*May 2021*

# PDM Project Report

Topic 2: Sentiment analysis for product rating


## Members

Trịnh Quang Anh - ITITIU19002
Lê Trần Phong - ITITIU19180
Nguyễn Ngọc Minh Nhật - ITITIU19172
Phạm Duy Thịnh - ITITIU19215
Lê Việt Khôi - ITITIU18300

# Contents

# 1. Introduction

In this report, we will introduce our sentiment analysis system for product rating as an e-commerce website API with the purpose of comprehending the hidden sentiments of customers in feedback comments as well as analyzing their product rating patterns. Besides, the project aims to develop a basic website with various simple but essential features in order to enhance the skills in building simple functionalities that access data stored in database servers as well as gain better understanding of entity-relationship, conceptual schema design, and key issues in conceptual modeling.

## 1.1 Technologies Utilization

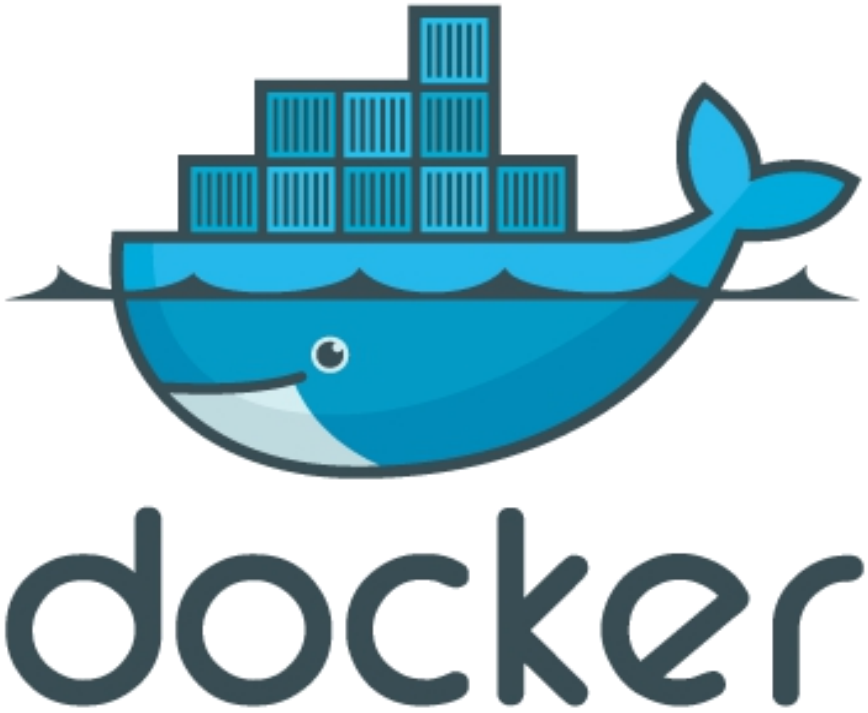
Before getting started with other aspects of the report, first take the time to consider which technologies are being applied to the website. In order to deploy the e-commerce website API as simple as possible, we utilized ***Node.js***, a platform built on ***Chrome's JavaScript*** runtime for easily building fast and scalable network applications. Node. js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices, and along with ***Express.js,*** a free and open-source web application framework for ***Node. js*** using for designing and building web applications quickly and easily.



Node. js (nodejs. org)

Express - Node. js web application framework (expressjs. com)

Moreover, we took advantages of using ***Typescript*** intending for simplifying the Javascript code to make it easier to read and debug. ***TypeScript*** provides highly productive development tools such as the combination of static type checking with intelligent code completion grants a significant boost in developer productivity. Besides, we also try to manipulate Docker, an open-source containerization platform enabling developers to package our application into containers which makes it easier to create, deploy, and run by using containers. Containers allow a developer to package up an application API with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. All of these technologies are utilized for our Back-end development, the server side of our e-commerce web that communicates between the database and the browser.




Empowering App Development for Developers | Docker

# 1.2 E-commerce Website API

Now, we will introduce a brief overview of the features and characteristics of our E-commerce website illustrating as APIs. But first, we want to give a definition of what is an E-commerce website.

An E-commerce, by definition, is a website that allows users who interacts with the website as a role of customers or sellers to purchase or sell tangible goods, digital products or services in an online platform or portal. These websites facilitate online transactions of goods and services through means of the transfer of information and funds over the internet.

E-commerce website is about combining three different systems: a Web server that can manage an online storefront and process transactions, making appropriate links to bank computers to check out buyers' credit card detail, a database system that can keep a check of the items the store has in stock (constantly updating as people make orders and ideally making new orders with suppliers when stocks run low), and a dispatch system linked to a warehouse where the goods can be instantly located and sent to the buyer as quickly as possible.

In our project, we decided to build up a very simple E-commerce website APIs featuring the sentiment analysis for product rating system. Besides this core feature, we also created other essential extra functionalities of the website API.

We implemented all the extra features of an online shopping website APIs in order to illustrate our abilities of handling problems of data querying and management corresponding to every cases of each feature. We have 5 **extra features** below except for the Feedback as the core one:

- User Authentication
- Product Review, Searching, and Filter
- Shopping Cart and Order

- Advertisement and Promotion

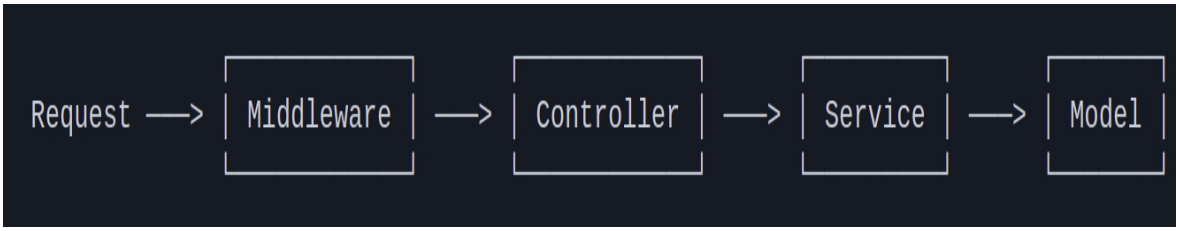


# 1.3 Contributions

| | |
|---|---|
| Database Design | All members |
| User Authentication | Trần Phong |
| Product Review | Duy Thịnh & Minh Nhật |
| Product Searching and Filter | Duy Thịnh |
| Shopping Cart and Order | Quang Anh & Việt Khôi |
| Advertisement and Promotion | Minh Nhật |
| Report | Quang Anh |

# 2. *MSC Architecture*

We mainly focused on applying the Model-Service-Controller (MSC) instead of the original Model-View-Controller, one of the most frequently used industry-standard web development frameworks, to our codebase project aiming at creating our scalable and extensible project as we are approaching building the back-ends of an E-commerce website and deploying APIs of each features in our project. In other words, we mainly focused on implementing the server side.to our codebase project aiming at creating our scalable and extensible project. It divides our codebase into three interconnected elements that allows my team to factor out the various components of the website and more easily to update new features.




***MSC DIAGRAM***

Model Layer: Responsible for data manipulation via database. For each entity, we will create a model representing as table with the attributes defined as column. In this layer, we also implement the “fillable” property to classify which fields are possibly modified throughout the system running process. Besides, we handle the relationship between those models acting as “table” by using relational method to reveal the foreign key of tables.

Controller Layer: Handling and contains the flow of logic for the application and determines what response to send back to a user whenever requests are sent. In this section, we decided that for each action of the user, there will be one controller to handle the problem. Each file in the controller will contain numerous logical functions gaining data from the Service Layer and handling one specific problem before returning the result.

Route: Responsibility for navigating the website's API corresponding to each section of a feature. To be specific, we divided several tasks of a feature to be handled depend on their path, or route, then their Controller will be called out as "an action" to send request to the Service Layer with the purpose of crawling data as JSON form. Some of the "paths" will also require *Access Token*, for instance, before running the "action", and the Middleware Layer is created to solve that crucial issue, especially in Authentication API.

Service Layer: Instead of relying on controller for getting data and implement the logical processing. We separated the two tasks that Service Layer will handle these business logics. To be specific, Service Layer get the data and logically handle it before being called by the Controller. Similar with the Controller, for each action or request of a user, we have one Service to handling the problems. The special thing is, in this section, we based on a few similarities of the SQL to querying and working with the data. We will fully illustrate about it in the Database Design and Management Section later in this report.

Middleware Layer: Responsible only for the request checking and processing before being sent to the Controller layer.

Besides, we also deploy the Validators especially for the User Authentication feature that automatically check whether the email or password entered is valid or not.

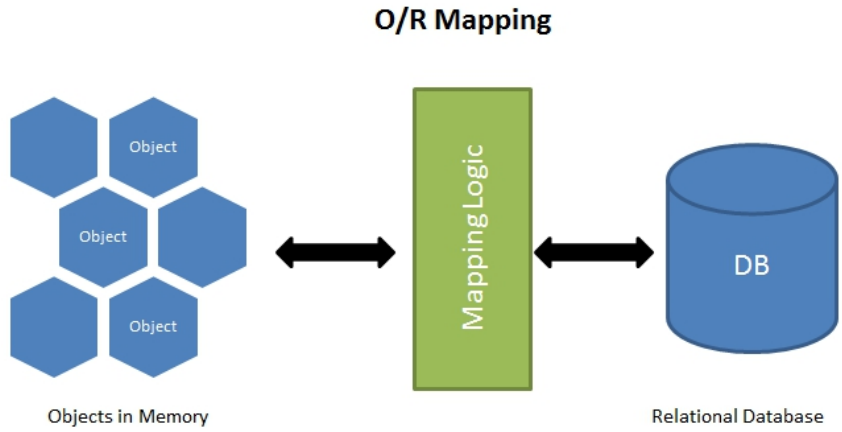
As utilizing the MVCS architecture framework, we benefited quite a lot. We considered it is crucial for our present project's development.

All the six features of our E-commerce website that we mentioned above have purely the same ways of querying data from the database that making use of the extraordinary query commands developed on our own with the purpose of converting those commands to the SQL for data processing based on the users' actions and requests.

# 3. *Database Design*

*Design the database system management with ORM*

In this section, we will illustrate our database system management using the few technique called ORM which stands for Object-Relational Mapping, a programming technique for converting data between incompatible type systems using object - oriented programming languages.




The basic feature of ORM is to encapsulate a database in one object. One part of the object will contain the data, and the other part will take care of how the data is processed and turn it into a relational database. ORM is the only solution for our project database management to have a rich, object-oriented business model and still be able to store it and write effective queries quickly against a relational database.

This is how we define the Models that we mentioned above, we also have the module handling query building for Typescript with the purpose of supporting the Service accessing and receiving data from the database utilizing ORM.

The query builder contains numerous methods of Models implementing inside the Services for querying data. Those methods are just the same as SQL.

In order to manage our database, we also use phpMyAdmin, a free software tool written in PHP. Through this software, we can try to write some simple query sentences for testing our database management. And for the APIs, we use Postman, a collaboration platform for API development helping us to build, test, and modify APIs as it has the ability to make various types of HTTP request such as (GET, POST,….) to the Route and by that redirects to the Controller, so it will be easy for our team to test our data query command through Services and Controllers corresponding to each feature of the website.


# 4. *Feedback System*

### *4.1 What is sentiment analysis for product rating system?*

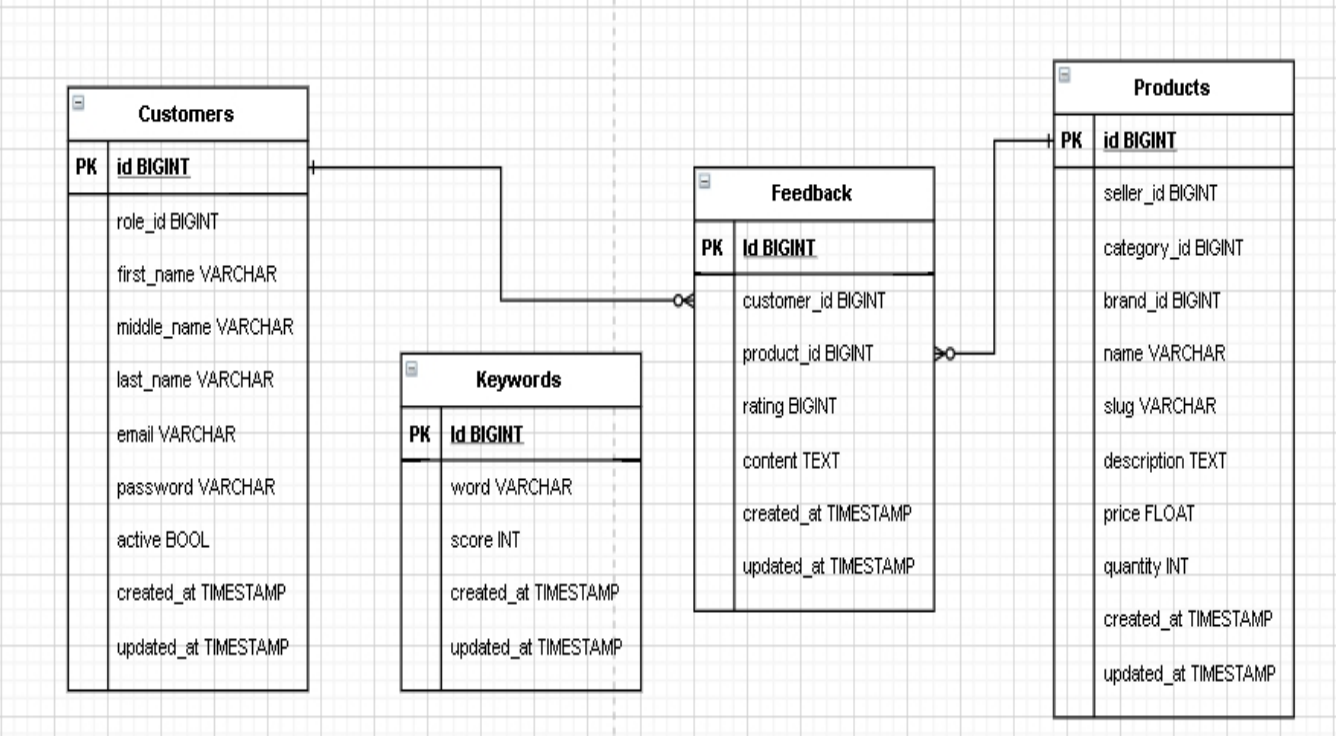
Sentiment analysis for product rating is a system, which rates any product based on hidden sentiments in the comments which allows the registered user to view the products and their features along with the option of commenting about the product. The system uses sentiment analysis methodology in order to achieve desired functionality. This is an E-commerce web application, so we integrated this system as the core feature in our E-commerce API.

### *4.2 Operation and Main objectives of the Feedback System*

For feedback feature, the core functionality for our sentiment analysis product rating system, we decided to build up a rate counting system for various customers to leave their comments on products aiming at understanding the hidden sentiments of customers in feedback and comments and analyze their product rating patterns. To be specific, when a user comments on a particular product, the sentiment analysis system analyzes the keywords in the comment to find the match with the keywords stored in the database. Database of the system has various keywords denoted as negative and positive words with a score, which helps the system to recognize the comments and rank them accordingly based on the calculation of average point. After analyzing the matches against the positive and negative keywords and sentiments, the system ranks a product based on their rating points corresponding to “good”, “bad” comments. The final average point will be modified overtime based on this point ranking system. Thus, users can use this system to find out reviews on a product and specify whether the product is good, or bad.

The main goal of this sentiment analysis system is to understand the hidden sentiments of customers in feedback and comments and analyze their product rating patterns.

### *4.3 Relational Models*




Customers (***id, role_id***, first_name, middle_name, last_name, email, password, active, created_at, updated_at).

Products (***id, seller_id, category_id***, brand, name, slug, description, price, quantity, created_at, updated_at).

Feedbacks (***id, customer_id, product_id***, rating, content, created_at, updated_at).

Keywords (***id***, word, score, created_at, updated_at).

In the Customers Table, we have many fields related to the information of the user and especially the role_id as Foreign Key, and we will talk about this later. Same with the Products Table, we have numerous fields for showing information of products and seller_id as Foreign Key to connect with the seller table as well as the category_id to categorize product types.

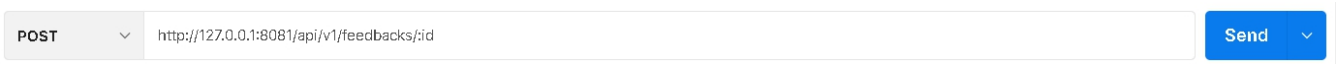
About the Feedback Table, of course, we will have the customer_id and product_id as Foreign Key to connect 3 tables. The field “rating” will represent the level of satisfaction of customers, we have 5 levels, and the field “content” is compulsory for customers to feedback on the products because we will rely on this field to analyze customers' sentiment on products.

The relationship one-to-many between Customer, Product, and Feedback tables are clearly illustrated, indicating that a customer can have as much feedback on products as he or she desires, and a product may have numerous feedbacks as well.

For the Keywords table, we use the field “word” to track the customers’ negative or positive comments along with the field “score” in the database with the purpose of calculating the average points of a product.

### *4.4 Queries Section*

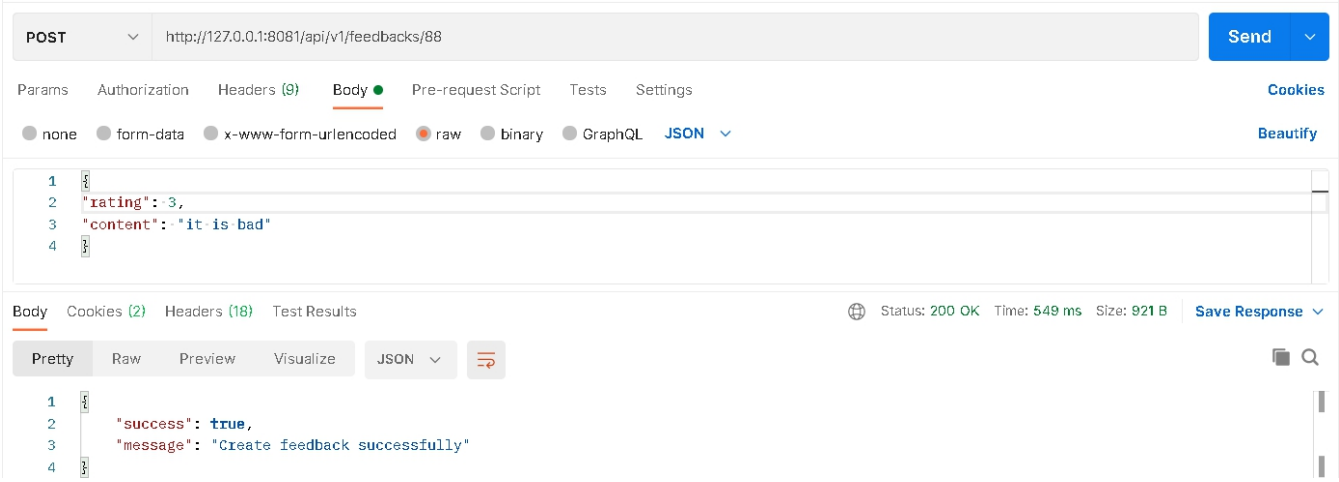
As we have not implemented the front-end for our system yet, so we just utilize the Postman application for testing the system.




Basically, this is where our URL is entered in order to call the Route for navigating actions called by the Controller.

### *4.4.1 Feedback Creation*

For this action, we will have to send a POST request for modifying data in the database. We enter our data as JSON, and our feedback will be created successfully. Let's have a look at the productId 88:




And here is how our query processing to the database:

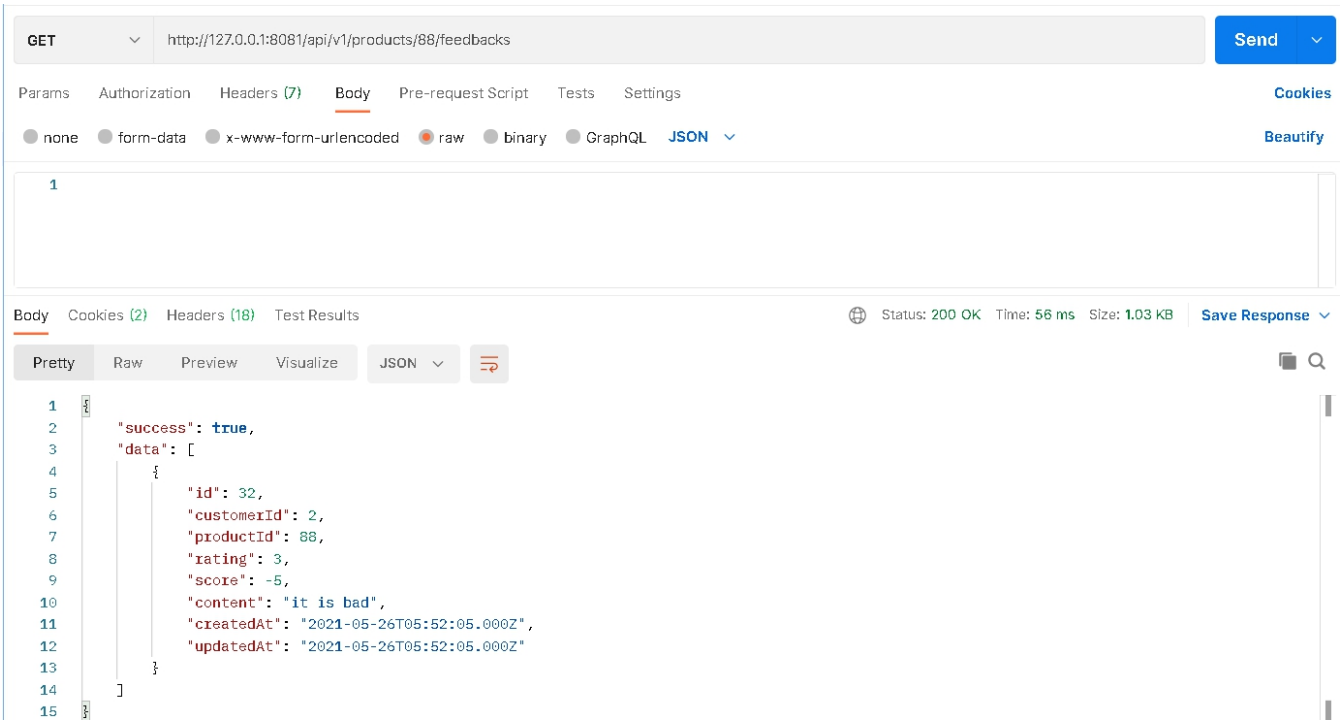
SELECT `customers`.`id`, `customers`.`firstName`, `customers`.`middleName`, `customers`.`lastName`, `customers`.`email`, `customers`.`password`, `customers`.`active`, `customers`.`createdAt`, `customers`.`updatedAt`
FROM (`customers`)
WHERE (`id` = '2')

SELECT `customers`.`id`, `customer_roles`.`id` AS `role-id`, `customer_roles`.`name` AS `role-name`, `customer_roles`.`createdAt` AS `role-createdAt`, `customer_roles`.`updatedAt` AS `role-updatedAt`
FROM (`customers`) left join (`customer_roles`) ON (`customers`.`roleId` = customer_roles.id )
WHERE (`customers`.`id` = '2' )

SELECT SUM(`keywords`.`score`) AS `diff`
FROM (`keywords`)
WHERE (`id` > 0 ) AND (`word` = 'it' OR `word` = 'is' OR `word` = 'bad' )
INSERT INTO feedbacks (`customerId`, `productId`, `rating`, `score`, `content`) VALUES ("2","88","3","-5","it is bad");

### *4.4.2 Show Feedback of a Product*

We go on testing our feedback on the productId 88 and display it on our screen:




And here is how the showing product feedback query worked:

SELECT `feedbacks`.`id`, `feedbacks`.`customerId`, `feedbacks`.`productId`, `feedbacks`.`rating`, `feedbacks`.`score`, `feedbacks`.`content`, `feedbacks`.`createdAt`, `feedbacks`.`updatedAt`
FROM (`feedbacks`)
WHERE (`productId` = 88 ) LIMIT 0, 10

### *4.4.3 Show Sentiment-based Feedbacks Sorted*

In order to display all the feedbacks of products sorted by average point, we also have the query:

SELECT `products`.`id`, `products`.`sellerId`, `products`.`categoryId`, `products`.`brandId`, `products`.`name`, `products`.`slug`, `products`.`price`, `products`.`description`, `products`.`quantity`, `products`.`createdAt`, `products`.`updatedAt`,
AVG(`feedbacks`.`score`) AS `score`
FROM (`products`)
INNER JOIN (`feedbacks`)
ON (`products`.`id` = `feedbacks`.`productId` )
GROUP BY (`products`.`id`)
ORDER BY (score) DESC LIMIT 0, 20

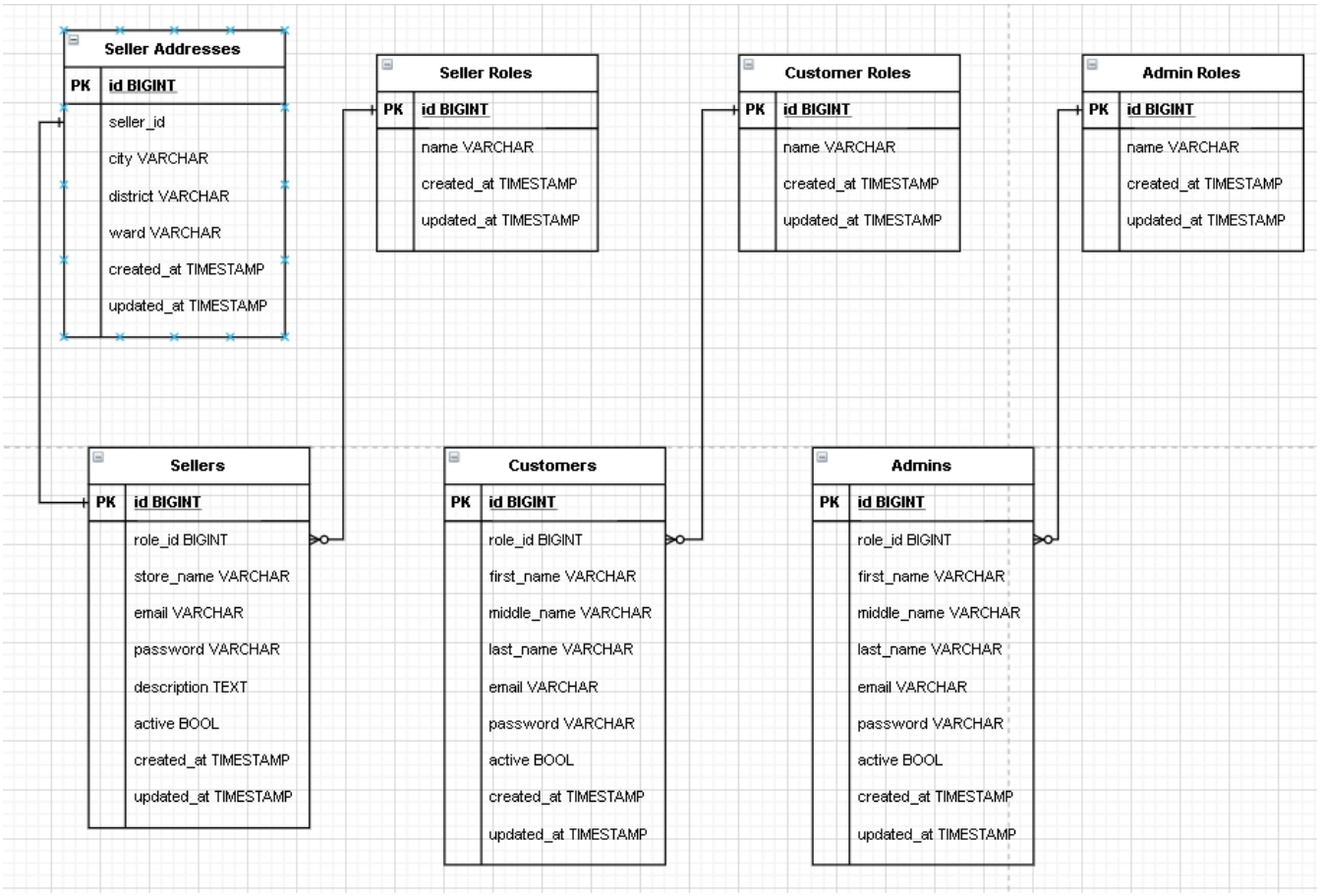
# 5. Extra features

6.

We created numerous tables illustrating attributes and relationship of the extra features in our project:

## 5.1 ***Relational Models for User Authentication***

We have the three Roles table which later categorizes into three main users of our website:




Sellers (***id, role_id***, store_name, email, pasword, description, active, created_at, updated_at)
Seller Roles (***id***, name, created_at, updated_at)

Customers (***id, role_id***, first_name, middle_name, last_name, email, password, active, created_at, updated_at).
Customer Roles (***id***, name, created_at, updated_at)

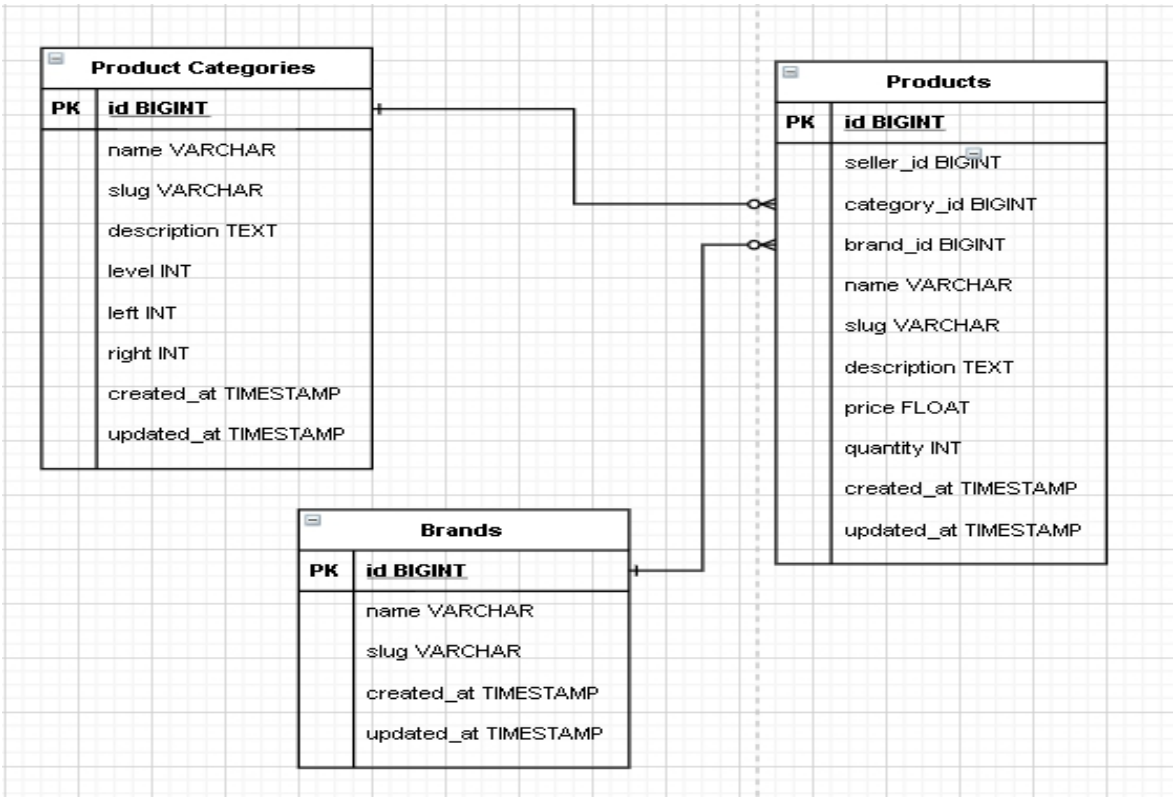
Admins (***id***, first_name, middle_name, last_name, email, password, active, created_at, updated_at).
Admin Roles (***id***, name, created_at, updated_at)

Seller Addresses (***id, seller_id***, city, district, ward, created_at, updated_at)

As we can see, about the role_id, we defined 3 objects as the user of our system: Seller, Customer, and Admins, these types of users own a distinguished foreign key called role_id for user classification as there are three one-to-many relationships on the Sellers, Customers, and Admins tables representing many users just belong to unique type of roles: Seller, Customer, Admins. We also have the seller table and its one-to-one relationship with addresses table, demonstrating the unique seller address.

### *5.2 Relational Models for Product*

We also design table for showing the relationship between Products, Product Categories and Brands:




http://mikehillyer.com/articles/managing-hierarchical-data-in-mysql/

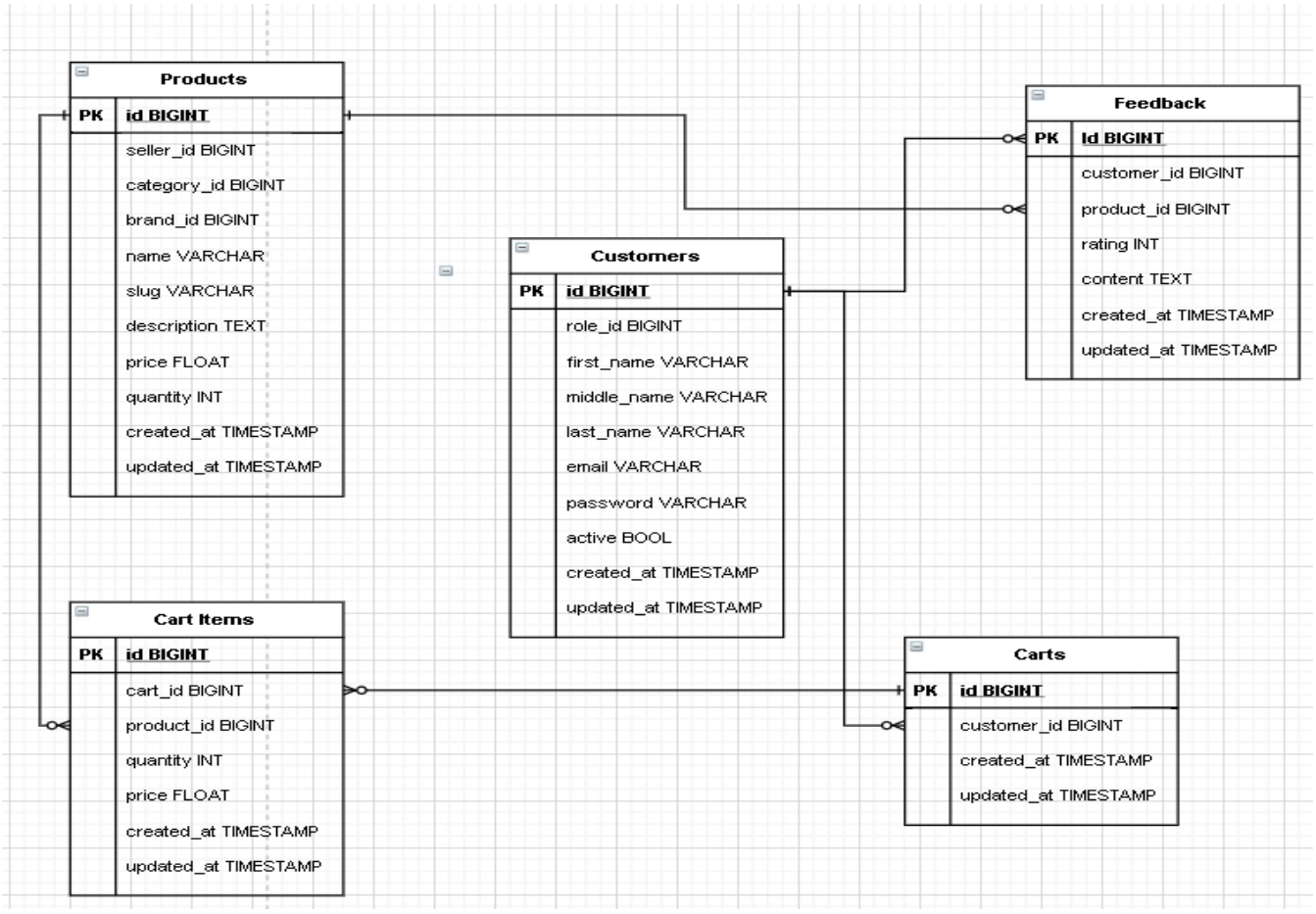
Products (***id, seller_id, category_id***, brand, name, slug, description, price, quantity, created_at, updated_at).

Brands (***id***, name, slug, created_at, updated_at)

Product Categories (***id***, name, slug, description, level, left, right, created_at, updated_at)

As can be seen in the picture above, we can clearly see that how the many-to-one relationship of Product has been illustrated. From the picture, we can easily realize the meaning of the two relationship that is one product can only belong to one brand and category.

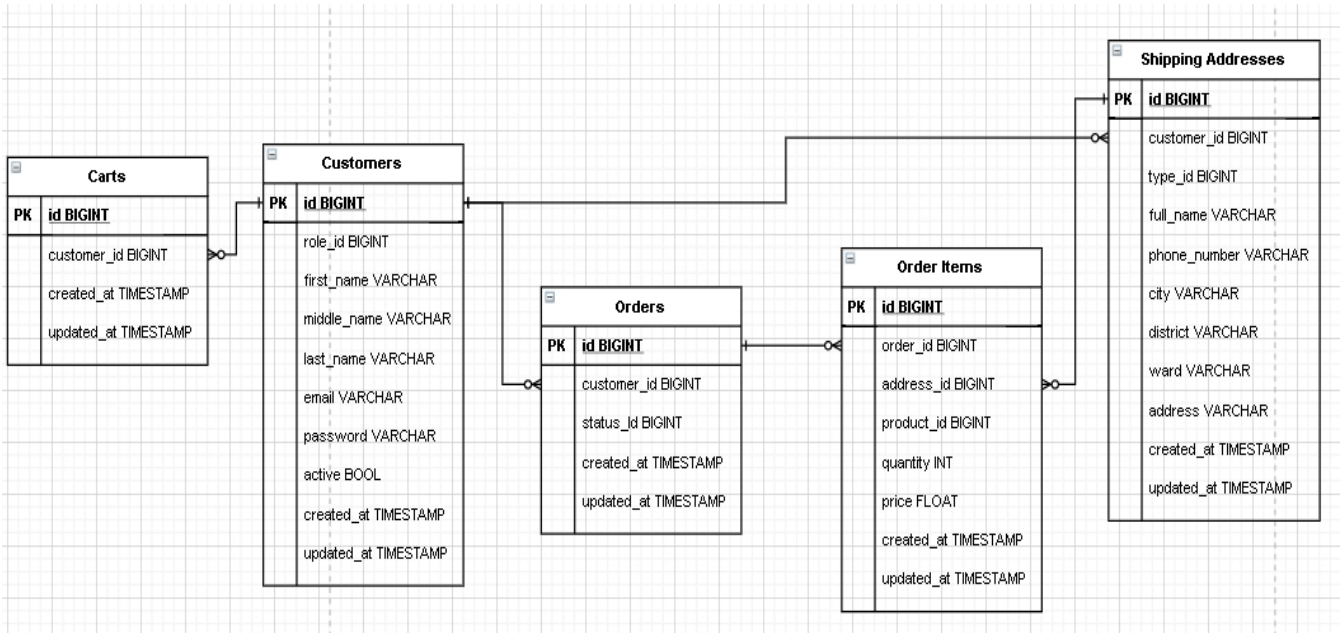
## *5.3 Relational Models for Shopping Cart and Order*




Carts (***id, customer_id***, created_at, updated_at)

Cart Items (***id, cart_id, product_id***, quantity, price, created_at, updated_at)

As we illustrate above, one customer can own many carts, and numerous cart items, or product inside the cart, must belong to only one cart only as cart_id as a foreign key in the Cart Items table. Moreover, one product can represent many cart items as well, product_id inside the Cart Items table as foreign key.

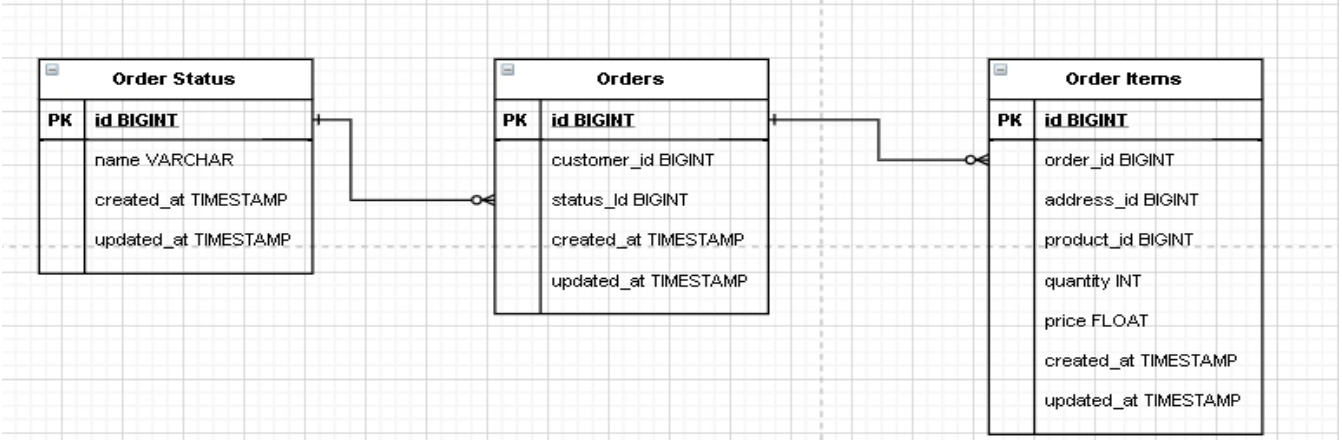



Orders (***id, customer_id, status_id***, created_at, updated_at)

Order Items (***id, order_id, address_id, product_id,*** quantity, price, created_at, updated_at)

Shipping Address(***id, type_id, customer_id***, full_name, phone_number, city, district, ward, address, created_at, updated_at)

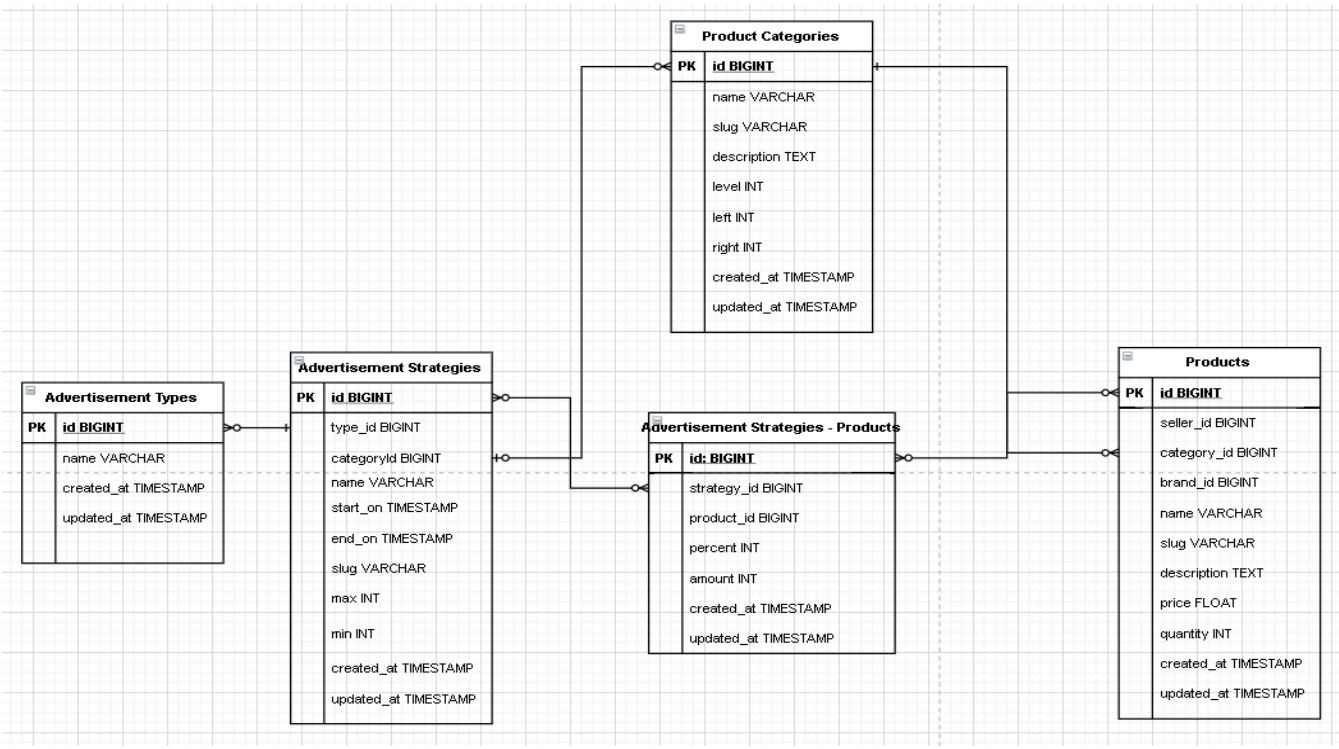
As clearly be seen in the picture, we can realize a customer will have a one-to-many relationship with cart and order meaning that a customer could have many carts as well as orders. In the Order Items table, we decided to create the Order Items table for generating bill for customer when they are at the payment section. Obviously, we have many items in an order and with one address so that is why it will contain address_id as foreign key from the Shipping Addresses table and order_id as well in order to classify which order based on customer_id.




Order Status (***id***, name, created_at, updated_at)

We also created Order Status table for classifying the status of order, which is processing, or delivering, or received, so one order will just have one status at a specific time only.

## *5.4 Relational Models for Advertisement and Promotion*




Advertisement Types (***id***, name, created_at).

Advertisement Strategies (***id, type_id, categoryId***, name, start_on, end_on, slug, max, min, created_at, updated_at).

Advertisement Strategies – Products (***id, strategy_id, product_id***, percent, amount, created_at, updated_at).

For the Advertisement extra feature, we can see that many products can have numerous strategies for advertising due to the many-to-many relationship and product_id as a foreign key inside the Advertisement Strategies for Product table. Moreover, the Advertisement Strategies table plays a significant role in this section as it requires category_id from the Product Categories table as foreign key and type_id as well showing that one strategies may belong to various types leading to the many-to-many relationship between Advertisement Strategies and Advertisement Strategies for Products table.

# 6. *Conclusion*

## *6.1 Learning Achievement from the project developement*

The knowledge and experience we gain from this project can prove highly valuable to the success of future projects. In this Principle of Database Management Project, we have learned numerous concepts and technologies such as Node.Js, Express.Js, Typescript, Docker, that we have not even heard or understand before getting started to deploy it. With the assistance of these technologies, we successfully built up the whole E-commerce website APIs with extremely basic extra features.

Moreover, we are now familiar with setting up the database system and fully understand the entity-relationship model as well as its characteristics. Our abilities of constructing database models for practical applications are being strengthened through out the course along with implementing our tasks individually.

Finally, we had the chance of teaming up and work effectively via the supportance of Git, and Github for managing our codebase repository through web-based platform used for version control. Git simplifies the process of working with other teammate and makes it easy to collaborate on projects from anywhere which enhances our ability of working as a team transparently.

## *6.2 Benefits and Disadvantages of Sentiment-based Product Rating System*

Firstly, about the benefits, User can easily share his view about the product and people can easily decide whether the product posted is good or bad using this feature. On the other hand, the system will match the comment with those keywords which are limited in the database and the rest of the words are ignored by the system, so it is difficult to analyze and evaluate complex feedbacks appropriately.