

# Project Report: REST API for Pool Management

Mai Chien Vi Thien

July 20, 2024

## 1 Introduction

The goal of this project is to design and implement a REST API with two POST endpoints for managing pools of values. The API allows for appending or inserting values into pools and querying these pools to retrieve quantile information. The project is implemented in Java and is focused on high performance and resiliency while maintaining simplicity.

## 2 Problem Description

The task is to develop a REST API with the following functionality:

- **POST Endpoint for Appending/Inserting Values:** This endpoint receives a JSON document with two fields: `poolId` (numeric) and `poolValues` (array of values). It should append values to an existing pool or insert a new pool if it does not exist.
- **POST Endpoint for Querying a Pool:** This endpoint receives a JSON document with two fields: `poolId` (numeric) and `percentile` (in percentile form). It should return the calculated quantile and the total count of elements in the specified pool.

## 3 Solution Approach

To address the problem, the following approach was taken:

1. **Define Data Structures:** Implement a data structure to hold pools of values and their associated metadata.
2. **Implement Endpoints:** Develop the REST API with two POST endpoints as specified.
3. **High Performance and Resiliency:** Ensure the implementation is optimized for performance and handle errors gracefully.

## 4 Architecture

The architecture of the REST API is based on a simple yet effective design. The main components include:

- **Controller:** Handles incoming HTTP requests and responses. It delegates business logic to the service layer.
- **Service:** Contains the core business logic for appending/inserting values and querying pools.
- **Model:** Defines the data structures for pools and their values.

### 4.1 Diagram

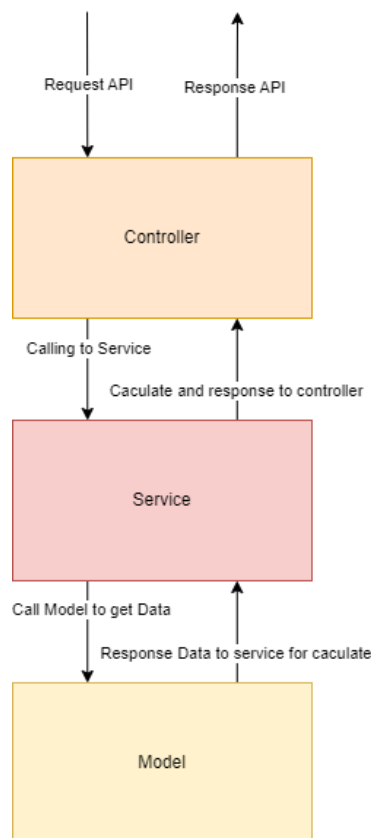


Figure 1: System Architecture

## 5 Running

To run and launch the system, follow the steps outlined in the `README.md` file available in the provided GitHub repository. The repository contains detailed instructions on how to set up the environment, build the project, and execute the application. You can find the repository at the following link:

<https://github.com/thiendsu2303/pool-api>

## 6 Testing

Testing is performed using JUnit and MockMvc. Below are some key test cases and their results.

### 6.1 Testing with Postman

The API was tested using Postman to ensure correct functionality. Below are six test cases, each with an example request and response screenshot.

#### 6.1.1 Postman Test Case 1: Insert New Pool

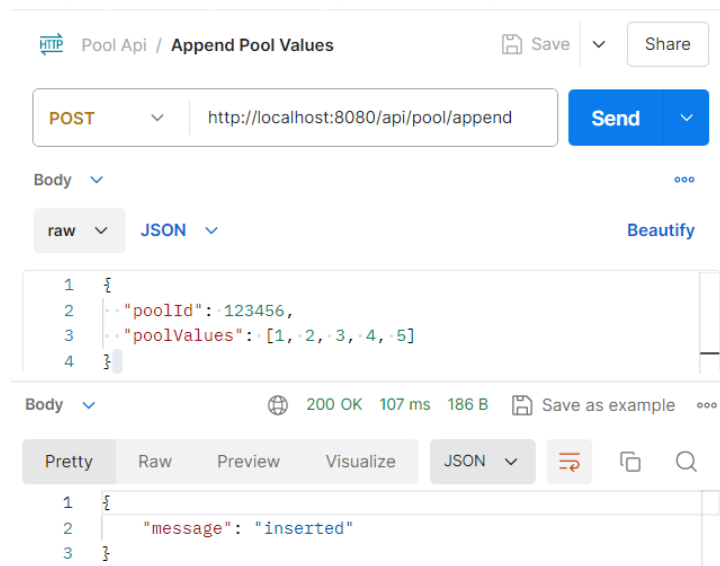


Figure 2: Postman Test Case 1: Insert New Pool

#### 6.1.2 Postman Test Case 2: Append to Existing Pool

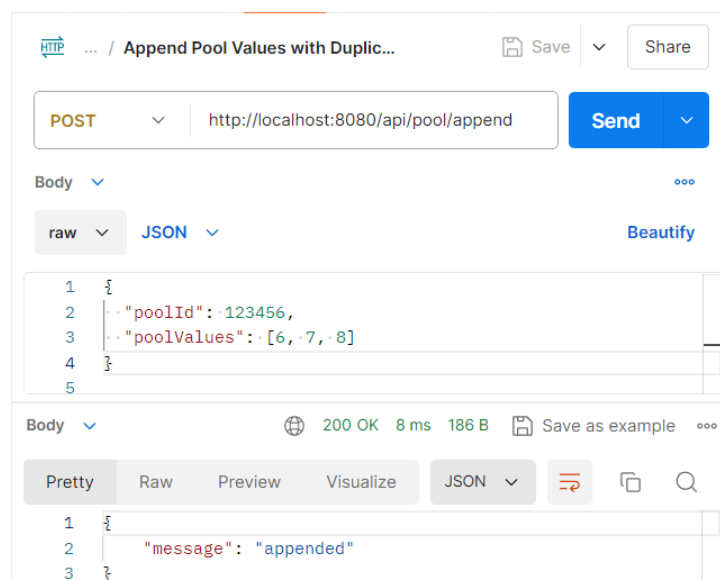


Figure 3: Postman Test Case 2: Append to Existing Pool

### 6.1.3 Postman Test Case 3: Query Pool with Valid Percentile

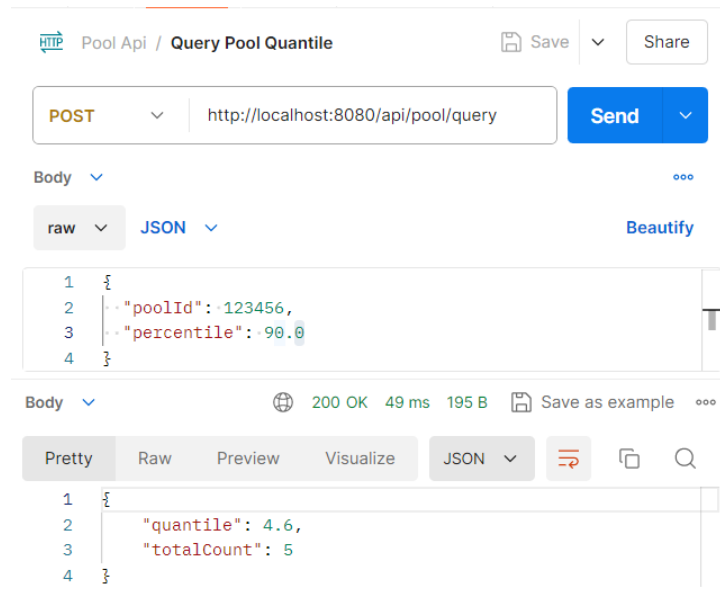


Figure 4: Postman Test Case 3: Query Pool with Valid Percentile

### 6.1.4 Postman Test Case 4: Query Non-Existing Pool

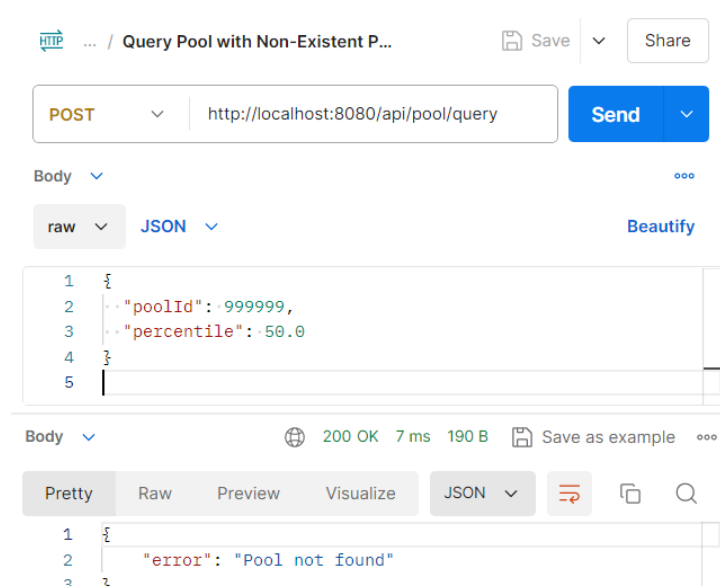


Figure 5: Postman Test Case 4: Query Non-Existing Pool

### 6.1.5 Postman Test Case 5: Query Pool with Invalid Percentile

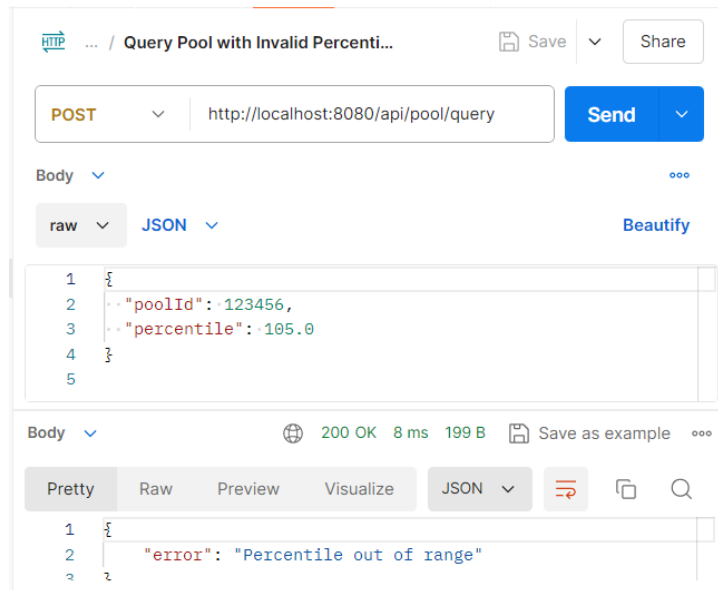


Figure 6: Postman Test Case 5: Query Pool with Invalid Percentile

### 6.1.6 Postman Test Case 6: Handle Empty Pool

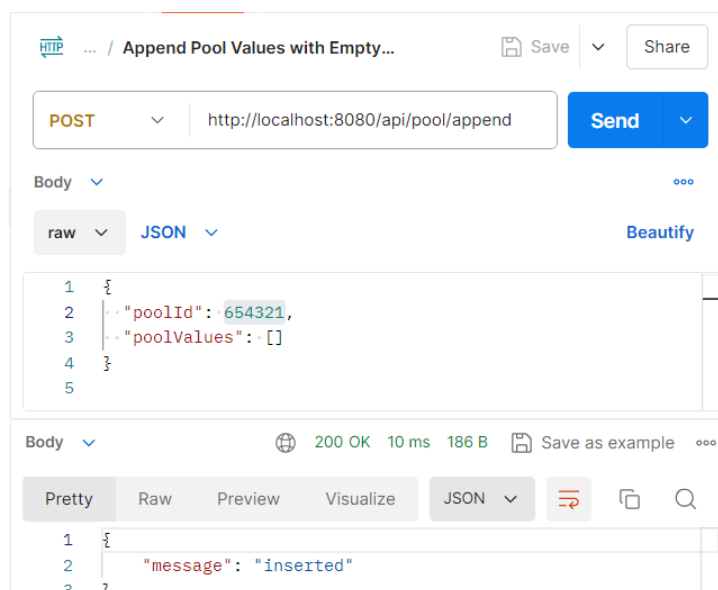
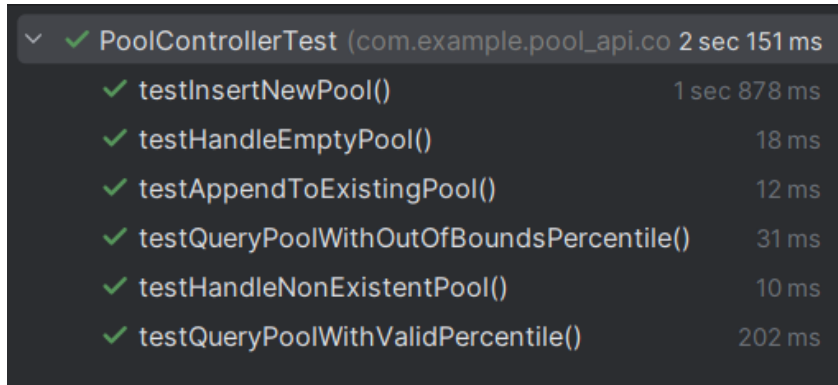


Figure 7: Postman Test Case 6: Handle Empty Pool

## 6.2 Write Unit Test In Java

Unit Test are implement for testing inside code repository.



A screenshot of a unit test runner interface. At the top, a summary line shows a green checkmark, the test class name 'PoolControllerTest', the package 'com.example.pool\_api.co', and the total execution time '2 sec 151 ms'. Below this, a list of individual test methods is shown, each with a green checkmark and its execution time. The tests are: 'testInsertNewPool()' (1 sec 878 ms), 'testHandleEmptyPool()' (18 ms), 'testAppendToExistingPool()' (12 ms), 'testQueryPoolWithOutOfBoundsPercentile()' (31 ms), 'testHandleNonExistentPool()' (10 ms), and 'testQueryPoolWithValidPercentile()' (202 ms).

✓ PoolControllerTest (com.example.pool_api.co 2 sec 151 ms
✓ testInsertNewPool() 1 sec 878 ms
✓ testHandleEmptyPool() 18 ms
✓ testAppendToExistingPool() 12 ms
✓ testQueryPoolWithOutOfBoundsPercentile() 31 ms
✓ testHandleNonExistentPool() 10 ms
✓ testQueryPoolWithValidPercentile() 202 ms

Figure 8: Unit Test

## 7 Conclusion

This project demonstrates the development of a simple yet efficient REST API for managing pools of values. The focus on performance and simplicity ensures that the solution is both practical and easy to maintain. Future work may involve optimizing for high availability and scalability.