Case Study

The Reinforcement Learning Specialist just created an agent cabable to provide the best three offers from a list of offers according to each customer's specifications/states and now we need to create an RestAPI to serve this agent to our clients.

The API needs 3 endpoints:

> Method: **GET**

> URI: /is-alive

> Response : `{ “status”: “I’m alive and running”, “agent-version”: f”{the_most_recent_version}-API”}`

Example Response:

```
{ “status”: “I’m alive and running”, “agent-version”: “1.0.0-API”}
```

> Method: **POST**

> URI: /predict

> Body: `{ “customer_id”: string, “timestamp”: date, “states”: string-dict, “offers”: string-dict}`

> Response: `{ “customer_id”: string, “response-date”: f”{todays date}”, “best-three-offers”: list}`

Example body:

```
{ “customer_id”: "152467”,
“timestamp”: “2022-01-01",
“states”: “{“age”: 18, “gender”: “M”, “client-since”: “2020-01-01", “region”: “Europa”, “last_offer”: “OFFER_5” }” ,
“offers”: “{“OFFER_1”: True, “OFFER_2”: True, “OFFER_3”: False, “OFFER_4”: True, “OFFER_5”:True, “OFFER_6”: False, “OFFER_7”: False, “OFFER_8”: True, “OFFER_9”: False, “OFFER_10”: True}”
```

Example Response:

```
{ “customer_id”: “152467”, “response-date”: “2022-04-05", “best-offers”: "
{'OFFER_2', 'OFFER_1', 'OFFER_3'}"}
```

> Obs: Notice that it only considered the offers with the flag **True**, which means this client has permission to see these offers (if the customer has an IPHONE 10 doesn't make sense to offer an IPHONE 9) and also don't return repeted offers.

> Method: **PUT**

> URI: /rewards

> Body: `{ "customer_id": string, "timestamp": date, "accepted_one_of the_three_offers": bool }`

> Response: `"Sucess"`

Example Body:

```
{ "customer_id": "152467", "timestamp": "2022-04-05", "accepted_one_of
the_three_offers": True}
```

Each endpoint will have auxiliar tables to read and/or write from:

- TB_AGENTS (/is-alive)
- TB_EPISODES (/predict)
- TB_REWARDS (/rewards)

TB_AGENTS

- ID (INTEGER) SEQUENCE
- VERSION (VARCHAR) NOT NULL
- LAST_UPDATE (DATETIME)

The TB_EPISODES

- ID (INTEGER) SEQUENCE
- CUSTOMER_ID (VARCHAR) NOT NULL
- STATES (VARCHAR) NOT NULL
- PREDICTED_OFFER_1 (VARCHAR) NOT NULL
- PREDICTED_OFFER_2 (VARCHAR)
- PREDICTED_OFFER_3 (VARCHAR)
- LAST_UPDATE(DATETIME)

- UP_TO_DATE(DATETIME) NOT NULL

TB_REWARDS

> Obs: **customer_id** must match with one from **TB_EPISODE**:

- ID (INTEGER) SEQUENCE
- CUSTOMER_ID (VARCHAR) NOT NULL
- ACCEPTED_OFFER (BOOL)
- LAST_UPDATE(DATETIME)

Here's the agent that will make the decisioning, you can just copy and paste into your code and call this method.

```
    def rl_agent(customer_state, offers):

        from random import choices as rl_agent_decisioning
        weights = []

        print(f"logging customer_state age: {customer_state['age']}" )
        if customer_state["age"] < 30:
            weights = [10, 8 ,6, 6, 6, 4]
        else:
            weights = [8 ,6, 10, 6, 4]

        print(f"logging customer_state region: {customer_state['region']}")
        if customer_state['region'] == 0:
            weights = [10, 8 ,6, 6, 10, 10]
        elif customer_state['region'] == 1:
            weights = [4, 8 ,6, 6, 10, 10]
        elif customer_state['region'] == 2:
            weights = [4, 10 , 10, 6, 4, 4]
        else:
            raise Exception(f"weight by region not applied, is categorical data ?")

        print(f"logging customer_state gender: {customer_state['gender']}" )
        if customer_state['gender'] == 0:
            weights = [10, 10 , 10, 6, 4, 4]
        elif customer_state['gender'] == 1:
            weights = [4, 4, 6, 10, 10, 10]
        else:
            raise Exception(f"weight by gender was not applied, is categorical data
?")

        return rl_agent_decisioning(offers, weights=weights, k=3)
```

With our API working and ready we can test with client's data

1. **Company_A** is very mature when comes to data and AI and knows how to handle rest requests, doesn't need further help from us, they only need the endpoint; You can test your API using the **company_A.json**.

--

2. **Company_B** knows nothing about requests but they do have a CSV file that they sent to us with all their customer's data (**company_B.csv**) and would be very pleased if our API could deliver a CSV with the results; After some analyses we notice that all OFFERS are True, which doesn't make sense, some offers should be False, we ask them about it and they said **customers in Europa can't have OFFERS 4,5 and 6** and **customers age above 65 can't have OFFERS 1,3 and 4**; So with the API working we need a module (**batch_process.py** for example) that reads this CSV, apply those rules, process each line to be in the format the API expected and deliver a new CSV with the results.

--

3. **Company_A** really liked our API and now they trust our product, they requested us if it's possible to, instead just the offer's number (ex: **{'OFFER_2', 'OFFER_1', 'OFFER_3'}**) in the response, deliver the name of the product of that offer to show up in real time in their website (ex: **[{"OFFER_4": "IPHONE 13 MAX", "OFFER_1": "IPHONE 13", "OFFER_10": "SAMSUNG NOTE 10"}**], so they sent to us this **company_A_products.csv** file which we'll save into a table called **TB_PRODUCTS_A** and we'll fetch those names from it each request they made. It's possible to create a **/predict-v2** for them?

    - Method: **POST**
    - URI: /predict-v2
    - Body: `{ "customer_id": string, "timestamp": date, "states": string-dict, "offers": string-dict}`
    - Response: `{ "customer_id": string, "response-date": f"{todays date}", "best-three-offers": string-dict}`

    Example Response:

```
  { "customer_id": "152467", "response-date": "2022-04-05", "best-
offers": "{'OFFER_2': 'Iphone 13', 'OFFER_1': 'Iphone 13 MAX',
'OFFER_3: 'Iphone 10'}"}
```

- TB_PRODUCTS
  - ID (INTEGER) SEQUENCE
  - OFFER_ID (VARCHAR)
  - PRODUCT_NAME (VARCHAR)

You can do it any way you want but if possible we would really like if you use:

--

- FastAPI as the Python framework
- Docker to be it’s infrastructure (one container for the API and another for the Postgres db)
- Documentation explaining how it works or Swagger
- If you want to use some cloud provider I would suggest Heroku as PaaS (and free) or GCP as IaaS (first $300 for new accounts)