Recipe Data Collection and API Development

Objective:

In this assessment, you will be given a JSON file containing recipes, and your task will be to parse the data, store it in a database, and develop an API to expose this data. The API should allow for pagination, sorting, and searching the recipe records based on various filters.

Task Overview:

1. **Parse the JSON Data**: Read and parse the provided JSON file that contains recipe data.
2. **Store Data in a Database**: Store the relevant information from the JSON into a database of your choice.
3. **Develop an API**:
    - Expose an endpoint to get all recipes in a paginated and sorted manner.
    - Expose an endpoint to search for recipes based on various fields.

Provided Data (Sample Recipe JSON):

Here is a sample recipe record from the JSON file that will be provided:

Unset
```
{
    "Contient": "North America",
    "Country_State": "US",
    "cuisine": "Southern Recipes",
    "title": "Sweet Potato Pie",
    "URL":
"https://www.allrecipes.com/recipe/12142/sweet-potato-pie-i/",
    "rating": 4.8,
    "total_time": 115,
    "prep_time": 15,
    "cook_time": 100,
    "description": "Shared from a Southern recipe, this
homemade sweet potato pie is easy to make with boiled sweet
potato. Try it, it may just be the best you've ever tasted!",
    "ingredients": [
        "1 (1 pound) sweet potato, with skin",
        "0.5 cup butter, softened",
        "1 cup white sugar",
        "0.5 cup milk",
        "2 large eggs",
        "0.5 teaspoon ground nutmeg",
        "0.5 teaspoon ground cinnamon",
        "1 teaspoon vanilla extract",
        "1 (9 inch) unbaked pie crust"
    ],
    "instructions": [
        "Place whole sweet potato in pot and cover with water;
bring to a boil. Boil until tender when pierced with a fork,
40 to 50 minutes.",
        "Preheat the oven to 350 degrees F (175 degrees C).",
        "Remove sweet potato from the pot and run under cold
water. Remove and discard skin.",
        "Break sweet potato flesh apart and place in a bowl.
Add butter and mix with an electric mixer until well combined.
Add sugar, milk, eggs, nutmeg, cinnamon, and vanilla; beat on
medium speed until mixture is smooth. Pour filling into
unbaked pie crust.",
        "Bake in the preheated oven until a knife inserted in
the center comes out clean, 55 to 60 minutes.",
        "Remove from the oven and let cool before serving."
    ],
    "nutrients": {
        "calories": "389 kcal",
        "carbohydrateContent": "48 g",
        "cholesterolContent": "78 mg",
        "fiberContent": "3 g",
        "proteinContent": "5 g",
        "saturatedFatContent": "10 g",
        "sodiumContent": "254 mg",
        "sugarContent": "28 g",
        "fatContent": "21 g",
        "unsaturatedFatContent": "0 g"
    },
    "serves": "8 servings"
}
```

Database Design:

Design a database schema to store the following fields from the recipe data:

1. **cuisine** (VARCHAR)
2. **title** (VARCHAR)
3. **rating** (FLOAT)
4. **prep_time** (INT)
5. **cook_time** (INT)
6. **total_time** (INT)
7. **description** (TEXT)
8. **nutrients** (JSONB)
9. **serves** (VARCHAR)

```
Unset

```

Handling `NaN` Values:

When parsing the JSON file and storing data in the database:

- If any numeric fields (like `rating`, `prep_time`, `cook_time`, or `total_time`) contain `NaN` values or invalid data, set those values to `NULL` before storing them in the database.

For example:

- If the `rating` is `"NaN"`, set it as `NULL`.
- If `prep_time` or `cook_time` is `"NaN"`, set those as `NULL`.

You can achieve this by checking for `NaN` values during the parsing process and handling them accordingly.

API Development:

Develop a RESTful API to expose the data from the recipes table.

1. **API Endpoint 1: Get All Recipes (Paginated and Sorted by Rating)**
    - **URL**: `/api/recipes`
    - **Method**: `GET`
    - **Query Parameters**:
        - `page`: Page number for pagination (default is 1).
        - `limit`: Number of recipes per page (default is 10).
    - **Response**: A list of recipes sorted by rating in descending order.
2. **Example Request**:

Unset
```
GET /api/recipes?page=1&limit=10
```

3. **Example Response**:

Unset
```
{
  "page": 1,
  "limit": 10,
  "total": 50,
  "data": [
    {
      "id": 1,
      "title": "Sweet Potato Pie",
      "cuisine": "Southern Recipes",
      "rating": 4.8,
      "prep_time": 15,
      "cook_time": 100,
      "total_time": 115,
      "description": "Shared from a Southern recipe, this
homemade sweet potato pie...",
      "nutrients": {
        "calories": "389 kcal",
        "carbohydrateContent": "48 g",
        "cholesterolContent": "78 mg",
        "fiberContent": "3 g",
        "proteinContent": "5 g",
        "saturatedFatContent": "10 g",
        "sodiumContent": "254 mg",
        "sugarContent": "28 g",
        "fatContent": "21 g"
      },
      "serves": "8 servings"
    }
  ]
}
```

4. **API Endpoint 2: Search Recipes**
   - **URL**: `/api/recipes/search`
   - **Method**: `GET`
   - **Query Parameters**:
     - `calories`: Filter by calories (greater than, less than, or equal to a specific value).
     - `title`: Search by recipe title (partial match).
     - `cuisine`: Filter by cuisine.
     - `total_time`: Filter by total time (greater than, less than, or equal to a specific value).
     - `rating`: Filter by rating (greater than, less than, or equal to a specific value).
5. **Example Request**:

```
Unset
GET /api/recipes/search?calories=<=400&title=pie&rating=>=4.5
```

6. **Example Response**:

```
Unset
{
  "data": [
    {
      "id": 1,
      "title": "Sweet Potato Pie",
      "cuisine": "Southern Recipes",
      "rating": 4.8,
      "prep_time": 15,
      "cook_time": 100,
      "total_time": 115,
      "description": "Shared from a Southern recipe, this
homemade sweet potato pie...",
      "nutrients": {
        "calories": "389 kcal",
        "carbohydrateContent": "48 g",
        "cholesterolContent": "78 mg",
        "fiberContent": "3 g",
        "proteinContent": "5 g",
        "saturatedFatContent": "10 g",
        "sodiumContent": "254 mg",
        "sugarContent": "28 g",
        "fatContent": "21 g"
      },
      "serves": "8 servings"
    }
  ]
}
```

Submission Instructions:

1. **Code**: Submit your source code preferably as a git repo, including the logic to parse the JSON file, store data in the database, and the APIs.
2. **Database Setup**: Provide the SQL schema for the database and any scripts necessary to set up the database.
3. **API Testing**: Include instructions or examples on how to test your API, along with any sample requests and responses.

Frontend (UI):

Requirements:

1. Call RESTful API to fetch all Recipes information and render it in a table with below mentioned columns.
    a. Title - **Truncated if width of the column is less than data**
    b. Cuisine
    c. Rating - It should use typical standard rating style (Star)
    d. Total Time
    e. No. of People serves
2. Clicking the row should open the detail view either right side drawer/pull over - where we should show below information.
    a. Title of the recipe and Cuisine should be at the header/title of the drawer.
    b. Key/Value pair - Description: <Actual Data>
    c. Key/Value pair - Total Time: <Actual Data> with expand icon, on expand to show Cook Time, Prep Time.
    d. Nutrition as separate section with small table
        i. Calories
        ii. carbohydrateContent
        iii. cholesterolContent
        iv. fiberContent
        v. proteinContent

    vi. saturatedFatContent
    vii. sodiumContent
    viii. sugarContent
    ix. fatContent

3. Add Field (Cell) level filter and use /search API to retrieve data based on user applied search in respective fields.
4. Also, handle pagination and results per page can be customizable by users starting from 15 to 50.
5. If no results are found, show a fallback screen with a message (Nice to Have).
6. If no data is found, show a fallback screen with a message (Nice to Have).