Overview

This assignment consists of two main tasks, each requiring the implementation of a different API endpoint. The tasks are designed to test your ability to work with databases, handle complex queries, and manage data integrity.

Database Schema

Tables and Columns

1. Listings Table (`test_listings`)
    - `listing_id`: String (Primary Key)
    - `scan_date`: DateTime
    - `is_active`: Boolean
    - `dataset_entity_ids`: Array of Integers
    - `image_hashes`: Array of Strings

    Sample Data:

```
Unset
[
  {
    "listing_id": "1111223",
    "scan_date": "2024-10-22 12:00:00",
    "is_active": true,
    "dataset_entity_ids": [1, 2],
    "image_hashes": ["2e32d2", "f54t45r"]
  },
  {
    "listing_id": "1111224",
    "scan_date": "2024-10-23 14:30:00",
    "is_active": false,
    "dataset_entity_ids": [3],
    "image_hashes": ["a1b2c3"]
  }
]
```

2. Properties Table (`test_properties`)
    - `property_id`: Integer (Primary Key)
    - `name`: String
    - `type`: String (Values can be 'string' or 'boolean')

    Sample Data:

```
Unset
[
  {
    "property_id": 123,
    "name": "some str property",
    "type": "string"
  },
  {
    "property_id": 456,
    "name": "some bool property",
    "type": "boolean"
  }
]
```

3. String Property Values Table (`test_property_values_str`)
    - `listing_id`: String (Foreign Key referencing `test_listings.listing_id`)
    - `property_id`: Integer (Foreign Key referencing `test_properties.property_id`)
    - `value`: String

    Sample Data:

```
Unset
[
  {
    "listing_id": "1111223",
    "property_id": 123,
    "value": "str value"
  }
```

```
]
```

4. Boolean Property Values Table (`test_property_values_bool`)
    - `listing_id`: String (Foreign Key referencing `test_listings.listing_id`)
    - `property_id`: Integer (Foreign Key referencing `test_properties.property_id`)
    - `value`: Boolean

    Sample Data:

```
Unset
[
  {
    "listing_id": "1111223",
    "property_id": 456,
    "value": false
  }
]
```

5. Dataset Entities Table (`test_dataset_entities`)
    - `entity_id`: Integer (Primary Key)
    - `name`: String (Unique)
    - `data`: JSON

    Sample Data:

```
Unset
[
  {
    "entity_id": 1,
    "name": "entity_one",
    "data": {"key1": "value1", "key2": 123}
```

```
  },
  {
    "entity_id": 2,
    "name": "entity_two",
    "data": {"key3": true, "key4": null}
  },
  {
    "entity_id": 3,
    "name": "entity_three",
    "data": {"key5": "another value", "key6": false}
  }
]
```

Task 1: Implement a Listings Retrieval Endpoint

Objective

Create an API endpoint that retrieves listings from a database based on various filters and returns the results in a structured format.

Requirements

1. Input Filters:
    - `page`: Integer for pagination (1, 2, 3, ...). Return 100 listings per page sorted by `listing_id`.
    - `listing_id`: String to filter by `test_listings.listing_id`.
    - `scan_date_from` and `scan_date_to`: Date filters for `test_listings.scan_date`.
    - `is_active`: Boolean filter for `test_listings.is_active`.
    - `image_hashes`: List of strings to filter listings containing any of the provided image hashes.
    - `dataset_entities`: Dictionary to filter listings linked to entities with matching JSON data.
    - Dictionary of property filters where keys are property IDs, and values are the expected property values.
2. Output:
    - Return a JSON object containing:
        - A list of listings with all fields from `test_listings`.
        - Each listing should include related properties and entities.
        - Total count of listings matching the filters.
3. Response Format:

Unset

```
{
  "listings": [
    {
      "listing_id": "1111223",
      "scan_date": "2024-10-22 12:00:00",
      "is_active": true,
      "dataset_entity_ids": [1, 2],
      "image_hashes": ["2e32d2", "f54t45r"],
```

```
      "properties": [
        {"name": "some str property", "type": "str", "value": "str value"},
        {"name": "some bool property", "type": "bool", "value": false}
      ],
      "entities": [
        {"name": "entity_one", "data": {"key1": "value1", "key2": 123}},
        {"name": "entity_two", "data": {"key3": true, "key4": null}}
      ]
    }
  ],
  "total": 10023
}
```

Task 2: Implement a Listings Insertion/Update Endpoint

Objective

Create an API endpoint to insert or update listings and their related data in the database.

Requirements

1. Input Format:
   JSON object containing listings and their associated properties and entities.

```
Unset
{
  "listings": [
    {
      "listing_id": "1111223",
      "scan_date": "2024-10-22 12:00:00",
      "is_active": true,
      "image_hashes": ["2e32d2", "f54t45r"],
      "properties": [
        {"name": "some str property", "type": "str", "value": "str value"},
        {"name": "some bool property", "type": "bool", "value": false}
      ],
      "entities": [
        {"name": "entity_one", "data": {"key1": "value1", "key2": 123}}
      ]
    }
  ]
}
```

2. Data Handling:

- Insert new listings and associated data if they do not exist.
- Update existing properties and entities based on unique indices.
- Ensure no duplicate entries are added.

Docker Compose Requirement

- Containerization: The project must be wrapped with Docker Compose to simplify setup and execution.
- Setup: Include a `docker-compose.yml` file that defines the necessary services (e.g., application server, database).
- Execution: The README file should contain instructions to run the entire application with a single command, such as `docker-compose up`.

Submission

- Provide your code in a public GitHub repository.
- Include a README file with instructions on how to set up and run your API using Docker Compose.
- Ensure your code is well-documented and follows best practices.

Evaluation Criteria

- Correctness and completeness of the solution.
- Code quality and organization.
- Efficiency of database queries.
- Handling of edge cases and errors.
- Ease of setup and execution using Docker Compose.