Introduction

This application implements a service layer where customer data is managed using Redis as a caching layer, and it utilizes **Resilience4j** to provide retry mechanisms for both Redis operations and database calls. The goal is to ensure high availability by retrying failed operations with exponential backoff and fallback strategies.

SETUP:

CLONE : https://github.com/SaiPranay-tula/DBCURDASYNC.git

Prerequisites

To set up this project, you need:

- **Java 17** : Required for running the Spring Boot application.
- **Maven**: To manage dependencies and build the project.
- **Redis**: Local or Dockerized Redis instance for caching.
- **Spring Boot**: Version 3.2.0 or later for the application setup.
- **Resilience4j**: For retry functionality and fault tolerance.

Overview

This application follows the **Layered Architecture**:

1. **Controller Layer**: Handles HTTP requests and responses.
2. **Service Layer**: Contains business logic, interactions with Redis, and database calls.
3. **Data Layer**: Handles interactions with the database via repositories.

4.2 Service Layer Architecture

- **RedisService**: Interacts with Redis for saving, retrieving, and deleting customer data, utilizing **Resilience4j** retry mechanisms.
- **CustomerService**: Manages business logic for customer-related operations, such as saving, retrieving, and deleting customers from both Redis cache and the database.

4.3 Controller Layer

- **CustomerController**: Exposes REST endpoints to interact with the customer data, using the service layer for processing.

START REDIS:

```
docker run --name redis-container -p 6379:6379 -d redis
```

```
CREATE TABLE customer (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255),
    email VARCHAR(255),
    phone VARCHAR(255)
);
```

API SPEC:

```
http://localhost:8000/put/customer
{
  "name":"cust21",
  "email":"2@d",
  "phone":"23232",
  "id":23
}
```

```
http://localhost:8000/get/customer?id=17
http://localhost:8000/get/all
```