National Institute Of Technology Karnataka, Surathkal

DEPARTMENT OF INFORMATION TECHNOLOGY ENGINEERING

Document Search System using Parallel Approach

IT301- Parallel Computing (Project Proposal)

Submitted by

Gaurav Chaurasia (181765, 181CV155)

gaurav.181CV155@nitk.edu.in,

9380042745

Upendra Pal (181780, 18ME282)

upendrapal.181ME282@nitk.edu.in,

6203139712

Batch 2022 EVEN SEM 2020-21

Submitted to

Dr. Geetha V

Assistant Professor

Table of Content

Acknowledgment

We take this opportunity to express our sincere gratitude and profound thanks to our guide **Dr. Geetha V**, Associate Professor, Department of Information Technology, National Institute of Technology Karnataka, Surathkal, and **Dr. Biju R Mohan**, Professor, and Head of the Department of Information Technology, National Institute of Technology Karnataka, Surathkal for providing us the opportunity to work on this relevant topic of Parallel search implementation on document search. We seek their regular guidance and encouragement to work on this project and get success. We hope to learn a lot about the practical application of our classes through their proper guidance.

Gaurav Chaurasia (181CV155)

Upendra Pal (181ME282)

Abstract

As we know today's world is full of information and most often we need to search for any key in the given documents for fulfilling our work. But most often, such large documents take a long time to perform a search. So, here is what we are going to work upon: to reduce the time of search by applying a parallel search process using multiple threads. Our project will not only focus on search on the .txt file but later on it will also do a search on the most popular pdf format**.** This will speed up the search and also will give some relevant output like where the word is found and also accompanied by the link to go to that portion of the document, word count, page number, and many more fascinating options.

This would be significant and easy to operate as the user can just upload the document on the website and search his required words or phrases and easily proceed further on his work. In our project, we will show how the parallel search using multi threads will take less time than sequential search, and how we can use this fact.

The idea behind this project is to assign different pages/portions of the documents to different threads which will be processed simultaneously and then combining the results of each thread and displaying them in a creative way for better UX. We can also use the searched word as a link and go to that portion of the page directly. Later on, we can extend our idea and make this an app.

Introduction

As the name "**Document Search System using Parallel Approach**" suggests that in this project, we are searching the word entered by the user in the uploaded document. This will not only search the word and give some portion of texts about the searched word, but it will also take less time to do so as we will be using the parallel approach by distributing the task among different threads. This will be a great help as we can go to the searched word directly in the lesser time and continue our work. The Results will be displayed on a web page to render the result.

Ultimately we will be building a web application that will have an option to upload documents and then give the string of data to which you want to search in the document

We will be using Nodejs as a backend for providing an option to upload the document which can be further used for different analytical purposes here we will be using also be using `Node Js Addons` to implement our search functionality in the `C++` programming language

Nodejs Addons

A Native Addon in the context of Node.js is a binary file that is compiled from a low-level language like `C` and `C++`. Hence like importing a JavaScript file using require, we would be importing a Native Addon.

Native Addon like any other `.js` file exposes its API on `module exports` or `exports` object. A collection of these files when wrapped inside a node module also called as Native Module

A Native Addon is normally written in C or C++ language and compiled using a standard compiler to a Dynamically Linked Library (DLL). It is also called a Shared Library or Shared Object (SO).

A DLL can be loaded into a program dynamically, at runtime. This DLL contains the compiled native code of our C or C++ program and an API to communicate with this compiled code

Ease of Use

This web application is going to be very user-friendly, as soon as the user goes to the web application he/she will be having an option to upload any sort of document, and as soon as the document gets uploaded he/she will get an option to do multiple operations on the document i.e. searching any string/word in the document. And our application will do the operation in parallel using multiple threads and then will show the output in a creative way so that the user can get all the relevant information. And the user experience(UX) is also good.

The technology we used...

C++
OpenMP (for parallelization of the code)
HTML, CSS, Javascript (Jquery), **Bootstrap** , **Mysql**(Database).
NodeJS,
NodeJS Addons (for executing C++ parallel code from javascript)

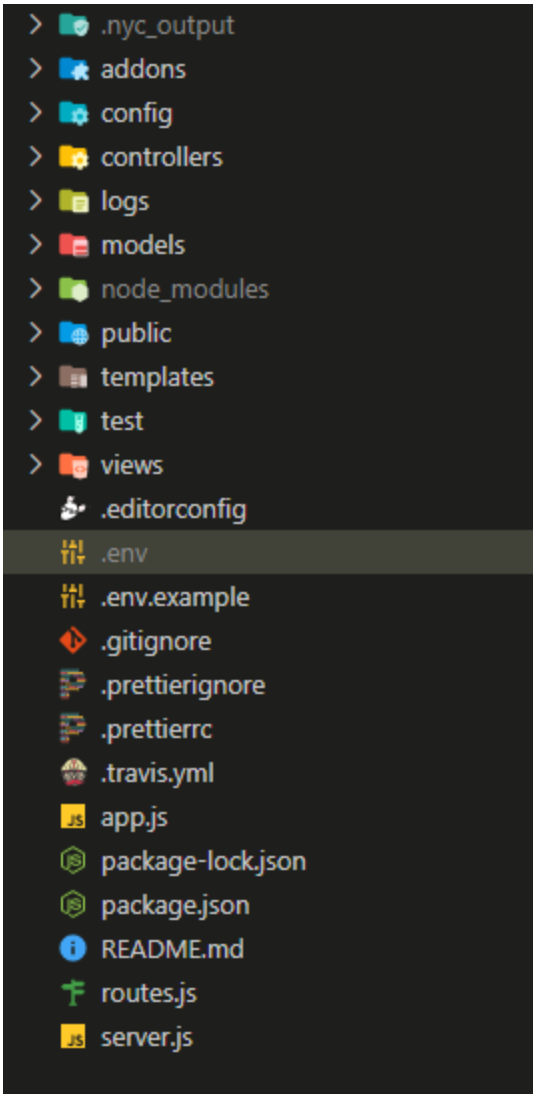
Linear Search Pseudocode

```
procedure linear_search (list, value)

   for each item in the list
      if match item == value
         return the item's location
      end if
   end for
end procedure
```

Implementation

- Here In this application we have tried it to make production ready and best practices like adding **Logger**, **unit testing and continuous integration** using **Travis-ci** ...etc
- Also we have used **MVC - Model View Controllers** architecture
- addons folder contains python script which was planned to then do document processing part the return the response



Parallel program for Document processing

- Initially idea was to some how use cpp OpenMP parallel program for document processing from NodeJs web application
- cpp is compiled language and a part of node js written in **cpp** and runs on v8 engine
- Using a feature in nodejs called **addons** we can write cpp programs and run it from node application
  It execute the c++ program and crates the c++ executable which can be imported as normal javascript package or function and called from node application

```
using namespace v8;
{
    using v8::FunctionCallbackInfo;
    using v8::Isolate;
    using v8::Local;
    using v8::Object;
    using v8::String;
    using v8::Value;
    using v8::Number;
    void Method(const FunctionCallbackInfo<Value> &args) {
        Isolate *isolate = args.GetIsolate();
        args.GetReturnValue().Set();
    }
    void Initialize(Local<Object> exports) {
        NODE_SET_METHOD(exports, "hello", Method);
    }
    NODE_MODULE(NODE_GYP_MODULE_NAME, Initialize)

```

Python script for Document Processing

Python is a interpreted language and can be used from node application very easily using `require('child_process').spawn`

```
ops.controller.js ×
controllers > ops.controller.js > ...
const pdf = require('pdf-parse');

// in case you have not add python to env variable
const PYTHON_EXECUTABLE_PATH = config.PYTHON_EXECUTABLE_PATH;
const PYTHON_SCRIPT_PATH = path.join(__dirname, '../addons/python/pdf_scra
// runner for script
function run_script(executable, path, params) {
    return spawn(executable, [path, params]);
}


exports.docOpsResponse = (req, res, next) => {
    const FILENAME = req.params.filename;

    let json_data = [];
    /**
     * @params {string} python executable
     * @params {string} python script path
     * @params {string} doc path
     */
    const python = run_script(PYTHON_EXECUTABLE_PATH, PYTHON_SCRIPT_PATH,

    python.stdout.on('data', (data) => {
        json_data.push(data);
    });

    python.on('close', (code) => {
        console.log(json_data.join(''));
```

Also we had access to the parameter pass from node application using `import sys` and `Sys.arg[0]`. But we faced some issues in running the python from node application, we were getting data from the files but most of the time some error and we were not able to resolve the error. But the python script is working well locally. We can access the pdf and do operations on it locally.

Node for Document Processing

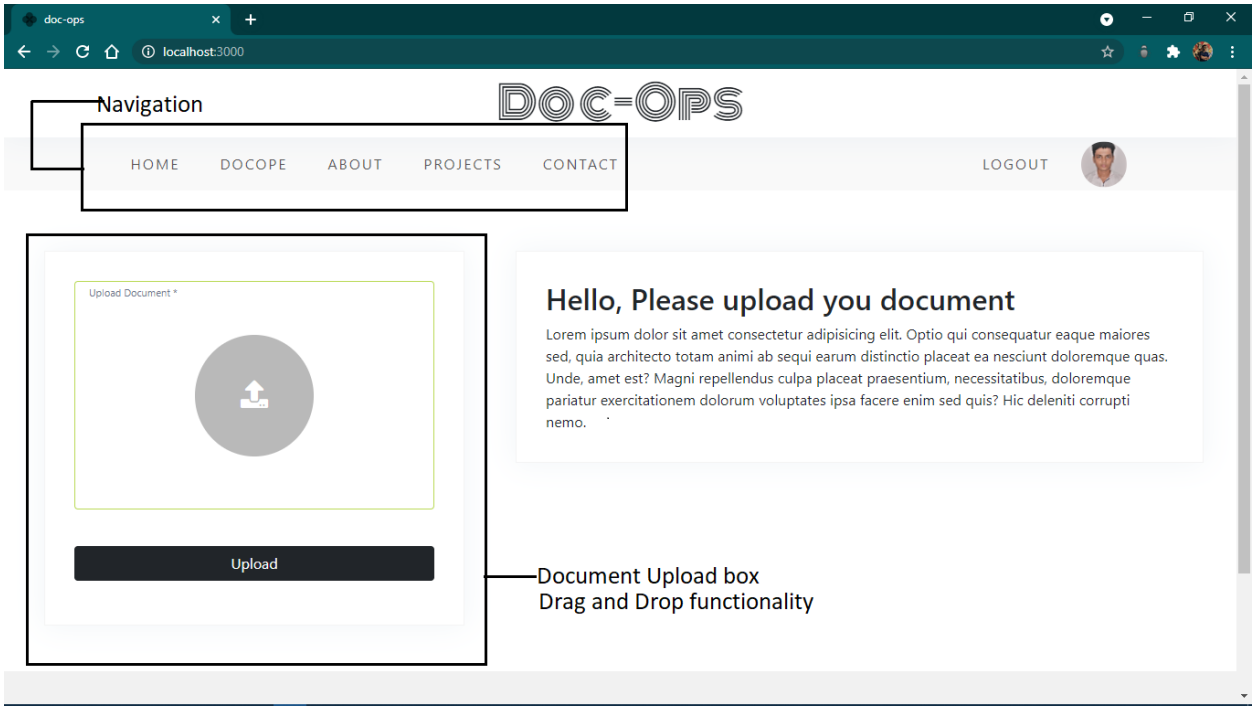
Node.js is an open-source, cross platform, back-end JavaScript runtime environment that runs on the V8 engine and is very powerful tool, so using node we parsed the document and extracted all the information from the data in the form of data steam /databuffer, In nodejs we parsed document and created Buffer

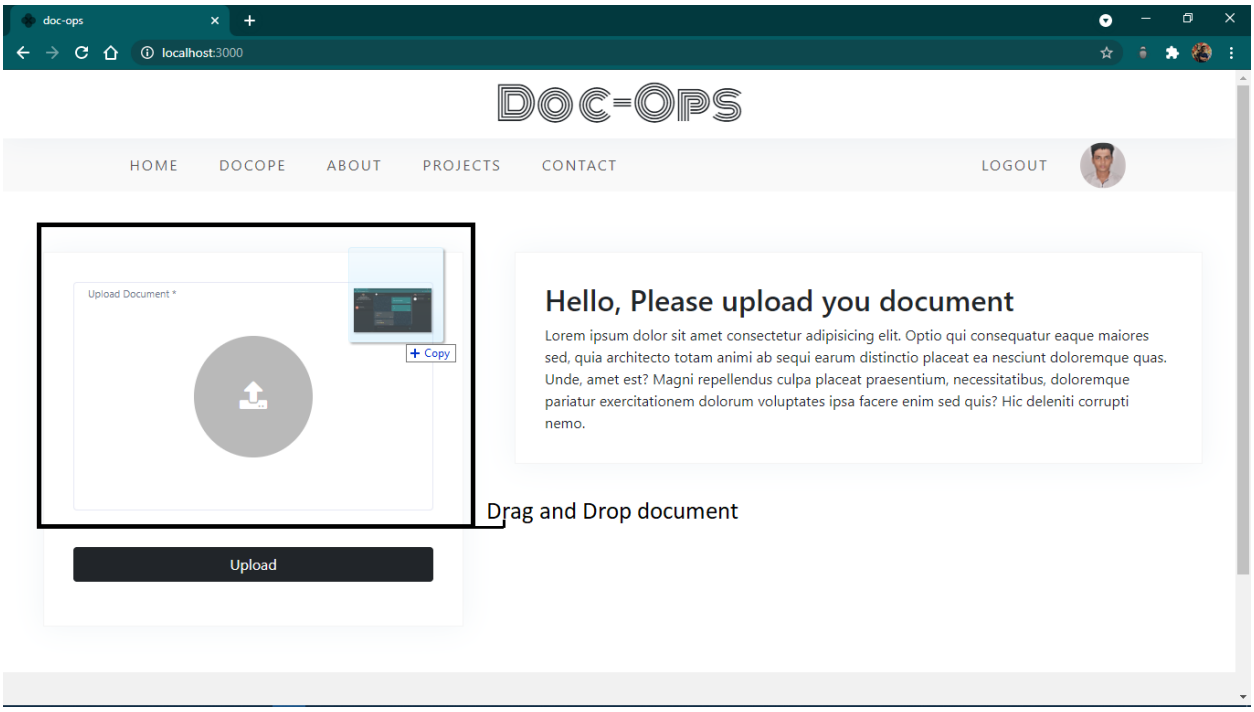
```
let dataBuffer = fs.readFileSync('path

pdf(dataBuffer).then(function(data) {
    // use data
})
.catch(function(error){
    // handle exceptions
})
```

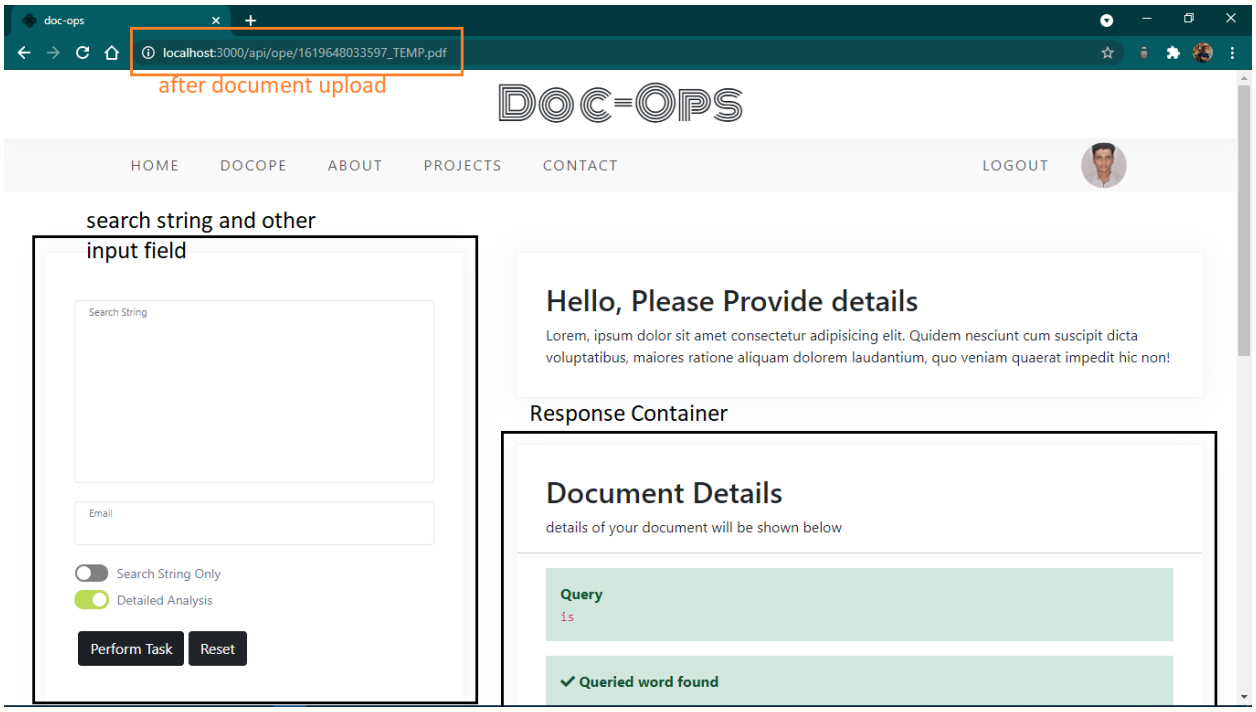
Result

- Below is the screenshot of working application
- Here we provide we an option to upload the document and you can also drago and drop you file to upload
- And hand full links in navigation

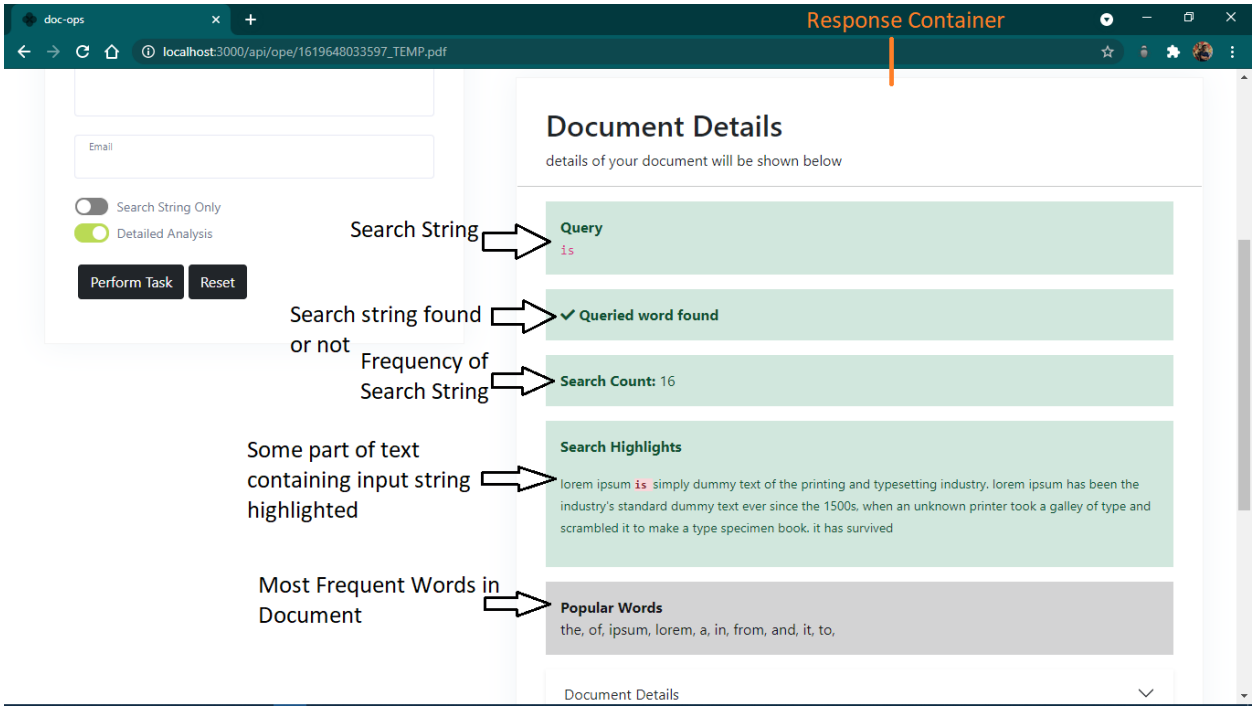




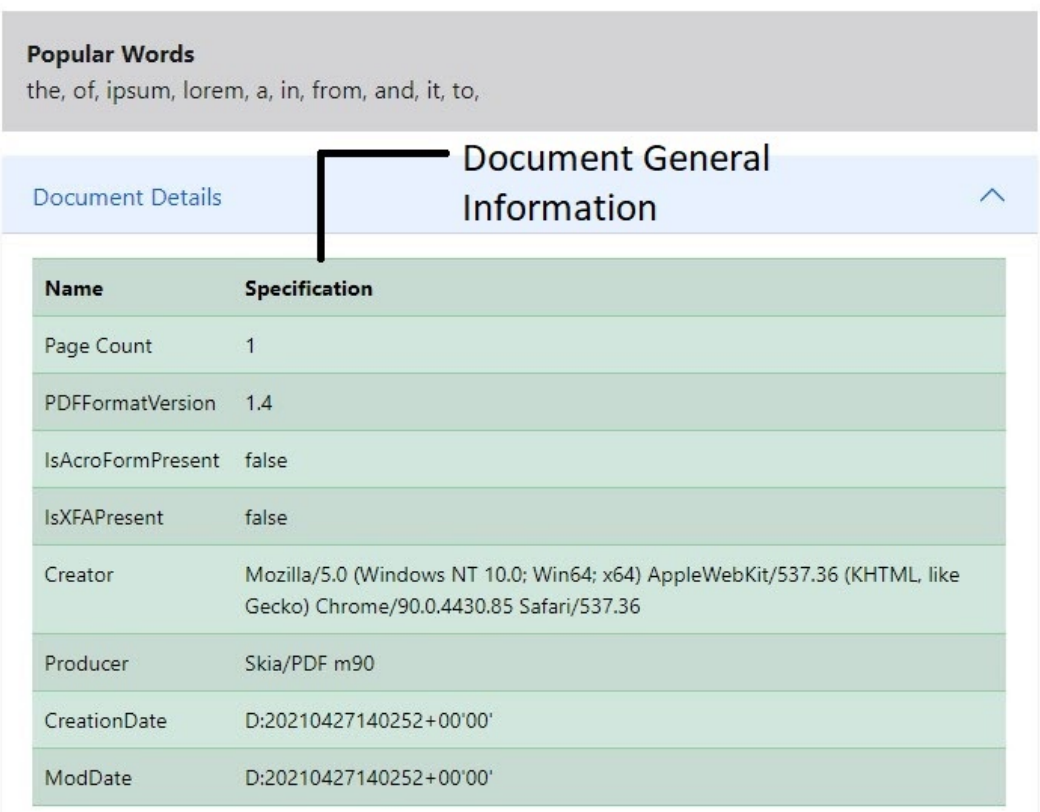
- ➔ The below screenshot is after we upload the document, we are redirect to this page were we can do as many query as we want, also we can see that after upload we add the file name in url params
- ➔ In right side we have the response container which basically hold all the prop of doc
- ➔ Like: **Query**, **Count for Input**, **Highlights**,



- Here we can see that we are getting some response back which includes **Query**, **Found**, **Count**, **Local_Text**, **info**, **popular_word**, ...etc
- Also it highlights the query in the complete texts to make it pop

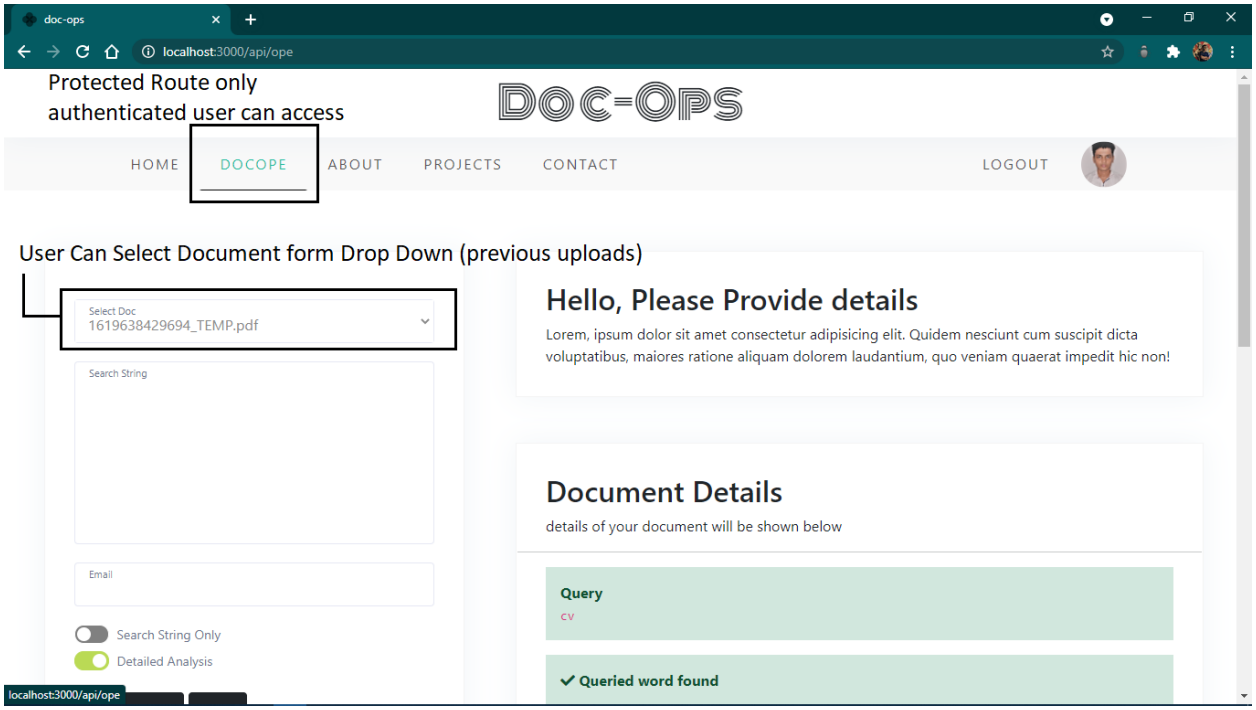


- We are also providing document general informations

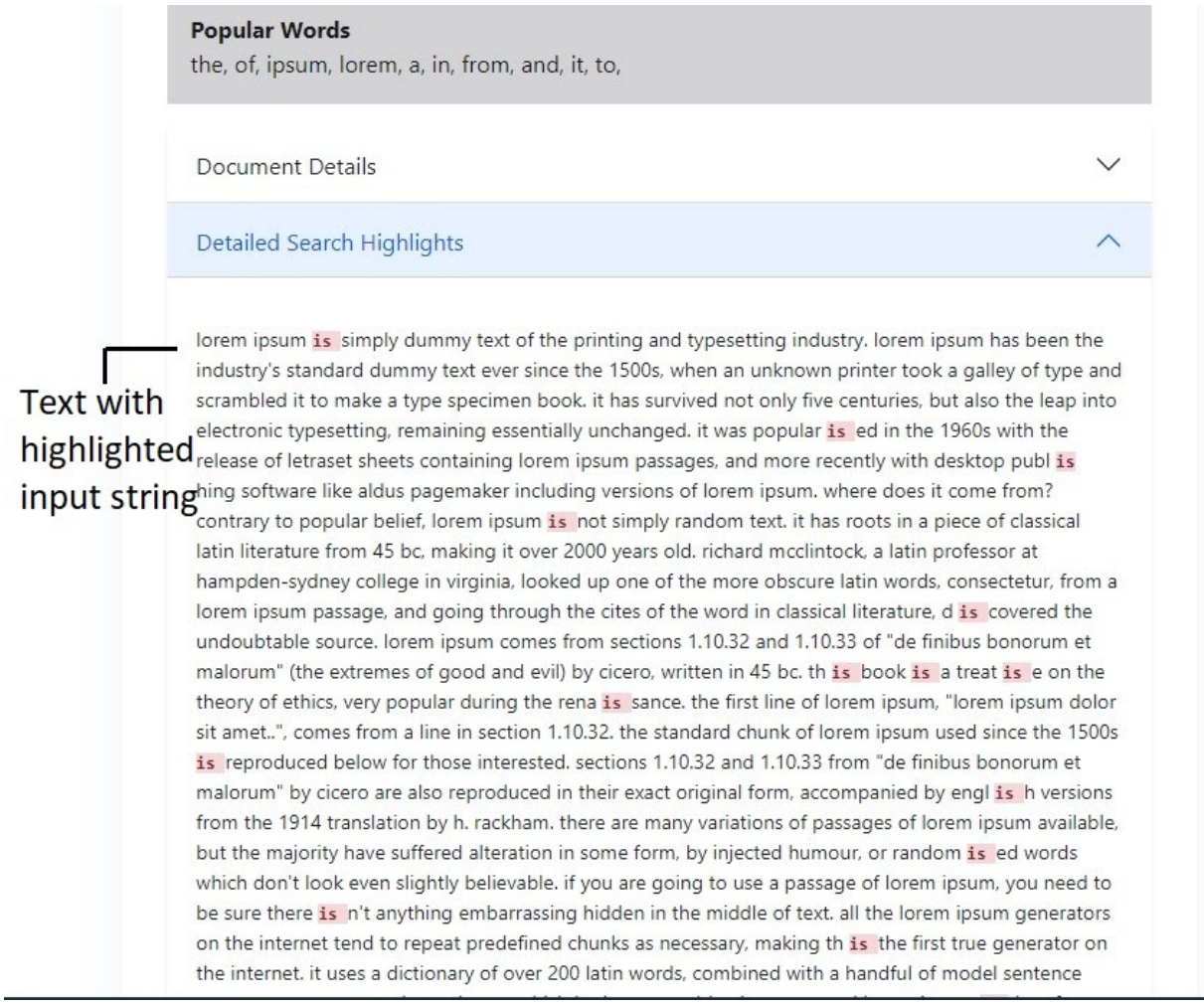


Name	Specification
Page Count	1
PDFFormatVersion	1.4
IsAcroFormPresent	false
IsXFAPresent	false
Creator	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85 Safari/537.36
Producer	Skia/PDF m90
CreationDate	D:20210427140252+00'00'
ModDate	D:20210427140252+00'00'

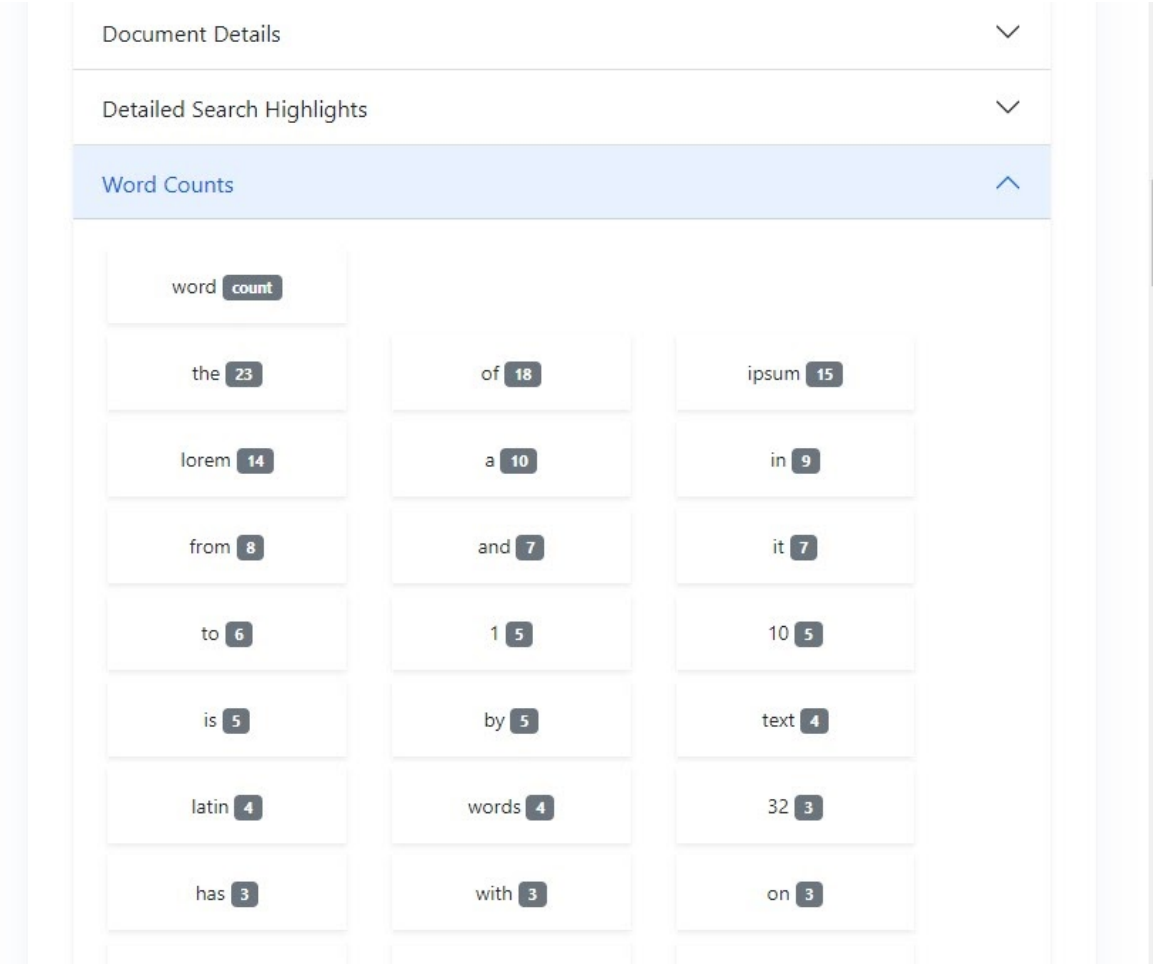
- ➔ `/appi/ope` is protected route so only an authenticated user can access it
- ➔ it shows the documents that he/she uploaded earlier and hence can simple select instead of uploading again and again



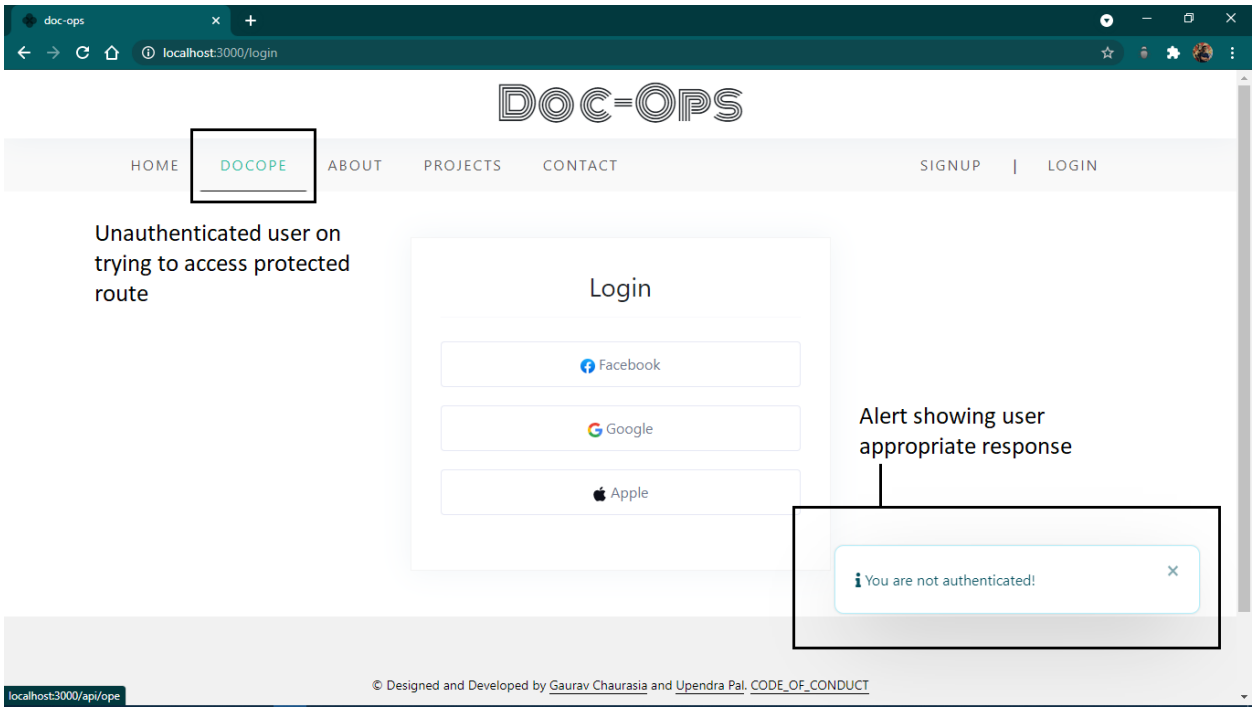
➔ In the below screenshot we highlight every occurrence of input string in the text extracted from document

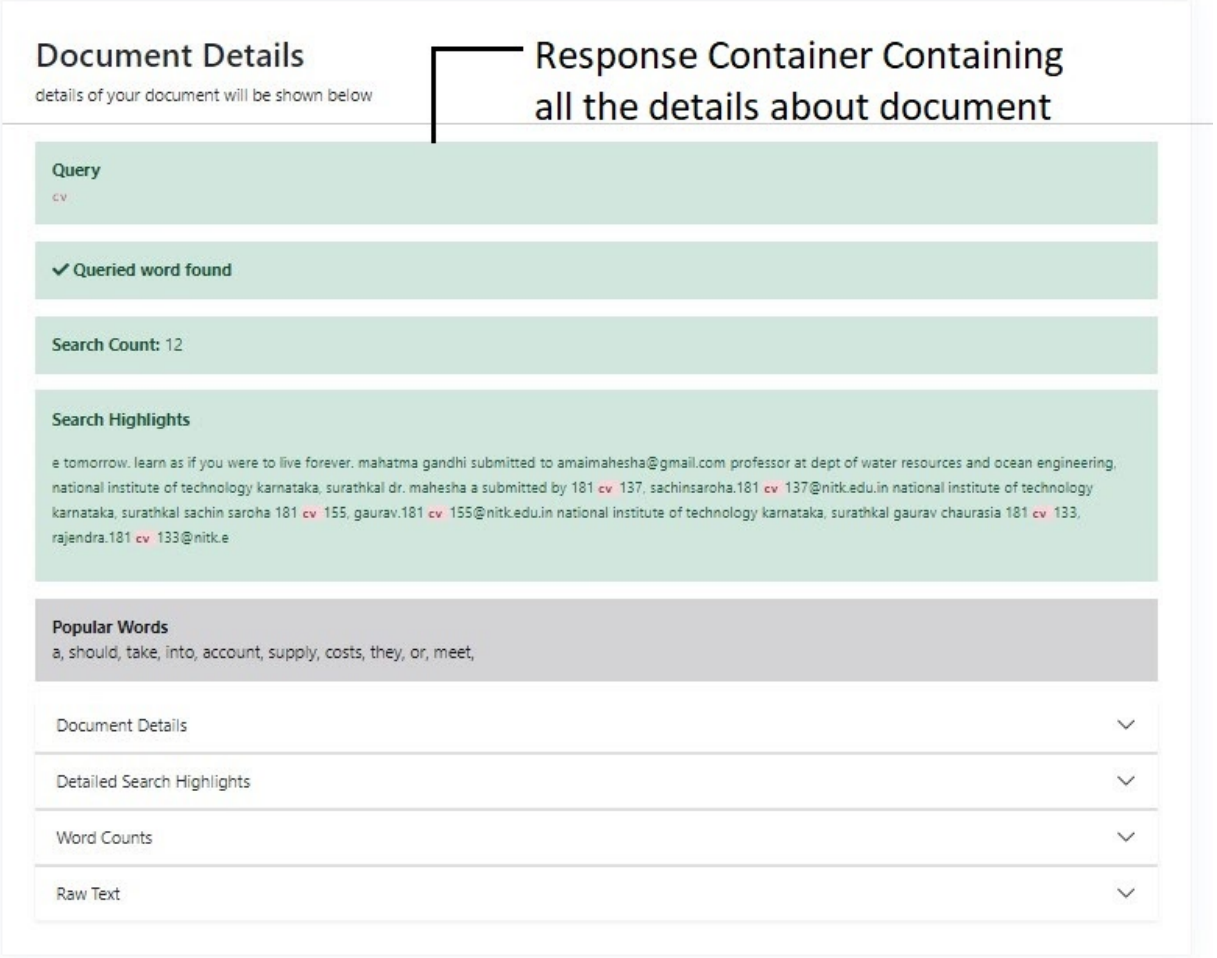


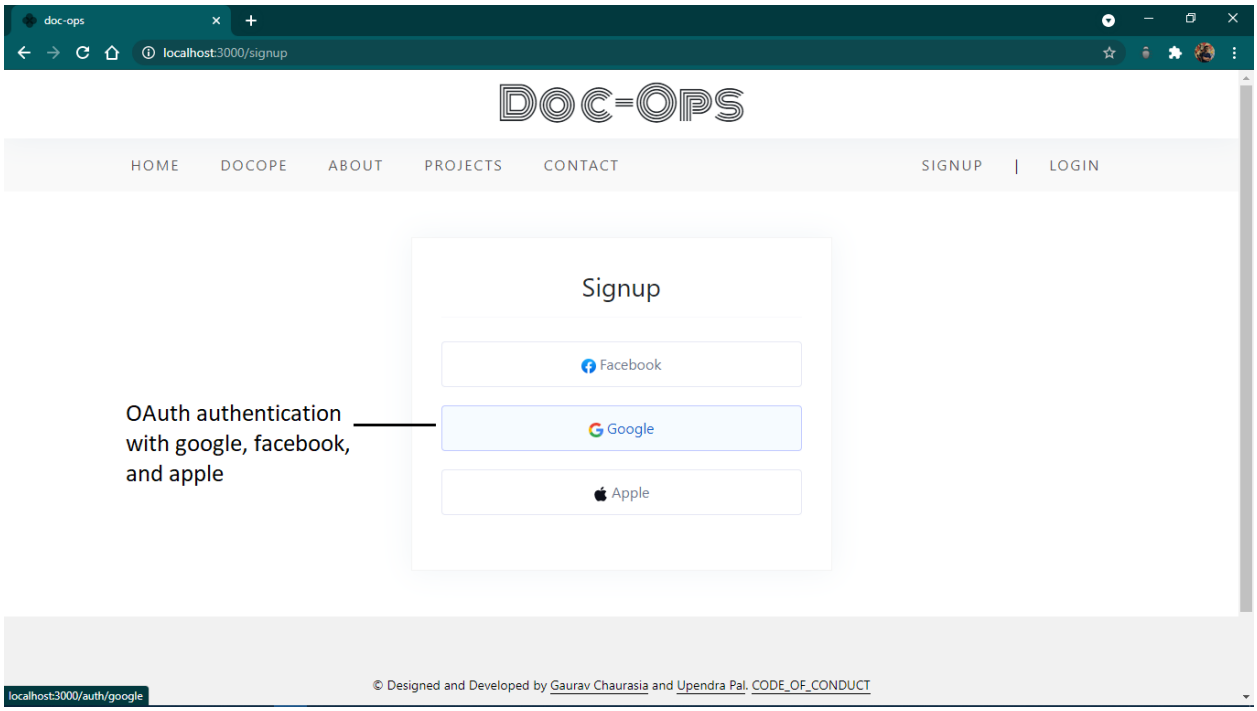
➔ We are also finding the occurence of words to get good insight on document

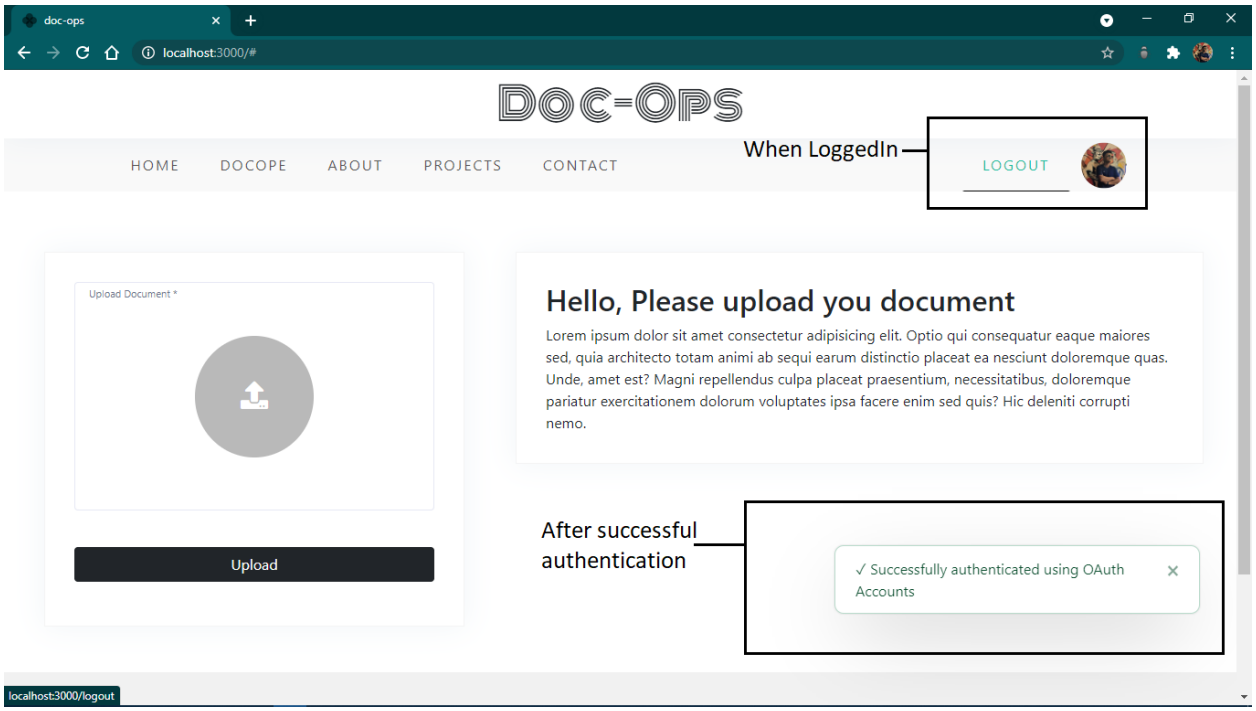


➔ `/appi/ope` is protected route so when a unauthenticated user tries to access the route is shows appropriate response to the user


doc-ops
localhost:3000/login
DOC-OPS
HOME
DOCOPE
ABOUT
PROJECTS
CONTACT
SIGNUP
LOGIN
Unauthenticated user on trying to access protected route
Login
Facebook
Google
Apple
Alert showing user appropriate response
You are not authenticated!
© Designed and Developed by Gaurav Chaurasia and Upendra Pal. CODE_OF_CONDUCT
localhost:3000/api/ope


Document Details
details of your document will be shown below
Response Container Containing all the details about document
Query
cv
Queried word found
Search Count: 12
Search Highlights
e tomorrow. learn as if you were to live forever. mahatma gandhi submitted to amaimahesha@gmail.com professor at dept of water resources and ocean engineering, national institute of technology karnataka, surathkal dr. mahesha a submitted by 181 cv 137, sachinsaroha.181 cv 137@nitk.edu.in national institute of technology karnataka, surathkal sachin saroha 181 cv 155, gaurav.181 cv 155@nitk.edu.in national institute of technology karnataka, surathkal gaurav chaurasia 181 cv 133, rajendra.181 cv 133@nitk.e
Popular Words
a, should, take, into, account, supply, costs, they, or, meet,
Document Details
Detailed Search Highlights
Word Counts
Raw Text


doc-ops
localhost:3000/signup
DOC-OPS
HOME
DOCOPE
ABOUT
PROJECTS
CONTACT
SIGNUP | LOGIN
Signup
Facebook
Google
Apple
OAuth authentication with google, facebook, and apple
© Designed and Developed by Gaurav Chaurasia and Upendra Pal. CODE_OF_CONDUCT
localhost:3000/auth/google


doc-ops
localhost:3000/#
DOC-OPS
HOME
DOCOPE
ABOUT
PROJECTS
CONTACT
When LoggedIn
LOGOUT
Upload Document *
Upload
Hello, Please upload you document
Lorem ipsum dolor sit amet consectetur adipisicing elit. Optio qui consequatur eaque maiores sed, quia architecto totam animi ab sequi earum distinctio placeat ea nesciunt doloremque quas. Unde, amet est? Magni repellendus culpa placeat praesentium, necessitatibus, doloremque pariatur exercitationem dolorum voluptates ipsa facere enim sed quis? Hic deleniti corrupti nemo.
After successful authentication
✓ Successfully authenticated using OAuth Accounts
localhost:3000/logout

Contribution

Upendra Pal

- Python script for document processing
- C++ program to search in sequential
- configure C++ addon for nodejs
- Sequential Search in JS
- Frontend Design, markup, styling, javascript

Gaurav Chaurasia

- Nodejs Application Setup
- Setting Basic routes
- Upload document functionality and data extraction from document
- configure C++ addon for nodejs
- Frontend Design, markup, styling, javascript

Similar Work

Python script for processing document

```
import PyPDF2
import re

filename = input("Enter filename:  ")

file = open(filename, 'rb')
reader = PyPDF2.PdfFileReader(file)
str = ""
# extracts texts from pdf for given page in range and adds them in string str
for i in range(0, reader.getNumPages()):
    str += reader.getPage(i).extractText()

print("Would you like to print the pdf file: ")
print("1. Yes\t2. No    ")
choice = int(input())
if(choice==1):
    print(str)
else:
    pass

str = str.lower()
findThis = input("Enter text to find in the pdf:  ").lower()

#program to search the pattern in pdf
matches = re.finditer(findThis, str)
matches_positions = [match.start() for match in matches]
# Get number of pages
NumPages = reader.getNumPages()
# Extract text and do the search
```

```
for i in range(0, NumPages):
    PageObj = reader.getPage(i)
    Text = PageObj.extractText()
    if re.search(findThis, Text):
        print(findThis, " found on page: ", i+1)

print("Position where ", findThis, " is found: ")
print(sorted(matches_positions))
print(findThis, " is found ", len(matches_positions), " times.")

regex = re.compile('[?/><.,@_!#$%^&*()-+=/{}]')
if regex.search(str)==None:
    print("No special character present in the pdf. ")
else:
    print("Special characters are present.")
```

Output

Screenshot - Result 1

```
"C:\Users\Upendra Pal\PycharmProjects\pythonProject1\venv\Scripts\python.exe" "C:/Users/Upendra Pal/PycharmProjects/pythonProject1/readpdf/search.py"
Enter filename:  sample.pdf
Would you like to print the pdf file:
1. Yes  2. No
2
Enter text to find in the pdf:  krishna
Position where  krishna  is found:
[23, 332, 345, 353, 640, 1115]
krishna  is found  6  times.
Special characters are present.

Process finished with exit code 0
```

Screenshot - Result 2

```
"C:\Users\Upendra Pal\PycharmProjects\pythonProject1\venv\Scripts\python.exe" "C:/Users/Upendra Pal/PycharmProjects/pythonProject1/readpdf/search.py"
Enter filename:  text2.pdf
Would you like to print the pdf file:
1. Yes  2. No
1
Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been
the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of


type and scrambled it to make a type specimen book. It has survived not only five centuries, but also
the leap into electronic typesetting,
remaining essentia
lly unchanged
. It
was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages,
and more recently with desktop publishing software like Aldus PageMaker including versions of
Lorem Ipsum. Where does it come from? Contrary

to popular belief, Lorem Ipsum is not simply
random text. It has roots in a


piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a
```

```
rator on the Internet. It uses a dictionary of over 200 Latin words, combined with a handful of
model sentence structures, to generate Lorem Ipsum which looks reasonable. The generated Lorem
Ipsum is therefore always free from repetition, injected humour,
or non
-
characteristic words et


Enter text to find in the pdf:  lorem
Position where  lorem  is found:
[0, 75, 457, 575, 643, 934, 1056, 1313, 1327, 1421, 1763, 1965, 2071, 2327, 2377]
lorem  is found  15  times.
Special characters are present.

Process finished with exit code 0
```