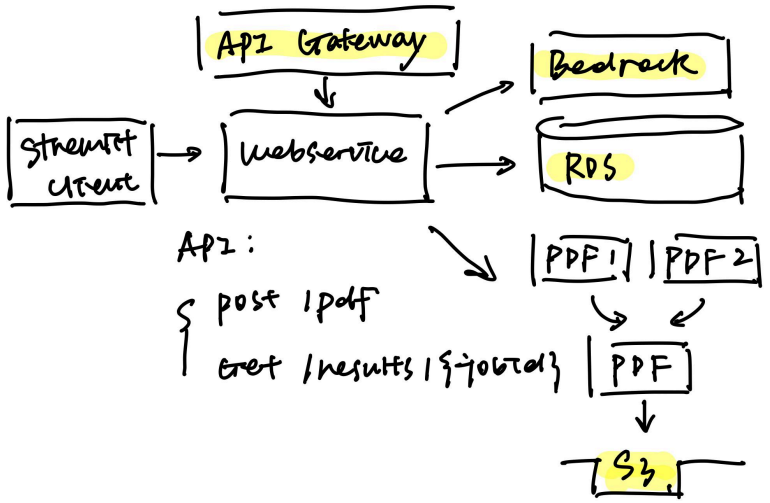
Resume Hero

Yuanrui Jerry Zhu

Overview

I would like to build a resume enhancement web application where users can upload resume and job descriptions and get feedback. The feedbacks include:

- Level of match in key words and compute a rated score in 0-100
- The way to improve the resume to fit the job description most
- A strong cover letter based on the job description



AWS Components

- API gateway
- Lambda
- RDS
- Bedrock

Database

Job table

jobid	bucketkey	ratingbucketkey	advicebucketkey	letterbucketkey
int	varchar(256)	varchar(256)	varchar(256)	varchar(256)

Lambda Functions and APIs

Upload [POST /pdf]

- Upload PDF to S3 bucket.
- Payload contains resumedata and descriptiondata in base64 strings.
- The server side decodes the data, combines and writes on a single file using PyPDF2.
- Generate the bucketkey with uuid.
- Insert bucketkey into the Job table (automatically create a new jobid at the same time).
- Upload PDF to the bucketkey specified.
- Return jobid.

Compute

- Automatically triggered when a new PDF is uploaded to the database.
- Download PDF from the bucketkey.
- Create 3 new text files which store ratings, advice, and cover letter respectively.
- Extract text strings from PDF.
- Define 3 prompts that take in the text strings and produce ratings, advice, and cover letter respectively, the prompts have been crafted carefully to produce a response as informative as possible (response is strictly in json format specified in the prompt).
- Use concurrent.features to submit the 3 concurrent requests to Bedrock to ensure we have a response in a timely manner (LLaMA-3.1-8b-instruct was used since it is a lightweight yet powerful and stable model).
- The responses are stored in corresponding text files and passed into S3.
- Job table is updated with 3 new bucketkeys of corresponding response text files.

Download [GET /results/{jobid}]

- Using the jobid we received from the upload function, query the Job table to check if all bucketkeys are ready.

- If not, send a status code of 202 specifying we are still processing files.
- If yes, read the text files and store them locally, compile them up to a single dictionary with 3 fields and send them back.
- Return jobid, status, and results dictionary.

Client

- The client calls endpoints defined by API using Lambda functions
- After uploading the PDFs, a jobid will appear, and users are notified the computation is in progress.
- The client calls the endpoint to pull a response periodically (5 seconds).
- After the response is ready, the client side converts the strings to correct structures and shows them in an elegant way.
- The client also has an option to enter a job id and directly get previously uploaded and computed results.