# Microsoft Take Home Exam

## Job ID: 1601528 (Software Engineer)

**Applicant Name: Tushar Saini**

# Contents

# Assumptions:

**High Availability:** The service is assumed to require 99.99% availability, necessitating a robust cloud infrastructure. Azure Cosmos DB is chosen for its multi-region replication and SLAs.

**Data Integrity:** Ensuring strict consistency after write operations is critical. The use of ETags and Optimistic Concurrency in Cosmos DB is assumed to prevent data conflicts.

**Security Compliance:** The system assumes compliance with industry-standard security practices, leading to the adoption of Azure Key Vault for managing sensitive information, adhering to the principle of least privilege and secure secret rotation policies.

**Global Distribution:** The service is assumed to cater to a global audience, requiring geo-replication and traffic routing capabilities. Azure Traffic Manager is presumed for routing users to the nearest data center hosting the service.

**Operational Monitoring:** The assumption that operational visibility is crucial for a public-facing service has guided the integration of Azure Application Insights for real-time monitoring and anomaly detection.

**Data Volume:** Handling billions of entities necessitates a data store with horizontal scaling capabilities. Cosmos DB's partitioning is assumed to efficiently distribute data and handle large volumes.

**Cache Coherence:** To maintain high performance with data integrity, a pattern of cache-aside in conjunction with Azure Redis Cache is assumed to keep frequently accessed data in sync with the primary data store.

**Disaster Recovery:** The design assumes a requirement for a comprehensive disaster recovery strategy. Azure's regional pairs and automated backups are presumed to be part of the solution to ensure business continuity.

**Load Adaptability:** The system assumes variable load patterns, necessitating a responsive scaling strategy. Azure's auto-scaling features are presumed to handle sudden spikes and drops in traffic.

# Architecture Diagram

**This can be found as a pdf in my github repository. Reason for it being in pdf is because I used Vsio to create the diagram and it's quite complex. As such it would be very compacted if I had tried to attach it as a picture here.**

**Location for architecture diagram in github repo: It's located in the "Documentation" folder. I have also emailed the diagram separately in my email.**

# Test Cases Outline

**Here's the schema for all the Employee object:**


****It's very important to make sure that the id and departmentId have the same value because departmentId is the partition key in cosmosDB. ****

**1) Create (POST /Employees) Test Cases:**

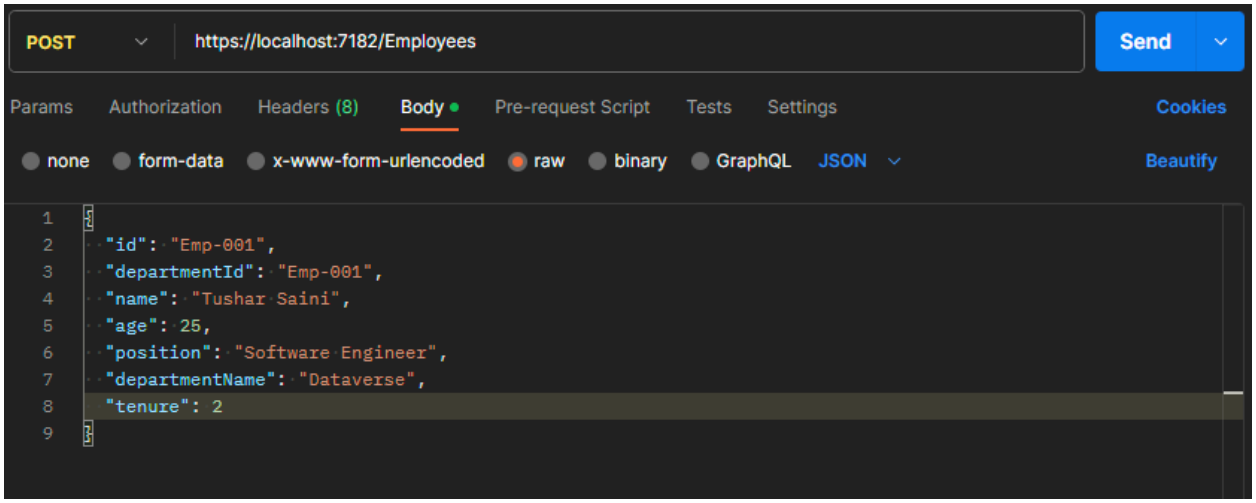
**1.1) Valid Creation:**
**Description:** Verify that creating a new employee with valid data results in a successful entry in the database.
**Steps:**

- Using Postman, send a POST request to http://localhost:port/Employees with a Employee object in the JSON request body.
- Check the HTTP status code in the response (expected: 201 Created).
- Extract the Location header from the response and store it.
- Send a GET request to the stored Location URL to retrieve the newly created employee.

**Expected Outcomes:**

- HTTP status code: 201 Created
- The retrieved employee matches the one sent in the request.

**Example of Valid Creation:**





**1.2) Invalid Data Handling:**

**Description:** Ensure that the service handles requests with incomplete or invalid data gracefully.

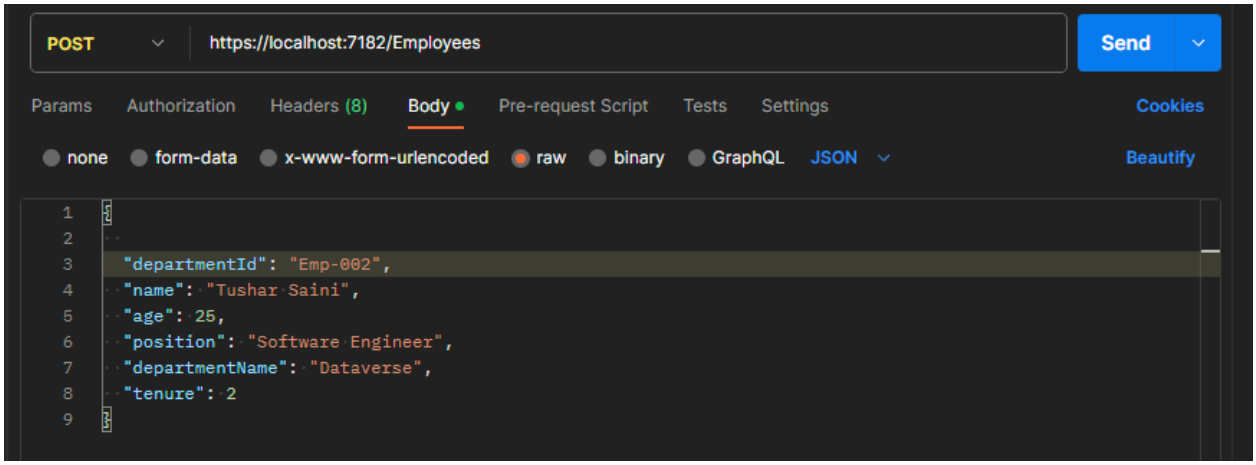
**Steps:**

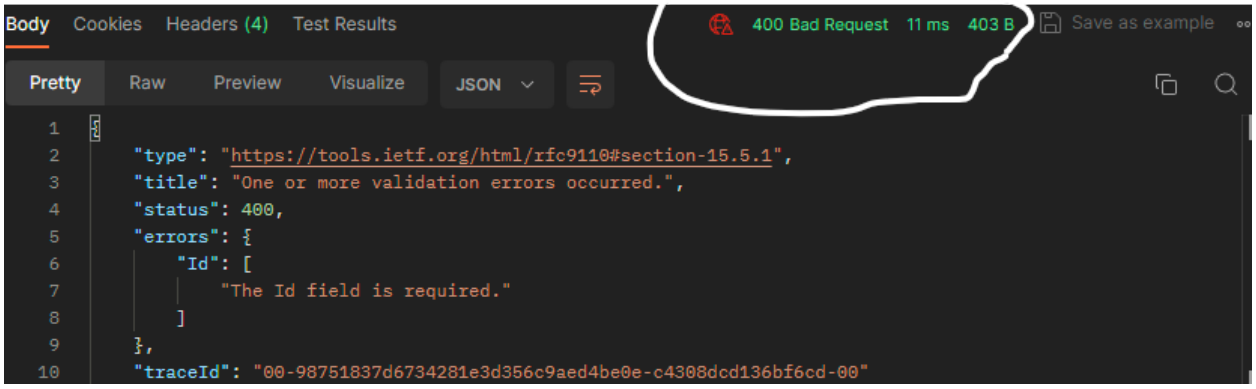
- Using Postman, send a POST request to http://localhost:port/Employees with an Employee object missing required fields in the request body.
- Check the HTTP status code in the response (expected: 400 Bad Request).
- Verify that the response body contains an error message indicating missing data.

**Expected Outcomes:**

- HTTP status code: 400 Bad Request
- Response body contains an error message.

**Example of invalid creation:**




**1.3) Duplicate Entry:**

**Description:** Test the service's response when attempting to create an employee with a duplicate ID.

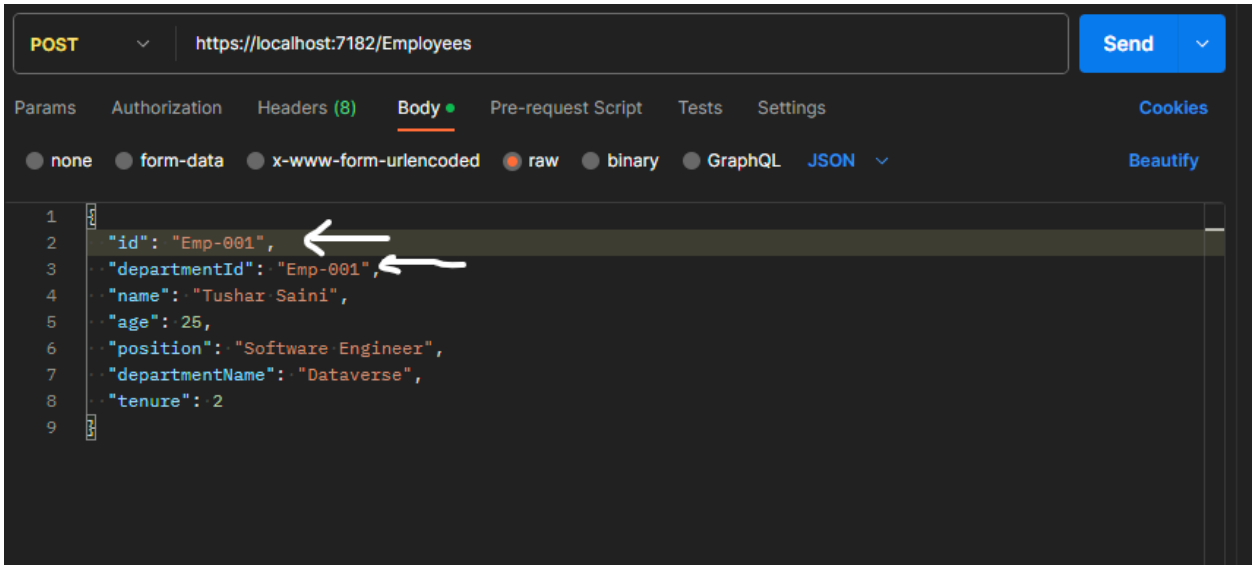
**Steps:**

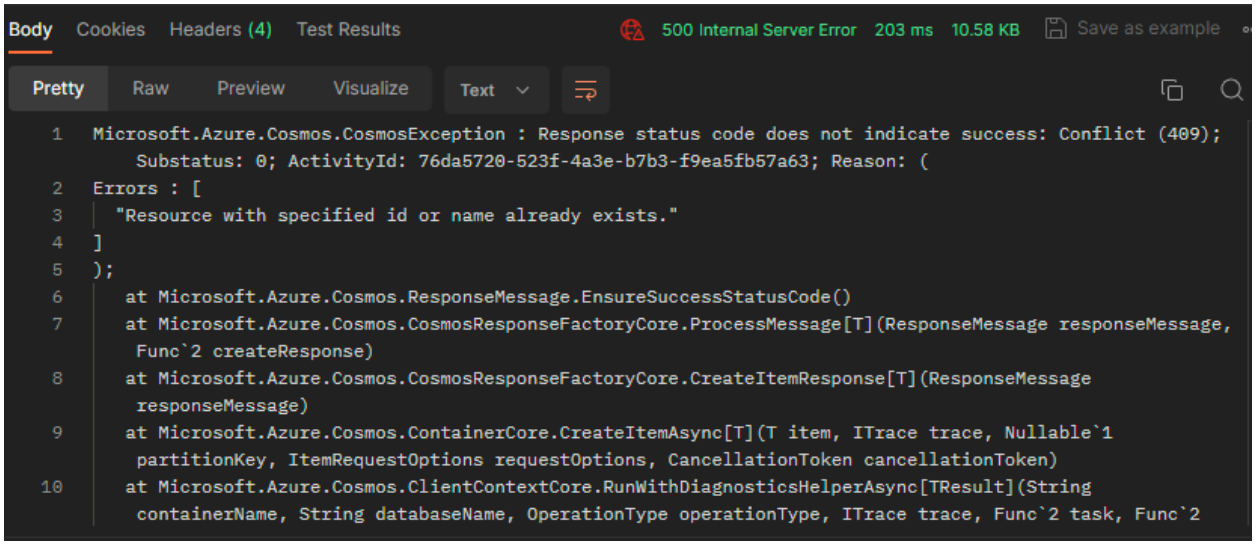
- Create an employee with a specific id using a POST request as in the "Valid Creation" test.
- Using Postman, send another POST request to http://localhost:port/Employees with the same id in the Employee object in the request body.
- Check the HTTP status code in the response (expected: 409 Conflict).

**Expected Outcomes:**

- HTTP status code: 409 Conflict

**Example of duplicate data entry**




## 2) Read (GET /Employees) Test Cases:

### 2.1) Valid Retrieval:

**Description:** Verify that retrieving an existing employee by their unique ID results in a successful response.

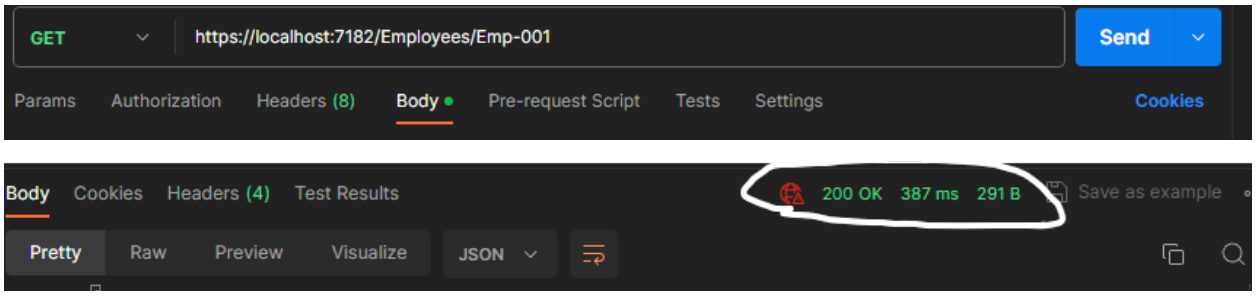
**Steps:**

- Using Postman, send a GET request to http://localhost:port/Employees/{employeeId} where {employeeId} is the ID of an existing employee in the database.
- Check the HTTP status code in the response (expected: 200 OK).
- Verify that the response body contains the details of the employee.

**Expected Outcomes:**

- HTTP status code: 200 OK
- The response body contains the details of the employee.

**Example of valid retrieval:**




**2.2) Non-Existent Employee:**

**Description:** Test the service's response when attempting to retrieve an employee with an ID that does not exist in the database.

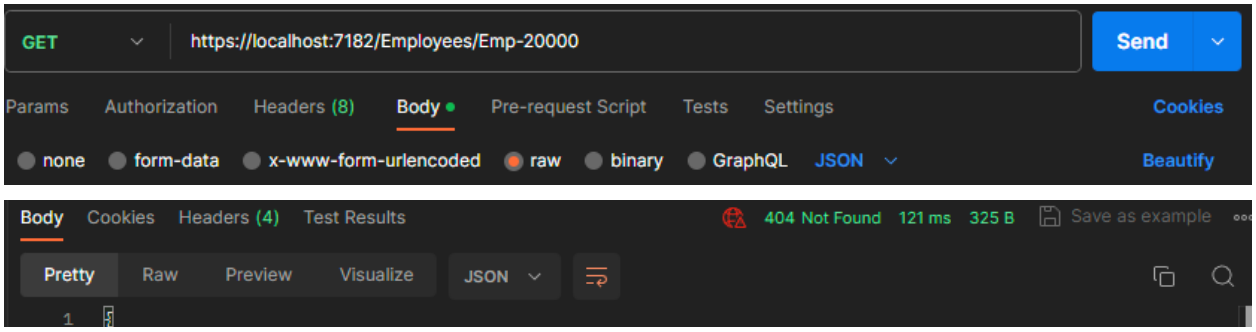
**Steps:**

- Using Postman, send a GET request to http://localhost:port/Employees/{nonExistentId} where {nonExistentId} is an ID that does not exist in the database.
- Check the HTTP status code in the response (expected: 404 Not Found).

**Expected Outcomes:**

- HTTP status code: 404 Not Found

**Example of non-existent employee**




**2.3) List All Employees:**

**Description:** Verify that retrieving a list of all employees returns the expected results.

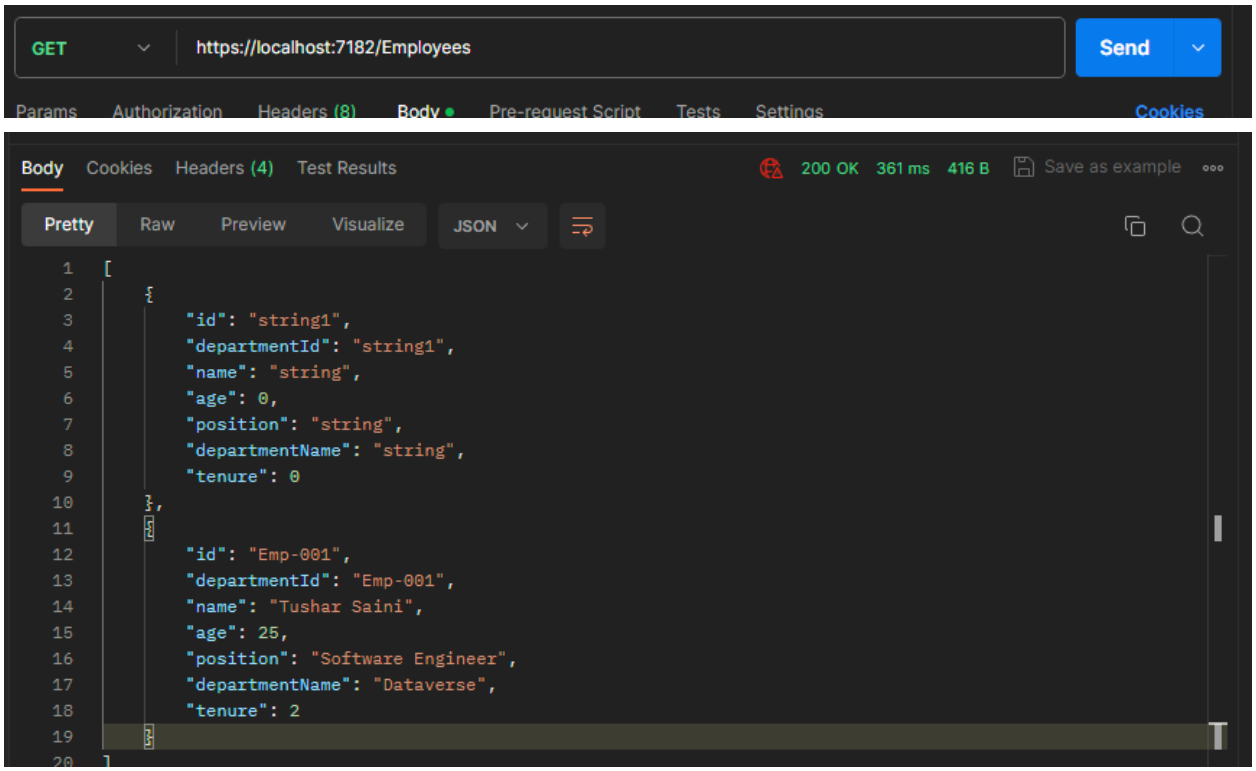
**Steps:**

- Using Postman, send a GET request to http://localhost:port/Employees.
- Check the HTTP status code in the response (expected: 200 OK).
- Verify that the response body contains a list of employee details.

**Expected Outcomes:**

- HTTP status code: 200 OK
- The response body contains a list of employee details.

**Example:**




## 3) Update (PUT /Employees/{id}) Test Cases:

### 3.1) Valid Update:

**Description:** Verify that updating an existing employee's information with valid data results in a successful update.

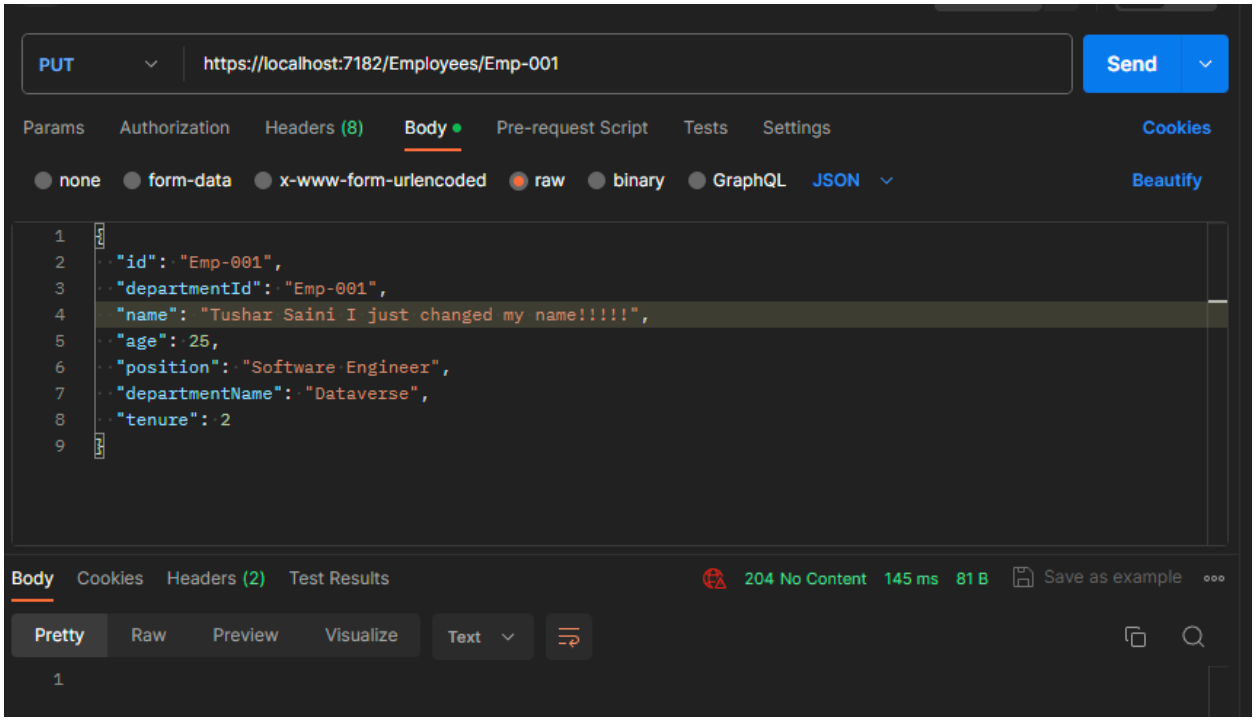
**Steps:**

- Create an employee with specific details using a POST request as in the "Valid Creation" test.
- Using Postman, send a PUT request to http://localhost:port/Employees/{employeeId} where {employeeId} is the ID of the created employee, with updated details in the request body.
- Check the HTTP status code in the response (expected: 204 No Content).
- Send a GET request to http://localhost:port/Employees/{employeeId} to retrieve the updated employee.

**Expected Outcomes:**

- HTTP status code: 204 No Content
- The retrieved employee matches the updated details.

**Example:**




**3.2) Invalid Update:**

**Description:** Test the service's response when attempting to update an employee's information with invalid data.

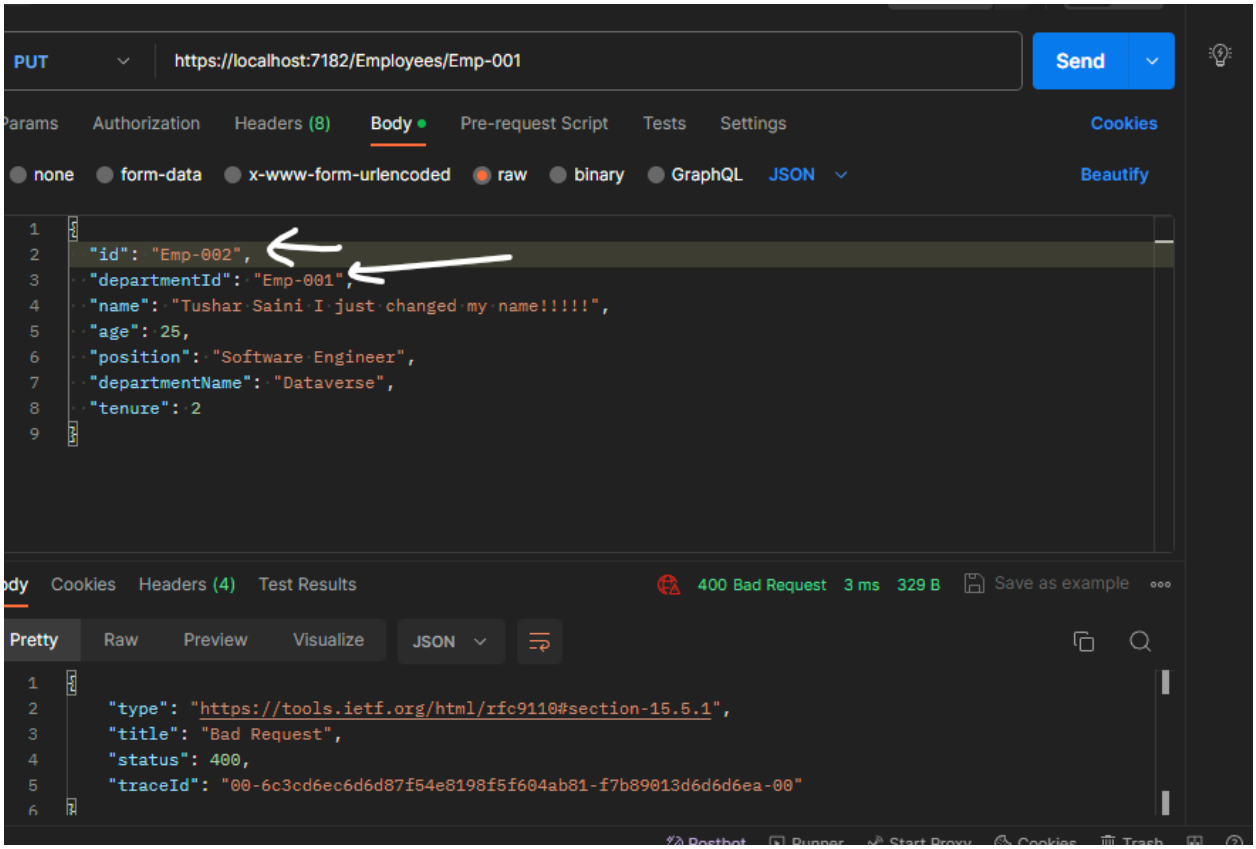
**Steps:**

- Using Postman, send a PUT request to http://localhost:port/Employees/{employeeId} where {employeeId} is the ID of an existing employee, with incomplete or invalid data in the request body.
- Check the HTTP status code in the response (expected: 400 Bad Request).

**Expected Outcomes:**

- HTTP status code: 400 Bad Request

**Example:**




## 4) Delete (DELETE /Employees/{id}) Test Cases:

### 4.1)Valid Deletion:

**Description:** Verify that deleting an existing employee by their ID results in a successful deletion.

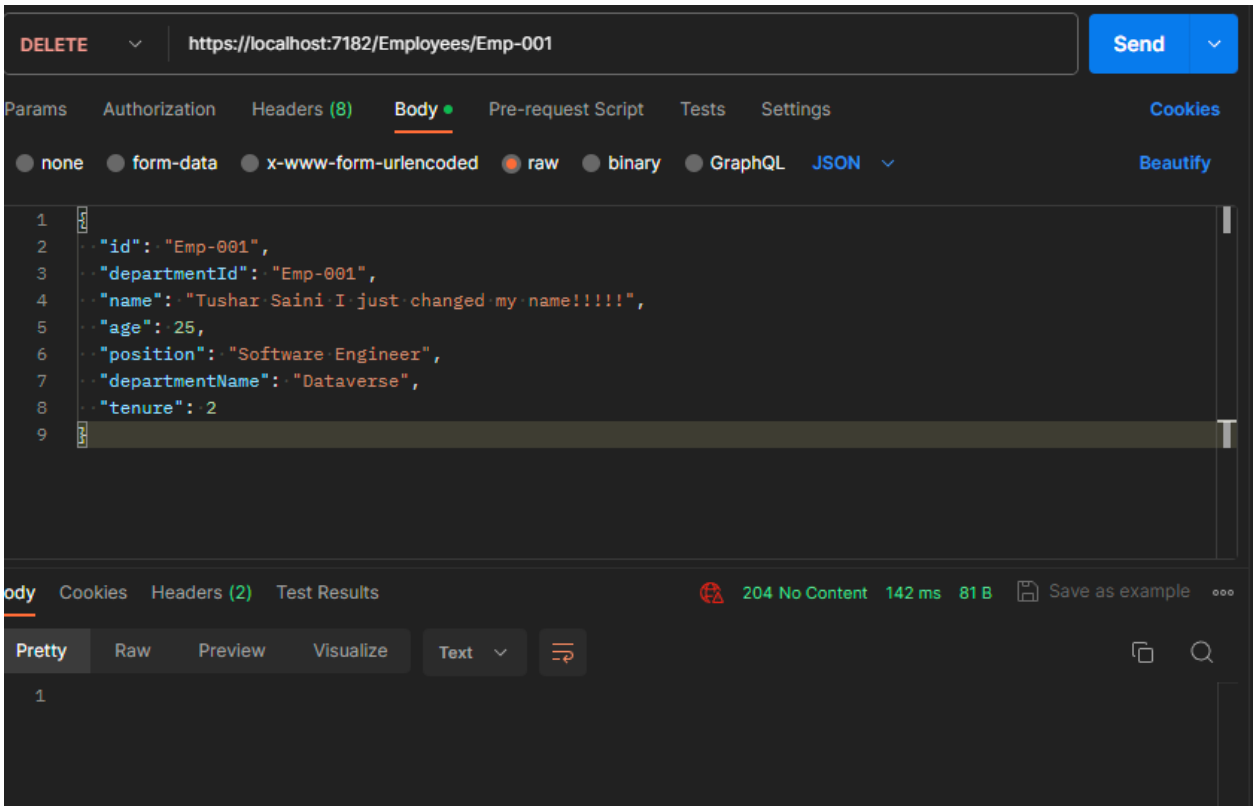
**Steps:**

- Create an employee with specific details using a POST request as in the "Valid Creation" test.
- Using Postman, send a DELETE request to http://localhost:port/Employees/{employeeId} where {employeeId} is the ID of the created employee.
- Check the HTTP status code in the response (expected: 204 No Content).
- Attempt to retrieve the deleted employee using a GET request to http://localhost:port/Employees/{employeeId}.

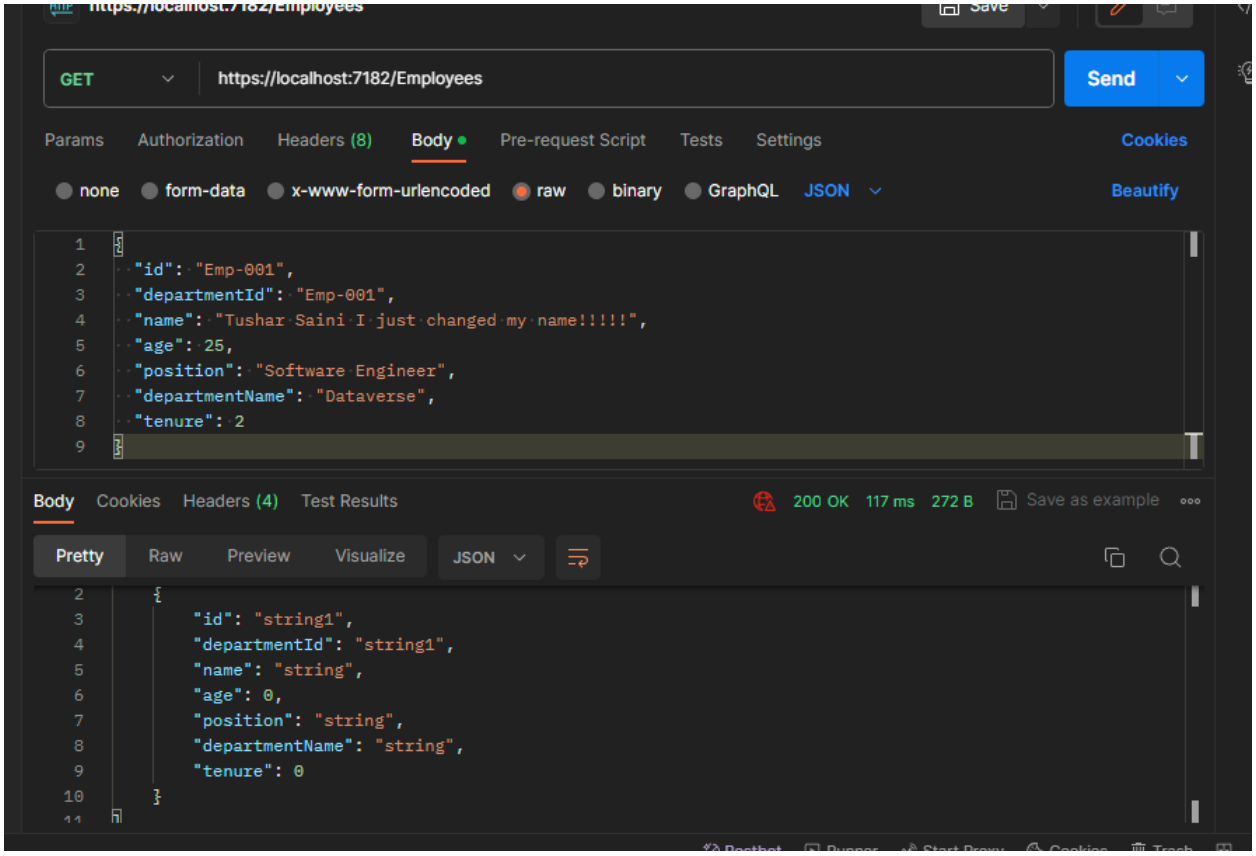
**Expected Outcomes:**

- HTTP status code: 204 No Content
- The attempt to retrieve the deleted employee results in a 404 Not Found.

**Example:**


DELETE
https://localhost:7182/Employees/Emp-001
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id": "Emp-001",
"departmentId": "Emp-001",
"name": "Tushar Saini I just changed my name!!!!!",
"age": 25,
"position": "Software Engineer",
"departmentName": "Dataverse",
"tenure": 2
}
ody
Cookies
Headers (2)
Test Results
204 No Content
142 ms
81 B
Save as example
Pretty
Raw
Preview
Visualize
Text
1

If you do a get employee request you will notice that the record is missing now:




**4.2) Non-Existent Employee Deletion:**

**Description:** Test the service's response when attempting to delete an employee that does not exist in the database.

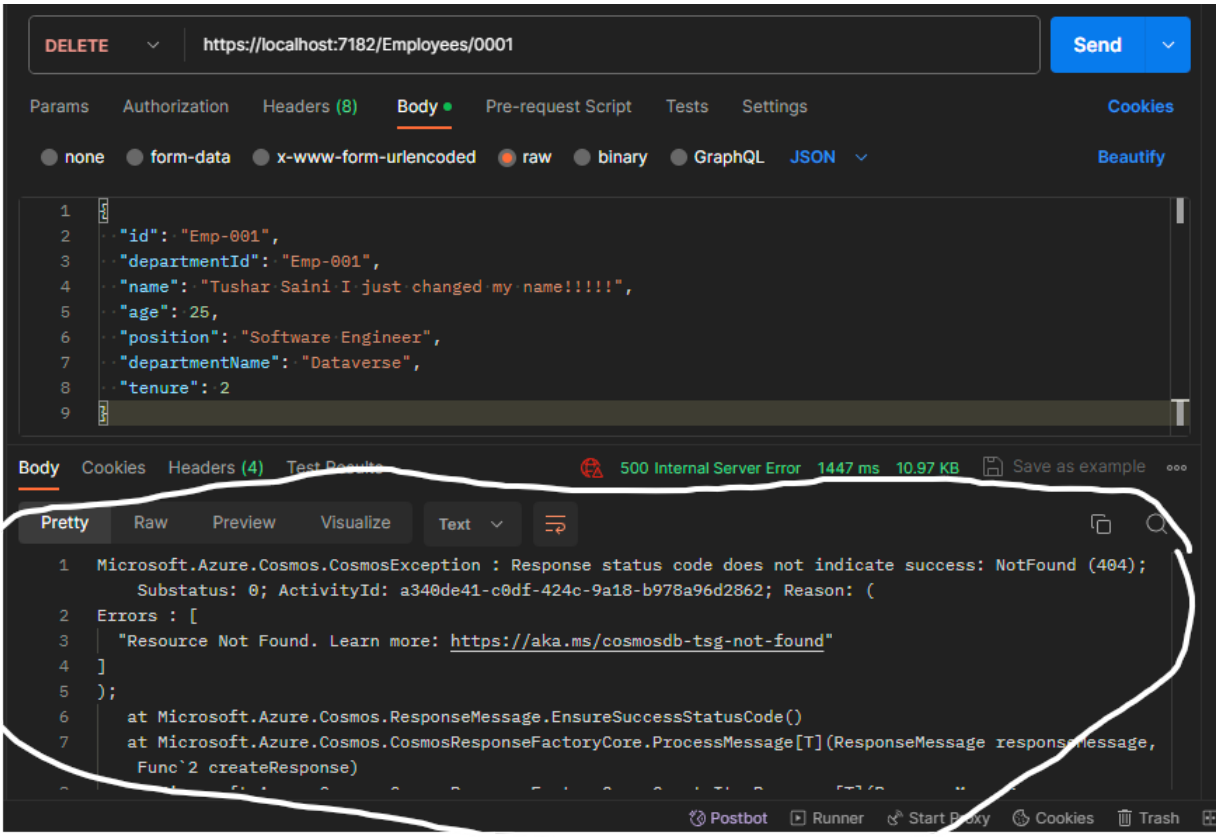
**Steps:**

- Using Postman, send a DELETE request to http://localhost:port/Employees/{nonExistentId} where {nonExistentId} is an ID that does not exist in the database.
- Check the HTTP status code in the response (expected: 404 Not Found).

**Expected Outcomes:**

- HTTP status code: 404 Not Found

**Example:**


DELETE
https://localhost:7182/Employees/0001
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id": "Emp-001",
"departmentId": "Emp-001",
"name": "Tushar Saini I just changed my name!!!!!",
"age": 25,
"position": "Software Engineer",
"departmentName": "Dataverse",
"tenure": 2
}
Body
Cookies
Headers (4)
500 Internal Server Error
1447 ms
10.97 KB
Save as example
Pretty
Raw
Preview
Visualize
Text
Microsoft.Azure.Cosmos.CosmosException : Response status code does not indicate success: NotFound (404);
Substatus: 0; ActivityId: a340de41-c0df-424c-9a18-b978a96d2862; Reason: (
Errors : [
"Resource Not Found. Learn more: https://aka.ms/cosmosdb-tsg-not-found"
]
);
at Microsoft.Azure.Cosmos.ResponseMessage.EnsureSuccessStatusCode()
at Microsoft.Azure.Cosmos.CosmosResponseFactoryCore.ProcessMessage[T](ResponseMessage responseMessage,
Func`2 createResponse)
Postbot
Runner
Cookies
Trash