Employee Management System

1.JSON Schema Definitions

1. Employee Request JSON Schema

This defines what a client sends to create or update an employee.

```
{
  "$schema": "http://json-schema.org/draft-
07/schema#",
  "title": "EmployeeRequest",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "dob": { "type": "string", "format": "date" },
    "salary": { "type": "number" },
    "departmentId": { "type": ["integer", "null"]
},
    "address": { "type": "string" },
    "role": { "type": "string" },
    "joiningDate": { "type": "string", "format":
"date" },
    "yearlyBonusPercentage": { "type": "number" },
    "reportingManagerId": { "type": ["integer",
"null"] }
  },
  "required": ["name", "dob", "salary",
"joiningDate"]
}
```

2.Employee Response JSON Schema

This defines what your API returns after creating or fetching an employee.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "EmployeeResponse",
  "type": "object",
  "properties": {
    "id": { "type": "integer" },
    "name": { "type": "string" },
    "dob": { "type": "string", "format": "date" },
    "salary": { "type": "number" },
    "department": {
      "type": ["object", "null"],
      "properties": {
        "id": { "type": "integer" },
        "name": { "type": "string" }
      }
    },
    "address": { "type": "string" },
    "role": { "type": "string" },
    "joiningDate": { "type": "string", "format": "date" },
    "yearlyBonusPercentage": { "type": "number" },
    "reportingManager": {
      "type": ["object", "null"],
      "properties": {
        "id": { "type": "integer" },
        "name": { "type": "string" }
      }
    }
  }
}
```

3.Department JSON Schema

Request

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "DepartmentRequest",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "createdDate": { "type": "string", "format": "date"
},
    "departmentHeadId": { "type": "integer" }
  },
  "required": ["name", "createdDate"]
}
```

Response

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "DepartmentResponse",
  "type": "object",
  "properties": {
    "id": { "type": "integer" },
    "name": { "type": "string" },
    "createdDate": { "type": "string", "format": "date" },
    "departmentHead": {
      "type": ["object", "null"],
      "properties": {
        "id": { "type": "integer" },
        "name": { "type": "string" }
      }
    }
  }
}
```

2.Database Structure

2.1 Department Table

Column Name	Data Type	Constraints	Description
id	BIGINT (PK)	Primary Key, Auto Increment	Unique ID for each department
name	VARCHAR(100)	NOT NULL, UNIQUE	Department name
created_date	DATE	NOT NULL	Date when department was created
department_head_id	BIGINT (FK)	Foreign Key → employees(id)	The employee who is the head of the department

Relationships:

- One **Department** → Many **Employees**
- Department head is one of the employees (self-reference via employee_id)

1.2 Employee Table

Column Name	Data Type	Constraints	Description
id	BIGINT (PK)	Primary Key, Auto Increment	Unique employee ID
name	VARCHAR(100)	NOT NULL	Employee full name
dob	DATE	NOT NULL	Date of birth
salary	DECIMAL(10,2)	NOT NULL	Employee salary

address	VARCHAR(255)		Employee address
role	VARCHAR(100)		Job role or title
joining_date	DATE	NOT NULL	Date when employee joined
yearly_bonus_percentage	DECIMAL(5,2)	DEFAULT 0	Yearly bonus percentage
department_id	BIGINT (FK)	Foreign Key → departments(id)	Department the employee belongs to
reporting_manager_id	BIGINT (FK)	Foreign Key → employees(id) (self)	Employee's manager ID

Relationships:

- Many **Employees** belong to one **Department** (department_id)
- One **Employee** can report to another **Employee** (reporting_manager_id)

2. API Logic Implementation

Description:
This section demonstrates the implementation of RESTful APIs to manage **Employee** and **Department** entities.The APIs include operations for **creating**, **updating**, **retrieving**, and **deleting** records with proper **error handling** and **validation**.

3.1) Employee

3.1.1) EmployeeController.java

```
package com.example.employee_management.Controller;

import com.example.employee_management.Entity.Employee;
import com.example.employee_management.Service.EmployeeService;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/employees")
public class EmployeeController {
   private final EmployeeService employeeService;
```

```
    public EmployeeController(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    // Create employee
    @PostMapping
    public Employee create(@RequestBody Employee employee) {
        return employeeService.addEmployee(employee);
    }

    // Get all employees with pagination and lookup
    @GetMapping
    public Map<String, Object> getAll(
            @RequestParam(required = false) Boolean lookup,
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "20") int size) {

        if (Boolean.TRUE.equals(lookup)) {
            // Return only ID and Name
            List<Map<String, Object>> employees =
employeeService.getEmployeeIdAndName();
            return Map.of(
                    "content", employees,
                    "page", 0,
                    "totalPages", 1,
                    "totalItems", employees.size()
            );
        }

        // Paginated full details
        Page<Employee> employeePage = employeeService.getPaginatedEmployees(page,
size);

        return Map.of(
                "content", employeePage.getContent(),
                "page", employeePage.getNumber(),
                "totalPages", employeePage.getTotalPages(),
                "totalItems", employeePage.getTotalElements()
        );
    }

    // Update employee
    @PutMapping("/{id}")
    public Employee update(@PathVariable Long id, @RequestBody Employee employee)
{
        return employeeService.updateEmployee(id, employee);
    }
```

```
  // Delete employee
  @DeleteMapping("/{id}")
  public void delete(@PathVariable Long id) {
    employeeService.deleteEmployee(id);
  }

  // Move an employee to another department
  @PatchMapping("/{employeeId}/department/{departmentId}")
  public Employee moveEmployeeToDepartment(@PathVariable Long employeeId,
@PathVariable Long departmentId) {
    return employeeService.moveEmployeeToDepartment(employeeId, departmentId);
  }
}
```

3.1.2) EmployeeService.java

```
package com.example.employee_management.Service;

import com.example.employee_management.Entity.Department;
import com.example.employee_management.Entity.Employee;
import
com.example.employee_management.Repository.DepartmentRepository;
import com.example.employee_management.Repository.EmployeeRepository;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;


import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Service
public class EmployeeService {
    private final EmployeeRepository employeeRepository;
    private final DepartmentRepository departmentRepository;

    public EmployeeService(EmployeeRepository employeeRepository,
                DepartmentRepository departmentRepository) {
      this.employeeRepository = employeeRepository;
      this.departmentRepository = departmentRepository;
    }
```

```
    // Add a new employee
    public Employee addEmployee(Employee employee) {

        // Set Department from departmentId
        if (employee.getDepartmentId() != null) {
            Department dept =
departmentRepository.findById(employee.getDepartmentId())
                    .orElseThrow(() -> new RuntimeException("Department not found"));
            employee.setDepartment(dept);
        }

        // Set Reporting Manager from reportingManagerId
        if (employee.getReportingManagerId() != null) {
            Employee manager =
employeeRepository.findById(employee.getReportingManagerId())
                    .orElseThrow(() -> new RuntimeException("Reporting Manager not
found"));
            employee.setReportingManager(manager);
        }

        return employeeRepository.save(employee);
    }

    // Get all employees
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    // Update an employee
    public Employee updateEmployee(Long id, Employee updatedEmployee) {
        Employee existing = employeeRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Employee not found"));

        existing.setName(updatedEmployee.getName());
        existing.setSalary(updatedEmployee.getSalary());
        existing.setRole(updatedEmployee.getRole());
        existing.setAddress(updatedEmployee.getAddress());
        existing.setDob(updatedEmployee.getDob());
        existing.setJoiningDate(updatedEmployee.getJoiningDate());

existing.setYearlyBonusPercentage(updatedEmployee.getYearlyBonusPercentag
e());

        // Update Department
```

```
        if (updatedEmployee.getDepartmentId() != null) {
          Department dept =
departmentRepository.findById(updatedEmployee.getDepartmentId())
              .orElseThrow(() -> new RuntimeException("Department not found"));
          existing.setDepartment(dept);
        } else {
          // explicitly set department to null if departmentId is null
          existing.setDepartment(null);
        }

        // Update Reporting Manager
        if (updatedEmployee.getReportingManagerId() != null) {
          Employee manager =
employeeRepository.findById(updatedEmployee.getReportingManagerId())
              .orElseThrow(() -> new RuntimeException("Reporting Manager not
found"));
          existing.setReportingManager(manager);
        } else {
          // explicitly set reporting manager to null if reportingManagerId is null
          existing.setReportingManager(null);
        }

        return employeeRepository.save(existing);
      }

      // Delete an employee
      public void deleteEmployee(Long id) {
        employeeRepository.deleteById(id);
      }

    // Move employee to another department
    public Employee moveEmployeeToDepartment(Long employeeId, Long
departmentId) {
      Employee employee = employeeRepository.findById(employeeId)
          .orElseThrow(() -> new RuntimeException("Employee not found"));

      Department newDepartment =
departmentRepository.findById(departmentId)
          .orElseThrow(() -> new RuntimeException("Department not found"));

      employee.setDepartment(newDepartment);

      return employeeRepository.save(employee);
    }
```

```
    // Paginated employees
    public Page<Employee> getPaginatedEmployees(int page, int size) {
        Pageable pageable = PageRequest.of(page, size);
        return employeeRepository.findAll(pageable);
    }

    // Lookup: return only id and name
    public List<Map<String, Object>> getEmployeeIdAndName() {
        return employeeRepository.findAll()
                .stream()
                .map(emp -> {
                    Map<String, Object> map = new HashMap<>();
                    map.put("id", emp.getId());
                    map.put("name", emp.getName());
                    return map;
                })
                .collect(Collectors.toList());
    }

}
```

3.1.3) EmployeeEntity.java

```
package com.example.employee_management.Entity;

import jakarta.persistence.*;
import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.annotation.JsonSetter;
import java.time.LocalDate;

@Entity
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @JsonFormat(pattern = "yyyy-MM-dd")
    private LocalDate dob;
```

```
private double salary;
private String address;
private String role;

@JsonFormat(pattern = "yyyy-MM-dd")
private LocalDate joiningDate;

private double yearlyBonusPercentage;

@ManyToOne
@JoinColumn(name = "department_id")
private Department department;

@ManyToOne
@JoinColumn(name = "reporting_manager_id")
@JsonIgnore  // prevents infinite recursion
private Employee reportingManager;

// Transient fields for JSON input
@Transient
private Long departmentId;

@Transient
private Long reportingManagerId;

// JSON setter for department
@JsonSetter("department")
public void setDepartmentId(Long departmentId) {
  this.departmentId = departmentId;
}

@JsonIgnore
public Long getDepartmentId() {
  return departmentId;
}

// JSON setter for reporting manager
@JsonSetter("reportingManagerId")
public void setReportingManagerId(Long reportingManagerId) {
  this.reportingManagerId = reportingManagerId;
}

@JsonIgnore
public Long getReportingManagerId() {
```

```
    return reportingManagerId;
  }

  // Constructors
  public Employee() {}

  public Employee(String name, LocalDate dob, double salary, String role) {
    this.name = name;
    this.dob = dob;
    this.salary = salary;
    this.role = role;
  }

  // Regular getters and setters
  public Long getId() { return id; }
  public void setId(Long id) { this.id = id; }

  public String getName() { return name; }
  public void setName(String name) { this.name = name; }

  public LocalDate getDob() { return dob; }
  public void setDob(LocalDate dob) { this.dob = dob; }

  public double getSalary() { return salary; }
  public void setSalary(double salary) { this.salary = salary; }

  public String getAddress() { return address; }
  public void setAddress(String address) { this.address = address; }

  public String getRole() { return role; }
  public void setRole(String role) { this.role = role; }

  public LocalDate getJoiningDate() { return joiningDate; }
  public void setJoiningDate(LocalDate joiningDate) { this.joiningDate =
joiningDate; }

  public double getYearlyBonusPercentage() { return yearlyBonusPercentage; }
  public void setYearlyBonusPercentage(double yearlyBonusPercentage) {
    this.yearlyBonusPercentage = yearlyBonusPercentage;
  }

  public Department getDepartment() { return department; }
  public void setDepartment(Department department) { this.department =
department; }
```

```
    @JsonProperty("department")
    @Transient
    public Object getDepartmentForJson() {
        if (department == null) return null;
        return new Object() {
            public Long id = department.getId();
            public String name = department.getName();
        };
    }

    public Employee getReportingManager() { return reportingManager; }
    public void setReportingManager(Employee reportingManager) {
this.reportingManager = reportingManager; }

    // Transient getter for JSON output to include reportingManagerId
    @JsonProperty("reportingManagerId")
    @Transient
    public Long getReportingManagerIdForJson() {
        return reportingManager != null ? reportingManager.getId() : null;
    }
}
```

3.1.4) EmploeeRepository.java

```
package com.example.employee_management.Repository;

import com.example.employee_management.Entity.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

3.2) Department

3.2.1) DepartmentEntity.java

```
package com.example.employee_management.Entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.annotation.JsonSetter;
import jakarta.persistence.*;
import java.time.LocalDate;
```

```
import java.util.List;

@Entity
public class Department {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String name;
  private LocalDate createdDate;

  @OneToOne
  @JoinColumn(name = "department_head_id")
  private Employee departmentHead;

  @OneToMany(mappedBy = "department")
  @JsonIgnore // hide employees by default
  private List<Employee> employees;

  // Transient field for JSON input/output
  @Transient
  private Long departmentHeadId;

  // JSON setter for departmentHeadId
  @JsonSetter("departmentHeadId")
  public void setDepartmentHeadId(Long departmentHeadId) {
    this.departmentHeadId = departmentHeadId;
  }

  // JSON getter for departmentHeadId
  @JsonProperty("departmentHeadId")
  @Transient
  public Long getDepartmentHeadId() {
    return departmentHead != null ? departmentHead.getId() :
departmentHeadId;
  }

  // Transient getter for employees when needed in JSON
  @Transient
  @JsonProperty("employees")
  public List<Employee> getEmployeesForJson() {
    return employees;
  }
```

```
    // Constructors
    public Department() {}
    public Department(String name, LocalDate createdDate) {
        this.name = name;
        this.createdDate = createdDate;
    }

    // Regular getters and setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public LocalDate getCreatedDate() { return createdDate; }
    public void setCreatedDate(LocalDate createdDate) { this.createdDate =
createdDate; }

    public Employee getDepartmentHead() { return departmentHead; }
    public void setDepartmentHead(Employee departmentHead) {
this.departmentHead = departmentHead; }

    public List<Employee> getEmployees() { return employees; }
    public void setEmployees(List<Employee> employees) { this.employees =
employees; }
}
```

3.2.2) DepartmentController.java

```
package com.example.employee_management.Controller;

import com.example.employee_management.Entity.Department;
import com.example.employee_management.Service.DepartmentService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/departments")
public class DepartmentController {

    private final DepartmentService departmentService;

    public DepartmentController(DepartmentService departmentService) {
```

```
        this.departmentService = departmentService;
    }

    // Create a department
    @PostMapping
    public Department create(@RequestBody Department department) {
        return departmentService.addDepartment(department);
    }

    // Get all departments (supports expand)
    @GetMapping
    public List<Department> getAll(@RequestParam(required = false) String
expand) {
        List<Department> departments = departmentService.getAllDepartments();

        // Include employees if expand is "true" or "employee"
        if ("true".equalsIgnoreCase(expand) ||
"employee".equalsIgnoreCase(expand)) {
            departments.forEach(dept -> dept.getEmployees().size()); // force
initialization
        } else {
            departments.forEach(dept -> dept.setEmployees(null)); // hide employees
        }

        return departments;
    }

    // Update a department
    @PutMapping("/{id}")
    public Department update(@PathVariable Long id, @RequestBody
Department department) {
        return departmentService.updateDepartment(id, department);
    }

    // Delete a department
    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        departmentService.deleteDepartment(id);
    }
}
```

3.2.3) DepartmentService.java

```
package com.example.employee_management.Service;
```

```
import com.example.employee_management.Entity.Department;
import com.example.employee_management.Entity.Employee;
import
com.example.employee_management.Repository.DepartmentRepository;
import com.example.employee_management.Repository.EmployeeRepository;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;

import java.util.List;

@Service
public class DepartmentService {

  private final DepartmentRepository departmentRepository;
  private final EmployeeRepository employeeRepository;

  public DepartmentService(DepartmentRepository departmentRepository,
                EmployeeRepository employeeRepository) {
    this.departmentRepository = departmentRepository;
    this.employeeRepository = employeeRepository;
  }

  // Add a new department
  public Department addDepartment(Department department) {
    if (department.getDepartmentHeadId() != null) {
      Employee head =
employeeRepository.findById(department.getDepartmentHeadId())
          .orElseThrow(() -> new ResponseStatusException(
              HttpStatus.BAD_REQUEST, "Department head not found"));
      department.setDepartmentHead(head);
    }
    return departmentRepository.save(department);
  }

  // Update a department
  public Department updateDepartment(Long id, Department
updatedDepartment) {
    Department existing = departmentRepository.findById(id)
        .orElseThrow(() -> new ResponseStatusException(
            HttpStatus.NOT_FOUND, "Department not found"));

    existing.setName(updatedDepartment.getName());
```

```
    if (updatedDepartment.getDepartmentHeadId() != null) {
      Employee head =
employeeRepository.findById(updatedDepartment.getDepartmentHeadId())
          .orElseThrow(() -> new ResponseStatusException(
              HttpStatus.BAD_REQUEST, "Department head not found"));
      existing.setDepartmentHead(head);
    } else {
      existing.setDepartmentHead(null); // allow removal of department head
    }

    return departmentRepository.save(existing);
  }

  // Get all departments
  public List<Department> getAllDepartments() {
    return departmentRepository.findAll();
  }

  // Delete a department
  public void deleteDepartment(Long id) {
    Department department = departmentRepository.findById(id)
        .orElseThrow(() -> new ResponseStatusException(
            HttpStatus.NOT_FOUND, "Department not found"));

    if (department.getEmployees() != null &&
!department.getEmployees().isEmpty()) {
      throw new ResponseStatusException(
          HttpStatus.BAD_REQUEST, "Cannot delete department. Employees are
assigned to it.");
    }

    departmentRepository.deleteById(id);
  }
}
```

3.2.4) DepartmentRepository.java

```
package com.example.employee_management.Repository;

import com.example.employee_management.Entity.Department;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface DepartmentRepository extends JpaRepository<Department,
```

Long> {
}

3.3 API Logic (CRUD Operations with Error Handling)

3.3.1) Employee APIs

Method	Endpoint	Description	Request Body / Params
POST	/employees	Create a new employee	{ "name": "John", "email": "john@example.com", "departmentId": 1 }
GET	/employees	Get all employees	Optional query params: page, size
GET	/employees/{id}	Get employee by ID	Path variable id
PUT	/employees/{id}	Update an employee	{ "name": "John", "email": "john@example.com" }
DELETE	/employees/{id}	Delete an employee	Path variable id

3.3.2) Department APIs

Method	Endpoint	Description	Request Body / Params
POST	/departments	Create a new department	{ "name": "IT", "createdDate": "2025-10-24", "departmentHeadId": 5 }
GET	/departments	Get all departments	Optional query param: expand=true (to include employees)
GET	/departments/{id}	Get department by ID	Path variable id
PUT	/departments/{id}	Update a department	{ "name": "IT", "departmentHeadId": 5 }
DELETE	/departments/{id}	Delete a department	Path variable id

Department API Testing

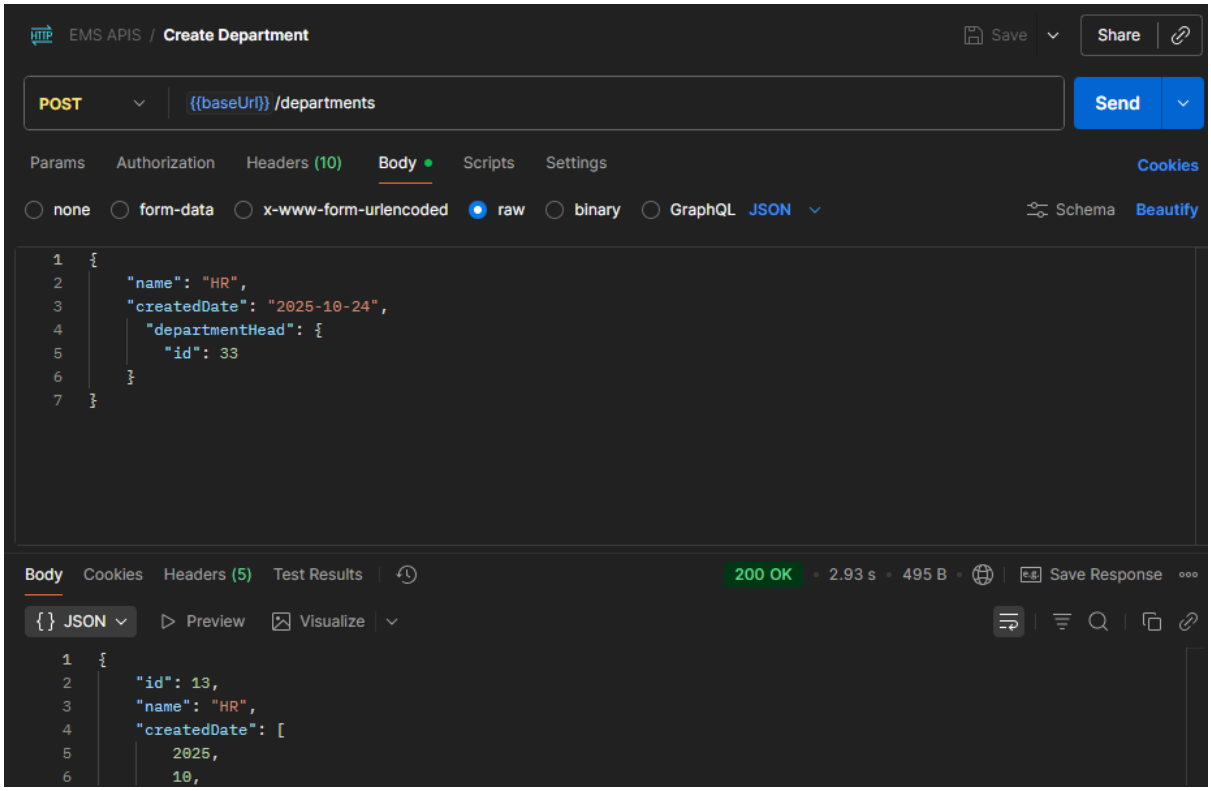

Fig 1.1 Creating a new department

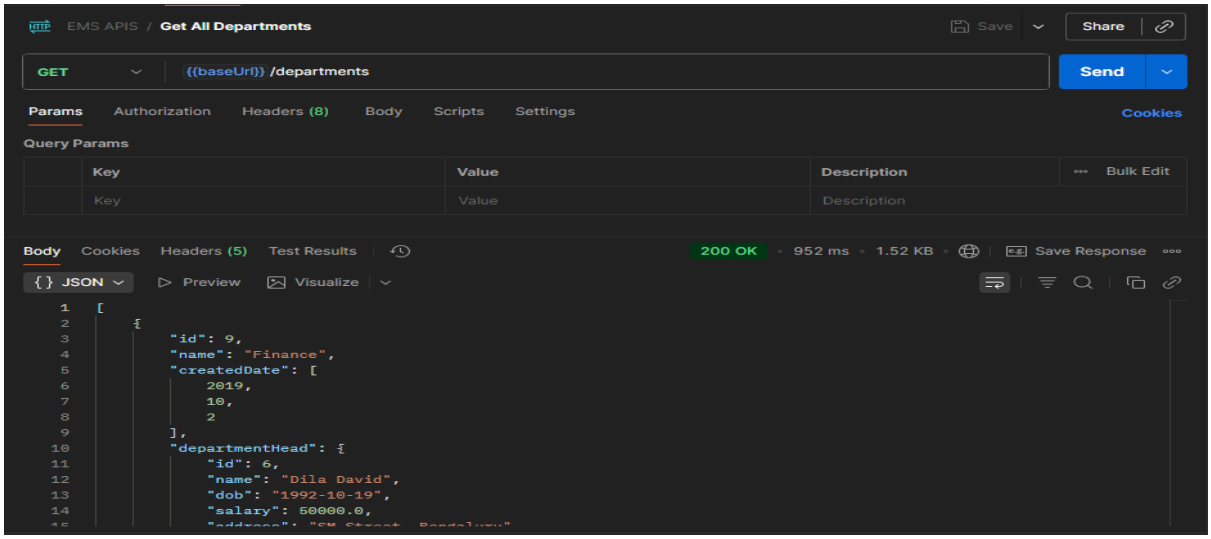

Fig 1.2 Get all Departments

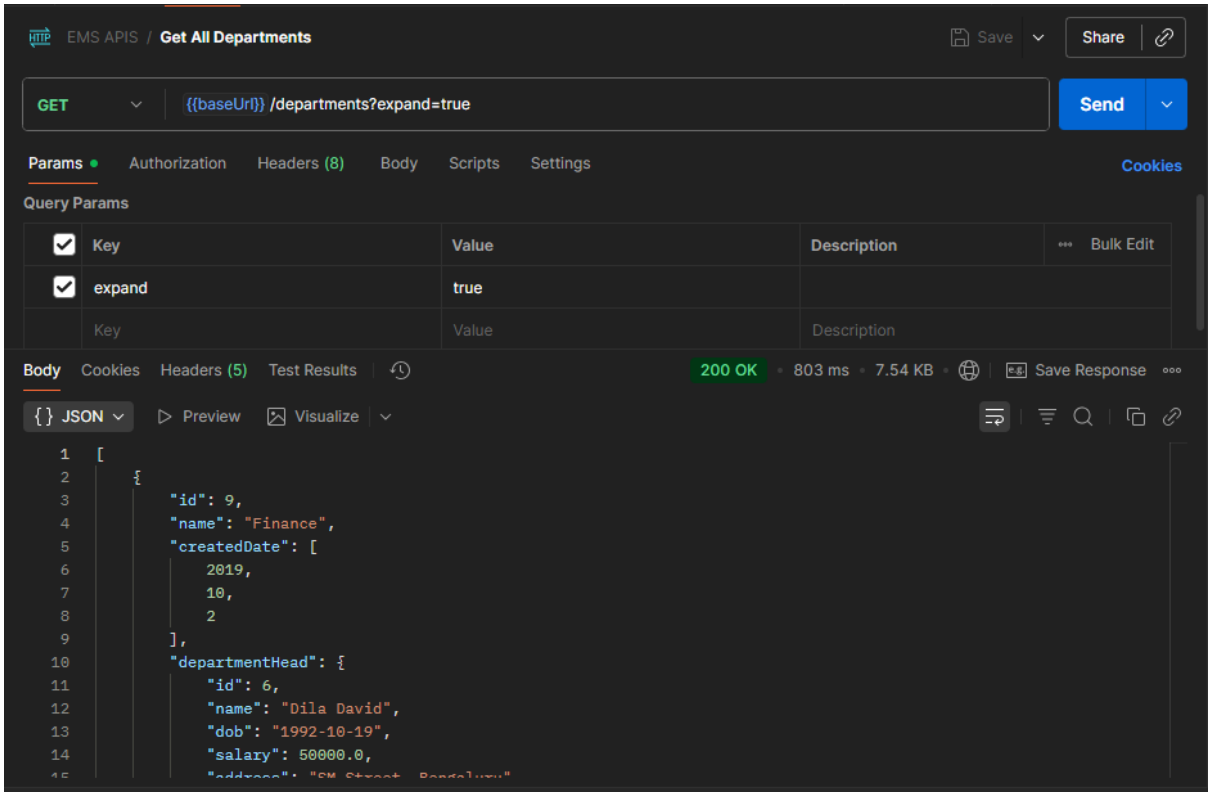

Fig 1.3 Get all Departments alond with list of employees in each department

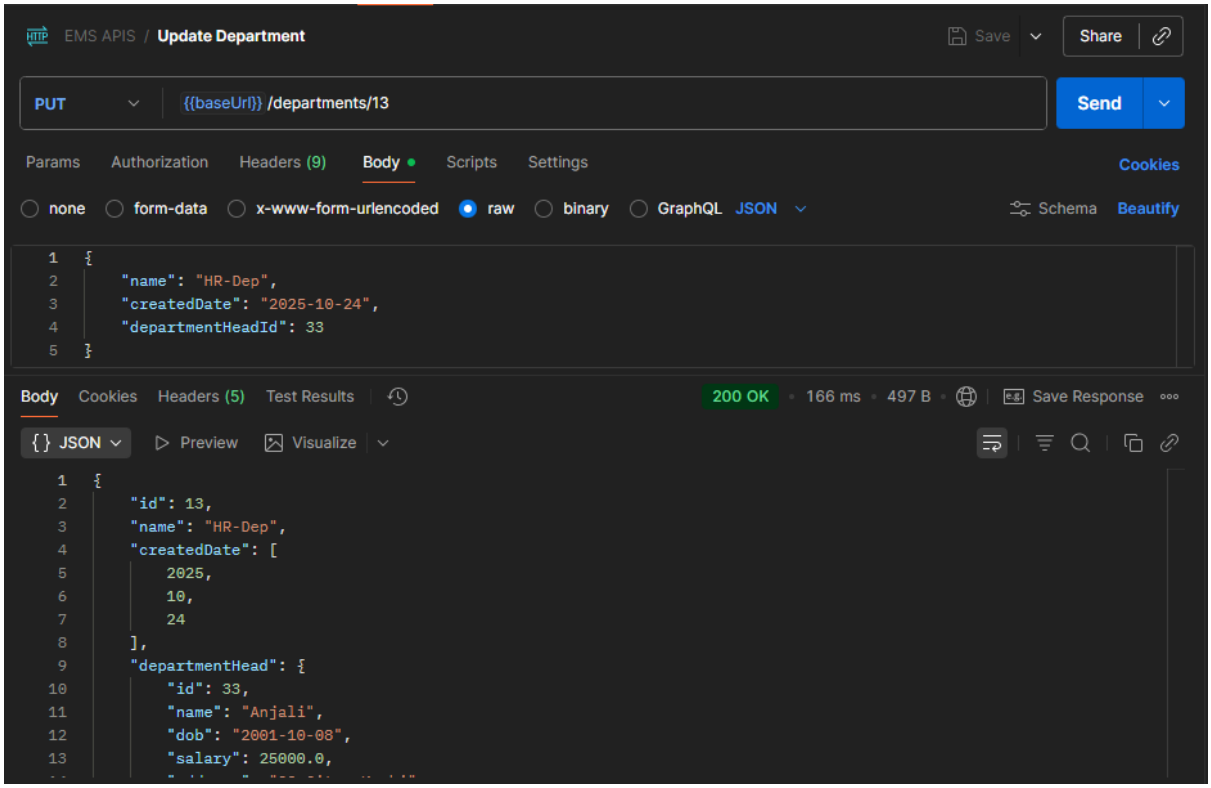

Fig 1.4 Update the department

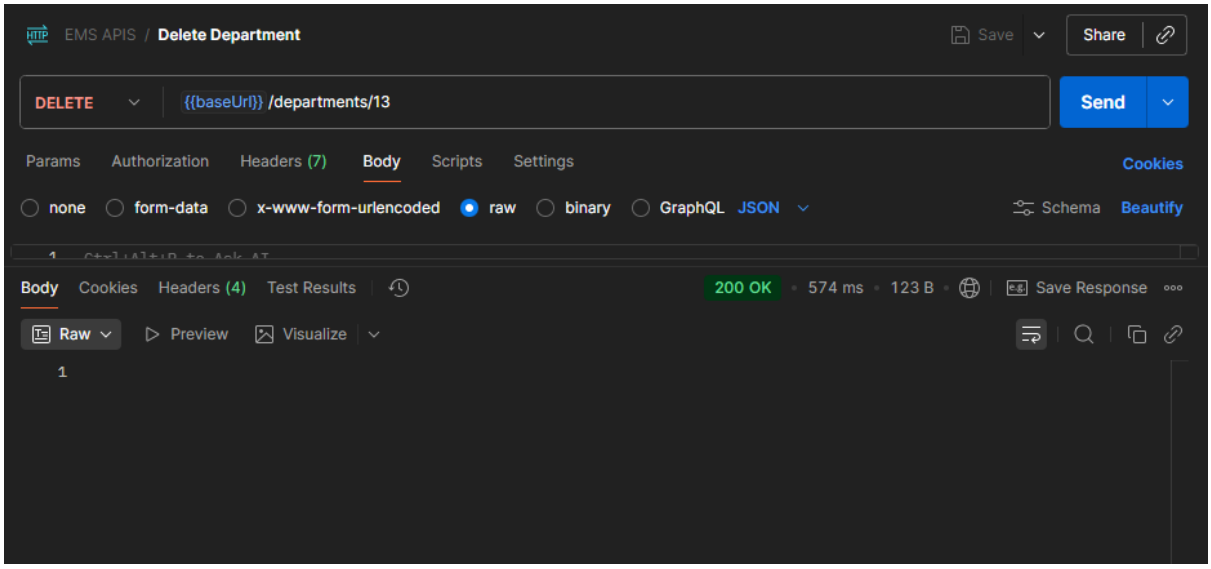

Fig 1.5 Delete the Department

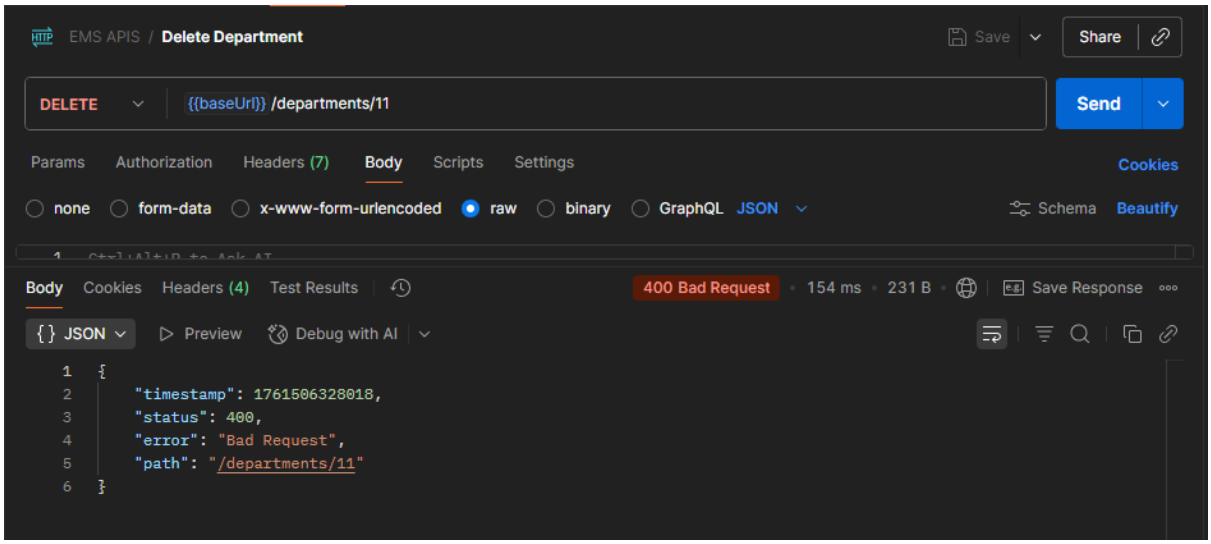

Fig 1.6 Trying to delete a department with employees

Employee API Testing

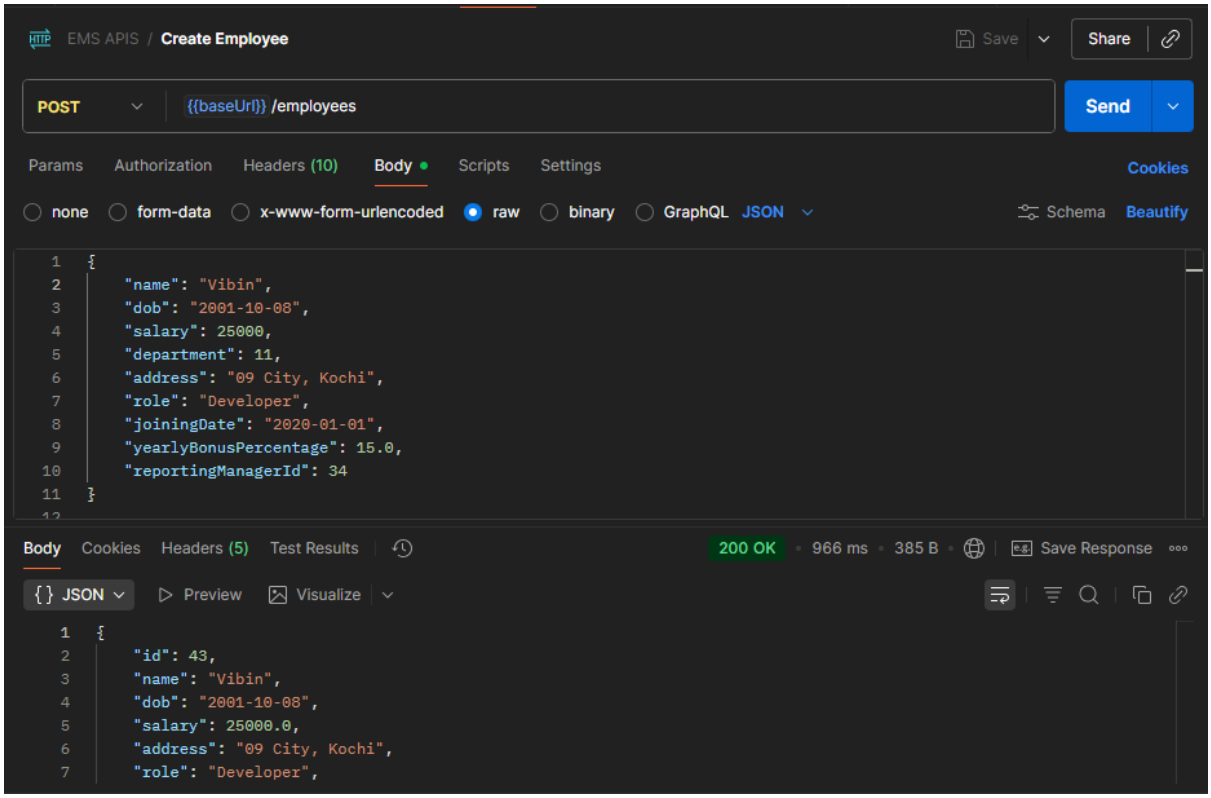

Fig1.7 create an employee

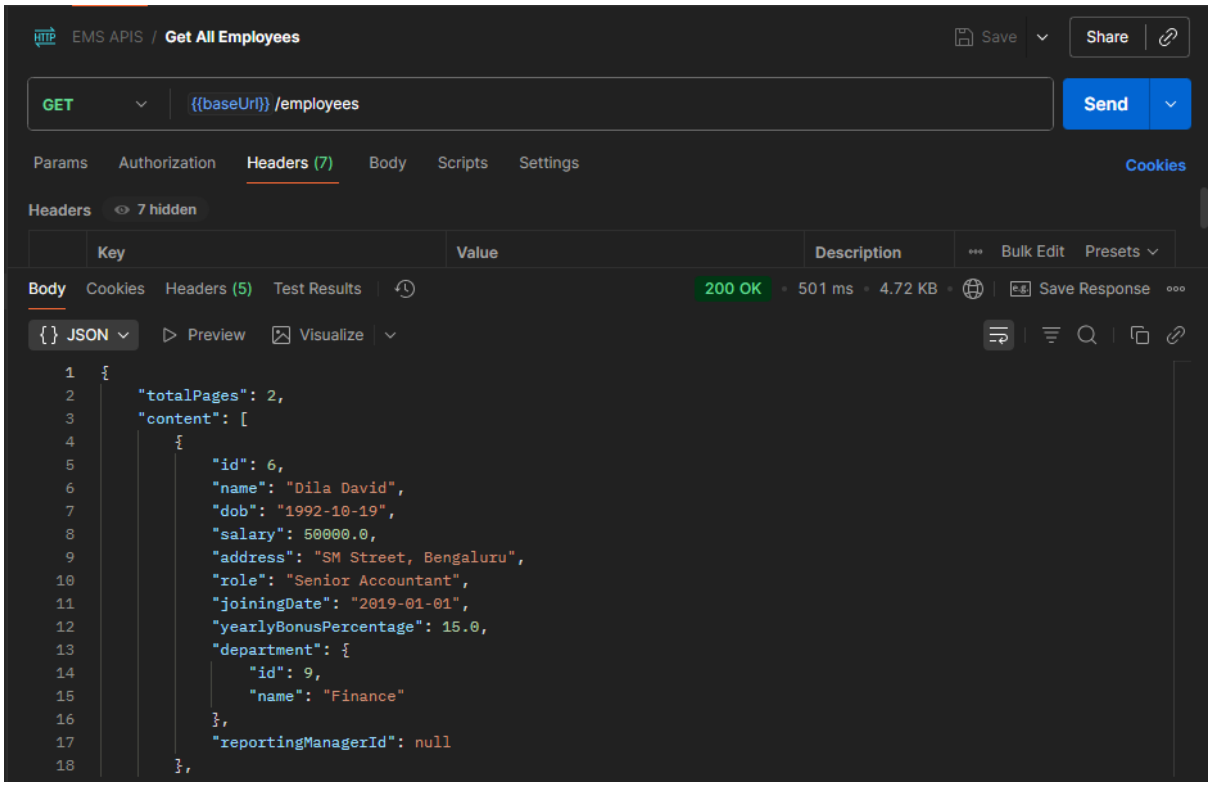

Fig 1.8 get all the employees

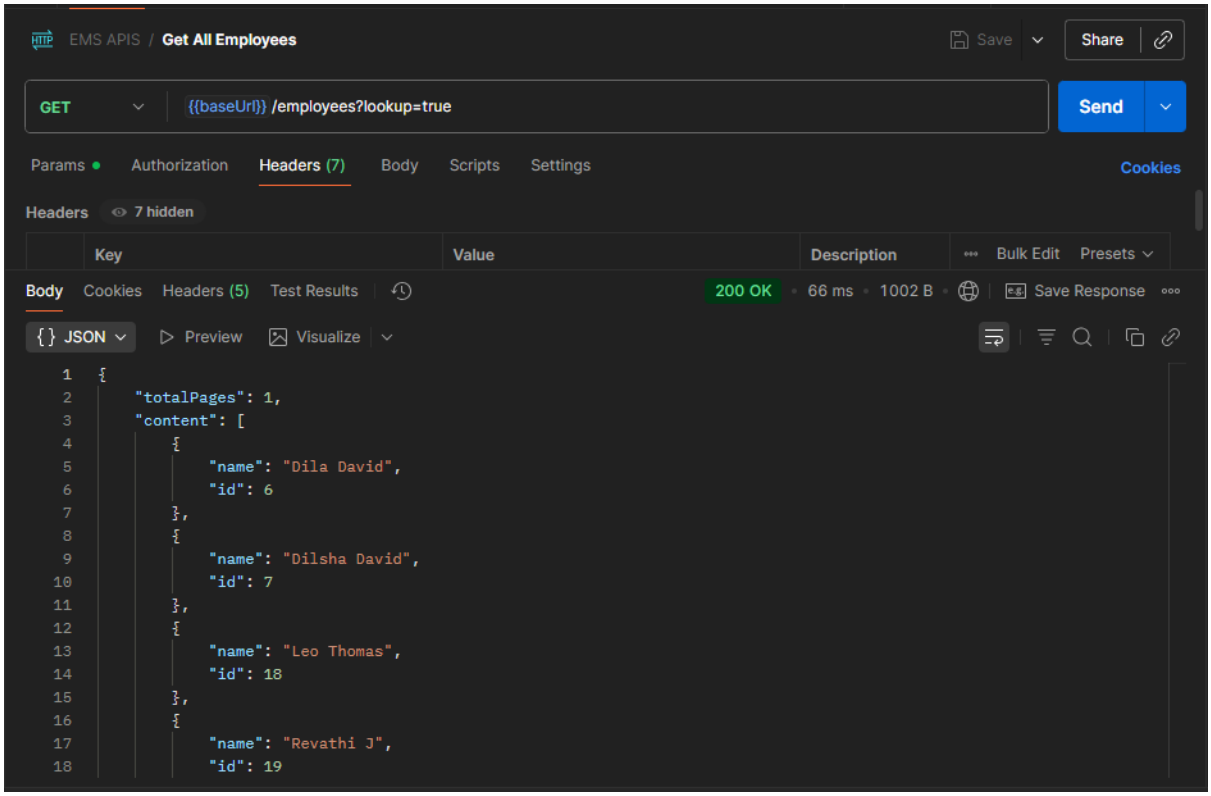

Fig 1.9 get all the employees with id and name

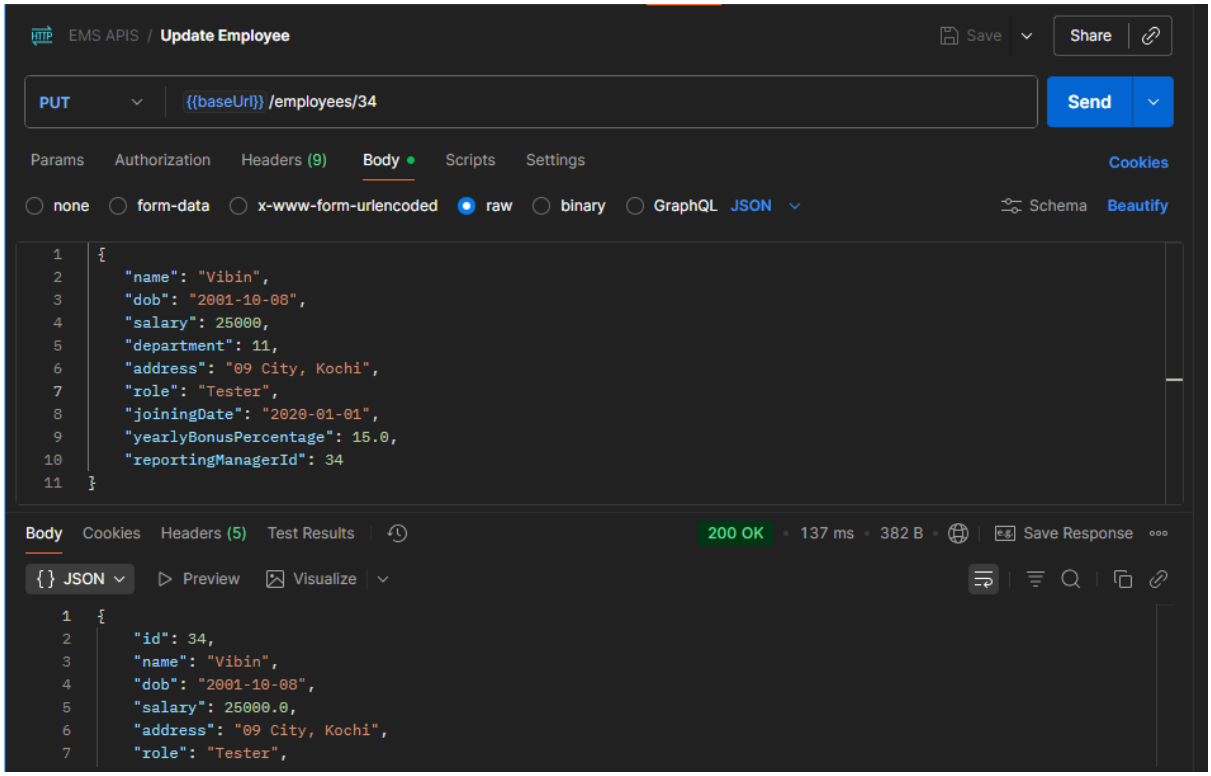

Fig 1.10 updated employee details

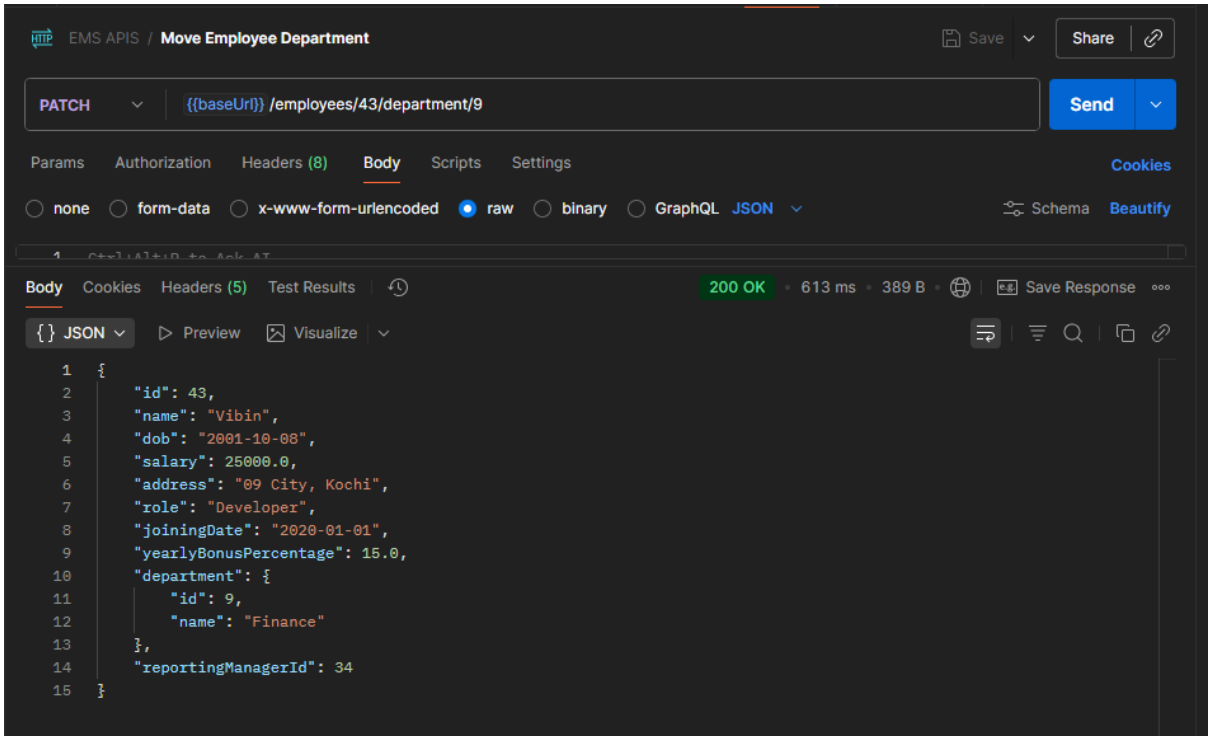

Fig 1.11 Moving one employee to another department

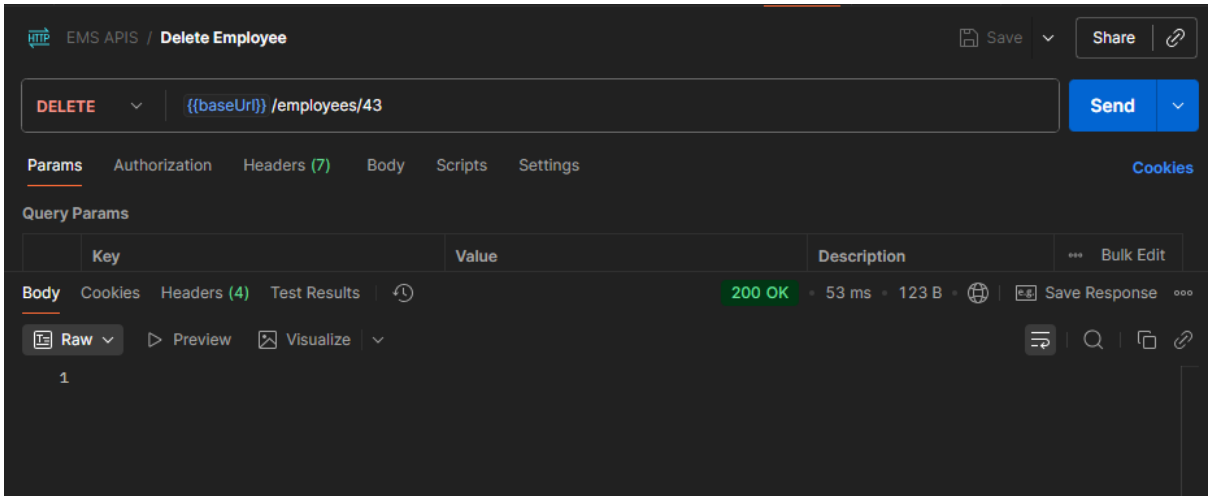

Fig 1.12 Delete an employee

3.4) Testing (Postman)

3.4.1) Test Cases for employee

API	Test Scenario	Expected Result
POST /employees	Create new employee	Status 200, employee data returned
POST /employees	Missing name	Status 400, error message
GET /employees	Get all employees	Status 200, list of employees

GET /employees/{id}	Valid ID	Status 200, employee data
GET /employees/{id}	Invalid ID	Status 404, error message
PUT /employees/{id}	Update existing employee	Status 200, updated data
DELETE /employees/{id}	Delete employee	Status 200, confirmation message
DELETE /employees/{id}	Non-existing ID	Status 404, error message

3.4.2) Test Cases for Department

API	Test Scenario	Request Data / Params	Expected Result
POST /departments	Create new department	{ "name": "IT", "createdDate": "2025-10-24", "departmentHeadId" : 5 }	Status 201, department created
POST /departments	Missing name	{ "createdDate": "2025-10-24" }	Status 400, error "Department name cannot be empty"
POST /departments	Invalid departmentHeadId	{ "name": "IT", "departmentHeadId" : 999 }	Status 400 or 404, error message
GET /departments	Get all departments	—	Status 200, list of departments
GET /departments?expand=true	Get all departments with employees	—	Status 200, list including employees
GET /departments/{id}	Valid department ID	/departments/1	Status 200, department details
GET /departments/{id}	Invalid department ID	/departments/999	Status 404, "Department not found"
PUT /departments/{id}	Update department name	{ "name": "HR", "departmentHeadId" : 3 }	Status 200, updated department returned

PUT /departments/{id}	Update non-existing department	/departments/999	Status 404, error message
DELETE /departments/{id}	Delete existing department	/departments/1	Status 200, "Department deleted successfully "
	Delete non-existing department	/departments/999	Status 404, error message

4.Code Logic

4.1) Loops and Conditional Statements

Example 1: Looping through departments to count employees

```
public Map<String, Long> getEmployeeCountByDepartment() {
  List<Department> departments = departmentRepository.findAll();
  Map<String, Long> result = new HashMap<>();

  for (Department dept : departments) {  // loop
    long count =
employeeRepository.countByDepartmentId(dept.getId());
    result.put(dept.getName(), count);
  }

  return result;
}
```

Explanation:

- **Loop:** for (Department dept : departments) iterates through all departments.
- **Conditional:** Prevents null errors or invalid data:

```
if (employee.getDepartmentId() != null) {
  Department dept =
departmentRepository.findById(employee.getDepartmentId())
      .orElseThrow(() -> new RuntimeException("Department not found"));
  employee.setDepartment(dept);
}
```

Example 2: Conditional logic for expanding employees in a department

```
if ("true".equalsIgnoreCase(expand) || "employee".equalsIgnoreCase(expand)) {
  departments.forEach(dept -> dept.getEmployees().size()); // force initialization
```

```
} else {
  departments.forEach(dept -> dept.setEmployees(null)); // hide employees
}
```

- **Conditionals:** Decide whether to include the employee list or not.
- **Loop (forEach):** Processes each department efficiently.

Example 3: Moving employee to another department

```
public Employee moveEmployeeToDepartment(Long employeeId, Long
departmentId) {
  Employee employee = employeeRepository.findById(employeeId)
      .orElseThrow(() -> new RuntimeException("Employee not found"));

  Department newDepartment = departmentRepository.findById(departmentId)
      .orElseThrow(() -> new RuntimeException("Department not found"));

  employee.setDepartment(newDepartment);
  return employeeRepository.save(employee);
}
```

- **Conditionals:** Checks for valid employee and department IDs.
- Ensures **data integrity** before updating.

4.2) Usage of Collections

Example 1: Transforming employees to a simple lookup list

```
public List<Map<String, Object>> getEmployeeIdAndName() {
  return employeeRepository.findAll()
      .stream()
      .map(emp -> {
        Map<String, Object> map = new HashMap<>();
        map.put("id", emp.getId());
        map.put("name", emp.getName());
        return map;
      })
      .collect(Collectors.toList());
}
```

- **List:** Stores multiple employee entries.
- **Map:** Stores key-value pairs (id and name).
- **Stream & map:** Efficiently converts list of objects into required format.

Example 2: Recursive collection processing for reporting chain

```
public List<Employee> getReportingEmployees(Long managerId) {
  List<Employee> directReports =
employeeRepository.findByReportingManagerId(managerId);
  List<Employee> allReports = new ArrayList<>(directReports);
  for (Employee e : directReports) {
    allReports.addAll(getReportingEmployees(e.getId())); // recursive
```

```
    }
    return allReports;
}
```

- **List:** Stores both direct and indirect reports.
- **Loop + recursion:** Traverses reporting hierarchy.
- **Conditional inside recursion:** Stops recursion when no direct reports are found.

4.3) Appropriate Data Types

Field	Data Type	Purpose / Reason
id	Long	Auto-generated primary key
name	String	Employee/Department name
salary	Double	Handles decimal salary values
departmentId, reportingManagerId	Long	Matches database references
employees	List<Employee>	Stores multiple employees in a department
employeeCountByDepartment	Map<String, Long>	Stores counts of employees by department name
dob, joiningDate	LocalDate	Date fields for employee info
yearlyBonusPercentage	Double	Decimal percentage value

5. Code Flow

Overview

In this project, the code follows a **layered architecture**:

Controller → Service → Repository (Database)

1. **Controller:** Receives HTTP requests (GET, POST, PUT, DELETE) and handles request parameters and responses.
2. **Service:** Contains the business logic, data validation, and processing.
3. **Repository / Database:** Interacts with the database to fetch, save, update, or delete data using JPA methods.

Example 1: Creating an Employee

Step 1: Controller receives the request

```
@PostMapping
public Employee create(@RequestBody Employee employee) {
  return employeeService.addEmployee(employee);
```

}

- The **EmployeeController** receives a POST request to /employees with JSON body containing employee details.
- It forwards the data to **EmployeeService**.

Step 2: Service processes the request

```
public Employee addEmployee(Employee employee) {

  if (employee.getDepartmentId() != null) {
    Department dept =
departmentRepository.findById(employee.getDepartmentId())
        .orElseThrow(() -> new RuntimeException("Department not found"));
    employee.setDepartment(dept);
  }

  if (employee.getReportingManagerId() != null) {
    Employee manager =
employeeRepository.findById(employee.getReportingManagerId())
        .orElseThrow(() -> new RuntimeException("Reporting Manager not
found"));
    employee.setReportingManager(manager);
  }

  return employeeRepository.save(employee);
}
```

- The **service layer** validates department and manager IDs.
- It sets the associations and then calls **employeeRepository.save()** to persist the employee in the database.

Step 3: Repository interacts with the database

```
Employee savedEmployee = employeeRepository.save(employee);
```

- Spring Data JPA handles the database operation.
- Data is inserted into the **employees** table, with foreign keys for department and reporting manager.
- The saved entity is returned to the service, and then to the controller as the response.

Example 2: Get Employee List with Pagination

Controller:

```
@GetMapping
public Map<String, Object> getAll(
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "20") int size) {
```

```
    Page<Employee> employeePage =
employeeService.getPaginatedEmployees(page, size);
    return Map.of(
        "content", employeePage.getContent(),
        "page", employeePage.getNumber(),
        "totalPages", employeePage.getTotalPages(),
        "totalItems", employeePage.getTotalElements()
    );
}
```

Service:

```
public Page<Employee> getPaginatedEmployees(int page, int size) {
    Pageable pageable = PageRequest.of(page, size);
    return employeeRepository.findAll(pageable);
}
```

Repository:

```
Page<Employee> employeePage = employeeRepository.findAll(pageable);
```

- **Controller** receives the GET request with pagination parameters.
- **Service** prepares the PageRequest object and calls the repository.
- **Repository** fetches paginated results from the database.
- **Controller** returns the results as JSON.

Example 3: Moving Employee to Another Department

Controller:

```
@PatchMapping("/{employeeId}/department/{departmentId}")
public Employee moveEmployeeToDepartment(@PathVariable Long employeeId,
@PathVariable Long departmentId) {
    return employeeService.moveEmployeeToDepartment(employeeId,
departmentId);
}
```

Service:

```
public Employee moveEmployeeToDepartment(Long employeeId, Long
departmentId) {
    Employee employee = employeeRepository.findById(employeeId)
        .orElseThrow(() -> new RuntimeException("Employee not found"));
    Department newDepartment = departmentRepository.findById(departmentId)
        .orElseThrow(() -> new RuntimeException("Department not found"));
    employee.setDepartment(newDepartment);
    return employeeRepository.save(employee);
}
```

Flow Explanation:

1. Controller receives request → /employees/{id}/department/{deptId}
2. Service validates employee and department existence
3. Repository updates the employee record in the database
4. Response is returned to the client

6.JPA Implementation

Overview

The project uses **JPA (Java Persistence API)** to handle database operations in a clean and object-oriented way.

- **Entities:** Represent database tables as Java classes.
- **Repositories:** Provide methods to perform CRUD operations without writing SQL.
- **Service Layer:** Uses repositories to interact with the database.
- **Spring Data JPA:** Automatically implements common methods like findAll(), save(), deleteById().

Employee Entity Example

```
@Entity
@Table(name = "employees")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private Double salary;
    private String role;
    private String address;

    private LocalDate dob;
    private LocalDate joiningDate;
    private Double yearlyBonusPercentage;

    @ManyToOne
    @JoinColumn(name = "department_id")
    private Department department;

    @ManyToOne
    @JoinColumn(name = "reporting_manager_id")
    private Employee reportingManager;

    // Getters and Setters
```

```
}
```

Explanation:

- @Entity: Marks this class as a JPA entity (database table).
- @Table(name = "employees"): Specifies the table name.
- @Id and @GeneratedValue: Primary key with auto-increment.
- @ManyToOne / @JoinColumn: Defines relationships to **Department** and **Reporting Manager**.

Department Entity Example

```
@Entity
@Table(name = "departments")
public class Department {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String name;

  @OneToOne
  @JoinColumn(name = "department_head_id")
  private Employee departmentHead;

  @OneToMany(mappedBy = "department")
  private List<Employee> employees;

  // Getters and Setters
}
```

Explanation:

- @OneToOne → Represents the **department head** relationship.
- @OneToMany(mappedBy = "department") → Maps all employees in this department.
- JPA automatically handles **foreign key relationships**.

Repository Layer

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
  List<Employee> findByReportingManagerId(Long managerId);
  long countByDepartmentId(Long departmentId);
}

public interface DepartmentRepository extends JpaRepository<Department,
Long> {
```

```
}
```

Explanation:

- **JpaRepository** provides standard CRUD methods (save, findById, findAll, deleteById).
- Custom methods like findByReportingManagerId and countByDepartmentId are automatically implemented by Spring Data JPA.

Service Layer Using JPA

Example: Adding a new employee:

```
public Employee addEmployee(Employee employee) {

  if (employee.getDepartmentId() != null) {
    Department dept =
departmentRepository.findById(employee.getDepartmentId())
        .orElseThrow(() -> new RuntimeException("Department not found"));
    employee.setDepartment(dept);
  }

  if (employee.getReportingManagerId() != null) {
    Employee manager =
employeeRepository.findById(employee.getReportingManagerId())
        .orElseThrow(() -> new RuntimeException("Reporting Manager not
found"));
    employee.setReportingManager(manager);
  }

  return employeeRepository.save(employee); // Persist in DB
}
```

- JPA handles **SQL generation** and **data persistence**.
- The service layer uses **repository methods** instead of manual SQL queries.

7.Database Script (DB Script)

8.1) Database: employee_db

```
-- Create Department table
CREATE TABLE departments (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  department_head_id BIGINT,
  CONSTRAINT fk_department_head
    FOREIGN KEY (department_head_id)
```

```
    REFERENCES employees(id)
    ON DELETE SET NULL
);

-- Create Employee table
CREATE TABLE employees (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  salary DOUBLE,
  role VARCHAR(100),
  address VARCHAR(255),
  dob DATE,
  joining_date DATE,
  yearly_bonus_percentage DOUBLE,
  department_id BIGINT,
  reporting_manager_id BIGINT,
  CONSTRAINT fk_department
    FOREIGN KEY (department_id)
    REFERENCES departments(id)
    ON DELETE SET NULL,
  CONSTRAINT fk_reporting_manager
    FOREIGN KEY (reporting_manager_id)
    REFERENCES employees(id)
    ON DELETE SET NULL
);
```

Explanation

1. **Auto Increment IDs**:
    - BIGINT AUTO_INCREMENT is used for id fields for unique primary keys.
2. **Foreign Keys**:
    - department_id in employees → references departments(id)
    - reporting_manager_id in employees → self-referencing FK to employees(id)
    - department_head_id in departments → references employees(id)
3. **ON DELETE SET NULL**:
    - Ensures that deleting a referenced employee or department does **not delete dependent records**, but sets the FK to NULL.