TECHNICAL TEST FOR DEVELOPER ROLE

General Instructions

- Please read carefully the following questions and answer them to the best of your knowledge.
- The use of AI-based tools it allowed but not recommended. All questions will be peer reviewed and if you are selected for interview, you will be asked to explain all your answers.
- You should submit a github repository with a folder-based structured containing one folder per question. Any answers submitted through .zip or any other methods will be desk rejected.
- You should submit a readme file with the instructions to run your answers. Any answers without a readme description will be desk rejected.
- Please consider any requirements for your code and include them in the instructions file (e.g. readme file).

1. Recursion and Colors

You are given n disks of different sizes and colors stacked on a source rod. The goal is to transfer all the disks to a target rod using an auxiliary rod, following these rules:

1. Only one disk can be moved at a time.
2. A larger disk cannot be placed on top of a smaller disk.
3. Disks of the same color cannot be placed directly on top of each other, even if they differ in size.
4. You must use recursion to solve the problem.
5. You must use python to solve the problem.

Input:

- An integer n ($1 \leq n \leq 8$), representing the number of disks.
- A list of n tuples where each tuple contains the size and color of a disk, sorted in descending order of size. For example:

```
disks = [(5, "red"), (4, "blue"), (3, "red"), (2, "green"), (1, "blue")]
```

Output

- The sequence of moves required to transfer all disks from the source rod to the target rod.

Example Input 1

```
n = 3
disks = [(3, "red"), (2, "blue"), (1, "red")]
```

Example Output 1

```
[
    (1, "A", "C"),
```

```
        (2, "A", "B"),
        (1, "C", "B"),
        (3, "A", "C"),
        (1, "B", "A"),
        (2, "B", "C"),
        (1, "A", "C")
    ]
```

Example Input 2

```
    n = 3
    disks = [(3, "red"), (2, "blue"), (1, "red")]
```

Example Output 2

```
-1  # Impossible to complete the transfer
```

2. File Handling and Array Operations

Write a Python script that implements a class, FileProcessor, with methods to handle file and data processing tasks. The class should have the following functionality:

Attributes:

base_path: A string representing the root folder for file operations.

logger: A logging object to handle and record errors.

Methods:

__init__(self, base_path: str, log_file: str):

- Initializes the base folder path.
- Configures logging to write to the specified log_file.

list_folder_contents(self, folder_name: str, details: bool = False) -> None

- Receives the folder name relative to base_path.
- Counts and prints the number of elements inside the folder.
- Prints the names and type (folder or file) of the elements.
- Additional: If details=True, includes file sizes (in MB) and last modified times in the output.
- Logs an error if the folder does not exist.

read_csv(self, filename: str, report_path: Optional[str] = None, summary: bool = False) -> None

- Receives a CSV filename in the base_path.
- Reads the CSV file and prints: number of columns and their names, number of rows, and the average and standard deviation for numeric columns.
- If report_path is provided, saves the analysis (averages and standard deviations) as a TXT.
- If summary=True, prints a summary of non-numeric columns, including unique values and their frequencies.
- Logs an error for: missing file, incorrect file format, columns with non-numeric data when attempting numeric operations.

read_dicom(self, filename: str, tags: Optional[List[Tuple[int, int]]] = None, extract_image: bool = False) -> None

- Receives a DICOM filename in the base_path.
- Reads the file using pydicom and prints: Patient's name, Study date, Modality.
- Optionally accepts any amount of tag numbers (e.g., [(0x0010, 0x0010)]) and prints the corresponding contents.
- If extract_image=True, extracts the DICOM image and saves it as a PNG in base_path.
- Logs errors for missing files, invalid DICOM format, or unsupported pixel data.

Additional Instructions:

- Use the provided files: `./sample-02-xxxxxxx.xxx` files
- You can use https://pydicom.github.io

Example Implementation

```
processor = FileProcessor(base_path="./data")

# List folder contents
processor.list_folder_contents(folder_name="test_folder", details=True)

# Analyze a CSV file
processor.read_csv(filename="sample-01-csv.csv", report_path="./reports",
summary=True)

# Analyze a DICOM file
processor.read_dicom(
    filename="sample-01-dicom.dcm",
    tags=[(0x0010, 0x0010), (0x0008, 0x0060)],
    extract_image=True
)
```

Example Output

```
Folder: ./data/test_folder
Number of elements: 5
```

```
Files:
  - file1.txt (1.2 MB, Last Modified: 2024-01-01 12:00:00)
  - file2.csv (0.8 MB, Last Modified: 2024-01-02 12:00:00)
Folders:
  - folder1 (Last Modified: 2024-01-01 15:00:00)
  - folder2 (Last Modified: 2024-01-03 16:00:00)

CSV Analysis:
Columns: ["Name", "Age", "Height"]
Rows: 100
Numeric Columns:
  - Age: Average = 30.5, Std Dev = 5.6
  - Height: Average = 170.2, Std Dev = 10.3
Non-Numeric Summary:
  - Name: Unique Values = 50
Saved summary report to ./reports

DICOM Analysis:
Patient Name: John Doe
Study Date: 2024-01-01
Modality: CT
Tag 0x0010, 0x0010: John Doe
Tag 0x0008, 0x0060: CT
Extracted image saved to ./data/sample-01-dicom.png
```

3. RESTful API

Create a RESTful API that performs CRUD (Create, Read, Update, Delete) operations for managing medical image processing results, which will be stored in a PostgreSQL database. You can use either django or fastapi for your solution.

The main functionality of the API is to receive JSON payloads similar to this one:

```
{
    "1": {
        "id": "aabbcc1",
        "data": [
            "78 83 21 68 96 46 40 11 1 88",
            "58 75 71 69 33 14 15 93 18 54",
            "46 54 73 63 85 4 30 76 15 56"
        ],
        "deviceName": "CT SCAN"
    },
    "2": {
        "id": "aabbcc2",
        "data": [
            "14 85 30 41 64 66 85 76 96 71",
            "68 53 85 9 35 52 68 0 17 5",
            "78 40 83 72 82 94 8 19 23 62"
        ],
        "deviceName": "CT SCAN"
```

```
    }
}
```

When receiving new data,

- The API should process all elements of the payload.
- The contents of the data field should be validated ensuring all items are numbers.
- The contents of the data field should be normalized to from 0 to 1. This means finding the max value and the use it as normalization value.
- The average of the data before and after normalization should be calculated.

Requirements:

1. Models: Create specific models to represent the JSON elements.
    - The main model should include fields for id, device_id (foreign_key), average before normalization, average after normalization, data size, created_date, updated_date.
    - An additional model should be created to store device information with id and device_name
2. Endpoints:
    - Create (POST): Accept JSON payload and store it in the PostgreSQL database. The data should be validated before storing it.
    - Read (GET): List all existing entries or retrieve a single entry by its ID. Please provide parameters to filter entries by created_date, updated_date, average_before_normalization, average_after_normalization, and data_size. When appropiate filters should involve greater than and lower than parameters.
    - Update (PUT/PATCH): Allow users to update the device_name and/or the id of an existing entry.
    - Delete (DELETE): Delete an entry by its ID.
3. Database:
    - Use PostgreSQL as the backend database. Ensure proper configuration and migrations.
4. Validation:
    - Ensure that the input data is validated and that any invalid data results in an appropriate HTTP error code.
5. Logging:
    - Log all API requests and responses, as well as any errors encountered.

Example API Endpoints:

- POST /api/elements/
    - Payload: JSON data as described above.

    - Action: Create a new entry in the database.
- GET /api/elements/
    - Action: List all entries.
- GET /api/elements/<id>/
    - Action: Retrieve a specific entry by ID.
- PUT /api/elements/<id>/
    - Payload: { "device_name": "new_device_name" }
    - Action: Update the device_name or id of an existing entry.
- DELETE /api/elements/<id>/
    - Action: Delete an entry by ID.

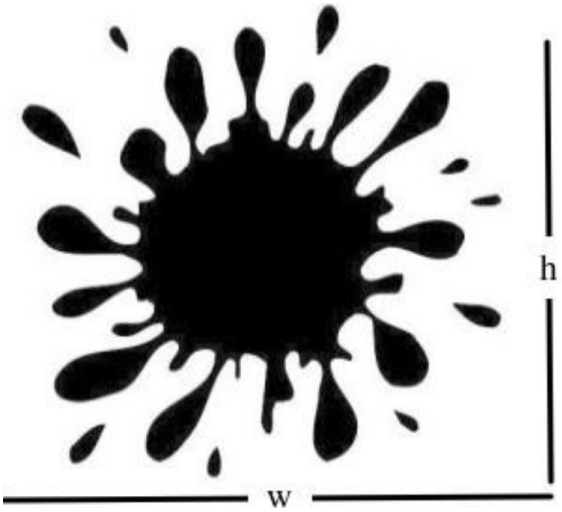
(You can use the provided files: `./sample-03-xxxxxxx.json`)

4. Create Angular App

You are tasked with developing a simple Angular application to compute the area of a stain in a binary image. The application should follow the provided methodology:

1. Upload a binary image where white pixels represent the stain, and black pixels represent the background.
2. Generate n random 2D points inside the image dimensions.
3. Count the number of random points that fall inside the stain (ni).
4. Estimate the area of the stain as Area=(Total Image Area)×(ni/n)

Example stain image:



Requirements and Constraints:

1. Interface Design:
   Use PrimeNG, Angular Material, or Tailwind CSS for styling the UI. The interface must:
   - Include two main tabs, one for uploading and calculating area, and the seconde one to display previos calculation results with a table.
   - Include a carousel or any step-by-step component to explain the methodology.
   - Include an upload button to upload the binary image.
   - Include an slider to choose the amount of random points to use.
   - Include a button to calculate the estimated stain area immediately and update the table contents.
2. Functionality: Use angular services to separate the logic for all steps:
3. State Management: Use RxJS Observables, Angular Signals, or Effects for managing data flows.
4. (Optional) Unit Testing: Write simple unit tests