# MEDICAL VIDEO AND DOCUMENT INDEXING SYSTEM

COMS 6998: CLOUD COMPUTING AND BIG DATA: FALL 2021

*Sakshi Arora (sa3871), Twisha Jain (tj2481), Deepak Dwarkanath (dd2676), Aayush Kumar Verma (av2955)*

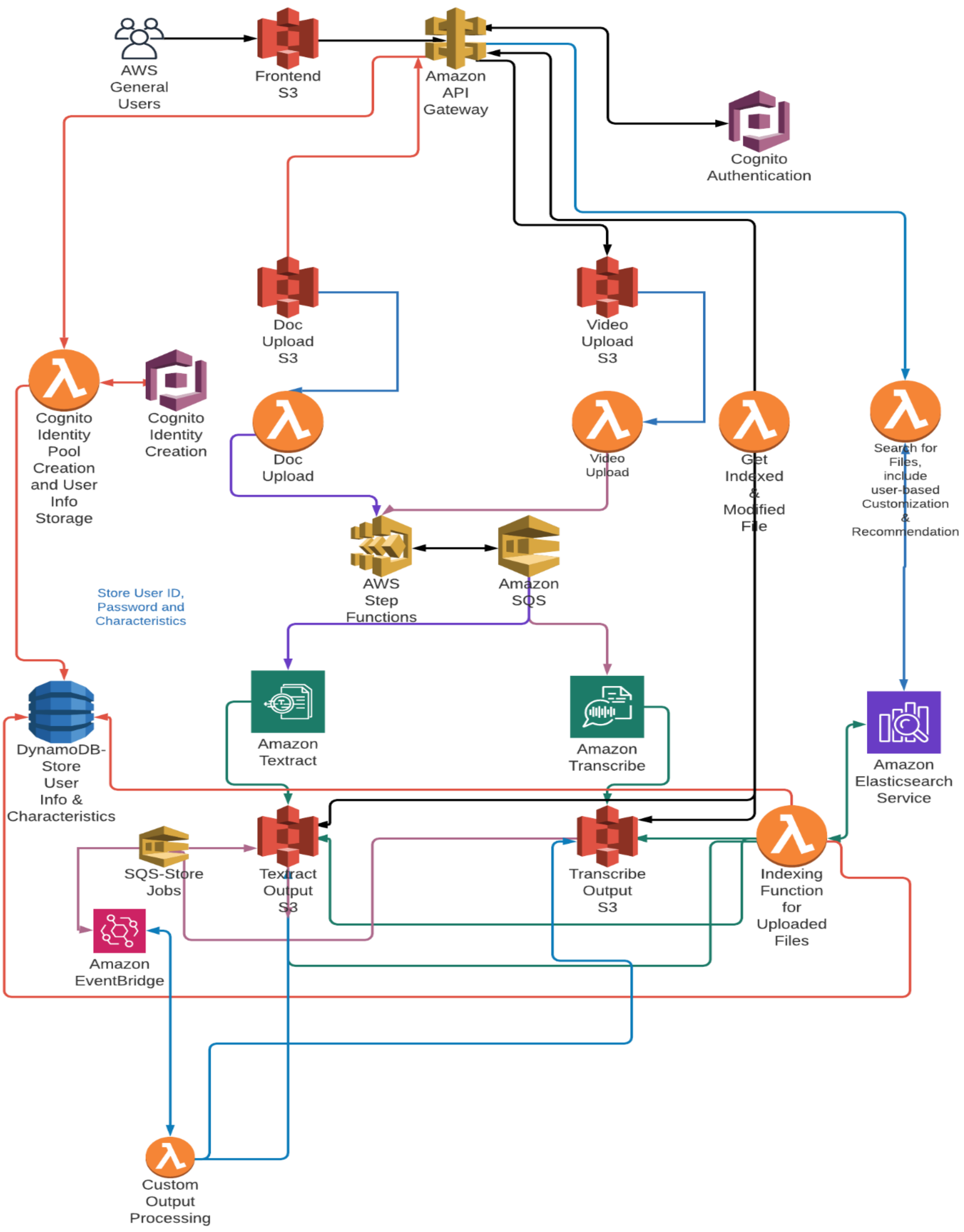
## Introduction

Medical journals are often dense in terms of information provided. Whether it is a standard textbook, research paper, or surgical videos, the content they contain is often thousands of pages/words long. In such a huge database of videos and documents, searching for any relevant video or document becomes a tedious task. With digitalization and exponentially growing repositories, the need for indexing and providing easy access on demand to the resources in these repositories becomes an important task. We envision a software that allows users to quickly search through the vast database and locate videos or documents of relevance to them based on key search words. Hence, we propose a system that indexes medical documents and videos by some key words which are identified in them so that the user when querying a particular keyword is redirected to all the relevant videos and documents.

## Problem Statement and Approach

Given the drastic increase in digital data, searching through it based on keywords and locating relevant videos and documents is a time consuming task. We propose a system which allows users to upload various videos and documents and upload them to the general medical repository which is common to all users. Our system interprets the data in the video or document and indexes it based on keywords found. This enables the users to later search through the entire repository providing keywords relevant to which they wish to see videos or documents. The entire repository is searched for relevant videos and documents and those are displayed to the user. This enables the creation of a common repository for the medical community where users can upload documents and videos and make them available to the entire community only a search away. The security of the system lies in the user signup and login which prevents malicious users from uploading data and provides future scope of blocking some spam users.

## Architecture and APIs


AWS General Users
Frontend S3
Amazon API Gateway
Cognito Authentication
Doc Upload S3
Video Upload S3
Cognito Identity Pool Creation and User Info Storage
Cognito Identity Creation
Doc Upload
Video Upload
Get Indexed & Modified File
Search for Files, include user-based Customization & Recommendation
AWS Step Functions
Amazon SQS
Store User ID, Password and Characteristics
DynamoDB-Store User Info & Characteristics
Amazon Textract
Amazon Transcribe
Amazon Elasticsearch Service
SQS-Store Jobs
Textract Output S3
Transcribe Output S3
Indexing Function for Uploaded Files
Amazon EventBridge
Custom Output Processing

The APIs used in the project include:

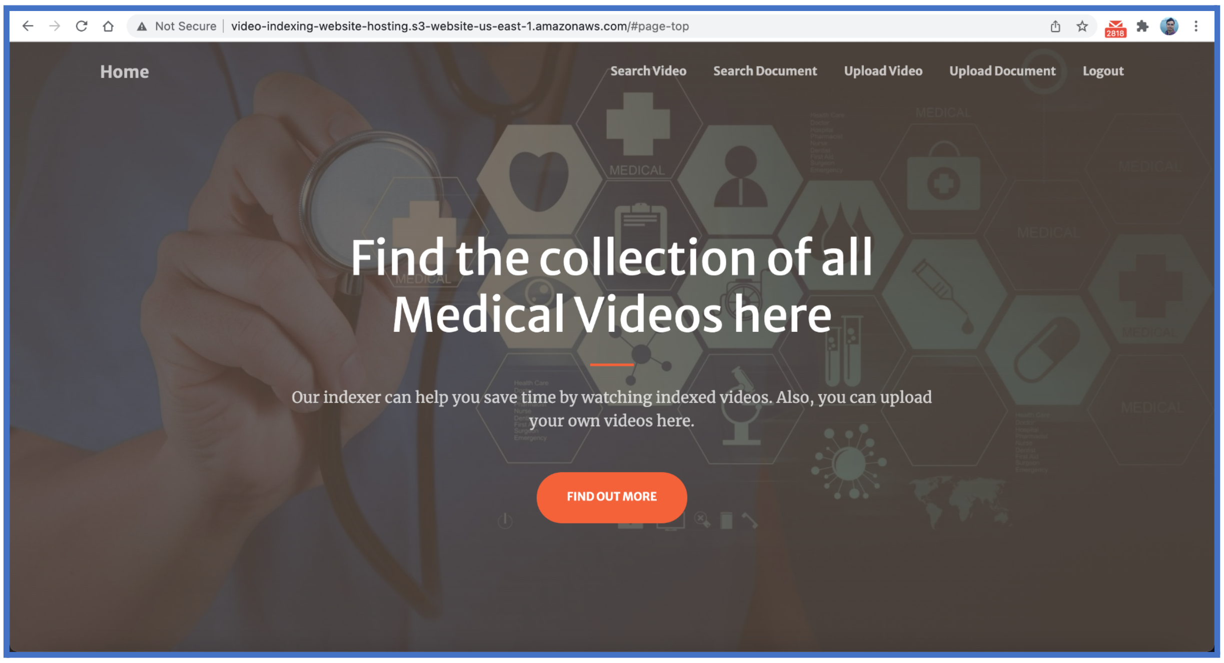
1. /search/documents – GET: This API is called when the user enters some keywords to be searched in the database. The API returns those documents which have been uploaded in the repository and are relevant to the search words entered by the user.
2. /search/videos – GET: This API is called when the user enters some keywords to be searched in the database. The API returns those videos which have been uploaded in the repository and are relevant to the search words entered by the user.
3. /login – GET: This API ensures only authenticated users are allowed to use the system once they login.
4. /signup – POST: This API allows users to sign up to our website using 2-factor authentication.

NOTE: We have implemented the upload functionalities such that the videos and documents get uploaded to S3 bucket directly using the AWS S3 API.

The data models are:

1. S3 – Videos and Documents: There are 2 S3 buckets, one for storing uploaded documents and the other for storing uploaded videos.
2. S3 – Textract and AWS Transcribe results: There are 2 S3 buckets, one for storing the Textract results and one for storing the Transcribe results.
3. S3 – Frontend Design: This S3 bucket hosts our frontend HTML pages and the landing pages can be invoked from here directly.
4. OpenSearch – Indexing: AWS Comprehend and Comprehend-Medical are called directly in Lambda functions to create a set of keywords for both images and videos, which are then passed to the OpenSearch indices for images and videos respectively, with the key:value stores as keyword set: Image/Video S3 URI.

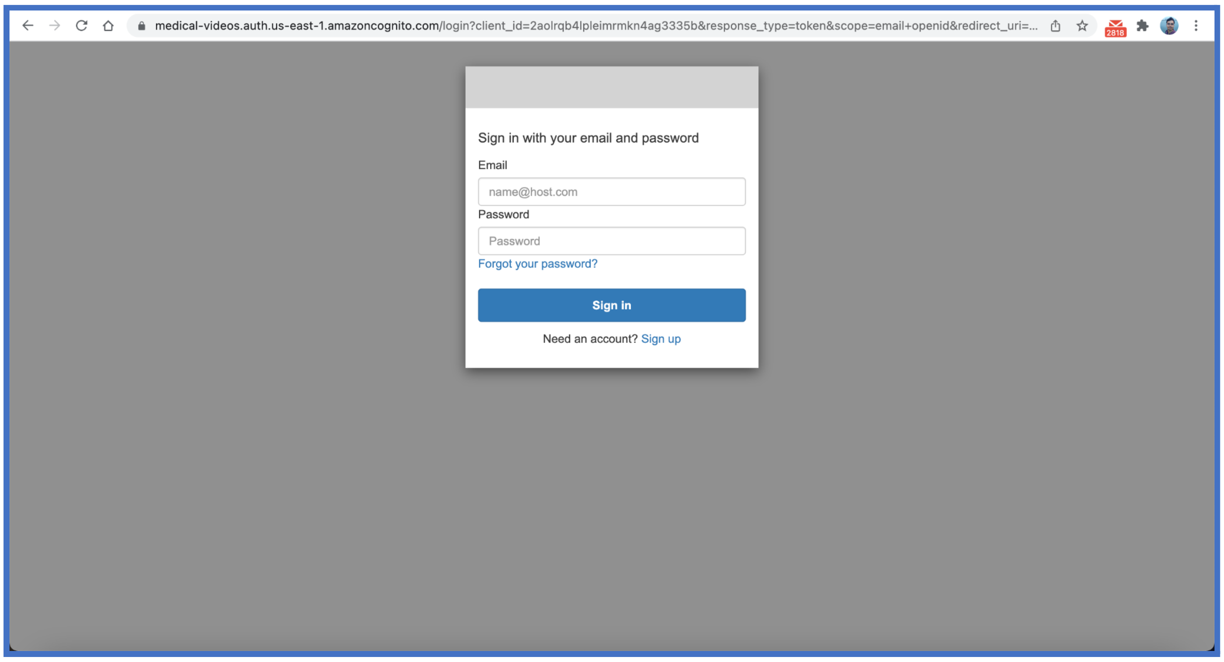
## Prototype


Not Secure | video-indexing-website-hosting.s3-website-us-east-1.amazonaws.com/#page-top
Home
Search Video
Search Document
Upload Video
Upload Document
Logout
FIND OUT MORE

Find the collection of all Medical Videos here

Our indexer can help you save time by watching indexed videos. Also, you can upload your own videos here.

**Image 1: Dashboard**


medical-videos.auth.us-east-1.amazoncognito.com/login?client_id=2aolrqb4lpleimrmkn4ag3335b&response_type=token&scope=email+openid&redirect_uri=...
Email
name@host.com
Password
Password
Forgot your password?
Sign in

Sign in with your email and password

Need an account? Sign up

**Image 2: Login/Signup**

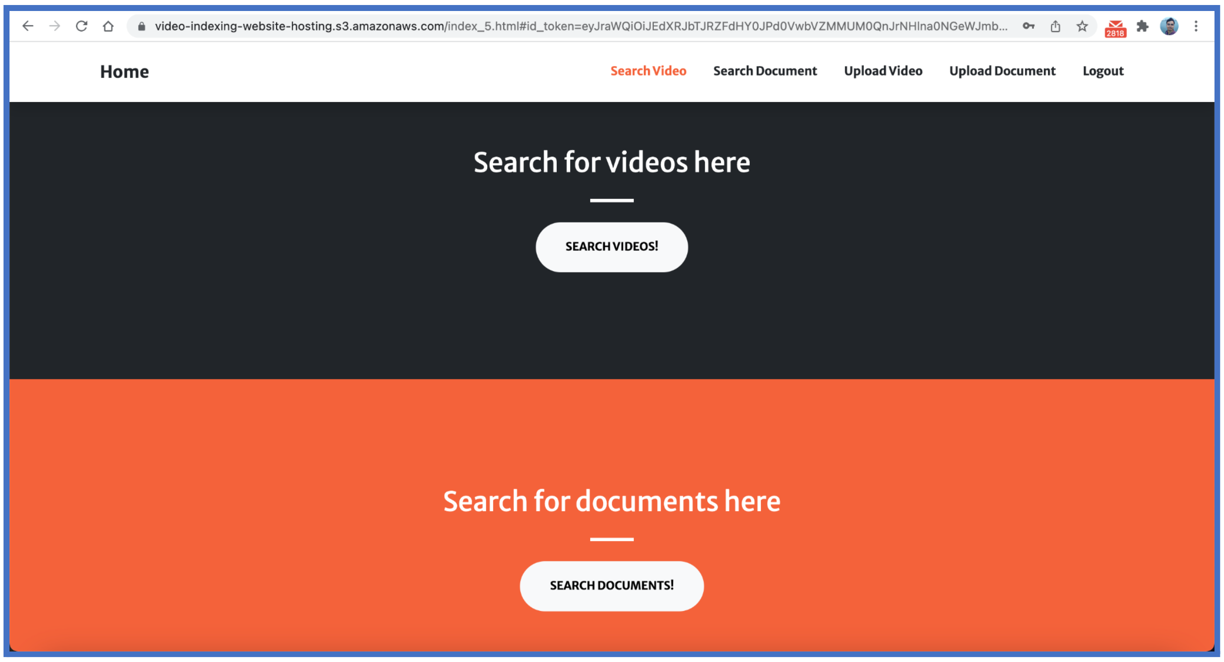

video-indexing-website-hosting.s3.amazonaws.com/index_5.html#id_token=eyJraWQiOiJEdXRJbTJRZFdHY0JPd0VwbVZMMUM0QnJrNHlna0NGeWJmb...
Home
Search Video
Search Document
Upload Video
Upload Document
Logout
Search for videos here
SEARCH VIDEOS!
Search for documents here
SEARCH DOCUMENTS!

**Image 3: Video and Document Search**

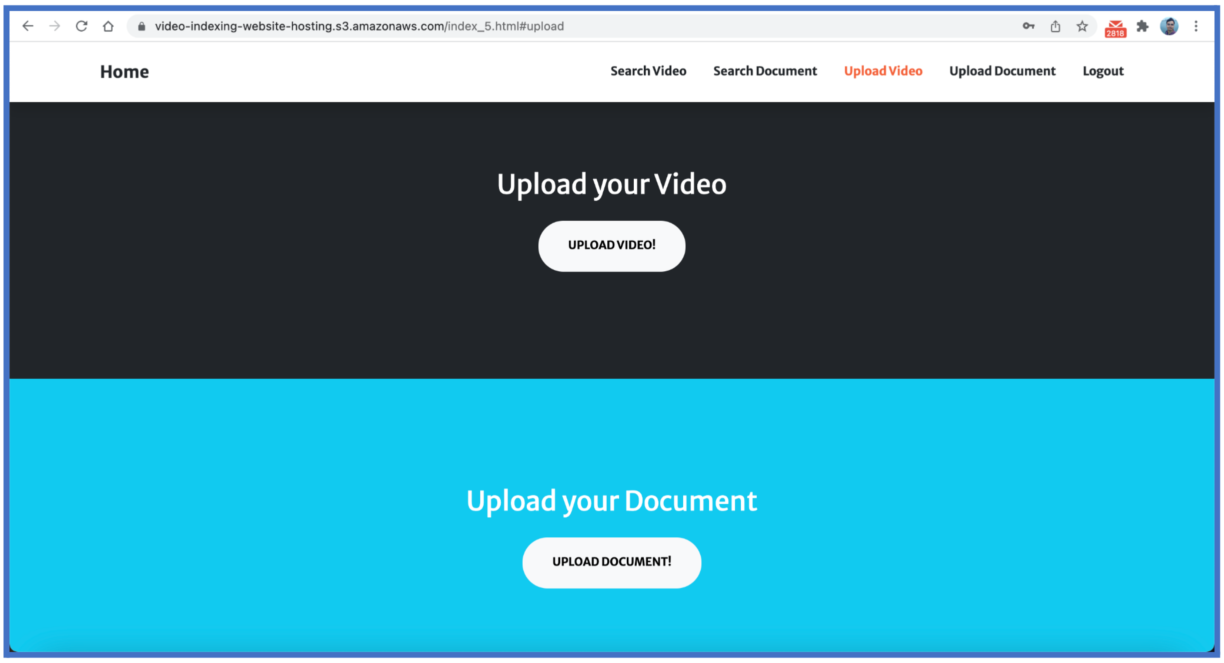


**Image 4: Video and Document Upload**

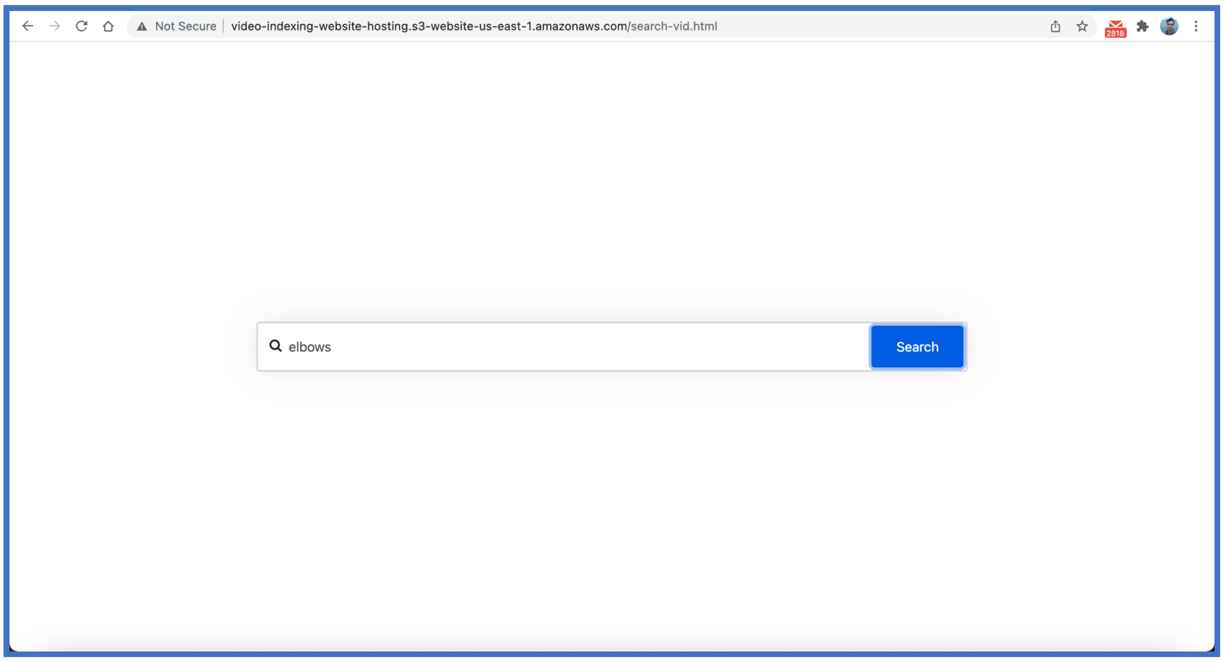

Not Secure
video-indexing-website-hosting.s3-website-us-east-1.amazonaws.com/search-vid.html
elbows
Search

**Image 5: Video and Document Search**

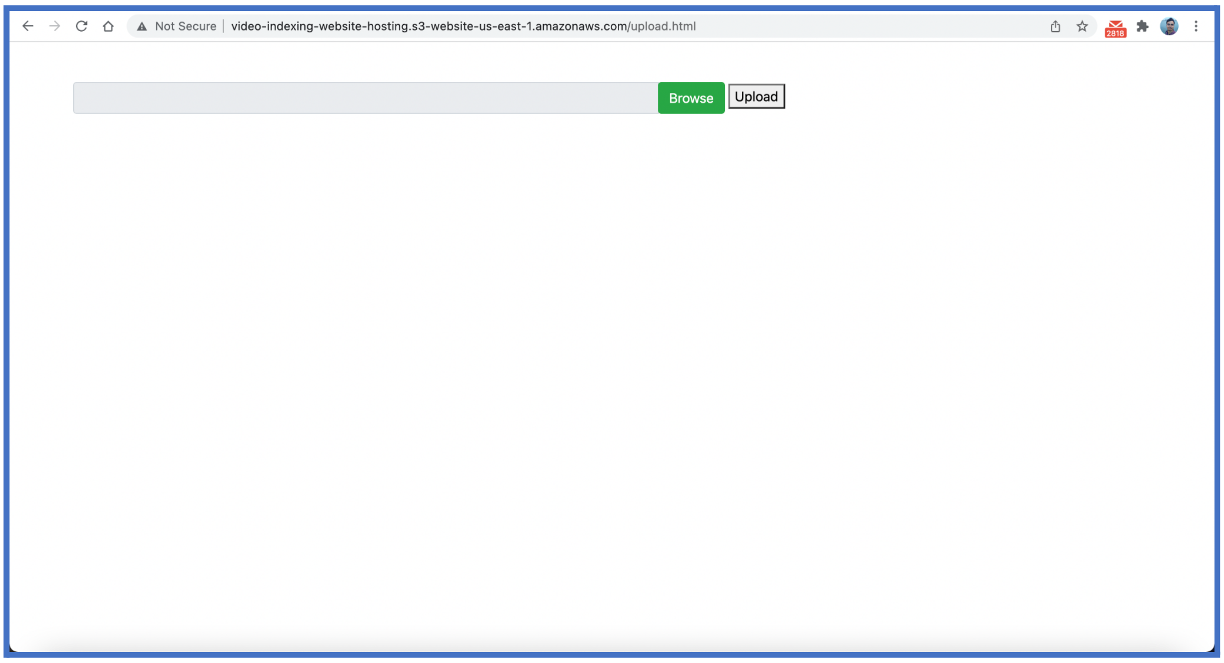

Not Secure
video-indexing-website-hosting.s3-website-us-east-1.amazonaws.com/upload.html
Browse
Upload

**Image 6: File Upload**

## Design Details and Code Structure

The flow of the project is as follows:

1. The user lands on the dashboard page.
2. The user has the option to either sign up or log in. Sign-ups require email verification through OTP.
3. Once the user has signed up or logged in, he/she can choose to upload a video, upload a document, search the video repository or search the document repository.
4. If the user selects to upload a video or a document, they are redirected to the respective upload pages where they can select the file and click upload. The system prompts the user to inform whether the upload is successful or not. On uploading, the software indexes the uploaded video using its transcription or the document by applying OCR and fetches keywords relevant to them using Amazon's comprehend which uses NLP to identify important entities from the text. The system further employs comprehend-medical to identify entities relevant from a medical standpoint.
5. If the user selects the option to search videos or documents, the user is redirected to the appropriate search page where they may enter some keywords. The software searches for videos/documents indexed with those keywords and displays them to the user.

   The code structure is as follows:

   a. **Signup, Login & Logout**

      First, the frontend UI was created using HTML, CSS, JavaScript, jQuery, and Bootstrap. This was the landing page of our website. Along with this, three other HTML pages were created for uploading videos and documents, searching and displaying videos, and searching and displaying documents.

      The signup, login, and logout features were implemented on the landing page with the help of Cognito in the backend. Every new user must first sign up to our website using a verifiable email ID and a strong password containing at least 1 uppercase letter, 1 lowercase letter, 1 number, 1 special character, and a total of at least 8 characters. They also get a verification code in their email and after passing the verification, they are logged in to our website.

   b. **Video Upload**

      The user can upload a video directly to the S3 bucket called 'project-temp-bucket'. Once the video is uploaded, the lambda function 'transcription-job-creator' is triggered which runs an asynchronous transcription job using AWS Transcribe on the video uploaded and then saves the result from the job to an S3 bucket called 'project-transcription-outputs'.

## c. Video Transcribe

Transcription of the video involved using Lambda functions that would trigger when video files were uploaded to an S3 bucket (bucket name: "project-temp-bucket"). The Lambda function was called "transcription-job-creator", and it used the Python boto3 API client to call various AWS services. The S3-PUT event would trigger "transcription-job-creator", where it would store the video file URI (its S3 bucket location), and create an AWS Transcribe job. This involves taking an audio or video file (.mp4 format in our case, also .wav, .mov), passing its S3 URI to Transcribe, which would create a transcript of the words it interpreted as being spoken in the file, as well as produce token/confidence score/timestamp tuples, all in JSON format. The end result JSON file would be stored in another S3 bucket, "project-transcription-outputs", where it would be called by other components of the project, specifically the indexing and search capabilities.

## d. Video Indexing and Search

Video Indexing involves creating a set of keywords that would be associated with each video file, and the keywords are searchable using AWS OpenSearch (which would return all video files associated with a keyword or set of keywords). The indexing process is initiated in the following sequence:

1. S3 bucket "project-transcription-outputs" triggers a Lambda function, "project-video-indexing", whenever a transcription job JSON is put into it. From this, ultimately it will create a set of keywords, starting with all the words in the transcript and get pared down significantly.
2. "project-video-indexing" will retrieve the transcript from the associated JSON transcription (but not the timestamps, which aren't required for this particular function).
3. At this point, we call AWS Comprehend, a service which can return key phrases and named entities, and AWS Comprehend-Medical, which returns identified medical terminology from the transcript. Both of these services will be called within the Lambda using a boto3 API client.
4. For AWS Comprehend, we first return a list of key phrases. We then go through each phrase and remove stopwords (commonly used English words that provide little context to a document), by checking if words within the key phrase are within a set of stopwords. We then add all distinct words from the key phrases set to the end keyword set/list.
5. We then use AWS Comprehend to return named entities (from the original transcript). Named entities are words that Transcribe identifies as names, locations, organizations, dates, titles, product names, etc. We will add all distinct named entities to the end keyword set/list.
6. Finally we will call AWS Comprehend-Medical, a unique medical extension of Comprehend, which can return medical terminology from

a text. We pass our Transcribe transcript and pull all medical terms, and add them to the end keyword set/list.

7. Finally we eliminate all redundant words in the final keyword set/list, and then call OpenSearch. The OpenSearch domain index we use is "https://search-photos-no-vpc-wnawrz54b6kh4bqyav7idotsdu.us-east-1.es.amazonaws.com/project_video/_doc" (We reused an old domain to save on costs). We will pass the final keyword set as labels (keys), and the URI of the original video file as the associated keyword value, as a POST request.

The search process itself is called on the front-end, from https://video-indexing-website-hosting.s3.amazonaws.com/search-vid.html, as follows:

1. https://qjuimep2u6.execute-api.us-east-1.amazonaws.com/test23 is the API called to search the videos using argument '../searchvid?q='+x ', 'method: GET', where 'x' is the entered keyword(s).
2. The API gateway 'searchvid/GET/.' HTTP uses a Lambda function, 'search-video' for its backend Integration Request.
3. 'search-video' takes the keywords as a string, separates them, and calls the OpenSearch index as before, except using a GET request (https://search-photos-no-vpc-wnawrz54b6kh4bqyav7idotsdu.us-east-1.es.amazonaws.com/project_video/_doc).
4. The GET request should return a list of 'hits', or successful URI matches for the keyword. The URI should point to the original video S3 bucket 'project-temp-bucket'.
5. Results should be returned as a CORS-compatible JSON, with the body containing the URIs of the video objects in the S3 bucket 'project-temp-bucket'.
6. The videos will be displayed on the front end page as a list, read from the search results JSON using JavaScript, along with the keywords.

### e. Video Play

To display and play the videos relevant to the user's search we use JavaScript code to create the required number of video objects dynamically and to play the video we add video controls using HTML properties.

### f. Document Upload

The user can upload a document from their computer. The frontend calls the uploadDocument API which stores the document uploaded in an S3 bucket called 'temp-bucket-testing-23'. Once the document is uploaded, the lambda function 'uploading-documents' is triggered which creates a boto3 client for Textract. The lambda function checks if the extension of the file is acceptable or not. It then calls the detect_document_text function on the Textract client and passes the document as argument. The result from this function is stored in an S3 bucket called 'textract-results-ccbdproj' as a json file.

g. **Document Text Extraction**

We extract text from the document images by calling AWS's Textract service. First, after images are uploaded to S3 bucket 'temp-bucket-testing-23', a Lambda function, "uploading-Documents", is triggered, and gets the image URI (S3 bucket location) from the event. It then calls AWS Textract using the boto3 API client, which detects text within the image using the "detect_document_text(...)" function. When this is complete, the result, a JSON file of words/tokens and confidence scores, is stored in another S3 bucket 'textract-results-ccbdproj'. This bucket will then trigger the indexing function, described in section h: Document Indexing and Search.

h. **Document Indexing and Search**

As with video indexing, document indexing involves creating a set of keywords that would be associated with each document/image file, and the keywords are searchable using AWS OpenSearch (which would return all document image files associated with a keyword or set of keywords). The <u>indexing process</u> is initiated in the following sequence (very similar to video indexing):

1. S3 bucket "textract-results-ccbdproj" triggers a Lambda function, "textract-convert-to-txt", whenever a textract job JSON is put into it. Unlike Transcribe, there is no continuous transcript produced from a Textract job, but we will create such a transcript in this Lambda function as a python string, and also convert the string to a text (.txt) file and store this file in another S3 bucket, "textract-results-ccbdproj-txt".
2. The created string will be used to again create a set of keywords, starting with all the words in the transcript and get pared down significantly (just like with video files).
3. Again, as with video, we call AWS Comprehend to return key phrases and named entities, and AWS Comprehend-Medical to return identified medical terminology from the document image's word/token string. Once again the boto3 API client will be used to call these services.
4. Once again, for AWS Comprehend, we first return a list of key phrases. We then go through each phrase and remove stopwords (commonly used English words that provide little context to a document), by checking if words within the key phrase are within a set of stopwords. We then add all distinct words from the key phrases set to the end keyword set/list.
5. We, once again, will use AWS Comprehend to return named entities (from the original transcript), i.e. names, locations, organizations, dates, titles, product names, etc. We will add all distinct named entities to the end keyword set/list.
6. Again, as with video indexing, we will call AWS Comprehend-Medical to return medical terminology from the string. We pass the string of Textract words/tokens and pull all medical terms, and add them to the end keyword set/list.

7. We then eliminate all redundant words in the final keyword set/list, and then call OpenSearch. The OpenSearch domain index we use is “https://search-photos-no-vpc-wnawrz54b6kh4bqyav7idotsdu.us-east-1.es.amazonaws.com/project_images/_doc”. We will pass the final keyword set as labels (keys), and the URI of the original document/image file as the associated keyword value, as a POST request.

As with video search, the photo search process itself is called from a front-end page, https://video-indexing-website-hosting.s3.amazonaws.com/search.html, as follows:

1. https://ykdkzg6745.execute-api.us-east-1.amazonaws.com/Test345 is the API called to search the videos using argument ‘../search?q='+x ‘, ‘method: GET’, where ‘x’ is the entered keyword(s).
2. The API gateway ‘searchvid/GET/.’ HTTP function uses a Lambda function, ‘project-keyword-search’ for its backend Integration Request.
3. ‘project-keyword-search’ takes the keywords as a string, separates them, and calls the OpenSearch index as before, except using a GET request (https://search-photos-no-vpc-wnawrz54b6kh4bqyav7idotsdu.us-east-1.es.amazonaws.com/project_images/_doc).
4. The GET request should return a list of ‘hits’, or successful URI matches for the keyword. The URI should point to the original document image S3 bucket ‘temp-bucket-testing-23’.
5. Results should be returned as a CORS-compatible JSON, with the body containing the URIs of the document image objects in the S3 bucket ‘temp-bucket-testing-23’.
6. The document images will be displayed on the front end page as a list, read from the search results JSON using JavaScript, along with the keywords.

## Results

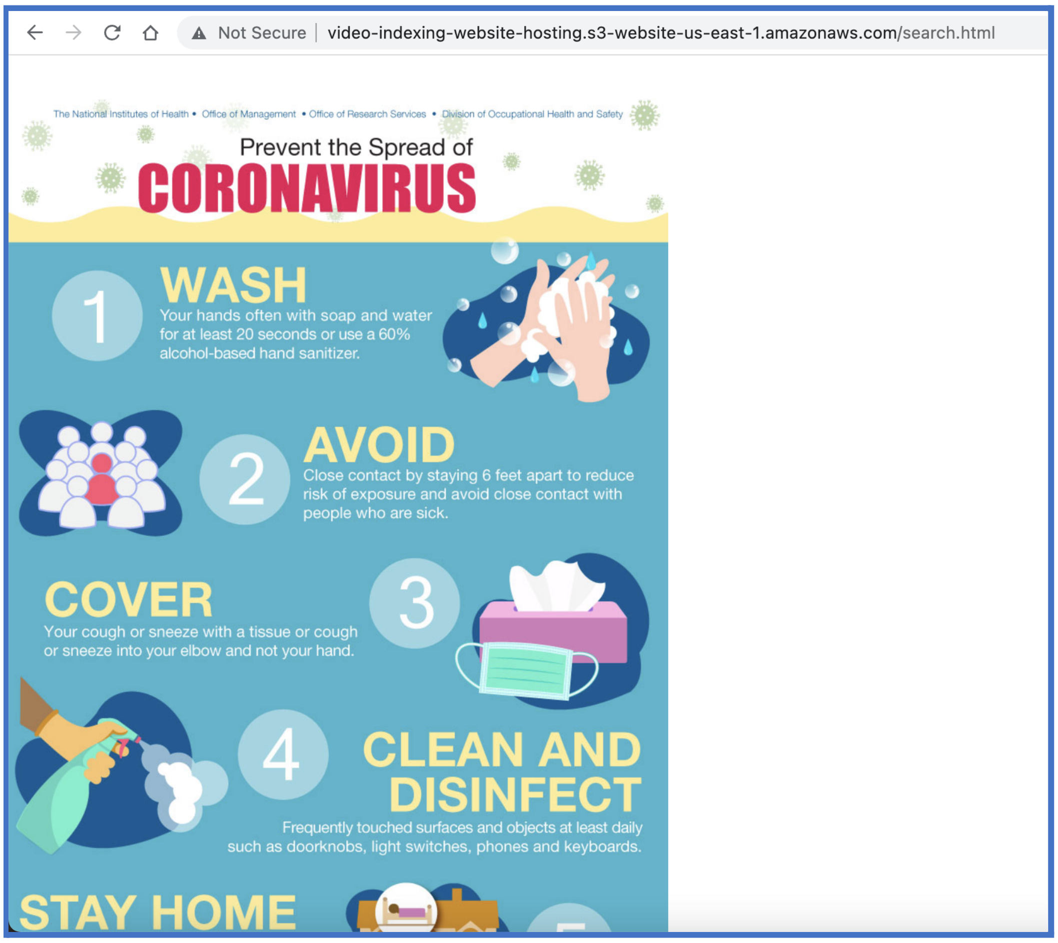
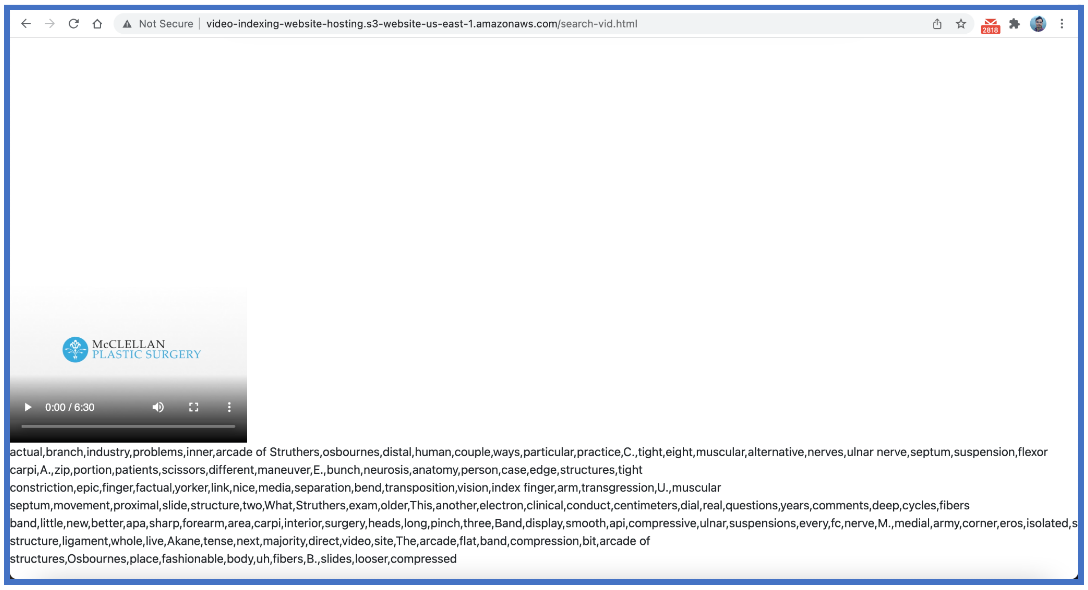

Not Secure
video-indexing-website-hosting.s3-website-us-east-1.amazonaws.com/search.html
The National Institutes of Health • Office of Management • Office of Research Services • Division of Occupational Health and Safety
Prevent the Spread of
CORONAVIRUS
1
WASH
Your hands often with soap and water for at least 20 seconds or use a 60% alcohol-based hand sanitizer.
2
AVOID
Close contact by staying 6 feet apart to reduce risk of exposure and avoid close contact with people who are sick.
COVER
Your cough or sneeze with a tissue or cough or sneeze into your elbow and not your hand.
3
4
CLEAN AND DISINFECT
Frequently touched surfaces and objects at least daily such as doorknobs, light switches, phones and keyboards.
STAY HOME

**Image 7: Document Search Result**


Not Secure
video-indexing-website-hosting.s3-website-us-east-1.amazonaws.com/search-vid.html
McCLELLAN PLASTIC SURGERY
0:00 / 6:30

actual,branch,industry,problems,inner,arcade of Struthers,osbournes,distal,human,couple,ways,particular,practice,C.,tight,eight,muscular,alternative,nerves,ulnar nerve,septum,suspension,flexor carpi,A.,zip,portion,patients,scissors,different,maneuver,E.,bunch,neurosis,anatomy,person,case,edge,structures,tight constriction,epic,finger,factual,yorker,link,nice,media,separation,bend,transposition,vision,index finger,arm,transgression,U.,muscular septum,movement,proximal,slide,structure,two,What,Struthers,exam,older,This,another,electron,clinical,conduct,centimeters,dial,real,questions,years,comments,deep,cycles,fibers band,little,new,better,apa,sharp,forearm,area,carpi,interior,surgery,heads,long,pinch,three,Band,display,smooth,api,compressive,ulnar,suspensions,every,fc,nerve,M.,medial,army,corner,eros,isolated,s structure,ligament,whole,live,Akane,tense,next,majority,direct,video,site,The,arcade,flat,band,compression,bit,arcade of structures,Osbournes,place,fashionable,body,uh,fibers,B.,slides,looser,compressed

**Image 8: Video Search Result**

## Summary

We deliver a system for the medical community which allows users to upload videos and documents for open source access and search through them so that they may view the documents or videos which are relevant to the keywords they wish to search for. It is a retrieval and indexing system which enables fast and easy access to data in a video and document repository. Users are able to sign up using email verification and log in, upload videos and documents to the database, and search through the contents of the database. The project features many capabilities like OCR through Textract, video transcription through Transcribe, NLP capabilities to identify keywords for indexing using Comprehend and Comprehend-Medical. Through this project, the task of searching through medical videos and images, to find ones relevant to the user's search becomes hassle free. The users are able to view their search results in real time and can play the videos or read the documents easily. They also see many keywords relevant to the videos and documents shown to them so that they get an idea of what that video/document is about and they can filter through them much faster.

## Project Link

https://github.com/aayush912/medical-video-and-document-indexing-system

## Project Demonstration Video

▶ Medical Video & Document Indexer