Backend Developer Interview Code Challenge

As we strive to find the best of the best to join our team, we believe that one of the most effective ways to assess a person's technical skills is to put them to practice.

Code Challenge: Sushi Shop

Overview:

The challenge is to build a simulated sushi shop server-side program that takes orders from the customers, processes the orders in parallel, shows and updates the order status.

The program should be built using the following frameworks/libraries/tools:

- Spring boot
- H2 database
- Maven/Gradle
- Any other libraries you feel you may need

Requirements:

1. The server should start on port 9000
2. Use an embedded in-memory H2 database with the following SQL to initialize the database when the server starts:

```
DROP TABLE IF EXISTS sushi;

CREATE TABLE sushi (
  id INT AUTO_INCREMENT  PRIMARY KEY,
  name VARCHAR(30),
  time_to_make INT DEFAULT NULL
);

DROP TABLE IF EXISTS sushi_order;

CREATE TABLE sushi_order (
  id INT AUTO_INCREMENT  PRIMARY KEY,
  status_id INT NOT NULL,
  sushi_id INT NOT NULL,
  createdAt TIMESTAMP NOT NULL default CURRENT_TIMESTAMP
);

DROP TABLE IF EXISTS status;
```

```
CREATE TABLE status (
  id INT AUTO_INCREMENT  PRIMARY KEY,
  name VARCHAR(30) NOT NULL
);

INSERT INTO sushi (name, time_to_make) VALUES
('California Roll', 30),
('Kamikaze Roll', 40),
('Dragon Eye', 50);

INSERT INTO status (name) VALUES
('created'),
('in-progress'),
('paused'),
('finished'),
('cancelled');
```

3. REST APIs:
    a. All the response bodies should include the following fields:
        i. code: **0** for success, and any other integers for failures
        ii. msg: A meaningful message represents the result

    b. Build the following REST APIs that accepts and returns **JSON** data:
        i. Submitting an order: POST /api/orders:
            - Request body:
```
{
    "sushi_name": "California Roll"
}
```
            - Response:
                - Code: 200
                - Body:
```
{
    "order": {
        "id": 10,
        "statusId": 1,
        "sushiId": 1,
        "createdAt": 1582643059540
    },
    "code": 0,
    "msg": "Order submitted"
}
```
            - Only **three** orders can be processed at the same time
            - When an order is submitted, the order record should be saved into database with status set to “created”

- When a chef is ready to process an order, the corresponding order record should be updated in the database with status set to "in-progress"
- The field "time_to_make" from **sushi** table represents how long it takes to make a specific kind of sushi. For example, a California Roll takes 30 seconds to make, thus a chef will be occupied for 30 seconds to finish making the sushi
- When an order is completed, the corresponding order record should be updated in the database with status set to "finished"

ii. Cancelling an order: PUT /api/orders/cancel/{order_id}
- Path parameter **order_id**
- Response:
  - Code: 200
  - Body:

```
{
      "code": 0,
      "msg": "Order cancelled"
}
```

- The chef who is working on making the ordered sushi should stop upon cancellation request
- The order should be updated in the database with status set to "cancelled"

iii. Pausing an order: PUT /api/orders/pause/{order_id}
- Path parameter **order_id**
- Response:
  - Code: 200
  - Body:

```
{
      "code": 0,
      "msg": "Order paused"
}
```

- When an order needs to be paused, the chef must pause the progress of making the sushi until the order is resumed
- The order should be updated in the database with status set to "paused"

iv. Resuming an order: PUT /api/orders/resume/{order_id}
- Path parameter **order_id**
- Response:
  - Code: 200

- Body:

```
{
      "code": 0,
      "msg": "Order resumed"
}
```

  - When a resuming order request is received, the chef should continue to process the order with high priority. A resumed order should only be processed base on the **remaining** processing time. For example, an order of California Roll is paused after 20 seconds since the order became in-progress, then it should take 10 more seconds to finish once resumed.
  - The order should be updated in the database with status set to "in-progress"

v. Displaying all orders: GET /api/orders/status
  - Response:
    - Code: 200
    - Body:

```
{
      "in-progress": [
            {
                  "orderId": 4,
                  "timeSpent": 23
            },
            {
                  "orderId": 5,
                  "timeSpent": 21
            }
      ],
      "pending": [
            {
                  "orderId": 6,
                  "timeSpent": 0
            }
      ],
      "paused": [
            {
                  "orderId": 2,
                  "timeSpent": 5
            }
      ],
      "cancelled": [
            {
                  "orderId": 3,
                  "timeSpent": 6
```

```
            }
        ],
        "completed": [
            {
                "orderId": 1,
                "timeSpent": 30
            }
        ]
    }
```

Evaluation:

1. Code completion and correctness
2. Code brevity and clarity
3. Code efficiency and readability