Rozpoznanie obrazu z wykorzystaniem rozwiązań chmurowych AWS

Wykonali:

Julian Przybysz 299213

Łukasz Strep 293016

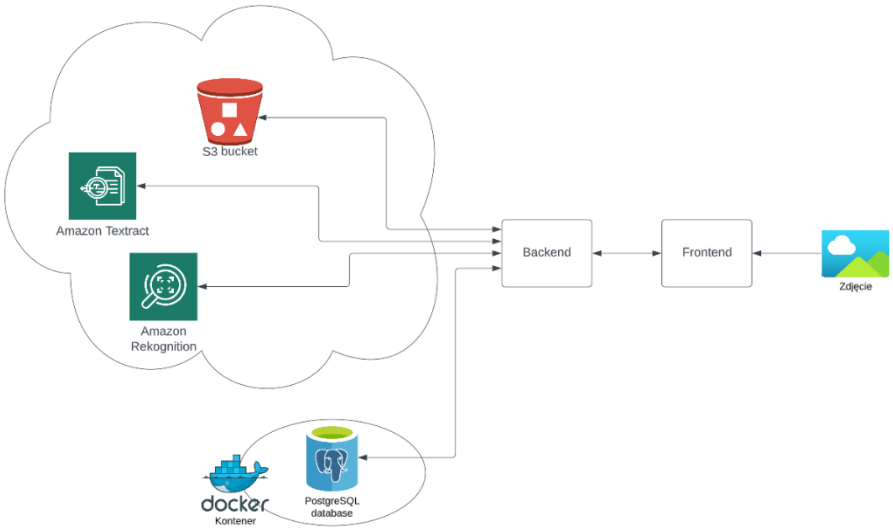
Jakub Słowikowski 299221

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, PostreSQL, 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:

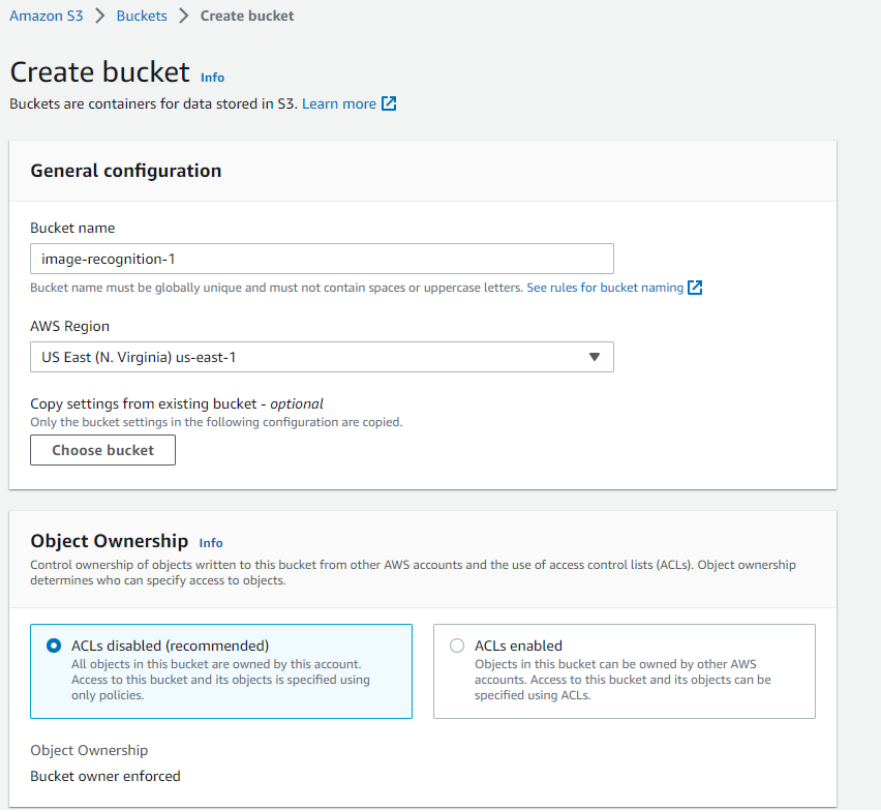
- Storage S3
- MFA
- Cloudtrail
- Budget
- Amazon rekognition
- Amazontextract

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:



Block Public Access settings for this bucket

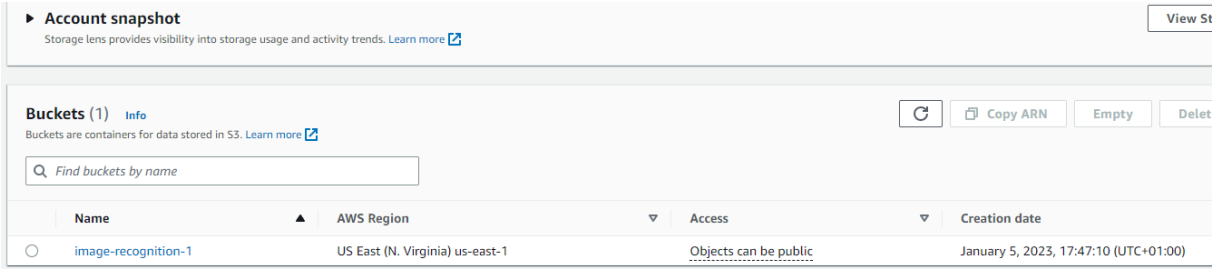
Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access 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.

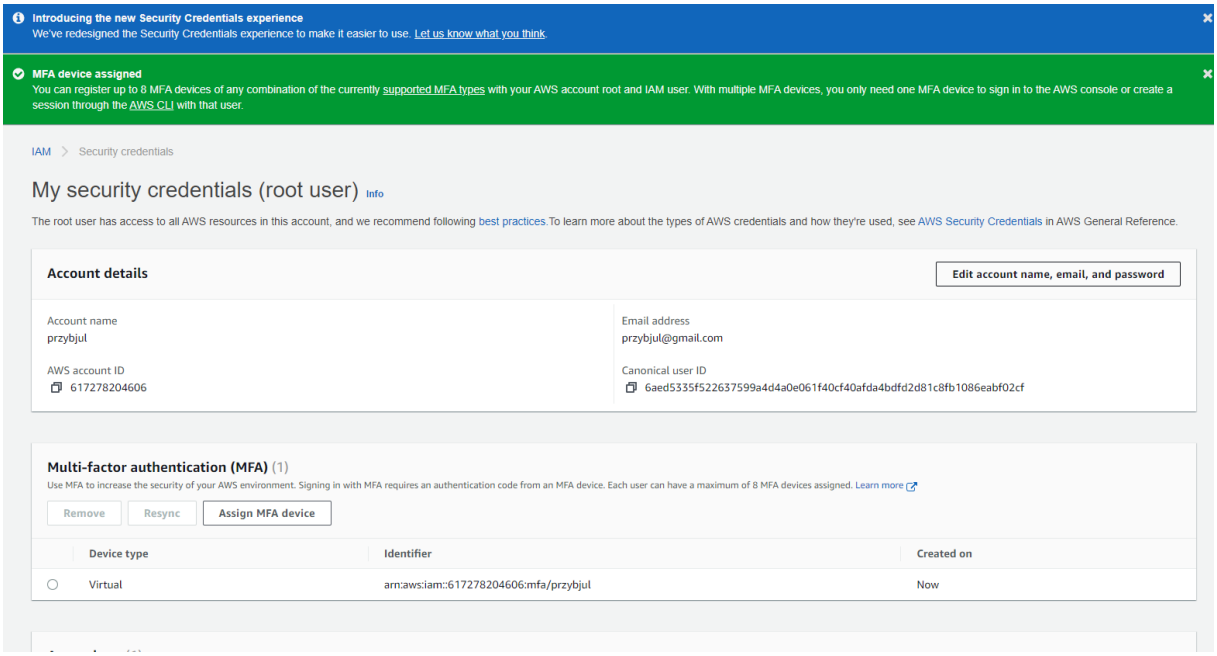
Otrzymany bucket:



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



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

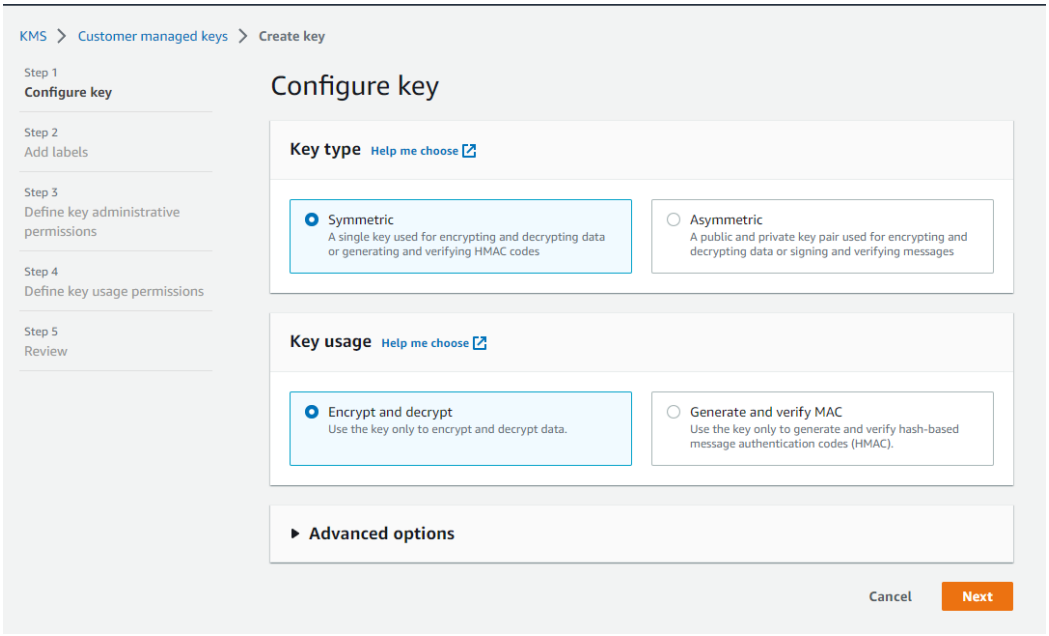
Submit

Troubleshoot MFA

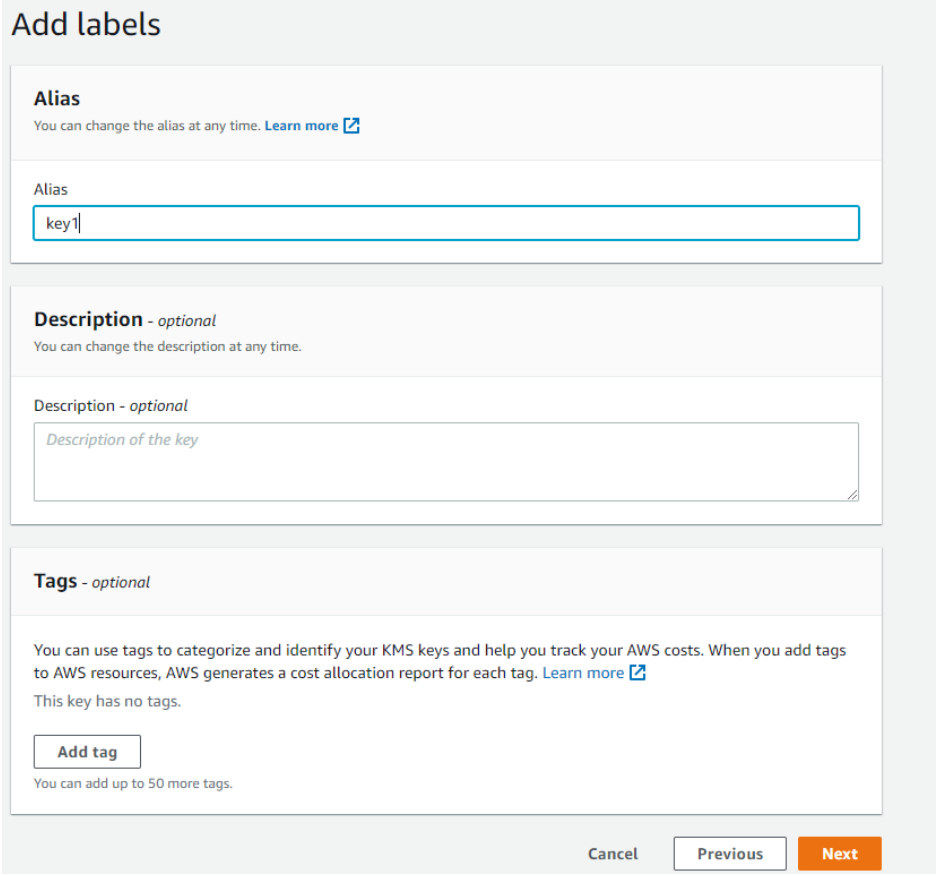
3) AWS CloudTrial

AWS CloudTrial jest narzędziem umożliwiającym monitorowanie i rejestrowanie działań wykonywanych na naszym koncie AWS. CloudTrial 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 CloudTrial w naszym projekcie zaimplementowaliśmy w kolejnych krokach:

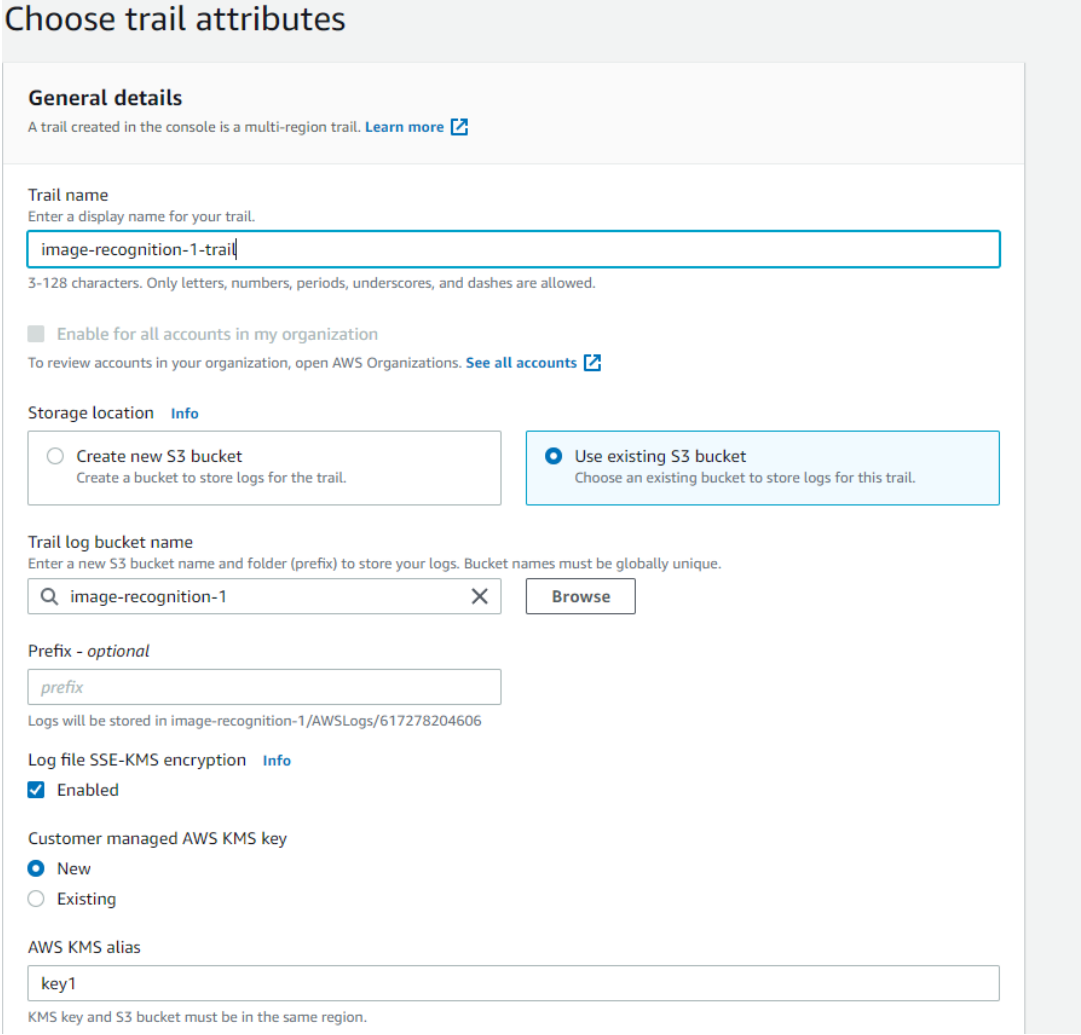
1. Stworzono potrzebny klucz KMS



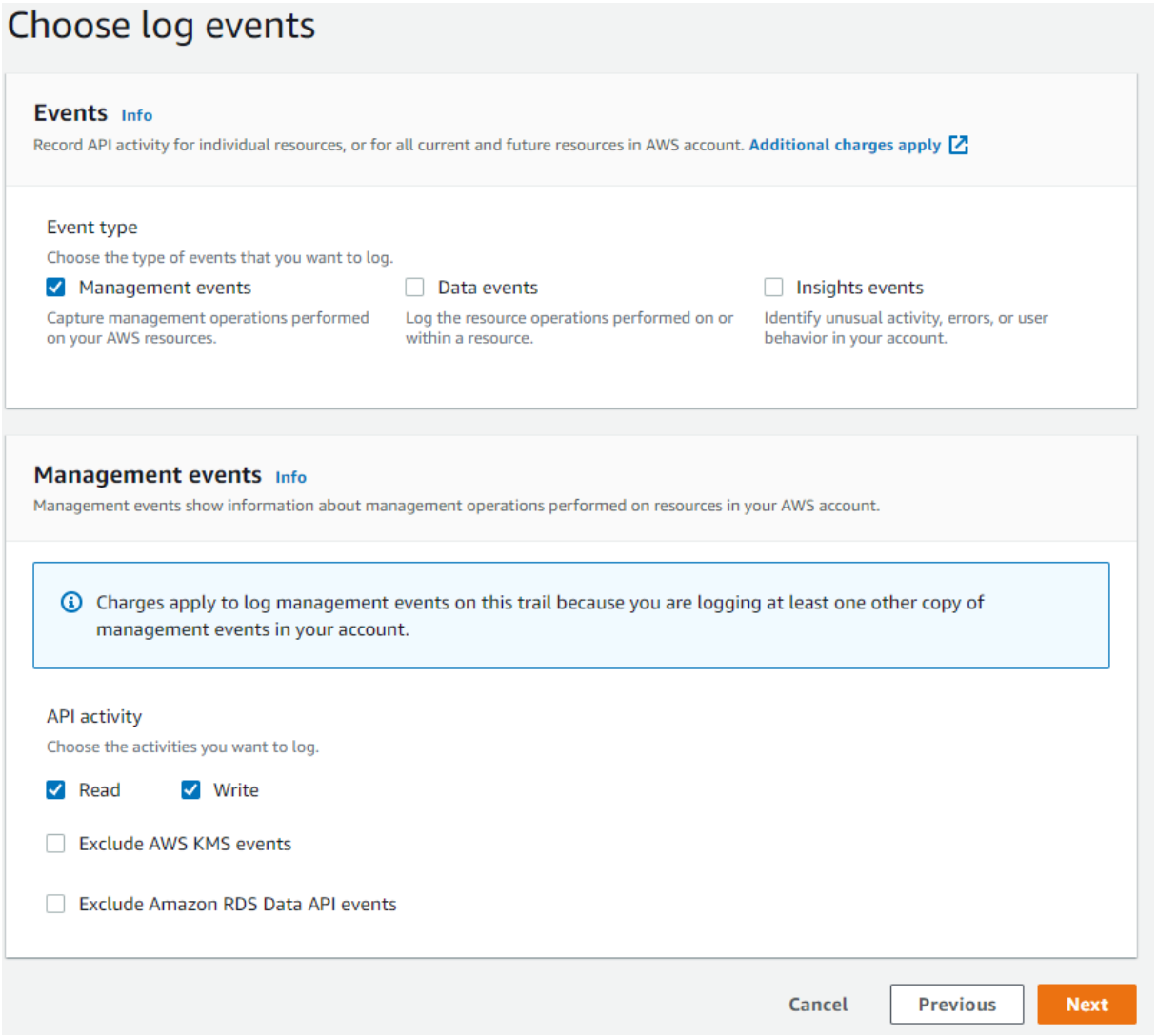
2. Dodano aliasy dla etykiet



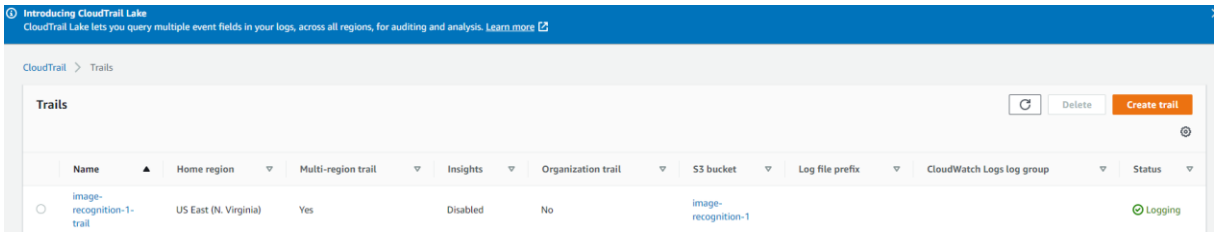
3. Nazwa Traila i wybranie atrybutów:



4. Wybranie wydarzeń do logowania:

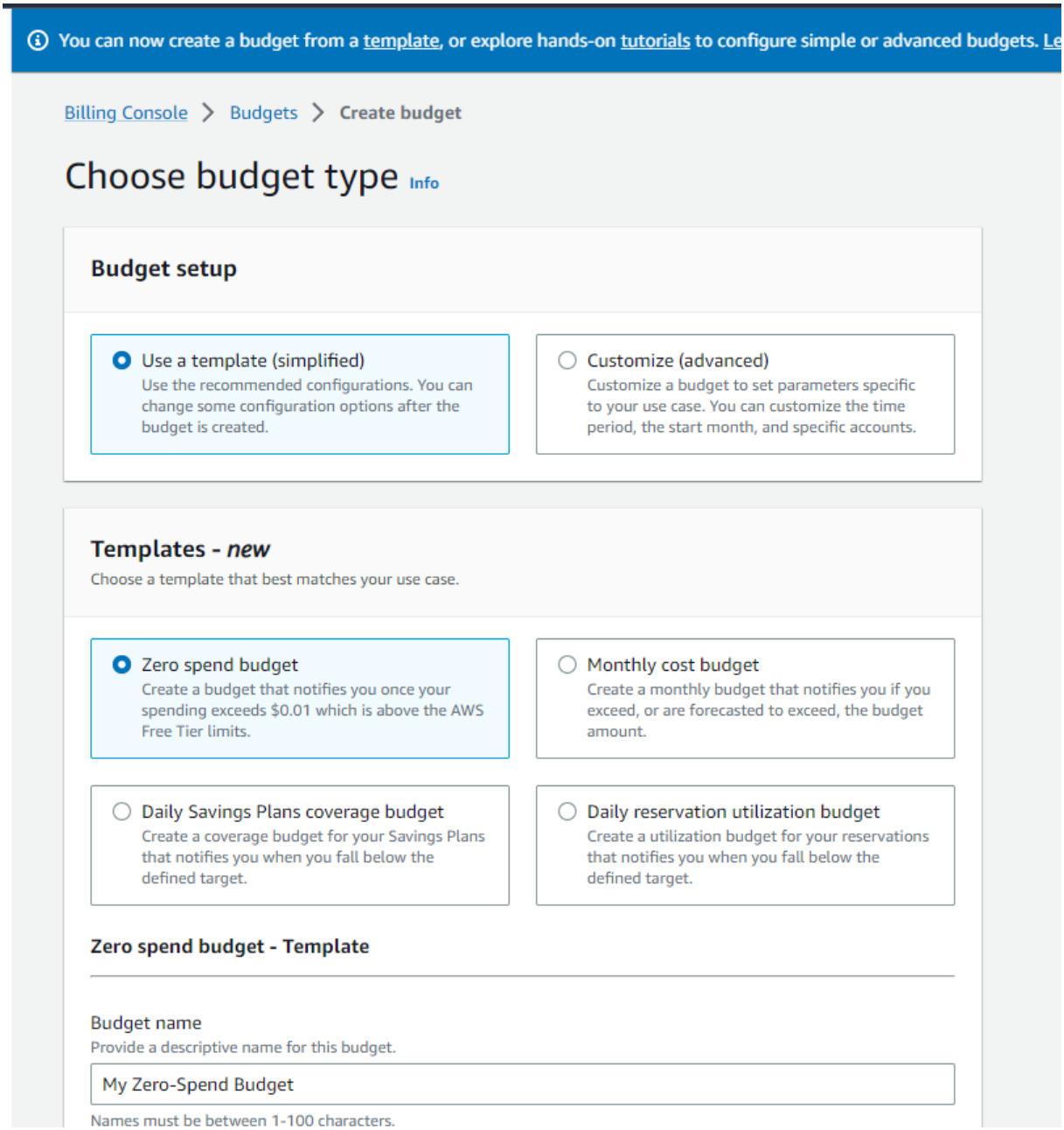


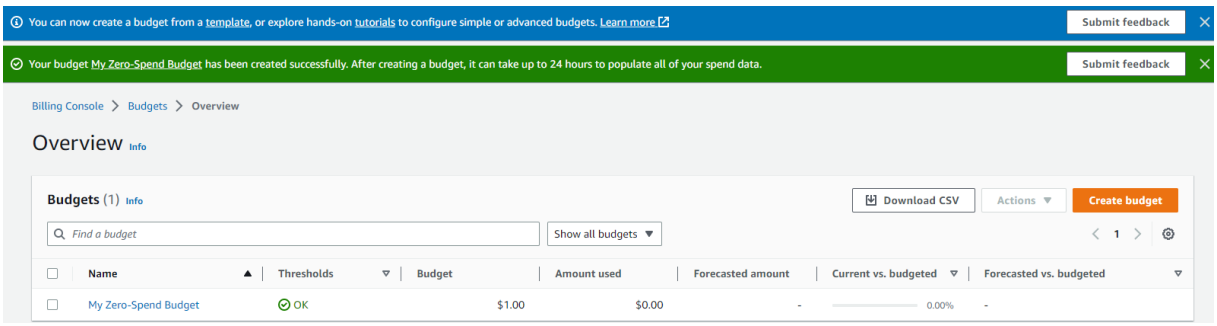
5. Utworzony AWS CloudTrail:



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.





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.

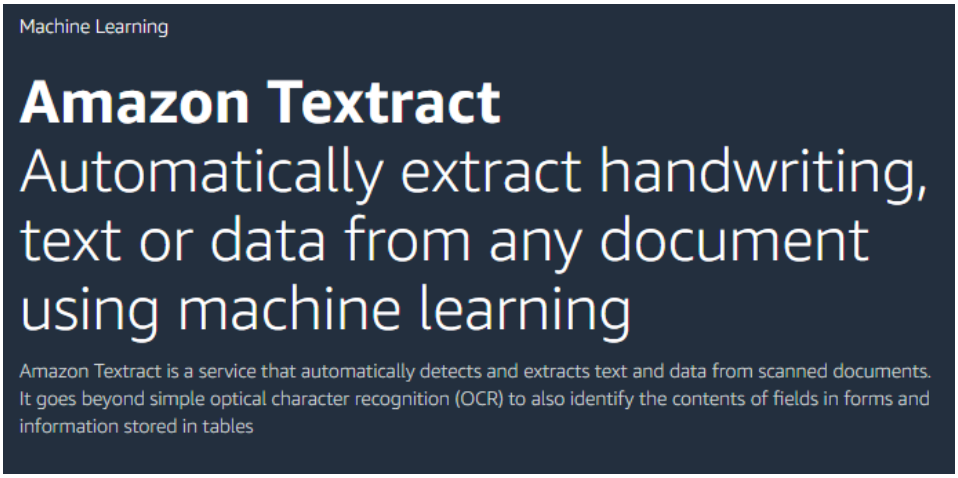
Aby korzystać z zasobów AWS skorzystaliśmy z biblioteki boto3, która jest Pyhonowym SDK dla AWSa. Biblioteka ta pozwala na bezpośrednie tworzenie, aktualizowanie, dostęp do oraz usuwanie danych bezpośrednio ze skryptów Pythonowych.

```python
import boto3

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      You, 12 hours ago • image transcription …
    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"]
                # print("This person is "+gender+". Age of this person i
                string = "This person is "+gender+". Age of this person
    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:

```python
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      You, 12 hours ago • image transcri
            }
        })
    text = ""
    for item in response["Blocks"]:
        if item["BlockType"] == "LINE":
            text = text +item["Text"] + " "
    return text
```

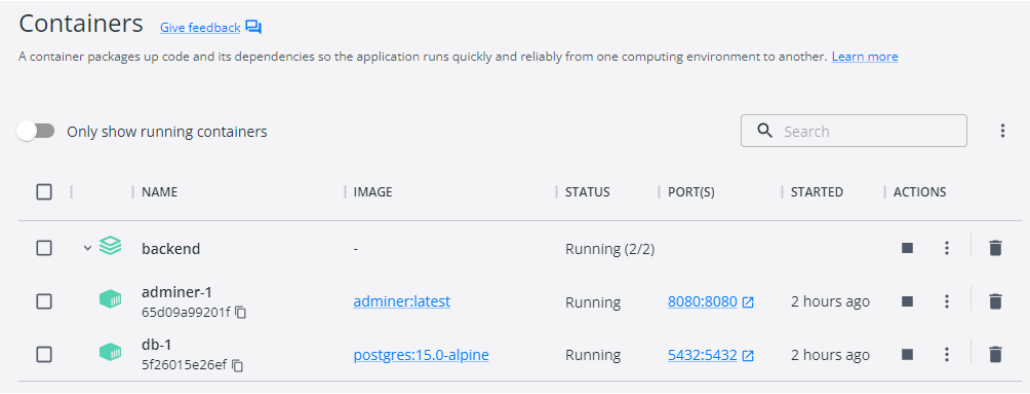
Budowa:

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

photo-upload-full-stack-master > backend > docker-compose.yaml

You, 2 weeks ago | 1 author (You)

```yaml
services:
  db:
    image: postgres:15.0-alpine
    ports:
      - 5432:5432
    restart: always
    environment:
      POSTGRES_USER: docker
      POSTGRES_PASSWORD: docker
      POSTGRES_DB: exampledb
  adminer:
    image: adminer
    restart: always
    depends_on:
      - db
    ports:
      - 8080:8080

volumes:
  db-photo-upload-service:
```



FastAPI użyto do tworzenia interfejsów Rest API:

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

Inicjalizacja + CORS middleware:

```
app = FastAPI(debug=True)
#Middleware do autoryzacji
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Web-server uvicorn na lokalnym porcie 8080:

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

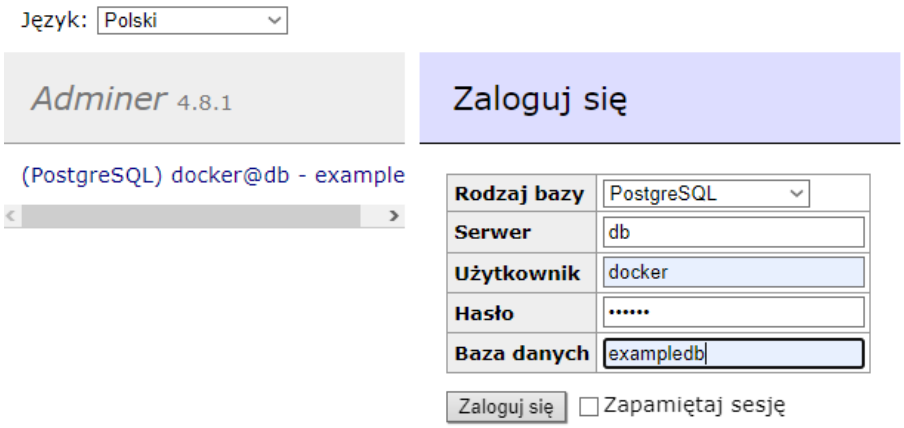
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  AZURE  GITLENS

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
```

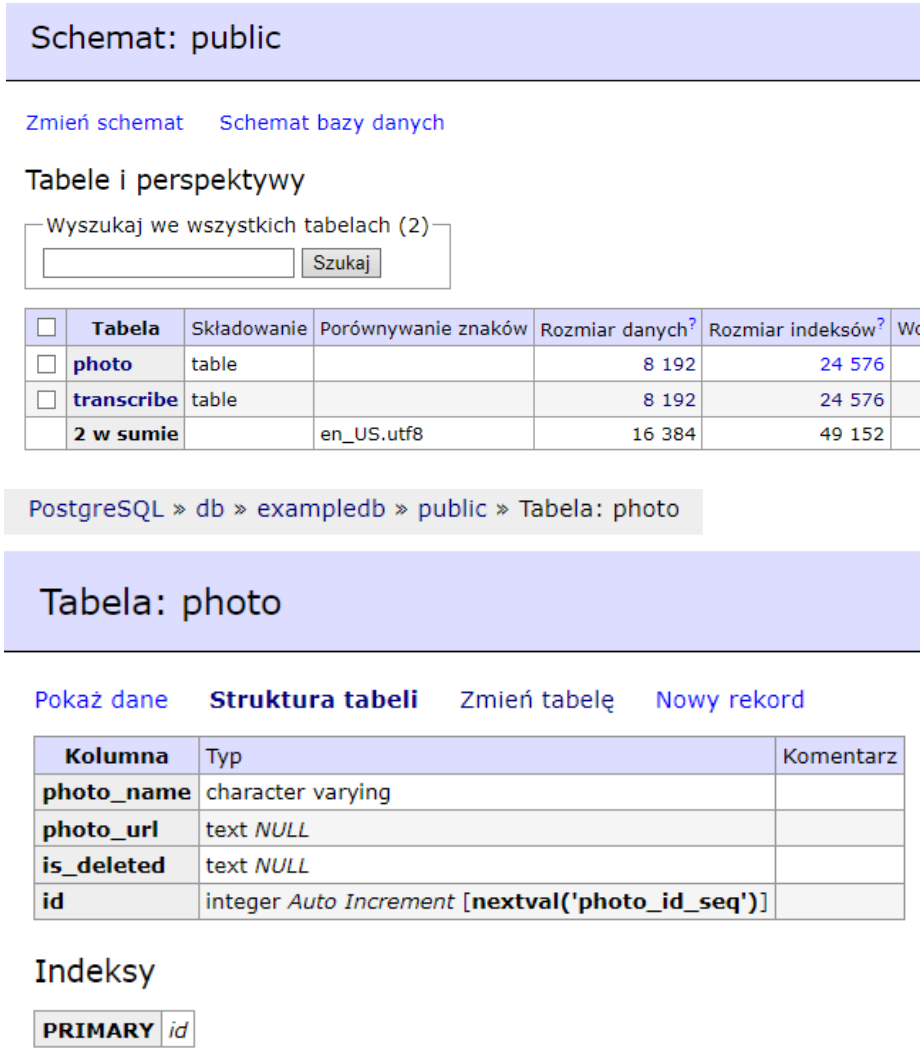
Adminer:

Narzędzie Adminer pozwala na zarządzanie bazami danych:



Dodawanie tabel do bazy:

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



Następnie wprowadzono przykładowe dane:

pokaż: photo

Pokaż dane Struktura tabeli Zmień tabelę Nowy rekord

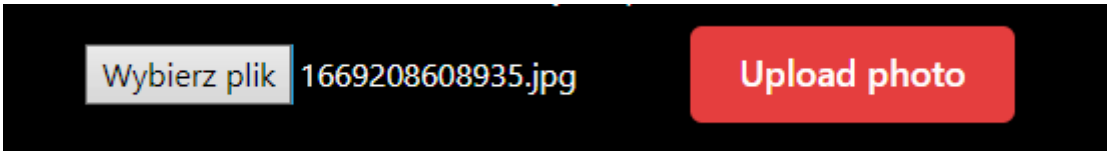
pokaż | Szukaj | Sortuj | Limit: 50 | Długość tekstu: 100 | Czynność: pokaż

```
SELECT * FROM "photo" LIMIT 50 (0.001 s)
```
Edytuj

Zmień	photo_name	photo_url	is_deleted	id
edytuj	1669208608935.jpg	https://image-recognition-1.s3.amazonaws.com/1669208608935.jpg	*NULL*	1
edytuj	Unauthorized_access.PNG	https://image-recognition-1.s3.amazonaws.com/Unauthorized_access.PNG	*NULL*	2
edytuj	zdj.jpg	https://image-recognition-1.s3.amazonaws.com/zdj.jpg	*NULL*	3
edytuj	1669209577236.jpg	https://image-recognition-1.s3.amazonaws.com/1669209577236.jpg	*NULL*	4
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 onInputChange = (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.

```python
@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="",       You, now • Uncommitted changes
    )
    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):

```javascript
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’

```python
@app.get("/photos", response_model=List[PhotoModel])
async def get_all_photos():
    # Connect to our database
    conn = psycopg2.connect(
        database="exampledb",
        user="docker",
        password="docker",
        host="localhost",
        port=5432,
    )
    cur = conn.cursor()
    cur.execute("SELECT * FROM photo ORDER BY id DESC")
    rows = cur.fetchall()
    formatted_photos = []
    for row in rows:
        print(row[0])
        photo_description = get_image_description(photo=row[0])
        formatted_photos.append(
            PhotoModel(
                id=row[3],
                photo_name=row[0],
                photo_url=row[1],
                is_deleted=row[2],
                description=photo_description,
            )
        )
        break
    print(f"{formatted_photos=}")
    cur.close()
    conn.close()
    return formatted_photos

```

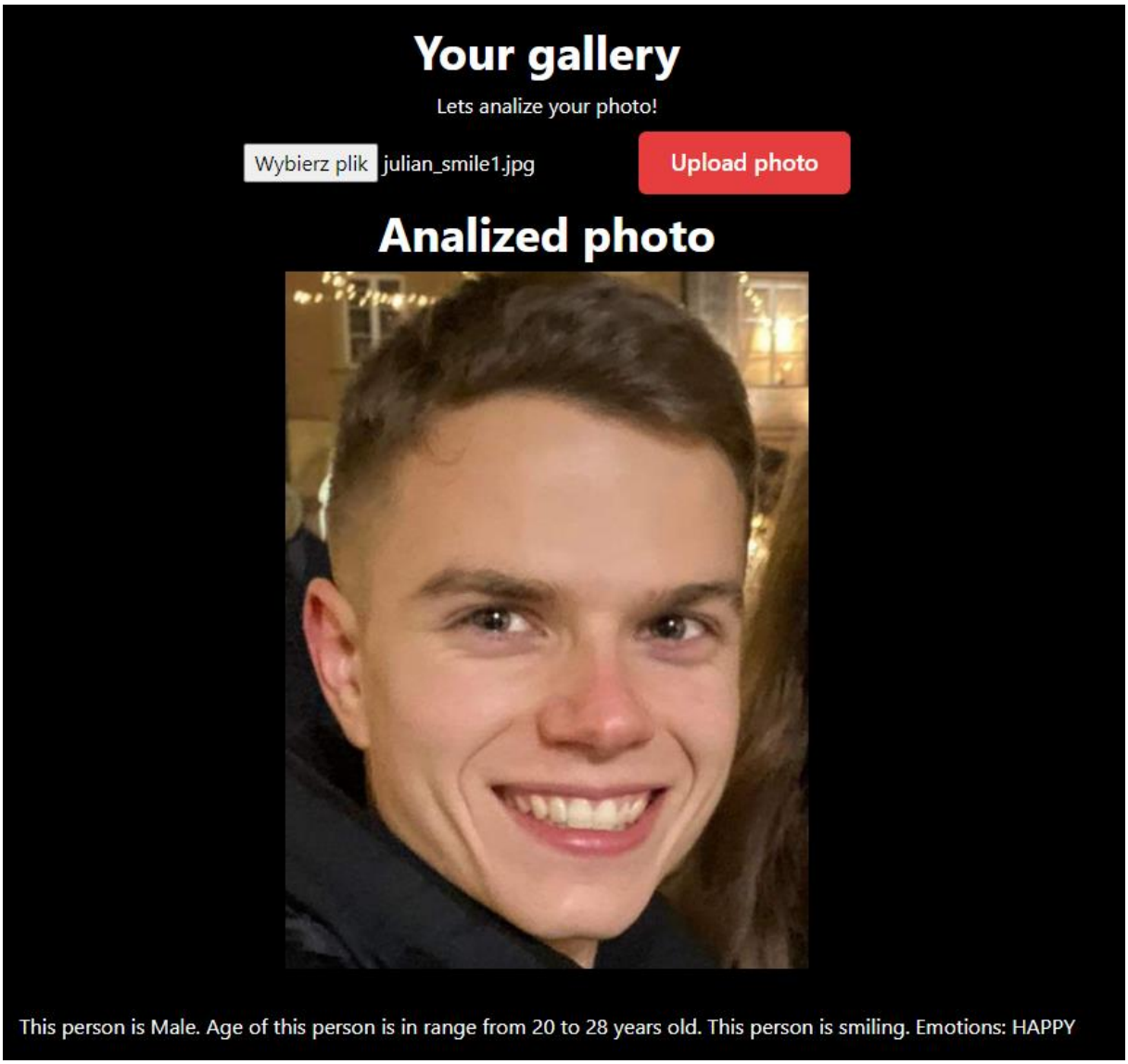
Następnie wywołuje funkcję ‘get_image_description’ w pliku orm, która łączy się z technologią AWS Rekognition:

```python
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
                )
```

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:

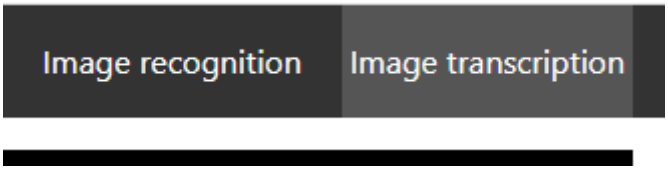
```jsx
</HStack>
<Heading>Analized photo</Heading>
<SimpleGrid columns={1} spacing={8}>
  {
    allPhotos.map(photo =>{
      console.log(photo["photo_url"])
      return(
        <Center>
          <Image width="400px" src={photo["photo_url"]} fallbackSrc="https://via.placeholder.com/150" objFit="cover">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



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):



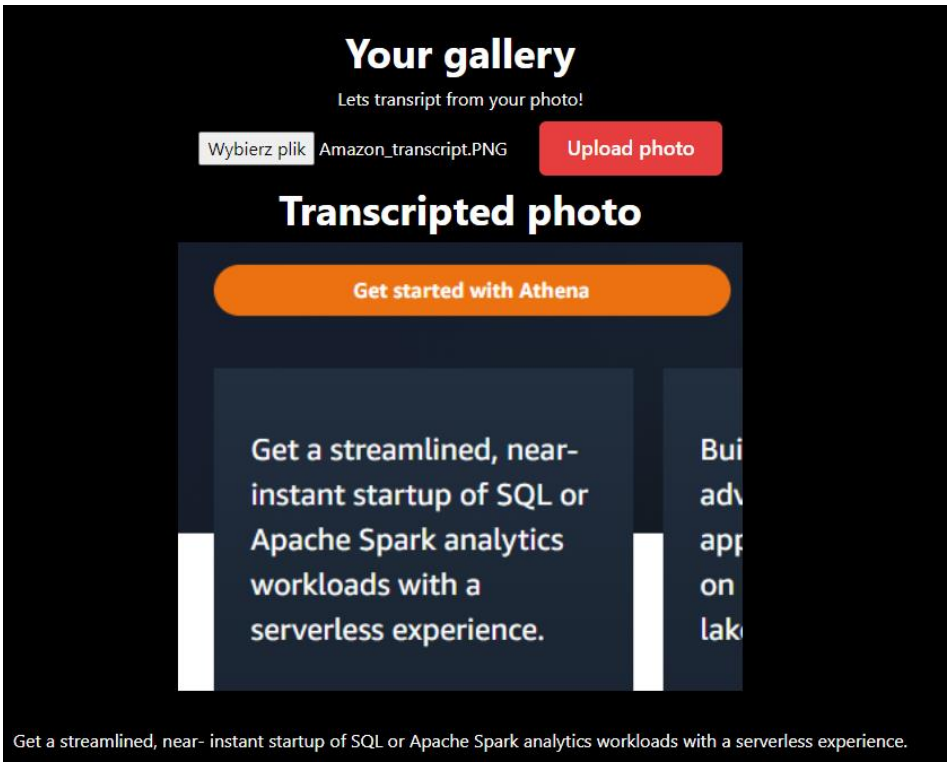
Budowa jest analogiczna do pierwszej funkcjonalności tylko korzystamy z innej tabeli postgres – transcribe oraz innego bucketu. Wywoływana jest inna funkcja - get_image_transcription, odwołującą się do innej technologii – AWS textract, który został opisany wcześniej.

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:

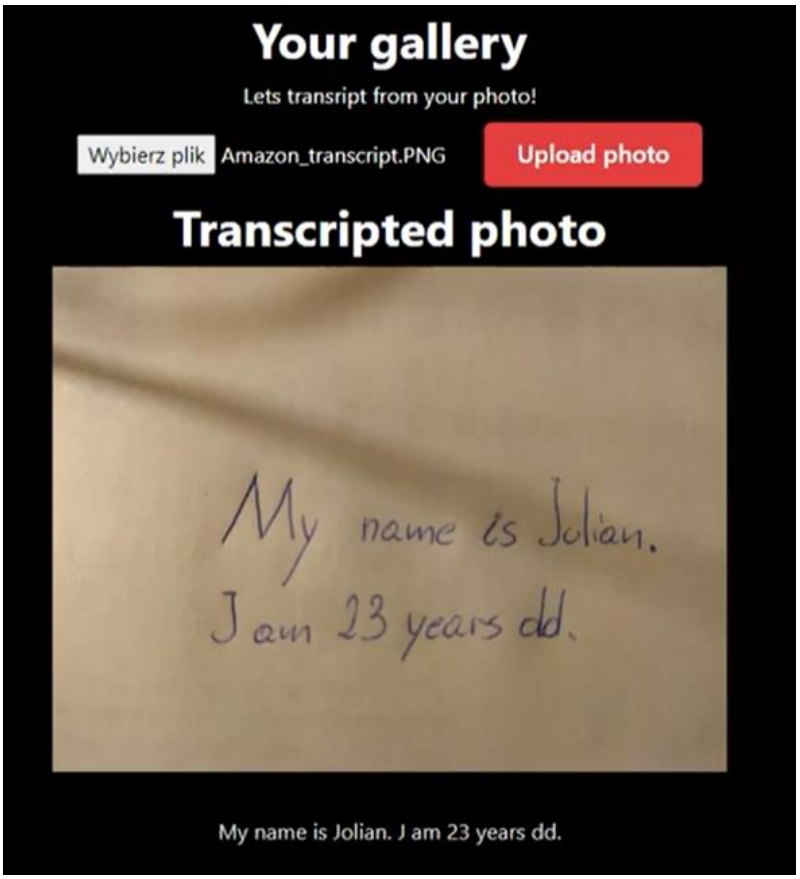
Zostało wgrane zdjęcie z bardzo czytelnym pismem drukowanym



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-ExIzZmIKRjOzM5HahtxSyPI-sQ8CP-JbJAk_GxKIk-v0wf-yrPKk - repo