

Azure-pohjainen CV-analyysipalvelu

Luodes, Eemil

Sisältöluettelo

1. Ohjelman tavoite.....	3
2. Ohjelman toiminta	3
3. Projektin vaiheet	6
Blob Storage	6
MongoDB-Atlas tietokanta	8
Lomakkeen tunnistus	9
Python ohjelma	10
4. Jatkokehitysideat.....	13

1. Ohjelman tavoite

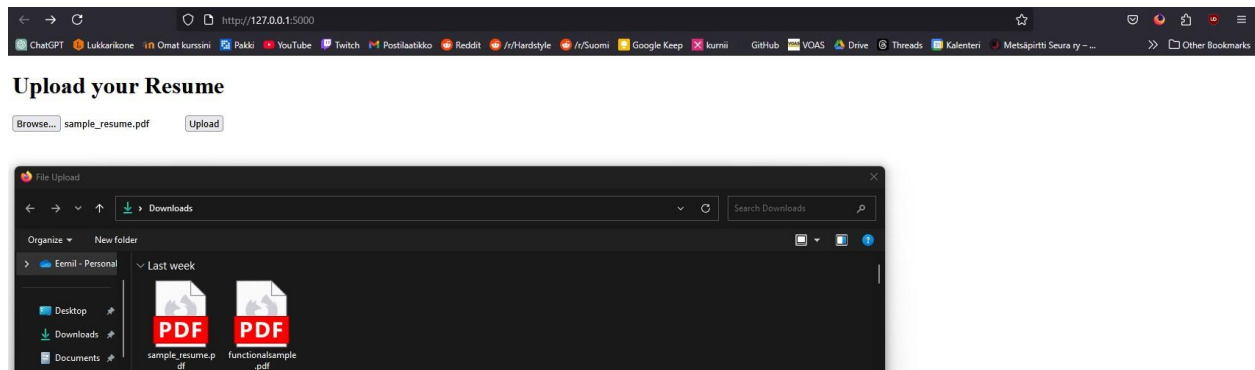
Ohjelman tavoite on luoda pilvipohjainen sovellus, joka mahdollistaa käyttäjien CV tiedostojen lataamisen verkkosivulle. Lataamisen jälkeen tiedostot tallennetaan Azure Blob Storageen ja niitä analysoidaan Azure Document Intelligence -palvelulla. Analyysin jälkeen poimittu data tallennetaan MongoDB-tietokantaan,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



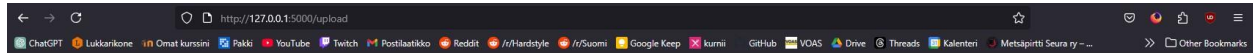
CV-tiedosto tallennetaan Azure Blob Storageen:

The screenshot shows the Microsoft Azure portal interface. The top navigation bar is blue with the 'Microsoft Azure' logo. Below it, the breadcrumb trail is 'Home > resumes >'. The main content area is divided into three sections:

- Left sidebar:** Contains the 'resumes' container name and a list of navigation options: Overview (selected), Diagnose and solve problems, Access Control (IAM), Settings, Shared access tokens, Access policy, Properties, and Metadata.
- Middle section:** Displays the 'resumes' container details. It includes an 'Authentication method' (Access key) and a 'Location' (resumes). There is a search bar for blobs and a list of blobs with checkboxes and file icons. The blobs listed are 'functionalsample.pdf' and 'sample_resume.pdf'.
- Right section:** Displays the details for the selected blob, 'sample_resume.pdf'. It includes a toolbar with actions like Save, Discard, Download, Refresh, Delete, and Change tier. Below the toolbar is the 'Overview' tab, which shows the blob's properties in a table.

Properties	
URL	https://resumeanalyzer1...
LAST MODIFIED	5/5/2025, 4:13:02 PM
CREATION TIME	5/5/2025, 1:50:19 PM
VERSION ID	-
TYPE	Block blob
SIZE	1.19 KiB
ACCESS TIER	Hot (Inferred)
ACCESS TIER LAST MODIFIED	N/A
ARCHIVE STATUS	-

Azure Document Intelligence analysoi tiedoston Blob Storagesta



Resume Data Extracted Successfully!

Extracted Information:

```
{\"_id\": \"6818b97175d89e8f1bb1a363\", \"content\": \"Name: John Doe\\nEmail: john.doe@example.com\\nPhone: +1 123-456-7890 Skills: Python, Azure, Machine Learning, Flask Experience:\\n- Software Engineer at ABC Corp (2020-2023)\\n- Intern at XYZ\"}
```

[Upload Another Resume](#)

Analysoitu data tallennetaan MongoDB:hen

resumes_db.resumes

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 14.6KB TOTAL DOCUMENTS: 21 INDEXES TOTAL SIZE: 36KB

[Find](#) [Indexes](#) [Schema](#) [Anti-Patterns](#) [Aggregation](#) [Search Indexes](#)

[Generate queries from natural language in Compass](#)

[Filter](#) Type a query: { field: 'value' } [Reset](#) [Apply](#) [Options](#)

[INSERT DOCUMENT](#)

```
{
  "_id": ObjectId("6818af124916f06b223ee00"),
  "personal_info": Object,
  "career_summary": "",
  "employment_history": Array (empty),
  "education": Array (empty),
  "skills": Array (empty),
  "pages": 1
}

{
  "_id": ObjectId("6818af3eb1a10bf8c0152bd6"),
  "fields": Object,
  "content": "Functional Resume Sample
  John W. Smith 2002 Front Range Way Fort Colli...",
  "pages": 1,
  "tables": Array (empty)
}

{
  "_id": ObjectId("6818afaf20057d5148915ccf"),
  "personal_info": Object,
  "career_summary": "",
  "employment_history": Array (empty),
  "education": Array (empty),
  "skills": Array (empty),
  "pages": 1
}

{
  "_id": ObjectId("6818b97175d89e8f1bb1a363"),
  "fields": Object,
  "content": "Name: John Doe
  Email: john.doe@example.com
  Phone: +1 123-456-7890 Skills: Python, Azure, Machine Learning, Flask Experience:
  - Software Engineer at ABC Corp (2020-2023)
  - Intern at XYZ",
  "pages": 1,
  "tables": Array (empty)
}
```

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.

Generate SAS

A shared access signature (SAS) is a URI that grants restricted access to an Azure Storage container. Use it when you want to grant access to storage account resources for a specific time range without sharing your storage account key. [Learn more about creating an SAS](#)

Signing method

☒ Account key ☐ User delegation key

Signing key ⓘ

Key 1 ▾

Stored access policy

None ▾

Permissions * ⓘ

2 selected ▾

Start and expiry date/time ⓘ

Start

05/05/2025  3:10:02 PM

(UTC+02:00) Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius

Expiry

05/05/2025  11:10:02 PM

(UTC+02:00) Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius

Allowed IP addresses ⓘ

for example, 168.1.5.65 or 168.1.5.65-168.1....

Allowed protocols ⓘ

☒ HTTPS only ☐ HTTPS and HTTP

Generate SAS token and URL

MongoDB-Atlas tietokanta

Luodaan uusi klusteri MongoDB Atlas -palveluun.

Klusteriin lisätään tietokanta ja kokoelma.



Tietokannalle lisätään käyttäjä jolla oikeudet hallinnoida klusteria.

Database Access

Database Users				
Custom Roles				
User	Description	Authentication Method	MongoDB Roles	Resources
resumeAppUser		SCRAM	atlasAdmin@admin	All Resources

Avataan tietokanta verkkoon.

Network Access

IP Access List			
Peering			
Private Endpoint			
+ADD IP ADDRESS			
You will only be able to connect to your cluster from the following list of IP Addresses:			
IP Address	Comment	Status	Actions
0.0.0.0/0 (includes your current IP address)		Active	EDIT DELETE

Lomakkeen tunnistus

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



resume-form-recognizer
Document intelligence

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.

```
1 from flask import Flask, render_template, request, redirect, url_for
2 import os
3 from azure.storage.blob import BlobServiceClient          # Azure Blob Storage -asiakas
4 from azure.ai.formrecognizer import DocumentAnalysisClient # Azure Document Intelligence (Form Recognizer) -asiakas
5 from azure.core.credentials import AzureKeyCredential      # Tunnistautumiseen käytettävä avain
6 from pymongo import MongoClient                          # MongoDB-asiakas
7
8 # Flask-sovellus
9 app = Flask(__name__)
10
11 # Azure Blob Storage and Form Recognizer Setup
12 blob_connection_string = ""
13 form_recognizer_endpoint = ""
14 form_recognizer_key = ""
15 sas_token = ""
16
17 # MongoDB setup
18 mongo_connection_string = ""
19 client = MongoClient(mongo_connection_string)
20 db = client["resumes_db"]
21 collection = db["resumes"]
22
23 # --- Azure Blob Storage -client ---
24 blob_service_client = BlobServiceClient.from_connection_string(blob_connection_string)
25 container_name = "resumes" # Blob-kontin nimi
26
27 # --- Azure Form Recognizer -client ---
28 form_recognizer_client = DocumentAnalysisClient(
29     endpoint=form_recognizer_endpoint,
30     credential=AzureKeyCredential(form_recognizer_key)
31 )
32
33 # --- Päänäkö: lomake tiedoston lataamista varten ---
34 @app.route('/')
35 def index():
36     return render_template('index.html')
37
38 # --- Lomakkeen POST-pyyntö tiedoston lähettämiseen ---
39 @app.route('/upload', methods=['POST'])
40 def upload_file():
41     # Tarkistetaan, että tiedosto on lomakkeessa mukana
42     if 'resume' not in request.files:
43         return redirect(request.url)
44     file = request.files['resume']
45     if file.filename == '':
46         return redirect(request.url)
47
48     # Luetaan tiedosto ja ladataan se Blob Storageen
49     file_data = file.read()
50     blob_client = blob_service_client.get_blob_client(container=container_name, blob=file.filename)
51     blob_client.upload_blob(file_data, overwrite=True)
52
53     # Luodaan tiedostolle SAS-URL, jota voidaan käyttää Form Recognizerin kanssa
54     sas_url = f"https://{blob_service_client.account_name}.blob.core.windows.net/{container_name}/{file.filename}?{sas_token}"
55
56     # Käynnistetään analyysi Form Recognizerin avulla tiedoston SAS-URL:illä
57     poller = form_recognizer_client.begin_analyze_document_from_url("prebuilt-document", sas_url)
58     result = poller.result()
59
60     # Rakennetaan tietorakenne poimituista tuloksista
61     extracted_data = {}
62     "fields": {},
63     "content": result.content,
64     "pages": len(result.pages),
65     "tables": [],
66 }
```

```

# 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__':

```

```

<!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>

```

```

<!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.