

PROJECT NAME: DISCOUNT CALCULATION FOR CUSTOMER ORDER.

1.0 OBJECTIVE

The goal of this microservice was to calculate discount for customer order. There are three requirements for discount to be given:

A customer who

- 1) buys over a thousand Euro worth of products gets 10% on whole order.
- 2) buys five category-2 products gets a sixth for free. As an example, this implies that, for 10 products, he gets 2 for free.
- 3) buys two or more category-1 products gets a 20% discount on the cheapest product in his order.

1.1 APPLICATION LAYER DESIGN.

From the above objective and discount rules, the following objects readily came to mind: Customer, Product, Category, Orders, Order Items. Hence, I proceeded to design the classes and relationships as follows:

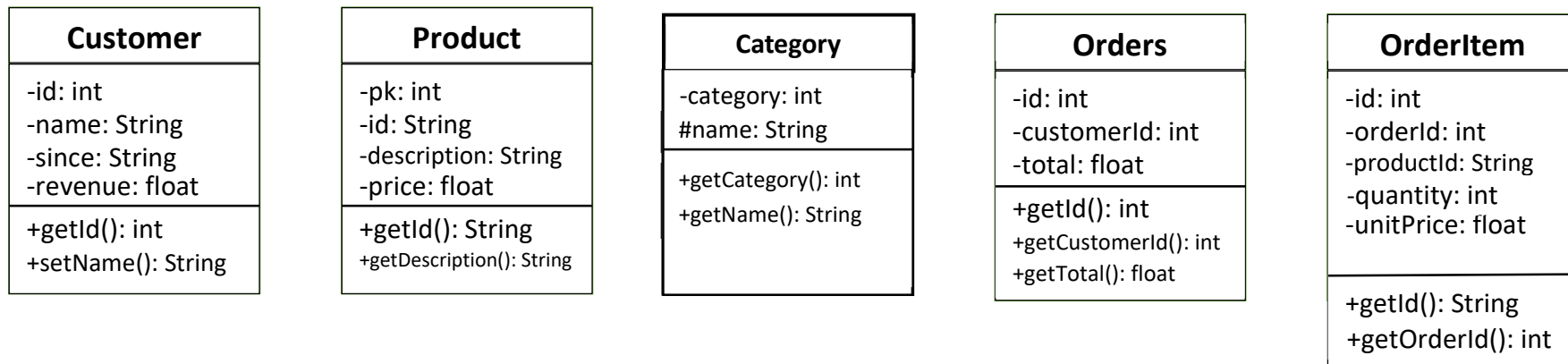


FIG. 1: Classes defined in the application layer.

1.1.1 HELPER CLASSES

These are some other classes providing the service of inserting or managing data in database tables.

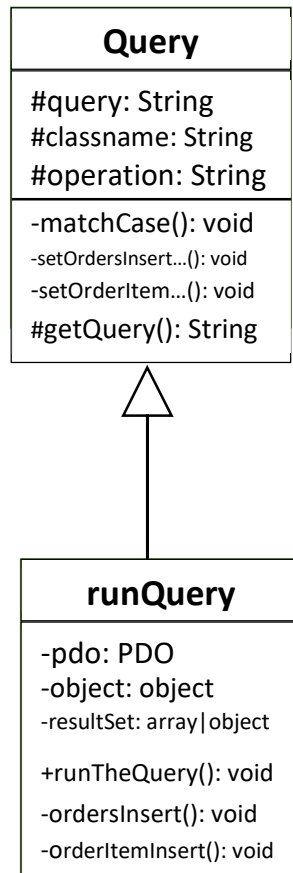


FIG. 2: runQuery class extends the Query class.

1.2 DATABASE LAYER DESIGN

A database in MySQL was created to house the Customer, Product, Category, Orders and Order Items tables. FIG. 3 shows the relationship across the tables.

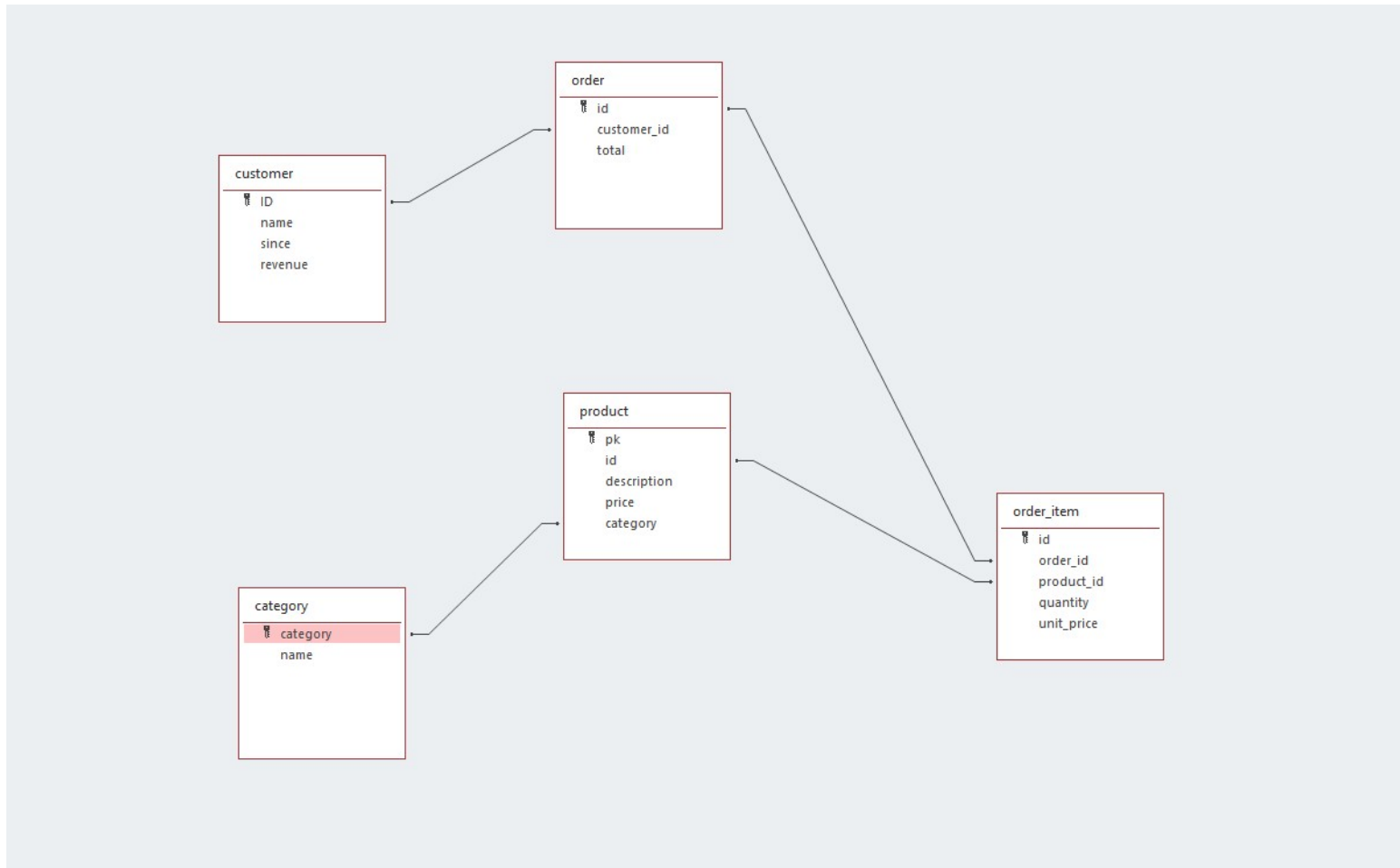


FIG 3: Database tables relationship. Arrows show the fields that connect tables.

1.3 THE DISCOUNT CLASS.

It is necessary to create a blue print that will guide how the discount should be computed. Thus, I created a discount class with the following characteristics.

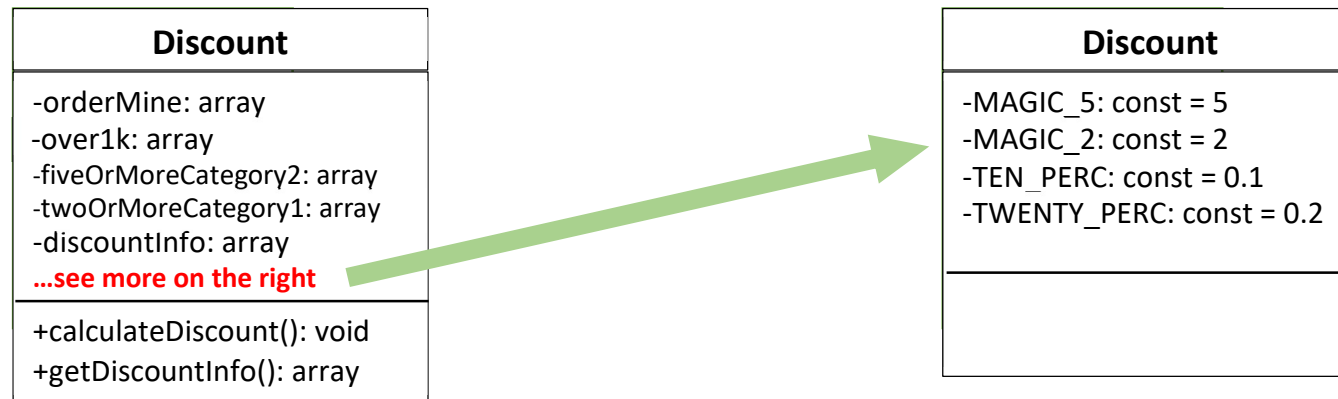


FIG 4: The Discount class has two public methods. The first does the computation and the other returns the result in an array.

The class is great for working with the current discount rules but to allow for new discount rules later on, I decided to use an Interface design. This will allow for new discount... classes to be created when new rules are given.

Any new class will have the same `calculateDiscount()` and `getDiscountInfo()` methods and implement the Discount interface. This way, there will be no break in the application. See FIG. 5 for the relationship between Discount class(es) and Interface.

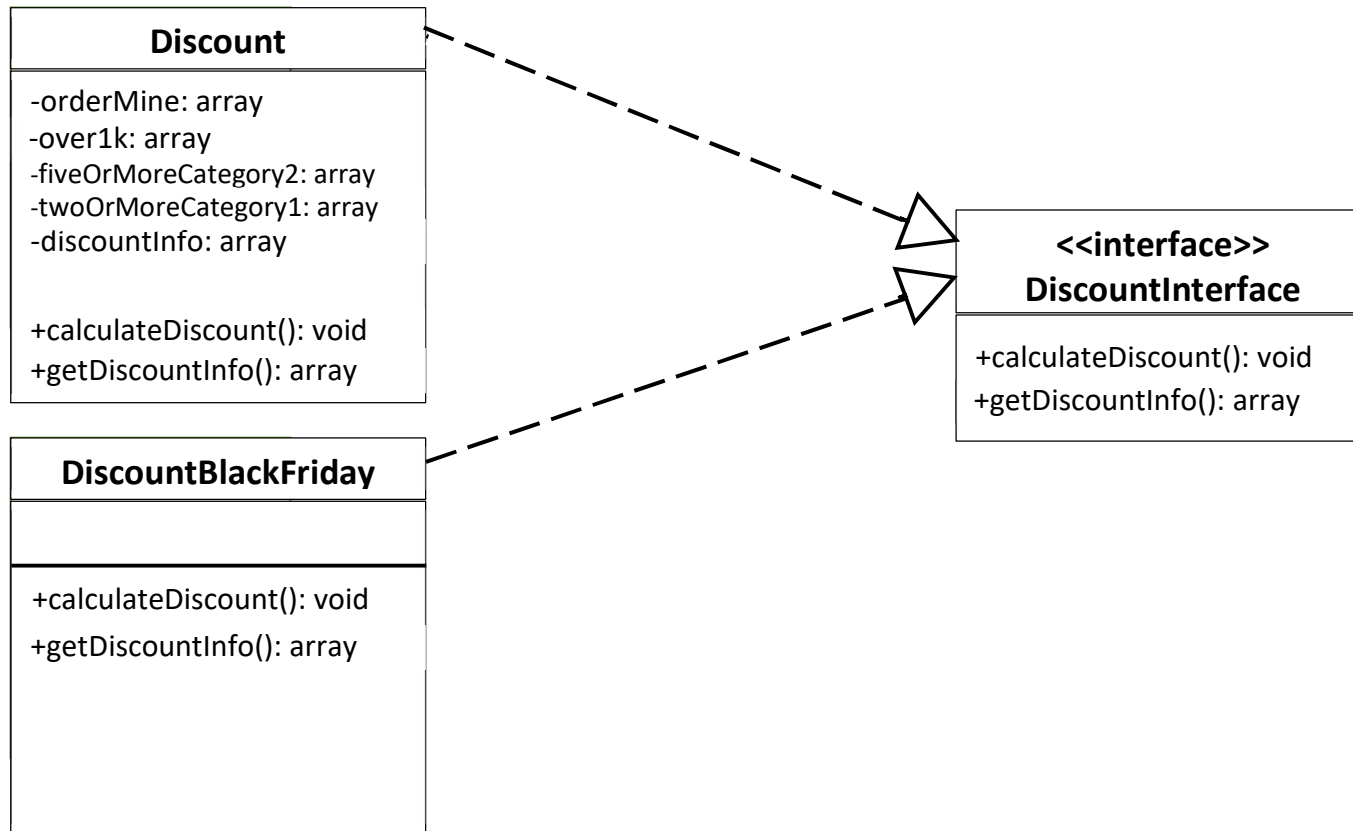


FIG. 5: Using an interface guarantees your code will not break when new discount classes are introduced because they will use the same methods in interface even though the body of the method with same names can be different.

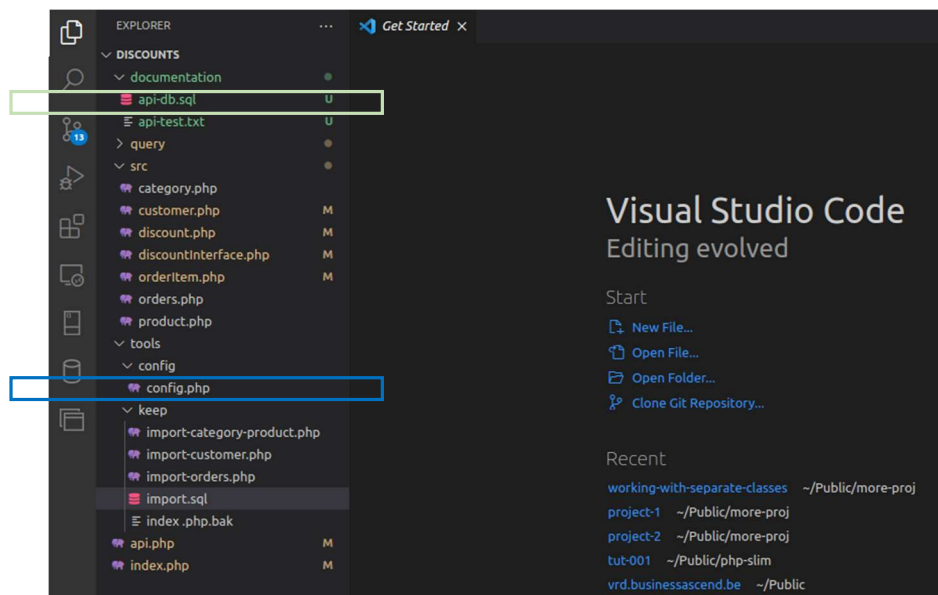
1.3.1 HOW TO USE THE API

To use the API, you have to send a POST request to the URL of the API. While there are many ways to do this, I will describe, perhaps, the simplest way. I have provided here in this repository all the layers to the API:

- i) The database layer: see dump file, documentation/api-db.sql.
- ii) The application layer

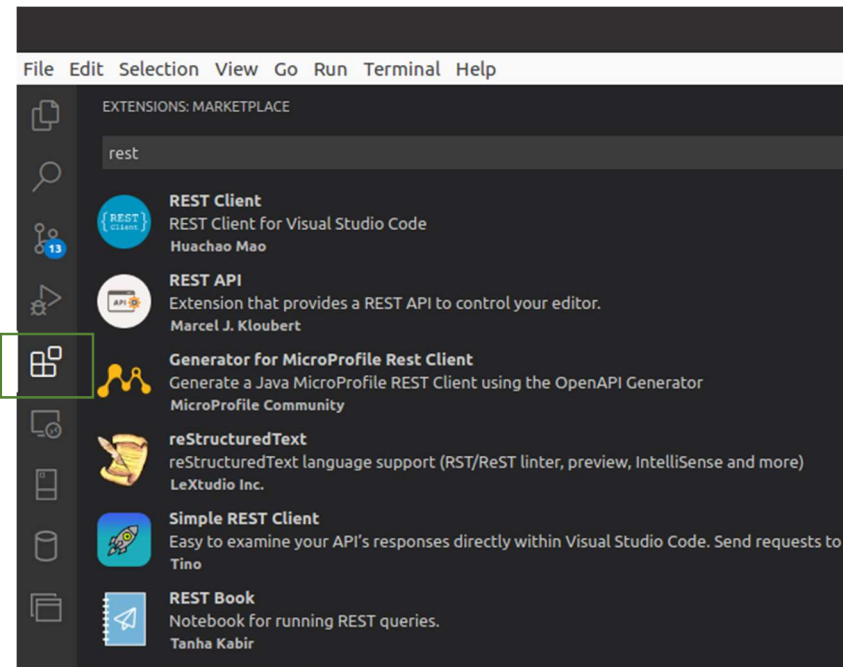
STEPS:

1. Clone the git repository to your local environment. The default root directory would be 'discounts'. You can change this if you will.
2. On a terminal, start the PHP local server: `php -S localhost:8080 -t .`
This application is developed with PHP 8, so, please, make sure to use version 8.
3. While in the root directory of application, run 'code .' to open entire project in Visual Studio Code:



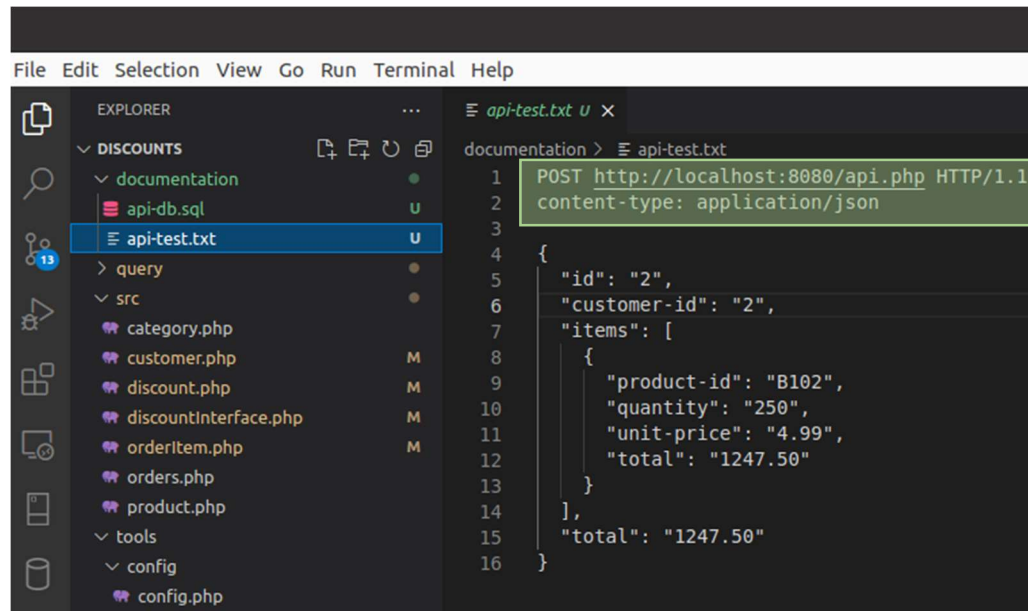
- ❖ Download the api-db.sql under documentation shown within the white box to install the database tables and data.
- ❖ Open the config.php file under tools/config shown within the blue box and enter in database connection parameters.

4. Install a REST Client API extension in Visual Studio code.



- ❖ Click the extension icon within the green box.
- ❖ Type 'rest' in the search box.
- ❖ Install REST Client for Visual Studio Code by Huachao Mao.
- ❖ You can read the usage instructions.
- ❖ 😊

5. Use the REST Client to send order.json to discount API.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project structure with folders 'DISCOUNTS', 'documentation', 'src', and 'tools'. The 'documentation' folder is expanded, showing 'api-db.sql' and 'api-test.txt'. The 'api-test.txt' file is selected and open in the editor. The editor shows a REST client request in the 'api-test.txt' file. The request is a POST to 'http://localhost:8080/api.php' with a 'content-type: application/json'. The request body is a JSON object representing an order item. The response is shown on the right side of the editor, indicating a successful POST operation.

```
1 POST http://localhost:8080/api.php HTTP/1.1
2 content-type: application/json
3
4 {
5   "id": "2",
6   "customer-id": "2",
7   "items": [
8     {
9       "product-id": "B102",
10      "quantity": "250",
11      "unit-price": "4.99",
12      "total": "1247.50"
13    }
14  ],
15  "total": "1247.50"
16 }
```

- ❖ Create a .txt file inside the project directory and call it whatever you want e.g. api-test.txt.
- ❖ Click to open it.
- ❖ Type in the first two lines shown in the green box.
- ❖ Leave the third row empty.
- ❖ Paste in the raw order.json data on line 4.
- ❖ Ensure you paste in the raw json file to avoid copying unwanted and, sometimes, invisible characters – just some caution 😊.
- ❖ Finally, press the CTRL+ALT+R keys to post to the API.

AND the returned object is shown on the right hand side of the window (see section 6) in a file named *Response*. See next page.

6. Result returned is the calculated discount

```
api-test.txt x
documentation > api-test.txt
1 POST http://localhost:8080/api.php HTTP/1.1
2 content-type: application/json
3
4 {
5   "id": "2",
6   "customer-id": "2",
7   "items": [
8     {
9       "product-id": "B102",
10      "quantity": "250",
11      "unit-price": "4.99",
12      "total": "1247.50"
13    }
14  ],
15  "total": "1247.50"
16 }

Response(7ms) x
1 HTTP/1.1 200 OK
2 Host: localhost:8080
3 Date: Sat, 20 Nov 2021 21:52:17 GMT
4 Connection: close
5 X-Powered-By: PHP/8.0.12
6 Access-Control-Allow-Origin: *
7 Content-Type: application/json; charset=utf-8
8
9 {
10   "order_id": 2,
11   "customer_id": 2,
12   "discount_calc": {
13     "over1k": {
14       "qualifies": true,
15       "discountRate": 0.1,
16       "adjustedTotal": 1122.75
17     },
18     "fiveOrMoreCategory2": {
19       "qualifies": true,
20       "freeProd": "50 qty of B102",
21       "newItemQty": "300 of B102"
22     },
23     "twoOrMoreCategory1": {
24       "qualifies": false,
25       "discountRate": 0.2,
26       "amountDeductible": 0
27     }
28   }
29 }
```

7. More result:

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left displays a project structure with folders like 'documentation' and 'src', and various PHP files. The main editor area is split into two panes. The left pane shows an API test in 'api-test.txt' with a POST request to 'http://localhost:8080/api.php' and a JSON body. The right pane shows the corresponding HTTP response in 'Response(4ms)'. The response is an 'HTTP/1.1 200 OK' with headers and a JSON body containing order details and discounts.

```
1 POST http://localhost:8080/api.php HTTP/1.1
2 content-type: application/json
3
4 {
5   "id": "3",
6   "customer-id": "3",
7   "items": [
8     {
9       "product-id": "A101",
10      "quantity": "2",
11      "unit-price": "9.75",
12      "total": "19.50"
13    },
14    {
15      "product-id": "A102",
16      "quantity": "1",
17      "unit-price": "49.50",
18      "total": "49.50"
19    }
20  ],
21  "total": "69.00"
22 }
```

```
1 HTTP/1.1 200 OK
2 Host: localhost:8080
3 Date: Sat, 20 Nov 2021 22:54:06 GMT
4 Connection: close
5 X-Powered-By: PHP/8.0.12
6 Access-Control-Allow-Origin: *
7 Content-Type: application/json; charset=utf-8
8
9 {
10   "order_id": 3,
11   "customer_id": 3,
12   "discount_calc": {
13     "over1k": {
14       "qualifies": false,
15       "discountRate": 0.1,
16       "adjustedTotal": 0
17     },
18     "fiveOrMoreCategory2": {
19       "qualifies": false,
20       "freeProd": 0,
21       "newItemQty": 0
22     },
23     "twoOrMoreCategory1": {
24       "qualifies": true,
25       "discountRate": 0.2,
26       "amountDeductible": "1.95"
27     }
28   }
29 }
```

8. Explanation on the result.

The result is returned as an object with the three main keys: "order_id", "customer_id" and "discount_calc". It is important to show the order and customer id so you can immediately see which customer and order the result is for.

"discount_calc" has three *sub-objects* corresponding to each of the rules:

- I. the "over1k" key holds the information on discount acquired when order total is above 1000 Eur. Thus showing the calculation for the first discount criteria.
Here, the "qualifies" key tells whether customer order qualifies for this class of discount or not.
The "discountRate" key spells out the discount rate to use or used and
The "adjustedTotal" is the total amount after applying the discount rate.

- II. the "fiveOrMoreCategory2" key gives the compilation of the discount applied for category-2 products.
Here also, the "qualifies" key tells us whether the customer order qualifies for this class of discount or not. He qualifies when he buys, at least, 5 of category-2 products.
The discount here is not given as a rate but as free product to be added. So, the "freeProd" key holds the number of free quantities. He gets the sixth for free for every 5 products of category 2 in order.
In the "newItemQty", the "freeProd" has been added to the total item quantity.

- III. the "twoOrMoreCategory1" key holds discount data for the case when customer buys two or more products of category-1.
Just as in previous cases, if a customer order qualifies for this, then "qualifies" will have the value of "true".
The discount rate applied here is also given as the value of the "discountRate" key. It was applied on the price of the cheapest product.
Lastly, the value of "amountDeductible" key gives the amount to be removed from order total.

9. My approach in achieving project objective.

The test has examples of product, customer and order data. These data were inserted into database tables adhering to the table structure and field names and types contained in section 1.2.

So, when API receives the POST json object, which contains the order information of a customer, it converts it to an associative array. Then it inserts the data into the orders and order_item tables, establishing the correct associations across tables.

I considered it necessary to first save the data in database so that, when calculating discount, I can run a query to select the category of item using the product table.

The next stage was to calculate the discount. Using the order json received, I created one instance of the Order class and instances of the OrderItem class with each instance representing a unique product. I put the Order class as the first element in an array and the OrderItem as second. More details on this can be found in api.php page (Section //BEGIN: calculating discounts).

Then I instanced the Discount class, passing in the order array. The Discount class uses an Interface and the first call made here was to the calculateDiscount() method. This method computed the discounts based on the three rules and puts the result of each discount rule in one of three class array properties. The second call was made to the getDiscountInfo() method and this simply combines the three arrays thus having the discount result in one and returns this to caller.

Lastly, the order_id and customer_id information are added. I did a json encode on this array and printed it out, giving you the fabulous result you see in sections 6 and 7.

10. CONCLUSION

While I consider this work a good one, there are, of course, many more areas where improvements can be made. Everything is a function of time and necessity.

APPRECIATION

I appreciate the opportunity given to me to do the test. It was worth it. Thank you.