Cluby

Coding Assignment

Version 2022.10.07 BE1

The goal for this assignment is to create a few API endpoints for an imaginary membership platform.

Requirements

Working Python development environment with PostgreSQL client libraries is required for this coding assignment. Newer Python versions (e.g. 3.9 or 3.10) are preferred but not mandatory.

You have to use a live development database provided by us, so internet connection is also a must. Development database is using PostgreSQL 13.7.

Credentials

Your **personal private** assignment database credentials are:

Hostname: `daed21f7cebd3ef828aead846e9851ad.db.cluby-assignment-api.com`
Port: `41357`
Username: `user_42be40e2395c63853b4810d28f69a9f1`
Password: `1a8d3900a07ab757634b20b906480f6422aed4c05714a7d437791fdd8a29cdb5`
Database: `cluby_assignment_2104c465e402abe36b11b895fc651ffb`

Even though you cannot do much damage with the credentials and those will automatically expire after some time, you should not store the credentials to any public or private source code repository etc. Use separate configuration files, environment variables, command line arguments, etc to pass the credentials for your application and try to keep those out of the version control.

Assignment

The goal for this assignment is to create a few API endpoints for an imaginary membership platform.

For the API endpoints, you need to implement HTTP request/response handling, and data fetching and processing. You can use any web framework you are familiar with (e.g. Flask, Django, FastAPI, etc)

All API endpoints are only reading the data from the database, no writing/modifications needed.

Scenario - Membership benefits & usage reporting

Venues offer membership/loyalty benefits for their customers. Every venue has their own membership benefits. Benefits may include significant discounts or even free products.

> Example benefit: “All burgers -50%” - weekly benefit

Benefits have a recurrence rule that defines how often the benefit can be used. Currently supported recurrence types are:

> **‘day’** - Daily benefit - can be used once a day (=once per 24 hours)
> **‘week’** - Weekly benefit - can be used once a week (=once per 7 days)
> **‘month’** - Monthly benefit - can be used once every month (=once per 30 days)

Users are not allowed to use the benefit again until the time set by recurrence rule is passed from the previous benefit usage.

We are collecting data from users visiting different venues and using membership benefits. Every time when a user uses membership benefits in the venue, we create one row to the database.

For every benefit usage, the following data is stored into the database:

- **id** - auto-increment primary key
- **personId** - foreign key, user who used the benefit
- **benefitId** - foreign key, used benefit
- **usageTimestamp** - timestamp when the benefit was used

Objective

1. Implement HTTP API endpoint that returns three most used benefits for every venue during the last 180 days

   Parameters:
   - None

   Example response:

```
[
 {
   "venueId": 1,
   "topBenefits180Days": [
     {
       "benefitId": 2,
       "usageCount": 45
     },
     {
       "benefitId": 3,
       "usageCount": 23
     },
     ...
   ]
 },
 {
   "venueId": 2,
   "topBenefits180Days": [
     ...
   ]
 },
 ...
]
```

2. Implement HTTP API endpoint that returns available benefits for the user for a specific venue at the given time (default=current time) based on the benefit recurrence rules and user's earlier benefit usage.

   Parameters:
   - personId
   - venueId
   - timestamp (default: current time)

   Example response:

```
[
 {
   "id": 1,
   "venueId": 1,
   "title": "Corona beer $6",
   "recurrence": "day"
 },
   {
   "id": 3,
   "venueId": 1,
   "title": "Cocktails -50%",
   "recurrence": "week"
 }
]
```

3. Implement HTTP API endpoint that lists all the inactivity periods of the benefits for the specific venue which have had significant inactivity periods during the last 180 days.

Inactivity periods for different benefit recurrence types:

Daily benefit: benefit was not used at all for at least 14 consecutive days
Weekly benefit: benefit was not used at all for at least 21 consecutive days
Monthly benefit: benefit was not used at all for at least 30 consecutive days

Parameters:

- venueId

Example response:

```
[
 {
   "benefitId": 1,
   "inactivityPeriods": [
     {
       "startTime": "2022-04-01 13:47:10",
       "endTime": "2022-04-26 17:22:28"
     },
     {
       "startTime": "2022-06-13 16:07:13",
       "endTime": "2022-06-29 09:38:52"
     }
   ]
 },
 {
   "benefitId": 2,
   "inactivityPeriods": [
     {
       "startTime": "2022-08-15 11:15:04",
       "endTime": "2022-09-02 14:56:59"
     }
   ]
 },
 ...
]
```

Expected functionality

- HTTP API endpoint
  - Handle the incoming request and its parameters
  - Fetch and process the data from the database
  - Construct and return JSON response
- Configuration
  - Make database credentials configurable through environment variables/configuration files or other similar ways. Do not hardcode the credentials into the source code.

Guidelines

- You are free to use any frameworks / libraries / patterns / tools / etc. Also for database access, you can use whatever patterns/libraries/tools you see fit (raw SQL, ORM, etc)
- You can use any URL/query string/parameter pattern for your endpoints
- It is up to you how you want to split your logic between the Python code and database queries. It is okay to implement complex logic into the database queries as well.
- HTTP responses should be in JSON. Response examples are just to show what data should be included in the response and what kind of higher level structure it should have. Your HTTP responses do not have to match 100% with the examples (as long as you can reason about why you have chosen to do it in a particular way).
- Keep it simple. Do not spend too much time with this. Based on candidate feedback, it should take around 3-10 hours to complete.
- It is better to get everything working from end-to-end first and then focus on the details.
- If you get stuck or have any questions, do not hesitate to contact us.
- You can return your solution to us by sharing your code repository (GitHub preferred) or as an email attachment, shared zip-archive etc. Remember to provide running instructions for exotic toolchains. In GitHub, just share your repository (e.g. add collaborator) with the `cluby-assignment-review` user (https://github.com/cluby-assignment-review)

Scoring

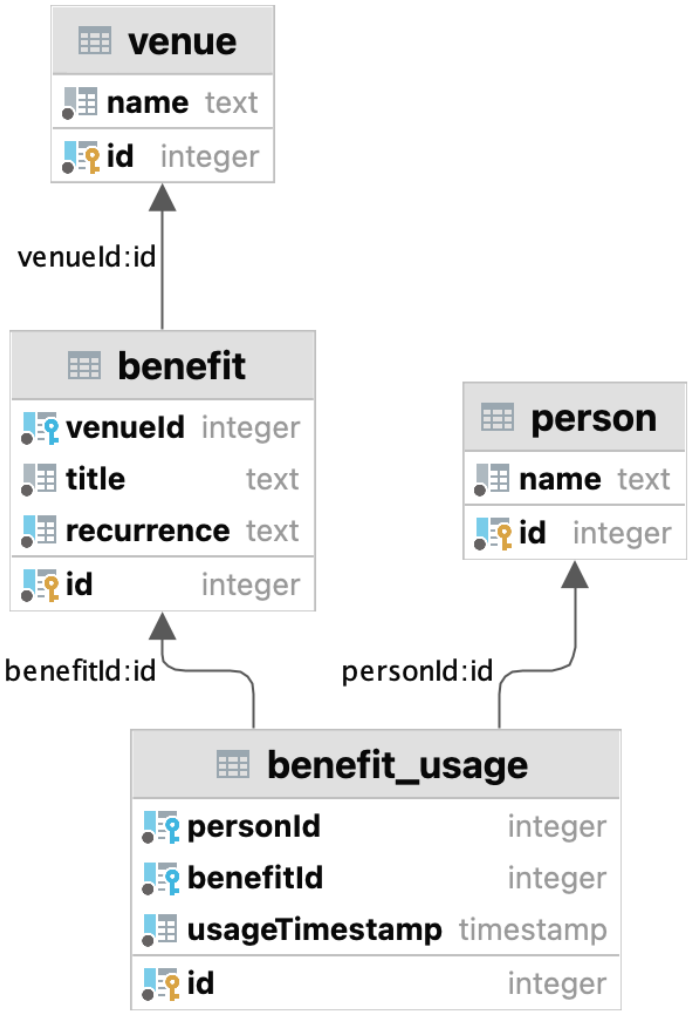
It is totally OK to return “incomplete” solutions. It is better to have less functionality that works than lots of functionality that does not work. You can always focus on the other areas if you feel that you cannot solve some particular problem.

List of areas/topics that will affect your final score:

- General
    - Selected approach to solve the given task
    - Assignment completion time
    - Used libraries, frameworks, tools etc.
    - Innovative solutions (algorithms, used libraries, wow-effect)
    - Usage of industry best practices (design patterns, functional programming, etc.)
    - Running instructions if using exotic toolsets (also for tests...)
- Code Quality
    - WTFs / minute
    - Readability
    - Structure
    - Coding style
    - Naming conventions
    - Automated tests
    - DRY
    - KISS
- Non-functional requirements
    - Quality
    - Performance
    - Security
    - Scalability
    - Robustness / Error handling
    - Documentation

Database schema

Database schema tries to be minimalistic and simple to understand. You can inspect the database structure and contents more by just connecting to it and looking around. Tables and columns should be quite self explanatory and the existing data can be used to understand the actual values.



Ps. If you have any questions or blockers, do not hesitate to ask for tips or advice!