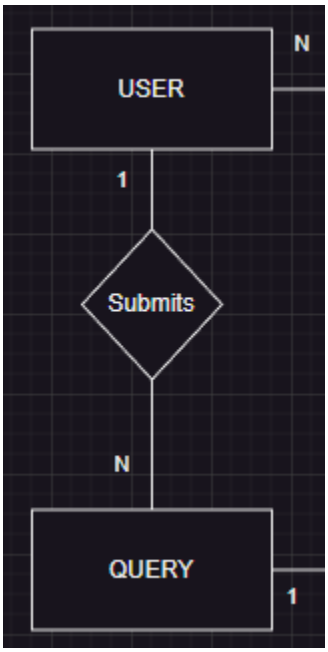
Vedran Deskovski

AI Internship task

By TeamLift

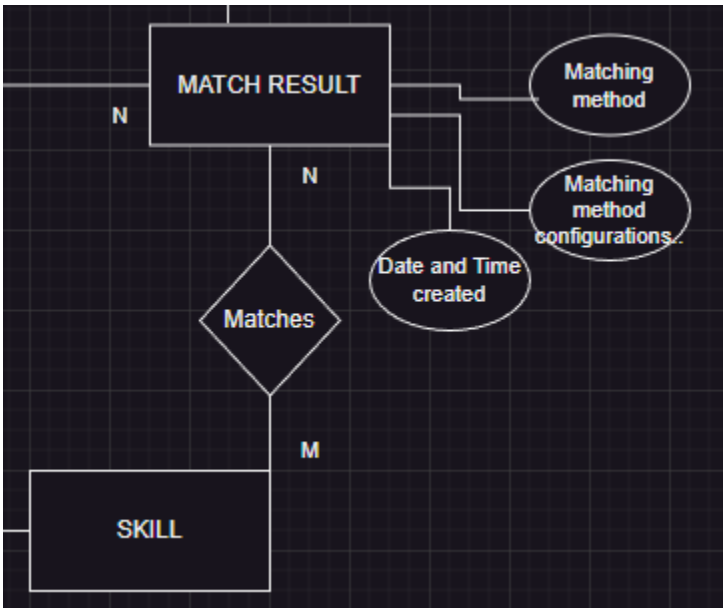
I. ER diagram

One USER can submit multiple queries and a query can be submitted by one USER:



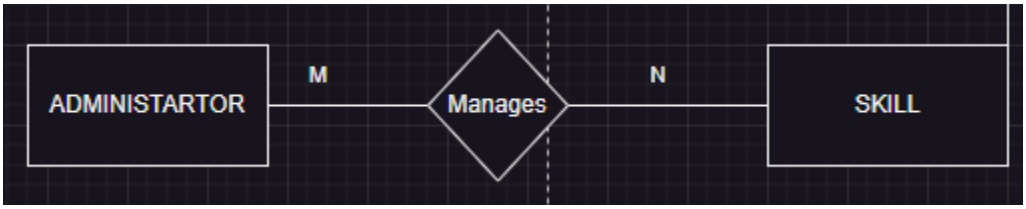
One QUERY can have multiple MATCH RESULTs, but only one MATCH RESULT corresponds to only one QUERY:

A MATCH RESULT can match multiple SKILLs and a SKILL can be matched in multiple MATCH RESULTs:



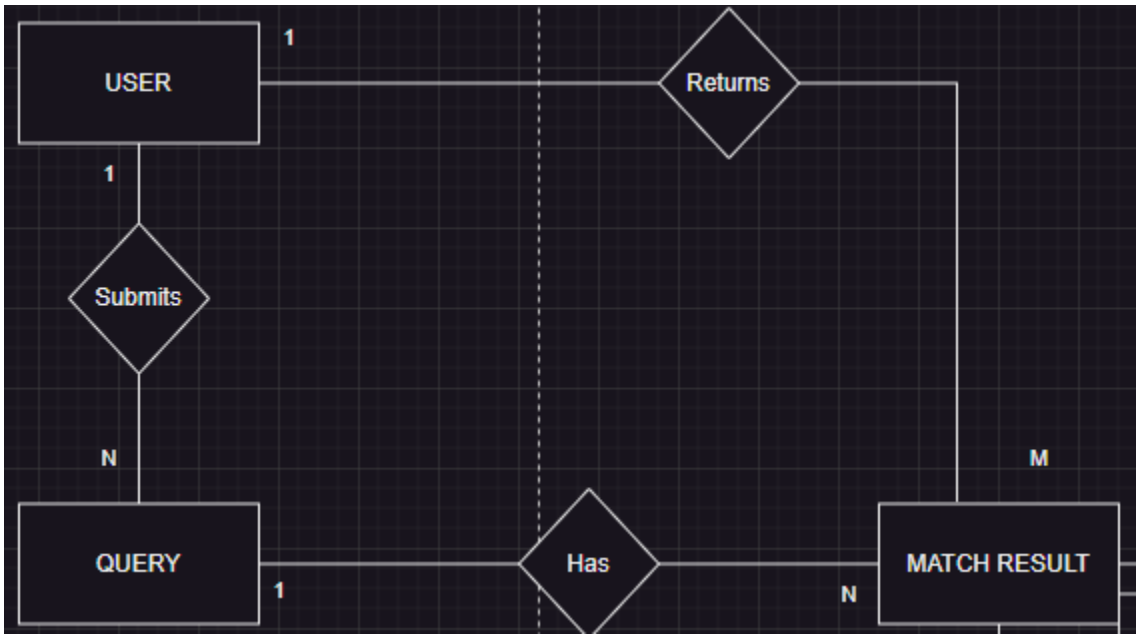
In the MATCH RESULT entity I store the method/algorithm to match the query to the administrator's list, its configuration and the date and time of when the result has been created.

The ADMINISTRATOR can manage (create, edit, delete) multiple SKILLs and a SKILL can be managed my many ADMINISTRATORs:



Depending on the project, a single SKILL can be managed by only one ADMINISTRATOR. My thought process was so that many administrators would log in into the system so they can share/divide their workload.

One MATCH RESULT can return to one USERs and one USER can receive multiple MATCH RESULTs:



For the returned data structure to the USER, I chose a list of dictionaries. Each dictionary has the following key value pairs: <String, Double>.
String for the name of the skill and a double for the score.

II. Python API

```
app = FastAPI()

ADMIN_LIST = ["Python", "relational database", "Software engineering",
"data science", "NLP", "natural language processing"]

def find_matches(query: str):
    query_words = set(query.lower().split())
    match_results = []
    for skill in ADMIN_LIST:
        skill_words = set(skill.lower().split())
        match_score = len(query_words & skill_words) / len(skill_words)
        match_results.append({"skill_name": skill, "match_score":
round(match_score,2)})
    match_results = sorted(match_results, key=lambda x: x["match_score"],
reverse=True)
    return match_results
```

```python
@app.post("/query")
async def submit_query(query: str = Form(...)):
    match_result = find_matches(query)
    return JSONResponse(content={"query": query, "match_results": match_result})


if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8081)
```

I created a simple API that accepts string (the query) and returns a JSONResponse holding the query and the match results. For the results, I firstly split the query into words, then I count how many times each word occurs into each of the items in the administrator's list. I take that count and I divide it by the number of words in that list item.

An example:

If input is "data analyst", the response would be:


```
{
  "query": "data analyst",
  "match_results": [
    {
      "skill_name": "data science",
      "match_score": 0.5
    },
    {
      "skill_name": "Python",
      "match_score": 0
    },
    {
      "skill_name": "relational database",
      "match_score": 0
    },
    {
      "skill_name": "Software engineering",
      "match_score": 0
    },
    {
      "skill_name": "NLP",
      "match_score": 0
    },
    {
      "skill_name": "natural language processing",
      "match_score": 0
    }
  ]
}
```


We notice a score of 0.5 for "data science". That is because we have length of the intersection (1) divided by the length of the skill name (2).

Now, this is a fairly easy implementation of the given task. If it were a big project, I would imagine the following:
The administrator's list would be a kind of dictionary of known words. Each of the list items I would have to tokenize (convert them into vector of integers) and then I would take every single vector and embed them into a dimension (for example, 512). These vectors can be inserted into a vector database (Pinecone) of a given dimension (512 in this case). When a user inputs a query, that same query gets embedded with the same embedding model we used for the dictionary, then gets mapped onto the vector database and by some metric (cosine similarity) we would retrieve the vectors that are most similar with the user's query.

For this specific task I used ChatGPT in order to optimize the 'find_matches' function in my API. I was not aware that we can retrieve the intersection of two sets easily by '&' symbol. In my previous code I had a nested 'for' loop that iterates through the query words and each skill and then I had to manually count their frequency (big time complexity).

As for my daily ChatGPT activities, I use it for:

- Code explanations: if the code is bloated or it uses unknown keywords that I have not seen before (like '&' in this case), it can be useful to further my knowledge by having it explain it to me.
- Giving examples: I am a person that learns through examples. Whenever a topic is too vague or a theory is barely scratching the surface of the point that is trying to make, I use ChatGPT to give me a real world example of the problem so that I can gain insight and have a better understanding of the topic.

I made an account on Teamo and I noticed it's like an AI hub. Based on the person's daily activities, they get recommended various models and can manage which models they want to work with. It can be very resourceful when you are trying to navigate through your daily tasks.