VistaPath Platform Engineer Technical Challenge

Instructions

VistaPath is trying to revolutionize the pathology industry by leveraging computer vision and cloud connectivity to boost accuracy and efficiency.

For this exercise, you will be provided a git source code repository containing a basic React application and an image of a "cassette" containing biopsy tissue samples. Your task is to build an API end-point that receives an image file upload of the cassette image, stores the image with some basic information (such as creation time, etc.), mimicks the processing of this image through a computer vision service, and returns bounding box data for the tissues in the image, as well as errors. The client-server API has the following contract:

```
URL: http://api:5000/runcv
method: POST
request payload: multipart/form-data; image/jpeg
response payload (JSON):

{
  "boxes": [
    {
      "x1": 0.5,
      "y1": 0.27,
      "x_off": 0.64,
      "y_off": 0.15
    },
    {
      "x1": 0.31,
      "y1": 0.28,
      "x_off": 0.1,
      "y_off": 0.1
    },
    {
      "x1": 0.25,
      "y1": 0.25,
      "x_off": 0.1,
      "y_off": 0.1
    }
  ],
  "errors": [
    {
      "type": "overlap",
      "boxIndexes": [1, 2]
    },
    {
```

```
      "type": "outsideImageArea",
      "boxIndex": 0
    }
  ]
}
```

The XY origin is located in the upper-left corner of the plane (`x1`, `y1`), with positive Y values going from top to bottom (`y_off`), and positive X values going from left to right (`x_off`). The scale of the image plane is 1x1, with X/Y values represented as fractions of 1.

A `sample-data.json` file has been provided in the repo root representing various bounding box combinations with and without errors. All you have to do is return is have your endpoint return three bounding box elements (as depicted above) from this hard-coded data, with error messaging handling, if appropriate. You can randomize or sequence subsequent requests in order to demonstrate responses with various types of errors, or without errors.

Requirements

- Flesh out the back-end API service in your language of choice (we use Python and Node).
- Create a backing service for the data persistence of each front-end CV request containing (or referencing) the image, basic meta-data (like creation date), and well as the bounding boxes that were generated for that particular upload. Use any sensible persistence mechanism you feel like, keeping in mind the app may grow with more potential input to be persisted from the user, such as manual measurements and annotations.
- Create a docker-compose file so that the complete system can be run locally with `docker-compose up`.
- Create a write-up describing how you would deploy this as a production system, including any details to account for scale, security, performance, maintaining dev/prod parity, etc.

Bonus

- Write a test or two.