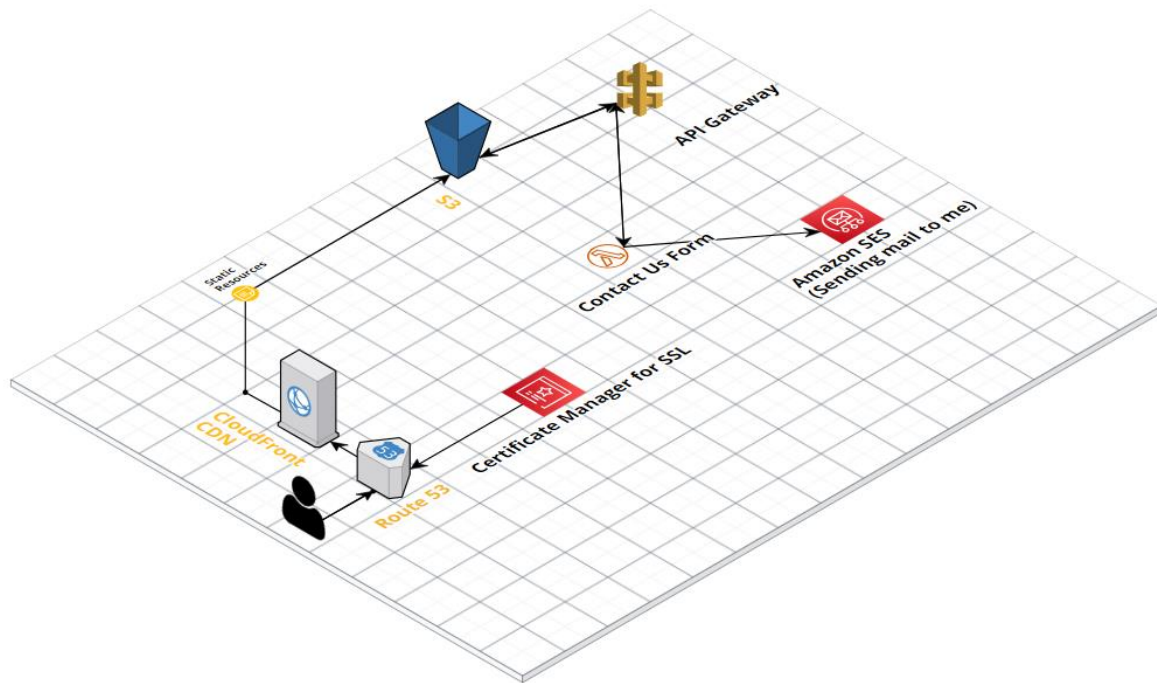


Auto Fetching Professional Website with LinkedIn

My existing professional website architecture looks like below image:



I have used Route 53 for domain registration and management, cloudfront cdn for delivery and certificate manager for SSL signature. My frontend is static and developed in next.js, built for production and uploaded into S3 bucket. Api Gateway is where my static code can call for api requests such as sending a Contact Us Form which is in my website to send an email to me containing the filled information on the form.

Using generated pdf from LinkedIn

Initially I had the idea of creating an auto fetching my resume from LinkedIn, due to the complexity and problems I will discuss later, I currently have only made a lambda code when ran, could generate data which I can plug in my static website before build to update my latest

LinkedIn automatically for my website, instead of updating every single thing on my website.

Code (lambda_function.py) looks like this and PyPDF2 is a dependency for this:

```
import json
import PyPDF2
import boto3

def endIndexOf(aStr, subStr):
    aStr = str(aStr)
    subStr = str(subStr)
    return aStr.index(subStr) + len(subStr)

def removeEmptyLines(aStr):
    aStr = str(aStr)
    return "\n".join([line.replace("\xa0", ' ').strip() for line in
aStr.split('\n') if line.strip()])

# Press the green button in the gutter to run the script.
def lambda_handler(event, context):
    # creating a pdf file object
    pdfFileObj = open('linkedinprofile.pdf', 'rb')

    # creating a pdf reader object
    pdfReader = PyPDF2.PdfReader(pdfFileObj)

    parts = []

    def visitor_body(text, cm, tm, fontDict, fontSize):
        if fontSize == 12:
            parts.append(text)

    # # printing number of pages in pdf file
    # print(len(pdfReader.pages))
    pageTxt = ""
    companiesTxt = ""
    for page in pdfReader.pages:
        pageTxt += page.extract_text()
        pageTxt = pageTxt[0:pageTxt.rindex("Page") - 3] # remove Page .. of
.. from the
        page.extract_text(visitor_text=visitor_body)

    summaryText = pageTxt[endIndexOf(pageTxt,
"Summary"):pageTxt.index("Experience")]
```

```

    experienceText = pageTxt[endIndexOf(pageTxt,
"Experience"):pageTxt.index("Education")]
    educationText = pageTxt[endIndexOf(pageTxt, "Education"):]

    companiesTxt += "".join(parts).strip("\n")
    companiesTxt = companiesTxt[companiesTxt.rindex(summaryText[-5:]) + 5:]
    companiesTxt = companiesTxt[:companiesTxt.rindex(educationText[:5]) - 5]
    # Company names
    companiesArr = removeEmptyLines(companiesTxt).split("\n") # remove empty
lines and compile companies

    summaryText = removeEmptyLines(summaryText)
    educationText = removeEmptyLines(educationText)
    experienceText = removeEmptyLines(experienceText)

    educationArr = educationText.split("\n")
    # School names
    schoolNamesArr = []
    # Degree names
    degreeNamesArr = []
    # School dates
    schoolDatesArr = []

    for i in range(len(educationArr)):
        if i % 2 == 0:
            schoolNamesArr.append(educationArr[i])
        else:
            degreeNamesArr.append(educationArr[i][:educationArr[i].index(".")].strip())
            schoolDatesArr.append(educationArr[i][endIndexOf(educationArr[i],
"."):].strip())

    experienceTextArr = experienceText.split("\n")
    companiesIndexesArr = [experienceTextArr.index(company) for company in
companiesArr]
    # Positions
    positionsArr = [experienceTextArr[companyInd + 1].replace("\xa0", '
').strip() for companyInd in
                    companiesIndexesArr]

    # Dates
    positionDateArr = [experienceTextArr[companyInd + 2].replace("\xa0", '
').strip() for companyInd in
                      companiesIndexesArr]

    # Locations
    locationArr = [experienceTextArr[companyInd + 3].replace("\xa0", '
').strip() for companyInd in companiesIndexesArr]

    expDescriptionsArr = []

    for i in range(len(companiesIndexesArr)):
        if i == (len(companiesIndexesArr) - 1):
            expDescriptionsArr.append("\n".join(experienceTextArr[companiesIndexesArr[i]
+ 4:]))
        else:
            expDescriptionsArr.append(
                "\n".join(experienceTextArr[companiesIndexesArr[i] +

```

```

4:companiesIndexesArr[i + 1]))

    # About
    aboutText = summaryText[:summaryText.index("Featured technologies
include:")]

    # closing the pdf file object
    pdfFileObj.close()

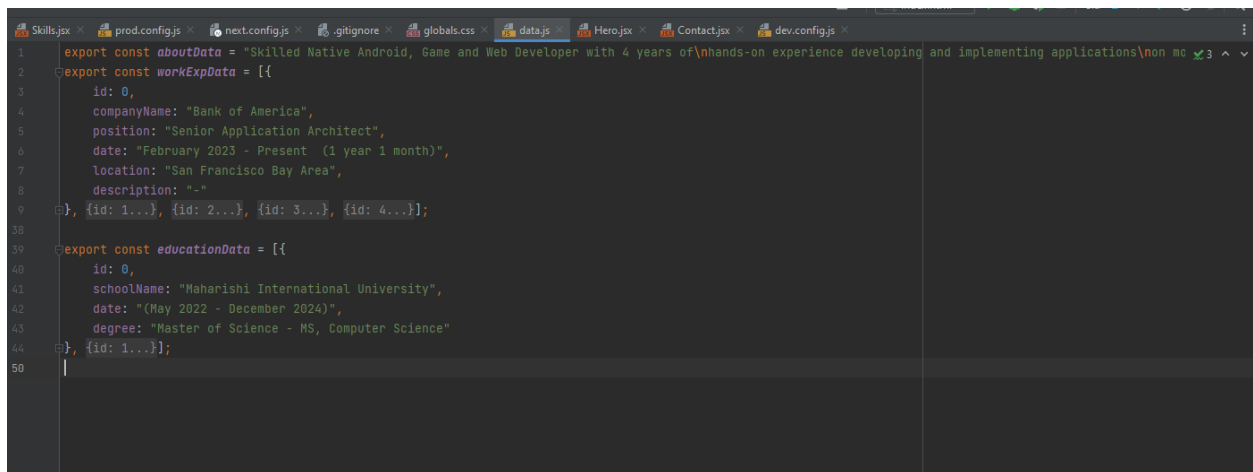
    compsFinalArr = [{
        "companyName": companiesArr[i],
        "position": positionsArr[i],
        "date": positionDateArr[i],
        "location": locationArr[i],
        "description": expDescriptionsArr[i]
    } for i in range(len(companiesArr))]
    edFinalArr = [{
        "schoolName": schoolNamesArr[i],
        "date": schoolDatesArr[i],
        "degree": degreeNamesArr[i]
    } for i in range(len(schoolNamesArr))]

    s3 = boto3.resource('s3')
    object = s3.Object('www.konuralpsenoglu.click', 'data.json')
    object.put(Body=json.dumps({
        "about": aboutText,
        "companies": compsFinalArr,
        "education": edFinalArr
    })))

    return {
        "statusCode": 200,
        "body": "Successfull!"
    }

```

When I run this code and upload my resume, it creates a json object which I can directly plug into my static website. Using lambda for storage and uploading a resume is counterintuitive for this example, however, this code at this point is not planned for production use, and only called by me for single time whenever my resume is updated, therefore storing it there it is not as bad as it looks and keeps it simple.



```
1 export const aboutData = "Skilled Native Android, Game and Web Developer with 4 years of\nhands-on experience developing and implementing applications\nnon mc 3 ^ v  
2 export const workExpData = [{  
3   id: 0,  
4   companyName: "Bank of America",  
5   position: "Senior Application Architect",  
6   date: "February 2023 - Present (1 year 1 month)",  
7   location: "San Francisco Bay Area",  
8   description: "-"  
9 }, {id: 1...}, {id: 2...}, {id: 3...}, {id: 4...}];  
38  
39 export const educationData = [{  
40   id: 0,  
41   schoolName: "Maharishi International University",  
42   date: "(May 2022 - December 2024)",  
43   degree: "Master of Science - MS, Computer Science"  
44 }, {id: 1...}];  
50 |
```

(Result)

Further possible improvements to consider

I have further possible improvements in my mind, where sadly I was not able to explore yet. The first thing would be to enable this process to be done automatically without me ever touching the code or building but instead by only providing the new pdf. For this to work I have to configure the aws s3 bucket to be built by ci/cd tools such as Amplify, so by using Amplify we can basically create build pipeline. First the code should be pulled, then this code would be executed to generate the data, then the build should happen with the new data in correct location, finally end result could be stored in the S3.

Next step to even improve it more could be having a code that can track and listen to changes on my LinkedIn and pull the latest resume pdf automatically. However this could be more difficult than it seems as I have experience creating crawlers and my own custom search engine for a course in my bachelor's degree.

Different approaches that could be taken and important points to consider!

There are actually many different options for this idea to work, the other best options I can see are crawling the website or using the official LinkedIn Api.

Let's start with first option, we can create a web crawler using python libraries, which have tons of options for crawling like BeautifulSoup, Selenium etc. Where we can navigate to a page, send user events like clicks and filling inputs where necessary, and of course save information from the page by navigating through dom and selecting/formatting correct elements.

We can implement this idea by using Lambda Layers running on Python, possibly DynamoDB for storage, also we can have a Lambda function where it checks if any new update or updates the page regularly to call this other function to crawl and update the website. We can then pull data from DynamoDB directly, but this just adds more complexity so, for having less calls, we can again use Amplify and rebuild the project with new data fed into it. Since we are not gonna change our LinkedIn like a madman, even than this should suffice.

Problems with first option are, we have to login to linkedin and provide a login information, where we can keep safely as environment variables, but site may block crawlers, even if you get past that, stuff such as captcha blocks bots/crawlers actions will definitely be almost impossible. Also sites might not allow legally for their sites to be crawled due to possible legal problems.

Second option is using LinkedIn Api, which is the best option of all, since it is much easier to call a linkedin api every once in a while through Lambda and update the website again in similar fashion. Here we don't

have to worry about legal implications, too much extra work for finding/formatting information, also no need to bypass captcha etc, and only keep API key as environment variables.

Murat Konuralp Senoglu

614610