Bajaj Finserv Health – Programming Challenge
SRM (25th-30th April'25)

Problem Statement

Build a Spring Boot application that automatically interacts with a remote API at application startup, without any manual HTTP trigger (i.e., no controller call).

Objective

The application must:

- Call the `/generateWebhook` endpoint on **startup**.
- Solve the assigned problem and store the result in a json.
- Send the result to the provided webhook with JWT authentication.

On Startup:

Make a POST request to:
`POST https://bfhldevapigw.healthrx.co.in/hiring/generateWebhook`

Request Body:

```
{
  "name": "John Doe",
  "regNo": "REG12347",
  "email": "john@example.com"
}
```

Sample Response:

```
{
  "webhook": "https://bfhldevapigw.healthrx.co.in/hiring/testWebhook",
  "accessToken": "asjdh89d7897asd89asdaskjdlasd8sa",
  "data": {
    "users": [
      { "id": 1, "name": "Aniket", "follows": [2, 3] },
      { "id": 2, "name": "Shubham", "follows": [1] },
      { "id": 3, "name": "Ashish", "follows": [2] }
    ]
  }
}
```

Example Output POST:

`POST https://bfhldevapigw.healthrx.co.in/hiring/testWebhook`

Headers:

```
Authorization: asjdh89d7897asd89asdaskjdlasd8sa
Content-Type: application/json
```

Body:

```
{
  "outcome": [[1, 2]]
}
```

Retry Policy

Retry POST to the webhook up to **4 times** upon failure. The remote server will eventually succeed within 4 attempts.

Technical Requirements

- Use `RestTemplate` or `WebClient` in Spring Boot.
- No REST controller or external trigger should be implemented.
- JWT token must be used in the Authorization header.

Test Instructions

Start the test with the following:
`POST https://bfhldevapigw.healthrx.co.in/hiring/generateWebhook`
Request Body:

```
{
  "name": "John Doe",
  "regNo": "REG12347",
  "email": "john@example.com"
}
```

Response will contain:

- A signed webhook URL
- Access Token
- Input JSON
- Assigned question (based on **last two digits** of `regNo`):
  - If odd → Question 1
  - If even → Question 2

Question 1: Mutual Followers

Identify mutual follow pairs where both users follow each other. Output only direct 2-node cycles as `[min, max]` once.

Example Input for Question 1

```
{
  "users": [
    {"id": 1, "name": "Alice", "follows": [2, 3]},
    {"id": 2, "name": "Bob", "follows": [1]},
    {"id": 3, "name": "Charlie", "follows": [4]},
    {"id": 4, "name": "David", "follows": [3]}
  ]
}
```

Output:

```
{
  "regNo": "REG12347",
  "outcome": [[1, 2], [3, 4]]
}
```

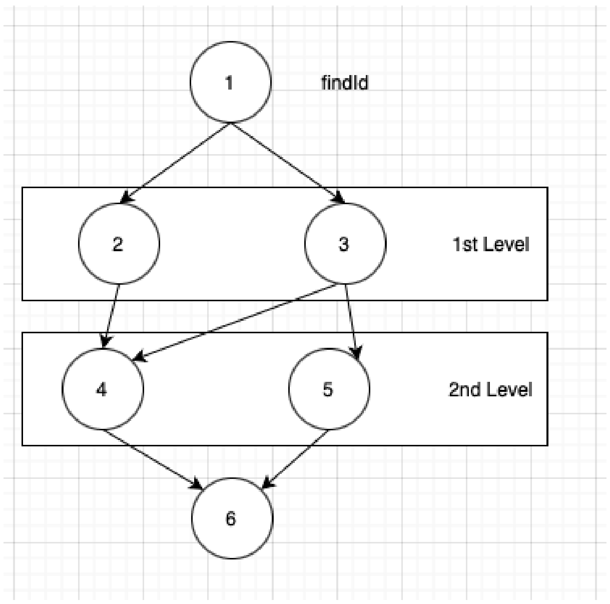
Explanation: User with id:1 follows id:2 and vice-versa. Therefore, add [min user id] and [max user id] to the list of results, which will be [[1,2]] Then, we can see than id:3 follows id:4 and vice-versa. Hence, as [3,4] to the list. Now the result will be [[1,2],[3,4]]

Question 2: Nth-Level Followers

Given a start ID (**findId**) and `n`th level, return user IDs that are exactly `n` levels away in the ”follows” list.

Example Input:

```
{
  "users": {
    "n": 2,
    "findId": 1,
    "users": [
      {"id": 1, "name": "Alice", "follows": [2, 3]},
      {"id": 2, "name": "Bob", "follows": [4]},
      {"id": 3, "name": "Charlie", "follows": [4, 5]},
      {"id": 4, "name": "David", "follows": [6]},
      {"id": 5, "name": "Eva", "follows": [6]},
      {"id": 6, "name": "Frank", "follows": []}
    ]
  }
}
```



Expected Output:

```
{
  "regNo": "REG12347",
  "outcome": [4,5]
}
```

Explanation: For given input, **findId:1** and **nth Level:2**, we can observe that at 2nd level id:1 has id:4 though id:2; and id:4 and id:5 through id:3. Hence the result is [4,5].

Submission

- Submit via this form: `https://forms.office.com/r/Lu9R50G5MC`
- Include the public GitHub repo with:
  - Code
  - Final JAR output
  - RAW downloadable GitHub link to the JAR

Good Luck!