Azure-pohjainen CV-analyysipalvelu

Luodes, Eemil

Sisältöluettelo

1. Ohjelman tavoite

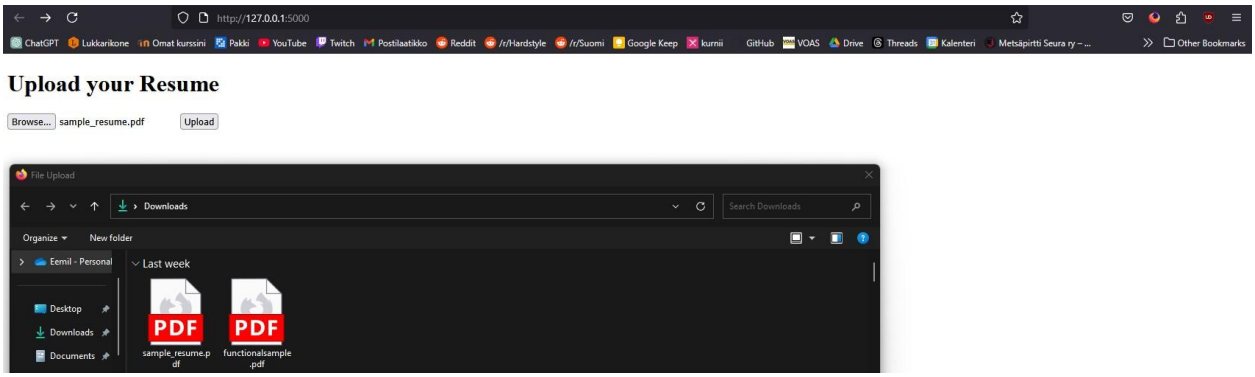
Ohjelman tavoite on luoda pilvipohjainen sovellus, joka mahdollistaa käyttäjien CV tiedostojen lataamisen verkkosivulle. Lataamisen jälkeen tiedostot tallennetaan Azure Blob Storageen ja niitä analysoidaan Azure Document Intelligence -palvelulla. Analyysin jälkeen poimittu data tallennetaan MongoDB-tietokantaan, jossa se on helposti saatavilla ja hyödynnettävissä jatkossa.

2. Ohjelman toiminta

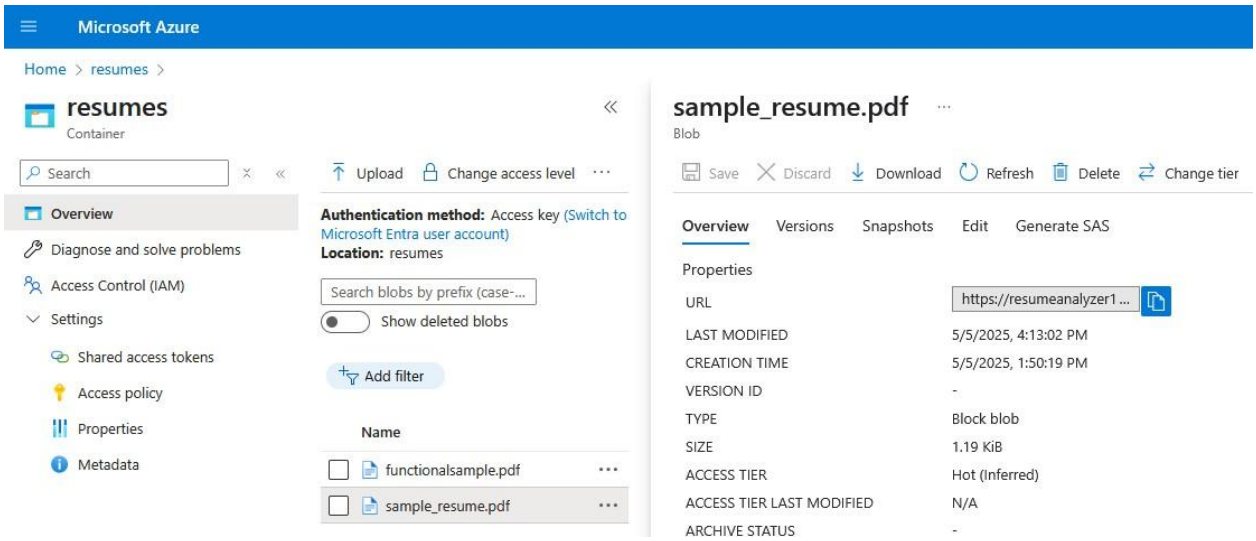
Käytetyt palvelut:

Azure Blob Storage	IaaS (Infrastructure as a Service)
Azure Document Intelligence	SaaS (Software as a Service)
MongoDB Atlas	PaaS (Platform as a Service)
Flask-sovellus	Ohjelmointi (Ei pilvipalvelu, mutta hyödyntää yllä mainittuja palveluja)

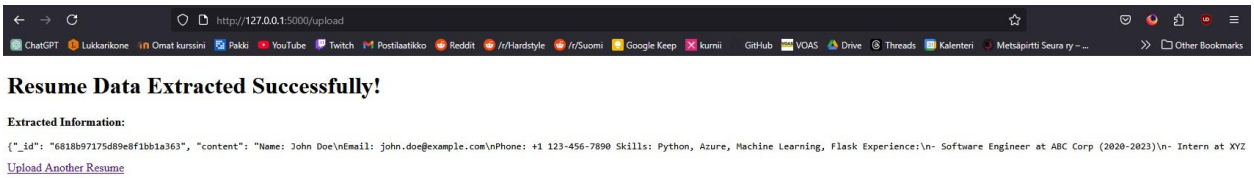
Käyttäjä lataa CV:n verkkosivulle:



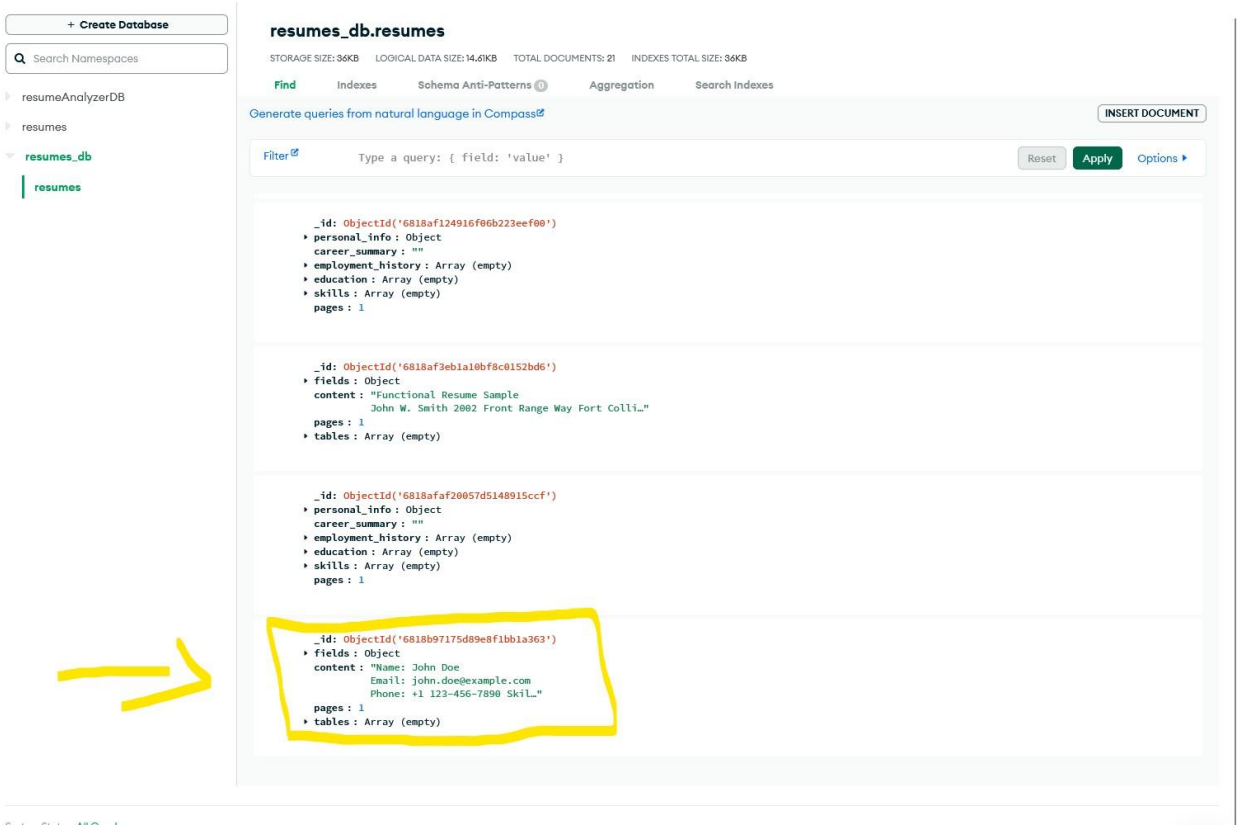
CV-tiedosto tallennetaan Azure Blob Storageen:



Azure Document Intelligence analysoi tiedoston Blob Storagesta



Analysoitu data tallennetaan MongoDB:hen



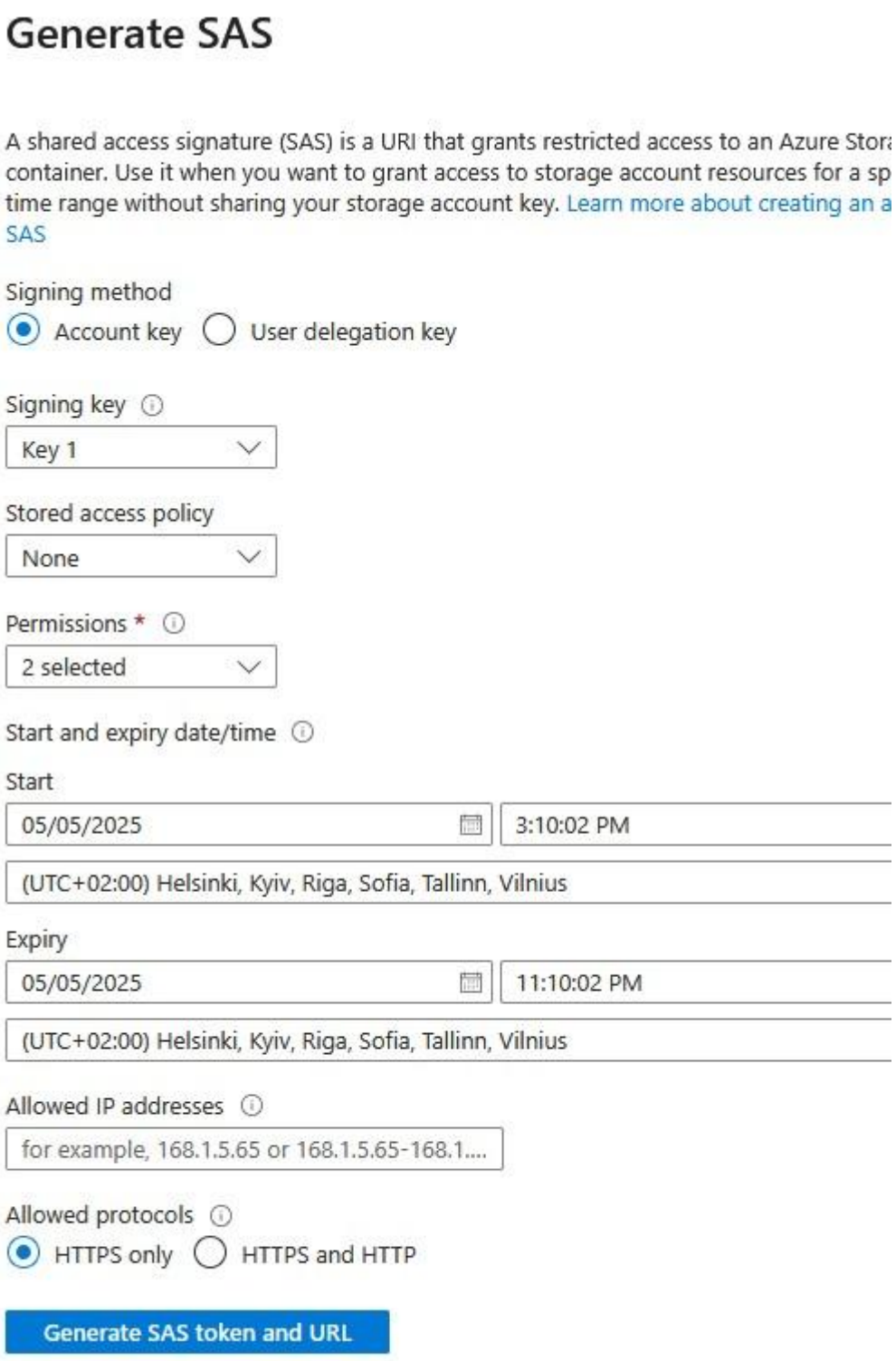
3. Projektin vaiheet

Blob Storage

Luodaan Azure Blob Storage Account.

Luodaan sisälle kontti, joka säilyttää käyttäjän lataamat CV-tiedostot.

Koska kontti on yksityinen luodaan Shared Access Signature(SAS), jonka avulla saadaan ohjelmalle pääsyoikeus konttiin.

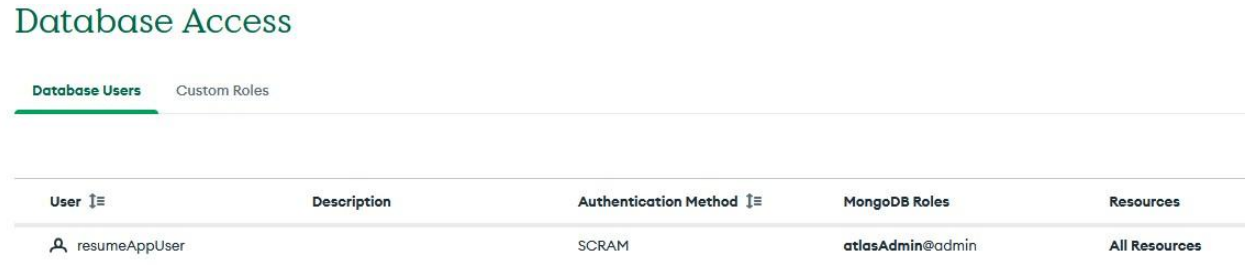


MongoDB-Atlas tietokanta

Luodaan uusi klusteri MongoDB Atlas -palveluun.

Klusteriin lisätään tietokanta ja kokoelma.

Tietokannalle lisätään käyttäjä jolla oikeudet hallinnoida klusteria.



Avataan tietokanta verkkoon.

Lomakkeen tunnistus

Luodaan Azureen Document Intelligence, jonka tehtävä on poimia data tiedostosta.

Kaikki tarpeelliset palvelut ovat nyt luotu, palvelujen yhdistämistä varten otetaan talteen:

blob connection string

container sas_token form

recognizer endpoint form

recognizer key mongoDB

connection string

Python ohjelma

Luodaan Flask sovellus, joka tarjoaa käyttöliittymän ja käsittelee logiikan käyttäjän ja pilvipalveluiden välillä.

```python
from flask import Flask, render_template, request, redirect, url_for
import os
from azure.storage.blob import BlobServiceClient          # Azure Blob Storage -asiakas
from azure.ai.formrecognizer import DocumentAnalysisClient # Azure Document Intelligence (Form Recognizer) -asiakas
from azure.core.credentials import AzureKeyCredential      # Tunnistautumiseen käytettävä avain
from pymongo import MongoClient                            # MongoDB-asiakas

# Flask-sovellus
app = Flask(__name__)

# Azure Blob Storage and Form Recognizer Setup
blob_connection_string = ""
form_recognizer_endpoint = ""
form_recognizer_key = ""
sas_token = ""

# MongoDB setup
mongo_connection_string = ""
client = MongoClient(mongo_connection_string)
db = client["resumes_db"]
collection = db["resumes"]

# --- Azure Blob Storage -client ---
blob_service_client = BlobServiceClient.from_connection_string(blob_connection_string)
container_name = "resumes"     # Blob-kontin nimi

# --- Azure Form Recognizer -client ---
form_recognizer_client = DocumentAnalysisClient(
    endpoint=form_recognizer_endpoint,
    credential=AzureKeyCredential(form_recognizer_key)
)

# --- Päänäkymä: lomake tiedoston lataamista varten ---
@app.route('/')
def index():
    return render_template('index.html')

# --- Lomakkeen POST-pyyntö tiedoston lähettämiseen ---
@app.route('/upload', methods=['POST'])
def upload_file():
    # Tarkistetaan, että tiedosto on lomakkeessa mukana
    if 'resume' not in request.files:
        return redirect(request.url)
    file = request.files['resume']
    if file.filename == '':
        return redirect(request.url)

    # Luetaan tiedosto ja ladataan se Blob Storageen
    file_data = file.read()
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=file.filename)
    blob_client.upload_blob(file_data, overwrite=True)

    # Luodaan tiedostolle SAS-URL, jota voidaan käyttää Form Recognizerin kanssa
    sas_url = f"https://{blob_service_client.account_name}.blob.core.windows.net/{container_name}/{file.filename}?{sas_token}"

    # Käynnistetään analyysi Form Recognizerin avulla tiedoston SAS-URL:illa
    poller = form_recognizer_client.begin_analyze_document_from_url("prebuilt-document", sas_url)
    result = poller.result()

    # Rakennetaan tietorakenne poimituista tuloksista
    extracted_data = {
        "fields": {},
        "content": result.content,
        "pages": len(result.pages),
        "tables": [],
    }
```

```python
    # Käydään läpi avain-arvoparit (jos löytyy) ja tallennetaan ne "fields"-kohtaan
    if result.key_value_pairs:
        for kv in result.key_value_pairs:
            key = kv.key.content if kv.key else ""
            value = kv.value.content if kv.value else ""
            extracted_data["fields"][key] = value

    # Taulukoiden kerääminen (jos dokumentissa on taulukoita)
    for table in result.tables:
        rows = []
        for row_idx in range(table.row_count):
            row = []
            for col_idx in range(table.column_count):
                # Etsitään solu riviltä ja sarakkeelta
                cell = next((c for c in table.cells if c.row_index == row_idx and c.column_index == col_idx), None)
                row.append(cell.content if cell else "")
            rows.append(row)
        extracted_data["tables"].append(rows)

    # Tallennetaan kerätty data MongoDB:hen
    insert_result = collection.insert_one(extracted_data)

    # Muutetaan ObjectId merkkijonoksi, jotta sen voi näyttää HTML-sivulla
    extracted_data["_id"] = str(insert_result.inserted_id)

    # Näytetään "onnistunut tallennus" -sivu, jossa näkyy analyysin tulokset
    return render_template('upload_success.html', data=extracted_data)

# --- Sovelluksen käynnistys ---
if __name__ == '__main__':
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Upload Resume</title>
</head>
<body>
    <h1>Upload your Resume</h1>
    <form action="{{ url_for('upload_file') }}" method="POST" enctype="multipart/form-data">
        <input type="file" name="resume" accept="application/pdf" required>
        <button type="submit">Upload</button>
    </form>
</body>
</html>
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Upload Successful</title>
</head>
<body>
    <h1>Resume Data Extracted Successfully!</h1>
    <p><strong>Extracted Information:</strong></p>
    <pre>{{ data | tojson }}</pre>
    <a href="/">Upload Another Resume</a>
</body>
</html>
```

Käytetyt Python-kirjastot:

Flask azure-storage-blob

azure-ai-formrecognizer

azure-core pymongo

dnspython python-dotenv

4. Jatkokehitysideat

Kovakoodatut merkkijonot (kuten blob_connection_string ja mongo_connection_string) on tarkoitettu vain demonstraatioon ja ne tulee korvata ympäristömuuttujilla tuotantoympäristössä.

Datan järjestely – tiedot loogisempaan muotoon. Tällä hetkellä yhdessä kasassa.