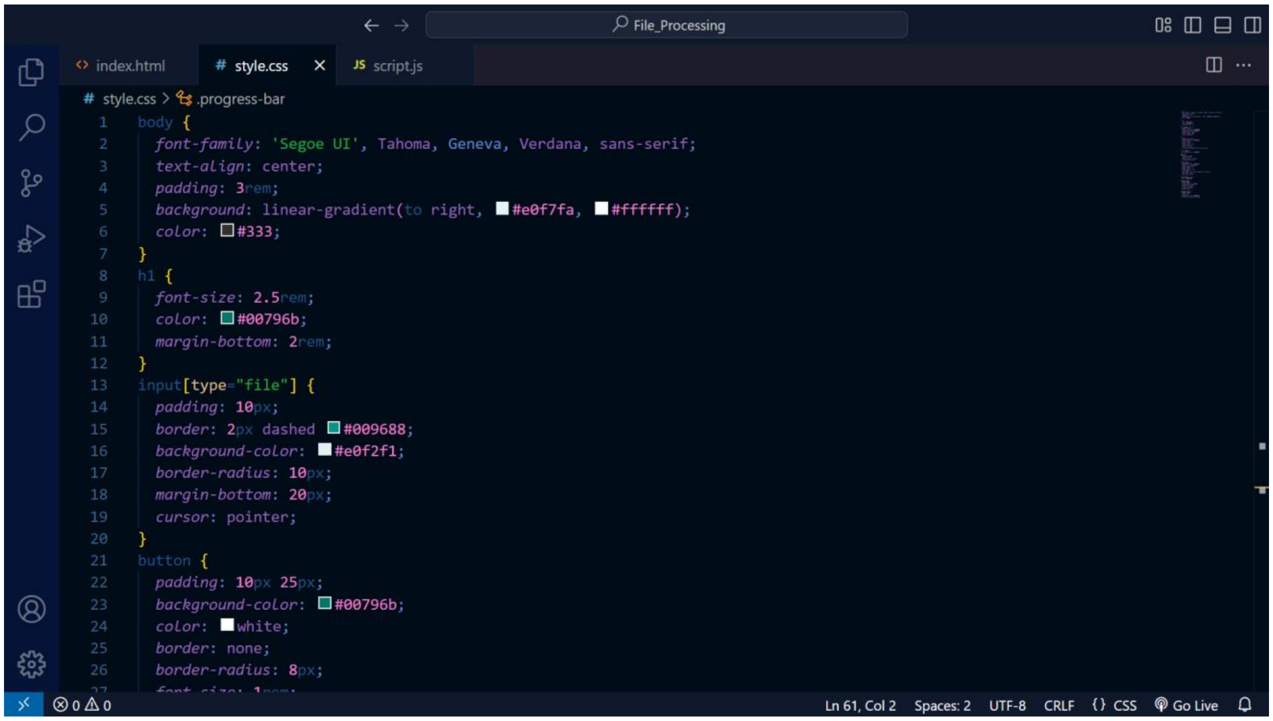
The application follows a modular design with clear separation between structure, style, and logic:

- HTML handles the basic structure and file input elements
- CSS is used to create a clean and modern interface, with progress bars for visual feedback
- JavaScript controls the entire functionality — from file reading to UI updates.

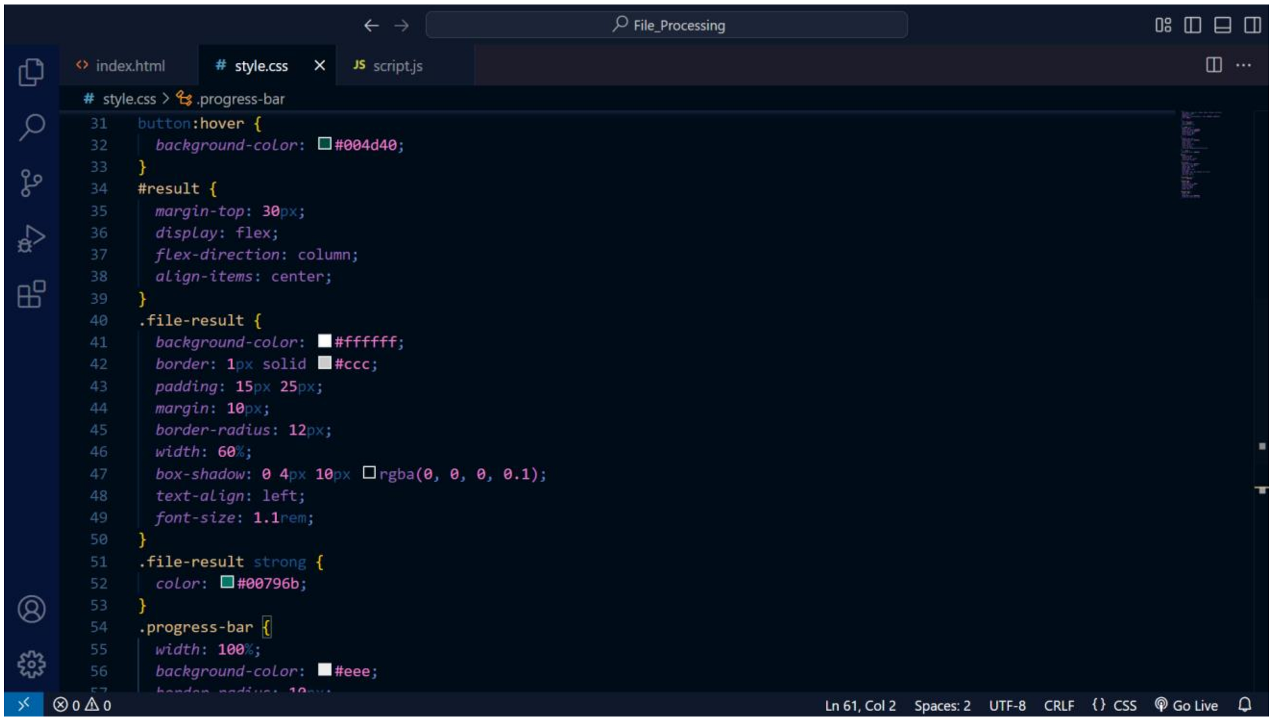
Each uploaded file is displayed individually along with a real-time progress bar, giving the user the perception of parallel processing. The final word count is presented clearly below each file name.



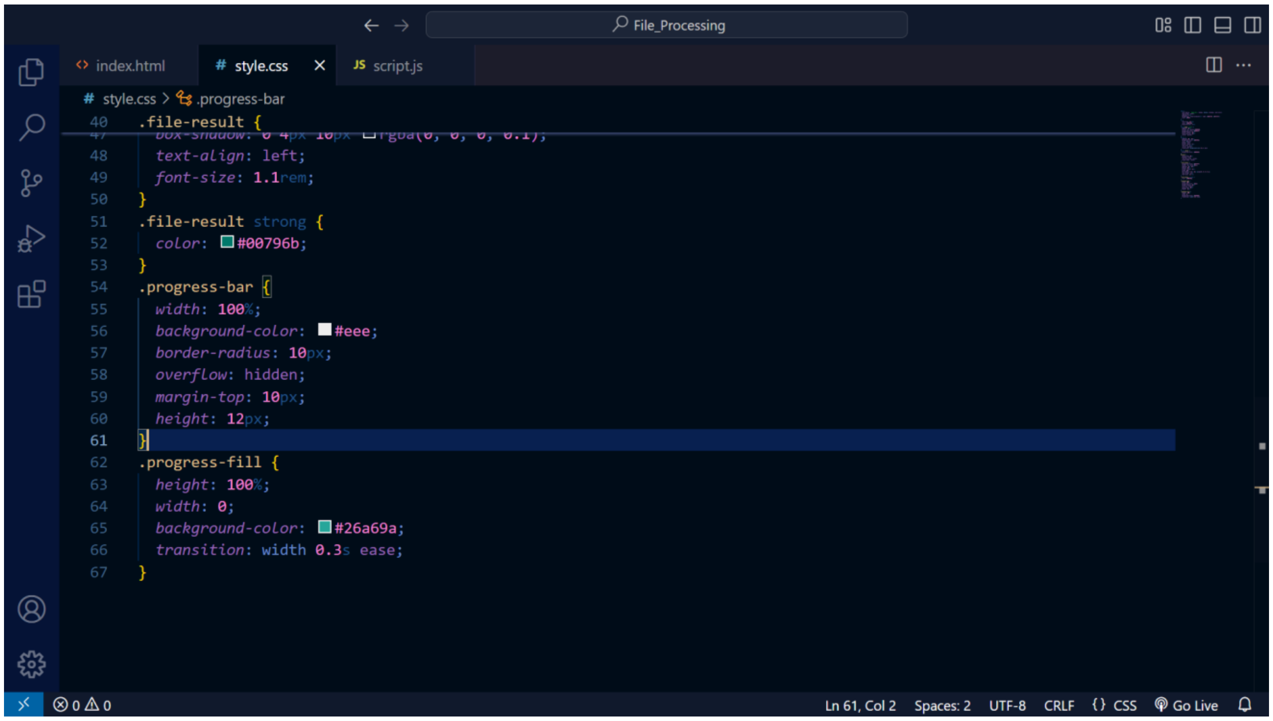
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Simple File Processor</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>📁 File Word Counter</h1>

  <!-- File input and button inside a wrapper -->
  <div style="margin-bottom: 30px;">
    <input type="file" id="fileInput" multiple accept=".txt">
    <br><br>
    <button onclick="processFiles()">Process Files</button>
  </div>

  <!-- Container to display results -->
  <div id="result"></div>

  <script src="script.js"></script>
</body>
</html>
```



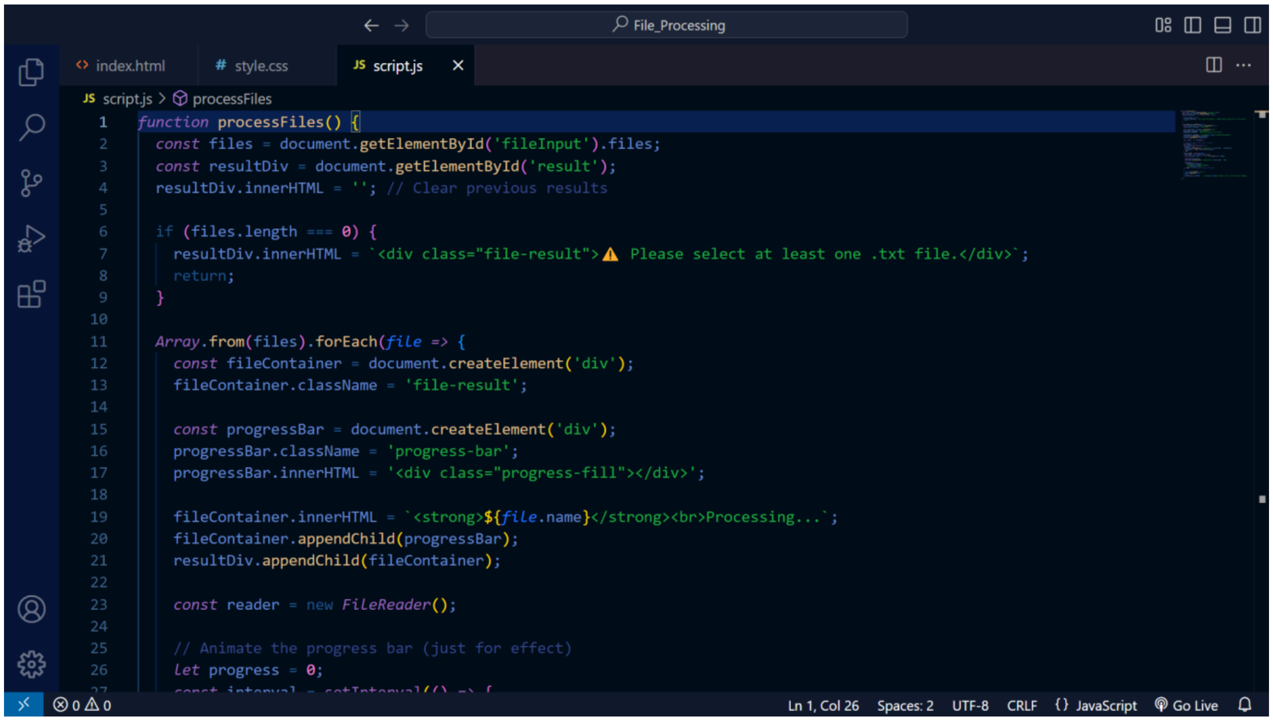
```
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  text-align: center;
  padding: 3rem;
  background: linear-gradient(to right, #e0f7fa, #ffffff);
  color: #333;
}
h1 {
  font-size: 2.5rem;
  color: #00796b;
  margin-bottom: 2rem;
}
input[type="file"] {
  padding: 10px;
  border: 2px dashed #009688;
  background-color: #e0f2f1;
  border-radius: 10px;
  margin-bottom: 20px;
  cursor: pointer;
}
button {
  padding: 10px 25px;
  background-color: #00796b;
  color: white;
  border: none;
  border-radius: 8px;
```



```
button:hover {
  background-color: #004d40;
}
#result {
  margin-top: 30px;
  display: flex;
  flex-direction: column;
  align-items: center;
}
.file-result {
  background-color: #ffffff;
  border: 1px solid #ccc;
  padding: 15px 25px;
  margin: 10px;
  border-radius: 12px;
  width: 60%;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  text-align: left;
  font-size: 1.1rem;
}
.file-result strong {
  color: #00796b;
}
.progress-bar {
  width: 100%;
  background-color: #eee;
```

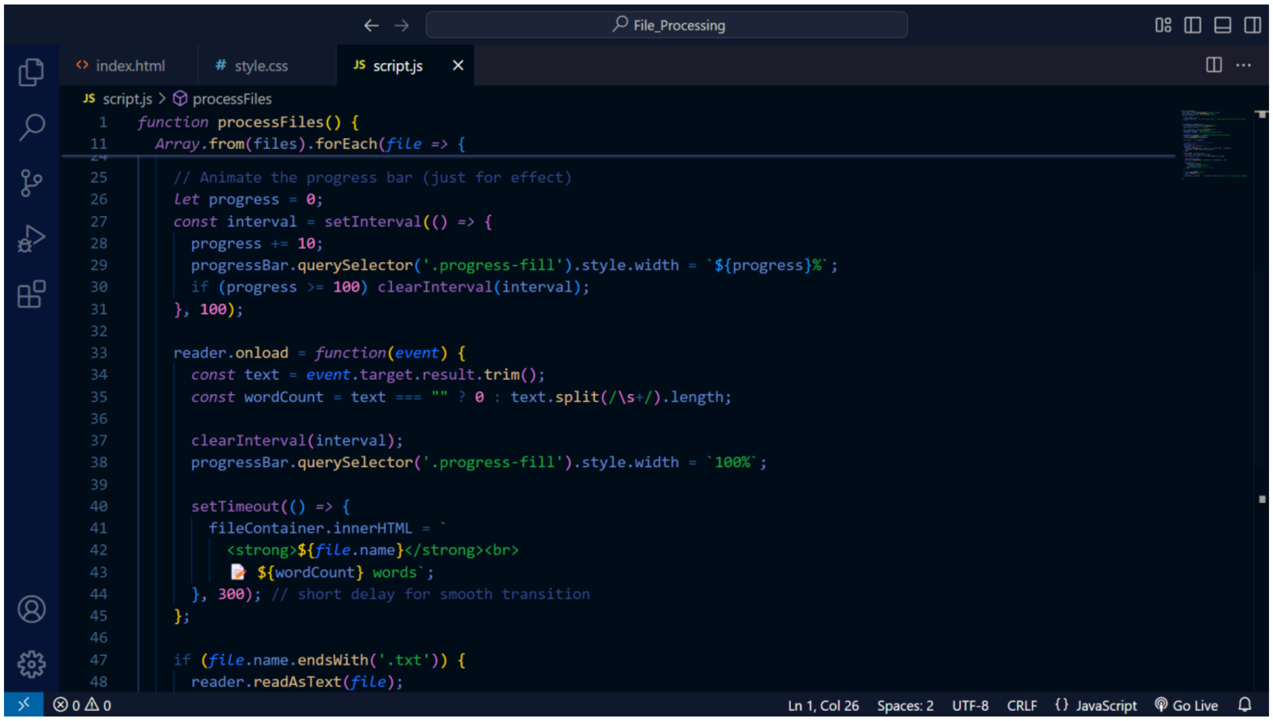


```css
.file-result {
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  text-align: left;
  font-size: 1.1rem;
}
.file-result strong {
  color: #00796b;
}
.progress-bar {
  width: 100%;
  background-color: #eee;
  border-radius: 10px;
  overflow: hidden;
  margin-top: 10px;
  height: 12px;
}
.progress-fill {
  height: 100%;
  width: 0;
  background-color: #26a69a;
  transition: width 0.3s ease;
}
```

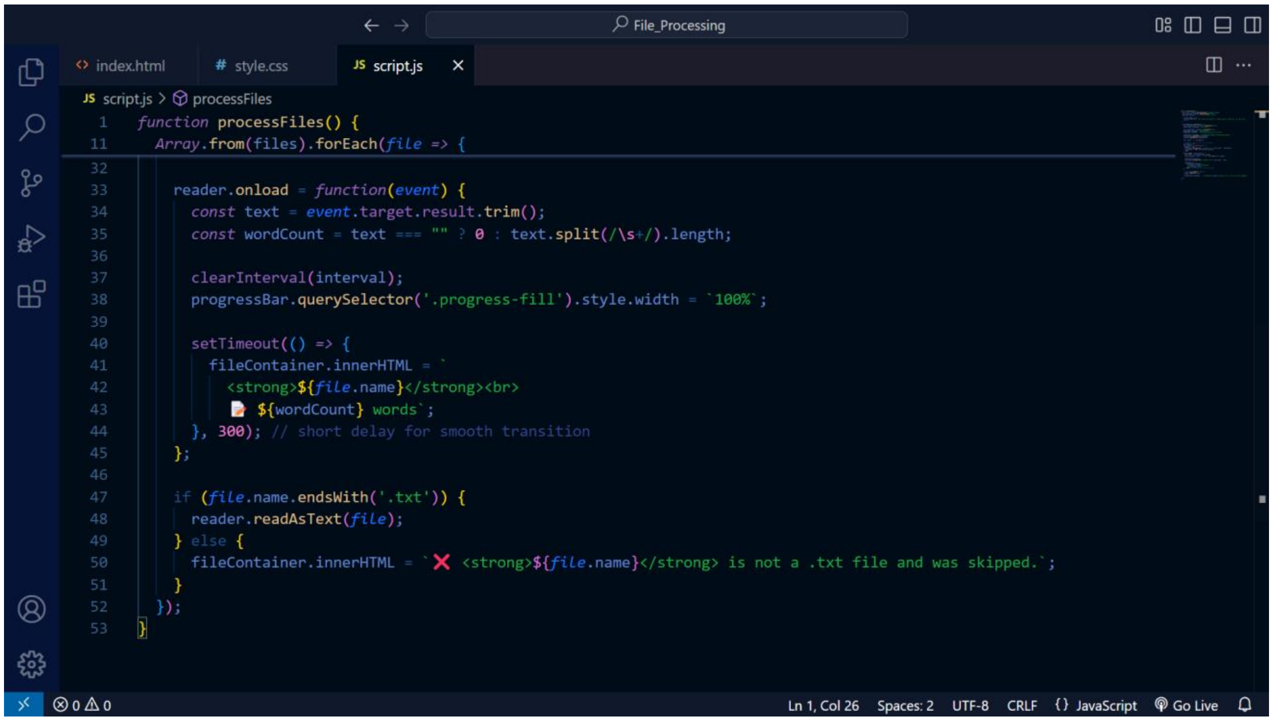


```javascript
function processFiles() {
  const files = document.getElementById('fileInput').files;
  const resultDiv = document.getElementById('result');
  resultDiv.innerHTML = ''; // Clear previous results

  if (files.length === 0) {
    resultDiv.innerHTML = `<div class="file-result">⚠️ Please select at least one .txt file.</div>`;
    return;
  }

  Array.from(files).forEach(file => {
    const fileContainer = document.createElement('div');
    fileContainer.className = 'file-result';

    const progressBar = document.createElement('div');
    progressBar.className = 'progress-bar';
    progressBar.innerHTML = '<div class="progress-fill"></div>';

    fileContainer.innerHTML = `<strong>${file.name}</strong><br>Processing...`;
    fileContainer.appendChild(progressBar);
    resultDiv.appendChild(fileContainer);

    const reader = new FileReader();

    // Animate the progress bar (just for effect)
    let progress = 0;
    const interval = setInterval(() => {
```

Chandigarh Engineering College Jhanjeri
Mohali-140307
Department of Artificial Intelligence and Data Sciences



```javascript
function processFiles() {
  Array.from(files).forEach(file => {
    // Animate the progress bar (just for effect)
    let progress = 0;
    const interval = setInterval(() => {
      progress += 10;
      progressBar.querySelector('.progress-fill').style.width = `${progress}%`;
      if (progress >= 100) clearInterval(interval);
    }, 100);

    reader.onload = function(event) {
      const text = event.target.result.trim();
      const wordCount = text === "" ? 0 : text.split(/\s+/).length;

      clearInterval(interval);
      progressBar.querySelector('.progress-fill').style.width = `100%`;

      setTimeout(() => {
        fileContainer.innerHTML = `
          <strong>${file.name}</strong><br>
          📝 ${wordCount} words`;
      }, 300); // short delay for smooth transition
    };

    if (file.name.endsWith('.txt')) {
      reader.readAsText(file);
```



```javascript
function processFiles() {
  Array.from(files).forEach(file => {

    reader.onload = function(event) {
      const text = event.target.result.trim();
      const wordCount = text === "" ? 0 : text.split(/\s+/).length;

      clearInterval(interval);
      progressBar.querySelector('.progress-fill').style.width = `100%`;

      setTimeout(() => {
        fileContainer.innerHTML = `
          <strong>${file.name}</strong><br>
          📝 ${wordCount} words`;
      }, 300); // short delay for smooth transition
    };

    if (file.name.endsWith('.txt')) {
      reader.readAsText(file);
    } else {
      fileContainer.innerHTML = `❌ <strong>${file.name}</strong> is not a .txt file and was skipped.`;
    }
  });
}
```

4.2 Implementation Details:

The core functionality revolves around the FileReader API, which reads .txt files asynchronously using readAsText(). Once the content is loaded:

- The word count is calculated using text.split(/\s+/) logic
- Progress bars are animated using setInterval() to simulate processing
- Asynchronous behavior ensures the interface remains responsive, even when processing multiple files

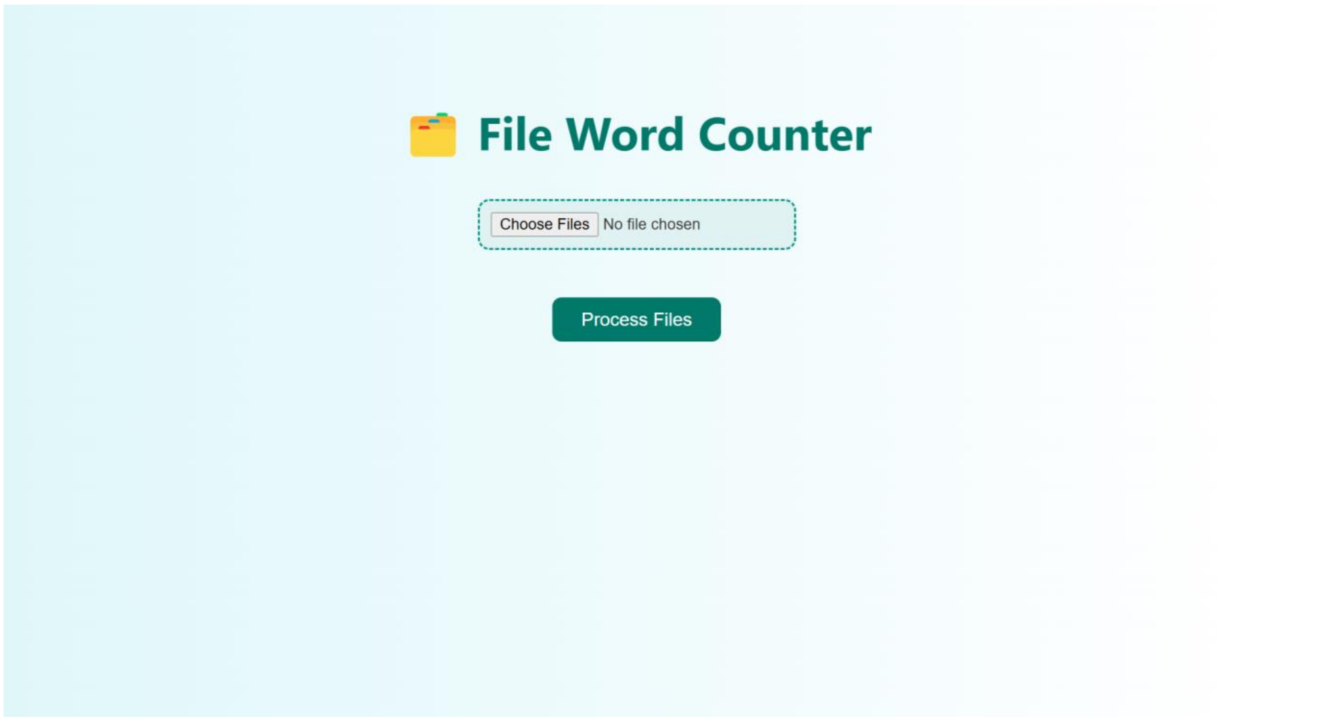
Error handling is included to skip unsupported file types and prevent crashes. The implementation avoids any backend processing, keeping the project fully client-side.

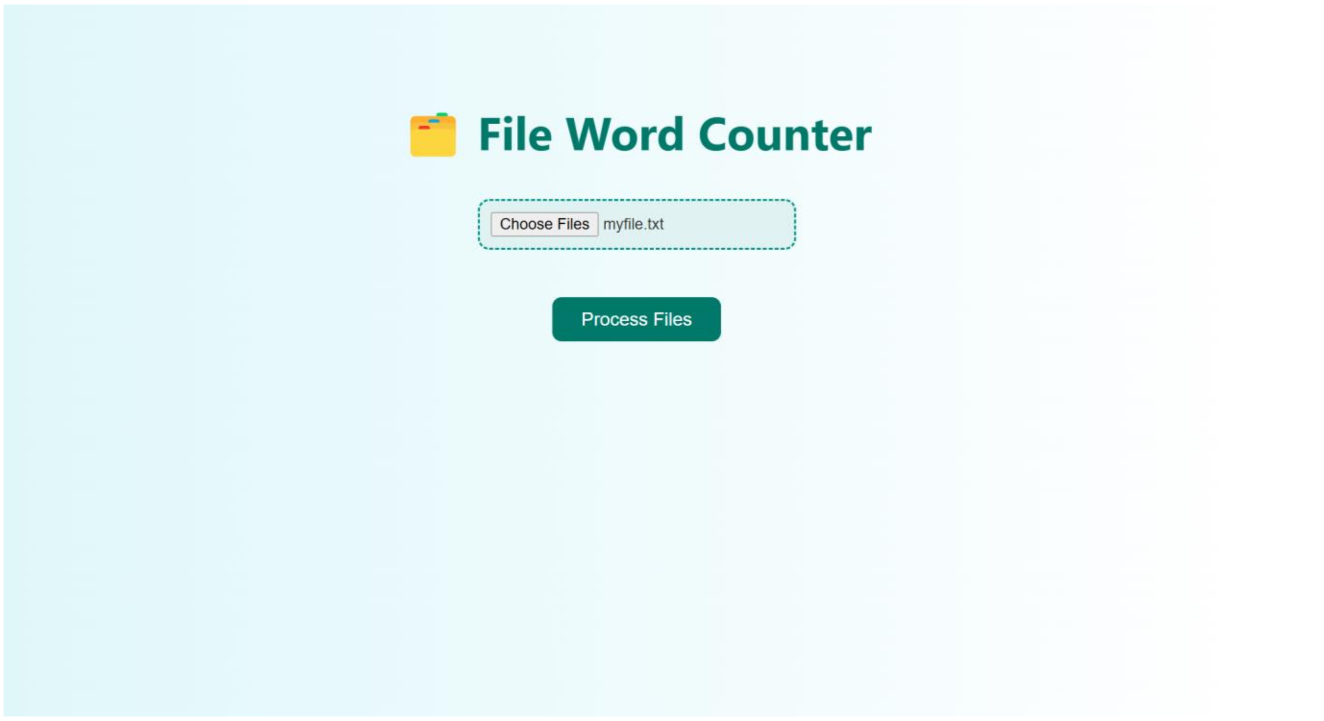
5. Results and Discussions:

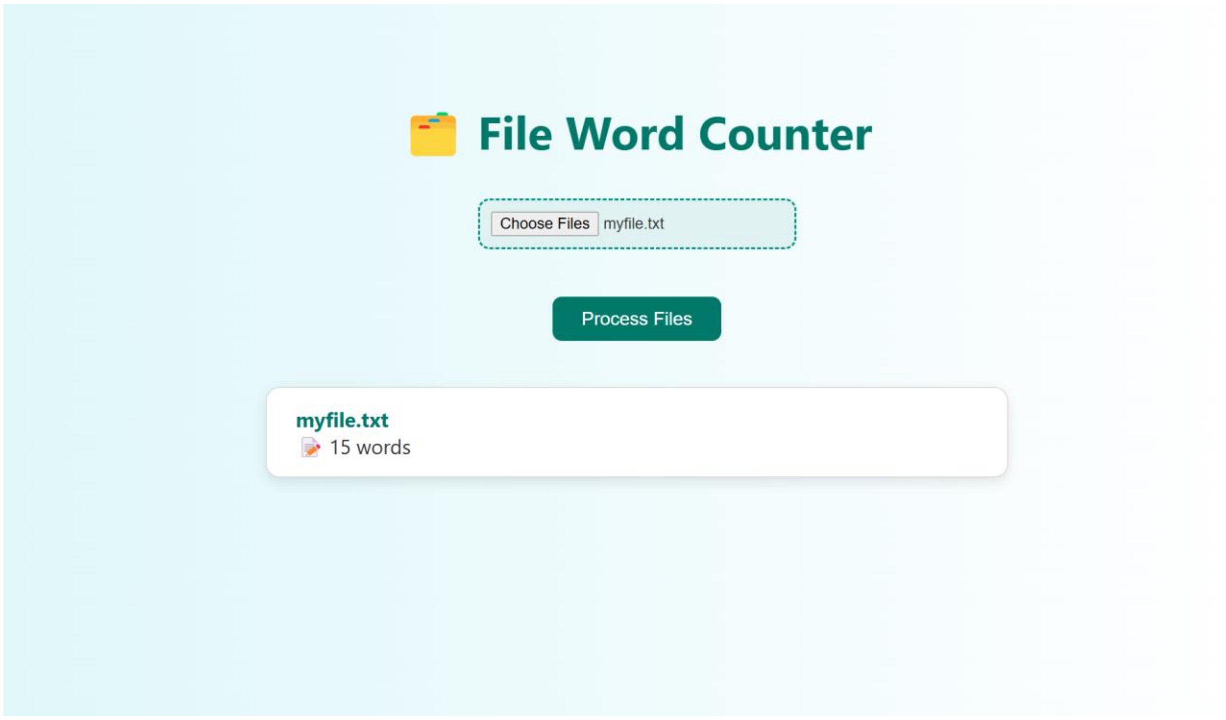
5.1 Output Summary:

Upon testing, the application successfully processed multiple .txt files of varying sizes and returned accurate word counts for each. The progress bars for individual files improved user engagement and provided visual feedback during file processing.

The application was tested on different browsers (Chrome, Firefox, Edge) and platforms (Windows, Android), and it worked consistently without requiring any installations or plugins.


File Word Counter
Choose Files
No file chosen
Process Files


File Word Counter
Choose Files
myfile.txt
Process Files



myfile.txt
15 words

5.2 User Experience and Performance:

User experience is smooth and intuitive. The use of progress bars gave a sense of activity, even though actual processing time was minimal. The asynchronous file handling ensured that no file blocked another, making the app appear multithreaded.

Overall, the application performed well in terms of:

- Responsiveness
- Accuracy
- User engagement
- Simulated concurrency

There were no noticeable lags or delays, even with 5–6 files processed together.

6. Conclusion and Future Scope:

6.1 Conclusion:

The project effectively demonstrates how web technologies can be used to solve simple but practical problems. It provides a convenient and efficient way to process text files and compute word counts, without relying on server-side logic or external tools.

By leveraging asynchronous features of JavaScript, the application simulates multithreaded behavior in a single-threaded environment, enhancing user experience while keeping the solution lightweight and accessible.

6.2 Future Scope:

This project has potential for several future enhancements, such as:

- Support for other file formats like .docx, .pdf, etc.
- Real-time charting or summary analysis (average word count, longest file, etc.)
- Saving results as downloadable reports (CSV, PDF)
- Integration of Web Workers for true multithreading
- Adding accessibility features and multi-language support

These features can transform the tool into a comprehensive online text analysis platform.

REFERENCES:

 Turcotte, A. (2021). *Optimizing Asynchronous JavaScript Applications*. Laurentian University.
https://reallytg.github.io/files/papers/alexi_thesis.pdf
Is thesis me asynchronous JavaScript applications ki inefficiencies ko analyze karke unhe optimize karne ke methods discuss kiye gaye hain.

 Sotiropoulos, T., & Livshits, B. (2019). *Static Analysis for Asynchronous JavaScript Programs*. arXiv preprint arXiv:1901.03575.
https://arxiv.org/abs/1901.03575
Ye paper asynchronous JavaScript programs ke static analysis ke techniques ko explore karta hai, jo tere project ke asynchronous behavior ko samajhne me madad karega.

 Microsoft Research. (2017). *Semantics of Asynchronous JavaScript*.
https://www.microsoft.com/en-us/research/wp-content/uploads/2017/08/asyncNodeSemantics.pdf
Is paper me Node.js ke asynchronous execution model ka formalization diya gaya hai, jo JavaScript ke event-driven nature ko samajhne me helpful hai.

 DigitalOcean. (2019). *How To Read and Process Files with the JavaScript FileReader API*.
https://www.digitalocean.com/community/tutorials/js-file-reader
Ye tutorial FileReader API ka practical implementation batata hai, jo tere project ke file processing module ke liye relevant hai.

 Sling Academy. (2023). *Enhance User File Interactions with the File API in JavaScript*.
https://www.slingacademy.com/article/enhance-user-file-interactions-with-the-file-api-in-javascript/
Is article me File API ke components jaise File, FileReader, aur Blob ke baare me detail me bataya gaya hai, jo client-side file interactions ke liye important hai.

Project Summary:

This project is a simple, user-friendly web application that allows users to upload one or more .txt files and instantly get a word count for each — all done right inside the browser. Built using just HTML, CSS, and JavaScript, the app doesn't need any server or installation. Everything runs on the user's side, making it fast, private, and super easy to use. Although JavaScript normally runs in a single thread, we've smartly used its asynchronous features to simulate multithreading.

This means each file is processed independently without freezing the screen, and a separate progress bar for every file gives a smooth, real-time feel of parallel processing. The design is kept clean and responsive, working well on both desktop and mobile.

Behind the scenes, it uses the FileReader API and non-blocking logic to handle multiple files at once without slowing down. It's been tested on various file sizes and performs really well, even with larger inputs. Because everything happens in the browser, users don't have to worry about privacy or internet speed.

Going forward, this project could be improved further by adding real multithreading with Web Workers, support for other file types like PDFs, export features, and more visual analytics. In short, this project shows how even simple web tech can be combined creatively to build something fast, efficient, and genuinely useful.