This document describes a technical challenge that forms part of the Hectre recruitment process for Spectre back-end engineers.

It should not take more than one working day. Don’t worry if the project isn’t complete by then

- just email back what you have with some notes on where you got to and what’s left to do.

# Background

Spectre AI tool is one of Hectre’s products that utilises machine learning, computer vision, and cloud computing to provide size & colour distributions for fruit bins within seconds from uploaded images/videos.

Here is one example of uploaded images for Kiwifruit with expected diameter lines.



Your task is to design and deploy a backend machine learning solution that could allow the frontend to upload image(s) and save the information to a structured database. The tasks are:

1. Design the solution/service architecture (ideally combining with cloud platform)

My solution

1. Deploy one Python backend API endpoint for uploading one or many images with at least the following information:
   * Image ID (Primary key)
   * Created date (UTC format)
   * Fruit type
   * User email
   * Tenant ID

And save this information into a relational database.

My solution to this objective is given in the files 1) “*initialize\_backendAPI.py*”, 2) “*db\_add\_initial.py*” and 3) “*db\_query.py*”. Basically, the code is run from 1) to 3) as labelled. “*initialize\_backendAPI.py*” utilized the Flask API and SQL Alchemy to create a database and provided a way to input data into the database through parsed arguments. Please excuse my descriptions as I may not be using accurate software development jargon. I also added an additional input argument for the file path of the image. My assumption for this problem was that the input arguments were already available to me through the front-end user interface, and somehow the user interface passed this information to my code. To test the code I used “*db\_add\_initial.py*” which inserted 3 samples of manually defined rows into the database. Basically the 3 samples were manually parsed, while in reality they would be automatically fed into the script. “*db\_query.py*” then did a few get/put/delete methods on the database to check that I defined the database correctly.

In terms of uploading the images, I specified the image path as an input argument and read the image in with open-cv, then saved it into a folder as I thought filesystem is more efficient that SQL databases for storing images. I just used the image provided in the first page as input/output such that the input path was simply my current repository, and the output path was just to a folder I just created inside the repository. However, I’m not entirely sure this is the correct way to do it as it feels kind of wrong (i.e. using open-cv inside Flask to read and write images like this). I also tried some other code at around line 193 of “*initialize\_backendAPI.py*” which attempts to route directly to an interface which receives the data, then stores it into the SQL database. But since I did not have an interface I did not test it so I commented it out.

For the “relational database” part, I could not demonstrate any relationality as there was only basically one table. I am also not entirely sure if the database I defined in SQL Alchemy is relational for this task (i.e. one to many, many to one mapping etc). But the documentation said SQL Alchemy used ORM, so I was not sure if relationality is just inherently present in all databases defined using this library.

1. Demo test with simple Unit test(s)

My solution

1. Document the detailed steps on how to deploy and run the code in Mac/Linux Environment

Most of the work involves installing the dependencies. I’m not sure exactly where I should outline this info so I am doing it here. I will also include it in the README. I’m also sure that you can have all the commands below inside one single bash script so you don’t have to do each step individually. But for now I’m just sticking to what works.

1. SSH into the desired server (could be cloud, hpc cluster, or just a local server)
2. Set up virtual environment (or could be container):
   1. mkdir Virtual\_ENV
   2. cd Virtual\_ENV
   3. virtualenv -p /usr/bin/python3.6 Virtual\_ENV

This will then install python/pip and other basic tools

1. Activate virtual environment:
   1. source Virtual\_ENV/bin/activate
2. Install the python dependencies which are in the requirements.txt file that I have provided in the repository:
   1. pip install –r requirements.txt
3. The python code can then be run through commands in the terminal etc:
   1. python sample.py

But make sure to activate the virtual environment in #4 every time you reconnect to the server.

1. Deploy the one of core blocks of solution on cloud with Infrastructure as code (IaC) (In case of designing solution with cloud)

My solution

1. One of Spectre AI features is grading fruit size. Assume that we have to develop a machine learning solution to find the pixel diameter of Kiwifruits. From the visualization image above, can you describe your practical solution:
   * Which kind of ML model should you use? Why?

My solution

* + Any idea about data, training ML?

My solution

* + Any idea about organizing/deploying ML model on production?

My solution

* + Post-process algorithm idea to the get expected diameter size

My solution

* + How can we best deal with the flat shape of a Kiwifruit?

My solution

The deliverable product should contain:

* A design solution with diagram(s) and explanations
* A Python application

Python files are all in the repository

* A .git folder with the commit history for the project

Folder provided in repository

* A README that explains how to install and use the project

README.md provided in repository

The project should work on macOS or Linux.

Feel free to document any assumptions made, or ideas for improving the project. Don’t upload the project to a public repo since it might be unfair for other candidates.

# Application notes

* The Python scripts, packages are compatible with Python 3.6, 3.7, 3.8.
* Use Flask API or other blueprint for backend APIs
* Use a PostgreSQL or other structured database, including DB script/code for creation.
* It is better to use ORM (Object Relational Mapper) and a suitable backend design pattern.
* Ideal unit test is Pytest but not limited to this option