SPRING DATA JPA

Daniel Evaristo Escalera Bonilla

Introduction

Spring data builds on top of the JPA to simplify data persistence and access in Java applications by abstraction of certain processes such as the repository CRUD operations which are generated by creating an object that extends a crud or jpa repository which already has them.

Implementation

To show the implementation of Spring Data JPA, the following entity class was created using JPA and lombok:

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name="timecard")
public class Timecard {

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

    @Column(name="name")
    private String name;

    @Column(name="department")
    private String department;

    @Column(name="entryTime")
    private Date entryTime;

    @Column(name="exitTime")
    private Date exitTime;

    @Column(name="lunchTime")
    private Date lunchTime;

}
```

Next a Repository class was created, this class extends the jpaRepository interface, giving it the main CRUD operations:

```java
package com.curso.v0.CrudJPA.dao;

import java.util.List;

public interface TimecardRepository extends JpaRepository<Timecard, Long>{

    List<Timecard> findByDepartment(String department);

}
```

A service interface and implementation were created to use the repository operations during execution:

```java
import java.util.List;

public interface TimecardService {

    List<Timecard> findAll();

    List<Timecard> findByDepartment(String department);

    Timecard findById(long theId);

    Timecard save(Timecard theTimecard);

    void deleteById(long theId);
}
```

```java
@Service
public class TimecardServiceImpl implements TimecardService{

    private TimecardRepository timecardRepository;

    @Autowired
    public TimecardServiceImpl(TimecardRepository timecardRepository) {
        this.timecardRepository = timecardRepository;
    }

    @Override
    public List<Timecard> findAll() {
        return timecardRepository.findAll();
    }

    @Override
    public Timecard findById(long theId) {
        Optional<Timecard> optionalTimecard = timecardRepository.findById(theId);

        Timecard timecard = null;

        if (optionalTimecard.isPresent()) {
           timecard = optionalTimecard.get();
        }
        else {
           throw new RuntimeException("Did not find employee id - " + theId);
        }

       return timecard;
    }
    @Override
    public List<Timecard> findByDepartment(String department) {
        return timecardRepository.findByDepartment(department);
    }

    @Override
    public Timecard save(Timecard theTimecard) {
        return timecardRepository.save(theTimecard);
    }

    @Override
    public void deleteById(long theId) {
        timecardRepository.deleteById(theId);
    }
```

Finally, a rest controller was created to define the diferent routes of the CRUD API and the body and parameters necesary to apply them:

```java
@RestController
@RequestMapping("/timecard-system")
public class TimecardRestController {

    private TimecardService timecardService;

    @Autowired
    public TimecardRestController(TimecardService timecardService) {
        this.timecardService = timecardService;
    }

    @GetMapping("timecards")
    public List<Timecard> getTimecards(){
        return timecardService.findAll();
    }

    @GetMapping("timecards/{id}")
    public Timecard TimecardById(@PathVariable long id){

        // Find the timecard by ID
        Timecard theTimecard = timecardService.findById(id);

        if (theTimecard == null) {
            throw new RuntimeException("Timecard id not found - " + id);
        }

        return theTimecard;
    }
```

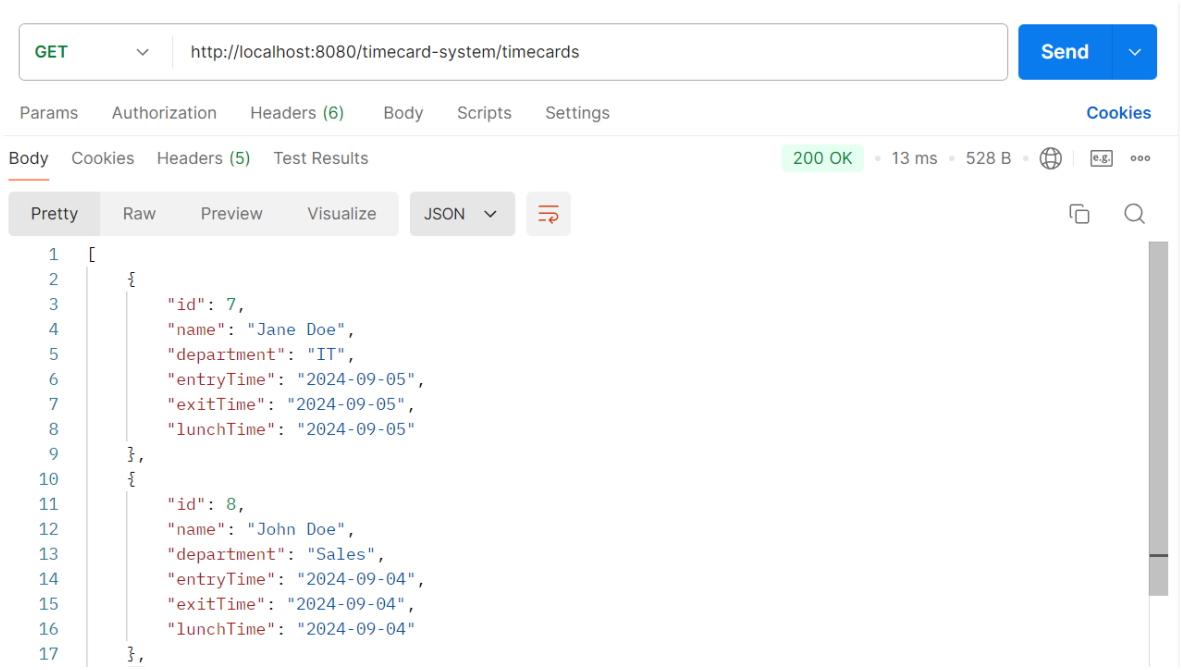
```java
@GetMapping("timecards/department/{department}")
public List<Timecard> getTimecard(@PathVariable String department) {
    return timecardService.findByDepartment(department);
}

@PostMapping("timecards")
public Timecard addTimecard(@RequestBody Timecard theTimecard) {

    theTimecard.setId(0);
    Timecard dbTimecard = timecardService.save(theTimecard);
    return dbTimecard;
}

@PutMapping("timecards")
public Timecard updateTimecard(@RequestBody Timecard theTimecard) {
    Timecard dbTimecard = timecardService.save(theTimecard);
    return dbTimecard;
}

@DeleteMapping("timecards/{id}")
public ResponseEntity<String> delete(@PathVariable Long id) {
    // Find the timecard by ID
    Timecard theTimecard = timecardService.findById(id);

    if (theTimecard == null) {
        throw new RuntimeException("Timecard id not found - " + id);
    }

    timecardService.deleteById(id);

    return ResponseEntity.ok("Timecard deleted successfully, id: " + id);
}
```

Results

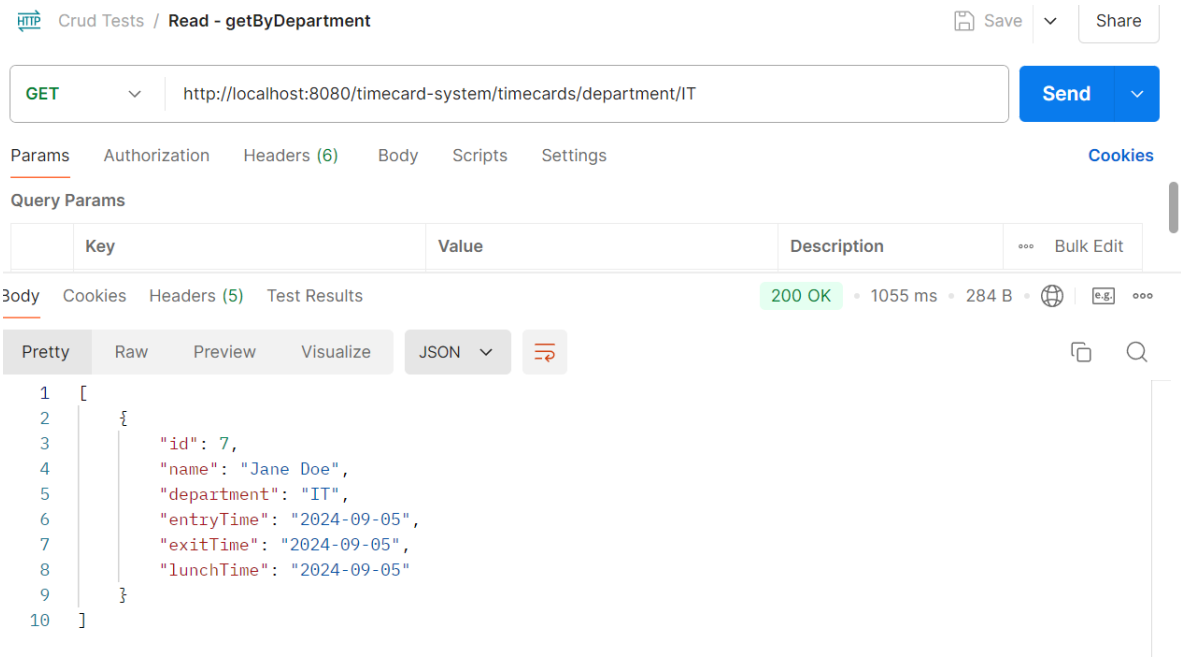
When the application is running we can use the specified URL's to perform operations in the database:

Read

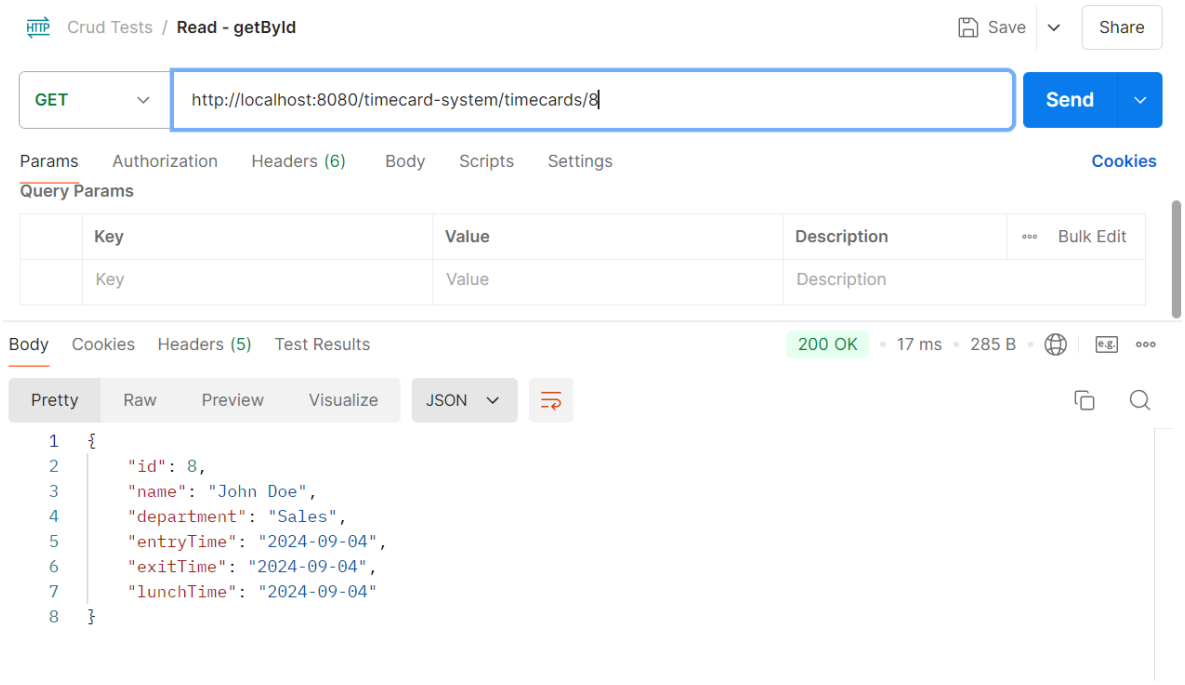
findAll



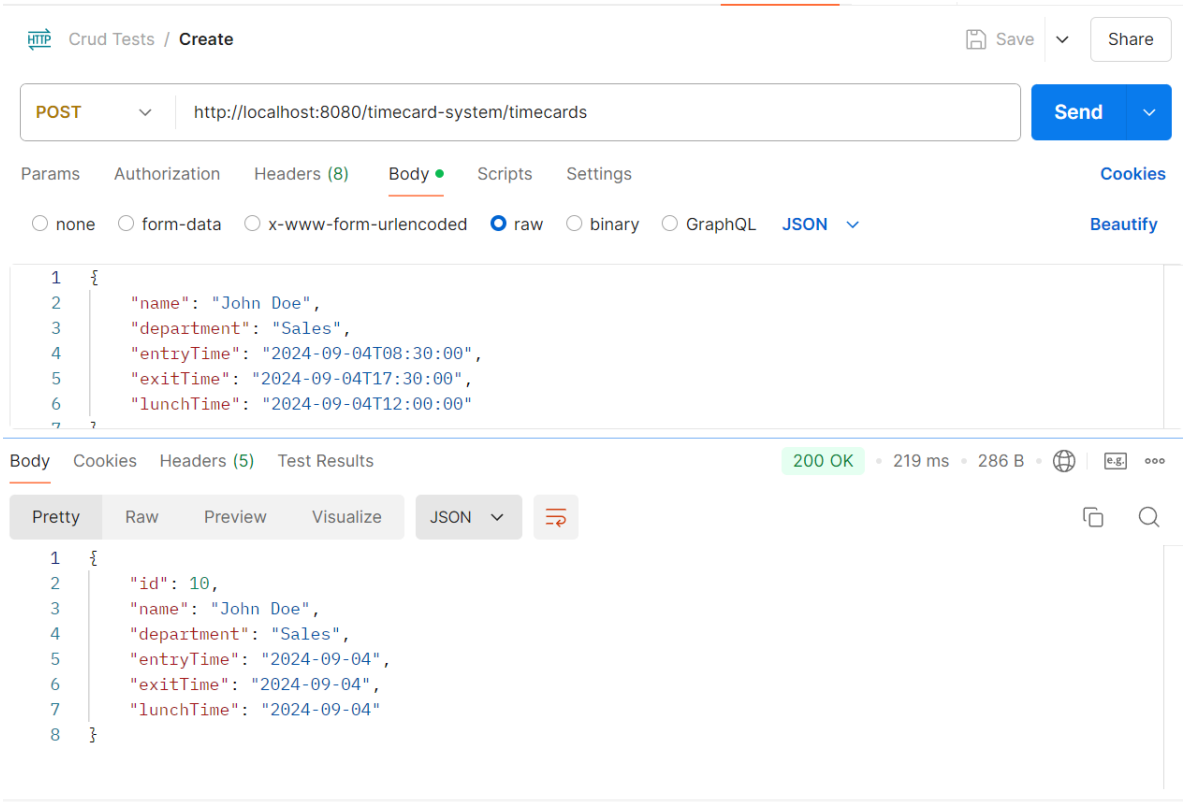
getByDepartment



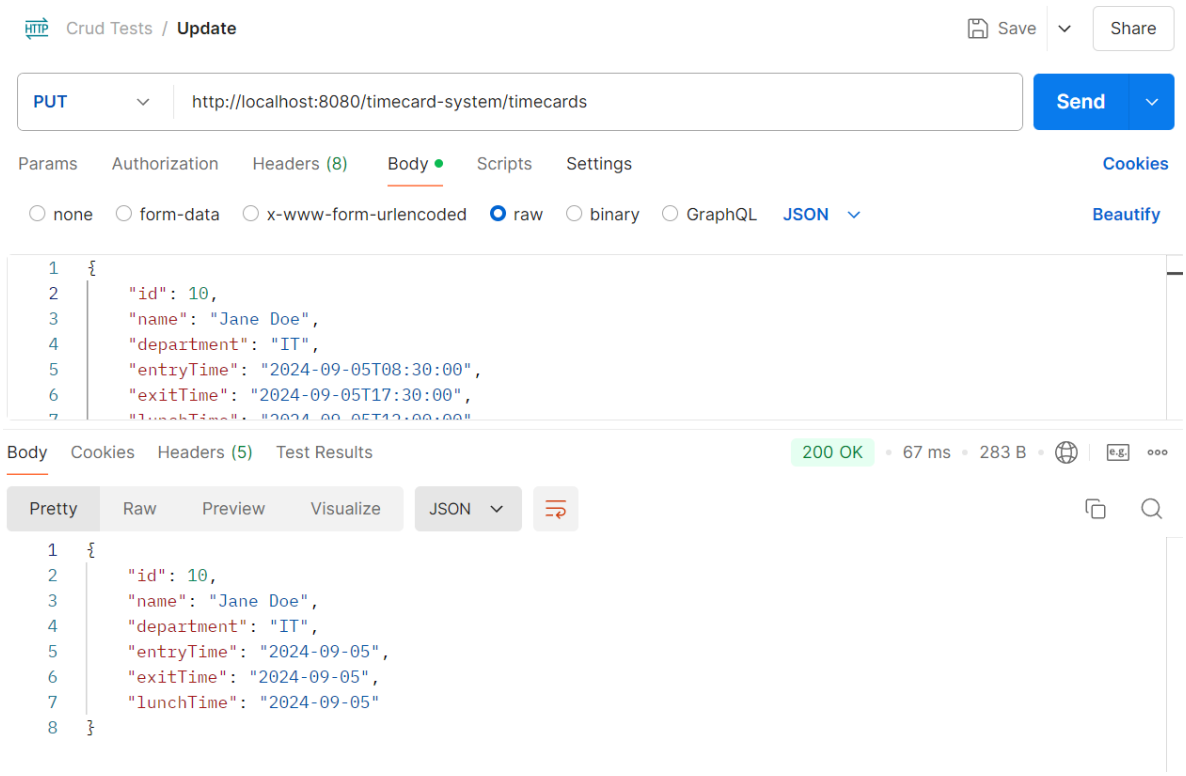
getById



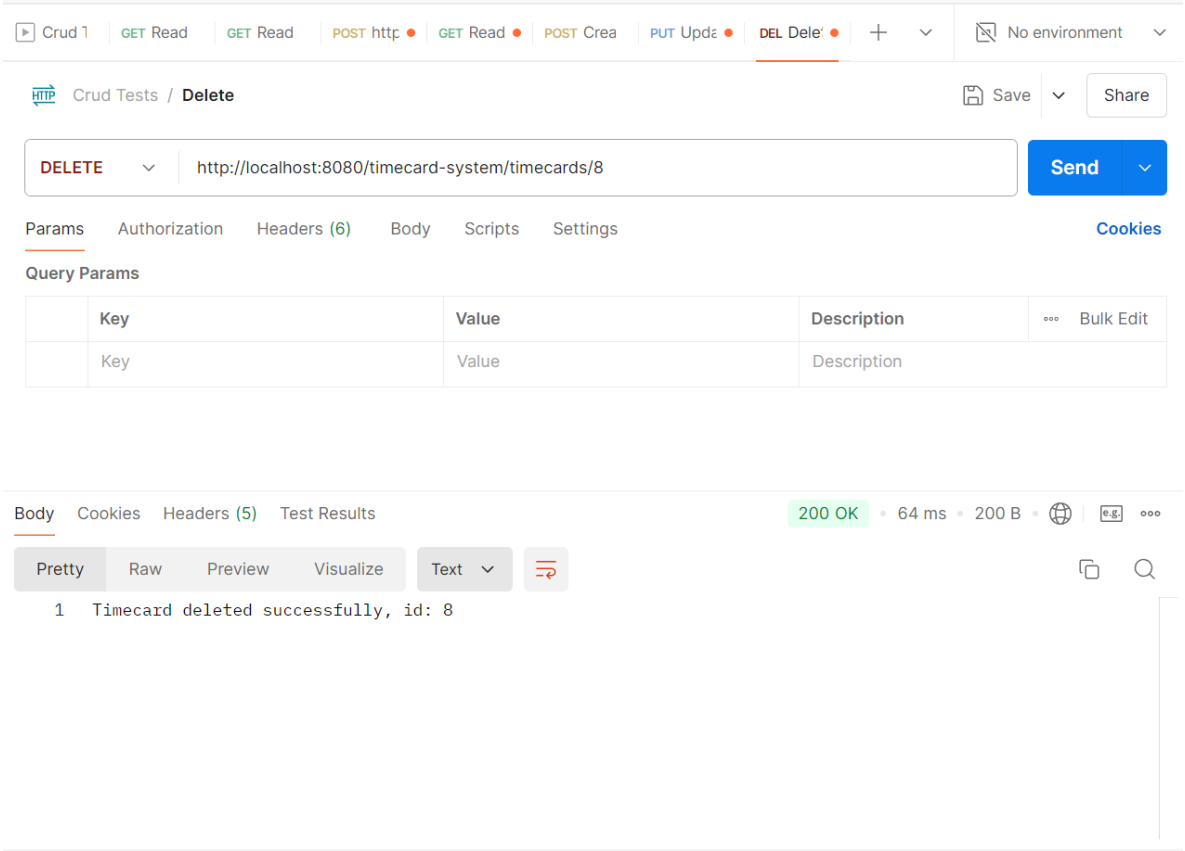
Create



Update


Crud Tests / Update
Save
Share
PUT
http://localhost:8080/timecard-system/timecards
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id": 10,
"name": "Jane Doe",
"department": "IT",
"entryTime": "2024-09-05T08:30:00",
"exitTime": "2024-09-05T17:30:00",
Body
Cookies
Headers (5)
Test Results
200 OK
67 ms
283 B
Pretty
Raw
Preview
Visualize
JSON
{
"id": 10,
"name": "Jane Doe",
"department": "IT",
"entryTime": "2024-09-05",
"exitTime": "2024-09-05",
"lunchTime": "2024-09-05"
}

Delete



Final DB records