Binary Bandits - Assignment 5

TASK 1 - RESEARCH

Binary Bandits

Online shopping has become a very popular choice opposed to in person shopping with around 48% of South Africans using e-commerce websites for general shopping in all categories such as clothing,appliances etc.. Around 84% of that was grocery shopping[8] which isn't surprising as online shopping has been shown to be 5% cheaper and saves 77% of the consumers time [6] being the biggest factor driving the growth of this sector. The focal point of this is an analysis of online grocery shopping apps, specifically the most popular in south africa consisting of Woolworths Dash,Checkers6060,pnpASAP and Spar2U, with checkers being the largest in the sector followed closely by woolworths dash. Online shopping has become a huge market for grocery stores with online shopping making up close to 50% of grocery sales for both checkers and woolworths and online grocery shopping growing by 150% for checkers and 75% for pick n pay with the entire sector growing by 16% from 2024 to 2025 [7]

The most popular items sold by any grocery store is always milk followed by bread[1] but each shop has its own category that its known for with premade meals and luxury products being sold the most by Woolworths dash with fruits and vegetables being the most common category of items sold by both Checkers and Spar and lastly pick n pay's most sold item being coca cola with their most popular category being shelf stable foods. Though all apps have their own most sold categories the main categories consumers shop for in online grocery stores consists of fresh vegetables and pantry staples[3]. Though all these apps have very different price points for the items[4]. Overall checkers was the cheapest in all retailers, with spar being the cheapest for only food, and woolworths being the most expensive in both food only and overall.

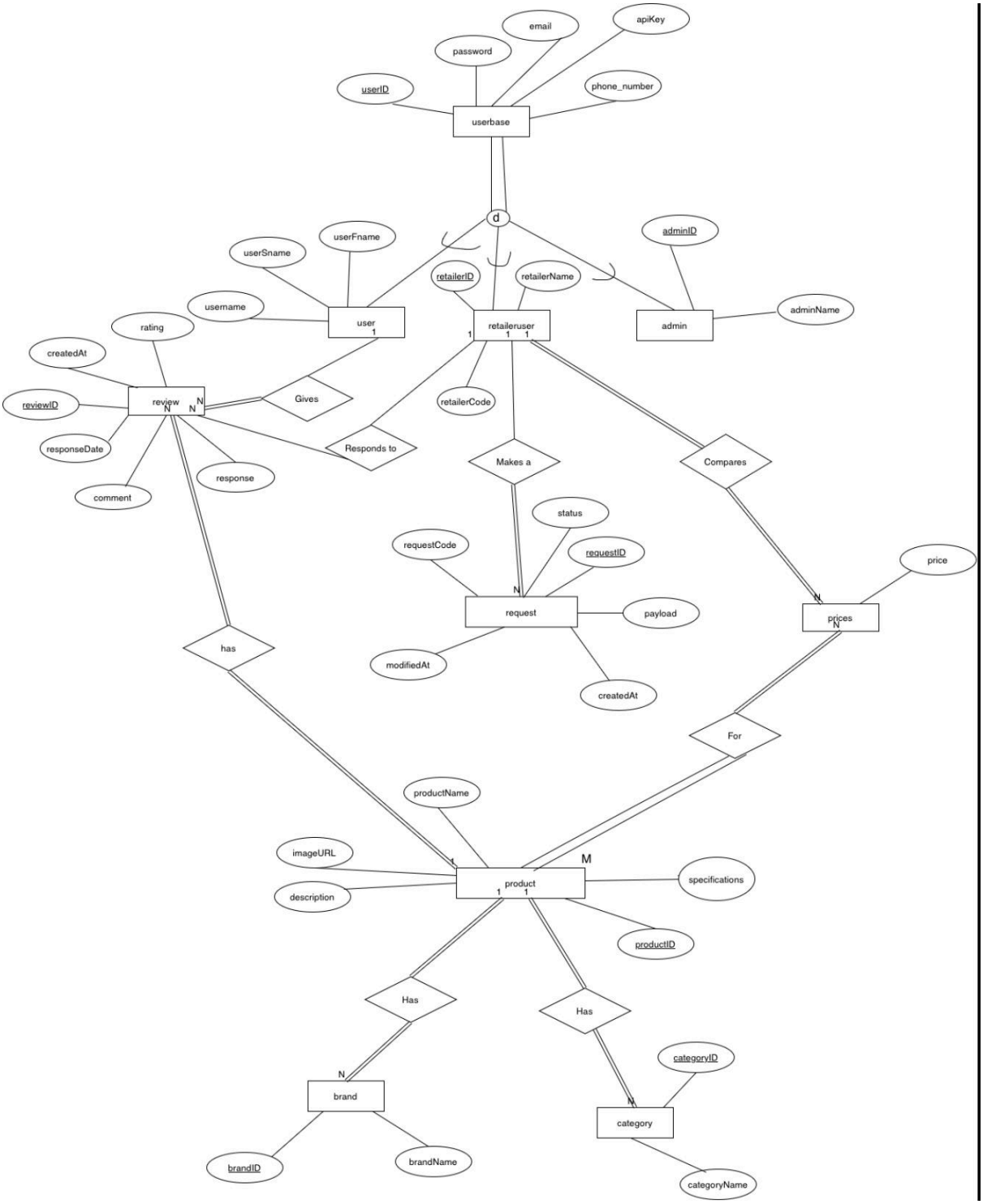
These statistics can only be seen to the consumer though comparison sites, whose purpose is to compare the price of the same item in different retailers. The most popular sites being used in south africa consist of PriceCheck,OneCart,Pryce and buck cheap[5] being some of the most common in south africa with PriceCheck being rated number one. Thought the comparisons are good not all of the websites include the biggest retailers with PriceCheck excluding Spar and all of their user interfaces being very difficult to manage with most of them requiring upwards of 3 to 5 minutes to set up or even having to search specifically for grocery comparisons in google to access the comparisons making the user experience very bad.

Bibliography

[1] GFreshMart, "Most selling supermarket items." [Online]. Available: https://www.gfreshmart.com/most-selling-supermarket-items/. [Accessed: 30-April-2025].
[2] News24, "We were finally able to make a purchase on Woolies Dash – here's how it went," Jan. 2021. [Online]. Available: https://www.news24.com/news24/xarchive/we-were-finally-able-to-make-a-purchase-on-woolies-dash-heres-how-it-went-2021-1. [Accessed: 29-April-2025].

[3] Shoprite Holdings, "Supplier Awards 2022," 2022. [Online]. Available: https://www.shopriteholdings.co.za/newsroom/2022/supplier-awards-2022.html. [Accessed: 30-April-2025]].
[4] BusinessTech, "Online food price war: Checkers Sixty60 vs. Pick n Pay ASAP vs. Woolies Dash vs. SPAR2U." [Online]. Available: https://businesstech.co.za/news/lifestyle/716076/online-food-price-war-checkers-sixty60-vs-pick-n-pay-asap-vs-woolies-dash-vs-spar2u/. [Accessed: 30-April-2025]].
[5] SimilarWeb, "Top websites in South Africa: E-commerce and shopping (price comparison)." [Online]. Available: https://www.similarweb.com/top-websites/south-africa/e-commerce-and-shopping/price-comparison/. [Accessed: 29-April-2025]].
[6] BusinessTech, "Online vs. in-store grocery shopping prices in South Africa." [Online]. Available: https://businesstech.co.za/news/internet/259577/online-vs-in-store-grocery-shopping-prices-in-south-africa/. [Accessed:30-April-2025]].
[7] WaveGrocery, "Online grocery shopping statistics 2024: Latest data summary," 2024. [Online]. Available: https://www.wavegrocery.com/blogpost/online-grocery-shopping-statistics-2024-latest-data-summary. [Accessed:28-April-2025]].
[8] SAJIM, "[Article Title]," South African Journal of Information Management. [Online]. Available: https://sajim.co.za/index.php/sajim/article/view/1637/2523. [Accessed: 29-April-2025].

TASK 2 - (E)ER DIAGRAM



TASK 3 - (E)ER-DIAGRAM TO RELATIONAL MAPPING

Step 1

USERBASE

UserID	password	emain	phoneNumber

PRODUCTS

ProductID	productName	description	brandID	categoryID	imageURL	specifications

RETAILER

retailerID	retailName

CATEGORY

categoryID	categoryName

PRICES

retailerID	productID	price

BRANDS

brandID	brandName

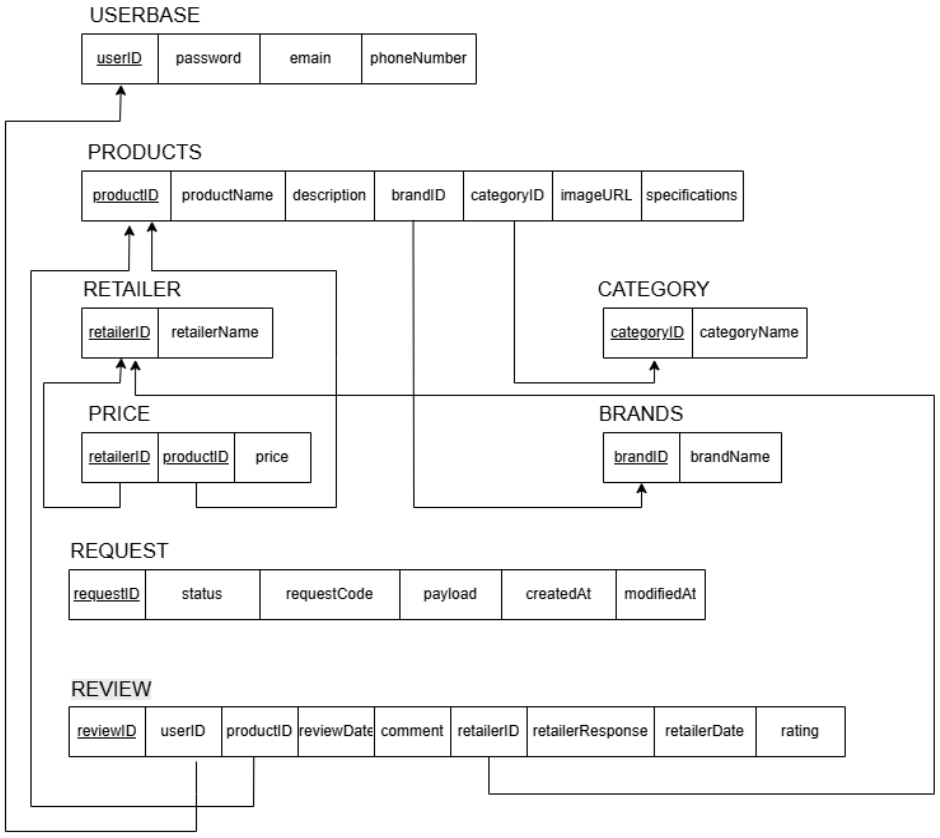
REQUEST

requestID	status	requestCode	payload	createdAt	modifiedAt

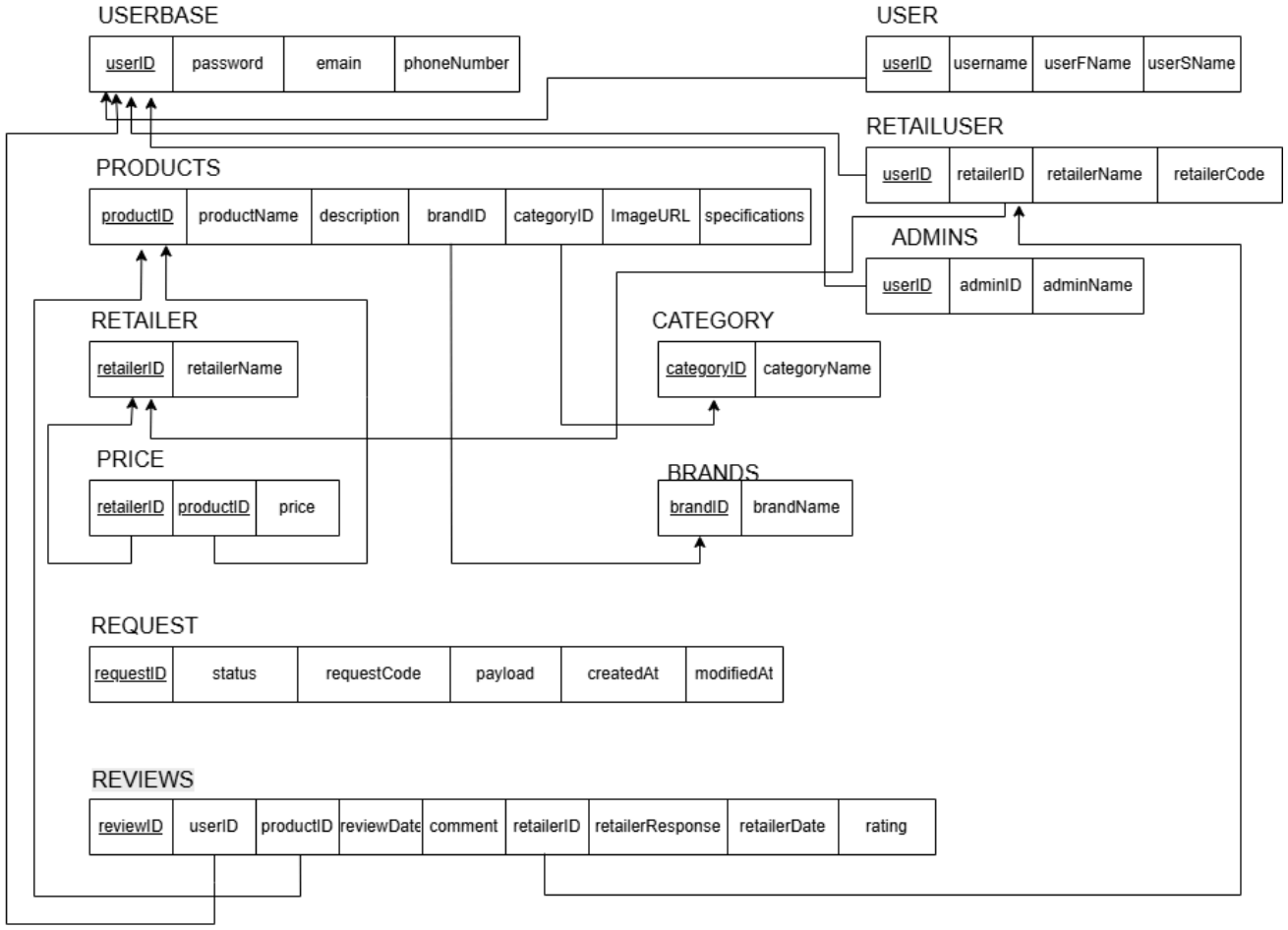
REVIEW

reviewID	userID	productID	reviewDate	comment	retailerID	retailerResponse	retailerDate	rating

Step 4



Step 8-final



Assumptions and seasonings:
The main design concept we stuck to was to separate the price from the product, the main reasoning behind this was a want for less redundancy within the tables if we left the price inside the products table there would be massive duplication every single product would have 4 copies of itself with a different price and a different retailerID, instead we chose to map the pricing to the productid and the retailerid so there is 240 prices in the prices table and 160 products(not all products we given prices due to time restraints and 240 seemed sufficient) which contains 480 cells of redundancy opposed to the 1 680 cells of redundant data if we used just a product table
The other main design concept we went for was using specialization instead of separate classes this was for some redundancy but it was mainly for referential integrity by having a userbase we were able to login and register everyone smoothly without running the risk of duplicate userid's which could allow a normal user accidental access to admin powers.
The main assumption that we used was the fact that only four retailers would be used for the purposes of demoing, it can be expanded upon but we felt it would be best to limit the pool so the products data would be more generalised
We made an assumption that all admins has an email pattern of admin@compareit.name@gmail.com and the retailers used an email pattern of retailer.name@gmail.com

TASK 4 - RELATIONAL SCHEMA

CREATING THE DATABASE

```
| compareit_binarybandits | CREATE DATABASE `compareit_binarybandits` /*!40100 DEFA
ULT CHARACTER SET utf8mb4 COLLATE utf8mb4_uca1400_ai_ci */ |
```

CREATING TABLES

```
MariaDB [compareit_binarybandits]> show tables ;
+-----------------------------------+
| Tables_in_compareit_binarybandits |
+-----------------------------------+
| admins                            |
| brands                            |
| categories                        |
| prices                            |
| products                          |
| requests                          |
| retailers                         |
| retailerusers                     |
| reviews                           |
| user                              |
| userbase                          |
+-----------------------------------+
11 rows in set (0.034 sec)
```

admins TABLE

```
| admins | CREATE TABLE `admins` (
  `userID` int(11) NOT NULL,
  `adminID` int(11) NOT NULL AUTO_INCREMENT,
  `adminName` varchar(255) NOT NULL,
  PRIMARY KEY (`adminID`),
  UNIQUE KEY `userID` (`adminID`),
  KEY `fk_admin_userbase` (`userID`),
  CONSTRAINT `fk_admin_userbase` FOREIGN KEY (`userID`) REFERENCES `userbase` (`use
rID`)
) ENGINE=InnoDB AUTO_INCREMENT=4302 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400
_ai_ci |
```

brands TABLE

```
| brands | CREATE TABLE `brands` (
  `brandID` int(11) NOT NULL AUTO_INCREMENT,
  `brandName` varchar(255) NOT NULL,
  PRIMARY KEY (`brandID`),
  UNIQUE KEY `brandName` (`brandName`)
) ENGINE=InnoDB AUTO_INCREMENT=129 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_
ai_ci |
```

categories TABLE

```
| categories | CREATE TABLE `categories` (
  `categoryID` int(11) NOT NULL AUTO_INCREMENT,
  `categoryName` varchar(100) NOT NULL,
  PRIMARY KEY (`categoryID`),
  UNIQUE KEY `categoryName` (`categoryName`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_ai
_ci |
```

prices TABLE

```
| prices | CREATE TABLE `prices` (
  `retailerID` int(11) NOT NULL,
  `productID` int(11) NOT NULL,
  `prices` decimal(65,2) NOT NULL,
  KEY `fk_prices_retailers` (`retailerID`),
  KEY `fk_prices_products` (`productID`),
  CONSTRAINT `fk_prices_products` FOREIGN KEY (`productID`) REFERENCES `products` (
`productID`),
  CONSTRAINT `fk_prices_retailers` FOREIGN KEY (`retailerID`) REFERENCES `retailers
` (`retailerID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_ai_ci |
```

products TABLE

```
| products | CREATE TABLE `products` (
  `productID` int(11) NOT NULL AUTO_INCREMENT,
  `productName` varchar(255) NOT NULL,
  `description` text NOT NULL,
  `brandID` int(11) NOT NULL,
  `categoryID` int(11) NOT NULL,
  `imageURL` varchar(255) NOT NULL,
  `specifications` text NOT NULL,
  PRIMARY KEY (`productID`),
  KEY `fk_products_brand` (`brandID`),
  KEY `fk_products_categories` (`categoryID`),
  CONSTRAINT `fk_products_brand` FOREIGN KEY (`brandID`) REFERENCES `brands` (`bran
dID`),
  CONSTRAINT `fk_products_categories` FOREIGN KEY (`categoryID`) REFERENCES `catego
ries` (`categoryID`)
) ENGINE=InnoDB AUTO_INCREMENT=160 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_
ai_ci |
```

requests TABLE

```
| requests | CREATE TABLE `requests` (
  `requestID` int(11) NOT NULL AUTO_INCREMENT,
  `status` varchar(55) DEFAULT 'pending',
  `requestCode` varchar(25) NOT NULL,
  `createdAt` datetime NOT NULL DEFAULT current_timestamp(),
  `modifiedAt` datetime NOT NULL DEFAULT current_timestamp(),
  `payload` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4_bin DEFAULT NULL CHECK (
json_valid(`payload`)),
  PRIMARY KEY (`requestID`)
) ENGINE=InnoDB AUTO_INCREMENT=57 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_a
i_ci |
```

reviews TABLE

```
---------------------+
| reviews | CREATE TABLE `reviews` (
  `reviewID` int(11) NOT NULL AUTO_INCREMENT,
  `productID` int(11) NOT NULL,
  `userID` int(11) NOT NULL,
  `rating` int(6) NOT NULL DEFAULT 0,
  `comment` text DEFAULT NULL,
  `response` varchar(255) DEFAULT NULL,
  `createdAt` datetime NOT NULL DEFAULT current_timestamp(),
  `retailerID` int(11) DEFAULT NULL,
  `responseDate` datetime DEFAULT NULL,
  PRIMARY KEY (`reviewID`),
  KEY `fk_reviews_retailers` (`retailerID`),
  KEY `fk_reviews_users` (`userID`),
  KEY `fk_reviews_products` (`productID`),
  CONSTRAINT `fk_reviews_products` FOREIGN KEY (`productID`) REFERENCES `products`
(`productID`) ON DELETE CASCADE,
  CONSTRAINT `fk_reviews_retailers` FOREIGN KEY (`retailerID`) REFERENCES `retailer
s` (`retailerID`),
  CONSTRAINT `fk_reviews_users` FOREIGN KEY (`userID`) REFERENCES `user` (`userID`)
) ENGINE=InnoDB AUTO_INCREMENT=51 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_a
i_ci |
+---------+-------------------------------------------------------------------------
```

retailers TABLE

```
-----------------------------------------+
| retailers | CREATE TABLE `retailers` (
  `retailerID` int(11) NOT NULL AUTO_INCREMENT,
  `retailerName` varchar(100) NOT NULL,
  PRIMARY KEY (`retailerID`),
  UNIQUE KEY `retailerName` (`retailerName`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_a
i_ci |
+-----------+-----------------------------------------------------------------------
```

userbase TABLE

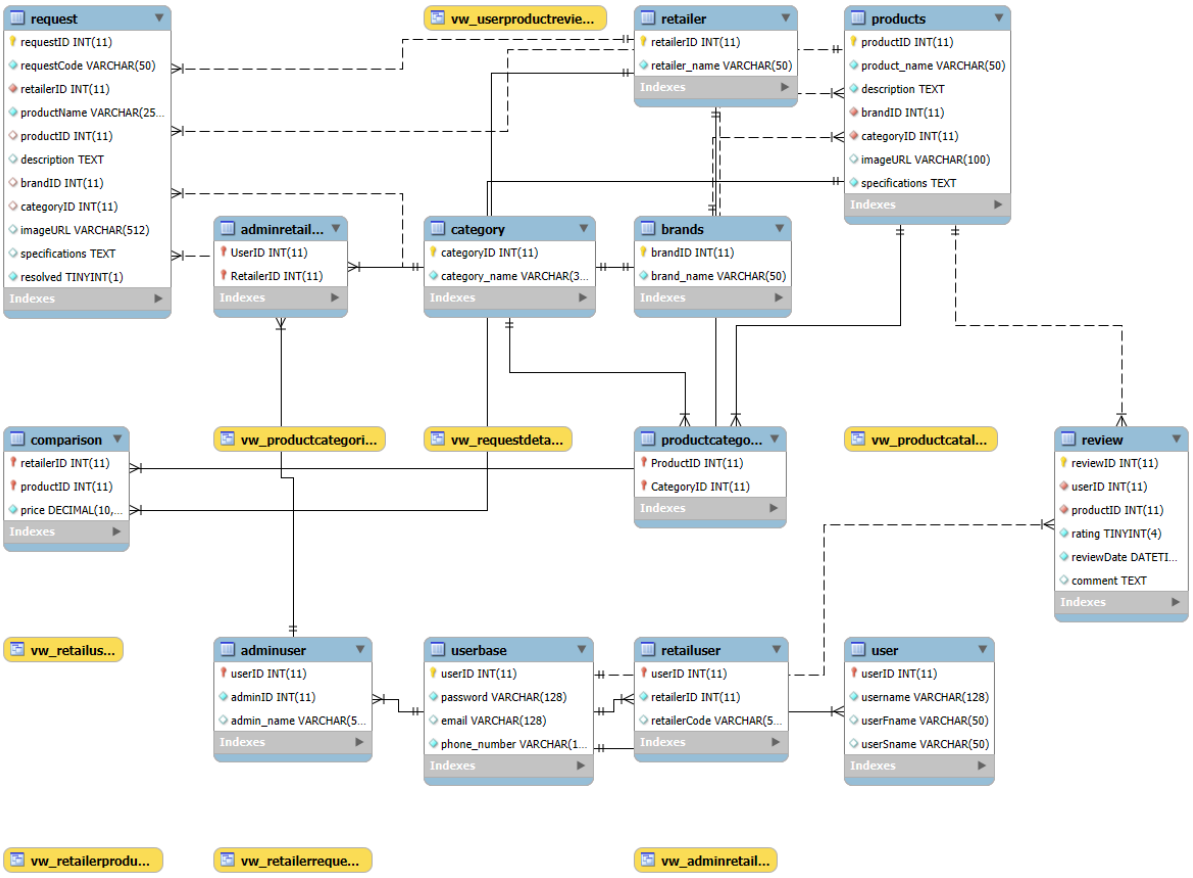
```
-----------------------------------+
| userbase | CREATE TABLE `userbase` (
  `userID` int(11) NOT NULL AUTO_INCREMENT,
  `password` varchar(255) NOT NULL,
  `email` varchar(255) DEFAULT NULL,
  `phoneNumber` varchar(10) NOT NULL,
  `apiKey` varchar(255) NOT NULL,
  PRIMARY KEY (`userID`),
  UNIQUE KEY `phoneNumber` (`phoneNumber`),
  UNIQUE KEY `apiKey` (`apiKey`),
  UNIQUE KEY `email` (`email`)
) ENGINE=InnoDB AUTO_INCREMENT=301 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_
ai_ci |
+----------+--------------------------------------------------------------------
```

user TABLE

```
--------------------------------------+
| user  | CREATE TABLE `user` (
  `userID` int(11) NOT NULL,
  `username` varchar(255) NOT NULL,
  `userFName` varchar(255) NOT NULL,
  `userSName` varchar(10) NOT NULL,
  PRIMARY KEY (`userID`),
  CONSTRAINT `fk_user_userbase` FOREIGN KEY (`userID`) REFERENCES `userbase` (`user
ID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_ai_ci |
+-------+-----------------------------------------------------------------------
```

retaileruser TABLE

```
-------------------------------------------------------------------------+
| retailerusers | CREATE TABLE `retailerusers` (
  `userID` int(11) NOT NULL,
  `retailerID` int(11) NOT NULL,
  `retailerName` varchar(100) NOT NULL,
  `retailerCode` varchar(255) NOT NULL,
  PRIMARY KEY (`userID`),
  UNIQUE KEY `retailerName` (`retailerName`),
  UNIQUE KEY `retailerCode` (`retailerCode`),
  KEY `fk_retailerusers_retailers` (`retailerID`),
  CONSTRAINT `fk_retailerUsers_userbase` FOREIGN KEY (`userID`) REFERENCES `userbas
e` (`userID`) ON DELETE CASCADE,
  CONSTRAINT `fk_retailerusers_retailers` FOREIGN KEY (`retailerID`) REFERENCES `re
tailers` (`retailerID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_ai_ci |
+---------------+-------------------------------------------------------------------------
```



TASK 5 - WEB-BASED APPLICATION

API Endpoints:

- Register, Login and Logout
    - Register
    - Login
    - Logout
- Products
    - GetAllProducts
    - GetAllRetailerProducts
- Reviews
    - GetAllReviewsAndResponses
    - GetAllUserReviews
    - Add Review
    - Update Review
    - Delete Review
    - Response To Review
    - Delete Response
    - Product Reviews
- Requests
    - GetAllRequests
    - GetAllRetailerRequests
    - Approve Add Request
    - Allow Update Request
    - Approve Delete Request
    - Decline Request
    - Add Product Request
    - Update Product Request
    - Delete Product Request
    - Edit Request
    - Delete Request

TASK 6 - DATA

The data population method we used was doing it by hand with the aid of DeepSeek ai generation, though the users and admins are completely from imagination, the people do not exist. The retailers were created by hand as there were only 4 of them, we only used 4 as we planned to have a lot of data in products. The rest of the data including products, reviews ,requests,brands,prices and categories were all created with AI using very guided commands, the brands and categories were based off real brands in south african grocery stores by constantly generating the data whilst introducing links, pictures, old data and adding context date such as discussing trends in the pricing like using checkers prices as the competitive price and telling the ai that for example woolworths is know for prestige price and providing examples of common price differences between things like bread and more “luxury products” such as rusks and chips which are not necessity. To help the ai stay as close to reality as possible, some product pricing being added by hand to help control the fluctuation of pricing.

The same was done for products and categories where the products were based off the brands generated as those brands would be in grocery stores therefore the products from those brands would be in grocery stores . Then we generated the data for the reviews and requests based on the data in products giving the ai model restrictions based on what data was in products such that no review or request was made of a fake or unreachable product in the database. Though the ai model could not generate images for the real items so that was done by hand to ensure the images were of good quality and to ensure the products and brands created were real.

The reason we decided to do it this way instead of data scraping or creating or taking from existing apis was because we wanted the data to be as close to real life as possible and the only apis,json repositories and any reliable data I could find was quite old and the pricing was not current and with pricing being the make comparing agent and purpose of this project it seemed better to take real data via scraping but the attempts to create a program to scrap was not working so we had to pivot there was not enough time to fix the program,gain the content, change the content into a useful information and add it to the table, it was just not feasible in the time frame we had left.

TASK 7 - ANALYSE AND OPTIMISE

All Products

From handleGetAllProducts() function, we have the following sql:
SELECT p.productID, p.productName, p.description, p.imageURL, p.specifications, c.categoryName, b.brandName, MIN(pr.prices) AS lowestPrice FROM products p LEFT JOIN categories c ON p.categoryID = c.categoryID LEFT JOIN brands b ON p.brandID = b.brandID LEFT JOIN prices pr ON p.productID = pr.productID WHERE c.categoryName = "Electronics" GROUP BY p.productID ORDER BY lowestPrice LIMIT 20;

```sql
SELECT p.productID, p.productName, p.description, p.imageURL, p.specifications, c.categoryName, b.brandName, MIN(pr.prices) AS
lowestPrice
FROM products p
LEFT JOIN categories c ON p.categoryID = c.categoryID
LEFT JOIN brands b ON p.brandID = b.brandID
LEFT JOIN prices pr ON p.productID = pr.productID
WHERE c.categoryName = "Electronics"
GROUP BY p.productID
ORDER BY lowestPrice
LIMIT 20;
```

Showing rows 0 - 6 (7 total, Query took 0.0212 seconds.)

```sql
SELECT p.productID, p.productName, p.description, p.imageURL, p.specifications, c.categoryName, b.brandName, MIN(pr.prices) AS lowestPrice FROM products p LEFT JOIN categories
c ON p.categoryID = c.categoryID LEFT JOIN brands b ON p.brandID = b.brandID LEFT JOIN prices pr ON p.productID = pr.productID WHERE c.categoryName = "Electronics" GROUP BY
p.productID ORDER BY lowestPrice LIMIT 20;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

productID	productName	description	imageURL	specifications	categoryName	brandName	lowestPrice ▲ 1
70	Sparletta Iron Brew	2L bottle	/groceries/sparlettaironbrew.jpg	{"volume":"2L","flavor":"iron brew","sugar":"high"...	Electronics	Sparletta	NULL
120	Sparletta Granadilla	2L bottle	/groceries/sparlettagranedilla.jpg	{"volume":"2L","flavor":"granadilla","sugar":"high...	Electronics	Sparletta	NULL
145	Sparletta Cola	2L bottle	/groceries/sparlettacola.jpg	{"volume":"2L","flavor":"cola","sugar":"high"}	Electronics	Sparletta	NULL
21	Nestlé Ricoffy	500g instant coffee	/groceries/recoffe.jpg	{"weight":"500g","type":"instant","cups":"100"}	Electronics	Ricoffy	18.99
29	Weet-Bix	1.5kg whole wheat biscuits	/groceries/weetbix.jpg	{"weight":"1.5kg","type":"whole wheat","servings":...	Electronics	Weet-Bix	22.99
41	Lucky Star Tuna	170g can in brine	/groceries/luckystartuna.jpg	{"weight":"170g","type":"tuna","servings":"1"}	Electronics	Lucky Star	24.99
24	Sparletta Cream Soda	2L bottle	/groceries/sparlettacreamsoda.jpg	{"volume":"2L","flavor":"cream soda","sugar":"high...	Electronics	Sparletta	25.99

<u>Output before optimization:</u>

- Join Type for most tables: LEFT
- MIN(pr.prices) creates a temporary column table view
- Grouping by categoryName in example or “Filter or sort” from website
- Ran for 0.0212 seconds

Optimized SQL:

```sql
CREATE INDEX idx_category_name ON categories(categoryName);
CREATE INDEX idx_product_category ON products(categoryID);
CREATE INDEX idx_product_brand ON products(brandID);
CREATE INDEX idx_price_product ON prices(productID, prices);
SELECT p.productID, p.productName, p.description, p.imageURL, p.specifications,
c.categoryName, b.brandName, MIN(pr.prices) AS lowestPrice
FROM products p
JOIN categories c ON p.categoryID = c.categoryID
JOIN brands b ON p.brandID = b.brandID
LEFT JOIN prices pr ON p.productID = pr.productID
WHERE c.categoryName = 'Electronics'
GROUP BY p.productID, p.productName, p.description, p.imageURL, p.specifications,
c.categoryName, b.brandName
ORDER BY lowestPrice ASC
```

LIMIT 20;

```sql
CREATE INDEX idx_category_name ON categories(categoryName);
CREATE INDEX idx_product_category ON products(categoryID);
CREATE INDEX idx_product_brand ON products(brandID);
CREATE INDEX idx_price_product ON prices(productID, prices);
SELECT p.productID, p.productName, p.description, p.imageURL, p.specifications, c.categoryName, b.brandName, MIN(pr.prices) AS
lowestPrice
FROM products p
JOIN categories c ON p.categoryID = c.categoryID
JOIN brands b ON p.brandID = b.brandID
LEFT JOIN prices pr ON p.productID = pr.productID
WHERE c.categoryName = 'Electronics'
GROUP BY p.productID, p.productName, p.description, p.imageURL, p.specifications, c.categoryName, b.brandName
ORDER BY lowestPrice ASC
LIMIT 20;
```

Output after optimization:

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0185 seconds.)

```sql
CREATE INDEX idx_category_name ON categories(categoryName);
```

[Edit inline] [Edit] [Create PHP code]

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0163 seconds.)

```sql
CREATE INDEX idx_product_category ON products(categoryID);
```

[Edit inline] [Edit] [Create PHP code]

Note: #1831 Duplicate index `idx_product_category`. This is deprecated and will be disallowed in a future release

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0113 seconds.)

```sql
CREATE INDEX idx_product_brand ON products(brandID);
```

[Edit inline] [Edit] [Create PHP code]

Note: #1831 Duplicate index `idx_product_brand`. This is deprecated and will be disallowed in a future release

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0083 seconds.)

```sql
CREATE INDEX idx_price_product ON prices(productID, prices);
```

[Edit inline] [Edit] [Create PHP code]

Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

Showing rows 0 - 6 (7 total, Query took 0.0184 seconds.)

```sql
SELECT p.productID, p.productName, p.description, p.imageURL, p.specifications, c.categoryName, b.brandName, MIN(pr.prices) AS lowestPrice FROM products p JOIN categories c
ON p.categoryID = c.categoryID JOIN brands b ON p.brandID = b.brandID LEFT JOIN prices pr ON p.productID = pr.productID WHERE c.categoryName = 'Electronics' GROUP BY
p.productID, p.productName, p.description, p.imageURL, p.specifications, c.categoryName, b.brandName ORDER BY lowestPrice ASC LIMIT 20;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

productID	productName	description	imageURL	specifications	categoryName	brandName	lowestPrice ▲ 1
70	Sparletta Iron Brew	2L bottle	/groceries/sparlettaironbrew.jpg	{"volume":"2L","flavor":"iron brew","sugar":"high"...	Electronics	Sparletta	NULL
120	Sparletta Granadilla	2L bottle	/groceries/sparlettagranedilla.jpg	{"volume":"2L","flavor":"granadilla","sugar":"high...	Electronics	Sparletta	NULL
145	Sparletta Cola	2L bottle	/groceries/sparlettacola.jpg	{"volume":"2L","flavor":"cola","sugar":"high"}	Electronics	Sparletta	NULL
21	Nestlé Ricoffy	500g instant coffee	/groceries/recoffe.jpg	{"weight":"500g","type":"instant","cups":"100"}	Electronics	Ricoffy	18.99
29	Weet-Bix	1.5kg whole wheat biscuits	/groceries/weetbix.jpg	{"weight":"1.5kg","type":"whole wheat","servings":...	Electronics	Weet-Bix	22.99
41	Lucky Star Tuna	170g can in brine	/groceries/luckystartuna.jpg	{"weight":"170g","type":"tuna","servings":"1"}	Electronics	Lucky Star	24.99
24	Sparletta Cream Soda	2L bottle	/groceries/sparlettacreamsoda.jpg	{"volume":"2L","flavor":"cream soda","sugar":"high...	Electronics	Sparletta	25.99

- Index created to be able to find a connecting property between tables faster
- Join Type for most tables: Normal JOIN
    - Since you're filtering on c.categoryName = 'Electronics', a LEFT JOIN makes no sense — if c is NULL, the WHERE clause filters it out anyway. A regular JOIN improves performance.
- Expanded GROUP BY to include all selected non-aggregate columns:
    - MySQL/MariaDB allows non-aggregated columns not in the GROUP BY, but it's inefficient and may produce unpredictable results. Expanding the GROUP BY helps the optimizer.

- Therefore sql ran for 0.0184 which is faster than the previous sql

TASK 8 - DEVELOPMENT

Link to Github: https://github.com/u20445564/COS221-Assignment