COS 221 Assignment 5

Team: Guardians of the Database

Team members: Kgaugelo Matsena (u23658462), Lerato Sibanda (u22705504), Neo Machaba (u23002167), Kiarin Naido (u23600897), Nyashadzashe Makwabarara (u22783386), Cleopatra Kwenda (u23547121- Group Leader)

GitHub Repo: https://github.com/Cleopatra-K/Gaurdians-of-the-Database

Task 1: Research

Research on the Retail and E-Commerce Industry in South Africa

General Overview and Explanation of E-commerce in South Africa

E-commerce in South Africa has witnessed rapid growth, particularly after the COVID-19 pandemic which accelerated digital implementation. In 2023, the South African online retail sector generated over R71 billion in revenue, with expectations of continued double-digit growth [1]. Despite this growth, challenges such as high data costs, delivery inefficiencies, and consumer trust issues still impact the user experience. As consumers seek greater transparency and value, price comparison tools have become increasingly relevant. Platforms such as Takealot, Makro, and Checkers Sixty60 are popular among South Africans, while early comparison platforms like PriceCheck and Hippo.co.za struggle with outdated data and limited retailer partnerships [2].

Categories of Products and Types of Retailers

South Africa's e-commerce space is dominated by popular product categories such as consumer electronics (e.g., smartphones, laptops, and TVs), groceries, fashion, and home improvement. Leading platforms for these include Takealot, Superbalist, Checkers Sixty60, and Makro. Retailers can be broadly grouped into pure-play online stores like Takealot, hybrid brick-and-click outlets like Pick n Pay and Woolworths, and online marketplaces such as Takealot Marketplace where multiple third-party vendors list their offerings.

Content Rating and Categorization Systems

Platforms such as Takealot use a 1-to-5 star rating system and written reviews from verified users to establish product credibility and trust. Products are organized by category, brand, price, and user popularity, with filtering tools available to refine searches. This approach enhances user experience and assists in decision-making during purchases.

Additional Platform Features

Modern South African e-commerce platforms now include several user-focused features like live stock updates and estimated delivery times, BNPL (Buy Now, Pay Later) options via Payflex or Mobicred, visual user reviews (with photos), promotions and seasonal discounts. Some platforms offer personalized product suggestions based on past browsing behaviour. However, the lack of comprehensive cross-store comparison features presents an opportunity for our product to innovate.

References

[1] World Wide Worx, “Online Retail in South Africa 2023,” Jan. 2024. [Online]. Available: https://www.worldwideworx.com/online-retail-2023/

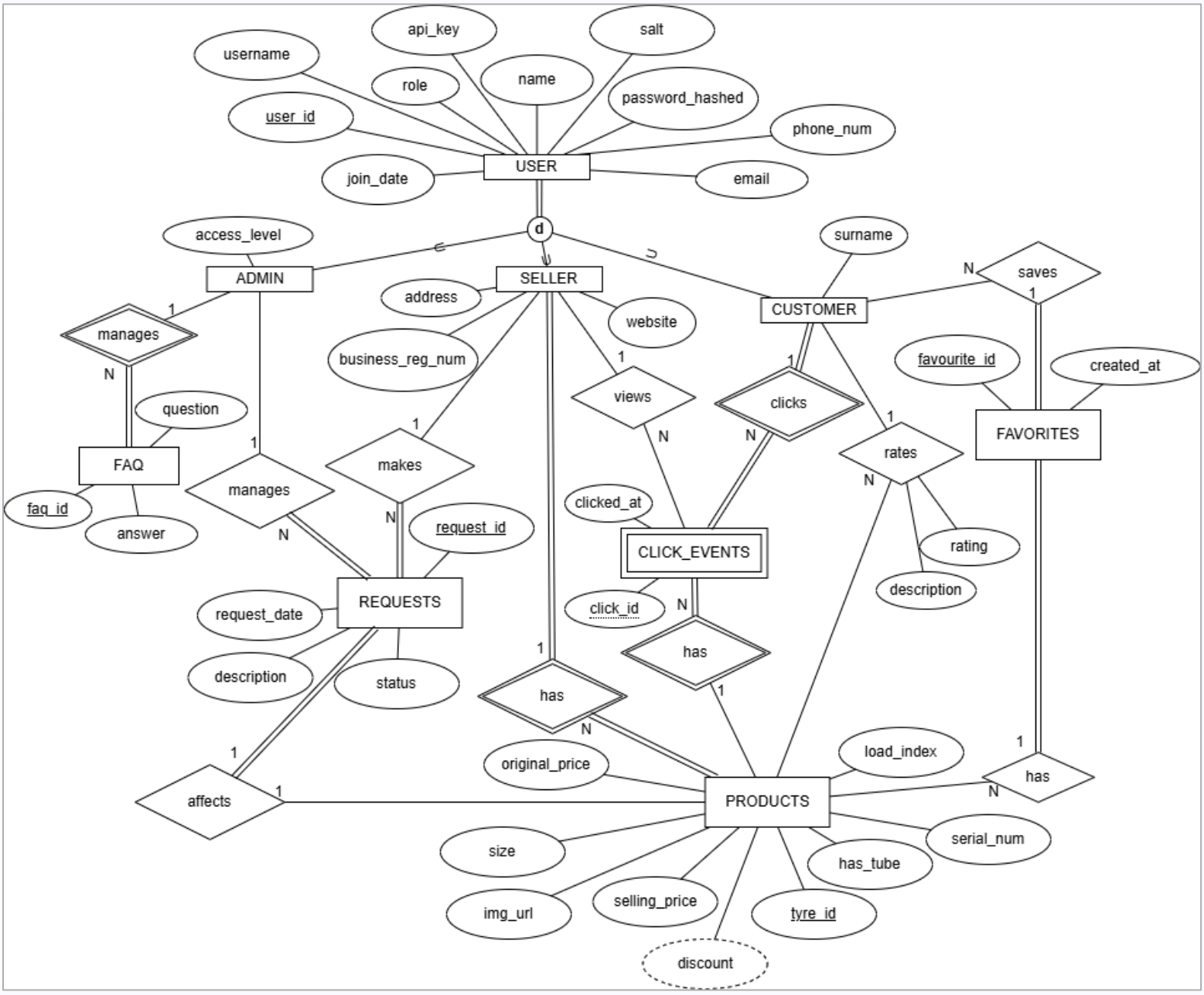
[2] A. Mofokeng, “Why South African price comparison tools haven’t scaled,” BusinessTech, Aug. 2022. [Online]. Available: https://businesstech.co.za/news/internet/618759/

[3] S. Nkosi, “E-commerce Trends in South Africa,” ITWeb, Mar. 2023. [Online]. Available: https://www.itweb.co.za/article/e-commerce-trends-2023

[4] N. K. Dube and T. Motau, “Consumer Trust in South African Online Retailers,” in Proc. IEEE Conf. ICT in Africa, pp. 77–83, 2023.

[5] A. de Wet, “Comparing Groceries Online in SA,” MyBroadband, Nov. 2023. [Online]. Available: https://mybroadband.co.za/news/shopping/507362/

Task 2: (E)ER Diagram

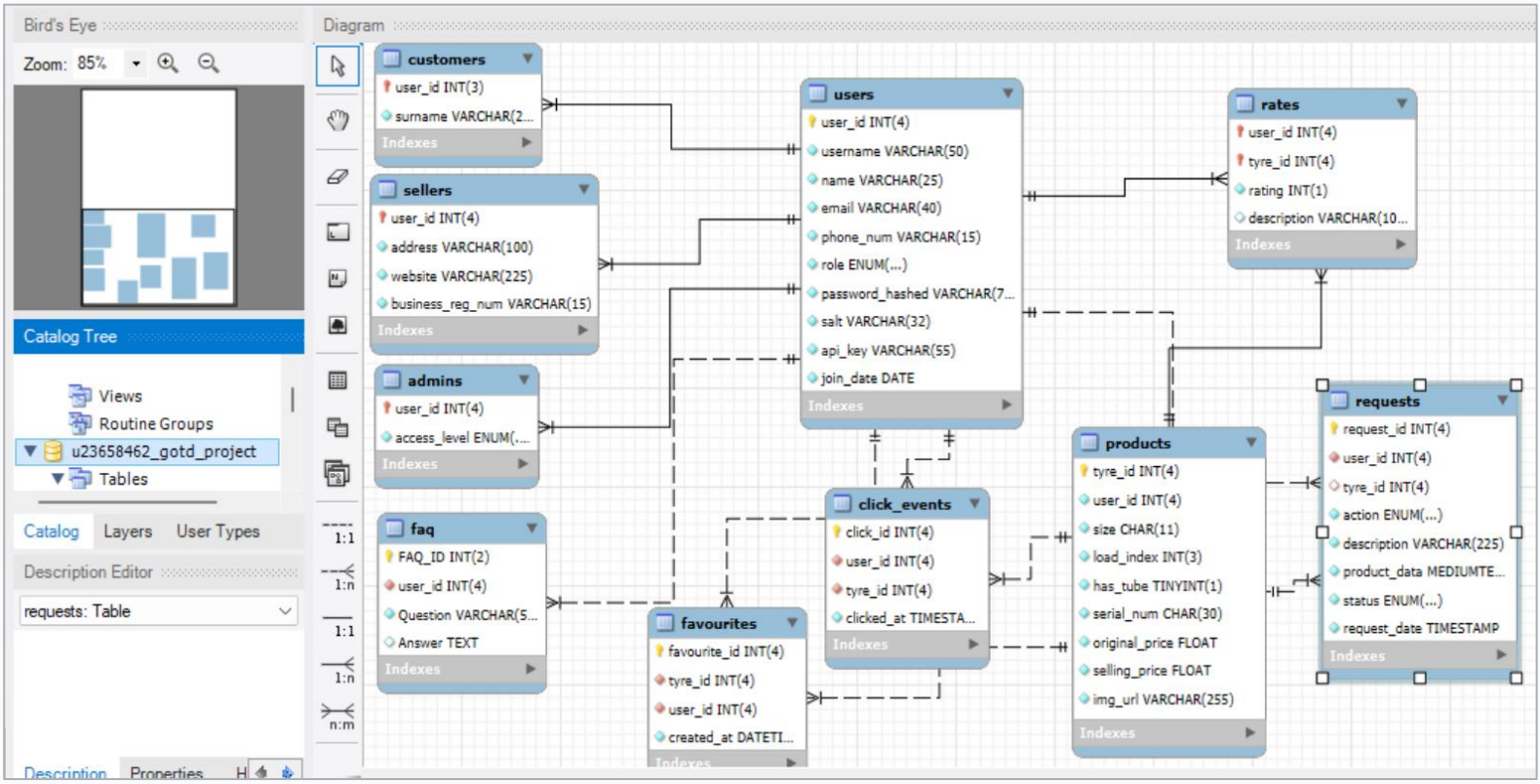


Task 3: (E)ER Diagram to Relational Mapping

(Relational Mapping PDF is attached with this file)

Task 4: Relational Schema

Relation Schema



SQL statements

1. products Table

```
CREATE TABLE `products` (
  `tyre_id` int(4) NOT NULL,
  `user_id` int(4) NOT NULL,
  `size` char(11) NOT NULL,
  `load_index` int(3) NOT NULL,
  `has_tube` tinyint(1) NOT NULL,
  `serial_num` char(30) NOT NULL,
  `original_price` float NOT NULL,
  `selling_price` float NOT NULL,
  `img_url` varchar(255) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `products`
  ADD PRIMARY KEY (`tyre_id`);
```

** this primary key constraint ensures no two or more tyres can have the same tyre_id, enforces entity integrity

```
ALTER TABLE `products`
  MODIFY `tyre_id` int(4) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=491;
```
** this NOT NULL constraint ensures that the primary key cannot be NULL and enforces data integrity.

2. users table

```
CREATE TABLE `users` (
  `user_id` int(4) NOT NULL,
  `username` varchar(50) NOT NULL,
  `name` varchar(25) NOT NULL,
  `email` varchar(40) NOT NULL,
  `phone_num` varchar(15) NOT NULL,
  `role` enum('Seller','Customer','Admin') NOT NULL,
  `password_hashed` varchar(70) NOT NULL,
  `salt` varchar(32) NOT NULL,
  `api_key` varchar(55) NOT NULL,
  `join_date` date NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;


ALTER TABLE `users`
  ADD PRIMARY KEY (`user_id`);

ALTER TABLE `users`
  MODIFY `user_id` int(4) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=47;
```

3. requests table

```
CREATE TABLE `requests` (
  `request_id` int(4) NOT NULL,
  `user_id` int(4) NOT NULL,
  `tyre_id` int(4) DEFAULT NULL,
  `action` enum('add','remove','update') NOT NULL,
  `description` varchar(225) NOT NULL,
  `product_data` mediumtext NOT NULL,
  `status` enum('Pending','Approved','Rejected') NOT NULL DEFAULT 'Pending',
  `request_date` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `requests`
  ADD PRIMARY KEY (`request_id`),
  ADD KEY `requests_ibfk_1` (`user_id`),
  ADD KEY `requests_ibfk_2` (`tyre_id`);
```
** the above SQL sets request_id as a foreign key, and sets indexes on user_id and tyre_id

```
ALTER TABLE `requests`
  MODIFY `request_id` int(4) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=8;
```

```
ALTER TABLE `requests`
 ADD CONSTRAINT `requests_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `requests_ibfk_2` FOREIGN KEY (`tyre_id`) REFERENCES `products`
(`tyre_id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

**the above SQL links user_id in requests to users table by setting and naming the constraint requests_ibfk_1. Links tyre_id in requests table to products table by naming a setting the constraint requests_ibfk_2. Both constraints ensure referential integrity by synchronizing changes across tables, using the ON DELETE CASCADE ON UPDATE CASCADE.

4. rates table

```
CREATE TABLE `rates` (
 `user_id` int(4) NOT NULL,
 `tyre_id` int(4) NOT NULL,
 `rating` int(1) NOT NULL,
 `description` varchar(100) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `rates`
 ADD PRIMARY KEY (`tyre_id`,`user_id`),
 ADD KEY `rates_ibfk_2` (`user_id`);

ALTER TABLE `rates`
 ADD CONSTRAINT `rates_ibfk_1` FOREIGN KEY (`tyre_id`) REFERENCES `products`
(`tyre_id`) ON DELETE CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `rates_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`)
ON DELETE CASCADE ON UPDATE CASCADE;
```

5. click_events table

```
CREATE TABLE `click_events` (
 `click_id` int(4) NOT NULL,
 `user_id` int(4) NOT NULL,
 `tyre_id` int(4) NOT NULL,
 `clicked_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `click_events`
 ADD PRIMARY KEY (`click_id`),
 ADD KEY `click_events_ibfk_1` (`user_id`),
 ADD KEY `click_events_ibfk_2` (`tyre_id`);

ALTER TABLE `click_events`
 MODIFY `click_id` int(4) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=34;
```

```
ALTER TABLE `click_events`
 ADD CONSTRAINT `click_events_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `click_events_ibfk_2` FOREIGN KEY (`tyre_id`) REFERENCES `products`
(`tyre_id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

6. favourites table

```
CREATE TABLE `favourites` (
 `favourite_id` int(4) NOT NULL,
 `tyre_id` int(4) NOT NULL,
 `user_id` int(4) NOT NULL,
 `created_at` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `favourites`
 ADD PRIMARY KEY (`favourite_id`),
 ADD UNIQUE KEY `unique_user_tyre` (`user_id`,`tyre_id`),
 ADD KEY `favourites_ibfk_1` (`tyre_id`);
```

** this unique key constraint ensures no duplicate combinations of user_id and tyre_id,so the same user cannot favourite a tyre more than once

```
ALTER TABLE `favourites`
 MODIFY `favourite_id` int(4) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=20;

ALTER TABLE `favourites`
 ADD CONSTRAINT `favourites_ibfk_1` FOREIGN KEY (`tyre_id`) REFERENCES `products`
(`tyre_id`) ON DELETE CASCADE ON UPDATE CASCADE,
 ADD CONSTRAINT `favourites_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

7. customers table

```
CREATE TABLE `customers` (
 `user_id` int(3) NOT NULL,
 `surname` varchar(20) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `customers`
 ADD PRIMARY KEY (`user_id`);

ALTER TABLE `customers`
 ADD CONSTRAINT `customers_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

8. sellers table

```
CREATE TABLE `sellers` (
 `user_id` int(4) NOT NULL,
 `address` varchar(100) NOT NULL,
```

```
  `website` varchar(225) NOT NULL,
  `business_reg_num` varchar(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

9. admins table

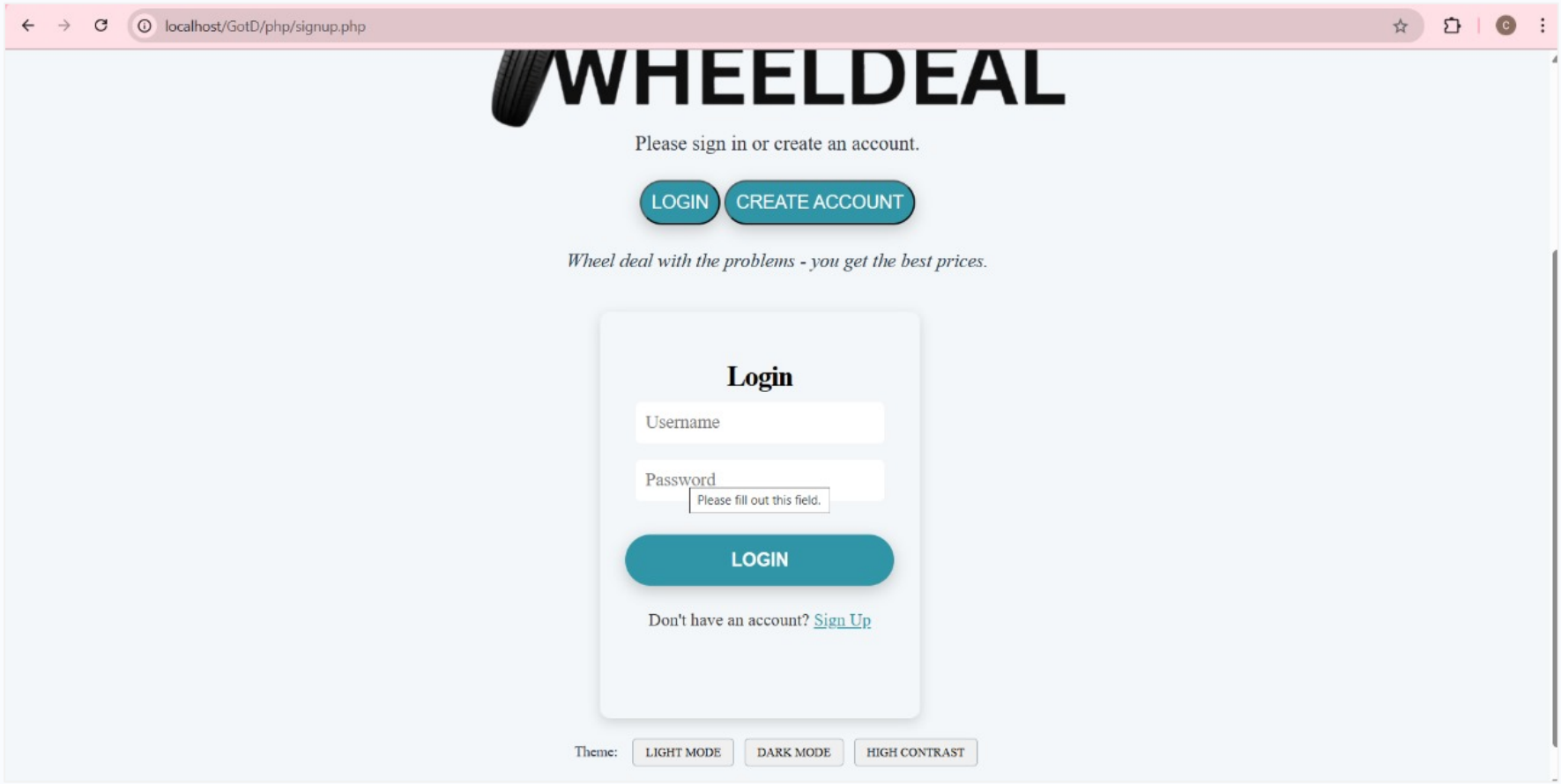
```
CREATE TABLE `admins` (
  `user_id` int(4) NOT NULL,
  `access_level` enum('Support','Maintenance','Super Admin') NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

ALTER TABLE `admins`
  ADD PRIMARY KEY (`user_id`);

ALTER TABLE `admins`
  ADD CONSTRAINT `admins_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE;
```

10. FAQ table

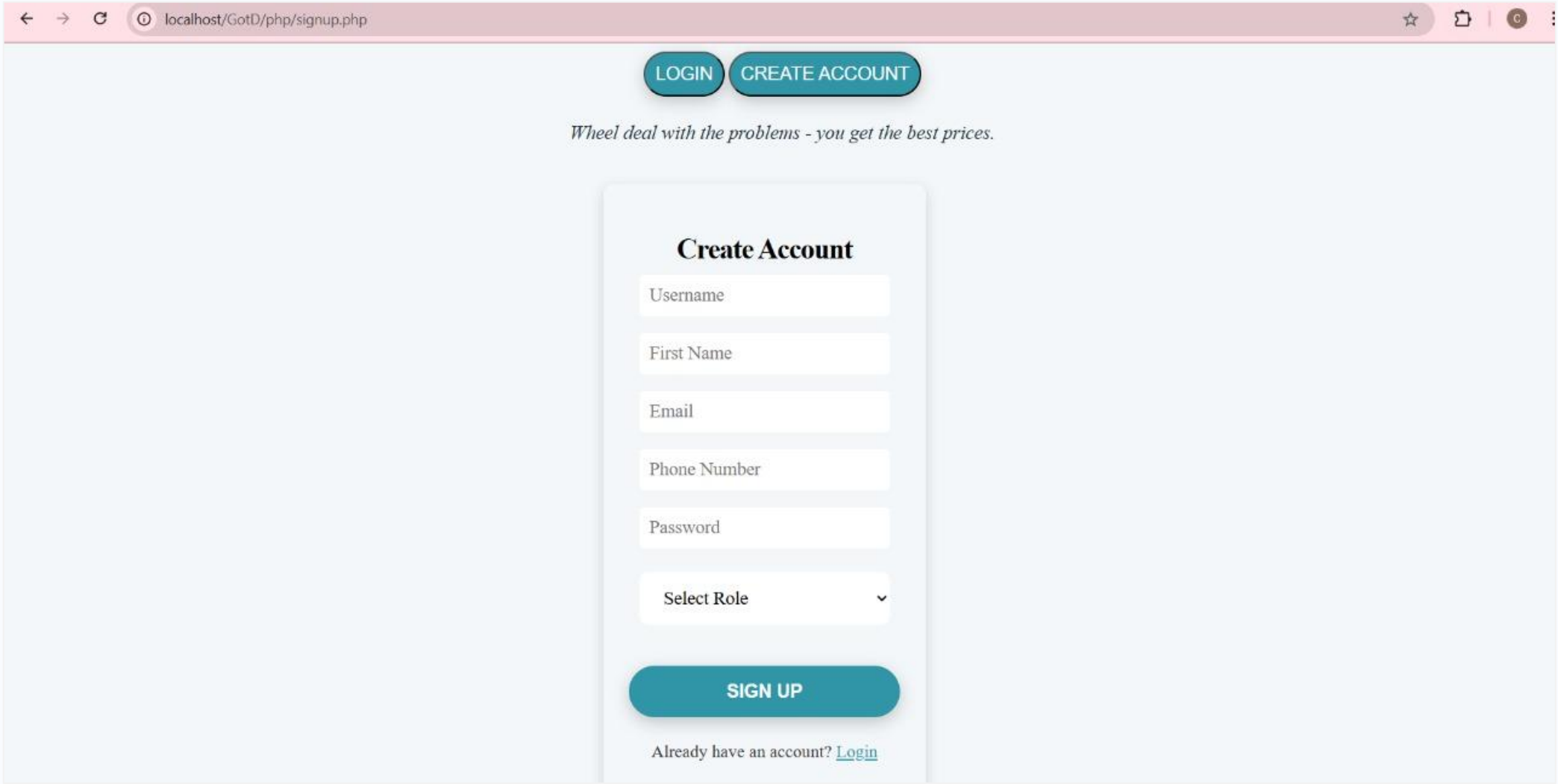
```
CREATE TABLE `FAQ` (
  `FAQ_ID` int(2) NOT NULL,
  `user_id` int(4) NOT NULL,
  `Question` varchar(50) NOT NULL,
  `Answer` text DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;


ALTER TABLE `FAQ`
  ADD PRIMARY KEY (`FAQ_ID`),
  ADD KEY `user_id` (`user_id`);

ALTER TABLE `FAQ`
  MODIFY `FAQ_ID` int(2) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=7;

ALTER TABLE `FAQ`
  ADD CONSTRAINT `FAQ_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`);
```

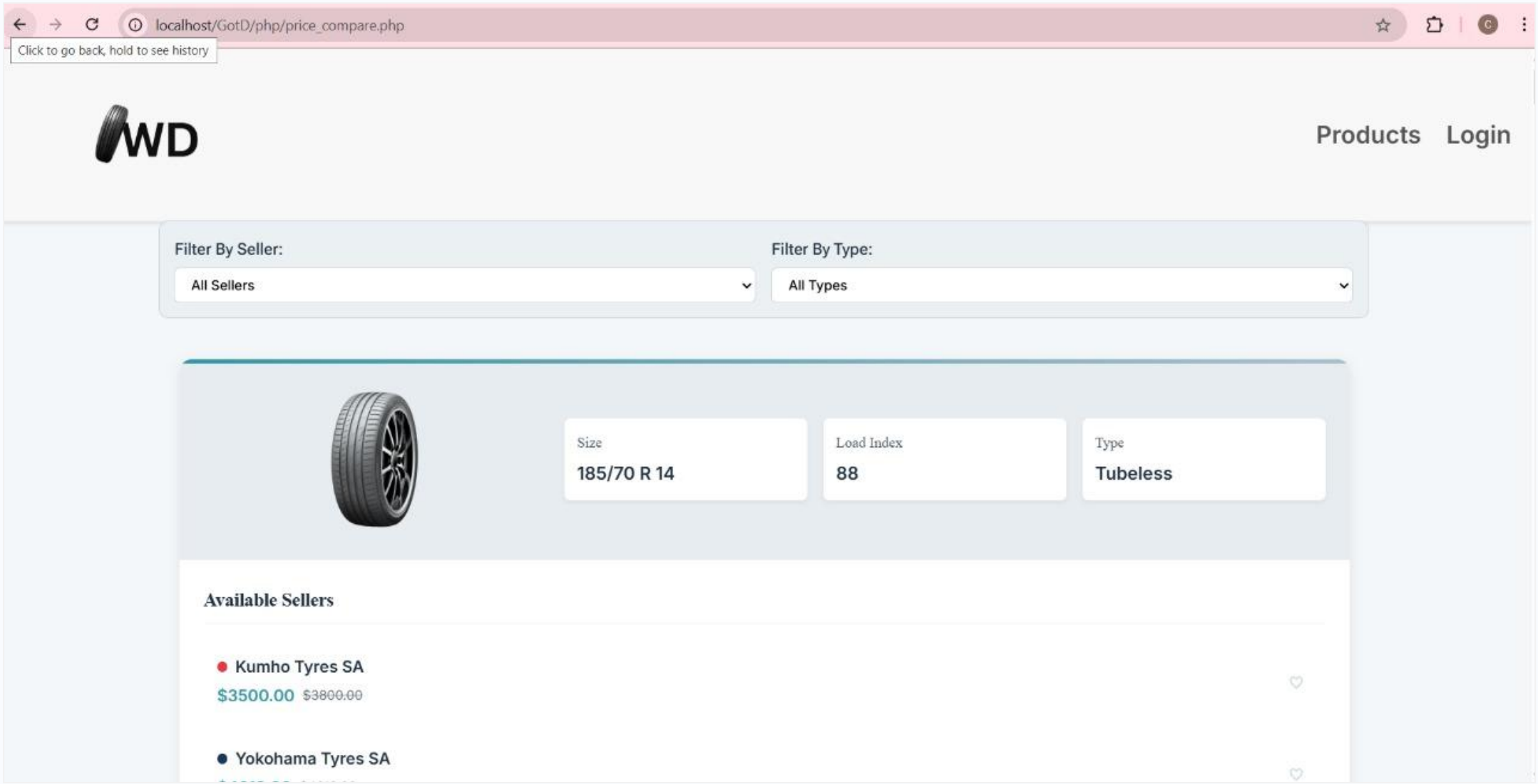
Task 5: Web-based application

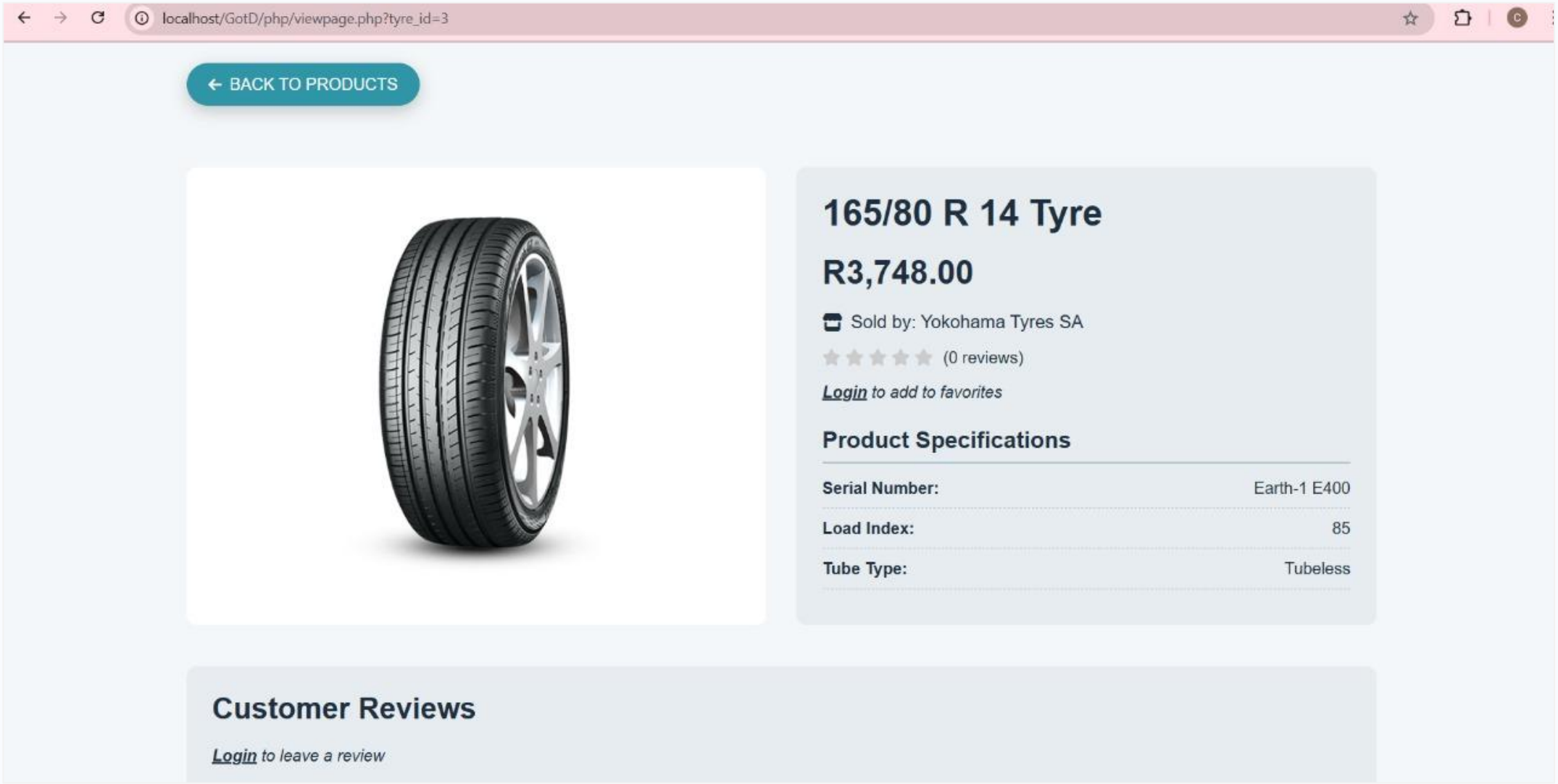
Below are some sample examples of our web-pages. With some characteristics of our minimal viable product. We will thoroughly go through our application during the demo.



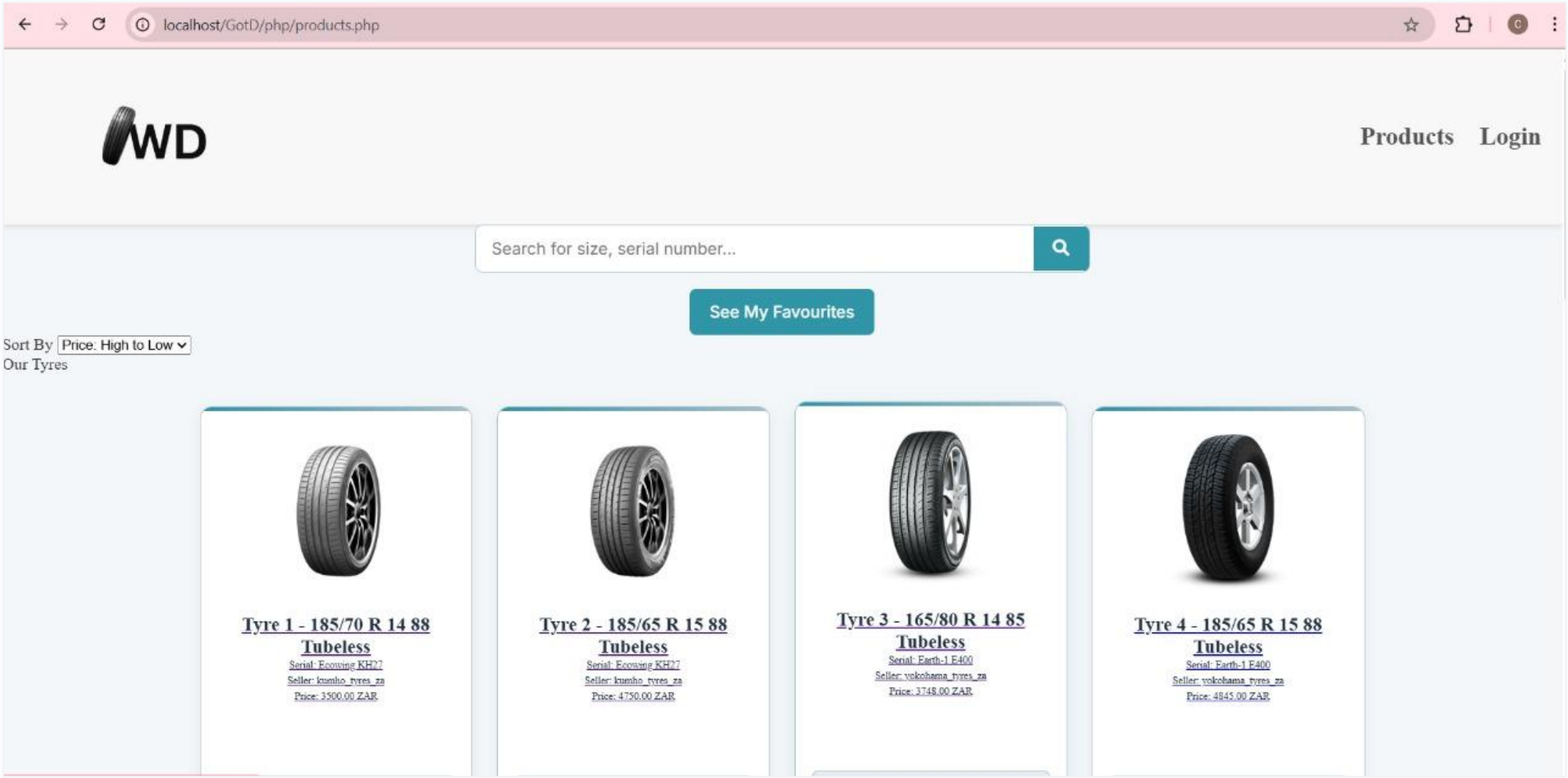
The login page above would be how we user login, and if the do not have an account, the would create an account through the sign-up page below.


localhost/GotD/php/signup.php
LOGIN
CREATE ACCOUNT
Wheel deal with the problems - you get the best prices.
Create Account
Username
First Name
Email
Phone Number
Password
Select Role
SIGN UP
Already have an account? Login


localhost/GotD/php/price_compare.php
Click to go back, hold to see history
WD
Products
Login
Filter By Seller:
All Sellers
Filter By Type:
All Types
Size
185/70 R 14
Load Index
88
Type
Tubeless
Available Sellers
Kumho Tyres SA
$3500.00
$3800.00
Yokohama Tyres SA



Above would be what the user sees when they click on a product. In order for them to leave a review they would have to be logged in.



Above would be the products page, a user would be able to see a variety of the products within our system.

Task 6: Data

How did we populate our data?

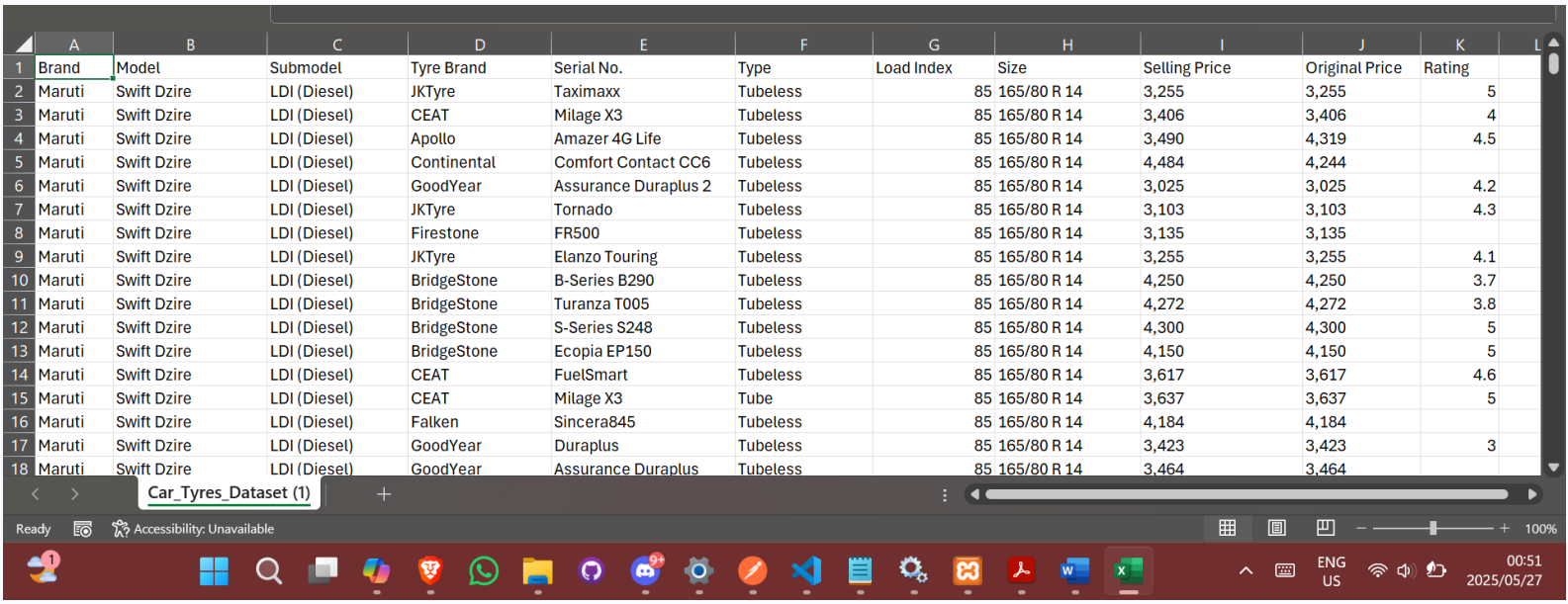
1. Products Table

We used a data set from Kaggle and downloaded a csv file.

From that csv file:

- we removed duplicates
- removed all columns that were unrelated to our products table
- we separated each Seller into separate csv files (using a filter by Tyre Brand)

Below is an example of how our initial dataset looked:



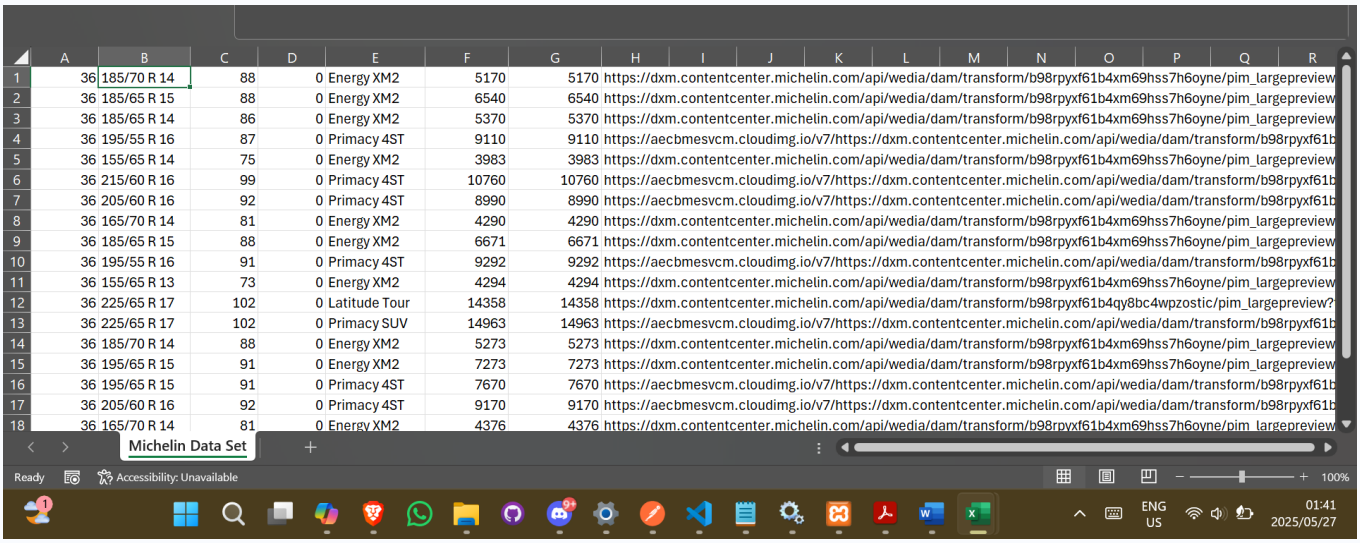
Brand	Model	Submodel	Tyre Brand	Serial No.	Type	Load Index	Size	Selling Price	Original Price	Rating
Maruti	Swift Dzire	LDI (Diesel)	JKTyre	Taximaxx	Tubeless	85	165/80 R 14	3,255	3,255	5
Maruti	Swift Dzire	LDI (Diesel)	CEAT	Milage X3	Tubeless	85	165/80 R 14	3,406	3,406	4
Maruti	Swift Dzire	LDI (Diesel)	Apollo	Amazer 4G Life	Tubeless	85	165/80 R 14	3,490	4,319	4.5
Maruti	Swift Dzire	LDI (Diesel)	Continental	Comfort Contact CC6	Tubeless	85	165/80 R 14	4,484	4,244	
Maruti	Swift Dzire	LDI (Diesel)	GoodYear	Assurance Duraplus 2	Tubeless	85	165/80 R 14	3,025	3,025	4.2
Maruti	Swift Dzire	LDI (Diesel)	JKTyre	Tornado	Tubeless	85	165/80 R 14	3,103	3,103	4.3
Maruti	Swift Dzire	LDI (Diesel)	Firestone	FR500	Tubeless	85	165/80 R 14	3,135	3,135	
Maruti	Swift Dzire	LDI (Diesel)	JKTyre	Elanzo Touring	Tubeless	85	165/80 R 14	3,255	3,255	4.1
Maruti	Swift Dzire	LDI (Diesel)	BridgeStone	B-Series B290	Tubeless	85	165/80 R 14	4,250	4,250	3.7
Maruti	Swift Dzire	LDI (Diesel)	BridgeStone	Turanza T005	Tubeless	85	165/80 R 14	4,272	4,272	3.8
Maruti	Swift Dzire	LDI (Diesel)	BridgeStone	S-Series S248	Tubeless	85	165/80 R 14	4,300	4,300	5
Maruti	Swift Dzire	LDI (Diesel)	BridgeStone	Ecopia EP150	Tubeless	85	165/80 R 14	4,150	4,150	5
Maruti	Swift Dzire	LDI (Diesel)	CEAT	FuelSmart	Tubeless	85	165/80 R 14	3,617	3,617	4.6
Maruti	Swift Dzire	LDI (Diesel)	CEAT	Milage X3	Tube	85	165/80 R 14	3,637	3,637	5
Maruti	Swift Dzire	LDI (Diesel)	Falken	Sincera845	Tubeless	85	165/80 R 14	4,184	4,184	
Maruti	Swift Dzire	LDI (Diesel)	GoodYear	Duraplus	Tubeless	85	165/80 R 14	3,423	3,423	3
Maruti	Swift Dzire	LDI (Diesel)	GoodYear	Assurance Duraplus	Tubeless	85	165/80 R 14	3,464	3,464	

When every csv file was separated based on the Seller. We decided to:

- replaced every instance of the Seller name with that Sellers user_id from our database.
- we added an image_url's that we found from the respective Sellers websites, that matched each tyre's serial_num

(To find the images of each tyre we had to manually search each Sellers website and copy the url.)

Below is an example of how the csv file for Micheline looked, before we imported it to the database:



	A	B	C	D	E	F	G	H
1	36	185/70 R 14	88	0	Energy XM2	5170	5170	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
2	36	185/65 R 15	88	0	Energy XM2	6540	6540	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
3	36	185/65 R 14	86	0	Energy XM2	5370	5370	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
4	36	195/55 R 16	87	0	Primacy 4ST	9110	9110	https://aecbmesvcm.cloudimg.io/v7/https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b
5	36	155/65 R 14	75	0	Energy XM2	3983	3983	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
6	36	215/60 R 16	99	0	Primacy 4ST	10760	10760	https://aecbmesvcm.cloudimg.io/v7/https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b
7	36	205/60 R 16	92	0	Primacy 4ST	8990	8990	https://aecbmesvcm.cloudimg.io/v7/https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b
8	36	165/70 R 14	81	0	Energy XM2	4290	4290	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
9	36	185/65 R 15	88	0	Energy XM2	6671	6671	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
10	36	195/55 R 16	91	0	Primacy 4ST	9292	9292	https://aecbmesvcm.cloudimg.io/v7/https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b
11	36	155/65 R 13	73	0	Energy XM2	4294	4294	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
12	36	225/65 R 17	102	0	Latitude Tour	14358	14358	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4qy8bc4wpzostic/pim_largepreview?
13	36	225/65 R 17	102	0	Primacy SUV	14963	14963	https://aecbmesvcm.cloudimg.io/v7/https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b
14	36	185/70 R 14	88	0	Energy XM2	5273	5273	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
15	36	195/65 R 15	91	0	Energy XM2	7273	7273	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview
16	36	195/65 R 15	91	0	Primacy 4ST	7670	7670	https://aecbmesvcm.cloudimg.io/v7/https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b
17	36	205/60 R 16	92	0	Primacy 4ST	9170	9170	https://aecbmesvcm.cloudimg.io/v7/https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b
18	36	165/70 R 14	81	0	Energy XM2	4376	4376	https://dxm.contentcenter.michelin.com/api/wedia/dam/transform/b98rpyxf61b4xm69hss7h6oyne/pim_largepreview

2. Other tables

- We also decided to submit particular entries in the database via PostMan, as we used it to test that our api.php was working. So entries from tables like rates, requests or click_events.
- Other entires like some Sellers form the users table, were populated using the signup.php page, for us to test that new users who sign up would be correctly and adequately added.

(Information about Sellers was acquired from their respective websites, and more niche information such as the business_reg_num was obtained through research if it was not readily available.)

Why did we choose to populate the data in this way?

1. Kaggle Dataset

- Scale: the dataset contained 2000+ tyre listings, which we felt was a significant sample size for analysis and considering this data was being used for price comparison's this was one of our priorities.
- Controlled Quality: Kaggle datasets are often sourced from reputable retailers or API's we felt comfortable about the credibility of our data.
- Real-World simulation: because the data had been sourced from real world brands and actual data, we felt this data was the closest thing we had to simulating a real-world environment.

Assumptions made: our Kaggle dataset was roughly 2-3 years old. However, we deemed it still relevant for analysis because, upon doing research we discovered that brands like Michelin advise tyre replacements after 5-10 years, so our data set prices would still be applicable.

2. Removing Duplicates

- Data Integrity: we ensured no redundant entries would skew analytics or inventory counts.

3. Separating by Seller/Brand

- Our your system handles multiple sellers, so separating tyres by brand made it easier to associate products with correct user_id.

4. Replacing Seller Names with user_id

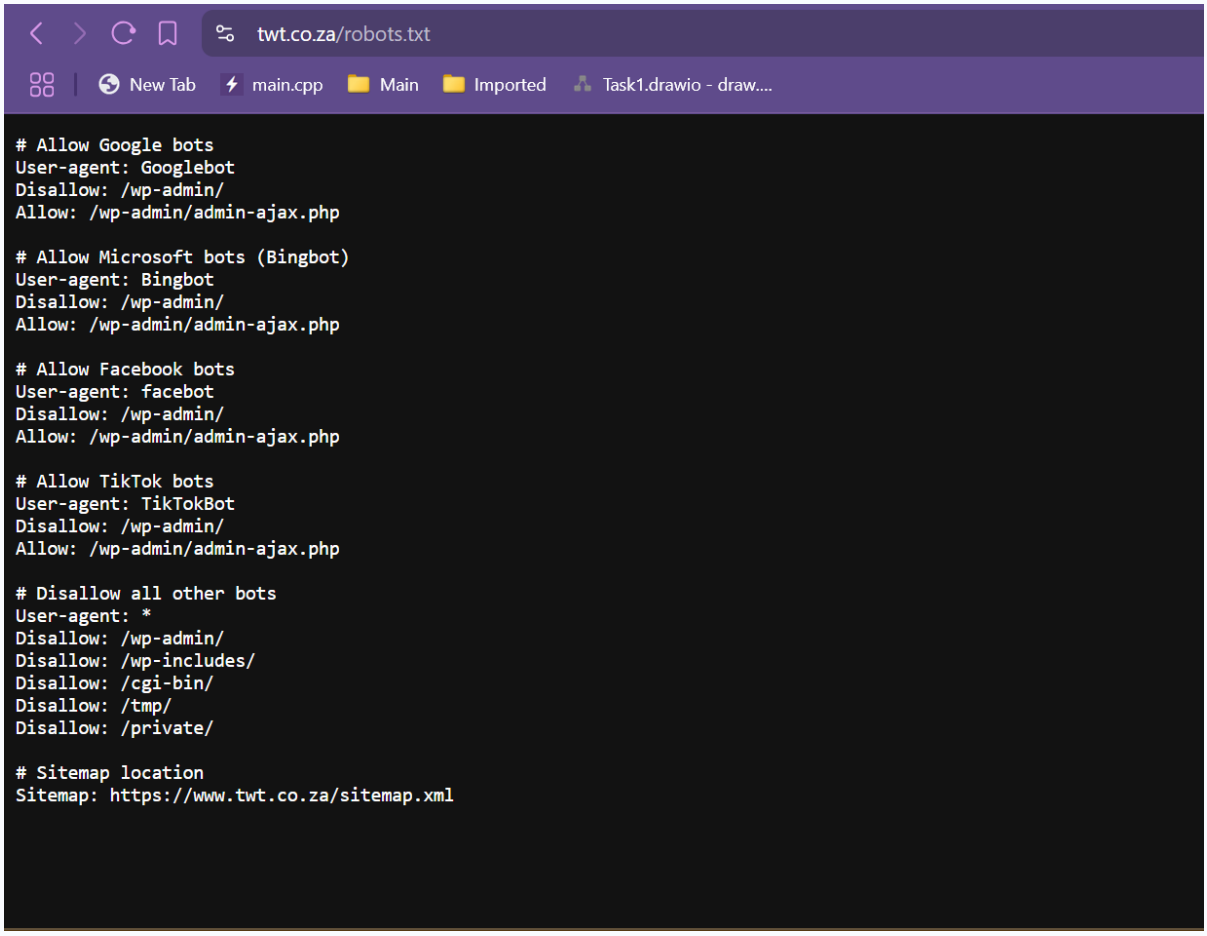
- Database Normalization: properly linked products to users via foreign keys instead of storing text names.

5. Using Postman for some entries

- API Validation: by manually testing entries via Postman we were able to verify that out API endpoints worked correctly. We were able to test edge cases, and verify error handling, and we confirmed that the database constraints and relationships behaved as expected.

6. Manually copying image url's

A lot of our Sellers websites either did not permit web-scraping, or they did not explicitly state that it was allowed. To air on the side of caution, we decided to manually add the product url's. Below is an example from Tiger Wheel & Tyre's robots.txt file (https://www.twt.co.za/robots.txt)



Task 7: Analyse and Optimise

We hosted our database in phpMyAdmin which primarily use MySQL or MariaDb as it's underlaying Relational Database Management System (RDBMS). We will do a query to analyse which involves filtering (WHERE), joining tables (JOIN), or aggregations (GROUP BY, ORDER BY).

What we will use to measure:
Type = ALL → Full table scan (inefficient)
Key = NULL → No index used
Rows = high number → Inefficient scan
Extra = Using temporary; Using filesort → Slow sorting/grouping

```
SELECT
   p.tyre_id,
   AVG(r.rating) AS avg_rating,
   COUNT(DISTINCT ce.click_id) AS click_count
FROM
   products p
LEFT JOIN rates r ON p.tyre_id = r.tyre_id
LEFT JOIN click_events ce ON p.tyre_id = ce.tyre_id
WHERE EXISTS (
   SELECT 1
   FROM requests rq
   WHERE rq.tyre_id = p.tyre_id
    AND rq.status = 'Approved'
    AND rq.request_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
)
GROUP BY p.tyre_id
ORDER BY avg_rating DESC
```

The above query will filters out only relevant approved requests. This will help us to test performance.

After running the SQL query:

d	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<subquery2>	ALL	distinct_key	*NULL*	*NULL*	*NULL*	4	Using temporary; Using filesort
1	PRIMARY	p	eq_ref	PRIMARY	PRIMARY	4	u23658462_GotD_project.rq.tyre_id	1	Using index
1	PRIMARY	r	ref	PRIMARY	PRIMARY	4	u23658462_GotD_project.rq.tyre_id	1	
1	PRIMARY	ce	ref	click_events_ibfk_2	click_events_ibfk_2	4	u23658462_GotD_project.rq.tyre_id	1	Using index
2	MATERIALIZED	rq	ALL	requests_ibfk_2	*NULL*	*NULL*	*NULL*	4	Using where

We see that there is poor performance.
type = ALL - full table scan, this is bad on large tables.
key = NULL – no index is being used.
Extra having using temporary and using filesort – means sorting in memory and it is expensive.

So we need to see:
type = ref or range
key
rows
Extra having Using index or using where with no Where

To improve:

```
CREATE INDEX idx_requests_tyre_status_date ON requests(tyre_id, status, request_date);

CREATE INDEX idx_rates_tyreid ON rates(tyre_id);
```

We ran the same SQL query. After Optimization:

d	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<subquery2>	ALL	distinct_key	*NULL*	*NULL*	*NULL*	3	Using temporary; Using filesort
1	PRIMARY	p	eq_ref	PRIMARY	PRIMARY	4	u23658462_GotD_project.rq.tyre_id	1	Using index
1	PRIMARY	r	ref	PRIMARY,idx_rates_tyreid	PRIMARY	4	u23658462_GotD_project.rq.tyre_id	1	
1	PRIMARY	ce	ref	idx_clicks_tyreid	idx_clicks_tyreid	4	u23658462_GotD_project.rq.tyre_id	1	Using index
2	MATERIALIZED	rq	index	idx_requests_tyre_status_date	idx_requests_tyre_status_date	10	*NULL*	4	Using where; Using index

We see a significant improvement.

Table	Type	Key	Rows	Extra
<subquery2>	ALL	distinct_key	3	Using temporary; Using filesort
products	eq_ref	PRIMARY	1	Using index
rates	ref	idx_rates_tyreid	1	
click_events	ref	idx_clicks_tyreid	1	Using index
requests	index	idx_requests_tyre_status_date	4	Using where; Using index

Requests – is scanned via index-only access with filtering (Using where; Using index), while the Products - uses efficient primary key lookups (eq_ref).
Both rates and click_events also use indexed access.
While a temporary table and filesort are still used for the top-level GROUP BY with LIMIT, the overall cost has dropped, and rows scanned are minimal (1–4 per table).

Task 8: Development

Usage of Git

We used Git through GitHub Desktop to commit changes, create branches and sync the project regularly. For committing changes logically, we grouped related updates with descriptive updates of the changes. For creating branches, we separated features to avoid conflicts. For syncing regularly, we pulled updates before pushing to ensure reduce conflicts.

1. Collaboration via GitHub

We merged branches via Pull Requests, and reviewed requests before accepting. We were all able to work on similar documents, and collaborate efficiently as long as we each pulled from the repository regularly and committed regularly.

2. GitHub Desktop

- GitHub desktop simplified the Git commands, so we were able to branch, commit, push and pull with ease.
- We were able to make small focused changes, make regular pushes to avoid lost work, and used the visual diff tools to verify edits and compare previous versions of code.

Benefits and what we learned:

- git diff (visual comparison), was able to catch unintended changes early between commits and branches, and this ensured that no code breaking changes occurred. Further it helped us verify changes.
- PDF's: these files were not diff-friendly, they bloat the repo size. So we only included them for critical documents. We tried our best to adhere to these practices by only uploading our EER diagrams, relational mapping etc. For text files that we did need to have exemplified by the apiRequest.txt we decided to make them .txt files instead because those are diff-friendly and lightweight. Further txt files helped enable our collaboration.

Data validation techniques

Below are some examples of our data validation techniques.

1. Input Validation

- Type Checking and casting to make sure the rating was between 1-5

```
In submitRating()
$tyreId = (int)$data['tyre_id'];
$rating = min(max((int)$data['rating'], 1), 5);
```

- Regex validation

```
 In handleRegistration()
if (!preg_match('/^[a-zA-Z\s\-]{2,50}$/', $data['name'])) {
   throw new Exception("Name format invalid");
}
```

- Required Field Checks

```
In handleRequest()
if (empty($data['type'])) {
   throw new Exception("Type parameter required", 400);
}
```

2. Role-Based Access Control

```
In editFAQ()
if ($currentUserRole !== 'Admin') {
   $this->sendErrorResponse("Admins only", 403);
}
```

In the above example, only Admin's could edit FAQ's

3. Data integrity validation
- Ensured referential integrity

```
In addFavourite()
$stmt = $this->connection->prepare("SELECT tyre_id FROM products WHERE tyre_id = ?");
```

In the example above, a tyre would have to exist before it can be favourited

- Unique constraints

```
In handleRegistration()
$stmt = $this->connection->prepare("SELECT user_id FROM users WHERE email = ?");
```

The code above prevents duplicate emails

4. Security Validations
- SQL injection prevention

```
Everywhere in our code ( exemplified in verifyApiKey())
$stmt = $this->connection->prepare("SELECT * FROM users WHERE api_key = ?");
$stmt->bind_param("s", $apikey);
```

We used bind parameters to prevent injections.

- Session Security

```
ini_set('session.cookie_httponly', 1);
ini_set('session.cookie_secure', 1); // HTTPS-only
```

5. Structural Validation
- JSON Validation

```
In handleRequest()
$data = json_decode($rawInput, true);
if (json_last_error() !== JSON_ERROR_NONE) {
   $this->sendErrorResponse("Invalid JSON", 400);
}
```

- Enum Validation

```
In handleEditRequest()
$validStatuses = ['Pending', 'Approved', 'Rejected'];
if (!in_array($newStatus, $validStatuses)) {
   throw new Exception("Invalid status");
}
```

To ensure that only valid data from the list was used

Package Managers

We did not use any.

**add a file structure

Task 9: Demo

Student	Contribution
Lerato Sibanda (u22705504)	1. Relational Mapping 2. Research (task 1) 3. Database creation 4. Frontend - login page css + js + php - signup page css + js + php - seller page css + js + php - admin page css + js + php
Kiarin Naidoo (u23600897)	1. Database creation 2. Frontend: - footer page - admin page php
Cleopatra Kwenda (u23547121)	1. EER Diagram 2. Database creation 3. Research (task 1) 4. Frontend: - price comparison page css + js + php - Linking and debugging all pages - header page php + css - homepage php + css 5. Backend: -modifiied some api functions
Neo Machaba (u23002167)	1. Admin: - meeting notes, created and managed Discord server - Managed GitHub branches 2. Frontend: - products page php 3. Database Creation
Kgaugelo Matsena (u23658462)	1. Relational Mapping 2. Frontend: - products page css + js + php - favourites page css + js + php - click_events page css + js + php 3. Backend - api.php creation and testing - api.php function requests ReadMe 4. Analysis and optimisation (task 7)
Nyashadzashe Makwabarara (u22783386)	1. EER Diagram 2. Database creation 3. Frontend: - viewpage.php + java + css 4. Backend: - api.php creation and testing

	- api.php function requests ReadMe 5. Database population via Kaggle dataset + PostMan api stress testing (task 6) 6. Creating and compiling this PDF 7. PowerPoint presentation

Task 10: Push the Boundaries

Enhanced UI/UX

- The site supports multiple themes including light, dark, to accommodate different user preferences and needs.
- Semantic HTML elements and keyboard navigation support improve usability for screen readers and users who rely on keyboards.

Security

- Password Hashing: We use SHA (Secure Hash Algorithm) for hashing passwords, ensuring data is stored securely.
- Session Management: Sessions are implemented with timeout policies and access control mechanisms restricting certain pages to authorized users only.
- We prevent SQL injection with prepared statements and bind parameters.
- Access control is strictly enforced to protect sensitive areas of the site.