Remitly Poland

Home exercise

Complete the home exercise and submit it through the **self-assessment form** no later than **May 5th.** *Please look into the form before you start coding to explore our expectations.*

Exercise description

A SWIFT code, also known as a Bank Identifier Code (BIC), is a unique identifier of a bank's branch or headquarter. It ensures that international wire transfers are directed to the correct bank and branch, acting as a bank's unique address within the global financial network.

Currently, SWIFT-related data for various countries is stored in a spreadsheet. While this format is convenient for offline management, we need to make this data accessible to our applications.

Your task is to create an application that will:

1. **Parse SWIFT codes Data**:

   Using the provided FILE parse the data following those guidelines:

   - **Code Identification**:
     - Codes ending with “XXX” represent a bank's headquarters, otherwise branch.
     - Branch codes are associated with a headquarters if their first 8 characters match.
     - Codes can represent both the branch and the headquarter of the bank.
   - **Formatting**:
     - Country codes and names must always be stored and returned as uppercase strings.
     - Redundant columns in the file may be omitted.

2. **Store the Data**:

   The parsed SWIFT data must be stored in a database, either relational or non-relational. The key requirement is that the chosen database technology must support fast, low-latency querying to ensure quick response times. The storage structure should allow efficient retrieval of individual SWIFT codes as well as country-specific data using ISO-2 codes.

3. **Expose a REST API**:

   The application must provide access to the SWIFT codes database through RESTful endpoints.

Endpoint 1: Retrieve details of a single SWIFT code whether for a headquarters or branches.
GET: `/v1/swift-codes/{swift-code}`:
Response Structure for headquarter swift code:

```
{
    "address": string,
    "bankName": string,
    "countryISO2": string,
    "countryName": string,
    "isHeadquarter": bool,
    "swiftCode": string
    “branches”: [
          {
               "address": string,
               "bankName": string,
               "countryISO2": string,
               "isHeadquarter": bool,
               "swiftCode": string
          },
          {
               "address": string,
               "bankName": string,
               "countryISO2": string,
               "isHeadquarter": bool,
               "swiftCode": string
          }, . . .
     ]
}
```

Response Structure for branch swift code:

```
{
    "address": string,
    "bankName": string,
    "countryISO2": string,
    "countryName": string,
    "isHeadquarter": bool,
    "swiftCode": string
}
```

Endpoint 2: Return all SWIFT codes with details for a specific country (both headquarters and branches).
GET: `/v1/swift-codes/country/{countryISO2code}`:
Response Structure :

```
{
    "countryISO2": string,
    "countryName": string,
    "swiftCodes": [
        {
            "address": string,
            "bankName": string,
            "countryISO2": string,
            "isHeadquarter": bool,
            "swiftCode": string
        },
        {
            "address": string,
            "bankName": string,
            "countryISO2": string,
            "isHeadquarter": bool,
            "swiftCode": string
        }, . . .
    ]
}
```

Endpoint 3: Adds new SWIFT code entries to the database for a specific country.
POST: `/v1/swift-codes`:
Request Structure :

```
{
    "address": string,
    "bankName": string,
    "countryISO2": string,
    "countryName": string,
    “isHeadquarter”: bool,
    "swiftCode": string,
}
```

Response Structure:

```
{
    "message": string,
}
```

Endpoint 4: Deletes swift-code data if swiftCode matches the one in the database.
DELETE: `/v1/swift-codes/{swift-code}`
Response Structure:

```
{
    "message": string,
}
```

Key Expectations:

1. Include all code in a GitHub repository with a well-structured and clear `README.md`.
2. Ensure solution correctness and maintain a high standard of code quality.
3. Verify that all endpoints and responses align with the structure outlined in the exercise description.
4. Handle all edge cases gracefully, with clear and informative error messages.
5. Provide thorough unit and integration tests to ensure reliability.
6. Containerize the application and database, ensuring the endpoints are accessible at `localhost:8080`.

What should you send us?

We require a link to a public GitHub repository containing your solution. The repository must include a `README.md` file with clear and detailed instructions for setting up, running, and testing your project. The solution must be implemented in **one of the following languages: Go (preferred), Kotlin, Java or TypeScript.**