# **BACKGROUND**

What we would like to see:

* Functionality
* Code efficiency
* Code readability
* Thinking out-of-the-box: the tasks will give you a specific method to implement. Are there more efficient ways to achieve the same outcome?

Develop a program to complete the following tasks using either

1. Nodejs on node 12.x or above
2. Python 3.X
3. .NET 3.x and above
4. MySQL or MongoDB

# **TASK 1**

## Data Transformation

Create a GET API with endpoint /drinks?type=<type>

This endpoint will hit 2 public APIs for a list of drinks.

* 1. Coffee: <https://api.sampleapis.com/coffee/hot>
  2. Beers: <https://api.sampleapis.com/beers/ale>

For the above endpoints, the returned result needs to be cleaned as:

1. There could be empty objects
2. It includes non-drink objects

These objects should be removed

The response of this endpoint that you’re creating should be an array with the following model

|  |  |
| --- | --- |
| Key | Description |
| name | The name of the drink  This will be ‘title’ from the API a and ‘name’ from the API b |
| price | The price of the drink.  It must be in the format “$X.99”  As there is no ‘price’ in API a, it should be generated randomly from a range of $8-$20  For API b, it is ‘price’ |
| rating | The rating of the drink ranging from 1-5 with 5 being the highest  It must be in 3 decimal points.  As there is no ‘rating’ in API a, it should be generated randomly from a range of 1-5  For API b, it is ‘rating.average’ |
| description | The description of the drink  For API a, it is ‘description’  As there is no ‘description’ in API b, it should map to the following description using the ‘name’ field   |  |  | | --- | --- | | Ale | Ale is a general category of beer: You'll find sub-categories like brown ales or pale ales. This is the oldest style of beer, which dates back to antiquity. What distinguishes an ale - and also makes this category of beer accessible for home brewers - is a warm-temperature fermentation for a relatively short period of time. In the brewing process, brewers introduce top-fermenting yeasts which, as the name suggests, ferment on the top of the brew. The fermentation process turns what would otherwise be a barley and malt tea into a boozy beverage. | | Porter | A type of ale, porter beers are known for their dark black color and roasted malt aroma and notes. Porters may be fruity or dry in flavor, which is determined by the variety of roasted malt used in the brewing process. | | Stout | Like porters, stouts are dark, roasted ales. Stouts taste less sweet than porters and often feature a bitter coffee taste, which comes from unmalted roasted barley that is added to the wort. They are characterized by a thick, creamy head. Ireland's Guinness may be one of the world's best-known stouts. | | Brown Ale | Brown ales range in color from amber to brown, with chocolate, caramel, citrus, or nut notes. Brown ales are a bit of a mixed bag, since the different malts used and the country of origin can greatly affect the flavor and scent of this underrated beer style. | | Pale Ale | An English style of ale, pale ales and known for their copper color and fruity scent. Don't let the name fool you: these beers are strong enough to pair well with spicy foods.  Related to the pale is the APA, or American Pale Ale, which is somewhat of a hybrid between the traditional English pale ale and the IPA style. American pale ales are hoppier and usually feature American two row malt. | | IPA | Originally, India Pale Ale or IPA was a British pale ale brewed with extra hops. High levels of this bittering agent made the beer stable enough to survive the long boat trip to India without spoiling. The extra dose of hops gives IPA beers their bitter taste. Depending on the style of hops used, IPAs may have fruit-forward citrus flavors or taste of resin and pine.  American brewers have taken the IPA style and run with it, introducing unusual flavors and ingredients to satisfy U.S. beer drinkers' love for the brew style. |   If the description cannot be found in the above table, it should be returned as empty |
| image | An url that links to the image of the drink  For coffees, it should hit <https://coffee.alexflipnote.dev/random.json> and retrieve it from ‘file’  For API b, it is ‘image’ |
| id | The Id of the object  It must be in a UUID format |

If a type is not provided, the endpoint should return a blend of both coffee and beers, sorted by the highest rating first.

If a type that is neither ‘beer’ nor ‘coffee’ is provided, the endpoint should return an error

If ‘coffee’ is provided as a type, the endpoint should only pull data from API a

If ‘beer’ is provided as a type, the endpoint should only pull data from API b

# **TASK 2**

## Data design and retrieval

Create a database of your own choice (either MySQL or MongoDB) where the program can connect to. You can design the database structure as you deem fit, but it needs to able to handle the following:

* Create an GET endpoint /cafes?location=<location>

The response of this endpoint should be the below and sorted by the highest number of employees first

|  |  |
| --- | --- |
| Key | Description |
| name | Name of the cafe |
| description | A short description of the cafe |
| employees | Number of the employees.  It must be an integer |
| logo | Logo of the cafe |
| location | Location of the cafe |
| id | UUID |

If a valid location is provided, it will filter the list to return only cafes that is within the area

If an invalid location is provided, it should return an empty list

* Create an GET endpoint /cafes/employees

The response of this endpoint should be the below and sorted by the highest number of days worked. It should list all the employees.

|  |  |
| --- | --- |
| Key | Description |
| name | Name of the employee |
| days\_worked | Number of days the employee worked  It must be an integer |
| cafe | Café’s name that the employee is under |
| id | Unique identifier in the format ‘UIXXXXXXX’ where the X is replaced with alpha numeric |

* Create a POST endpoint /cafe

This should create a new café in the database

* Create a POST endpoint /cafe/employee

This should add a new employee to the café. HOWEVER, no same employee can work in 2 cafes.

You should also provide the seed data for the database design you have designed.

# **Bonus**

Dockerize everything above and provide instructions on operating it

# **Submission**

You may upload the technical test to Github and provide us the link, or you may zip it up and send it over via email. Please ensure that your code can compile and is able to run. You should also provide a short Readme on how to compile and run the code.

All the best!