**SmartCampus-Search-Application**

**Introduction**

SmartCampus-Search-Application refers to the application of artificial intelligence (AI) and machine learning (ML) techniques to analyze, understand, and extract valuable insights from documents. In the context of document management, document intelligence plays a pivotal role in revolutionizing how organizations handle vast amounts of unstructured data contained within various document types.

In traditional document management systems, documents are typically stored as static files, making it challenging to unlock the wealth of information they contain. SmartCampus-Search-Application solutions address this challenge by employing advanced algorithms to automatically classify, extract, and organize data from documents, transforming them into actionable assets.

**Project scope and objectives**

The objective of this project is to design, develop, and implement a robust and efficient search engine for documents within our organization's document repository. The search engine will enable users to quickly and accurately retrieve relevant documents based on their content, metadata, and user-defined criteria. By implementing this search engine, we aim to enhance document discoverability, improve information retrieval efficiency, and facilitate knowledge sharing and collaboration across departments and teams.

**Methodology**

Document intelligence (DI) in SmartCampus-Search-Application utilizes machine learning (ML) to extract meaningful information from documents. One crucial step in this process is preparing the document for analysis by an IDP (Intelligent Document Processing) system. Here's a breakdown of the methodology for document preparation using document intelligence:

**1. Pre-processing:**

This stage focuses on making the document digitally digestible for the IDP system. Here's what it entails:

* **Format Conversion:** Transforming the document into a standardized format, typically PDFs or high-resolution images.
* **Image Enhancement:** Techniques like deskewing (straightening skewed scans) and noise reduction improve readability for Optical Character Recognition (OCR).
* **OCR:** Converting scanned text into a machine-readable format.
* **Layout Analysis:** Identifying the document structure, like headers, tables, and paragraphs.

**2. Content Classification:**

Here, the IDP system categorizes the document type based on pre-defined templates or machine learning models. This helps tailor the information extraction process for different document types like invoices, contracts, or emails.

**3. Data Extraction:**

Once the document is prepped and categorized, the IDP system locates and extracts specific data points. This can involve:

* **Template Matching:** For standardized documents, predefined templates guide data extraction from specific locations.
* **Form Understanding:** For semi-structured forms, AI recognizes form fields and extracts data accordingly.
* **Information Retrieval:** For unstructured documents, advanced NLP (Natural Language Processing) techniques are used to identify and extract relevant information based on context.

**4. Data Validation:**

Extracted data undergoes quality checks to ensure accuracy. This may involve:

* **Rule-based Validation:** Applying pre-defined rules to verify data based on format or expected values.
* **Human-in-the-Loop-Review:** Incorporating human oversight for complex documents or critical data points.

By following these steps, document intelligence prepares documents for efficient and accurate information extraction, making IDP systems a valuable tool for automating document processing tasks.

**Project Plan**

This project aims to develop a cloud-based document intelligence service that automates document preparation tasks. Here's a breakdown of the key phases:

**1. Define Requirements & Scope:**

* Identify target documents (invoices, contracts, etc.)
* Determine functionalities (data extraction, formatting, classification)
* Set accuracy and performance benchmarks
* Choose a cloud service provider (e.g., AWS, Azure, Google Cloud Platform)

**2. Data Acquisition & Preprocessing:**

* Collect a diverse set of sample documents
* Ensure data privacy and security compliance
* Preprocess data (cleaning, structuring, labeling)
* Split data into training, validation, and testing sets

**3. Model Development & Training:**

* Select appropriate AI models.
* Train models on the prepared dataset
* Continuously monitor and improve model performance through hyperparameter tuning and retraining

**4. API Development & Integration**

* Design and develop an API for document processing
* Integrate cloud service features (storage, security, scalability)
* Implement functionalities for user interaction (file upload, results download)
* Conduct through API testing for functionality and security

**5. Deployment & Monitoring**

* Deploy the document intelligence service to the chosen cloud platform
* Configure access controls and security measures
* Set up monitoring tools for performance and error tracking
* Conduct final user acceptance testing (UAT)

**6. Post-Launch & Maintenance (Ongoing):**

* Gather user feedback and iterate on functionalities
* Continuously retrain models with new data for improved accuracy
* Monitor resource usage and optimize cloud infrastructure
* Address security vulnerabilities and update software components

**Additional Considerations:**

**Security:** Implement robust security measures to protect sensitive data throughout the document processing pipeline.

**Scalability**: Design the service to handle increasing document volumes and user demands.

**User Interface:** Develop a user-friendly interface for easy interaction with the document intelligence service (optional).

**Tools & Technologies**:

Cloud service provider platform (e.g., Azure ML)

Cloud storage services (e.g., S3 buckets, Kendra, cli)

**Resource requirements:**

For document intelligence in a cloud service, you would typically require a combination of cloud-based services and libraries tailored for natural language processing (NLP), machine learning (ML), and document analysis. Here's a general outline of the resource requirements we used:

**Cloud Service Provider**: Choose a cloud service provider (e.g., AWS, Google Cloud Platform, Microsoft Azure) that offers services suitable for document intelligence tasks. Each provider offers various tools and services that can be leveraged for document processing and analysis.

**Document Storage**: Utilize cloud storage solutions for storing your documents securely. Options include Amazon S3, Google Cloud Storage, or Azure Blob Storage.

**Process:**

1. Creation of AWS root account.
2. Install AWS CLI in your system.
3. Configure it through the command prompt. That means connect your aws account with the command prompt or local host.
4. Creation of Users in the IAM.  
    1. Click on Create button   
    2. Give name  
    3. Click on Next button  
    4. Attach the Policy named “AdministatorAccess”.  
    5. Click on Create button
5. Creation of IAM Role.  
    1. Click on Create Button  
    2. In Services section select S3  
    3. Click on Next button  
    4. Select the polocies named “AdministratorAccess” , “S3FullAccess”, “KendraFullAccess”.   
    5. Click on Next button.  
    6. Give the role name.  
    7. The code should be changed in selected trust policy.  
     **Code:**

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": {

"Service": "kendra.amazonaws.com"

},

"Action": "sts:AssumeRole"

}

]

}

1. Click on your role.
2. Go to Trust Relationships and edit the code with the given above code and click on Update Policy.
3. Now create a folder named Document Intelligence.
4. Open command prompt from the created folder.
5. Install the boto3 and flask by using the commands

pip install boto3

pip install flask

1. Key Generation  
    1. Go to IAM account.  
    2. Click on Manage Access keys.  
    3. Click on Create Access Key.  
    4. Click on check box  
    5. Click on Create.
2. Copy the both keys which are generated “Secret key”, “Access Key”.
3. Create a S3 Bucket with a name and deselect the block all Public access.
4. Upload a file in the s3 bucket and make sure that file should be in the same folder.
5. Now go to the created IAM role and select the URL of the role(which is known as ARN).
6. Go to the folder and create a python file and write the following code.

**Code:**

import boto3

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_)

# Initialize the Boto3 client for Kendra

kendra\_client = boto3.client('kendra', region\_name='us-east-1')  # Replace with your AWS region

index\_id = "acd6e4ef-afe7-4fb2-8ec8-b5dcfe3de742"  # Replace with your Kendra index ID

def run\_kendra\_query(query\_text):

    try:

        response = kendra\_client.query(

            IndexId=index\_id,

            QueryText=query\_text

        )

        return response['ResultItems']

    except Exception as e:

        return f"An error occurred: {str(e)}"

def highlight\_search\_word(text, search\_word):

    # Highlight the search word in the text with a more generic approach

    return text.replace(search\_word, f"<mark>{search\_word}</mark>")

@app.route('/', methods=['GET', 'POST'])

def index():

    if request.method == 'POST':

        query\_text = request.form['query\_text']

        results = run\_kendra\_query(query\_text)

        formatted\_output = []

        for result in results:

            if 'DocumentExcerpt' in result:

                # Access the entire document text (if available)

                document\_text = result.get('DocumentAttributeFieldValues', {}).get('Content', '')

                if document\_text:

                    # Highlight the search word in the entire document text

                    highlighted\_text = highlight\_search\_word(document\_text, query\_text)

                    formatted\_output.append(highlighted\_text)

                else:

                    # Fallback to using the excerpt if document text is unavailable

                    excerpt = result['DocumentExcerpt']['Text']

                    formatted\_output.append(highlight\_search\_word(excerpt, query\_text))

        return render\_template('index.html', query=query\_text, output=formatted\_output)

    return render\_template('index.html', output=None)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, port=5000)

1. Now in the same folder create index.html file and write the following code.  
    **Code:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Intelligence Search</title>

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css"> <!-- Font Awesome CSS -->

    <style>

        /\* Reset CSS \*/

        \*, \*:after, \*:before {

            box-sizing: border-box;

            -moz-box-sizing: border-box;

            -webkit-box-sizing: border-box;

        }

        .cf:before,

        .cf:after {

            content: "";

            display: table;

        }

        .cf:after {

            clear: both;

        }

        /\* Body styles \*/

        body {

            background-image: url('C:/DocumentIntelligence/docs2.jpg'); /\* Updated image path \*/

            background-size: cover;

            background-repeat: no-repeat;

            background-color: #96ded1;

            color: #000; /\* Change text color to black \*/

            font: 20px/25px 'Times New Roman', Lucida, Arial, sans-serif;

            margin: 0;

            padding: 0;

        }

        a,

        a:visited {

            color: #000; /\* Change link color to black \*/

        }

        /\* Search Form Styles \*/

        .search-container {

            text-align: center;

            margin-top: 50px;

        }

        .searchform {

            background: #f4f4f4;

            background: rgba(244, 244, 244, .79);

            border: 1px solid #d3d3d3;

            padding: 2px 5px;

            width: 339px;

            box-shadow: 0 4px 9px rgba(0, 0, 0, .37);

            -moz-box-shadow: 0 4px 9px rgba(0, 0, 0, .37);

            -webkit-box-shadow: 0 4px 9px rgba(0, 0, 0, .37);

            border-radius: 10px;

            -moz-border-radius: 10px;

            -webkit-border-radius: 10px;

            display: inline-block;

        }

        .searchform input,

        .searchform button {

            float: left;

        }

        .searchform input {

            background: #fefefe;

            border: none;

            font: 12px/12px 'HelveticaNeue', Helvetica, Arial, sans-serif;

            margin-right: 5px;

            padding: 10px;

            width: 216px;

            box-shadow: 0 0 4px rgba(0, 0, 0, .4) inset, 1px 1px 1px rgba(255, 255, 255, .75);

            -moz-box-shadow: 0 0 4px rgba(0, 0, 0, .4) inset, 1px 1px 1px rgba(255, 255, 255, .75);

            -webkit-box-shadow: 0 0 4px rgba(0, 0, 0, .4) inset, 1px 1px 1px rgba(255, 255, 255, .75);

            border-radius: 9px;

            -moz-border-radius: 9px;

            -webkit-border-radius: 9px;

        }

        .searchform input:focus {

            outline: none;

            box-shadow: 0 0 4px #0d76be inset;

            -moz-box-shadow: 0 0 4px #0d76be inset;

            -webkit-box-shadow: 0 0 4px #0d76be inset;

        }

        .searchform input::-webkit-input-placeholder {

            font-style: italic;

            line-height: 15px;

        }

        .searchform input:-moz-placeholder {

            font-style: italic;

            line-height: 15px;

        }

        .searchform button {

            background: rgb(52, 173, 236);

            background: -moz-linear-gradient(top, rgba(52, 173, 236, 1) 0%, rgba(38, 145, 220, 1) 100%);

            background: -webkit-gradient(linear, left top, left bottom, color-stop(0%, rgba(52, 173, 236, 1)), color-stop(100%, rgba(38, 145, 220, 1)));

            background: -webkit-linear-gradient(top, rgba(52, 173, 236, 1) 0%, rgba(38, 145, 220, 1) 100%);

            background: -o-linear-gradient(top, rgba(52, 173, 236, 1) 0%, rgba(38, 145, 220, 1) 100%);

            background: -ms-linear-gradient(top, rgba(52, 173, 236, 1) 0%, rgba(38, 145, 220, 1) 100%);

            background: linear-gradient(to bottom, rgba(52, 173, 236, 1) 0%, rgba(38, 145, 220, 1) 100%);

            filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#34adec', endColorstr='#2691dc', GradientType=0);

            border: none;

            color: #fff;

            cursor: pointer;

            font: 13px/13px 'HelveticaNeue', Helvetica, Arial, sans-serif;

            padding: 10px;

            width: 106px;

            box-shadow: 0 0 2px #2692dd inset;

            -moz-box-shadow: 0 0 2px #2692dd inset;

            -webkit-box-shadow: 0 0 2px #2692dd inset;

            border-radius: 9px;

            -moz-border-radius: 9px;

            -webkit-border-radius: 9px;

        }

        .searchform button:hover {

            opacity: .9;

        }

    </style>

</head>

<body>

    <h1 style="text-align: center;">Document Intelligence</h1>

    <div class="search-container">

        <form class="searchform cf" method="POST">

            <input type="text" name="query\_text" placeholder="Enter your search query">

            <button type="submit">Search</button>

        </form>

    </div>

    <!-- Output results will be displayed here -->

    {% if output %}

        <h2>Results for "{{ query }}":</h2>

        <ul>

            {% for sentence in output %}

                <li>{{ sentence }}</li>

            {% endfor %}

        </ul>

    {% endif %}

</body>

</html>

1. Use the following commands.

**1. aws kendra create-index --name psindex --edition DEVELOPER\_EDITION --role-arn arn:aws:iam::759940468733:role/ps\_role**  
 Output: "Id": "**83ef9cd6-a571-41fa-8621-77a7c92a46e0**"  
Copy that id.

**2. aws kendra create-data-source --name "ps\_datasource" --index-id "83ef9cd6-a571-41fa-8621-77a7c92a46e0" --type "S3" --configuration "S3Configuration={BucketName=ps485861,InclusionPatterns=[\"\*.pdf\",\"\*.docx\"],ExclusionPatterns=[\"\*.txt\"]}" --role-arn "arn:aws:iam::759940468733:role/ps\_role"**

Output: "Id": "**646f1ea6-2883-482b-9efa-0365d28454cb**"

**3.** **aws kendra start-data-source-sync-job --index-id "83ef9cd6-a571-41fa-8621-77a7c92a46e0" --id "646f1ea6-2883-482b-9efa-0365d28454cb"**

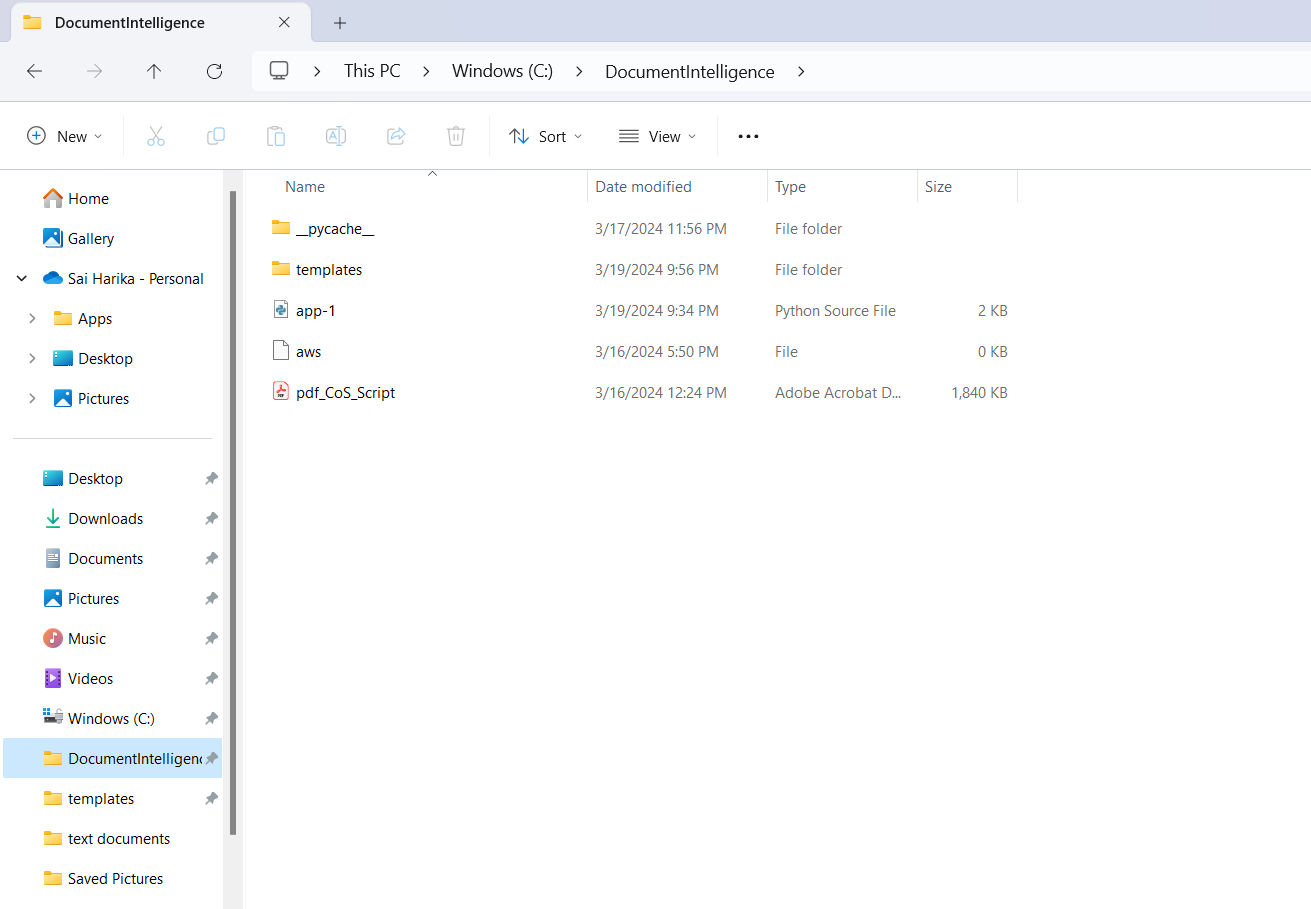
Output:

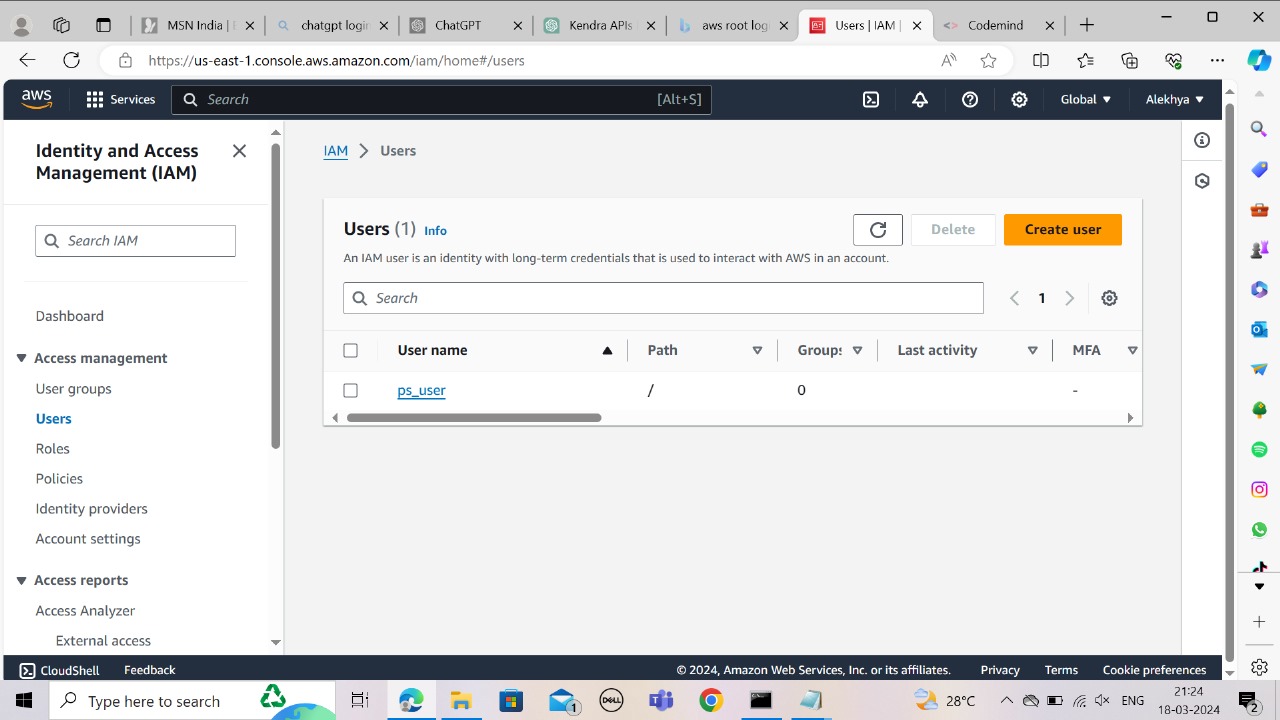
"ExecutionId": **"c7732f43-89d6-464f-aac7-9a2ca2394168". (**it will generate new execution id every time when we execute the above command).

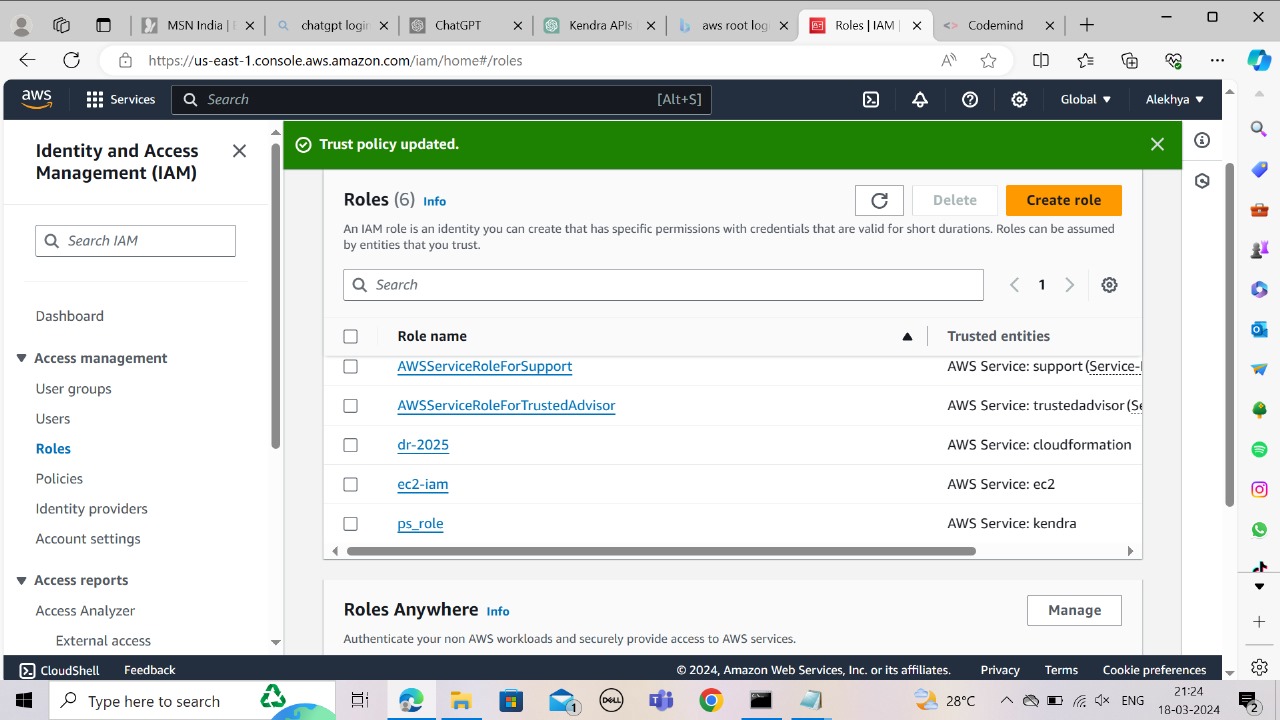
1. Now run in the command prompt using the following command.

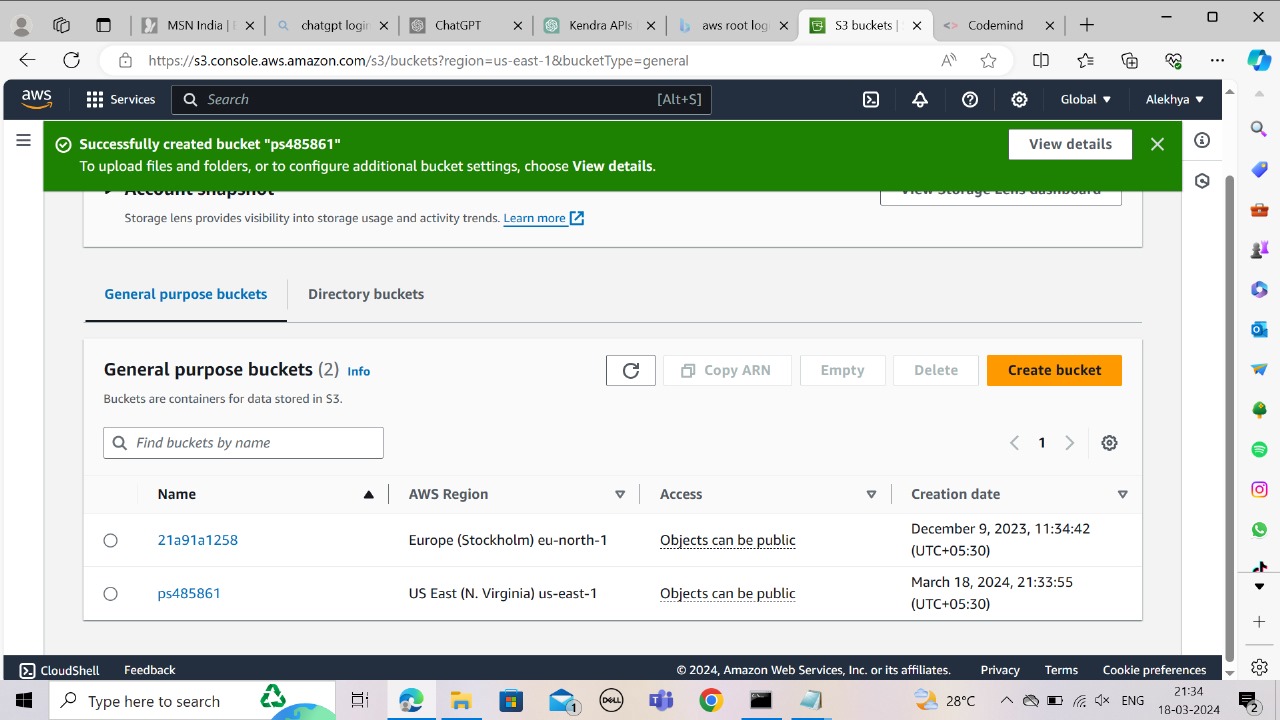
**Python app-1.py** (as app-1.py is file name).

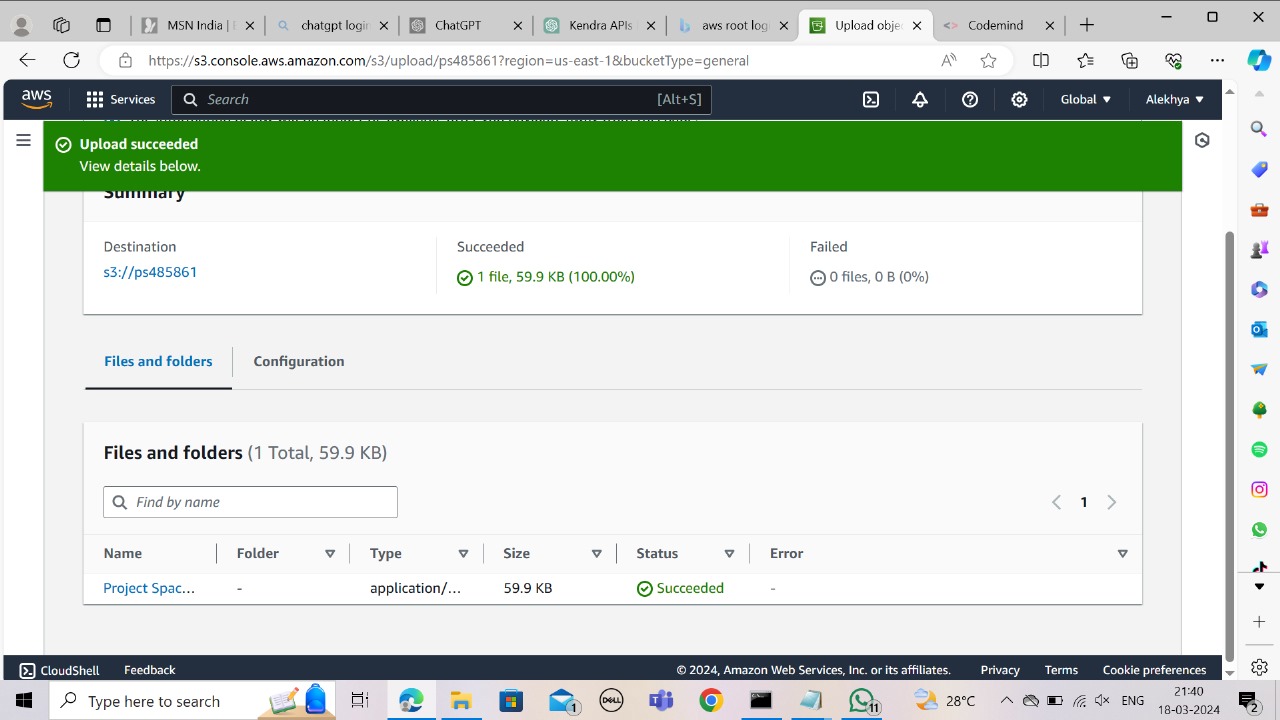
The output in the every step

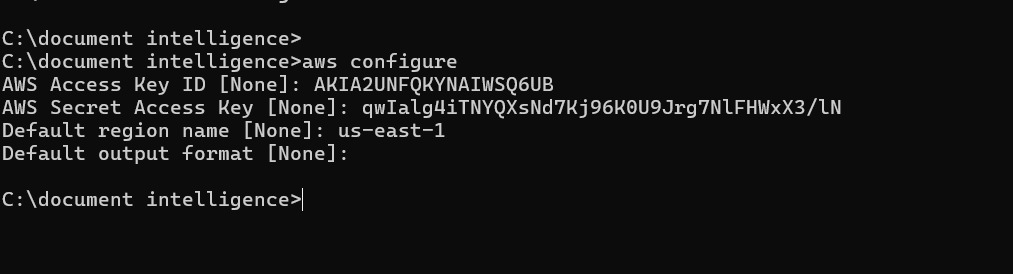


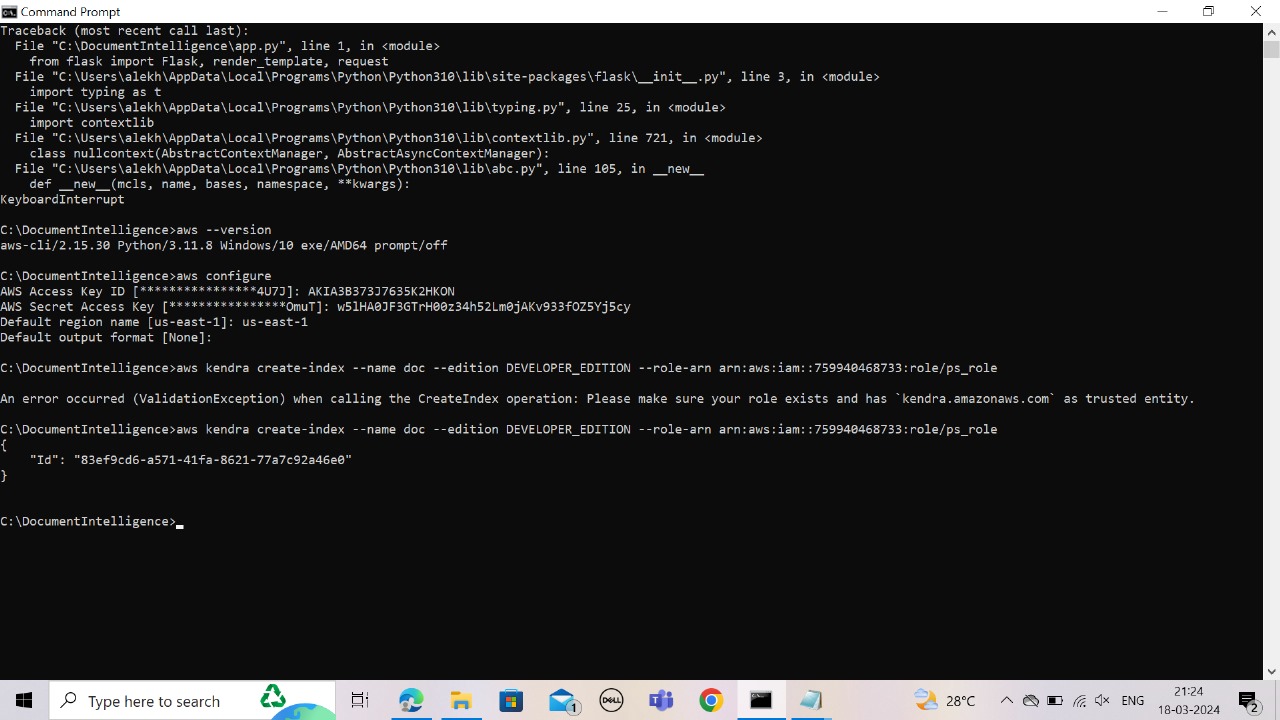


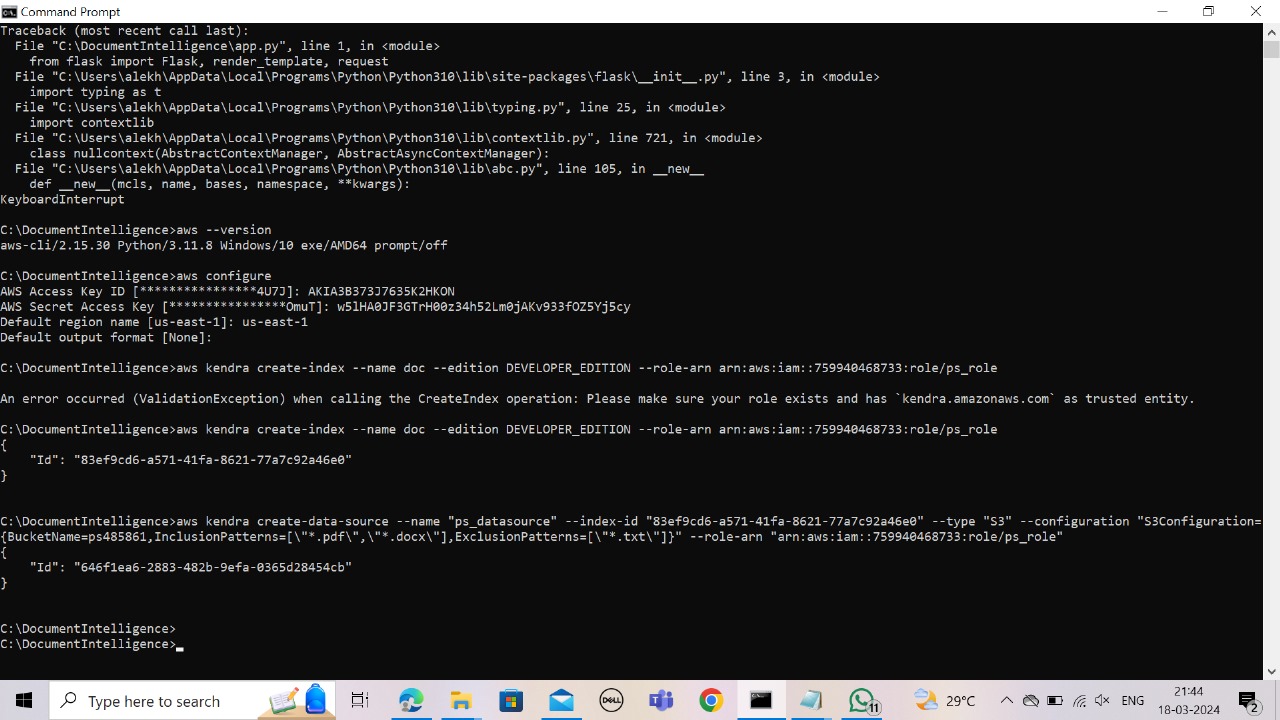
**

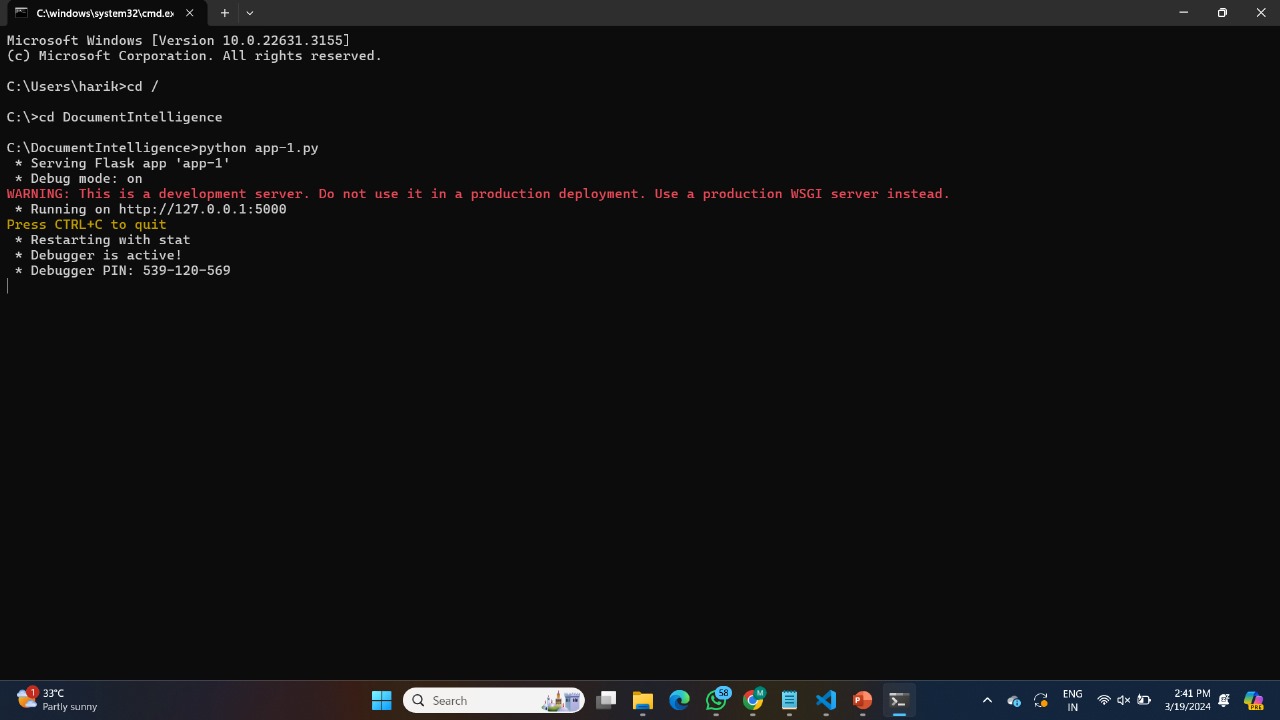


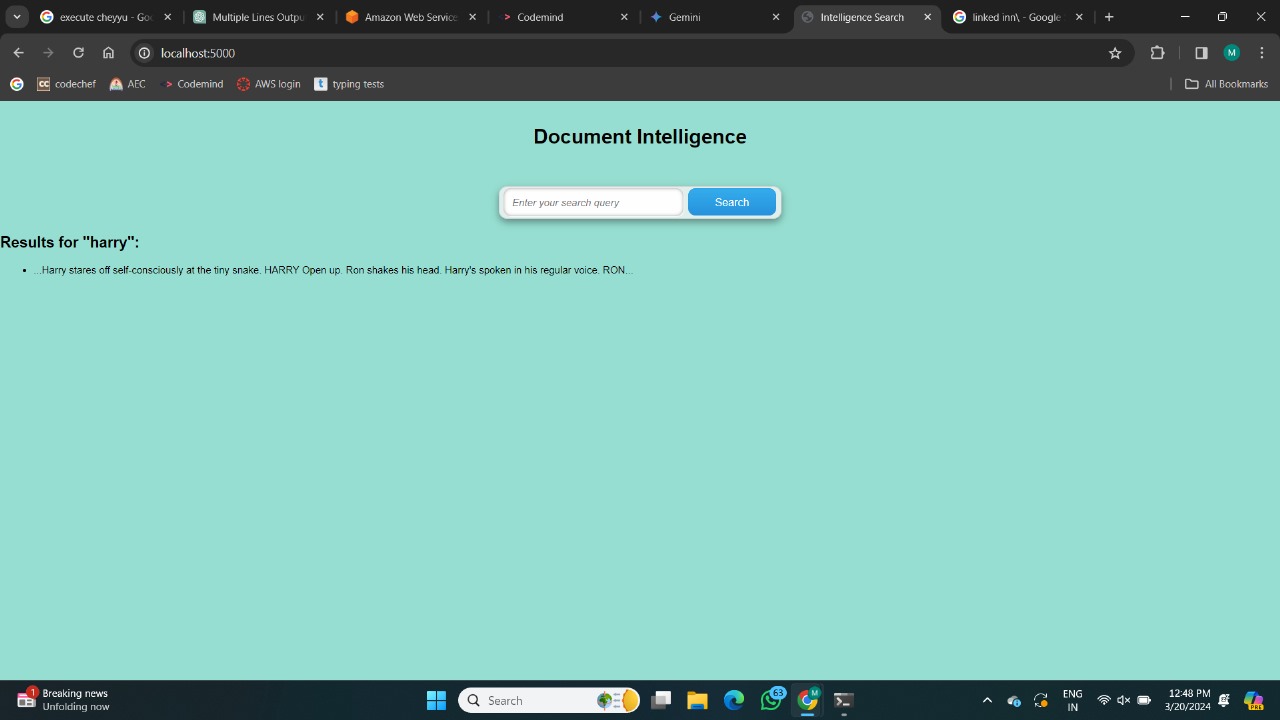


****





****

****

**Evaluation and Monitoring**

**Conclusion**

**References:**