**Exercise 6: Employee Management System - Implementing Pagination and Sorting**

Business Scenario:

Add pagination and sorting capabilities to your employee search functionality.

**1. Introduction**

The Employee Management System (EMS) is designed to manage employee and department data effectively. This documentation covers the implementation of key features such as pagination and sorting within the system.

**2. System Overview**

The system consists of several components:

* **Entities**: Department and Employee
* **Repositories**: For data access and CRUD operations
* **Controllers**: To handle HTTP requests and responses
* **Configuration**: Spring Boot application setup and data source configuration

**3. Entities**

**3.1 Department**

* **Attributes**:
  + id: Unique identifier for the department (Long)
  + name: Name of the department (String)
* **Relationships**:
  + One-to-Many relationship with Employee (employees)

**3.2 Employee**

* **Attributes**:
  + id: Unique identifier for the employee (Long)
  + name: Name of the employee (String)
  + email: Email address of the employee (String)
* **Relationships**:
  + Many-to-One relationship with Department (department)

**4. Repositories**

**4.1 DepartmentRepository**

* **Purpose**: Provides CRUD operations for Department entities.
* **Methods**: Inherits from JpaRepository, allowing standard database operations.

**4.2 EmployeeRepository**

* **Purpose**: Provides CRUD operations for Employee entities.
* **Methods**:
  + findByName(String name): Finds employees by their name.
  + findByEmail(String email): Finds an employee by their email using JPQL.

**5. Controllers**

**5.1 DepartmentController**

* **Endpoints**:
  + GET /departments: Retrieves all departments.
  + POST /departments: Creates a new department.
  + GET /departments/{id}: Retrieves a department by ID.
  + PUT /departments/{id}: Updates an existing department by ID.
  + DELETE /departments/{id}: Deletes a department by ID.

**5.2 EmployeeController**

* **Endpoints**:
  + GET /employees: Retrieves employees with optional pagination and sorting.
  + GET /employees/sorted: Retrieves employees with sorting options.

**6. Pagination and Sorting**

**6.1 Pagination**

* **Implementation**: Handled in the EmployeeController using the Pageable interface.
* **Endpoint**: GET /employees uses Pageable parameters to fetch a paginated list of employees.

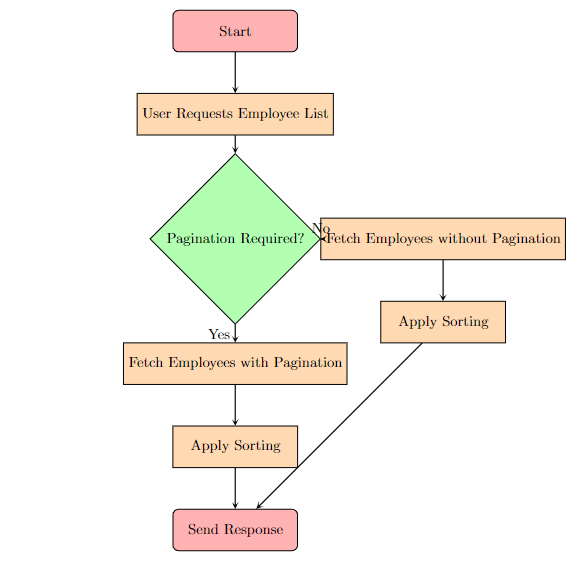
**6.2 Sorting**

* **Implementation**: Sorting is applied in the EmployeeController using the Sort class.
* **Endpoint**: GET /employees/sorted allows sorting employees based on a specified field.

**7. Application Configuration**

* **Database**: Uses H2 in-memory database for development and testing.
* **Properties**:
  + spring.datasource.url: Database URL
  + spring.datasource.driver-class-name: Database driver class name
  + spring.datasource.username: Database username
  + spring.datasource.password: Database password
  + spring.jpa.database-platform: Hibernate dialect for H2

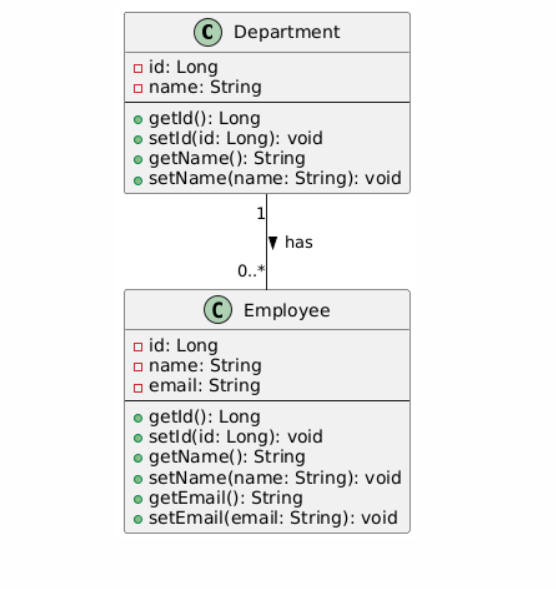
**FLOWCHART :**



**Explanation**

1. **Start**: The entry point of the flowchart.
2. **User Requests Employee List**: Represents the action where the user requests a list of employees.
3. **Pagination Required?**: Decision point to determine if pagination is required based on user request parameters.
   * **Yes**: If pagination is needed, fetch employees with pagination.
   * **No**: If pagination is not needed, fetch employees without pagination.
4. **Fetch Employees with Pagination**: Process to retrieve employees with pagination parameters.
5. **Fetch Employees without Pagination**: Process to retrieve employees without pagination.
6. **Apply Sorting**: Process to sort the employees based on the requested sort criteria.
7. **Send Response**: Final step to send the sorted and paginated list (or just sorted list) of employees back to the user.

**CLASS DIAGRAM :**



**Explanation**

1. **Department Class**:
   * **Attributes**:
     + id: Unique identifier for the department.
     + name: Name of the department.
   * **Methods**:
     + getId(), setId(id: Long): Accessor and mutator for id.
     + getName(), setName(name: String): Accessor and mutator for name.
2. **Employee Class**:
   * **Attributes**:
     + id: Unique identifier for the employee.
     + name: Name of the employee.
     + email: Email address of the employee.
   * **Methods**:
     + getId(), setId(id: Long): Accessor and mutator for id.
     + getName(), setName(name: String): Accessor and mutator for name.
     + getEmail(), setEmail(email: String): Accessor and mutator for email.
3. **Relationship**:
   * **Department "1" -- "0..\*" Employee**: Indicates a one-to-many relationship where one department can have multiple employees. The has label represents this relationship.