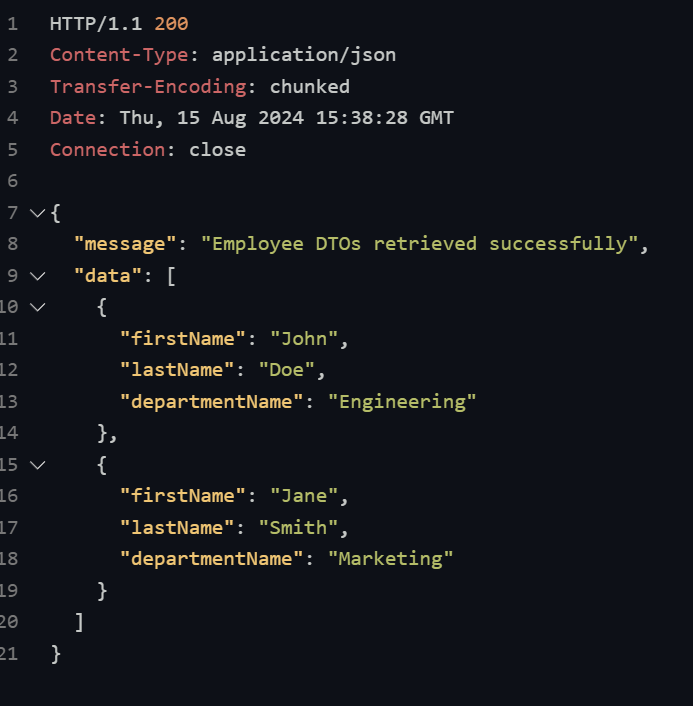
**Exercise 8: Employee Management System - Creating Projections**

Create projections to fetch specific data subsets from the employee and department entities.

**Projections:**

1. ***Interface-Based Projections:***
   * ***Definition***: Interface-based projections allow you to define an interface with getter methods for the properties you want to project from the entity. Spring Data will generate an implementation at runtime that retrieves only the specified fields.
   * ***Usage***: These projections are used when you only want to retrieve specific fields of the entity rather than the entire entity, improving performance by reducing data transfer.
2. ***Class-Based Projections:***
   * ***Definition***: In class-based projections, a class is created with a constructor that takes in the specific fields you want to project. You then use Spring's @Query annotation to map the selected fields to the constructor.
   * ***Usage***: Useful when you need more control over the retrieved data, such as performing data transformations or validations in the constructor.
3. ***@Value and Constructor Expressions****:*
   * **@*Value***: This annotation allows you to inject specific fields from the entity into your projection class. It can also be used with constructor expressions to extract custom subsets of data.
   * ***Constructor* *Expressions***: These are used in JPQL queries to create new objects directly from the query result. For example, you can create a DTO from the selected fields without needing to map them afterward.

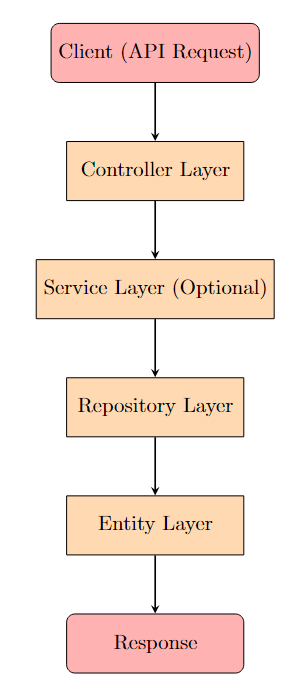
**The output of the program:**

**Send Requests:** Use REST client tools like Postman or VS Code's REST Client to send HTTP requests (*GET, POST, PUT, DELETE*) to the endpoints defined in your *EmployeeController* and *DepartmentController*.

**Receive Responses:** The responses will include data subsets based on the projections, such as specific fields from Employee and Department entities.

**View Data:** Check the response body to see the results of your queries, including any filtered or projected data.

**Verify Accuracy:** Ensure that the data matches the requested projections and check for any discrepancies or errors in the data returned.

* **Flowchart of the program:**

The flowchart below outlines the general flow of the program:

1. **Controller Layer:**

* Receives API requests for employee and department operations.
* Calls the appropriate methods in the repository layer.

1. **Service Layer (Optional):**

* Handles business logic (if any), like manipulating or transforming the data.
* Calls the repository layer to fetch data using projections.

1. **Repository Layer:**

* Fetches data using projections (either interface-based or class-based).
* Defines JPQL queries or methods using @Query and constructor expressions to retrieve custom data subsets.

1. **Entity Layer:**

* Maps to the database tables (Employee, Department).
* Defines relationships and fields that can be used for projections.

1. **Response Layer:**

* Returns the data (fetched through projections) back to the client in the requested format (e.g., JSON).