Name: Collin Gee Microsoft CRUD microservice Submission  
Status: Completed

**Overview**

**Assumptions**

1. Using only one instance of the microservice and therefore load balancing is not needed
2. No authentication needed even though it is a public API
3. No security roles needed to distinguish between who can perform certain CRUD commands
4. Update request does not insert new Employee if employeeId does not exist
5. Having read all and read by id is beneficial
6. Update request allowed to overwrite current employee without verifying and fields (e.g. making sure the names match between the http request and database Employee)
7. Only employeeId can be null.
8. employeeId is overwritten if sent in the create request

**High Level Architecture Diagram**

**A screenshot of a computer

Description automatically generated**

**Supporting Documents**

1. https://github.com/collingee/Microsoft\_CRUD\_Microservice

**Test Cases (No Code)**

Empty the database using the H2 Console. See Microsoft CRUD API Testing.postman\_collection.json for examples of how to run each of the CRUD requests.

1. Assert Success: Read all Employees
   1. Return only response code 204
2. Assert Failure: Attempt to read by any employeeId
3. Assert Failure: Attempt to update by any employeeId
4. Assert Failure: Attempt to create Employee with all of the failures in behavior examples
5. Assert Success: Create several different Employees, some with employeeId, some without (overridden by database)
   1. Each will return the Employee back as well as 200 OK response
6. Assert Success: Read all Employees
   1. Will return list of all Employees in database and 200 OK response
7. Assert Success:
   1. Read employeeId 1
   2. Call update with the returned object but change the values.
      1. Will return the changed Employee with the employeeId 1
   3. Verify update with read by employeeId 1
8. Assert Success: Delete employeeId 1
   1. Will return 204 No Content
   2. Verify deletion with read by employeeId 1

**Behavior Examples**

1. Create 1 Employee:
   1. Failure:
      1. Empty body in http method (400 Bad Request)
      2. All or any values as null (employeeId can be null) (400 Bad Request)
      3. Incorrect data types (400 Bad Request)
      4. Negative salary (400 Bad Request)
   2. Success:
      1. All values valid (with or without employeeId) (200 OK)
2. Read All Employees:
   1. Failure:
   2. Success:
      1. Empty database (204 No Content)
      2. Returns all Employees in database (200 OK)
3. Read 1 Employee:
   1. Failure:
      1. employeeId not in database (404 Not Found)
   2. Success:
      1. Valid employeeId returns the Employee from database (200)
4. Update 1 Employee:
   1. Failure:
      1. employeeId missing (404 Not Found)
   2. Success:
      1. Valid employee returned (200 OK)
5. Delete 1 Employee:
   1. Failure:
   2. Success:
      1. Deletes Employee if found, no error if not found (204 No Content)

**Problem Identified**

We need to perform CRUD operations on billions of Employee entities with high RPS.

**Goals Identified**

Create a microservice capable of handlings CRUD operations.

**Technical overview**

**Existing System Architecture**

Employee entity and the callers of the CRUD microservice.

**Execution Plan**

Create a CRUD handler using Spring Boot.

**Data Model**

The database schema is as follows:

table Employee(

employeeId bigint auto\_increment,

firstName varchar(255) NOT NULL,

lastName varchar(255) NOT NULL,

department varchar(255) NOT NULL,

isActive boolean NOT NULL,

salary int varchar(255) NOT NULL

);

**Interface/API Definitions**

Base address: http://localhost:8080

1. Create 1 Employee:
   1. Method: Post
   2. Path: /employee
   3. Data: Send Employee entity in the body of http method
2. Read All Employees:
   1. Method: Get
   2. Path: /employee
   3. Data: None
3. Read 1 Employee:
   1. Method: Get
   2. Path: /employee/{id}
   3. Data: Add employeeId in the path
4. Update 1 Employee:
   1. Method: Put
   2. Path: /employee/{id}
   3. Data: Send new Employee in body of http method and add employeeId in the path
5. Delete 1 Employee:
   1. Method: Delete
   2. Path: /employee/{id}
   3. Data: Add employeeId in the path

**Fault Tolerance**

For update, if the employeeId in the path is not in the database, it will not add an Employee. If there are any errors in the controller, 500 Bad Request will be returned and the microservice will continue to run.

**Future Ideas**

1. Authentication for different users of different security levels (e.g. general users should only be able to read)
2. Load balancing
   1. Will allow for multiple instances of the microservice to run together
3. CI/CD
   1. Research the best database and programming language for this microservice
   2. Create a test suite that automatically verifies the functionality for successes and failures