

# **Campus Buy / Sell Portal Management**

## **CS 432: Databases**

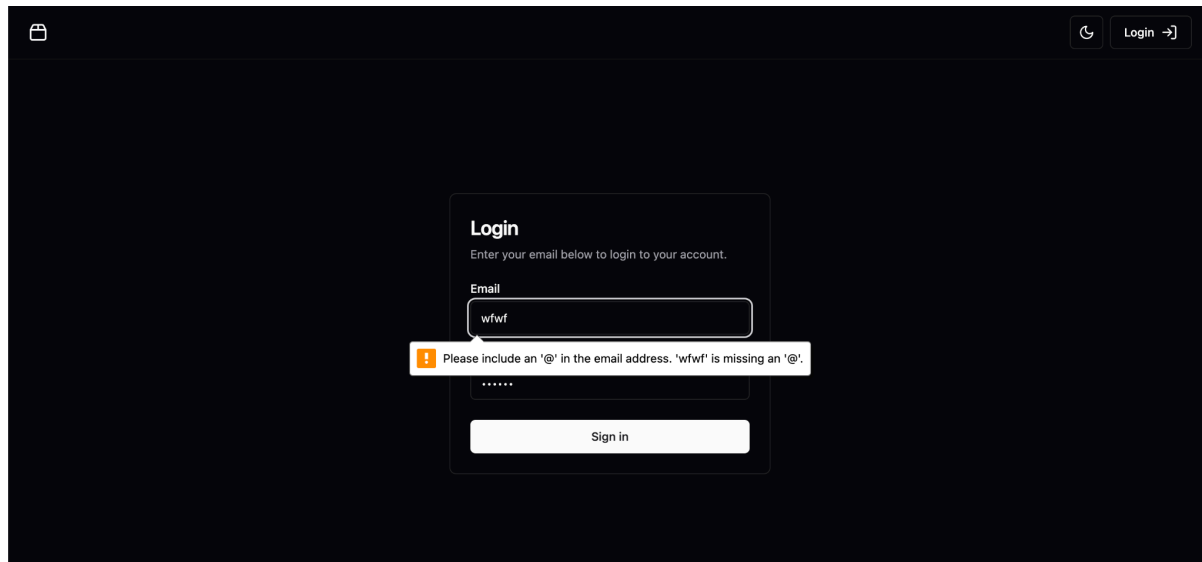
### **Assignment - 4**

#### **By: Group 'Lobby Bois'**

Manas Kawal	(21110119)
Naman Dharmani	(21110136)
Parth Deshpande	(21110151)
Rachit Verma	(21110171)
Sahil Das	(21110184)
Tirth Patel	(21110225)
Yash Bothra	(21110242)

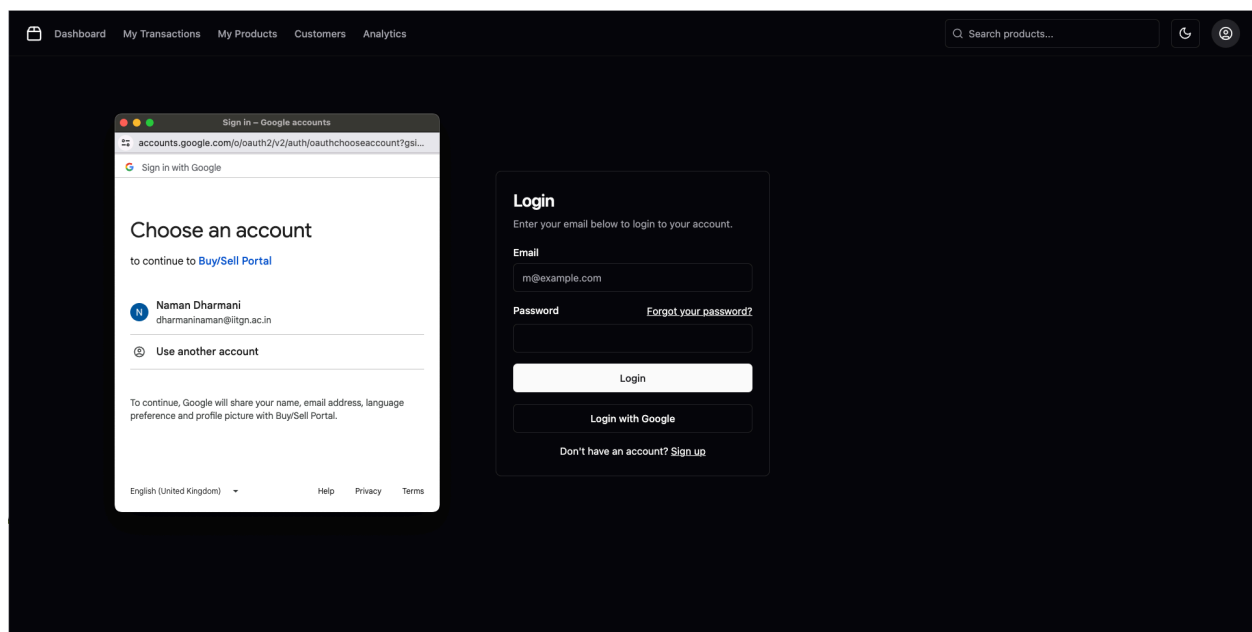
The web application serves as a dynamic buy-and-sell portal, offering users a platform to browse, list, and purchase a wide range of products. With its user-friendly interface and robust features, the portal provides a seamless experience for both buyers and sellers. Users can explore different categories, view detailed product listings, and interact with sellers directly.

This is the login page for the Web Application.



The screenshot shows a dark-themed login page. At the top right, there is a home icon, a refresh button, and a "Login" button with a right arrow. The main content area features a "Login" form with the instruction "Enter your email below to login to your account." The "Email" input field contains the text "wfwf". Below the input field, a red error message box states: "Please include an '@' in the email address. 'wfwf' is missing an '@'." Below the error message is a password input field with six dots. At the bottom of the form is a "Sign in" button.

We can see it lacks Google Authentication; here is the login page with the implementation of Google Authentication.



The screenshot shows the same dark-themed login page, but with Google Authentication implemented. The top navigation bar includes links for "Dashboard", "My Transactions", "My Products", "Customers", and "Analytics", along with a search bar labeled "Search products...". The "Login" form now includes an "Email" field with "m@example.com" and a "Password" field. A "Forgot your password?" link is next to the password field. Below the password field is a "Login" button. Further down is a "Login with Google" button. At the very bottom, there is a link: "Don't have an account? Sign up".

On the left side of the page, a "Sign in - Google accounts" window is open. It shows the "Choose an account" screen for "Buy/Sell Portal". It lists the account "Naman Dharmani" with email "dharmaninaman@iitgn.ac.in" and a "Use another account" option. At the bottom of the window, it says "To continue, Google will share your name, email address, language preference and profile picture with Buy/Sell Portal." and includes links for "Help", "Privacy", and "Terms".

Pagination retrieves only the desired amount of products and their details from the database per page. Here in the example shown for a catalog of 10 products, five are displayed per page.






DashboardMy TransactionsMy ProductsCustomersAnalytics

Q Search products...

FilterExportAdd Product

Products

Manage your products and view their sales.

	Name	Subcategory	Status	Price	Quantity	Created at	
	Macbook M1	Smartphones	Available	954.00	4	2024-04-05 10:26 PM	...
	Iphone 15	Laptops	Available	634.00	2	2024-04-05 10:26 PM	...
	Ipad Air	Tablets	Available	12.00	1	2024-04-05 10:26 PM	...
	Boat Headphones	Headphones	Available	364.00	3	2024-04-05 10:26 PM	...
	Nikon DSLR	Cameras	Available	5.00	4	2024-04-05 10:26 PM	...

Showing 1-5 of 10 products

These are obtained with respect to the seller persona of the user. A query is run on the listings table with the user id of the user to get the relevant products.






DashboardMy TransactionsMy ProductsCustomersAnalytics

Q Search products...

FilterExportAdd Product

Active Products

Manage your listed products.

	Name	Subcategory	Status	Price	Quantity	Created at	
	Sony SmartTV	Televisions	Available	269.00	3	2024-04-05 10:26 PM	...
	Apple Watch Series 9	Smart Watches	Available	560.00	4	2024-04-05 10:26 PM	...
	Mixer-Grinder	Gaming Consoles	Available	186.00	3	2024-04-05 10:26 PM	...
	PS5	Accessories	Available	812.00	4	2024-04-05 10:26 PM	...
	Heater	Home Appliances	Available	166.00	3	2024-04-05 10:26 PM	...

Uploading new products:

This is a utility to add a new listing by the present user. This uses the POST HTTP method to update the relevant tables - the product and listing tables.

DashboardMy TransactionsMy ProductsCustomersAnalytics

Q Search products...

DiscardCreate Product

Product Details

Title

Nothin Phone1

Description

#glyph #nothing  
Nothing by Nothing

Listed Price

33500

Quantity

1

Product Category

Category

Electronics

Subcategory

Smartphones


Product Status

Status

Available

Product Images

Upload or remove an image



New Product Added:

DashboardMy TransactionsMy ProductsCustomersAnalytics






Q Search products...

AllActiveSoldArchived

FilterExportAdd Product

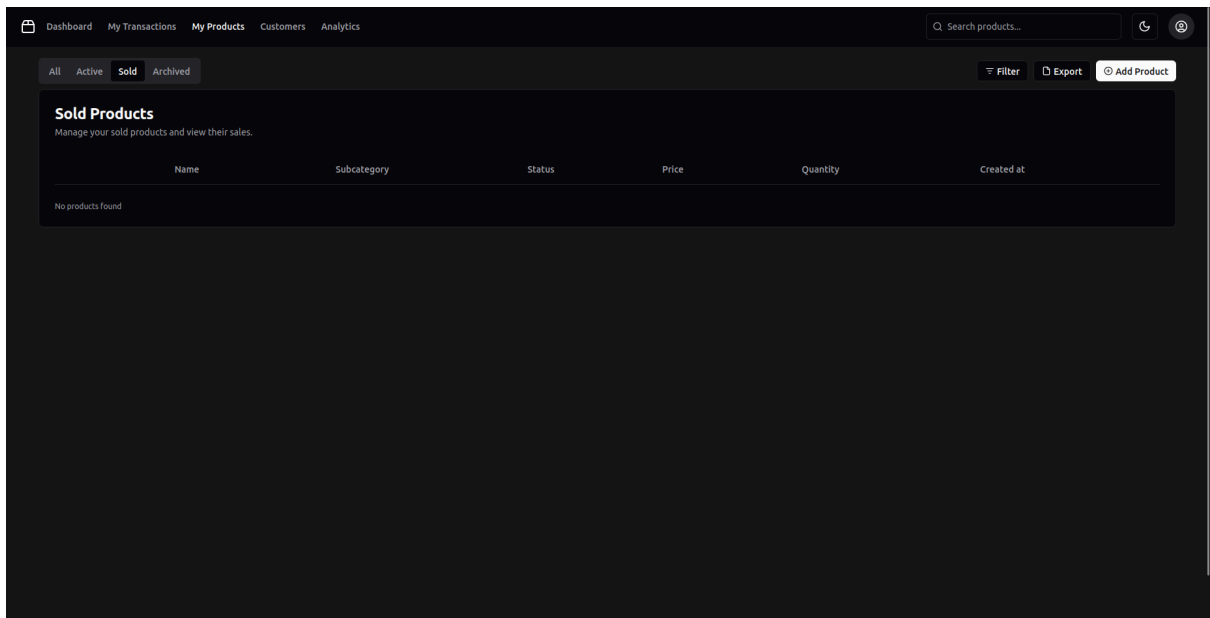
Products

Manage your products and view their sales.

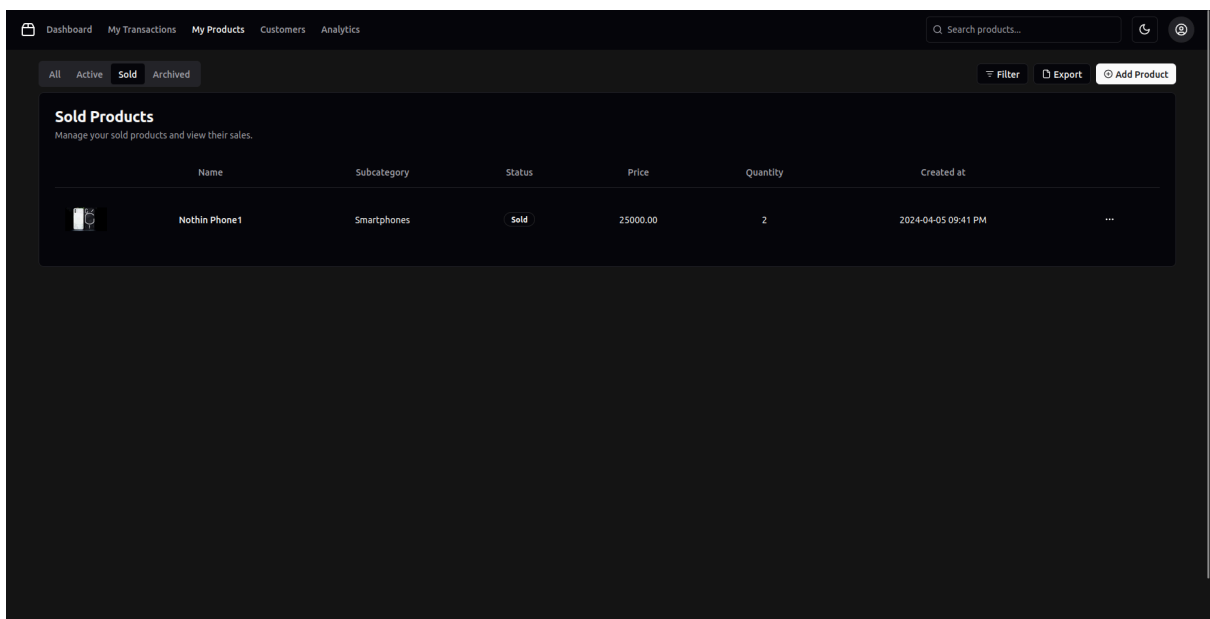
	Name	Subcategory	Status	Price	Quantity	Created at	
	Nothin Phone1	Smartphones	Available	33500.00	1	2024-04-05 10:29 PM	...
	Macbook M1	Smartphones	Available	954.00	4	2024-04-05 10:26 PM	...
	Iphone 15	Laptops	Available	634.00	2	2024-04-05 10:26 PM	...
	Ipad Air	Tablets	Available	12.00	1	2024-04-05 10:26 PM	...
	Boat Headphones	Headphones	Available	364.00	3	2024-04-05 10:26 PM	...

Showing 1-5 of 11 products

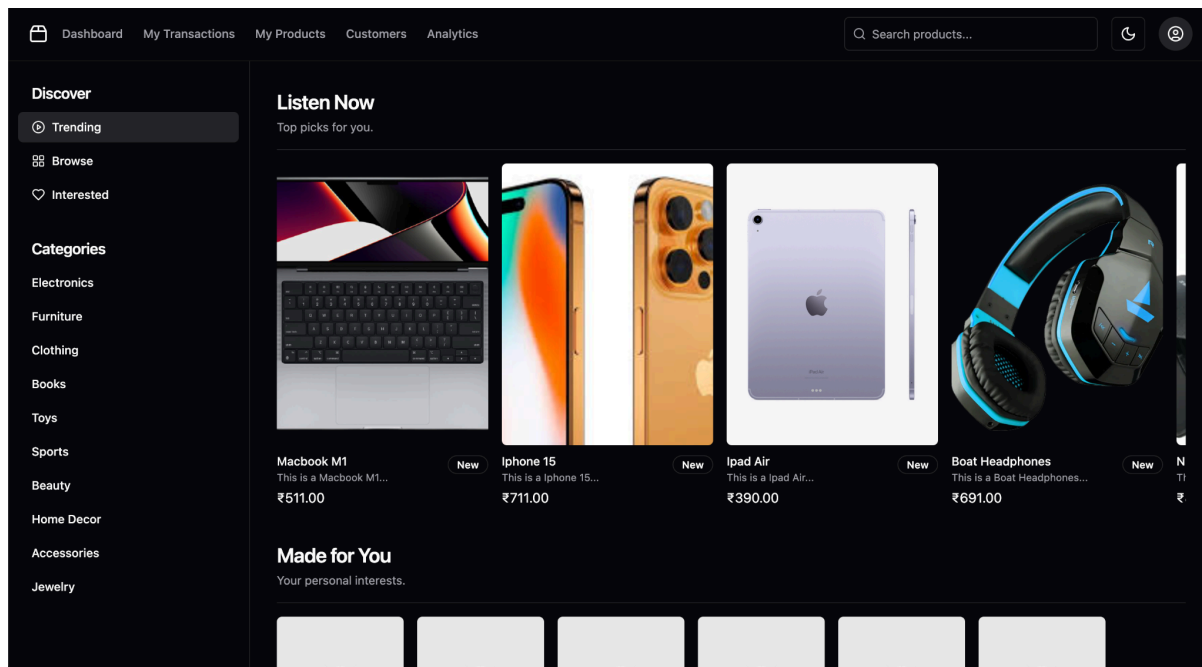
Initially, when no product has been sold by the user:



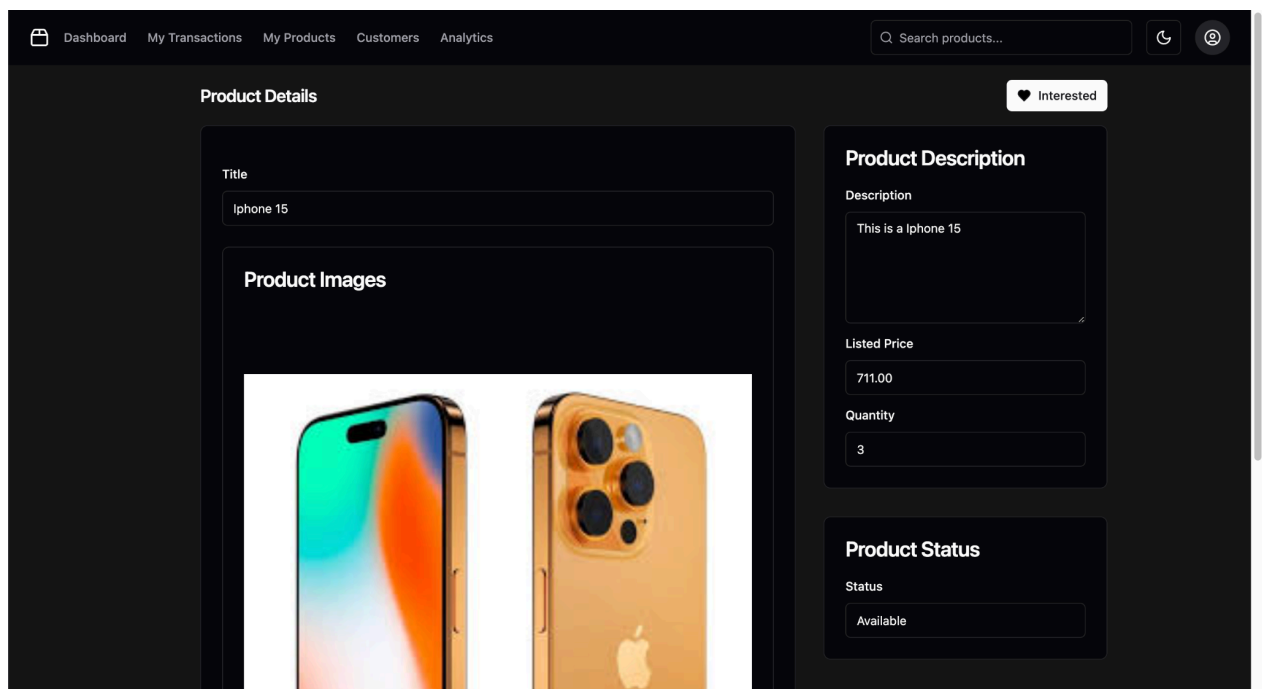
After selling the product. Product status changed to sold:

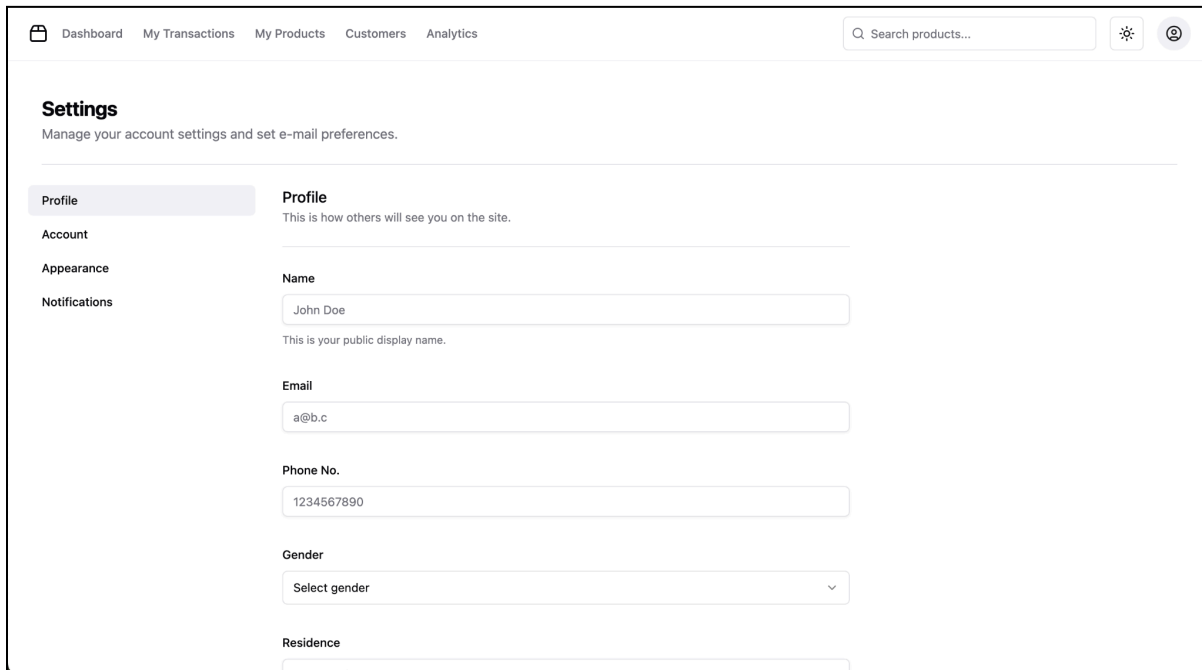
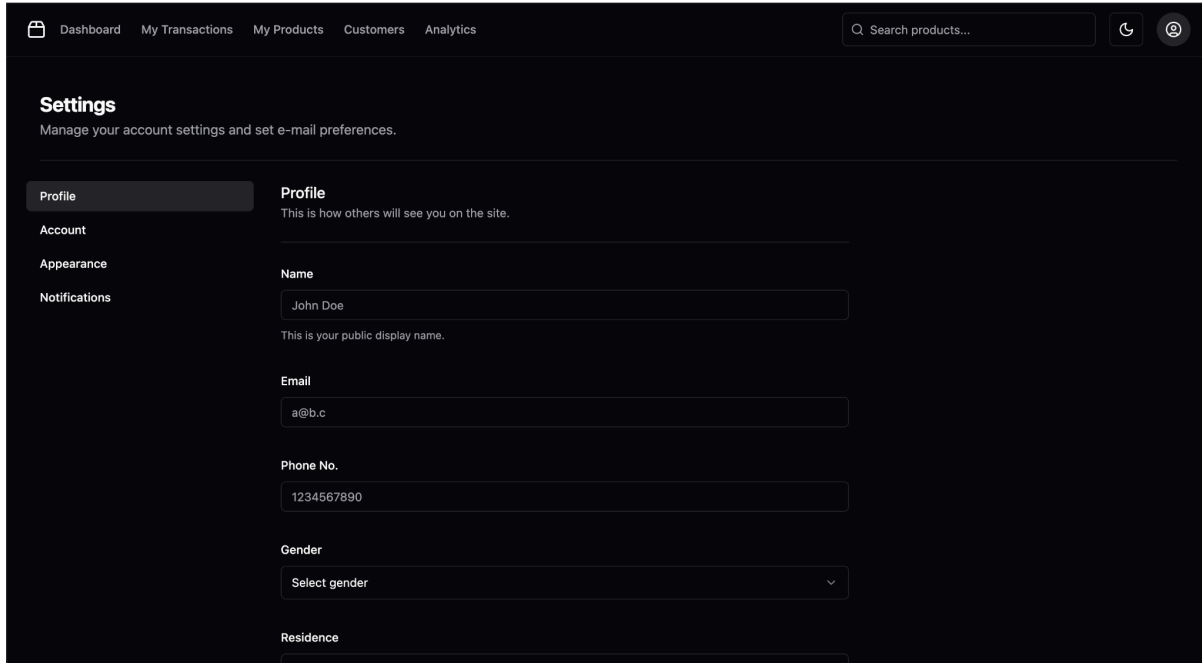


This is a page from the buyer's perspective. As is visible, the user has displayed the currently listed products, along with their respective images and prices. The links are clickable, after which the user is taken to another product page, this time from the buyer's perspective.



The product page from the buyer's perspective. The buyer is displayed only the relevant information and is exposed only to the relevant actions - like marking the product as interested.





The screenshots above demonstrate the option available to users to switch the application theme between dark and light modes. This feature is accessible on all pages and can be set as default based on the user's preference.

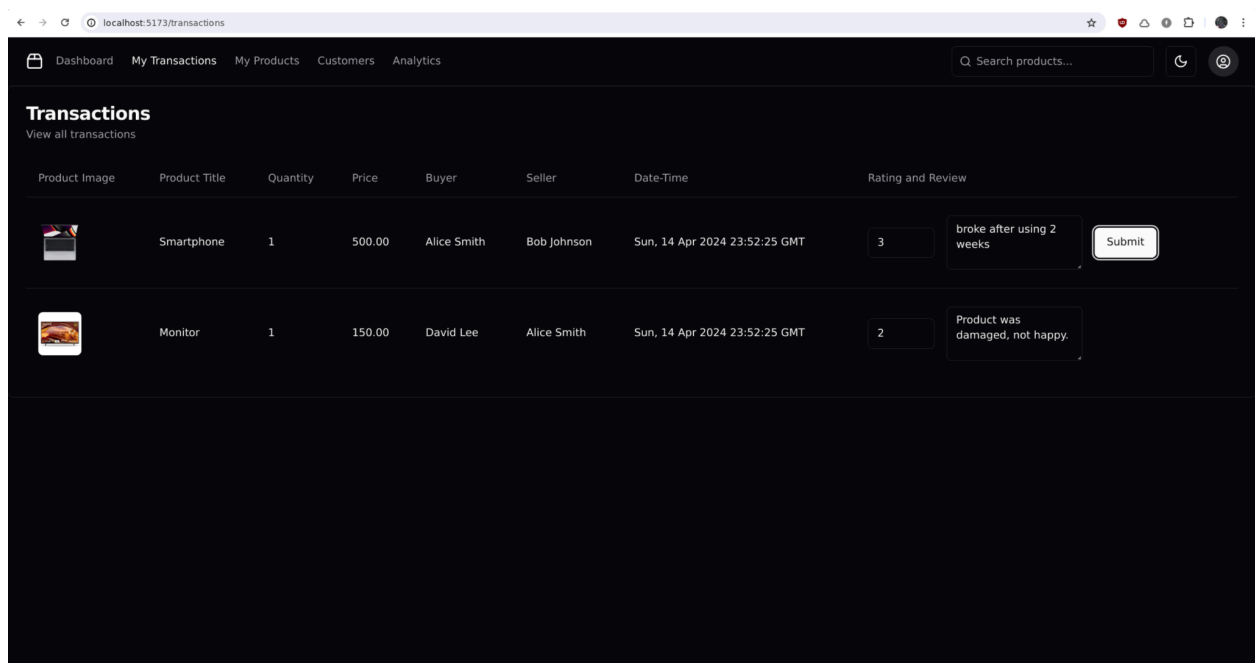
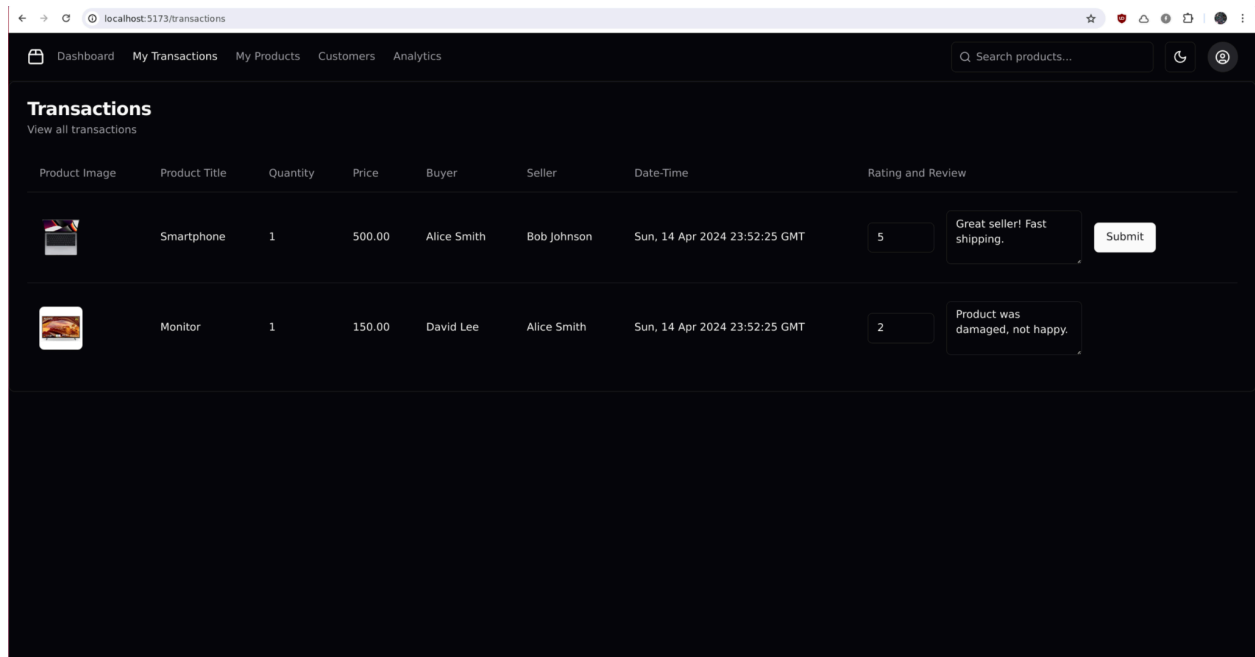
The web application has been refined based on feedback, incorporating enhancements to improve user experience.

The two major feedbacks given were based on:

**1) Implementation of the feedback:**

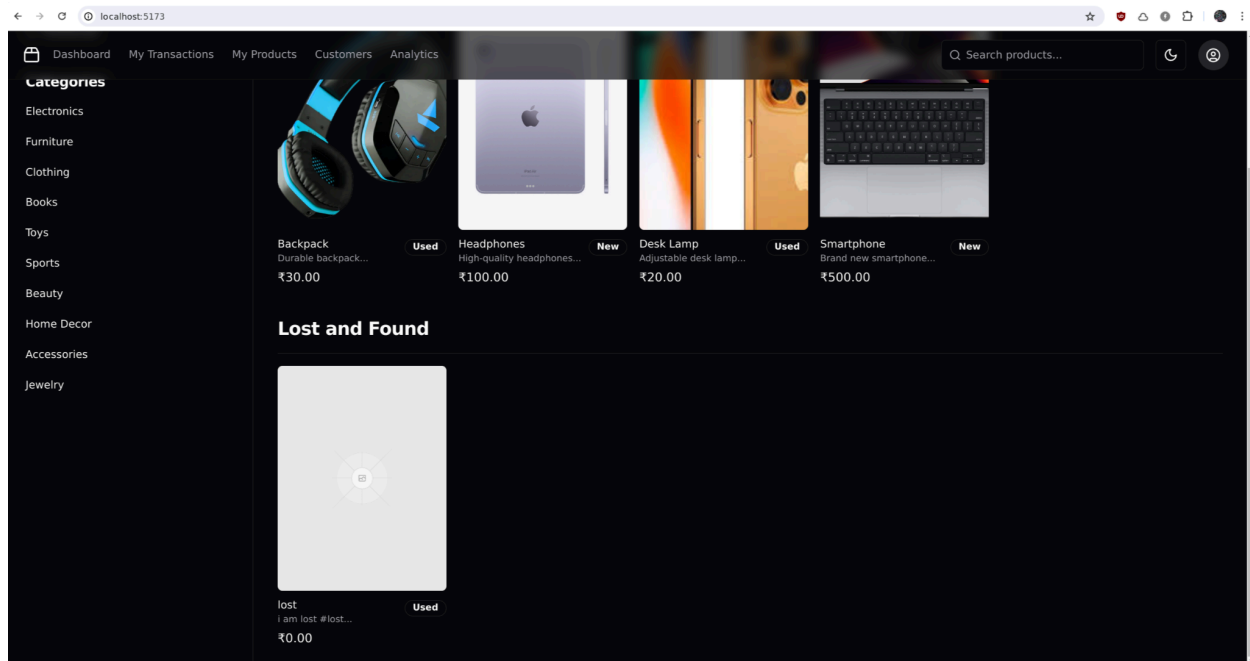
Implementing a feedback system that displays ratings from previous buyers would greatly enhance the user experience. This feature would provide valuable insights into the seller's credibility based on past transactions, helping users make more informed decisions when engaging in deals.





## 2) Campus Lost and Found:

Incorporating a dedicated section for lost and found items on the campus website would greatly streamline the management of lost items. This feature would enable anyone who finds lost items without an owner to easily report and manage them on the website, facilitating a more efficient process for reuniting lost items with their owners.



## SQL Injection:

To inject malicious SQL queries into the backend. We looked for suitable input fields. The product creation form was a perfect choice. Since it is about the listing of a new product, the structure of the SQL query internally would look something like this-

```
INSERT INTO Product (col1, col2, ..) VALUES(val1, val2, ..);
```

Firstly we tried to delete all the entries in the Product table by utilizing the concept of nested query. In the product\_title section, we put (DELETE FROM Product) as the input. However, a newly created product appeared and the previously listed ones were still there. This was because we are using SQLAlchemy ORM to interact with the database. SQLAlchemy automatically escapes any data that it sends in SQL queries, which helps to prevent SQL injection attacks. Therefore, as there is no raw SQL query in the backend SQL Injection becomes tough.

Also, we submitted another form where the inner nested query is:

```
); DROP TABLE Product; --
```

The idea was that the VALUES would be terminated using '); and the remaining part of the query would be commented using --. So, the first query that would run would be DROP TABLE Product. But again, it was added as a new entry in the product table. Moreover, in some cases, parameterized input is used, but the variables that are passed would be treated as a data only and not a part of the SQL queries.

```
mysql> select * from Product;
```

prod_id	prod_title
1	Smartphone
2	Desk Lamp
3	Headphones
4	Backpack
5	Camera
6	Monitor
7	(DELETE FROM Product)
8	); DROP TABLE Product; --

```
8 rows in set (0.00 sec)
```

### XSS Attacks:

In the routes section, there are checks to validate the query variables in the URL. But there is one route that is independent of checks, i.e., `/products/?page=<value>`. There it was possible to run a script. The following script was added to it-

```
'>};</script><script>alert(0)</script>//
```

Alert was used to test whether it was possible or not to exploit the vulnerability. By running it, no such alert popped. To see what went wrong, we changed the function in `ProductList.jsx` to `console.log` the query value and see what is returned at the end. The following screenshot shows the changed function.

```
async function loader({ request }) {
  const url = new URL(request.url);
  const page_num = url.searchParams.get("page") ?? 1;

  console.log("query_value:", page_num);
  const return_stm = fetch(`${import.meta.env.VITE_URL}/products?page=${page_num}`);
  console.log("return_stm:", return_stm);

  return return_stm;
}
```

After the values were printed on the console. It became evident that the the ASCII value of `<` is encoded to `%3C`, and thus it didn't work.

[vite] connecting...	<a href="#">client.ts:19:8</a>
[vite] connected.	<a href="#">client.ts:173:14</a>
query_value: }`);</script><script>alert(0)</script> //	<a href="#">ProductList.jsx:61:10</a>
return_stm:	<a href="#">ProductList.jsx:63:10</a>
▼ Promise { <state>: "pending" } <state>: "fulfilled"	
▼ <value>: Response { type: "cors", url: " <a href='\"http://127.0.0.1:5000/products?page=}\"'>http://127.0.0.1:5000/products?page=</a> `)};%3C/script%3E%3Cscript%3Ealert(0)%3C/script%3E%20//", redirected: false, ... }	
▶ body: ReadableStream { locked: true } bodyUsed: true	
▶ headers: Headers { "content-length" → "4160", "content-type" → "application/json" }	
ok: true	
redirected: false	
status: 200	
statusText: "OK"	
type: "cors"	
url: " <a href='\"http://127.0.0.1:5000/products?page=}\"'>http://127.0.0.1:5000/products?page=</a> `)};%3C/script%3E%3Cscript%3Ealert(0)%3C/script%3E%20//"	
▶ <prototype>: ResponsePrototype { clone: clone(), arrayBuffer: arrayBuffer(), blob: blob(), ... }	
▶ <prototype>: Promise.prototype { ... }	

React has XSS protection by design. Let's say a malicious script is inserted into a input field then even if it is not escaped, then also it will not work. This is because it would be treated as a string type instead of HTML tags. For example, we also tried to use alert(0) without the script tag, and as expected, it didn't work.

string	<a href="#">ProductList.jsx:62:10</a>
query_value: }`);alert(0);//	<a href="#">ProductList.jsx:64:10</a>
return_stm:	
▼ Promise { <state>: "pending" } <state>: "fulfilled"	
▼ <value>: Response { type: "cors", url: " <a href='\"http://127.0.0.1:5000/products?page=}\"'>http://127.0.0.1:5000/products?page=</a> `)};alert(0);//", redirected: false, ... }	
▶ body: ReadableStream { locked: true } bodyUsed: true	
▶ headers: Headers { "content-length" → "4160", "content-type" → "application/json" }	
ok: true	
redirected: false	
status: 200	
statusText: "OK"	
type: "cors"	
url: " <a href='\"http://127.0.0.1:5000/products?page=}\"'>http://127.0.0.1:5000/products?page=</a> `)};alert(0);//"	
▶ <prototype>: ResponsePrototype { clone: clone(), arrayBuffer: arrayBuffer(), blob: blob(), ... }	
▶ <prototype>: Promise.prototype { ... }	

```
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - abs 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - abt 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - abu 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - abv 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - abw 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - abx 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - aby 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - abz 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - aca 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - acb 401
127.0.0.1 - - [14/Apr/2024 23:47:40] "POST /login HTTP/1.1" 401 - acc 401
127.0.0.1 - - [14/Apr/2024 23:47:41] "POST /login HTTP/1.1" 401 - acd 401
127.0.0.1 - - [14/Apr/2024 23:47:41] "POST /login HTTP/1.1" 401 - ace 200
127.0.0.1 - - [14/Apr/2024 23:47:41] "POST /login HTTP/1.1" 401 - FOUND: ace
127.0.0.1 - - [14/Apr/2024 23:47:41] "POST /login HTTP/1.1" 401 -
127.0.0.1 - - [14/Apr/2024 23:47:41] "POST /login HTTP/1.1" 200 - ...[!?] via 2, v3.10.12 (.venv)
```

Defense: Flask\_Limiter is used.

## Concurrency

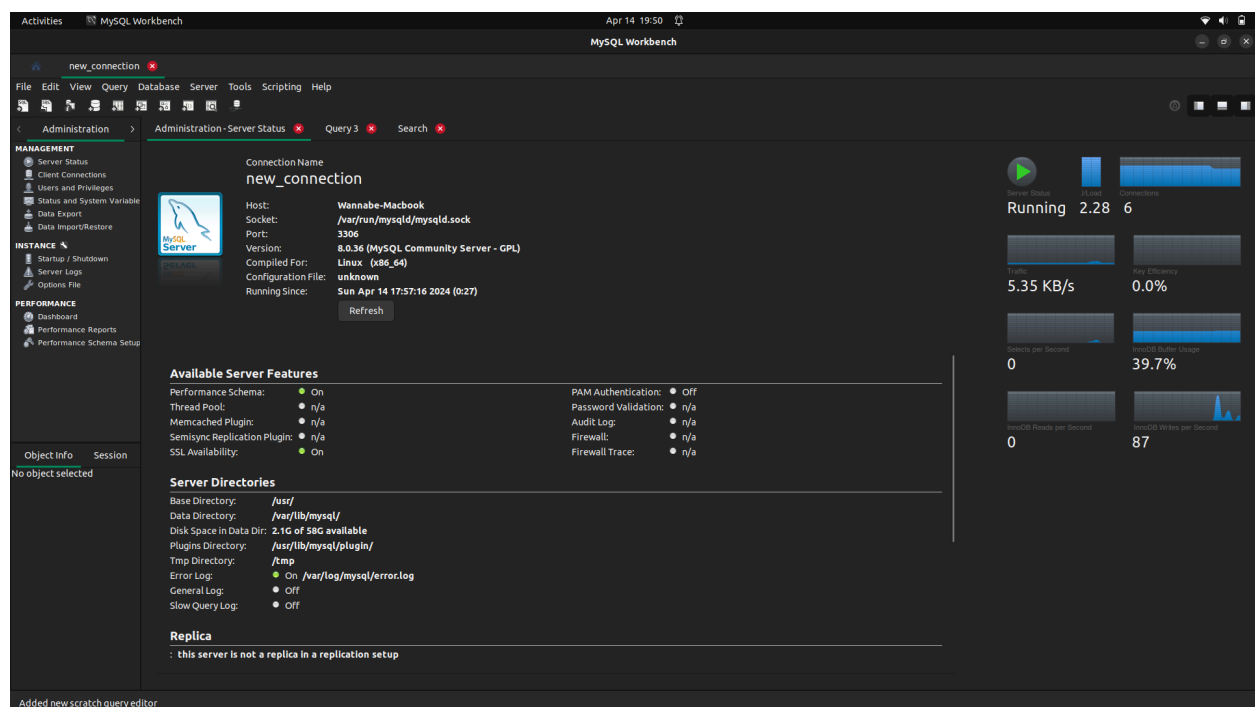
SQLAlchemy has features that support concurrency out of the box. SQL is also robust to concurrent requests through the use of transactions. To get higher robustness, we use `db.session.begin_nested()` and `db.rollback()` to process every user request as a transaction. We also configure the flask app to process every request in an independent thread using the `threaded` parameter. We also changed the app configuration to allow more than the default 5 threads to spawn.

In our case, locking the tables themselves is unnecessary and slows down the performance owing to the projected load, the design, and the complexity of queries. Hence the use of transactions is a better solution in the given context.

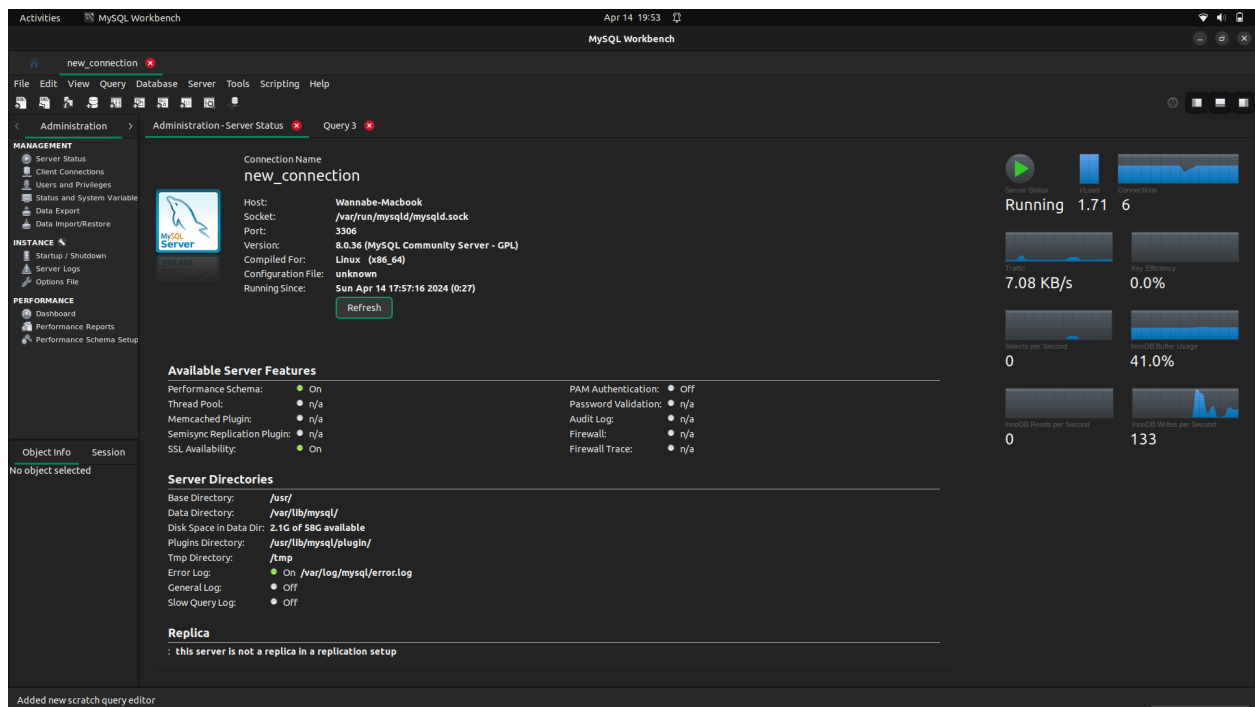
Moreover, most transactions can be accomplished quickly since they are relatively short queries. Since multiple server threads are being spawned, the system can handle multiple concurrent requests well.

We also included `db.rollback()` to rollback in case any of the transactions go wrong.

We wrote a python script to emulate 1000 users trying to send a post request. The server handled these concurrent requests while maintaining the data integrity.



This is the mysql server when the application server is idle.



This is the mysql server when the clients are sending concurrent requests. As is visible, the traffic has now increased. The number of connections does not increase due to the nature of the requests themselves, most of which can be handled very quickly.

## Views

Since we are using sqlalchemy, we have not written raw SQL queries but used the orm's methods to accomplish the same. Moreover, the operations themselves are designed not to be too complex. Hence, using views to abstract complex operations is unnecessary and does not apply to our case.

However, there needs to be an admin route in the backend, which would be used by a future administrator to view all the users and products that are up for sale in the application. However, in this case, the users' privacy needs to be protected. The administrator does not need to be exposed to users' sensitive information. Hence, we create a view and use that table to display only the essential user information to the administrator.

---

```
1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `exp`@`localhost`
4     SQL SECURITY DEFINER
5     VIEW `buy_sell`.`admin_view_user` AS
6     SELECT
7         `buy_sell`.`User`.`user_id` AS `user_id`,
8         `buy_sell`.`User`.`name` AS `name`,
9         `buy_sell`.`User`.`email` AS `email`
10    FROM
11        `buy_sell`.`User`
```

This is the definition of admin\_view\_user, the view the admin\_user route uses to view relevant information about all the users.

---

```
1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `exp`@`localhost`
4     SQL SECURITY DEFINER
5     VIEW `buy_sell`.`admin_view_prod` AS
6     SELECT
7         `buy_sell`.`User`.`name` AS `name`,
8         `buy_sell`.`User`.`email` AS `email`,
9         `buy_sell`.`Product`.`prod_title` AS `prod_title`
10    FROM
11        ((`buy_sell`.`User`
12        JOIN `buy_sell`.`listings`)
13        JOIN `buy_sell`.`Product`)
```

This is the definition of admin\_view\_prod, the view the admin\_prod route uses to view relevant information about all the products.

This view can be further expanded to other tables, where similar essential information will be accessible to all the admin routes.

Both of the routes have only been granted read-only permissions, since the admin would not be allowed to make changes to the database, and will need to contact the maintainers of the application to do the following changes.



## Contributions

Name	Group	Responsibilities & Contributions
Manas Kawal	G1	<ul style="list-style-type: none"><li>Helped in coming up with the right designs for frontend with the right set of routes to handle all the user interactions.</li><li>Took feedback from stakeholders.</li><li>Handled some tailwind classes.</li></ul>
Naman Dharmani	G1	<ul style="list-style-type: none"><li>Helped in setting up Google authentication and url route authorisation.</li><li>Incorporating the feedback on the frontend side.</li><li>Helped in documentation.</li></ul>
Parth Deshpande	G2	<ul style="list-style-type: none"><li>Made sure that the feedback provided by the stakeholders was implemented and made necessary changes accordingly.</li><li>Assisted in developing appropriate frontend designs.</li><li>Helped in documentation.</li></ul>
Rachit Verma	G2	<ul style="list-style-type: none"><li>Studying concurrency models and deciding the best ways of implementing concurrency for our application.</li><li>Implemented the final concurrency-related changes.</li><li>Wrote scripts to automate concurrency testing.</li><li>Studied the application for possible application of views and implemented them in code.</li></ul>
Sahil Das	G2	<ul style="list-style-type: none"><li>Designed and studied various possible attacks for our application.</li><li>Implemented the various attacks and tested them.</li><li>Helped in documentation</li></ul>
Tirth Patel	G2	<ul style="list-style-type: none"><li>Helped in setting up Google authentication.</li><li>Incorporating the feedback on the frontend side. Added "Lost and Found" status to products.</li><li>Add section of lost and found in Home page</li><li>Add Transactions page with rating and review editable</li><li>Helped in documentation.</li></ul>
Yash Bothra	G1	<ul style="list-style-type: none"><li>Made sure that the feedback provided by the stakeholders was implemented and made necessary changes accordingly.</li><li>Assisted in developing appropriate frontend designs.</li><li>Helped in documentation.</li></ul>