Spring Data JPA with Hibernate Part 1

(1) Create an Employee Entity which contains following fields

Name

Id

Age

Location

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "employee")
public class Employee {
    @Id
    @GeneratedValue
    private int id;
    private String name;
    private int age;
    private String location;

}
```

(2) Set up EmployeeRepository with Spring Data JPA

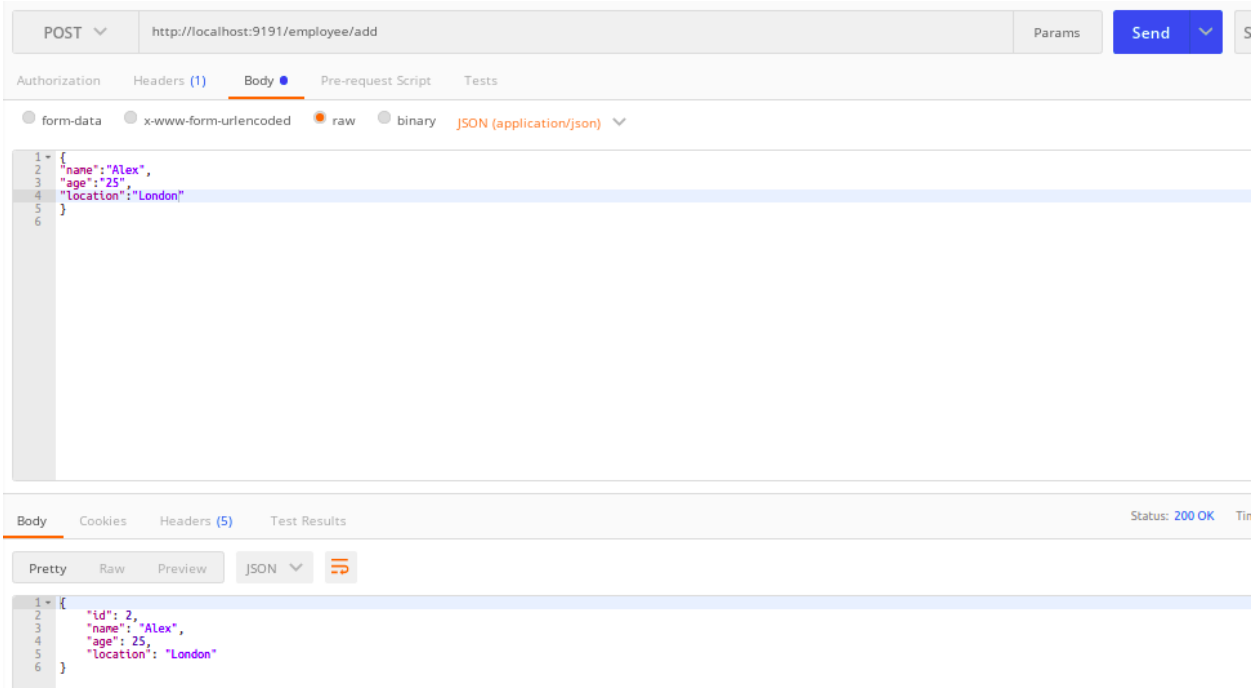
```java
package com.ttn.bootcamp.springDataJPA.employee.respository;

import com.ttn.bootcamp.springDataJPA.employee.entities.Employee;
import org.springframework.data.repository.CrudRepository;

public interface EmployeeRepository extends CrudRepository<Employee, Integer> {


}
```

(3) Perform Create Operation on Entity using Spring Data JPA

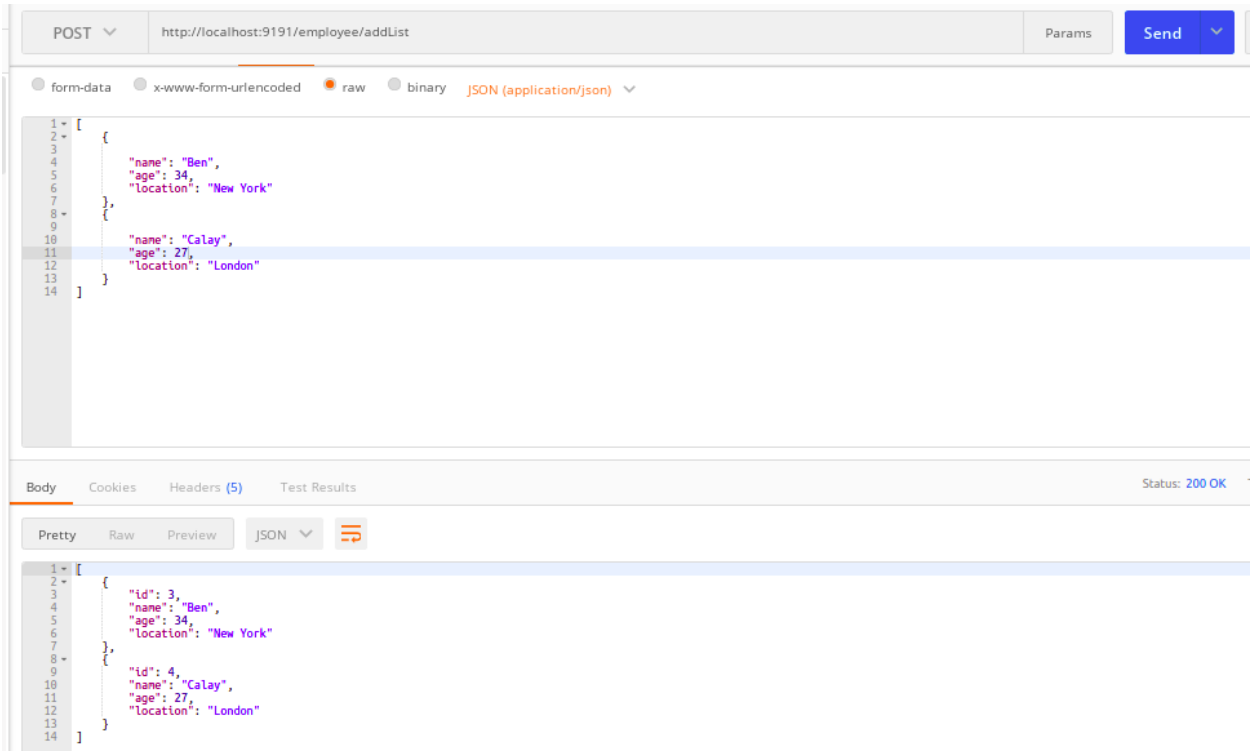
```java
@Override
public Employee save(Employee employee) {
    return employeeRepository.save(employee);
}

@Override
public List<Employee> saveList(List<Employee> employees) {
    return (List<Employee>) employeeRepository.saveAll(employees);
}
```

```java
@PostMapping(path = "/add")
public Employee add(@RequestBody Employee employee) {
    return employeeService.save(employee);

}

@PostMapping(path = "/addList")
public List<Employee> addList(@RequestBody List<Employee> employees) {
    return employeeService.saveList(employees);

}
```

Added one employee details



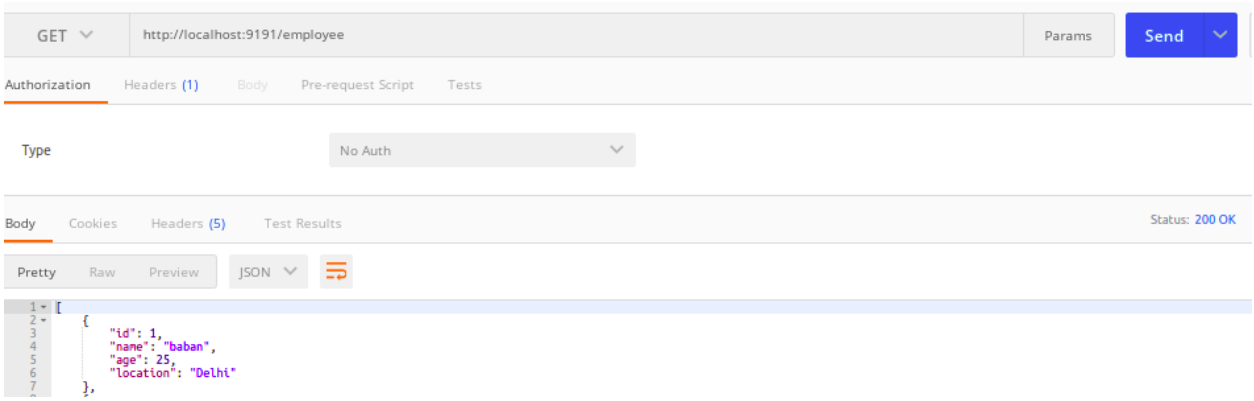
Added list of employees details



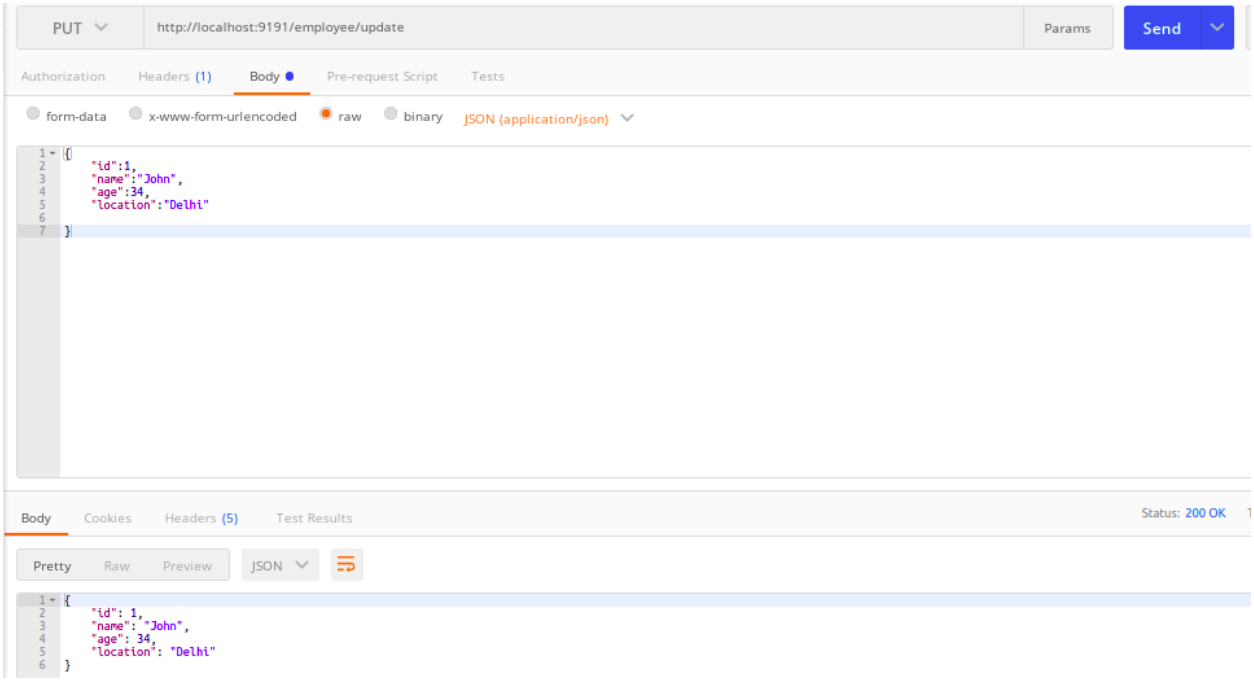
(4) Perform Update Operation on Entity using Spring Data JPA

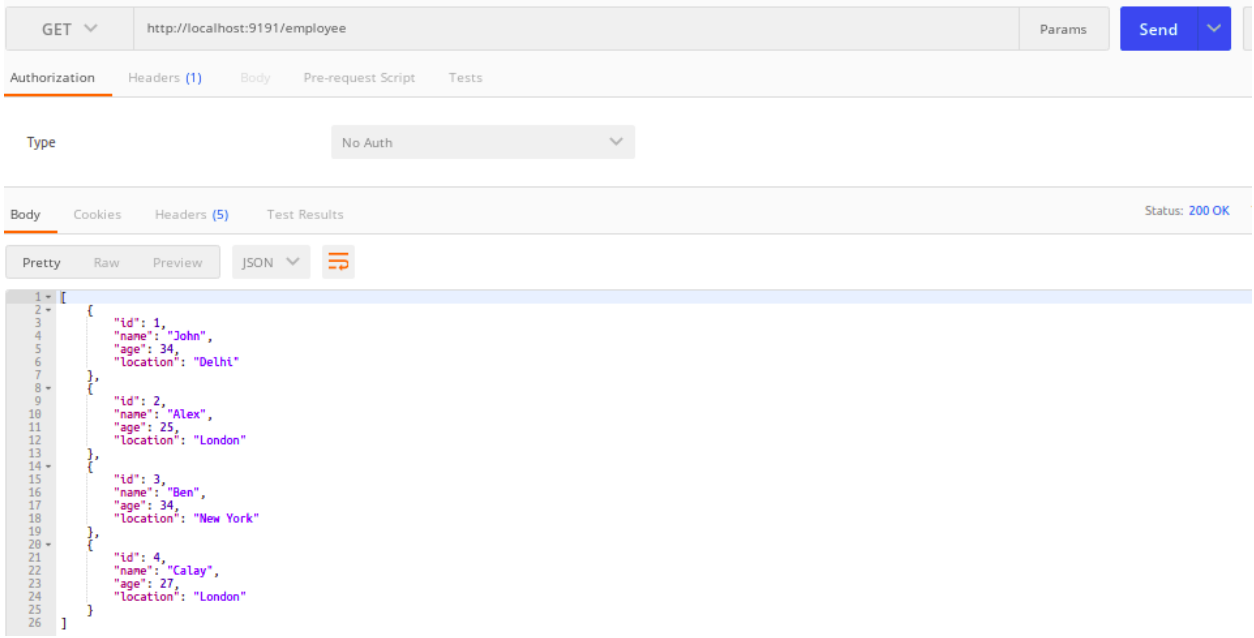
```java
@Override
public Employee update(Employee employee) {
    Employee existingEmployee = employeeRepository.findById(employee.getId()).orElse( other: null);
    existingEmployee.setName(employee.getName());
    existingEmployee.setAge(employee.getAge());
    existingEmployee.setLocation(employee.getLocation());
    return employeeRepository.save(existingEmployee);
}
```

Updating employee with id 1



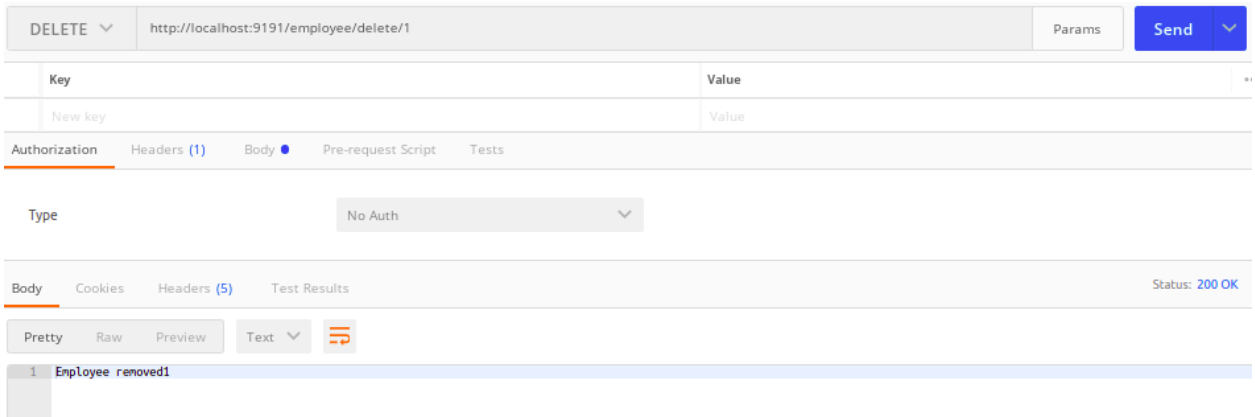
Updated by put request





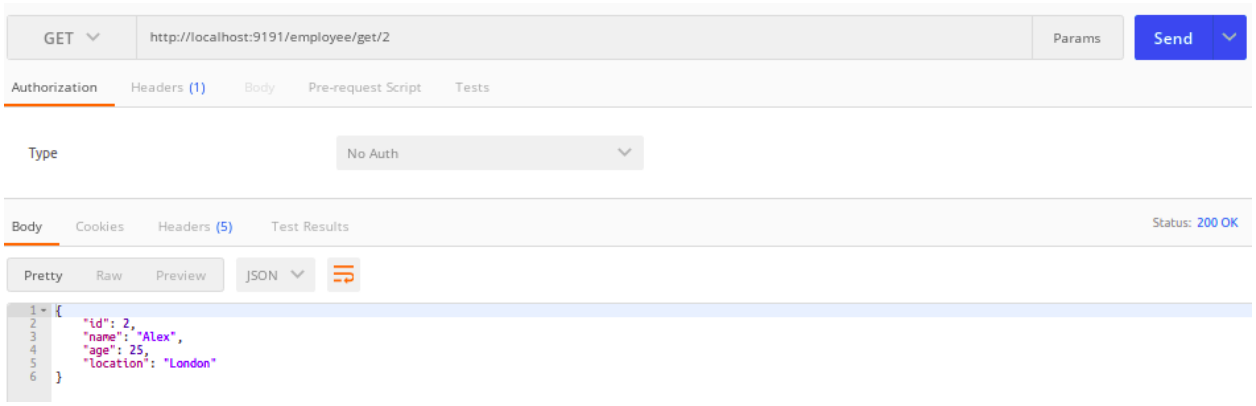
(5) Perform Delete Operation on Entity using Spring Data JPA

```java
@Override
public String delete(Integer id) {
    employeeRepository.deleteById(id);
    return "Employee removed" + id;
}
```



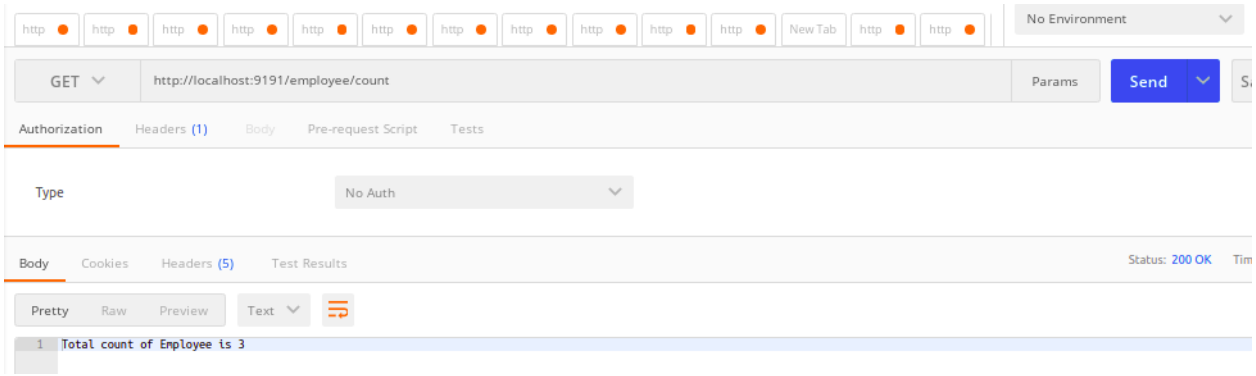
(5) Perform Read Operation on Entity using Spring Data JPA

```java
@Override
public List<Employee> findAll() {
    return (List<Employee>) employeeRepository.findAll();
}

@Override
public Optional<Employee> findById(Integer id) {
    return employeeRepository.findById(id);

}
```



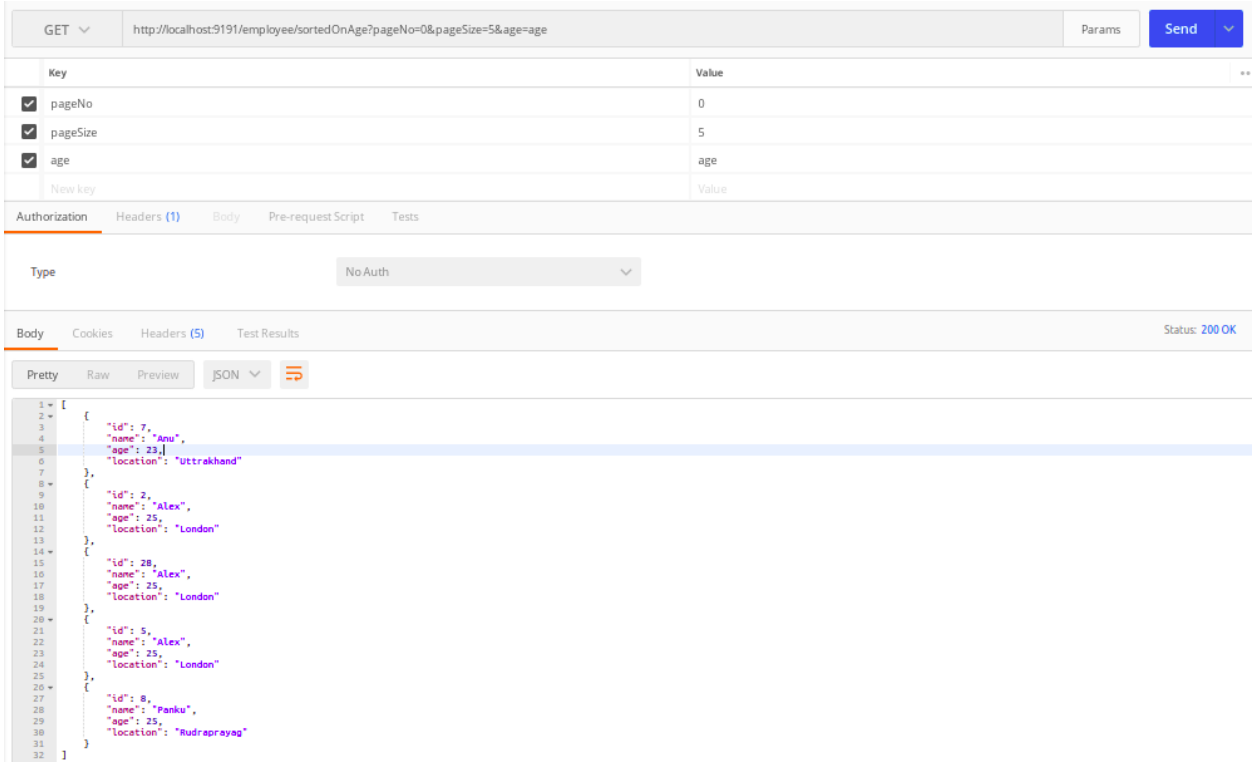
(6) Get the total count of the number of Employees

```java
@Override
public String count() {
    int count= (int) employeeRepository.count();
     return "Total count of Employee is "+count;
}
```



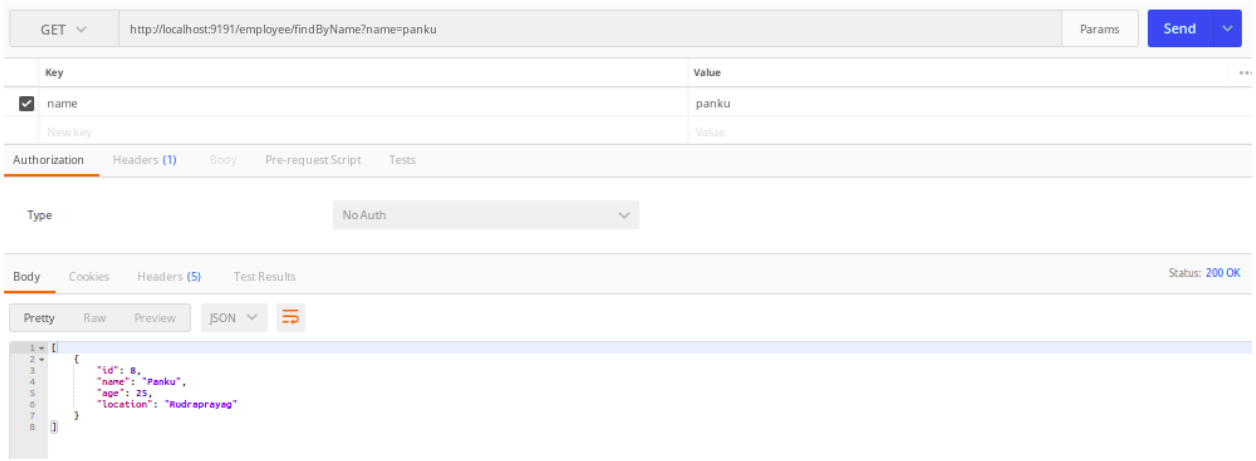
(7) Implement Pagination and Sorting on the bases of Employee Age

```
public List<Employee> getPageSortedOnAge(Integer pageNo, Integer pageSize, String sortBy) {
    Pageable paging = PageRequest.of(pageNo, pageSize, Sort.by(Sort.Order.asc(sortBy)));
    Page<Employee> pagedResult = employeePageRepository.findAll(paging);

    if (pagedResult.hasContent()) {
        return pagedResult.getContent();
    } else {
        return new ArrayList<Employee>();
    }
}
```



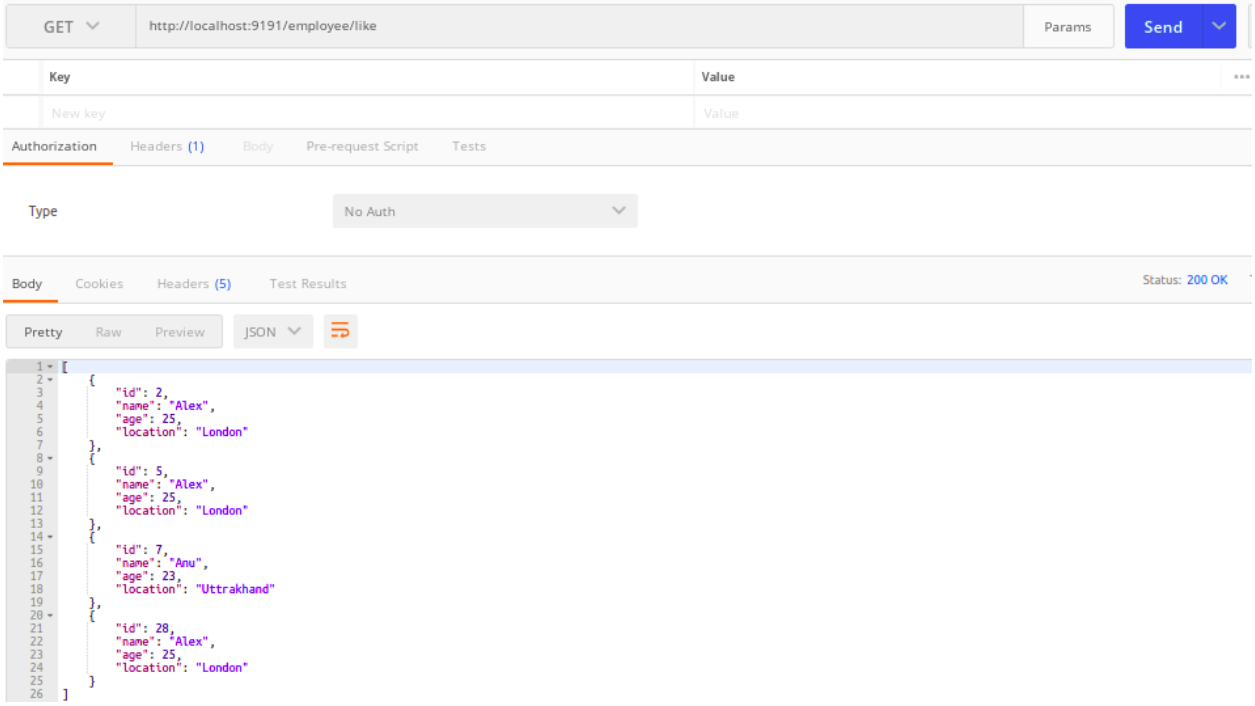
8 Create and use finder to find Employee by Name

```
public List<Employee> findByName(String name) {
    return employeeRepository.findByName(name);
}
```



(9) Create and use finder to find Employees starting with A character

```
public List<Employee> findByNameLikeEmployee() {
    return employeeRepository.findByNameLike("A%");
}
```



(10) Create and use finder to find Employees Between the age of 28 to 32

```
public List<Employee> findByAgeBetweenEmployee(int age1, int age2) {
    return employeeRepository.findByAgeBetween(age1, age2);
}
```