

Payment Application IV

Problem Statement:

← Classroom

Hibernate and Relationships
Payment Application 4

?

V

≡

Problem Submissions Solutions Doubts

Payment Application 4

Easy • Score 80/80 • java • Spring Hibernate

Problem statement

Send feedback

In your **Payment Application**, you need to add a Payment Review feature that will allow users to submit queries and receive support from the customer service team. Establish a **One-to-Many** relationship between **Payment** and **PaymentReview**. You need to complete the project based on the given template.

Tasks:-

1. In the **PaymentReview** entity class, there are four attributes **id(Integer)**, **queryPersonName(String)**, **queryType(String)**, and **queryDescription(String)**. Write the necessary methods and add appropriate annotations.

2. Since a **payment** can be linked with multiple **payment reviews**. Create a one-to-many mapping between the **Payment** and **PaymentReview** classes.

3. Create the following APIs in the **PaymentController** class:

- GET "/payment/review/{paymentId}": It fetches the list of all paymentReview associated with the given payment id.
- GET "/payment/queryType/{queryType}": It fetches the list of all payments from the given queryType.

4. Implement the following methods in the **PaymentService** class:

- getAllPaymentsByQueryType(String queryType): This method fetches the list of all paymentReview associated with the given payment id.
- getPaymentReviews(int paymentId): This method fetches the list of all payments from the given queryType.

5. In the **PaymentReviewController** class complete the methods to handle HTTP requests with the required annotation for the following APIs:

- GET "/review/{id}": It fetches PaymentReview for a specific id.
- POST "/review/save" (RequestBody: PaymentReview): It saves PaymentReview in the database.
- DELETE "/review/{id}": It deletes a PaymentReview for a specific id.
- GET "/review/queryType/{queryType}": It fetches the list of all payment reviews containing the given query type from the database.
- GET "/review/allReviews": It fetches the list of all payment reviews from the database.

Step 1
Download starter kit

Step 2
Complete project on local IDE

Step 3
Export code as .zip file

Step 4
Upload .zip file max 50mb

Drag and drop .zip here or [browse](#)

6. Complete the **PaymentReviewDALImpl** implementation class as mentioned below:

a. Autowire **EntityManager**.

b. Override the following methods:

- **getById(int id)**: This method fetches **PaymentReview** for a specific id.
- **getAllPaymentReview()**: This method fetches the list of all **PaymentReview** from the database.
- **save(PaymentReview paymentReview)**: This method saves the **PaymentReview** entity into the database.
- **delete(int id)**: This method deletes the **PaymentReview** entity from the database for a specific id.
- **getByQueryType(String queryType)**: This method fetches the list of **PaymentReview** based on the **queryType** received.

7. Complete the **PaymentReviewService** class as mentioned below:

a. Autowire **PaymentReviewDAL**

b. Complete the following methods:

- **getPaymentReviewById(int id)**: This method fetches **PaymentReview** for a specific id.
- **getAllPaymentReviews()**: This method fetches the list of all **PaymentReviews**.
- **savePaymentReview(PaymentReview newPaymentReview)**: This method saves the **PaymentReview** entity.
- **delete(int id)**: This method deletes **PaymentReview** for a specific id.
- **getPaymentReviewByQueryType(String queryType)**: This method fetches the list of **PaymentReview** based on the **queryType** received.

8. Test the application using Postman.

Output:

The screenshot shows a REST client interface with a POST request to `localhost:8080/review/save`. The request body is a JSON object with the following structure:

```
1 {
2   "queryPersonName": "Saurabh",
3   "queryType": "Bank Issue",
4   "queryDescription": "Description of bank server issue",
5   "payment": {
6     "id": 6
7   }
8 }
9 }
```

The response status is `200 OK` with a response time of `26 ms` and a response size of `123 B`. The response body is currently empty.

The screenshot shows a REST client interface with a GET request to `localhost:8080/payment/id/6`. The response status is `200 OK` with a response time of `17 ms` and a response size of `380 B`. The response body is a JSON object with the following structure:

```
1 {
2   "id": 6,
3   "paymentType": "Cash",
4   "description": "Description 2",
5   "paymentDetails": null,
6   "paymentReviews": [
7     {
8       "id": 3,
9       "queryPersonName": "Saurabh",
10      "queryType": "Bank Issue",
11      "queryDescription": "Description of bank server issue"
12    }
13  ]
14 }
```

GETlocalhost:8080/review/queryType/Bank Issue

Send

ParamsAuthorizationHeaders (7)BodyPre-request ScriptTestsSettingsCookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

This request does not have a body

BodyCookiesHeaders (5)Test Results200 OK982 ms394 BSave as Example

PrettyRawPreviewVisualizeJSON

```
1  [
2    {
3      "id": 1,
4      "queryPersonName": "John",
5      "queryType": "Bank Issue",
6      "queryDescription": "Description of bank server issue"
7    },
8    {
9      "id": 3,
10     "queryPersonName": "Saurabh",
11     "queryType": "Bank Issue",
12     "queryDescription": "Description of bank server issue"
13   }
14 ]
```