RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Go, change the world®

DEPARTMENT OF

INFORMATION SCIENCE AND ENGINEERING

Innovative Experiment
Report On

"Image to text"

By

1RV23SIT03 Ashwini K J

1RV23SIT05 Gahana S

1RV23SIT16 Varuni H Kulkarni

Under the Guidance of

Prof. Rashmi R

Assistant Professor

Dept. of ISE

RV College of Engineering®

Course Name: API Development and Integration Lab

Course Code: MIT438L

SEPT 2023 - 24

Table of Contents

Extracting Text from Images in Multiple Languages using OCR

INTRODUCTION

In today's data-driven world, the ability to efficiently convert printed text into digital formats is crucial for accessibility, information management, and automation. Optical Character Recognition (OCR) technology plays a key role in enabling this by extracting text from images, making it machine-readable and actionable. This project, titled "Extracting Text from Images in Multiple Languages using OCR," focuses on developing a system that recognizes and extracts text from images in Kannada, English, and Sanskrit. The application, built using Spring Boot and Java, leverages OCR technology to automate the conversion of printed materials into digital formats. This is particularly useful in a variety of fields, such as education, business, healthcare, and government, where the digitization of documents enhances searchability, archiving, translation, and data entry processes. The project addresses the growing need for efficient text extraction solutions that can handle multilingual content and offers a practical approach to modern-day document management.

SOFTWARE REQUIREMENTS WITH VERSION, INSTALLATION PROCEDURES

Following is the list of software requirements specified in order of installation:

Software Name	Version	Installation link	Purpose
Java Development Kit (JDK)	11 or higher	https://www.oracle.com/java/technologies/downloads/#java11	Development environment for Java applications.
Spring Boot	2.6.7 or compatible	https://start.spring.io/	Framework for building RESTful APIs and microservices.
Eclipse IDE	2023-03 or higher	https://eclipseide.org/	Integrated development environment (IDE) for Java development.
Tesseract OCR	5.0.1 or higher	https://github.com/tesseract-ocr/tesseract	OCR library for extracting text from images.
Maven	3.8.6 or higher	https://maven.apache.org/download.cgi	Build tool for managing dependencies and project lifecycle.
Postman	10.1 or higher	https://www.postman.com/downloads/	Tool for testing and interacting with APIs.

SOURCE CODE LINK (GITHUB):

OVERVIEW OF THE ESSENTIAL ASPECTS OF THE API

The tables below provides a clear and concise overview of the essential aspects of the created APIs, making it easy for users to understand and reuse:

Section	Details
API Overview	**Name: Image Text Extractor API** **Version: 1.0** **Base URL: http://localhost:8080/api**
Authentication	**No authentication required for this version.**
Endpoints	**GET /extract-text** **Description: Extract text from an uploaded image using OCR.** **Parameters: file (multipart/form-data)** **Request Example: POST /extract-text** **Response Example: { "extracted_text": "The quick brown fox..." }**
Error Handling	**400 Bad Request:: If no image file is provided in the request or if the image is invalid**
Rate Limiting	**Limit: 1000 requests per hour** **Handling: Returns 429 status code if exceeded**
Usage Examples	**Python Example:** **import requests** **response = requests.get('https://api.example.com/v1/users/123',** **headers={'Authorization': 'Bearer YOUR_API_KEY'})** **print(response.json())**
Additional Resources	**API Documentation: Full Documentation** **Support: Contact Support**

LIST OF APIS WITH ITS PURPOSE

API Name	**Requirements**	**Installation link**	**Purpose**
Tesseract OCR API	Endpoint: Internal API (Local Library Call) Request Method: Direct image file processing Authentication: None Response Format: Extracted text from image (String)	https://github.com/tesseract-ocr/tesseract/wiki	Used to extract text from the provided image files using Optical Character Recognition (OCR) technology.
Hugging Face GPT-2	Endpoint: /v1/text-generation (Provided by Hugging Face API) Request Method: POST Authentication: API Key Request Format: JSON with inputs parameter (String) Response Format: JSON with generated_text field containing the processed or refined text		Used for refining and enhancing the extracted text from OCR by applying a text generation model to correct or augment the output.
Image Text Extractor API	Endpoint: /extract-text (Local) Request Method: POST with Multipart Form Data (Image File) Authentication: None Response Format: Plain text (String) containing extracted text from image	Not Applicable (User Application)	Acts as a service to extract text from uploaded images using the Tesseract OCR and, optionally, refines it using the Hugging Face API.

DESCRIPTION ABOUT EACH MODULE

1]. ImageTextController Class (API Endpoint)

The ImageTextController class is the core component of the REST API, enabling users to upload images and extract text using Tesseract OCR. It defines a POST endpoint /extract-text, where users can send image files. The class handles the image upload, processes it (though currently without modification), runs OCR to extract text using Tesseract, and returns the extracted text as the response. It includes a key method, preprocessImage, which is designed to preprocess the image to improve OCR accuracy but currently returns the original image without any modifications.

2]. ImageTextExtractorApplication Class

The ImageTextExtractorApplication class serves as the entry point for the Spring Boot application. It is responsible for starting up the Spring application context. The main method, public static void main(String[] args), initializes and launches the Spring Boot application, setting up the necessary environment for the REST API and other application components to function correctly.

3]. HTML Frontend (for Image Upload)

The index.html file provides a user-friendly frontend for uploading images to the backend service. It features a form with a file input element (<input type="file" name="file" accept="image/*" required>) that allows users to select image files. The form uses the POST method to submit the image to the /api/extract-text endpoint, enabling the backend to process and extract text from the uploaded image.

4]. pom.xml (Maven Project Object Model)

The pom.xml file is the Maven configuration file for the project, detailing the project's dependencies, build configurations, and plugins. It ensures that essential libraries and tools, such as Spring Boot and Tesseract OCR, are included and correctly managed. This configuration file enables smooth project build and dependency management, facilitating the development and execution of the application.

IMPLEMENTATION DETAILS WITH TOOLS USED

1. Spring Boot Framework

- Tool Used: Spring Boot
- Purpose: Spring Boot was chosen as the framework for developing the backend of the application due to its ease of use and ability to create stand-alone, production-grade Spring-based applications with minimal configuration.

Implementation Details:

- The Spring Boot Starter Web dependency is used for setting up the web server and handling HTTP requests.
- The main application class is ImageTextExtractorApplication, which is the entry point for the Spring Boot application, and it uses the @SpringBootApplication annotation to enable component scanning and auto-configuration.
- The ImageTextController class is responsible for exposing RESTful API endpoints to handle file uploads and process image data.

2. REST API for Image Upload and Text Extraction

- Tool Used: Spring MVC
- Purpose: Spring MVC (Model-View-Controller) is used to implement REST APIs that allow users to upload images and retrieve the extracted text.

Implementation Details:

- The API endpoint /api/extract-text is defined using the @PostMapping annotation in the ImageTextController class.
- It accepts file uploads through the @RequestParam annotation with the parameter MultipartFile file.
- The file is saved to a temporary directory, pre-processed, and passed to the OCR module for text extraction.

3. **Optical Character Recognition (OCR)**

- Tool Used: Tesseract OCR via Tess4J
- Purpose: Tesseract is an open-source OCR engine that reads images and extracts textual data. Tess4J is a Java wrapper for Tesseract, making it easy to integrate with Java applications.

Implementation Details:

- The Tesseract class from the Tess4J library is initialized with the path to the Tesseract data files (tessdata).
- The method doOCR() is used to perform the actual OCR processing on the uploaded image.
- The OCR engine is configured to support both English and Kannada languages, as indicated by the tesseract.setLanguage("kan+eng") configuration.

4. **Frontend for File Upload**

- Tool Used: HTML/CSS
- Purpose: A simple web form is created to allow users to upload image files for text extraction.

Implementation Details:

- An HTML form with the enctype="multipart/form-data" attribute is used to allow file uploads.
- The form contains an input field for file selection and a submit button that triggers the file upload to the /api/extract-text endpoint.
- Basic CSS is used to style the form, providing a user-friendly interface.

5. **Maven for Dependency Management**

- Tool Used: Maven
- Purpose: Maven is used to manage the dependencies and build the project efficiently.

Implementation Details:

- Dependencies like spring-boot-starter-web (for REST APIs) and tess4j (for OCR functionality) are defined in the pom.xml file.
- The Spring Boot Maven Plugin is used to package the application as a JAR file, making it easy to deploy.

WORKING PROCEDURE

The Image Text Extractor application is designed to extract text from images using Optical Character Recognition (OCR). The following outlines the step-by-step working procedure of the application:

1. Uploading an Image

- Frontend (User Interaction):
    - The user accesses a web interface created using HTML/CSS.
    - The interface contains a simple form where the user can upload an image file (in formats such as JPEG, PNG, etc.).
    - The form sends an HTTP POST request to the server when the user clicks the "Extract Text" button.

2. Receiving the Uploaded Image

- Backend (Spring Boot Controller):
    - The server listens for the POST request at the /api/extract-text endpoint.
    - The ImageTextController class in the backend handles this request. It uses the @PostMapping annotation to define the API endpoint.
    - The image is received as a MultipartFile object and stored temporarily in the system's temporary directory.

3. Preprocessing the Image

- Image Preprocessing (Optional):
    - The application has a placeholder method preprocessImage() that currently does not alter the image but can be used to enhance the quality of the image for better OCR results.
    - Preprocessing techniques such as converting to grayscale, resizing, or increasing contrast could be added to this step if required.

4. Optical Character Recognition (OCR) with Tesseract

- Text Extraction:
    - After preprocessing, the image is passed to the Tesseract OCR engine, integrated via the Tess4J Java wrapper.

- The Tesseract instance is initialized and configured with the necessary language data files (both English and Kannada are supported in this application).
- The doOCR() method is used to extract text from the processed image.

5.Returning the Extracted Text

- Response:
  - The text extracted from the image is returned as a string response from the server to the frontend.
  - This response is displayed on the user's browser as the extracted text.

6.Error Handling

- Handling Failures:
  - If any error occurs during image processing, OCR, or file handling, the system catches the exceptions (TesseractException or IOException) and returns a meaningful error message to the user.

CODE

main

```
package com.example.imagetextextractor;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ImageTextExtractorApplication {

    public static void main(String[] args) {
        SpringApplication.run(ImageTextExtractorApplication.class, args);
    }
}
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>image-text-extractor</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <!-- Spring Boot Starter Web -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>2.7.18</version>
        </dependency>
        <!-- Tesseract for OCR -->
        <dependency>
            <groupId>net.sourceforge.tess4j</groupId>
            <artifactId>tess4j</artifactId>
            <version>4.5.5</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

create one more file named: application.properties

```
server.port=8085
```

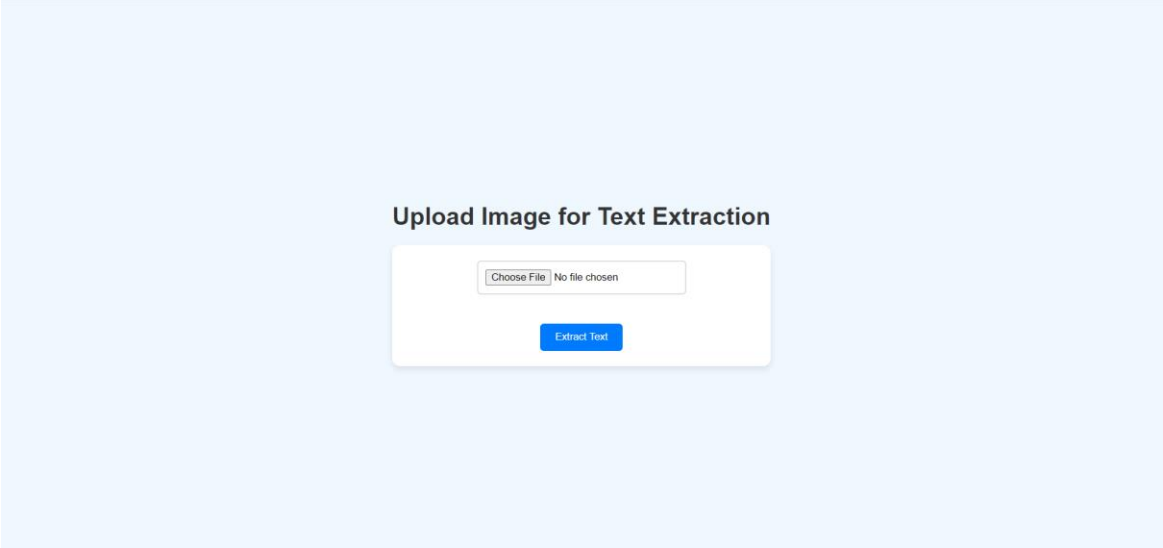
controller code

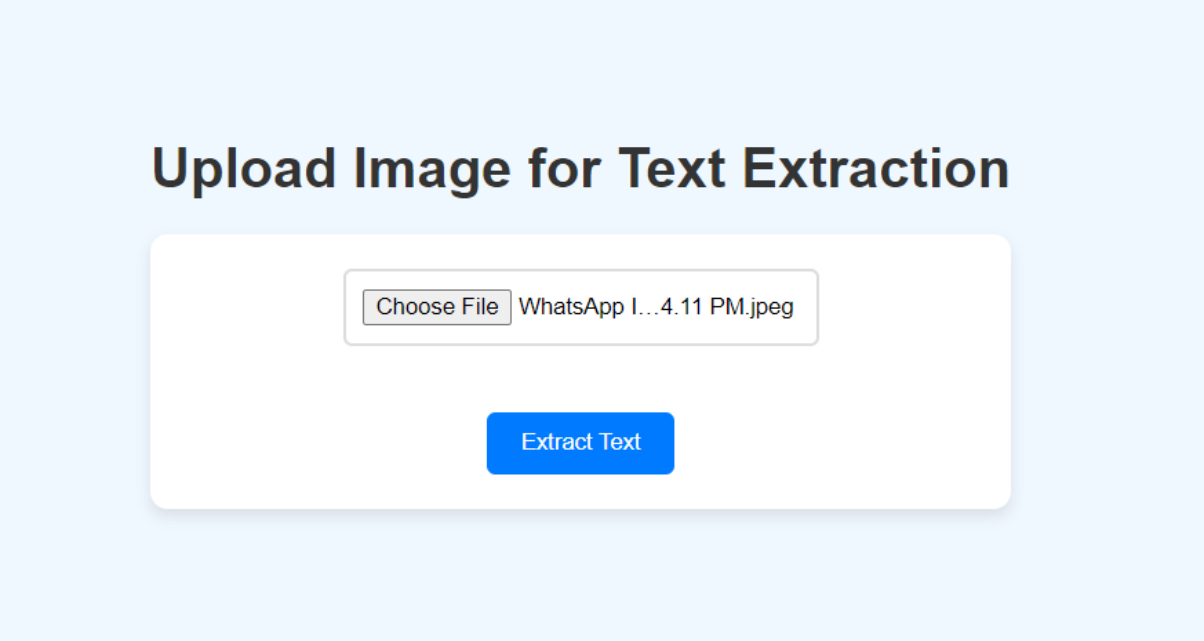
```
package com.example.imagetextextractor;

import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import net.sourceforge.tess4j.*;

import java.io.File;
import java.io.IOException;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

@RestController
@RequestMapping("/api")
public class ImageTextController {

   @PostMapping("/extract-text")
   public String extractTextFromImage(@RequestParam("file") MultipartFile file) {
      try {
         File convFile = new File(System.getProperty("java.io.tmpdir") + "/" +
file.getOriginalFilename());
         file.transferTo(convFile);

         BufferedImage image = ImageIO.read(convFile);
         BufferedImage processedImage = preprocessImage(image);
         File processedFile = new File(System.getProperty("java.io.tmpdir") + "/processed_" +
file.getOriginalFilename());
         ImageIO.write(processedImage, "jpeg", processedFile);

         Tesseract tesseract = new Tesseract();
         tesseract.setDatapath("C://Program Files//Tesseract-OCR//tessdata");
         tesseract.setLanguage("kan");

         String text = tesseract.doOCR(processedFile);

         return text;

      } catch (TesseractException | IOException e) {
         e.printStackTrace();
         return "Error occurred while extracting text from image.";
      }
   }

   private BufferedImage preprocessImage(BufferedImage image) {

      return image;
   }
}
```

html code

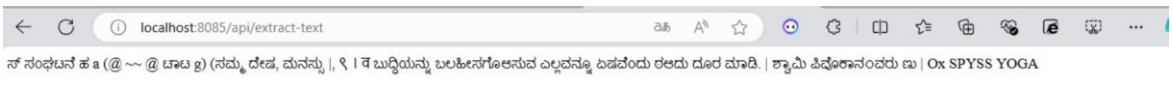
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Image Text Extractor</title>
<style>
body {
background-color: #f0f8ff; /* Light blue background */
font-family: Arial, sans-serif;
display: flex;
justify-content: center;
align-items: center;
height: 100vh;
margin: 0;
}
h1 {
color: #333;
text-align: center;
margin-bottom: 20px;
}
form {
background-color: #fff;
padding: 20px;
border-radius: 10px;
box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
text-align: center;
}
input[type="file"] {
padding: 10px;
margin-bottom: 20px;
border: 2px solid #ddd;
border-radius: 5px;
cursor: pointer;
}
input[type="submit"] {
padding: 10px 20px;
border: none;
background-color: #007bff;
color: white;
border-radius: 5px;
cursor: pointer;
transition: background-color 0.3s ease;
}
input[type="submit"]:hover {
background-color: #0056b3;
}
</style>
</head>
<body>
<div>
<h1>Upload Image for Text Extraction</h1>
<form action="/api/extract-text" method="post" enctype="multipart/form-data">
<input type="file" name="file" accept="image/*" required>
<br><br>
<input type="submit" value="Extract Text">
</form>
</div>
</body>
</html>
```

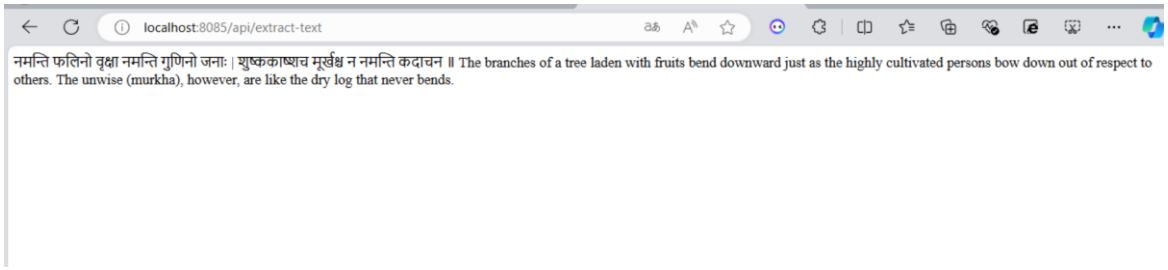
SCREENSHOTS

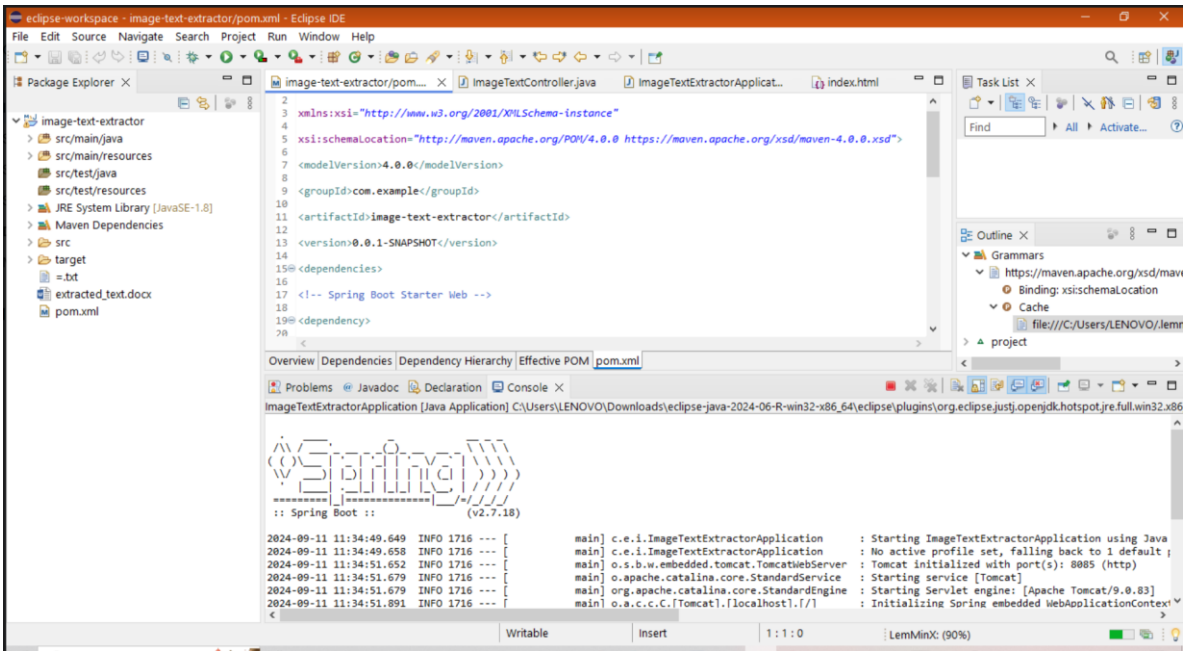


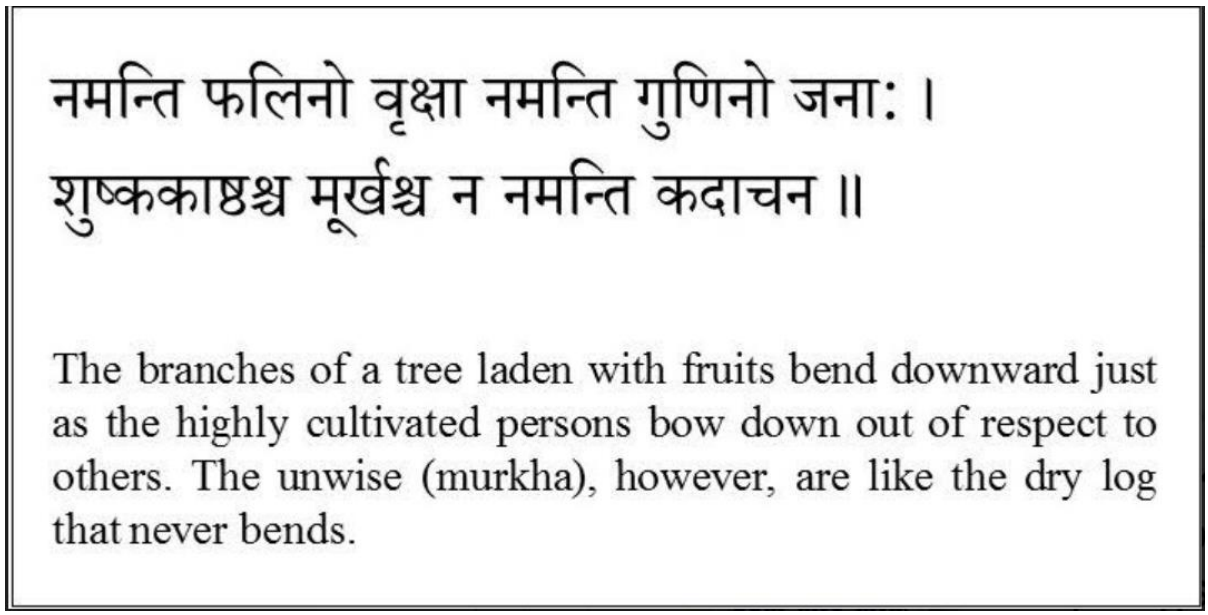



ಸಂಸ್ಕಾರ
ಸಂಘಟನೆ
ಸೇವೆ
SPYSS
Free Yoga Since 1980
ಶ್ರೀಪಯೋಶಿಸ
1980 ರಿಂದ ಉಚಿತ ಯೋಗ
ನಿಮ್ಮ ದೇಹ, ಮನಸ್ಸು
ಮತ್ತು ಬುದ್ಧಿಯನ್ನು
ಬಲಹೀನಗೊಳಿಸುವ
ಎಲ್ಲವನ್ನೂ ವಿಷವೆಂದು
ತಿಳಿದು ದೂರ ಮಾಡಿ.
ಸ್ವಾಮಿ ವಿವೇಕಾನಂದರು
SPYSS YOGA


localhost:8085/api/extract-text
ಸ್ ಸಂಘಟನೆ ಹ a (@ ~~ @ ಟಾಟ ಔ) (ನಮ್ಮ ದೇಹ, ಮನಸ್ಸು |, ೪ । ಡ ಬುದ್ಧಿಯನ್ನು ಬಲಹೀನಗೊಆಸುವ ಎಲ್ಲವನ್ನೂ ವಿಷವೆಂದು ಠಳದು ದೂರ ಮಾಡಿ. | ಶ್ಯಾಮಿ ವಿವೇಕಾನಂದರು ಣು | Ox SPYSS YOGA


localhost:8085/api/extract-text
नमन्ति फलिनो वृक्षा नमन्ति गुणिनो जनाः | शुष्ककाष्ठाच मूर्खश्च न नमन्ति कदाचन ॥ The branches of a tree laden with fruits bend downward just as the highly cultivated persons bow down out of respect to others. The unwise (murkha), however, are like the dry log that never bends.


eclipse-workspace - image-text-extractor/pom.xml - Eclipse IDE
File Edit Source Navigate Search Project Run Window Help
Package Explorer
image-text-extractor
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
=.txt
extracted_text.docx
pom.xml
image-text-extractor/pom....
ImageTextController.java
ImageTextExtractorApplicat...
index.html
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"
<modelVersion>4.0.0</modelVersion>
<groupId>com.example</groupId>
<artifactId>image-text-extractor</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>
<!-- Spring Boot Starter Web -->
<dependency>
Overview Dependencies Dependency Hierarchy Effective POM pom.xml
Problems Javadoc Declaration Console
ImageTextExtractorApplication [Java Application] C:\Users\LENOVO\Downloads\eclipse-java-2024-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86
:: Spring Boot :: (v2.7.18)
2024-09-11 11:34:49.649 INFO 1716 --- [main] c.e.i.ImageTextExtractorApplication : Starting ImageTextExtractorApplication using Java
2024-09-11 11:34:49.658 INFO 1716 --- [main] c.e.i.ImageTextExtractorApplication : No active profile set, falling back to 1 default
2024-09-11 11:34:51.652 INFO 1716 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8085 (http)
2024-09-11 11:34:51.679 INFO 1716 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-09-11 11:34:51.679 INFO 1716 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.83]
Task List
Outline
Grammars
Binding: xsi:schemaLocation
Cache
project
Writable
Insert
1 : 1 : 0
LemMinX: (90%)


नमन्ति फलिनो वृक्षा नमन्ति गुणिनो जनाः ।
शुष्ककाष्ठश्च मूर्खश्च न नमन्ति कदाचन ॥
The branches of a tree laden with fruits bend downward just as the highly cultivated persons bow down out of respect to others. The unwise (murkha), however, are like the dry log that never bends.

CONCLUSION

The Image Text Extractor project demonstrates the successful integration of Optical Character Recognition (OCR) technology into a Spring Boot web application, providing an efficient solution for extracting text from images. Through the use of the Tesseract OCR engine, the application enables users to upload images via a simple web interface and retrieve the extracted text quickly. This functionality can be useful in various domains, including document digitization, data extraction from scanned documents, and accessibility solutions.
The project ensures that text extraction is not only accurate but also flexible, with support for multiple languages such as English and Kannada. The architecture of the application, built on Spring Boot, allows for scalability, making it easy to extend with additional features or deploy in production environments. The modular design enables further enhancements, such as adding more sophisticated image preprocessing or integrating advanced AI models for text refinement.

In summary, the Image Text Extractor project highlights the power of combining Spring Boot's backend capabilities with Tesseract OCR to create a user-friendly, scalable application for text extraction. It serves as a solid foundation for future developments in image-based text analysis and demonstrates how modern web technologies can be effectively used to build practical and innovative solutions.