JPA

JAVA ACADEMY-XIDERAL MTY

Daniel Evaristo Escalera Bonilla

Introduction

Java Persistence API is a set of concepts that allows the persistance of data outside of the Java space, normally with a relational database, as it is just a specification, it needs other tool to implement it, such as Hibernate.

Implementation

To tests the concepts of JPA, I created a CRUD application that connects to a database to créate, read, update and delete objects, the objects in this projects are simulating timecards, which contain information from employees and their entrance, lunch and exit time, the object was defined with the help of lombok to créate its getters, setters and constructors.

```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;

}
```

Here many JPA annotations were used, including Entity and Table to define the object as an entity and assign it to a table, then ID and it's generated value to créate a primary key for the entity and perform an automatic uptade of its value when inserted and finally, the Column annotation to indicate more properties of the object as columns in the database.

To specifiy all the operations needed to opérate the entity, a Data Access Object was created, in which the methods that are going to be applied to the object are defined:

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

import java.util.List;

public interface TimecardDAO {

    List<Timecard> findAll();

    List<Timecard> findByDepartment(String department);

    Timecard findById(long theId);

    Timecard save(Timecard theTimecard);

    void deleteById(long theId);
}
```

The methods include three Read operations in findAll, findById and findByDepartment, a write and update operation in save and a Delete operation in deleteById.

A DAO implementation class was created to define the logic of these methods an perform the actions through an Entity Manager which is the element that interacts with the database:

```java
@Repository
public class TimecardDAOImpl implements TimecardDAO {

    private EntityManager entityManager;

    @Autowired
    public TimecardDAOImpl(EntityManager entityManager) {
        super();
        this.entityManager = entityManager;
    }

    @Override
    public List<Timecard> findAll() {

        TypedQuery<Timecard> theQuery = entityManager.createQuery("from Timecard", Timecard.class);

        List<Timecard> timecards = theQuery.getResultList();

        return timecards;
    }

    @Override
    public Timecard findById(long theId) {
        Timecard theTimecard = entityManager.find(Timecard.class, theId);

        return theTimecard;
    }
```

```java
@Override
public List<Timecard> findByDepartment(String department) {
    TypedQuery<Timecard> theQuery = entityManager.createQuery("SELECT t FROM Timecard t WHERE t.department = :departmentName", Timecard.class);

    theQuery.setParameter("departmentName", department);

    List<Timecard> timecards = theQuery.getResultList();

    return timecards;
}

@Override
public Timecard save(Timecard theTimecard) {

    Timecard dbTimecard = entityManager.merge(theTimecard);

    return dbTimecard;
}

@Override
public void deleteById(long theId) {

    Timecard theTimecard = entityManager.find(Timecard.class, theId);

    entityManager.remove(theTimecard);
}
```

To actívate the dao methods a Service component was defined:

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

import java.util.List;

public interface TimecardService {

    List<Timecard> findAll();

    List<Timecard> findByDepartment(String name);

    Timecard findById(long theId);

    Timecard save(Timecard theTimecard);

    void deleteById(long theId);
}
```

It contains a method for each operation defined in the DAO, and calls it operation with the parameters used during the execution:

```
private TimecardDAO timecardDAO;

@Autowired
public TimecardServiceImpl(TimecardDAO timecardDAO) {
    this.timecardDAO = timecardDAO;
}

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

@Override
public Timecard findById(long theId) {
    return timecardDAO.findById(theId);
}

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

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

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

Finally a Rest controller defines the routes used to access the methods defined in the service, and the parameters and values that must be introduced during the request:

```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;
    }

```

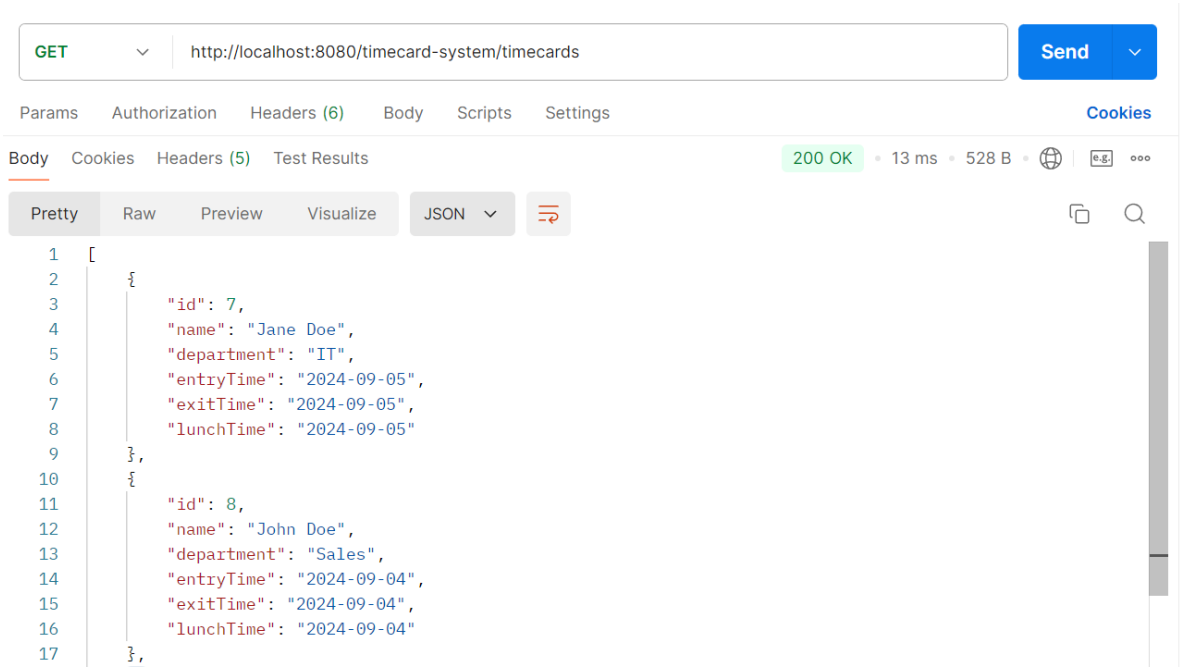
```java
@GetMapping("timecards/department/{department}")
public List<Timecard> getTimecard(@PathVariable String department) {
    return timecardService.findByDepartment(department);
}

@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

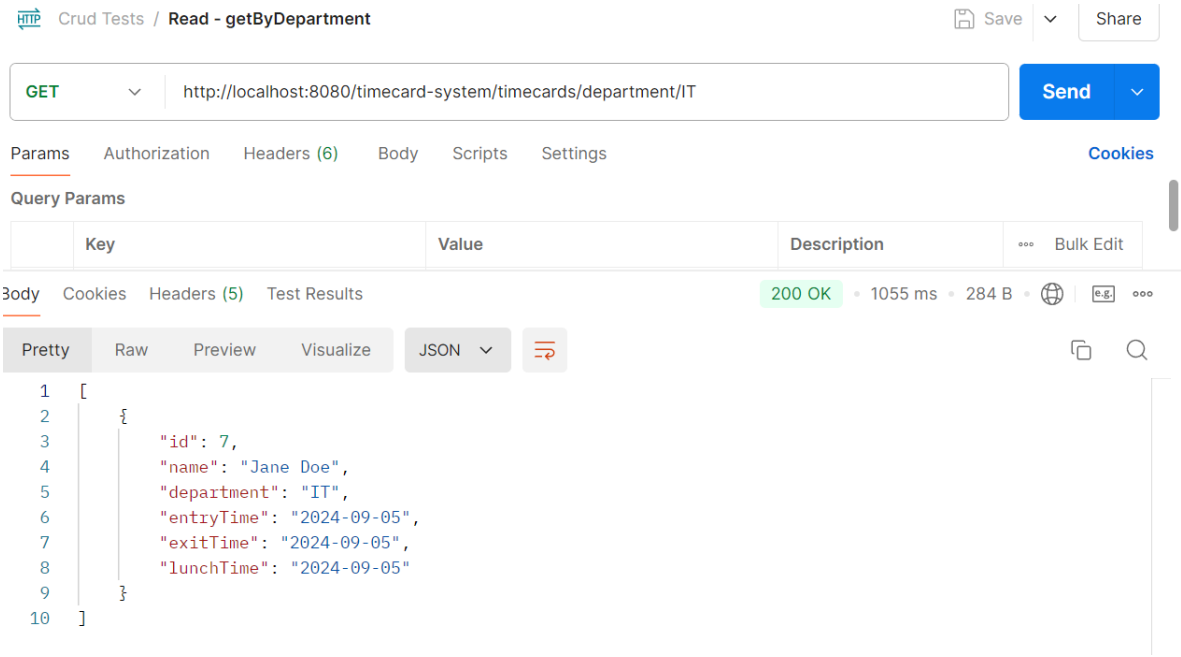
When the application is running we can use the specified URL's to perform operations in the database:

Read

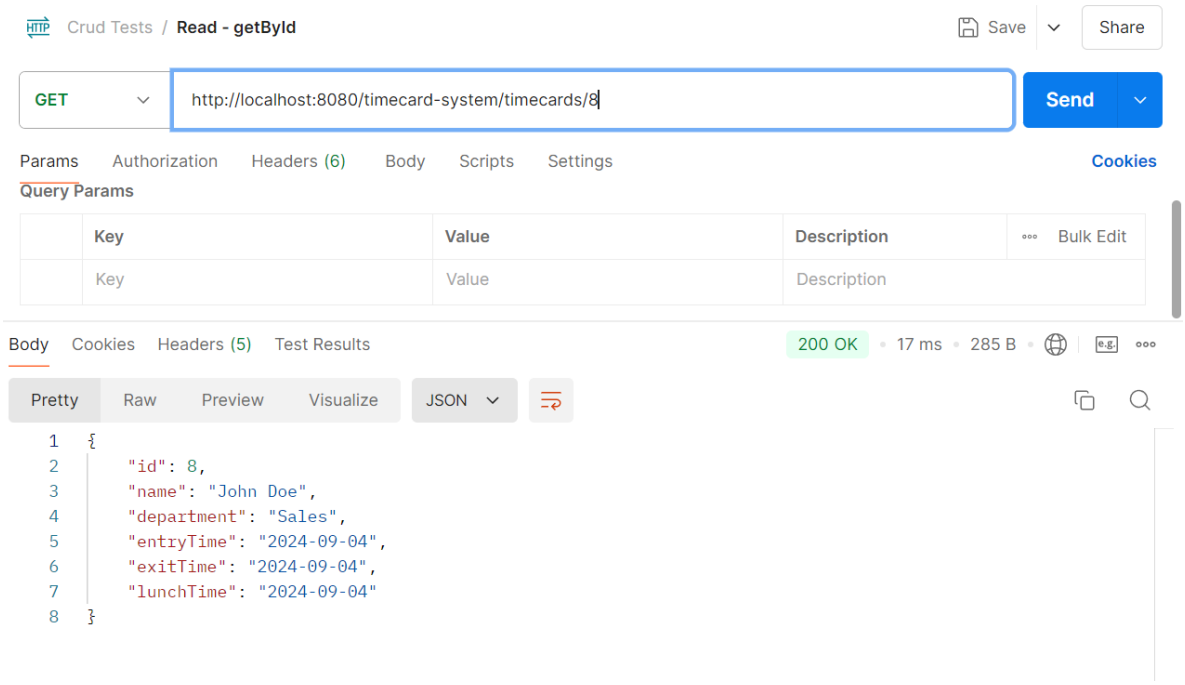
findAll



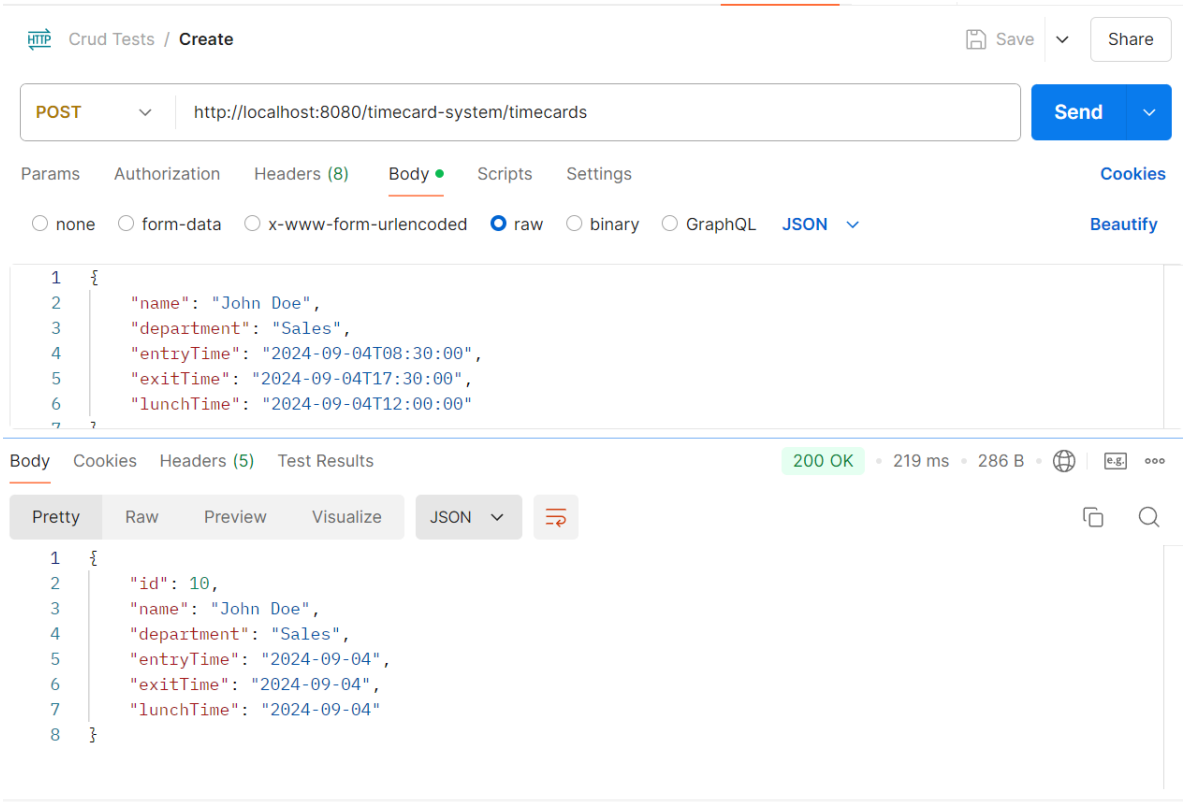
getByDepartment



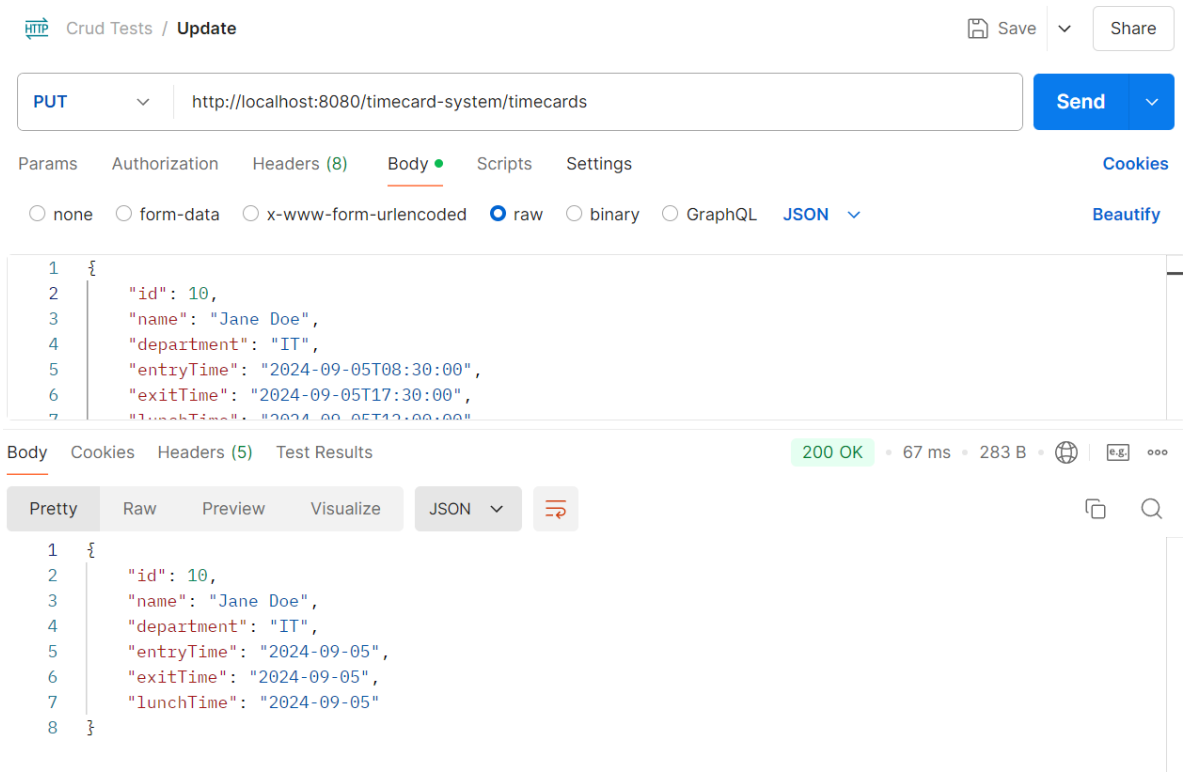
getById



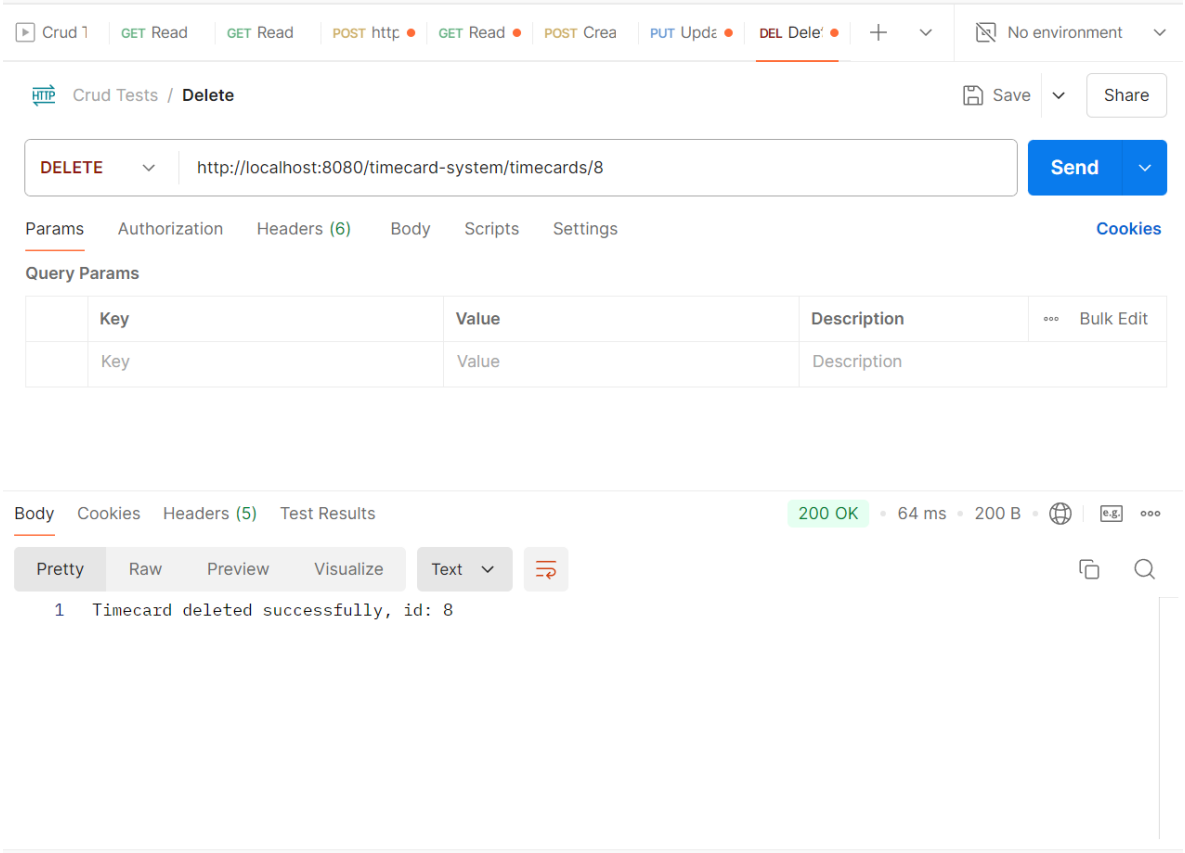
Create



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