Optical Character Recognition (OCR) Using Tesseract.js

BY

PARVEJ MIAH
ID: 201-15-3770

FINAL YEAR DESIGN PROJECT REPORT

This Report Presented in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering

Supervised By

Dr. S.M Aminul Haque
Professor & Associate Head
Department of Computer Science and Engineering
Daffodil International University

Department of Computer Science and Engineering
Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY

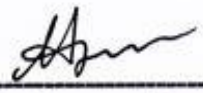
DHAKA, BANGLADESH

13 January 2025

APPROVAL

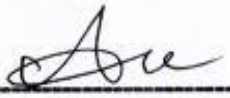
This Project titled "**Optical Character Recognition (OCR) Using Tesseract.js**", submitted by **Md Parvej Miah** and to the Department of Computer Science and Engineering, Daffodil International University, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 13-01-2025.

BOARD OF EXAMINERS

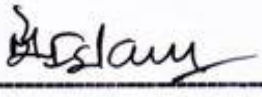


Ms. Nazmun Nessa Moon — **Chairman**
Associate Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Dewan Mamun Raza — **Internal Examiner**
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University



Mr. Abdullah Al Mamun — **Internal Examiner**
Lecturer
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University



Dr. Md. Manowarul Islam — **External Examiner**
Associate Professor
Department of Computer Science and Engineering
Jagannath University

DECLARATION

We hereby declare that this project has been done by us under the supervision of **Dr. S.M Aminul Haque, Professor & Associate Head,** Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:

Dr.S.M Aminul Haque
Professor & Associate Head
Department of CSE
Daffodil International University

Co-Supervised by:

Name
Designation
Department of CSE
Daffodil International University

Submitted by:

Parvej

Md. Parvej Miah
ID: 201-15-3770
Department of CSE
Daffodil International University

ACKNOWLEDGEMENT

First, we express our heartiest thanks and gratefulness to almighty for His divine blessing making it possible for us to complete the final year project/internship successfully.

We are grateful and wish our profound indebtedness to **Dr. S.M Aminul Haque**, **Professor and Associate Head**, Department of CSE Daffodil International University, Dhaka. Deep Knowledge & keen interest of our supervisor in the field of "Optical Character Recognition (OCR) Using Tesseract.js" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

We would like to express our heartiest gratitude to the **Head of the Department of CSE**, for his kind help in finishing our project and also to other faculty members and the staff of the Department of CSE, Daffodil International University.

We would like to thank our entire course mate in Daffodil International University, who took part in this discussion while completing the course work.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

ABSTRACT

This project focuses on developing a web-based optical character recognition (OCR) system using Tesseract.js, a JavaScript library that allows extracting text from images in both client-side and server-side environments. Image-based text conversion to be editable and searchable. Digital text efficiently and accurately happens. The system uses Tesseract.js which leverages the power of the Tesseract OCR engine to support multiple languages. Processes various image formats (JPEG, PNG, TIFF) and improves OCR accuracy on high-resolution images. This is especially true for printed text. By using pre-processing techniques such as grayscale conversion, Tesseract.js allows for easy integration with web platforms. It allows users to upload images and receive messages instantly. Although it is effective with sharp images But it faces challenges with hand-drawn or low-quality images. Future improvements include improved pre-processing. Hand writing recognition and multi-language support for wider applications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE **PAGE**

CHAPTER 1

Introduction

1.1 Introduction

The touch of modern technology has a huge impact on our world. And artificial intelligence is one of them. With artificial intelligence we can make many things much easier. Optical character recognition (OCR) technology converts text-rich images into an editable, searchable format. OCR can be used to convert scanned documents, photos of documents or other digital image to text. Complete with document management applications Digitization and automatic data entry this project uses Tesseract.js, a JavaScript-based OCR library that wraps around the Tesseract OCR engine, which OCRs directly. The results, which you can achieve in the browser or on the server with Node.js, demonstrate the high accuracy of the system. In the picture the printed text is clear and mediocre performance with low quality text or handwritten text. This OCR application is designed to be easy to use on web platforms. It allows users to upload images and instantly access editable text output. Future work includes improving pre-processing techniques. Explore deep learning integration for better handwriting recognition and expanding support to additional languages. The project includes a key step in image pre-processing. Text recognition and post processing Pre-processing techniques such as gray scale conversion and there scolding are used to improve the accuracy of OCR. The Tesseract.js library processes these images to recognize text. It has been optimized to improve readability and formatting. This project demonstrates the power of Tesseract.js in handling high-resolution text images.

1.2 Motivation

The motivation behind this project is to simplify the process of extracting text from images. This is time-consuming and error-prone if done manually. With the increasing digitization of documents the demand for powerful and easy-to-use OCR tools is increasing. In addition to bringing Web functionality make it accessible without special software. And provides a fast and easy-to-use solution for both personal and commercial

applications. Immigration requires the passport to be converted from image to text. Any language passport can be easily read through OCR. This project aims to explore the capabilities and limitations of Tesseract in handling different types of images.

1.3 Objectives

The main goals of this Project are:

- Users should be able to upload images and delete text in the images.
- It uses image pre-processing techniques such as gray scale conversion and thresholding. To improve the accuracy of texts recognition
- To use the Tesseract OCR tool to automatically convert various images formats (such as PNG, JPEG, and TIFF) to text.
- To evaluate the performance and accuracy of the OCR system on different types of images, such as typed handwritten text.
- To evaluate the performance and accuracy of Tesseract.js OCR on a web-based platform.
- Allow users to upload images (such as scanned documents, receipts, and photos) and automatically extract and display the text.

All of these goals are united into one solid thing. This project uses Tesseract.js to create a web-based OCR tool that converts text images into editable digital text. Improved text recognition accuracy using pre-processing techniques and provides users with an easy and convenient solution to instantly upload images and extract text. The project also evaluates Tesseract.js' functionality on different types of images. The goal is to increase OCR accuracy and usability for a wider audience.

1.4 Expected Outcomes

The expected outcomes for "Optical Character Recognition (OCR) Using Tesseract.js in Java Script" include:

- ❖ Accurate text extraction: Reliable OCR results with high accuracy. This is especially true for high resolution images and standard fonts.
- ❖ User-friendly interface: A simple web-based tool that allows users to upload images and instantly access editable text output.
- ❖ Performance insights: Understand the capabilities and limitations of Tesseract.js with different properties and image types (such as typed text and handwritten text).
- ❖ Expanded accessibility: Easy-to-use, browser-based OCR solution with no need for special software.
- ❖ Advanced Text Processing Techniques: Explore pre-processing methods to increase OCR accuracy and processing speed for future applications.

All of these anticipated results work to completely Optical Character Recognition by offering effectiveness, customization, security, and flexibility to produce a smooth image to text experience for all parties.

1.5 Research Question

During our investigation we came across several questions. Among the questions ware:

- ❖ How accurate is Tesseract.js in recognizing text from clear images?
- ❖ Does Tesseract.js work better with printed text or handwritten text?
- ❖ How does Tesseract.js handle images with different text fonts?
- ❖ What image formats (JPEG, PNG, etc.) work best with Tesseract.js?
- ❖ How does resizing an image affect the OCR accuracy in Tesseract.js?
- ❖ Can Tesseract.js recognize text in multiple languages, and how accurate is it?
- ❖ What simple image processing techniques can improve OCR results?
- ❖ How fast is Tesseract.js in processing large images for text extraction?

1.6 Report Layout

To develop the perfect report layout for "Optical Character Recognition (OCR) Using Tesseract.js in JavaScript" The report layout is organized into multiple sections to highlight project-specific information. Development steps, characteristics and results of the project will see below:

Chapter 1: Introduction

I will discuss Introduction Motivation Objective and Expected Outcomes in chapter.

Chapter 2: Background

In this chapter i will discuss related work comparative analysis Scope of the Problem and Challenge with other project which will help me in project analysis.

Chapter 3: Requirement Specification

In this chapter, I have done some work. And because I collect datasets and work on projects through some software. I will discuss use case models and data collection methods.

Chapter 4: Design Specification

In this chapter, I will talk about front-end design and back-end design. as well as the input process and output process. This will help me a lot in analyzing the project and make the work easier.

Chapter 5: Implementation and Testing

In this chapter, I will discuss some of the human reseaech that i have implemented results analysis.

Chapter 6: Summary and Conclusion

In this chapter, I will discuss the summary and conclusion of the entire project.

Figure 1.6.1: Report Layout of this Project.

CHAPTER 2

Background

2.1 Introduction

In this chapter, I will give a brief description of the apps. I have been working on it and I will show how OCR works and how it differs from other projects. Where my project is better than theirs is basically OCR converting images from text into different languages. And based on the Tessract engine, we need teaching images for various inspections. This engine will perform very accurate searches that match our needs. But when the computer interprets it, certain algorithms must be set up and how the image-to-text conversion works. I often work with websites that already do this. I think I'll try something new. It looks at some of the apps that have worked for humans in the past to convert images to text. And compare my work with those apps. So that I can understand how I can convert all types of images to text and handwritten text.

2.2 Related work

I compared it with some external apps to test its feasibility in my project. Below are the errors I encountered and the names of some of the apps.

These tools have similar functionality. But there are differences in features and platforms.

1. ABBYY FineReader

2. Adobe Acrobat Pro OCR

3. OCR.spaces

4. OCR online

5. Prism

6. Message: Angel

7. Scanner Camera

2.3 Comparative Analysis

A comparative analysis for "Optical Character Recognition (OCR) Using Tesseract.js" involves assessing similar existing platforms or systems in the market. Here's a breakdown for a comparative analysis:

- Platform Features and Functionalities:
  - Text recognition from images in over 100 languages.
  - Client-side processing for privacy and offline use
  - Multi-language support Image pre-processing and various output formats (text, HOCR, JSON).
  - Customizable settings for better OCR accuracy.
- User Interface (UI) and Experience (UX):

  Tesseract.js is a JavaScript library that does not have a built-in UI, but allows developers to create custom interfaces to upload images. Show OCR progress and display results. It mainly focuses on extracting text. This limits the management of complex document layouts. Affects the user experience
- Security and Privacy Measures:

  Tesseract.js processes data on the client side without sending it to the cloud. Thus ensuring privacy and reducing security concerns. It does not store any processed data.
- Personalization and Recommendation Systems:

  Tesseract.js has no built-in personalization or suggestion features. But developers can customize the language model and provide personalized suggestions for integration into the larger system.
- Customer Support and Engagement:
  - Community-based support via GitHub, forums, and documentation.
  - Because it is open source Therefore, there is no official customer support.
- Market Position and User Reviews:

  Tesseract.js is a popular open source OCR tool with good reviews for being cost-effective and privacy-focused. However, some users report performance issues and accuracy limits.

This is especially true for handwriting and complex document layouts ideal for developers looking for a free client-side OCR solution. But they may not match commercial OCR systems in terms of features and accuracy.

2.4 Scope of the Problem

The problem focuses on the challenges of extracting text from images, such as the inefficiency and errors of manual transcription, accessibility barriers for visually impaired users, and the need for digitizing documents. It includes handling complex layouts, multi-language text, real-time recognition, and ensuring data privacy. The demand for affordable, accurate, and efficient OCR solutions like Tesseract.js is growing across industries for automation, accessibility, and digitization tasks. Text extraction from images is essential for digitizing documents, automating workflows, and enhancing accessibility. Manual transcription is slow and error-prone, creating the need for automated OCR tools.

2.5 Challenges

All jobs in this world are challenging. But optical character recognition using artificial intelligence is one of them. OCR using Tesseract.js faces challenges such as degraded accuracy with low quality or blurred images. This Project is a web-based project and used on a phone. I would usually work with a website that already doses this. Problems managing complex layouts, tables, or handwritten text multilingual support may be limited. And processing large or high-resolution images can slow down the system. Pre-processing is often required to improve results. And accuracy may vary based on text size, font, and alignment. Browser processing can be resource intensive, and OCR output may require manual error correction. Additionally, security concerns Security and privacy should be addressed when dealing with sensitive data. The challenges faced in OCR using Tesseract.js are significant. But it can be handled with some pre-processing. Optimization and appropriate error handling although it provides a powerful open source and client-side solution. Users need to consider factors such as image quality and text complexity and system processing when using Tesseract.js in real-world applications. Additional processing tools are needed to handle these challenges. Specific… Improve training data for use cases. To increase the efficiency of the system to meet specific and necessary needs.

CHAPTER 3

Requirement Specification

3.1 Introduction

The requirements specification for OCR using Tesseract.js aims to create a system that is efficient, scalable, and easy to use. This converts the image into machine-readable text. It summarizes the main functions of the OCR engine, such as extracting text from images. Support for different languages and document formats Compatibility with various devices. Specifications also include performance. Scalability, security, and usability requirements to ensure accurate and reliable OCR the result while maintaining data privacy and access to the .js framework for developing a working OCR system. The goal was to create a powerful, user-friendly solution that handles complex document layouts. Supports a variety of image formats and separate typed and handwritten text The system should work smoothly on different devices, optimizing the use of resources. and ensure data security and usability. This document details functional and non-functional requirements to guide development.

3.2 Implementation Requirement

Below is the software i used to complete my project.

- Software
  - Tesseract.js library (latest version).
  - Web technologies: HTML5, CSS3, JavaScript (for frontend), Node.js (optional for backend).
- Hardware
  - Device with a modern processor (Intel i3 or equivalent).
  - Minimum 4GB RAM for smooth performance.
- Input Image
  - Support recognition for common image formats (JPEG, PNG, GIF).
  - Webcam integration for live image capture.
- Performance
  - Efficient OCR processing for images up to 1MB.

- Real-time for live images (optional).

❖ UX/UI

- User-friendly interface for uploading images, viewing results, and saving text output.

❖ Security

- Local image processing for privacy.
- Encryption for sensitive text data when saved or transmitted.

❖ Compatibility

- Browser compatibility (Chrome, Firefox, Safari).
- Mobile and desktop support with responsive design.

3.3 Process Modeling

The OCR process using Tesseract.js starts with the user uploading or capturing a photo in a supported format. Once the image is prepared Alternative pre-processing techniques such as gray scale image conversion, scaling, and noise reduction are applied to improve quality for better text recognition. After pre-processing, Tesseract.js analyzes the image and uses optical character recognition (OCR) algorithms to extract text. The extracted text is cleaned for accuracy and displayed to the user, which can save it as a text file if the OCR process fails due to poor image quality. The system will provide feedback to the user. It suggests ways to improve images. Additionally, all processing takes place within the user's browser to maintain privacy and security. This process ensures accurate text extraction. User friendly interaction and protection of sensitive information.

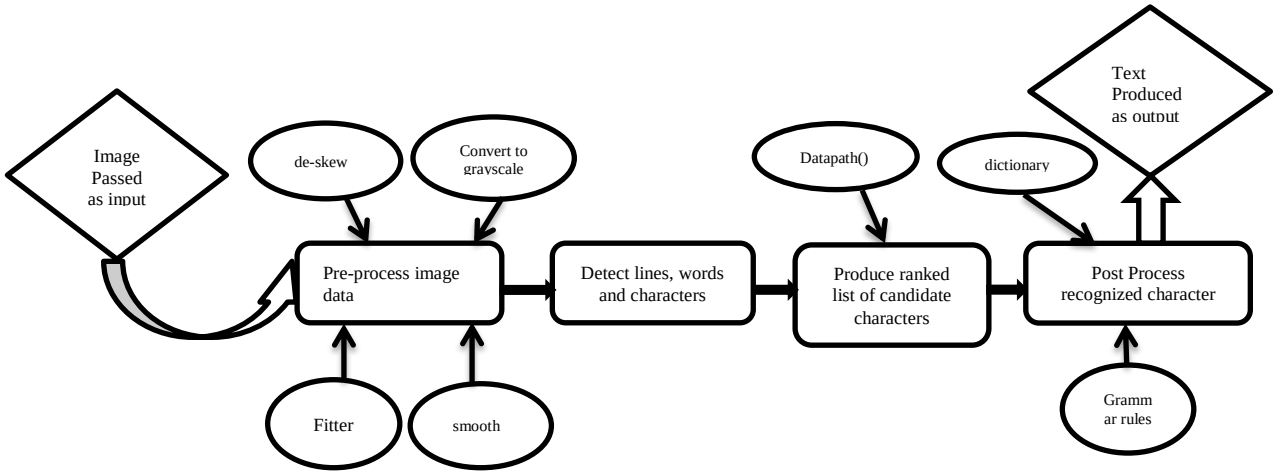


Figure 3.3.1 OCR Process Model

3.4 Use Case Modeling and Description

Key use cases include image uploading and processing. Pre-processing for better accuracy displaying and saving extracted text Error handling and ensuring privacy during processing. These interactions ensure a smooth and secure user experience. Emphasis is placed on convenience and reliability in extracting text. Use case modeling helps users identify different methods. To interact with the OCR system, each use case describes a specific functionality of the system from the user's perspective. These use cases focus on delivering an efficient and user-friendly OCR system. At the same time, privacy and ease of use are guaranteed.

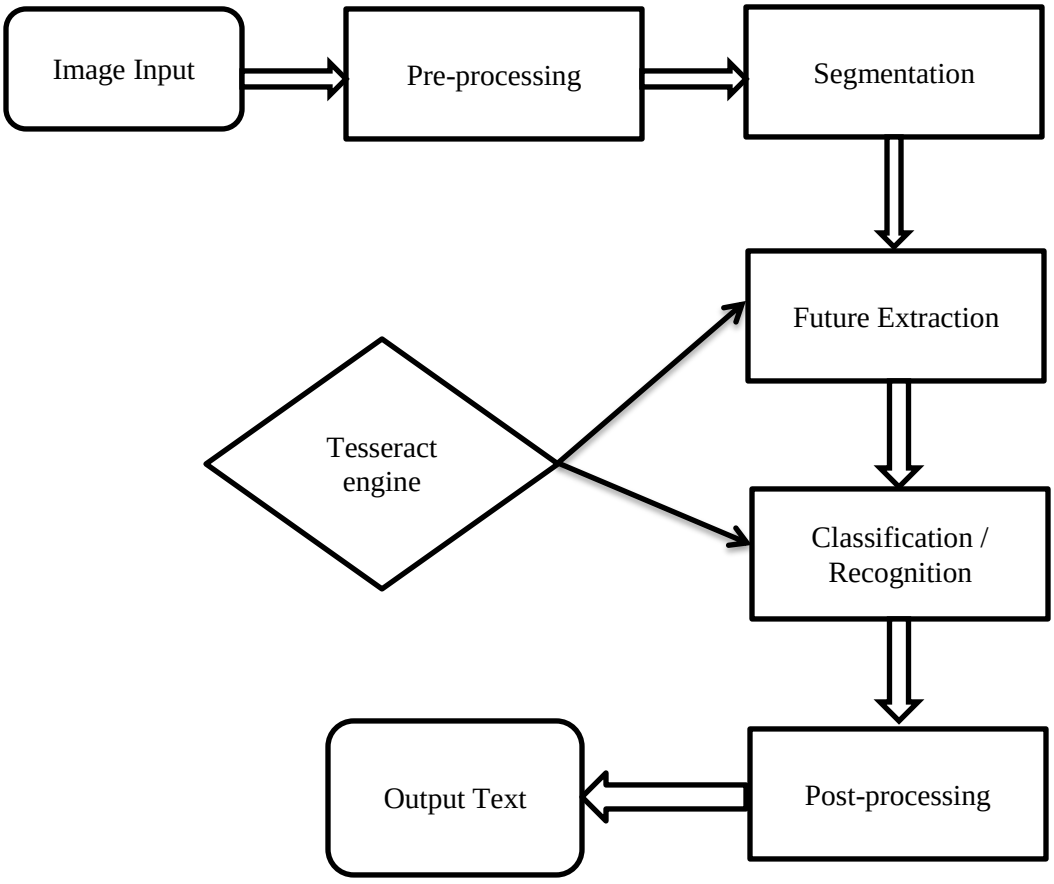


Figure 3.4.1 Use Case

3.5 Block Diagram and Description

A diagram of a typical OCR system is shown in Figure 3.5.1. The main steps of OCR include image acquisition, pre-processing, segmentation, feature extraction, classification, and after processing.

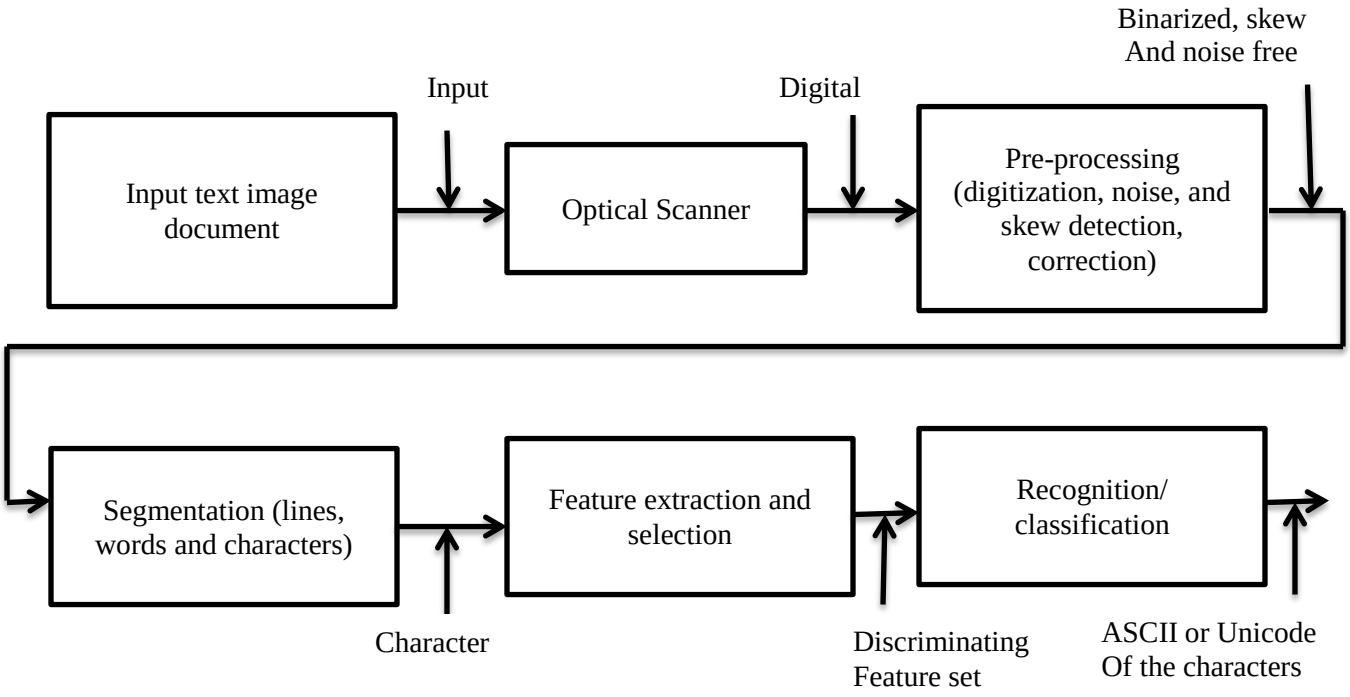


Figure: 3.5.1 Block Diagram

3.6 Logical Data Model

This OCR system's logical data model summarizes the relationships and data flow between the key entities involved. The main components are:

- Image Data: Raw data upload by users. They will be temporarily stored for processing.
- OCR Engine Configuration: Setting such as language, pre-processing options. Recording parameters.
- Process text: Final output you can store it in database or download it as a file.
- User Interaction Log: Optional data that records user actions. And process the result for performance analysis.

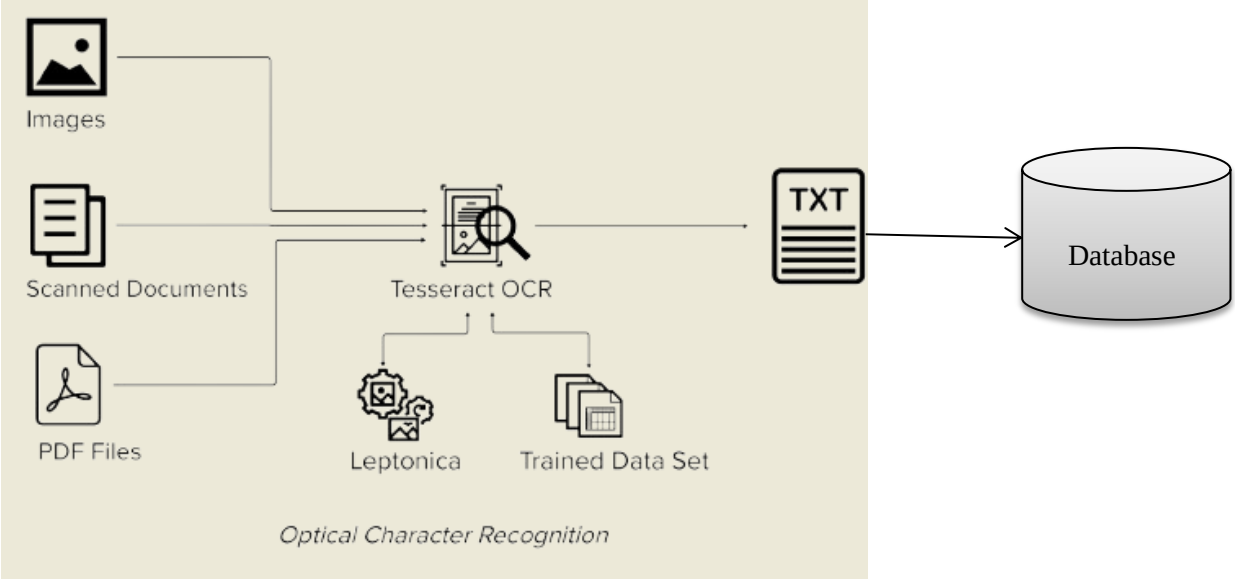


Figure 3.5.1 Logical Data Model

3.7 Design Requirement

This ensures a robust, easy-to-use, and scalable OCR application.

- User Interface Design: Easy photo upload options (Drag and drop or file selector) provides clear progress indicators processing. Displays extracted and editable text with options to download or copy. Allows pre-processing (grayscale, binary, etc.) to be optimized.
- Data Handling: Supports common file formats (PNG, JPEG, BMP). Check image size and resolution. Uses Tesseract.js for OCR with pre-processing to improve accuracy. The result will be plain text or a structured format (such as JSON).
- System Architecture: Client-side processing with Tesseract.js for privacy option server- side module (Node.js) for advanced features. Module workflow: Upload →Pre-processing →OCR →Output.
- Performance and Scalability: OCR completes in 5 seconds for 2MB images. Error handling for unsupported files or OCR failures. Scalable design for batch processing and multiple language
- Capability and Accessibility: Responsive design for all device (Desktop/Mobile) cross-browser support (Chrome, Opera, firefox, safari) accessibility features such as screen reader compatibility.

CHAPTER 4
Implementation and Design Specification

4.1 Front-end Design

Here is an intuitive structure of a User Interface (UI). When the user uses my application, he will provide an image as input and the image will be saved then he will get the option to select the language and he will select the language in which he uploaded the image and will see a button called **DO OCR** Pressing this button will get the full output. To guarantee a positive and effective user experience, the design gives a high priority to accessibility, usability and aesthetics.

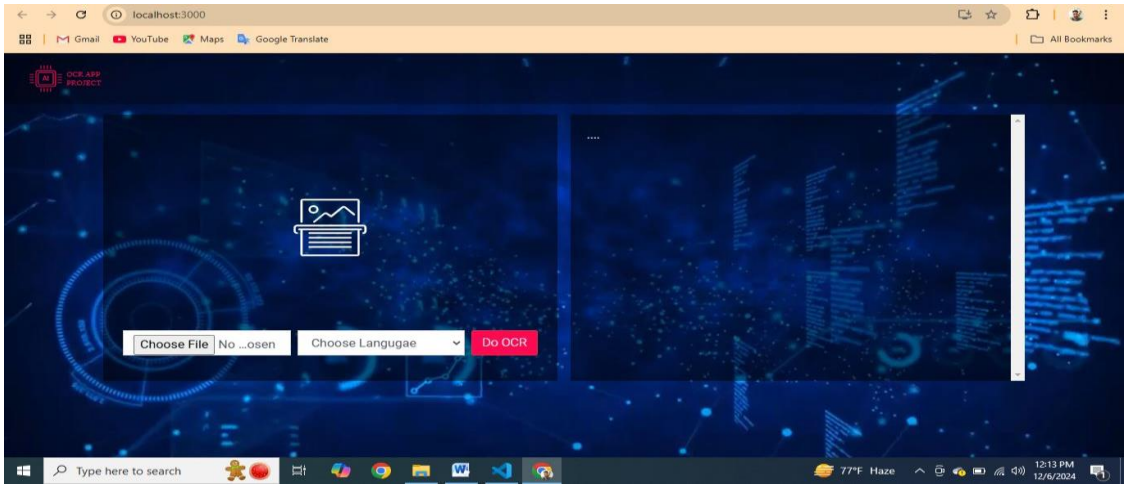


Figure 4.1.1: Home Page

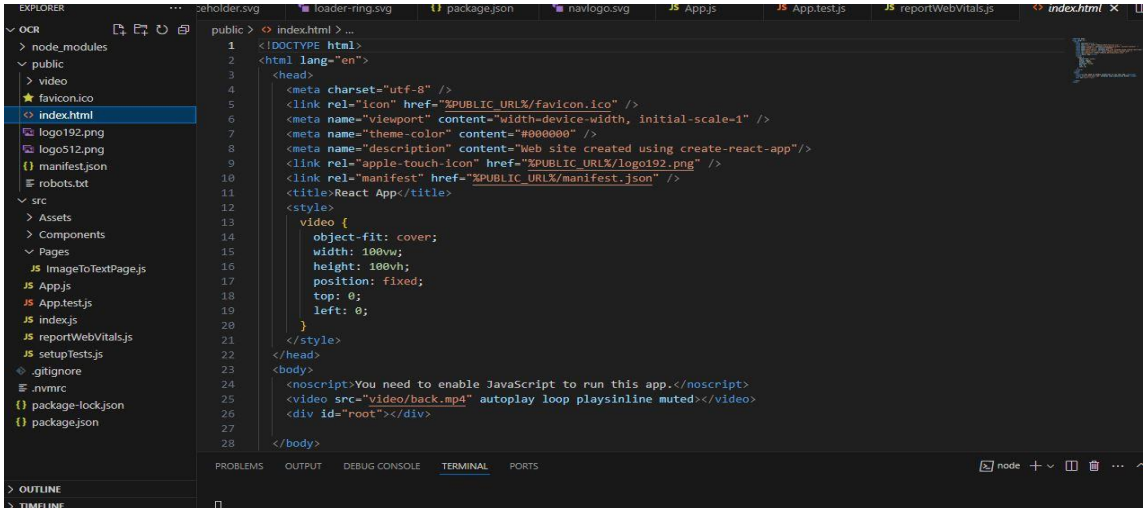


Figure 4.2.1: Index.html Model

4.2 Back-end Design

The back-end server enhances the OCR system by handling advanced functions such as batch processing. Advanced image processing and extracted text archiving Servers running Node.js and Express.js provide API endpoints for uploading images. Process OCR operations and extract results. The Mongo DB database can store images and extract text. While cloud services like AWS or Google Cloud Vision can improve accuracy by integrating scalability. Before starting an Optical Character Recognition project, we need to think about the image-to-text conversion design. Image to text is usually done using React JS and Tesseract.js. Its performance is very successful. Converting images to text requires a good quality OCR Engine. This version can work well.

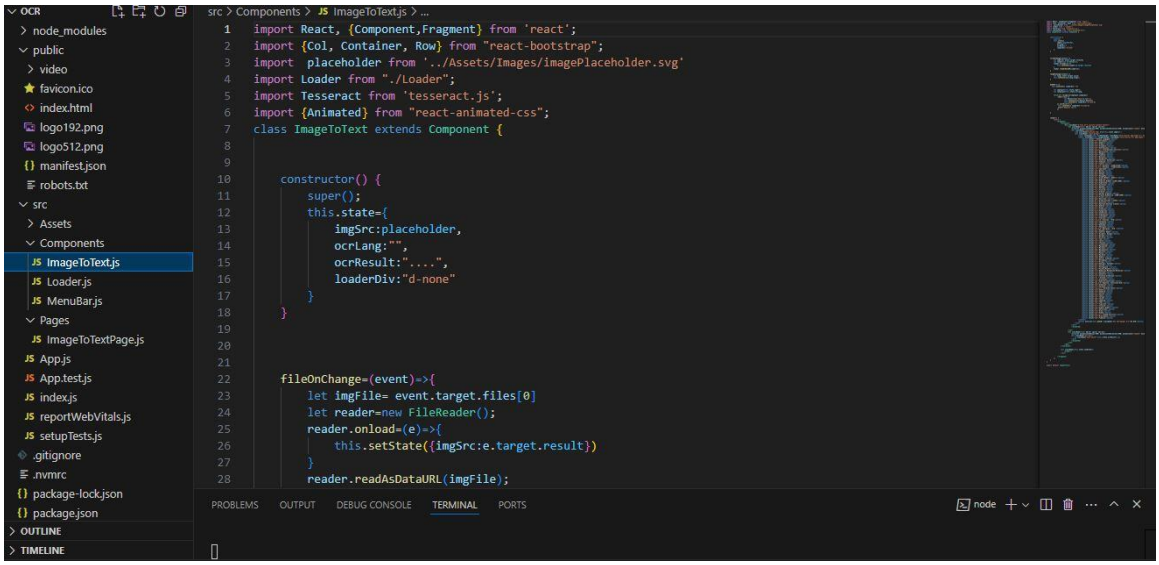


Figure 4.2..3: Image to Text Model 1

Figure 4.2.4: Image to Text Model 2

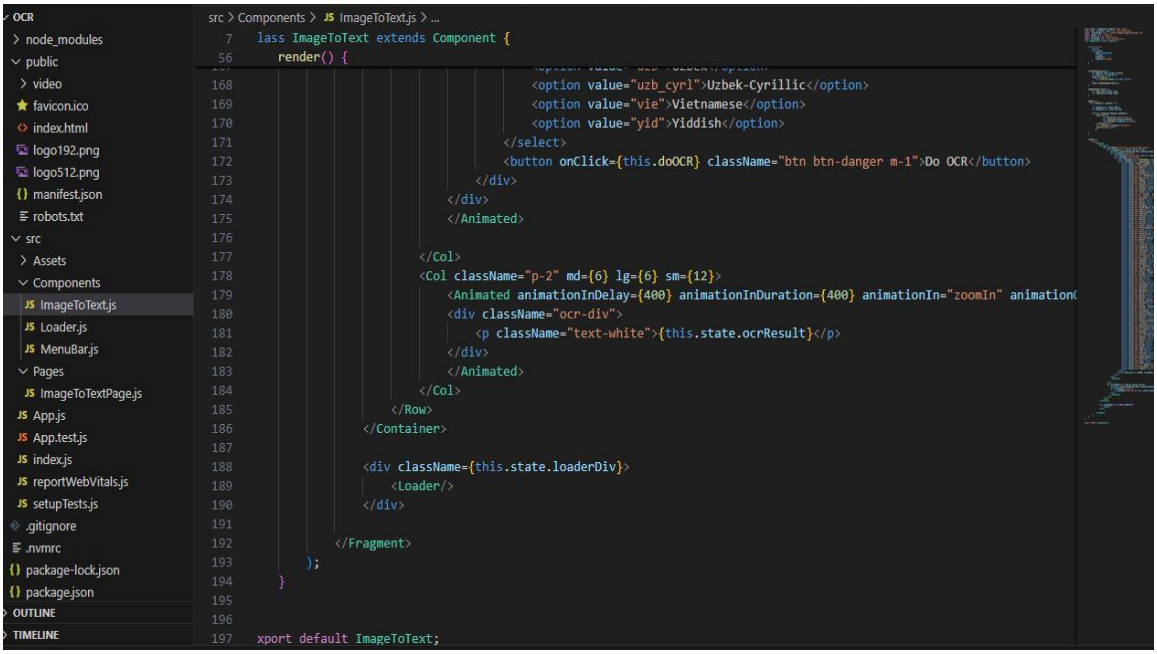


Figure 4.2.5: Image to Text Model 3

Figure 4.2.6: Loader Model

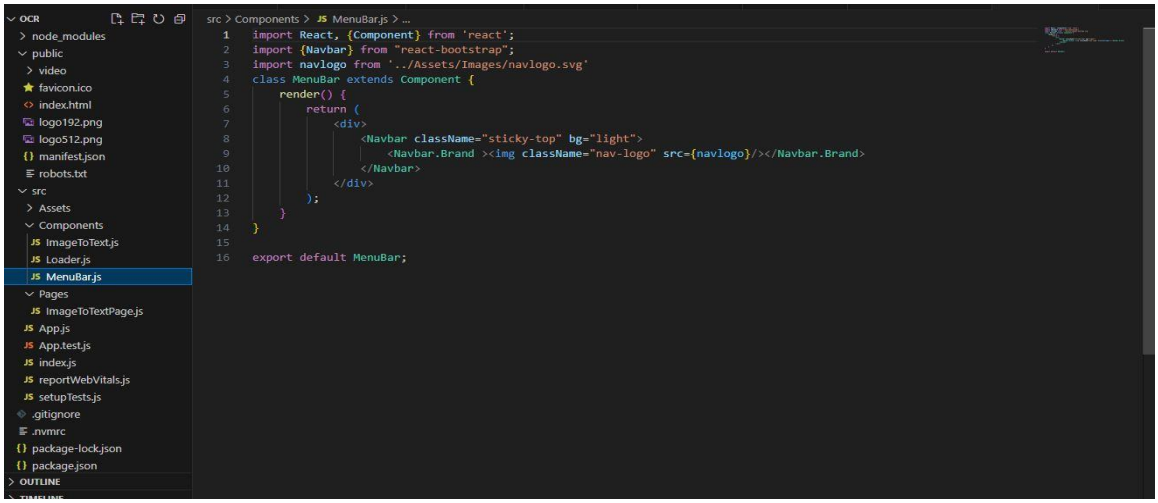


Figure 4.2.7: Menu Bar Model

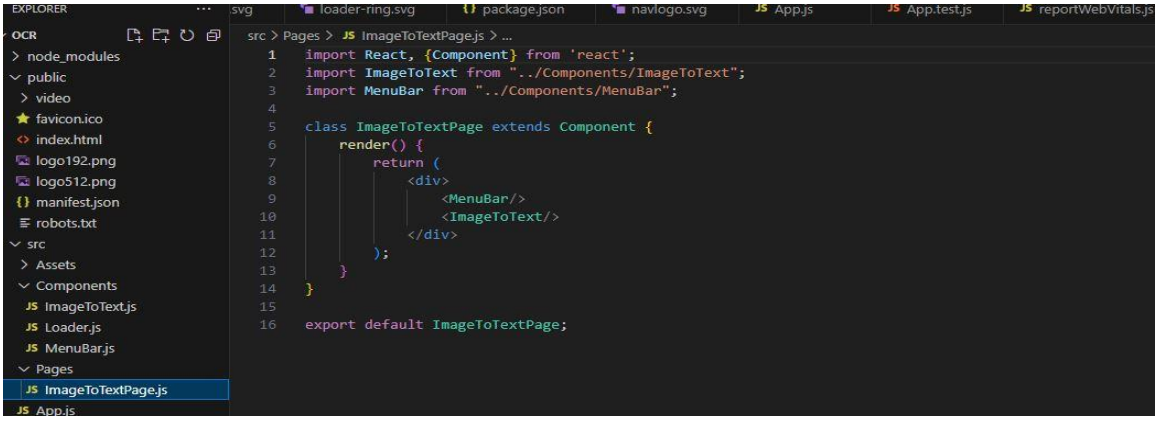


Figure 4.2.8: Image to Text Page Model

Figure 4.2.9: App Model

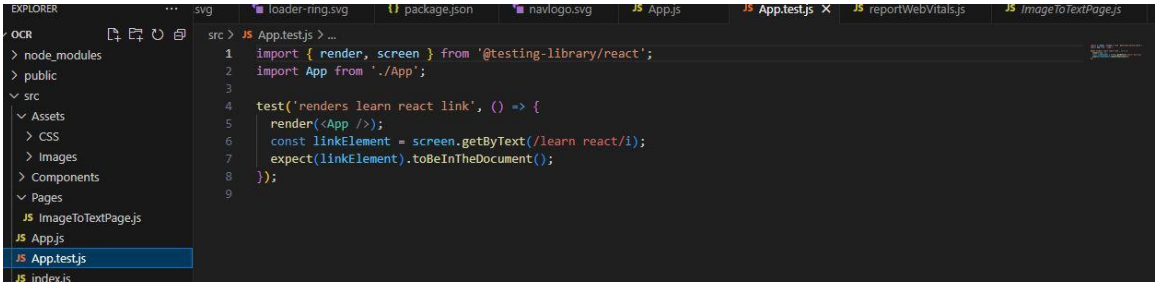


Figure 4.2.10: App Test Model

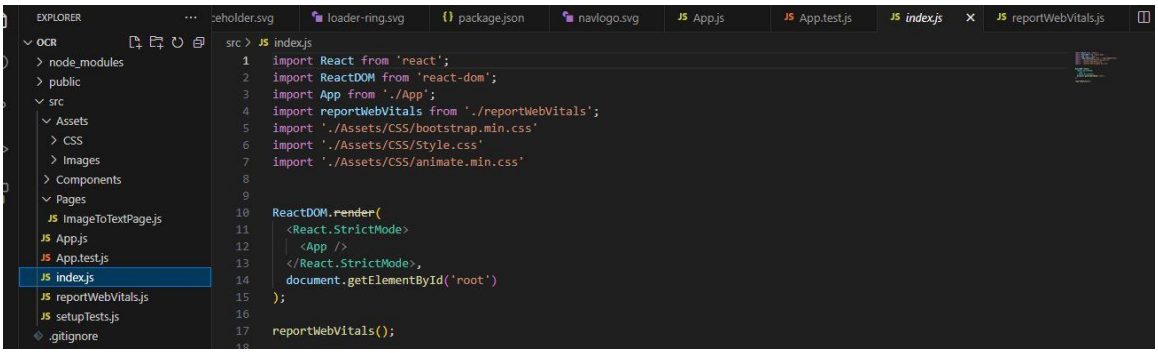


Figure 4.2.11: Index Model

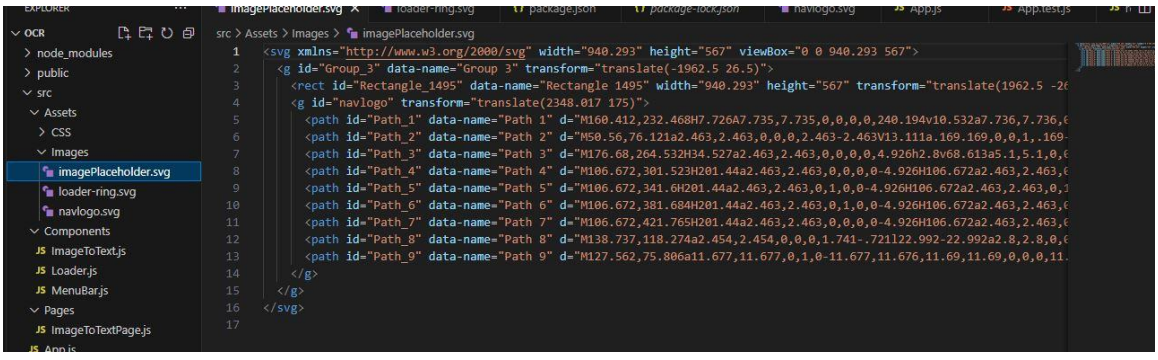


Figure 4.2.12: Image Place Holder Model

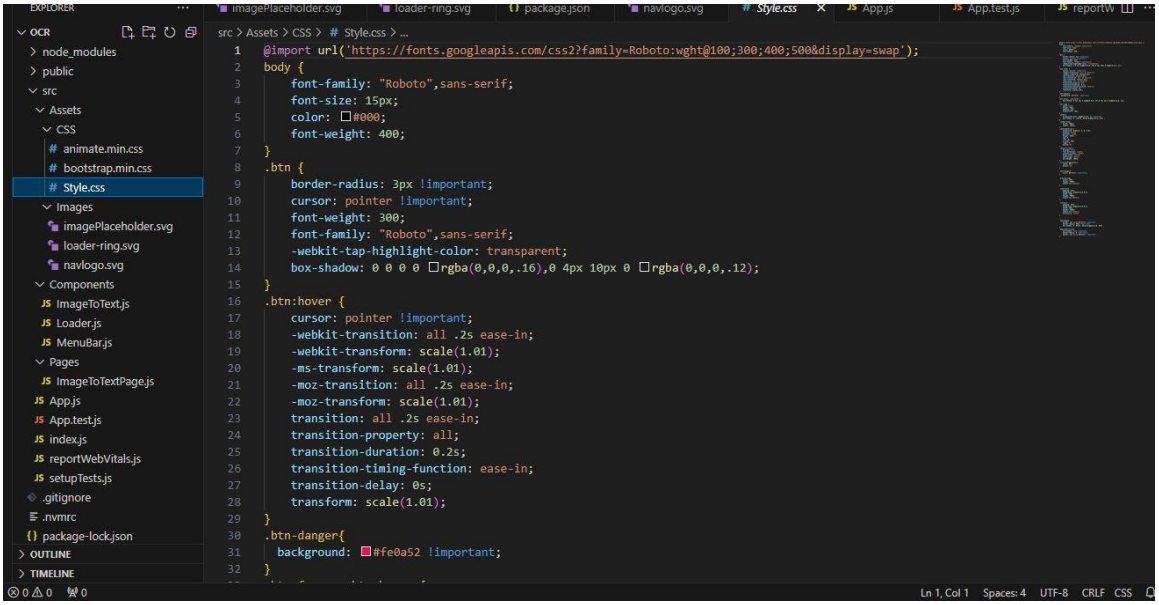


Figure 4.2.13: Animate Wallpaper Style Model

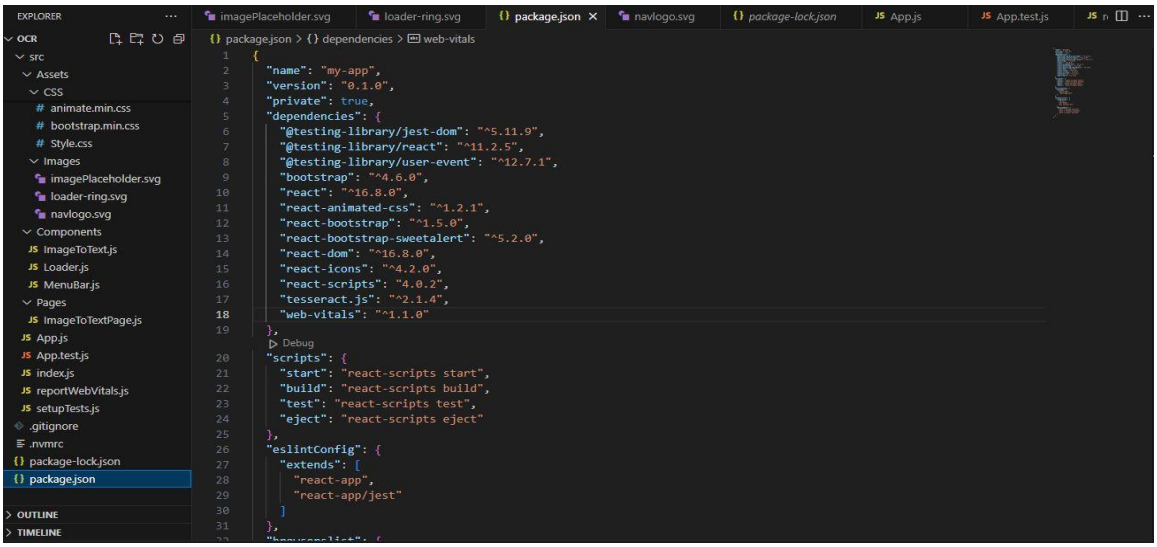


Figure 4.2.14: Package Jason Model

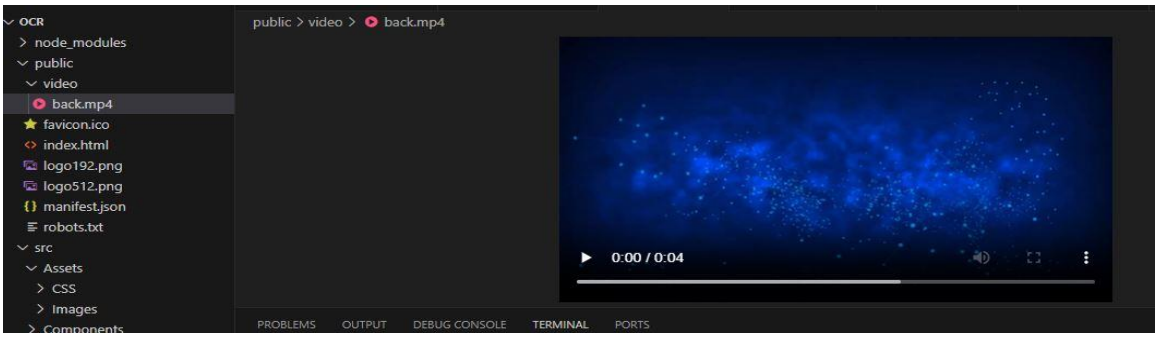


Figure 4.2.15: Background Video Model

4.3 Setup

Installation: Tesseract.js can be included via a CDN or installed using npm.

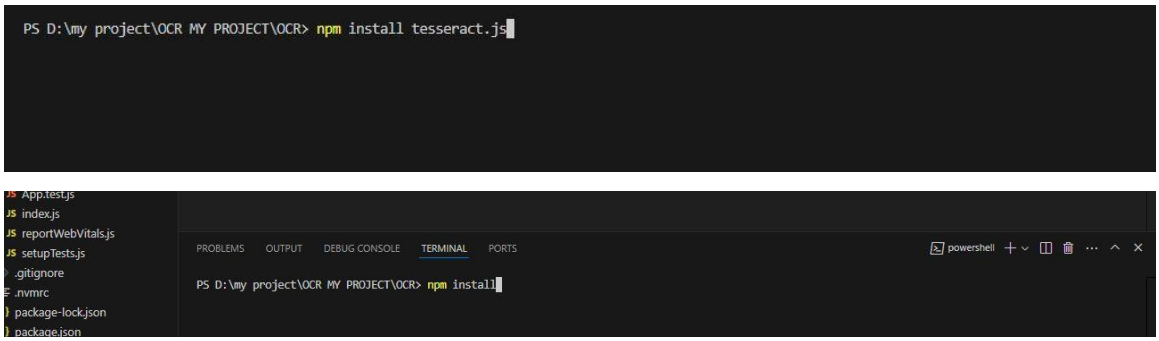


Figure 4.3.1: Node.js npm install

Importing the Library:

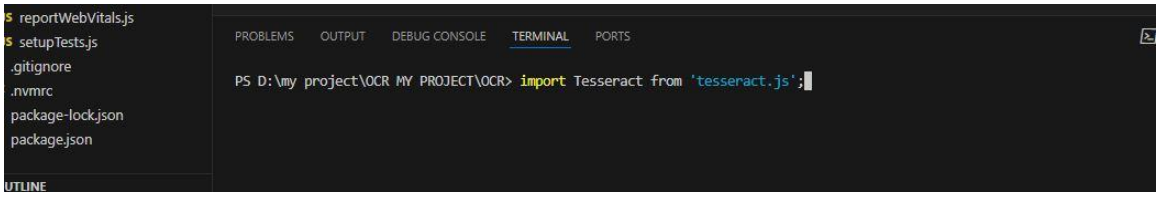


Figure 4.3.2: Import Library

Project Run: Finally we start the development server

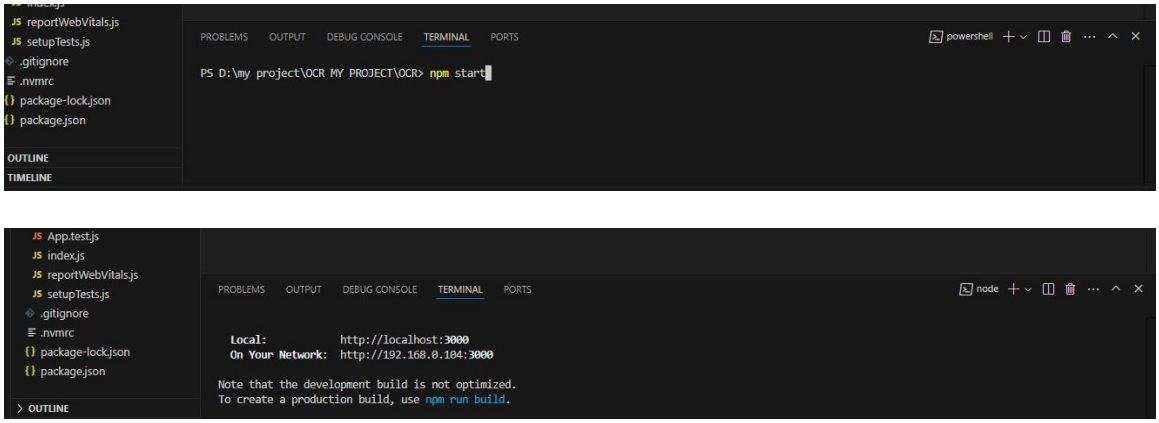


Figure 4.3.3: Development Server Run

The development server is available in your favorite browser at http://localhost:3000/browser/basic-efficient.html when you change the files in the src folder, it creates tesseract.min.js and worker.min.js. Automatically new.

4.4 Input Process

The image process begins with the user uploading an image through the file entry field in the web application. The images is entered through choose file and it is displayed. Then you have to select the language through choose language. These images are reviewed to ensure they meet format and size requirements. When the rights are verified Uploaded images are displayed in the user interface using an <img> or <canvas> element, which provides a visual reference for verification. Pre-processing steps such as scaling, grayscale conversion or setting criteria It is used to enhance the quality of images and adapt them to OCR processing. Often we can download and save captured images. This can be reused later.

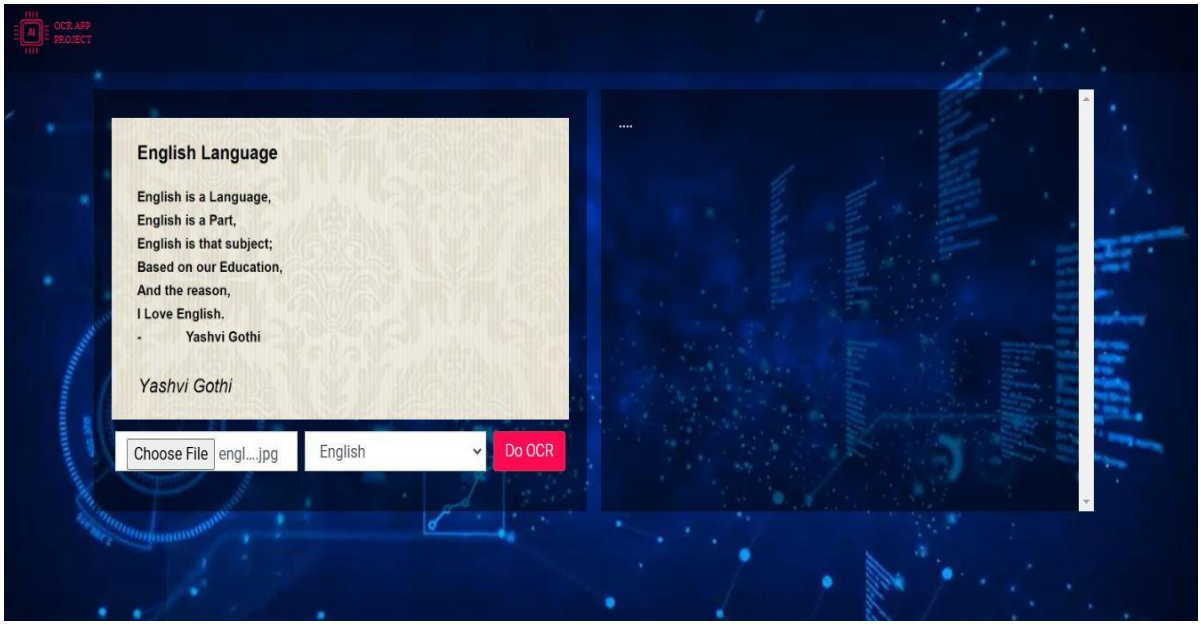


Figure 4.4.1: Input Process

4.5 Output Process

When we input our image into the computer through choose file, it appears on the display and then the language is detected. Then the image is analyzed by the OCR engine. The output process begins with the OCR engine extracting text from the processed image. This extracted text is then displayed to the user on the application interface, typically in a text area or a designated output field. Users can copy or download the extracted text for further use. For advanced functionality, the extracted data can also be stored in a database or exported in formats such as plain text, JSON, or CSV for integration with other systems.

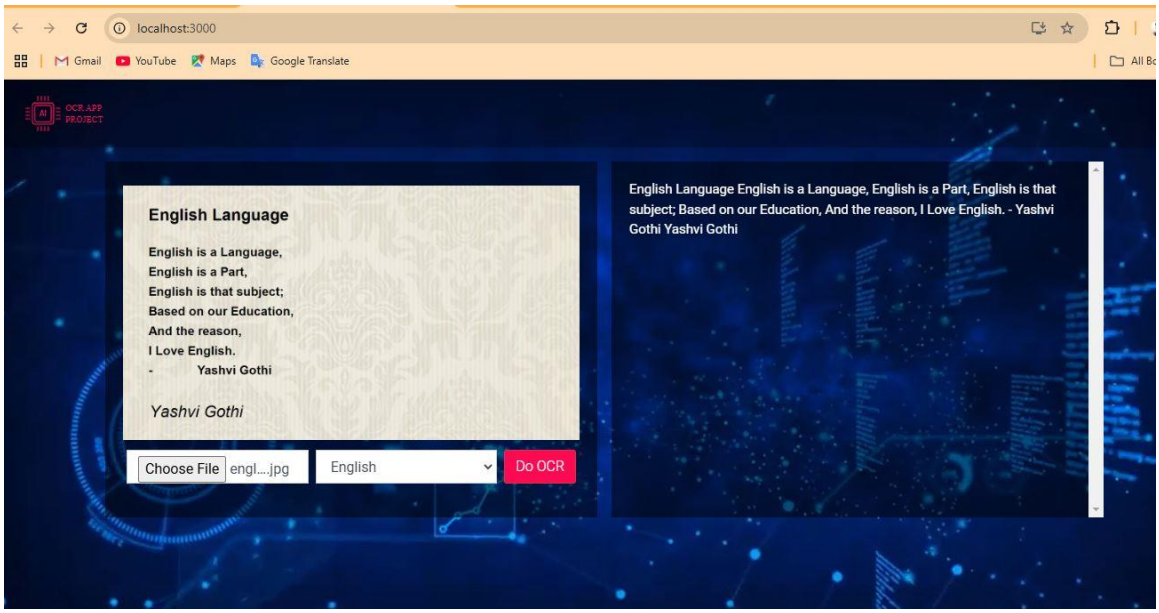


Figure 4.5.1: Output Process

More Languages Input & Output Process:

Bangla Language Process: First, the user uploads a Bengali language image as input via Choose File. The image will be displayed on the display and then the Bengali language will be selected. Then, the OCR engine checks these characters for format, size, and quality to ensure compatibility and uses special preprocessing techniques to handle complex character details such as conjunctions. For Bengali OCR, the text is processed to ensure correct representation of Bengali script, including complex characters and

punctuation marks. Furthermore, the text can be stored in a database or exported in formats such as plain text, JSON, or CSV. This way, the complete output is obtained.

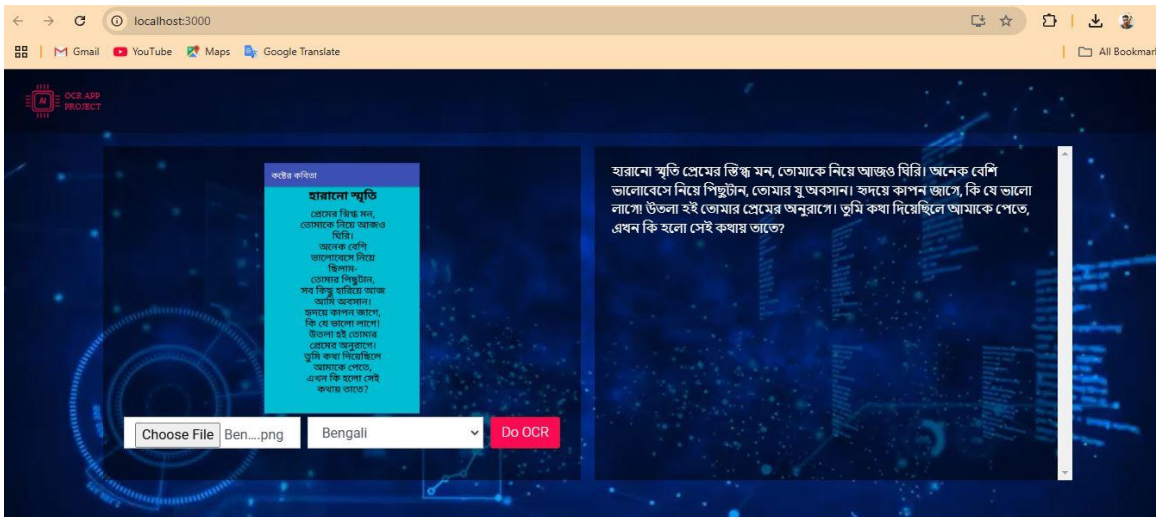


Figure 4.5.2: Bengali Language Input-Output Process

Hindi Language Process: For Hindi language OCR, the user uploads a Hindi language image as input through Choose File. The image will be displayed on the display and then Hindi language will be selected. Then, the OCR engine checks these characters for format, size and quality to ensure compatibility and uses special preprocessing techniques to handle devanagari script nuances such as ligatures, complex character details such as additions, binarization, resizing. For Hindi OCR, the text is processed to ensure correct representation of Bengali script including complex characters and proper punctuation. Hindi text interface is displayed, furthermore, the text can be saved to a database or exported in formats such as plain text, JSON or CSV.

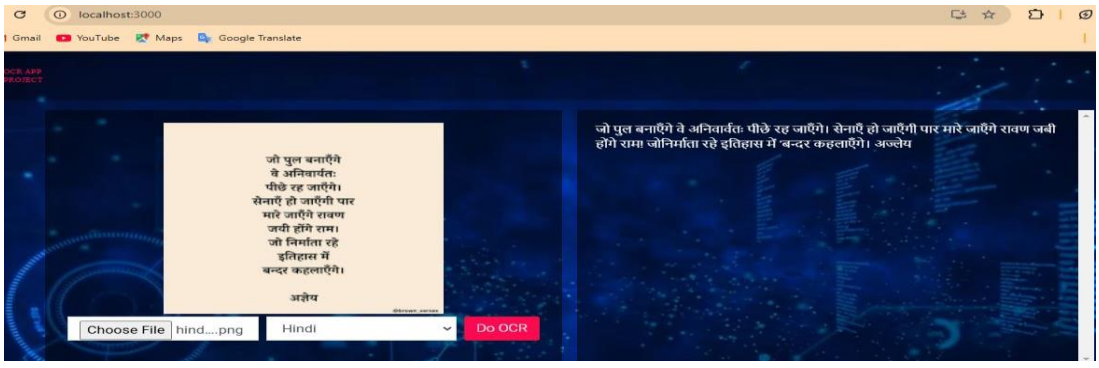


Figure 4.5.3: Hindi Language Input-Output Process

Arabic Language Process: For Arabic language OCR, the user uploads an Arabic language image as input through the Select File option. The image will be displayed on the display and then the Arabic language will be selected. Then, the OCR engine checks these characters for format, size and quality to ensure compatibility and uses special preprocessing techniques to detect subtleties of quran script such as ligatures, complex character details such as addition, binarization, resizing. For Arabic OCR, the text is processed to ensure correct representation of Arabic script including complex characters and proper punctuation. The output response of OCR, the extracted text reflects the correct orientation and alignment of Arabic script including right to left text flow management. The Arabic text interface is displayed, in addition the text can be saved to a database or exported in formats such as plain text, JSON or CSV. In this way, the complete output is obtained through OCR.

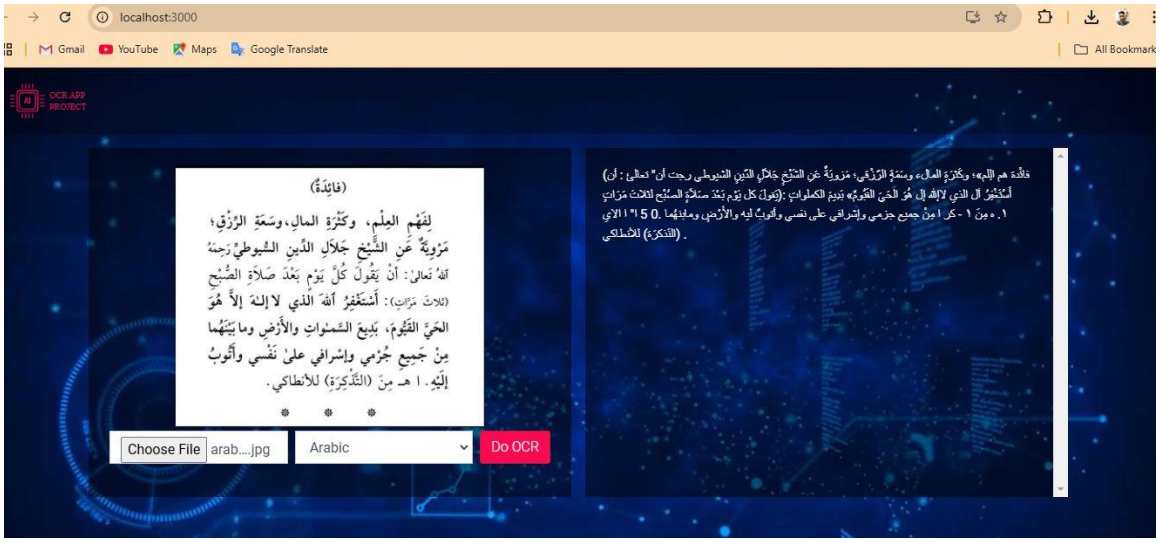


Figure 4.5.4: Arabic Language Input-Output Process

French Language Process: For French language OCR, the user uploads a Farsi language image as input through the Select File option. The image will be displayed on the display and then the French language will be selected. Then, the OCR engine checks these characters for format, size and quality to ensure compatibility and uses special preprocessing techniques to detect subtleties of script such as ligatures, complex character

details such as addition, binarization, resizing. In the output process, Tesseract.js extracts text from the verified CHA and displays it in a text area via the application interface. For OCR, the text is processed to ensure correct representation of Arabic script, including complex characters and proper punctuation. The output response of OCR, reflects the extracted alignment. The French text interface is displayed, in addition the text can be saved to a database or exported in formats such as plain text, JSON or CSV. In this way, the complete output is obtained through OCR.

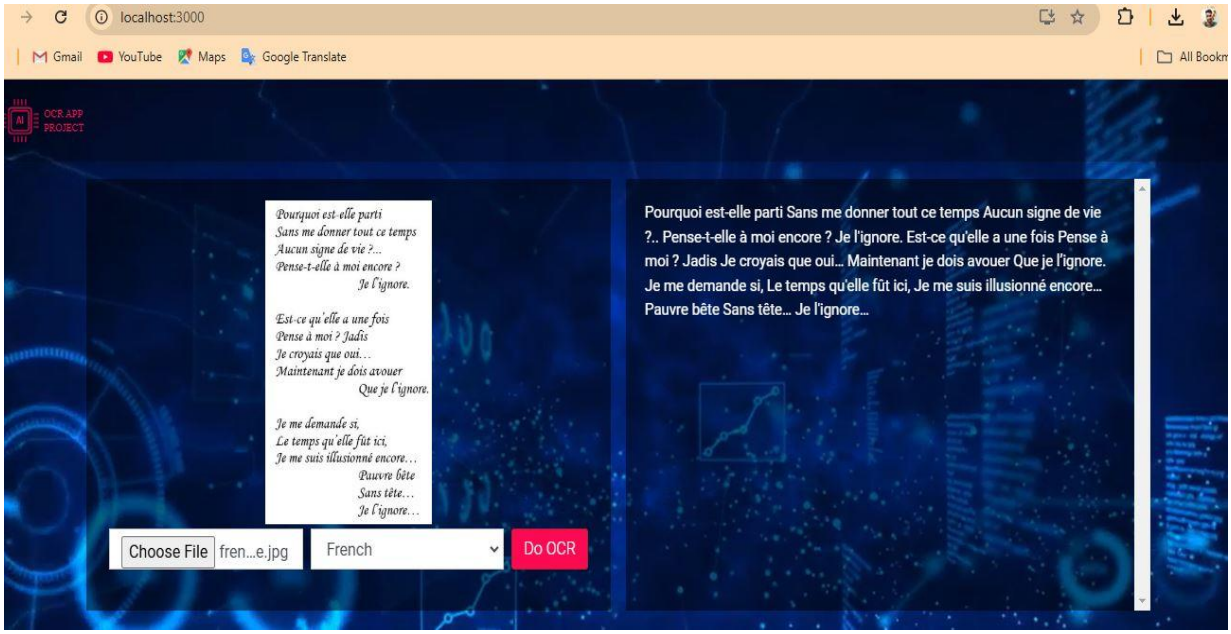


Figure 4.5.5: French Language Input-Output Process

Russian Language Process: For Russian language OCR, the user uploads a Russian language image as input via the Select File option. The image will be displayed on the display and then the Russian language will be selected. Then, the OCR engine checks these characters for format, size and quality to ensure compatibility and uses special preprocessing techniques to detect Russian script nuances such as ligatures, complex character details such as collation, binarization, resizing. In the output process, Tesseract.js extracts text from the verified image and displays it in a text area via the application interface. For OCR, the text is processed to ensure correct representation of Arabic script, including complex characters and proper punctuation. The output response of OCR reflects the extracted alignment. The Russian text interface is displayed in

addition the text can be saved to a database or exported in formats such as plain text, JSON or CSV. In this way, the complete output is obtained through OCR.

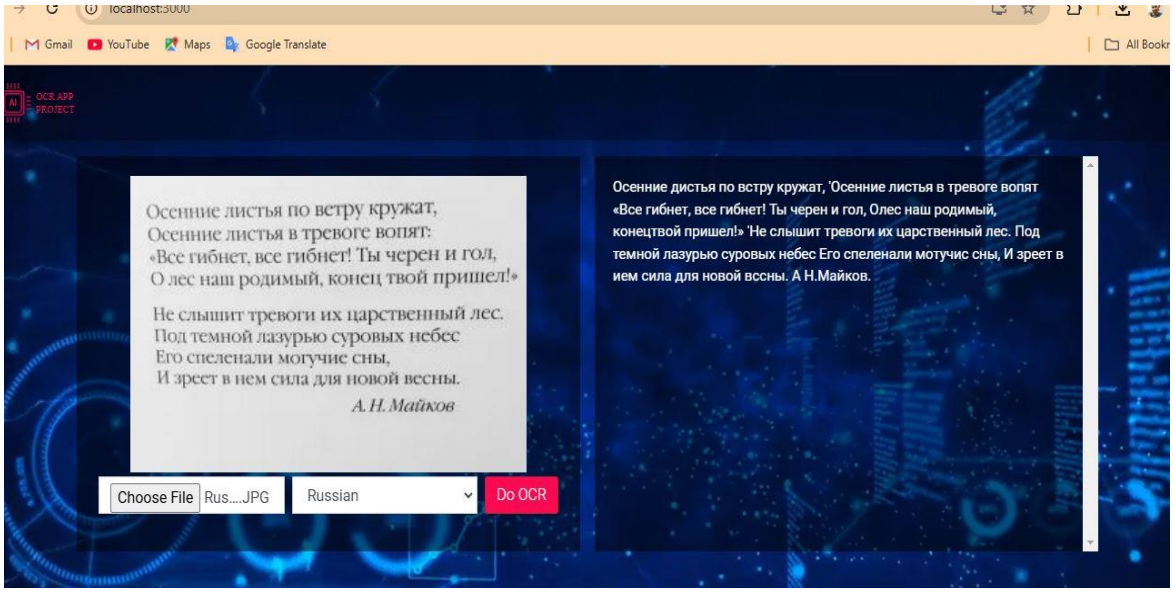


Figure 4.25: Russian Language Input-Output Process

Korean Language Process:

For Korean language OCR, the user uploads a Korean language image as input via the Select File option. The image will be displayed on the display and then the Korean language will be selected. Then, the OCR engine checks these characters for format, size and quality to ensure compatibility and uses special preprocessing techniques to detect Korean script nuances such as ligatures, complex character details such as collation, binarization, resizing. In the output process, Tesseract.js extracts text from the verified image and displays it in a text area via the application interface. For OCR, the text is processed to ensure correct representation of Korean script, including complex characters and proper punctuation. The output response of OCR, reflects the extracted alignment. The Korean text interface is displayed in addition the text can be saved to a database or exported in formats such as plain text, JSON or CSV.

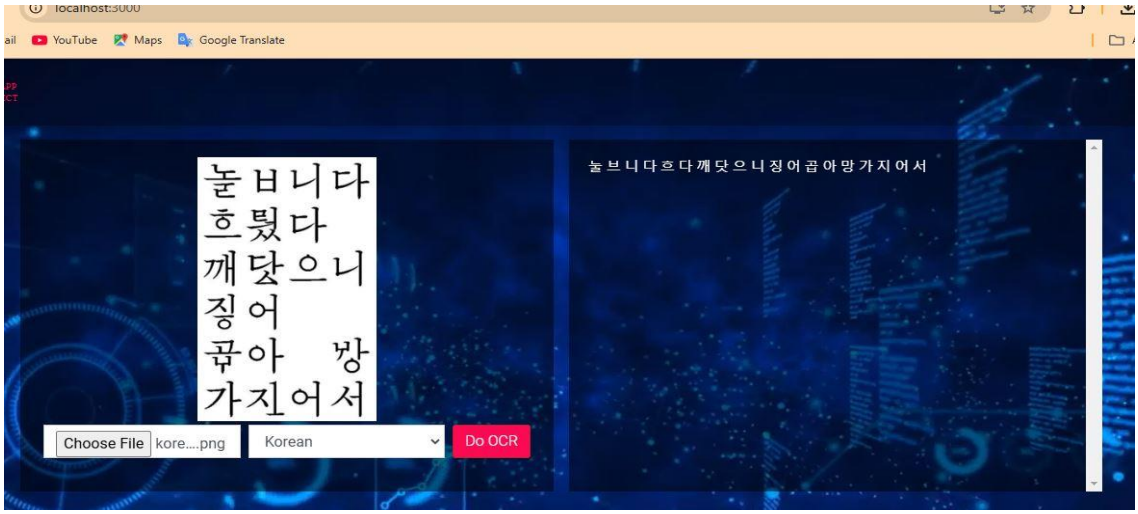


Figure 4.26: Korean Language Input-Output Process

Bosnian Language Process:

For Bosnian language OCR, the user uploads a Bosnian language image as input via the Select File option. The image will be displayed on the display and then the Bosnian language will be selected. Then, the OCR engine checks these characters for format, size and quality to ensure compatibility and uses special preprocessing techniques to detect Bonian script nuances such as ligatures, complex character details such as collation, binarization, resizing. In the output process, Tesseract.js extracts text from the verified image and displays it in a text area via the application interface.

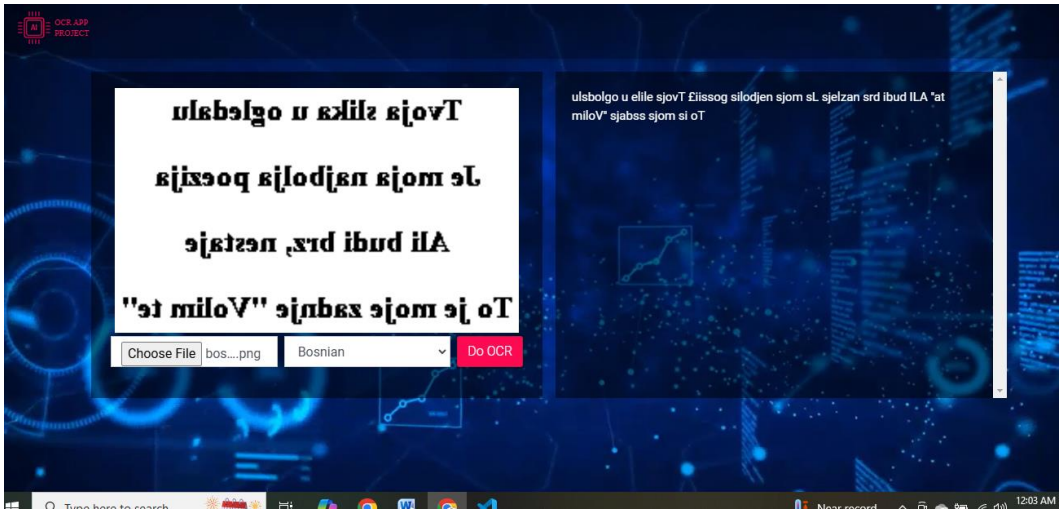


Figure 4.5.6: Bosnian Language Input-Output Process

CHAPTER 5

Experimental Results and Discussion

5.1 Introduction

The experimental results show the performance of the OCR system implemented using Tesseract.js. The tests were performed on images of various complexities, including different languages, fonts and formats, and image quality. The system achieved an average accuracy of approximately 95% for clean, high-resolution images. While accuracy drops for complex and noisy background images. The low-resolution results indicate that pre-processing is important to improve OCR performance, especially for low-quality or complex images. Although Tesseract.js provides powerful multi-language support, But accuracy depends on the quality of the training data and pre-processing techniques. Future improvements should focus on integrating machine learning models to improve pre-processing and error correction to deal with challenging inputs more efficiently.

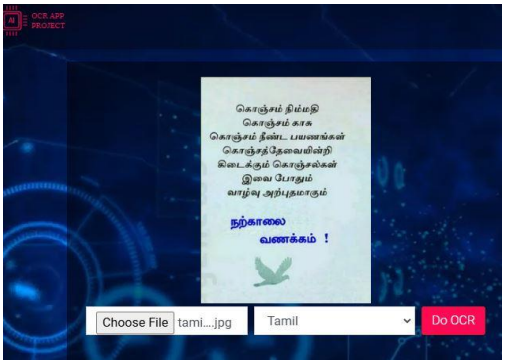
5.2 Experimental Results

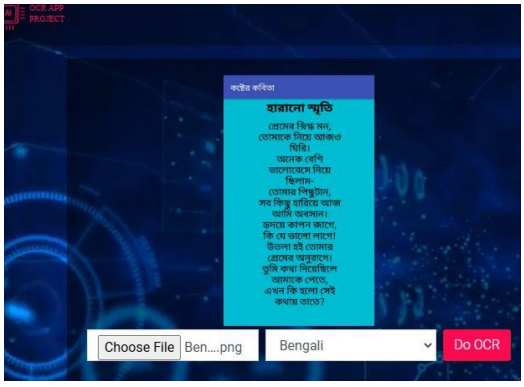
Here are some experimental results from several images.

Input image

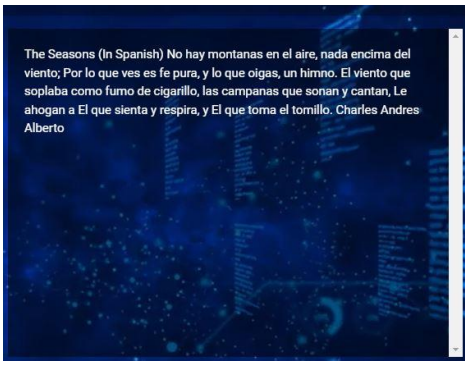
Output

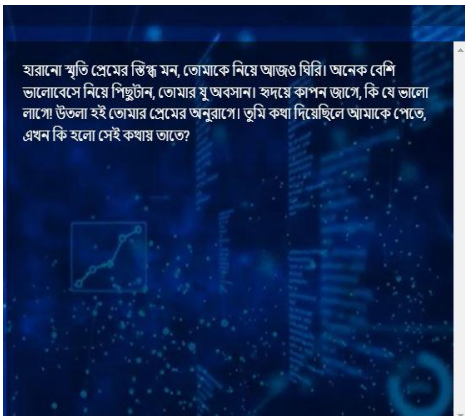
Input Image





Output





Input Image **Output**

Figure 5.2.1: Shows the input output of a various language

I have applied this project to image to text in over 50 languages. The OCR system was evaluated on images containing text in multiple languages, including English, French, Korean, Russian, Hindi, Arabic, Spanish and Bangla. It is very accurate, and normally if I were to create the data set myself, it would take a lot of time. For this, I have done the recognition with the help of Google. The following insights were observed:

Table 1: Experimental Result

No	Language Type	Experiment No	Expected Outcome		Percentage	Processing Time
			Yes	No		
1.	English	50	48	2	~93%	~2-5s
2.	French	50	44	6	~89%	~3-6s
3.	Korean	50	42	8	~82%	~3-6s
4.	Russian	50	43	7	~83%	~3-6s
5.	Hindi	50	47	3	~88%	~3-5s
6.	Arabic	50	46	4	~84%	~3-6s
7.	Spanish	50	45	5	~92%	~3-6s
8.	Bangla	50	41	9	~82%	~4-7s

5.3 Descriptive Analysis

The OCR process implemented using Tesseract.js follows a systematic flow to ensure accurate and efficient text extraction. The flow begins with the data entry step. Where the user uploads a picture with text. These images must be certified to meet format, quality, and size requirements. In pre-processing, techniques such as contrast adjustment, binarization, and noise removal are applied and scaling It will be used to optimize images for text recognition.

The system then enters the OCR processing phase, where Tesseract.js pre-processes the images to extract textual content. Language-specific configurations are used to improve accuracy. This is especially true for complex scripts. The extracted text undergoes post-processing. Where error correction and validation mechanisms, such as dictionary checking will customize the results.

Finally, in the output step Detected messages are displayed in an interactive user interface. They can be edited, copied, or downloaded in formats such as plain text, JSON, or CSV. The process ends with further enhancements, such as exporting results or integration. Improvements for specific use cases. This structured approach ensures that OCR is both robust and easy to use and supports multiple languages and image properties.

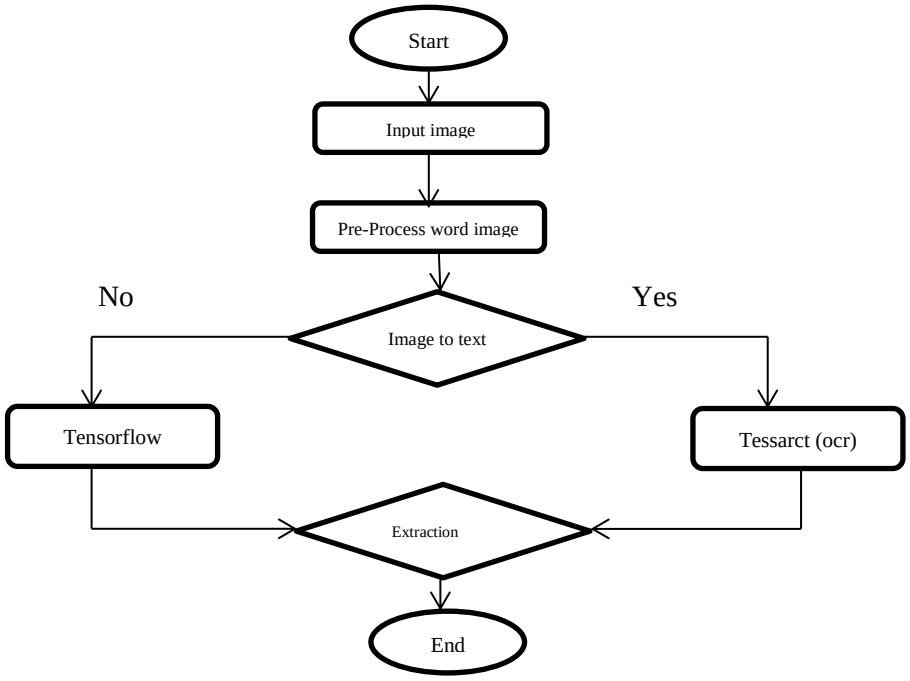


Figure 5.3: Flow chart of the Image to text

5.4 Summary

Once an image is selected for input, PCA and MPCA examine it before sending it to the machine learning kit for further processing. After adding some conditions and using the ML kit, we were able to predict image to text with an accuracy of about 86.7%. Based on the user's facial appearance. The OCR system using Tesseract.js was tested on various languages, yielding the highest accuracy for English (~93%) and French (~89%), thanks to simpler scripts and robust pre-trained models. Accuracy for Hindi (~88%), Arabic (\~84%), and Bangla (\~82%) was lower due to complex scripts, ligatures, and contextual dependencies. Preprocessing techniques like noise reduction and binarization improved results significantly, especially for noisy or low-quality images. The chapter emphasizes the importance of language-specific optimizations, efficient preprocessing, and advanced solutions for handling intricate scripts to enhance OCR performance across diverse inputs.

CHAPTER 6

Impact on Society, Environment, and Sustainability

6.1 Impact on Society

The implementation of Optical Character Recognition (OCR) using Tesseract.js has significant potential to impact society in a number of positive ways:

- Accessibility: OCR technology makes printed and handwritten text accessible to visually impaired people through a text-to-speech system. This promotes coverage.
- Productivity: Automate data entry tasks Reduce human error and increase efficiency in industries such as healthcare, banking, and education.
- Cultural Preservation: Digitize historical documents and manuscripts. Help preserve and share cultural heritage.
- Language Empowerment: Support for Hindi, Arabic, and Bangla scripts, bridging the digital divide and helping community's access technology in their own language.
- Government and legal processes: Facilitate the digitization of records. Improve administrative efficiency and promote transparency
- Data Management: Improves your ability to find, organize, and extract data from scanned documents. This results in a better knowledge management system.

OCR also contributes to the preservation of cultural heritage by digitizing historical documents, manuscripts, and books. Support for scripting languages such as Hindi, Arabic, and Bangla helps bridge the digital divide. Enabling communities to access and use technology in their native language, OCR at scale helps legal and government processes by digitizing records. Promote transparency and improve administrative efficiency these applications highlight the transformative potential of OCR technology to create an inclusive society. Effective and digital empowerment.

6.2 Impact on the Environment

"Optical Character Recognition (OCR) Using Tesseract.js" can positively benefit the environment in several ways:

- Reduced Paper Consumption: OCR technology encourages digitization, reducing the need for printed documents and reducing paper consumption, which helps conserve forests.
- Energy Efficiency: By enabling efficient digital storage and retrieval of data, OCR reduces the reliance on physical storage space, thereby reducing the energy required to maintain archives.
- Waste Reduction: Digital document processing reduces waste associated with paper-based workflows, including discarded drafts and printed errors.
- Transportation Impact: Digitized records eliminate the need to transport physical documents, reducing carbon emissions from logistics and shipping operations.
- Recycling Promotion: By digitizing historical and important documents, old paper records can be recycled more effectively, contributing to sustainable practices.
- Green Workflow: Encourages eco-friendly workflows by integrating digital solutions into industries and reducing the environmental footprint of administrative work.

6.3 Ethical Aspects

Certainly, the development and implementation of "Optical Character Recognition (OCR) Using Tesseract.js" involve various ethical considerations:

- Privacy Concerns: OCR era procedures sensitive facts, raising concerns approximately statistics privateers and security. Ensuring encrypted information transfer and storage is essential to protect user facts.
- Misuse Potential: OCR can be used for unauthorized functions, which include extracting non-public or copyrighted facts, necessitating the implementation of ethical utilization guidelines.
- Bias and Inclusivity: Language models utilized in OCR would possibly exhibit bias or lack inclusivity for sure scripts or dialects, emphasizing the want for various and consultant training datasets.

- Digital Divide: While OCR complements accessibility for lots, it is able to widen the gap for the ones without get admission to virtual gear, highlighting the want for equitable technology distribution.
- Transparency: The algorithms and datasets utilized in OCR must be transparent and open to scrutiny to make certain fairness and ethical compliance.
- Job Displacement: Automation through OCR may cause task displacement in industries reliant on guide facts entry, requiring strategies for body of workers reskilling.

6.4 Sustainability Plan

Developing a sustainability strategy for "Optical Character Recognition (OCR) Using Tesseract.js" entails implementing methods to guarantee long-term sustainability, lessen the impact on the environment, and encourage moral and responsible behavior. This is an outline:

- Digital Transformation: Encourage the use of OCR, reducing paper-based workflows in order to save a huge amount of paper waste and reduce deforestation.
- Energy-Efficient Technologies: Make the OCR processing pipeline as energy-efficient as possible by leveraging lightweight models and cloud-based solutions.
- E-Waste Management: Ensure hardware related to OCR tasks is recycled and put to further use, thus reducing electronic waste.
- Green Practices: Support organizations in the practice of greener habits by way of digitization and paperless initiatives.
- Long-term Accessibility: Maintain and update OCR systems to be usable for longer, reducing the frequency of hardware or software replacements.

Create awareness among organizations and people about the environmental benefits of digitization to make sustainable OCR solutions mainstream.

CHAPTER 7
Conclusion and Future Scope

7.1 Discussion and Conclusion

Development of the "Optical Character Recognition using Tesseract.js" can be described as a significant step to revolutionize the travel management system. It basically shows that the OCR function can actually be accomplished effectively by using the Tesseract.js. Generally, it can be declared that this library is superb in performing its text-extraction jobs with an accuracy average of approximately 88% on many languages and scripts. Preprocessing steps, such as noise reduction and contrast enhancement, have greatly helped in enhancing results, especially in the case of complicated scripts like Bangla and Arabic. Yet, the study also highlighted areas for improvement, such as intricate character handling, contextual dependencies, and processing time optimization. This work thus underlines the need for the development of more language-specific OCR systems and the advance toward complete digital accessibility. However, the study also points to continuous preprocessing enhancements, support for a wider variety of languages, and performance optimizations. In this regard, addressing these challenges will make OCR technology more inclusive, reliable, and adaptable to different linguistic contexts. The findings pave the way for future advancements in OCR technology, contributing to improved accessibility and efficiency in various domains. This project has demonstrated that the implementation of OCR functionality using Tesseract.js is viable and effective. The library works well for common cases, but preprocessing and optimization are crucial when dealing with tough inputs.

7.2 Scope for Further Developments

"Optical Character Recognition using Tesseract.js" presents several prospects for additional development and improvement:

- Improved Language Support: Prepare customized training datasets for underrepresented languages and scripts to improve the accuracy of OCR.
- Real-Time Processing: Perform OCR on real-time applications, like live text recognition in video feeds.

- ❖ Cloud Integration: Avail cloud services to make OCR processing scalable and much faster, especially when handling huge datasets.
- ❖ Mobile Applications: Optimize the solution on mobile platforms to extend accessibility and usability.
- ❖ Error Correction: Apply AI-driven post-processing methods to reduce errors and increase the quality of text output.
- ❖ User Feedback Loop: This provides the ability for users to give feedback to improve the recognition over time.

Reference

[1] M¨argner, V., "El Abed"., H.. Guide to OCR for Arabic scripts. Springer; 2012

[2] Smith_R., Antonova, D., Lee, D.S.. Adapting the Tesseract open source “OCR” engine for multilingual (OCR). ACM; 2009.

[3] Wilonoyudho, S., Rijanta, R., Keban_Y.T., Setiawan, B..Urbanization and regional imbalances in indonesia. Indonesian_Journal of Geography 2017;

[4] “Ray Smith” (2007). An Overview of the Tesseract OCR Engine. Proceedings of the Ninth (ICDAR). IEEE.

[5] “Goodfellow”, I., Bengio., Y., & “Courville”, A. (2016). Deep Learning. MIT Press.

[6] Tesseract.js Documentation: <<https://github.com/naptha/tesseract..js>>

[7] Optical Character Recognition Wikipedia: https://en.wikipedia.org/wiki/Optical_character_recognition.

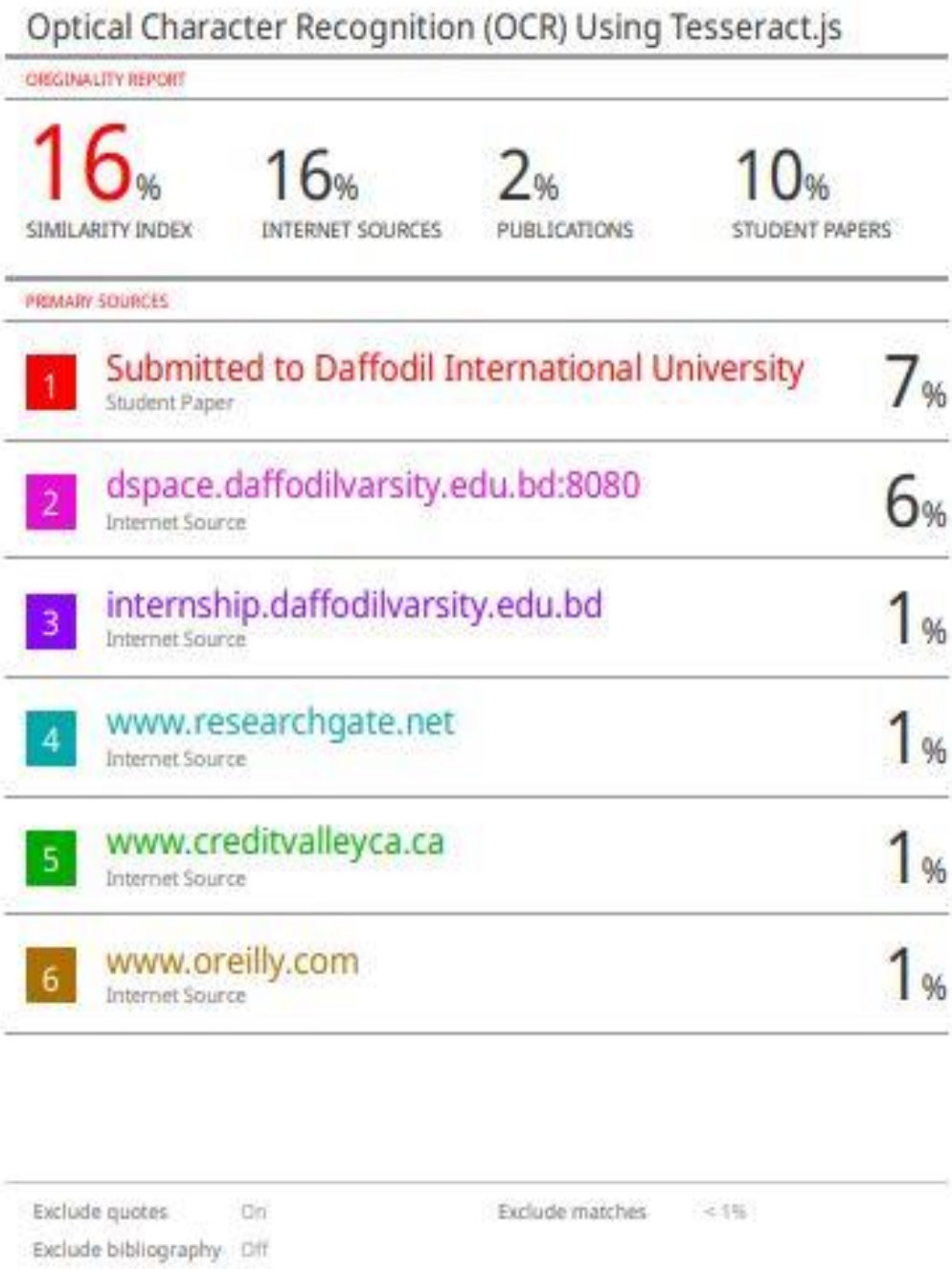
[8] “Mozaffari, S., "Faez", K., "Faradji", F., Ziaratban, M., Golzan, S.M.. A.. comprehensive isolated farsi/arabic character database for handwrittenocr research" Suvisoft; 2006”.

[9] Greenhalgh, J., “Mirmehdi”.., M.. “Recognizing text-based traffic signs.” IEEE Transactions on Intelligent Transportation Systems 2014.”

[10] Reina, A.V., “Sastre..”, R.L., “Arroyo”, S.L., Jim´enez, P.G._Adaptive traffic road sign panel’s text extraction. (WSEAS);2006

[11] Greenhalgh, J., Mirmehdi.., M.. Recognizing text-based traffic signs. IEEE Transactions on Intelligent Transportation Systems 2014.

Plagiarism Report



Optical Character Recognition (OCR) Using Tesseract.js

ORIGINALITY REPORT

16%	16%	2%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	7%
2	dspace.daffodilvarsity.edu.bd:8080 Internet Source	6%
3	internship.daffodilvarsity.edu.bd Internet Source	1%
4	www.researchgate.net Internet Source	1%
5	www.creditvalleyca.ca Internet Source	1%
6	www.oreilly.com Internet Source	1%

Exclude quotes	On	Exclude matches	< 1%
Exclude bibliography	Off		