Welcome!

Please complete Part 1 and Part 2 of the take home challenge. As you create any files or make changes, please do your best to explain them in comments or separate documents so that people can follow your method. ## Part 1 The purpose of part 1 is to create a small Python server that can interact with a PostgreSQL database. This server is built in a docker container and can be run using the command `docker compose up`. Once the server has started you should be able to run `curl localhost:8000` (or `Invoke-WebRequest -Uri "http://localhost:8000"` in PowerShell). which should return

```
{"message":"Hello World"}
```

Your task is to extend this starting point by adding some API routes which will pull data from the database.

Task 1 - Choose your technologies

Docker, Python, FastAPI, and PostgreSQL are the required technologies for this project. You can add any packages/libraries you need beyond that. For example, you will need to connect to a PostgreSQL database from the Python app. You need to choose a library to allow you to connect to a database from the code. Scan through the rest of the requirements and decide what technologies you want to add (the documentation for these frameworks might be useful).

Task 2 - Create database tables to store product information

The PostgreSQL database will be used to store information about products at a hypothetical grocery store. We'll first need to be able to store information about the individual products that the store sells and their prices. For this exercise they are:

```
Wheat Bread $2.00
Rye Bread $3.10
Sugar Cookies $1.50
Oatmeal Cookies $2.10
Chocolate Chip Cookies $2.30
Apples $2.20
Oranges $3.10
Celery $1.60
```

We also need to store a hierarchy (of arbitrary depth) of categories that the products fall into. For example:

```
All Products
    Bakery
        Bread
        Cookies
    Produce
        Fruits
        Vegetables
```

Create whatever tables and additional files for better organization to store this information in your database and populate them with the data above. It would be useful to include a unique id for each item in the database (we'll use it later in the API server to lookup individual products and categories).

Note: you can open a tool to view the database with this command: `docker exec -it server bash -c 'PGPASSWORD=password123 psql -U demouser -h db demo'`

For example:

```
docker exec -it server bash -c 'PGPASSWORD=password123 psql -U demouser -h db demo'

psql (13.5 (Debian 13.5-0+deb11u1), server 14.0)
WARNING: psql major version 13, server major version 14.
         Some psql features might not work.
Type "help" for help.

demo=# select * from changeme;
 stuff
-------
(0 rows)

demo=#
```

Type `\q` or `ctrl+d` to exit the tool.

Task 3 - Create an API route which will return product information from the database.

Using the database above create a new route in your server which will return all the products and their prices in JSON format. For example, an HTTP GET request to `/products` should return:

```
[
    {
        product: "Wheat Bread",
        price: 2.00
    },
    {
        product: "Rye Bread",
        price: 3.10
    },
    ...
    etc.
]
```

Task 4 - Create an API route which will return the average product price for a given category.

This route would take a category id and return the average price for that category. For example, an HTTP GET request to "/category/2/avg-price" would compute the average product price for category id 2 (we'll assume it is "Bakery" in this case) and return an average price for all the products in the "Bakery" category. In this case, the route would return:

```
{"category_name":"Bakery","average_price":2.2}
```

(For Task 3 & 4, you can use utils in HTTPrequests.py for testing, but feel free to write your own tests) ## Task 5 - CRUD routes, Testing Optionally (time permitting. Not required, but very nice to have) - add additional routes which can be used to create, read, update, or delete products and categories. How would you test your API? Bonus points for working pytests.

Task 6 - Lessons learned, room for improvement?

You can answer during your interview, but we'd like to know: What problems did you encounter?, how did you solve them? What design tradeoffs did you make? Given more time how could you improve your project (improve its performance, make it more usable, maintainable, testable, etc.)?

Part 2

The file titled `bad_code.py` has code in it that needs to be improved. This script is designed to automate the process of generating and sending reports. Your job is to improve it as you see fit.

Files that can be edited: - bad_code.py - test_bad_code.py

Do not edit: - The `test_send_report` unit test in `test_bad_code.py` - mock_service.py

Running The Tests

You can run tests locally if you have python3 installed. To do this run `./test_bad_code.py` in your terminal.

You may also run tests in the docker container by running these commands in your terminal:

- `docker compose up`

- `docker exec -it server /bin/bash`
- `./test_bad_code.py`

Once submitted `test_send_report` must pass. Be prepared to talk about what changes you made and why.