

Rozpoznanie obrazu z wykorzystaniem rozwiązań chmurowych AWS

Wykonali:

Julian Przybysz 299213

Łukasz Strep 293016

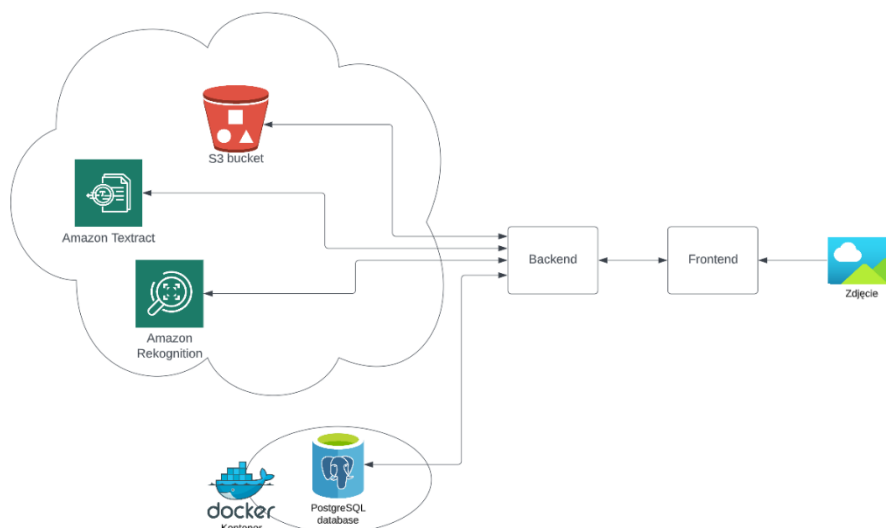
Jakub Słowikowski 299221

Konrad Dudek 299188

Wstęp

W poniższym sprawozdaniu opisano projekt umożliwiający rozpoznanie obrazu z wykorzystaniem rozwiązań chmurowych AWS. Implementację wykonano jako aplikację webową z użyciem Pythona, PostgreSQL, FastAPI oraz Reacta. Aplikacja posiada dwie główne funkcjonalności: analizę zdjęcia osoby oraz transkrypcja załączonego zdjęcia. Upload obrazu odbywa się do Storage S3, następnie zdjęcie jest poddawane analizie. Dla pierwszej funkcjonalności wykorzystano technologię deep-learning Amazon Rekognition, natomiast dla drugiej – Amazon Textract - jest to serwis, który automatycznie wykrywa i ekstrahuje tekst z zeskanowanego zdjęcia.

Schemat architektury



Architektura stworzonej aplikacji

Technologie

Zastosowano 6 technologii:

- Storage S3
- MFA
- Cloudtrail
- Budget
- Amazon rekognition
- Amazontextract

1) Stworzenie bucketu S3:

S3 to storage służący do przechowywania plików w chmurze. Oferuje dużą skalowalność, dostępność danych oraz bezpieczeństwo. Dzięki dużej możliwości konfiguracji dobrze dostosowuje się do użytkownika, niezależnie od skali do której będzie wykorzystywany.

W naszym projekcie użyliśmy bucketa S3 do przechowywania wgranych do aplikacji zdjęć

Tworzenie bucketa:

Amazon S3 > Buckets > Create bucket

Create bucket [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name

image-recognition-1

Bucket name must be globally unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

AWS Region

US East (N. Virginia) us-east-1

Copy settings from existing bucket - *optional*
Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

Object Ownership [Info](#)

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

☒ **ACLs disabled (recommended)**
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

☐ **ACLs enabled**
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership

Bucket owner enforced

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

- ☐ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
- ☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ **Block public access to buckets and objects granted through any access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☐ **Block public access to buckets and objects granted through new public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☐ **Block public and cross-account access to buckets and objects through any public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.



Turning off block all public access might result in this bucket and the objects within becoming public

AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Otrzymany bucket:

Account snapshot

Storage lens provides visibility into storage usage and activity trends. [Learn more](#)

View St

Buckets (1) Info

Buckets are containers for data stored in S3. [Learn more](#)

Copy ARN

Empty

Delete

Find buckets by name

Name	AWS Region	Access	Creation date
image-recognition-1	US East (N. Virginia) us-east-1	Objects can be public	January 5, 2023, 17:47:10 (UTC+01:00)

2) MFA

MFA to inaczej AWS Multi-Factor Authentication – czyli weryfikacja wieloetapowa. Jest to technologia pozwalająca zapewnić dodatkową warstwę bezpieczeństwa. Kiedy jest włączona, użytkownik logując się do konsoli AWS oprócz hasła i loginu musi też podać dodatkowy kod - który jest generowany na osobnym urządzeniu. Dzięki temu, mamy większą pewność, że nikt nie włamie się na nasze konto i nie wykorzysta naszych zasobów z karty 😊 Istnieje też możliwość używania zabezpieczeń biometrycznych

Ustawienia MFA

Introducing the new Security Credentials experience

We've redesigned the Security Credentials experience to make it easier to use. [Let us know what you think.](#)

MFA device assigned

You can register up to 8 MFA devices of any combination of the currently [supported MFA types](#) with your AWS account root and IAM user. With multiple MFA devices, you only need one MFA device to sign in to the AWS console or create a session through the AWS CLI with that user.

IAM > Security credentials

My security credentials (root user) [Info](#)

The root user has access to all AWS resources in this account, and we recommend following [best practices](#). To learn more about the types of AWS credentials and how they're used, see [AWS Security Credentials](#) in AWS General Reference.

Account details

Account name

przybjul

Email address

przybjul@gmail.com

AWS account ID

617278204606

Canonical user ID

6aed5335f522637599a4d4a0e061f40cf40afda4bdfd2d81c8fb1086eabf02cf

Edit account name, email, and password

Multi-factor authentication (MFA) (1)

Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. [Learn more](#)

Remove

Resync

Assign MFA device

Device type	Identifier	Created on
<input type="radio"/> Virtual	arn:aws:iam::617278204606:mfa/przybjul	Now

Logowanie z użyciem MFA



Multi-factor authentication

Your account is secured using multi-factor authentication (MFA). To finish signing in, turn on or view your MFA device and type the authentication code below.

Email address: przybjul@gmail.com

MFA code

Submit

[Troubleshoot MFA](#)

3) AWS CloudTrail

AWS CloudTrail jest narzędziem umożliwiającym monitorowanie i rejestrowanie działań wykonywanych na naszym koncie AWS. CloudTrail pozwala nam na śledzenie aktywności poprzez identyfikację kto lub co wykonało dane działanie, kiedy ono miało miejsce oraz na jakich zasobach zostało wykonane. Rozwiązanie to pozwala na lepsze zabezpieczenie naszych danych oraz zrozumienie wszystkich przepływów na naszym koncie. AWS CloudTrail w naszym projekcie zaimplementowaliśmy w kolejnych krokach:

1. Stworzono potrzebny klucz KMS

KMS > Customer managed keys > Create key

Step 1
Configure key

Step 2
Add labels

Step 3
Define key administrative permissions

Step 4
Define key usage permissions

Step 5
Review

Configure key

Key type [Help me choose](#)

☒ **Symmetric**
A single key used for encrypting and decrypting data or generating and verifying HMAC codes

☐ **Asymmetric**
A public and private key pair used for encrypting and decrypting data or signing and verifying messages

Key usage [Help me choose](#)

☒ **Encrypt and decrypt**
Use the key only to encrypt and decrypt data.

☐ **Generate and verify MAC**
Use the key only to generate and verify hash-based message authentication codes (HMAC).

► **Advanced options**

Cancel **Next**

2. Dodano aliasy dla etykiet

Add labels

Alias
You can change the alias at any time. [Learn more](#)

Alias

Description - optional
You can change the description at any time.

Description - optional

Tags - optional

You can use tags to categorize and identify your KMS keys and help you track your AWS costs. When you add tags to AWS resources, AWS generates a cost allocation report for each tag. [Learn more](#)

This key has no tags.

Add tag

You can add up to 50 more tags.

Cancel Previous Next

3. Nazwa Traila i wybranie atrybutów:

Choose trail attributes

General details
A trail created in the console is a multi-region trail. [Learn more](#)

Trail name
Enter a display name for your trail.

3-128 characters. Only letters, numbers, periods, underscores, and dashes are allowed.

☐ **Enable for all accounts in my organization**
To review accounts in your organization, open AWS Organizations. [See all accounts](#)

Storage location [Info](#)

☐ **Create new S3 bucket**
Create a bucket to store logs for the trail.

☒ **Use existing S3 bucket**
Choose an existing bucket to store logs for this trail.

Trail log bucket name
Enter a new S3 bucket name and folder (prefix) to store your logs. Bucket names must be globally unique.

×

Browse

Prefix - optional

Logs will be stored in image-recognition-1/AWSLogs/617278204606

Log file SSE-KMS encryption [Info](#)
☒ **Enabled**

Customer managed AWS KMS key
☒ **New**
☐ **Existing**

AWS KMS alias

KMS key and S3 bucket must be in the same region.

4. Wybranie wydarzeń do logowania:

Choose log events

Events [Info](#)
Record API activity for individual resources, or for all current and future resources in AWS account. [Additional charges apply](#) [↗](#)

Event type
Choose the type of events that you want to log.
☒ **Management events**
Capture management operations performed on your AWS resources.
☐ **Data events**
Log the resource operations performed on or within a resource.
☐ **Insights events**
Identify unusual activity, errors, or user behavior in your account.

Management events [Info](#)
Management events show information about management operations performed on resources in your AWS account.

i Charges apply to log management events on this trail because you are logging at least one other copy of management events in your account.

API activity
Choose the activities you want to log.
☒ **Read** ☒ **Write**
☐ **Exclude AWS KMS events**
☐ **Exclude Amazon RDS Data API events**

[Cancel](#) [Previous](#) [Next](#)

5. Utworzony AWS CloudTrail:

Introducing CloudTrail Lake
CloudTrail Lake lets you query multiple event fields in your logs, across all regions, for auditing and analysis. [Learn more](#) [↗](#)

CloudTrail > Trails

Trails [↻](#) [Delete](#) [Create trail](#) [⊕](#)

	Name ▲	Home region ▼	Multi-region trail ▼	Insights ▼	Organization trail ▼	S3 bucket ▼	Log file prefix ▼	CloudWatch Logs log group ▼	Status ▼
<input type="radio"/>	image-recognition-1-trail	US East (N. Virginia)	Yes	Disabled	No	image-recognition-1			Logging

4) AWS Budgets

AWS Budgets służą do monitorowania używania zasobów oraz naszych kosztów. Zapewniają wiele różnych funkcjonalności ustawiania limitów wykorzystania zasobów lub pieniędzy, a także przesyłają odpowiednie powiadomienia, gdy te limity są przekraczane.

You can now create a budget from a [template](#), or explore hands-on [tutorials](#) to configure simple or advanced budgets. [Learn more](#)

[Billing Console](#) > [Budgets](#) > [Create budget](#)

Choose budget type [Info](#)

Budget setup

☒ **Use a template (simplified)**
 Use the recommended configurations. You can change some configuration options after the budget is created.

☐ **Customize (advanced)**
 Customize a budget to set parameters specific to your use case. You can customize the time period, the start month, and specific accounts.

Templates - *new*

Choose a template that best matches your use case.

☒ **Zero spend budget**
 Create a budget that notifies you once your spending exceeds \$0.01 which is above the AWS Free Tier limits.

☐ **Monthly cost budget**
 Create a monthly budget that notifies you if you exceed, or are forecasted to exceed, the budget amount.

☐ **Daily Savings Plans coverage budget**
 Create a coverage budget for your Savings Plans that notifies you when you fall below the defined target.

☐ **Daily reservation utilization budget**
 Create a utilization budget for your reservations that notifies you when you fall below the defined target.

Zero spend budget - Template

Budget name

Provide a descriptive name for this budget.

Names must be between 1-100 characters.

You can now create a budget from a [template](#), or explore hands-on [tutorials](#) to configure simple or advanced budgets. [Learn more](#)

[Submit feedback](#)

Your budget **My Zero-Spend Budget** has been created successfully. After creating a budget, it can take up to 24 hours to populate all of your spend data.

[Submit feedback](#)

[Billing Console](#) > [Budgets](#) > [Overview](#)

Overview [Info](#)

Budgets (1) [Info](#)

[Show all budgets](#)

[Download CSV](#)
[Actions](#)
[Create budget](#)

<input type="checkbox"/>	Name	▲	Thresholds	▼	Budget	Amount used	Forecasted amount	Current vs. budgeted	▼	Forecasted vs. budgeted	▼
<input type="checkbox"/>	My Zero-Spend Budget		OK			\$1.00	\$0.00	-		0.00%	-

5) Amazon rekognition

Amazon Rekognition to usługa sztucznej inteligencji oferowana przez Amazon Web Services, która pozwala na analizowanie obrazów i wideo. Umożliwia ona rozpoznawanie twarzy, ludzi, obiektów, tekstu, scen, a także emocji wyrażanych na twarzy – co zostało użyte w naszym projekcie.

Usługa ta umożliwia również tworzenie i zarządzanie katalogami zawierającymi zdjęcia i informacje o nich, co pozwala na szybkie i łatwe porównywanie i rozpoznawanie obrazów. Dzięki Amazon rekognition, można automatycznie rozpoznawać i kategoryzować obrazy, co pozwala na automatyzację procesów biznesowych i zwiększenie efektywności działań.

Amazon Rekognition jest przydatny w różnych dziedzinach, takich jak: bezpieczeństwo, marketing, automatyzacja procesów biznesowych, rozpoznawanie twarzy, analiza obrazów i wideo, rozpoznawanie tekstu, a także w analizie trendów, opinii i emocji wyrażanych na twarzy.



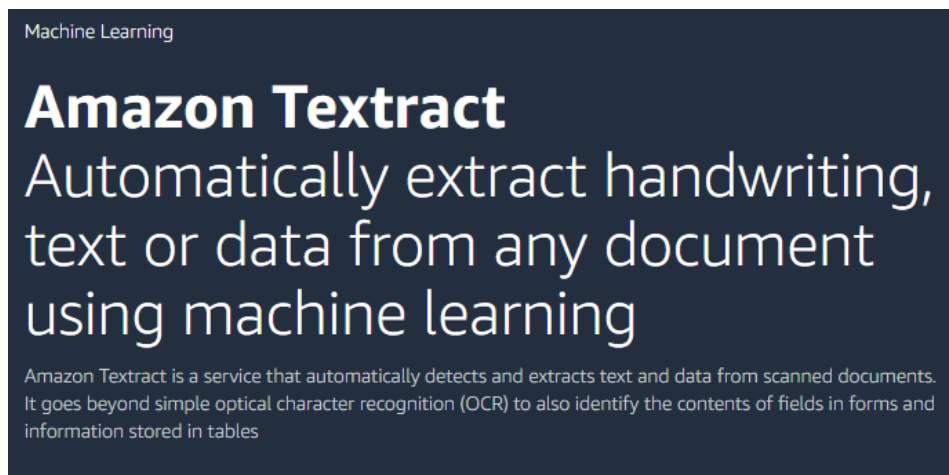
Aby korzystać z zasobów AWS skorzystaliśmy z biblioteki boto3, która jest Pythonowym SDK dla AWSa. Biblioteka ta pozwala na bezpośrednie tworzenie, aktualizowanie, dostęp do oraz usuwanie danych bezpośrednio ze skryptów Pythonowych.

```
1 import boto3
2
3 def get_image_description(photo: str):
4     client = boto3.client('rekognition', aws_access_key_id="",
5                             aws_secret_access_key="", region_name='us-east-1')
6     #rozpoznanie twarzy You, 12 hours ago * image transcription ...
7     response = client.detect_faces(Image={'S3Object':{
8         "Bucket":"image-recognition-1",
9         "Name":photo
10      }},Attributes=['ALL'])
11
12     for key, value in response.items():
13         if key == 'FaceDetails':
14             for people_att in value:
15                 gender = str(people_att["Gender"]["Value"])
16                 low_age_range = str(people_att["AgeRange"]["Low"])
17                 high_age_range = str(people_att["AgeRange"]["High"])
18                 smile = str(people_att["Smile"]["Value"])
19                 if smile == "True":
20                     is_smiling = "is smiling."
21                 else:
22                     is_smiling = "is not smiling."
23                 emotion = people_att["Emotions"][0]["Type"]
24                 # print("This person is "+gender+. Age of this person is "+low_age_range+. Age of this person is "+high_age_range+. Emotion is "+emotion)
25                 string = "This person is "+gender+. Age of this person is "+low_age_range+. Age of this person is "+high_age_range+. Emotion is "+emotion
26     return string
```

W zaprezentowanym fragmencie kodu łączymy się z Bucketem „image-recognition-1”, przekazując jako jeden z parametrów nazwę zdjęcia. W odpowiedzi otrzymujemy atrybuty takie jak płeć osoby znajdującej się na zdjęciu, zakres wieku, informacja o tym czy osoba się uśmiecha oraz typ emocji. Na podstawie tych informacji tworzona jest informacja zwrotna wyświetlana na frontendzie.

6) Amazon textract:

Jest to serwis wykorzystujący uczenie maszynowe do rozpoznawania przesłanego tekstu - zarówno dla pisma ręcznego jak i drukowanego. Z jego użyciem można w prosty sposób wyciągać tekst z dokumentów w sposób automatyczny. Jest to na tyle zaawansowana technologia, że pozwala na przetwarzanie danych w tabelach.



Funkcja wyciągająca tekst z odpowiedniego obrazu w bazie:

```
def get_image_transcription(photo: str):
    # S3 Bucket Data
    s3BucketName = "image-transcribe-1"
    # Amazon Textract client
    textractmodule = boto3.client('textract', aws_access_key_id="",
                                   aws_secret_access_key="", region_name='us-east-1')

    response = textractmodule.detect_document_text(
        Document={
            'S3Object': {
                'Bucket': s3BucketName,
                'Name': photo
            }
        })
    text = ""
    for item in response["Blocks"]:
        if item["BlockType"] == "LINE":
            text = text + item["Text"] + " "
    return text
```

Budowa:

Do pracy z aplikacją użyto relacyjnego silnika bazodanowego PostgreSQL. Zdecydowano się na użycie Dockera. Poniżej zamieszczono konfigurację docker-compose:

```
photo-upload-full-stack-master > backend > 🚀 docker-compose.yml
You, 2 weeks ago | 1 author (You)
1  services:
2    db:
3      image: postgres:15.0-alpine
4      ports:
5        - 5432:5432
6      restart: always
7      environment:
8        POSTGRES_USER: docker
9        POSTGRES_PASSWORD: docker
10       POSTGRES_DB: exampledb
11  adminer:
12    image: adminer
13    restart: always
14    depends_on:
15      - db
16    ports:
17      - 8080:8080
18
19  volumes:
20    db-photo-upload-service:
21
```

Containers [Give feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

☐ Only show running containers

<input type="checkbox"/>	NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	backend	-	Running (2/2)			
<input type="checkbox"/>	adminer-1 65d09a99201f	adminer:latest	Running	8080:8080	2 hours ago	
<input type="checkbox"/>	db-1 5f26015e26ef	postgres:15.0-alpine	Running	5432:5432	2 hours ago	

FastAPI użyto do tworzenia interfejsów Rest API:

```
from fastapi import FastAPI, UploadFile
from fastapi.middleware.cors import CORSMiddleware
```

Inicjalizacja + CORS middleware:

```
22
23 app = FastAPI(debug=True)
24 #Middleware do autoryzacji
25 app.add_middleware(
26     CORSMiddleware,
27     allow_origins=["*"],
28     allow_credentials=True,
29     allow_methods=["*"],
30     allow_headers=["*"],
31 )
32
```

Web-server uvicorn na lokalnym porcie 8080:

```
173
174 if __name__ == "__main__":
175     uvicorn.run(app, host="localhost", port=8000, reload=False)
176
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  AZURE  GITLENS

INFO:      Started server process [9800]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
INFO:      Uvicorn running on http://localhost:8000 (Press CTRL+C to quit)
```

Uruchomienie frontendu przy pomocy komendy *npm start*:

```
PS C:\Users\przyb\Desktop\Projekt_chmury\AWS_image_recognition\photo-upload-full-stack-master\frontend> npm start

> frontend@0.1.0 start
  Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000
```

Adminer:

Narzędzie Adminer pozwala na zarządzanie bazami danych:

Język: Polski

Adminer 4.8.1

(PostgreSQL) docker@db - example

< >

Zaloguj się

Rodzaj bazy	PostgreSQL
Serwer	db
Użytkownik	docker
Hasło
Baza danych	exampledb

Zaloguj się ☐ Zapamiętaj sesję

Dodawanie tabel do bazy:

Tabele wykorzystujemy do przechowywania nazw uploadowanych obrazów oraz linków do nich w store'ie S3. Stworzyliśmy dwie nowe tabele – pierwsza analizy twarzy osoby, a druga do transkrypcji:

Schemat: public

[Zmień schemat](#) [Schemat bazy danych](#)

Tabele i perspektywy

Wyszukaj we wszystkich tabelach (2)

<input type="checkbox"/>	Tabela	Składowanie	Porównywanie znaków	Rozmiar danych?	Rozmiar indeksów?	Właściwości
<input type="checkbox"/>	photo	table		8 192	24 576	
<input type="checkbox"/>	transcribe	table		8 192	24 576	
	2 w sumie		en_US.utf8	16 384	49 152	

PostgreSQL » db » exampledb » public » Tabela: photo

Tabela: photo

[Pokaż dane](#) [Struktura tabeli](#) [Zmień tabelę](#) [Nowy rekord](#)

Kolumna	Typ	Komentarz
photo_name	character varying	
photo_url	text NULL	
is_deleted	text NULL	
id	integer Auto Increment [nextval('photo_id_seq')]	

Indeksy

PRIMARY id

Następnie wprowadzono przykładowe dane:

pokaż: photo

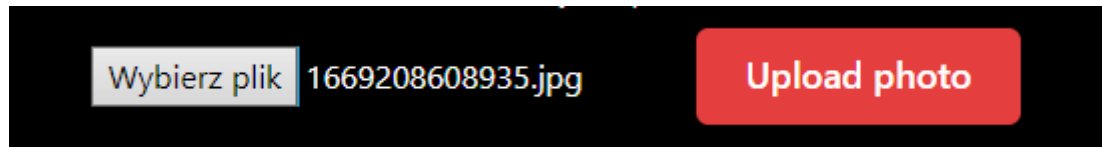
Pokaż dane Struktura tabeli Zmień tabelę Nowy rekord

SELECT * FROM "photo" LIMIT 50 (0.001 s) [Edytuj](#)

<input type="checkbox"/> Zmień	photo_name	photo_url	is_deleted	id
<input type="checkbox"/> edytuj	1669208608935.jpg	https://image-recognition-1.s3.amazonaws.com/1669208608935.jpg	NULL	1
<input type="checkbox"/> edytuj	Unauthorized_access.PNG	https://image-recognition-1.s3.amazonaws.com/Unauthorized_access.PNG	NULL	2
<input type="checkbox"/> edytuj	zdj.jpg	https://image-recognition-1.s3.amazonaws.com/zdj.jpg	NULL	3
<input type="checkbox"/> edytuj	1669209577236.jpg	https://image-recognition-1.s3.amazonaws.com/1669209577236.jpg	NULL	4
<input type="checkbox"/> edytuj	1669208608935.jpg	https://image-recognition-1.s3.amazonaws.com/1669208608935.jpg	NULL	5

Funkcjonalność 1 (analiza obrazu) krok po kroku:

Aby wybrać zdjęcie, które ma zostać poddane analizie, należy nacisnąć przycisk 'Uplot photo'. Następnie tworzone jest żądanie HTTP POST pod endpoint '/photos'.



```
const onChange = (e) => {
  setIsSelected(true);
  setSelectedFile(e.target.files[0]);
};
const onFileUpload = (e) => {
  setShowSpinner(true);
  const formData = new FormData();
  formData.append("file", selectedFile, selectedFile.name);
  fetch("http://localhost:8000/photos", {
    method: "POST",
    body: formData,
  })
    .then((response) => response.json())
    .then((data) => {
      console.log("Success posting");
      setUploadSuccessful(!uploadSuccessful);
      setShowSpinner(false);
    });
};
```

Następnie backend łączy się z Amazonem poprzez "boto 3" i wgrywa obraz na Storage S3 na odpowiedni bucket. Potem tworzony jest adres URL wgranego pliku, nawiązywane jest łączy się z bazą danych i dodawany w bazie nowy rekord do tabeli photos z nazwą pliku i url do bucketu s3.

```

@app.post("/photos", status_code=201)
async def add_photo(file: UploadFile):
    print("Create endpoint hit!!")
    print(file.filename)
    print(file.content_type)

    # Upload file to AWS S3
    s3 = boto3.resource(
        "s3",
        aws_access_key_id="",
        aws_secret_access_key="",
    )
    bucket = s3.Bucket(S3_BUCKET_NAME)
    bucket.upload_fileobj(file.file, file.filename, ExtraArgs={"ACL": "public-read"})

    uploaded_file_url = f"https://{S3_BUCKET_NAME}.s3.amazonaws.com/{file.filename}"

    # Store URL in database
    conn = psycopg2.connect(
        database="exampledb",
        user="docker",
        password="docker",
        host="localhost",
        port="5432",
    )
    cur = conn.cursor()
    cur.execute(
        f"INSERT INTO photo (photo_name, photo_url) VALUES ('{file.filename}', '{uploaded_file_url}')"
    )
    conn.commit()
    cur.close()
    conn.close()

```

Następnie frontend wywołuje endpoint GET, z którego wyciąga plik json z nazwą zdjęcia, url do S3 oraz opisem(description):

```

useEffect(() =>{
    fetch("http://localhost:8000/photos")
    .then((response) => response.json())
    .then((data) => {
        console.log(data);
        setAllPhotos(data);
    });
},[uploadSuccessful]);

```

Następnie endpoint łączy się z bazą, bierze pierwszy od góry rekord z tabeli 'photo'

```

39 @app.get("/photos", response_model=List[PhotoModel])
40 async def get_all_photos():
41     # Connect to our database
42     conn = psycopg2.connect(
43         database="exampledb",
44         user="docker",
45         password="docker",
46         host="localhost",
47         port=5432,
48     )
49     cur = conn.cursor()
50     cur.execute("SELECT * FROM photo ORDER BY id DESC")
51     rows = cur.fetchall()
52     formatted_photos = []
53     for row in rows:
54         print(row[0])
55         photo_description = get_image_description(photo=row[0])
56         formatted_photos.append(
57             PhotoModel(
58                 id=row[3],
59                 photo_name=row[0],
60                 photo_url=row[1],
61                 is_deleted=row[2],
62                 description=photo_description,
63             )
64         )
65     break
66     print(f"{formatted_photos}")
67     cur.close()
68     conn.close()
69     return formatted_photos
70

```

Następnie wywołuje funkcję 'get_image_description' w pliku orm, która łączy się z technologią AWS Rekognition:

```
def get_image_description(photo: str):
    client = boto3.client(
        "rekognition",
        aws_access_key_id="",
        aws_secret_access_key="",
        region_name="us-east-1",
    )
    # rozpoznanie twarzy
    response = client.detect_faces(
        Image={"S3Object": {"Bucket": "image-recognition-1", "Name": photo}},
        Attributes=["ALL"],
    )

    for key, value in response.items():
        if key == "FaceDetails":
            for people_att in value:
                gender = str(people_att["Gender"]["Value"])
                low_age_range = str(people_att["AgeRange"]["Low"])
                high_age_range = str(people_att["AgeRange"]["High"])
                smile = str(people_att["Smile"]["Value"])
                if smile == "True":
                    is_smiling = "is smiling."
                else:
                    is_smiling = "is not smiling."
                emotion = people_att["Emotions"][0]["Type"]
                string = (
                    "This person is "
                    + gender
                    + ". Age of this person is in range from "
                    + low_age_range
                    + " to "
                    + high_age_range
                    + " years old. This person "
                    + is_smiling
                    + " Emotions: "
                    + emotion
                )
    return string
```

Z tej funkcji wyciągany jest string zawierający informację o płci, zakresie wieku, uśmiechu oraz emocji.

Następnie informacje z bazy i funkcji rekognition są przekazywane do photo_model, który następnie zwracany jest do frontendu, który wyświetla obraz i opis:

```
</HStack>
<Heading>Analyzed photo</Heading>
<SimpleGrid columns={1} spacing={8}>
  {
    allPhotos.map(photo =>{
      console.log(photo["photo_url"])
      return(
        <Center>
          <Image width="400px" src={photo["photo_url"]} fallbackSrc="https://via.placeholder.com/150" objFit="cover"></Image>
        </Center>
      )
    })
  }
  {
    allPhotos.map(photo =>{
      console.log(photo["description"])
      return(
        <Text>{photo["description"]}
      </Text>
    })
  }
</SimpleGrid>
```

Your gallery

Lets analyze your photo!

Wybierz plik julian_smile1.jpg

Upload photo

Analized photo

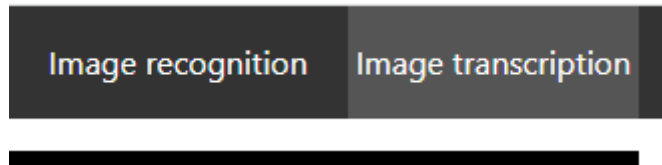


This person is Male. Age of this person is in range from 20 to 28 years old. This person is smiling. Emotions: HAPPY

Po wgraniu zdjęcia program wyświetla opis osoby w którym znajduje się informacja o płci, przedziale wieku oraz emocjach. Podczas testów program miał bardzo wysoką dokładność rozpoznania.

Druga funkcjonalność (transkrypcja):

Aby skorzystać z drugiej funkcjonalności naszego programu trzeba zmienić zakładkę znajdującą się w górnym prawym rogu ekranu (Image transcription):



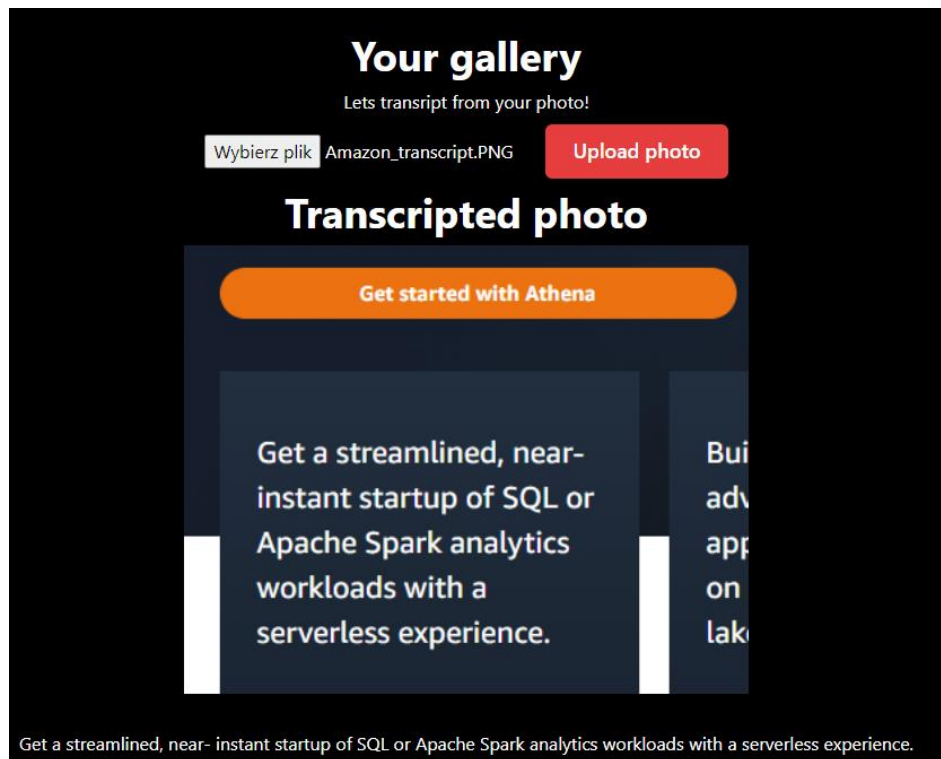
Budowa jest analogiczna do pierwszej funkcjonalności tylko korzystamy z innej tabeli postgres – transcribe oraz innego bucketu. Wywoływana jest inna funkcja - `get_image_transcription`, odwołującą się do innej technologii – AWS textract, który został opisany wcześniej.

Aby skorzystać z funkcjonalności:

1. Wgrywamy zdjęcie, na którym znajduje się tekst napisany ręcznie lub w programie komputerowym
2. Rozpoznany tekst drukowany jest pod zdjęciem

Przykład 1:

Zostało wgrane zdjęcie z bardzo czytelnym pismem drukowanym



Program perfekcyjnie rozpoznał tekst, a następnie wydrukował go pod zdjęciem.

Przykład 2:

Zostało wgrane zdjęcie z pismem ręcznym



Tekst nie został rozpoznany idealnie. Program jest wrażliwy na różne charaktery pisma.

Przydatne linki:

1. <https://youtu.be/dqCGewXc-j0> - film opisujący wszystkie funkcjonalności
2. https://github.com/przybjul/AWS_image_recognition?fbclid=IwAR3Uuh-ExlzZmIKRjOzM5HahtxSyPI-sQ8CP-JbJAK_GxKlk-v0wf-yrPKk - repo