**UTIBU HEALTH REPORT**

**BY STEPHEN MUTETI**

You can access the application through the following link. Log in with the following admin details to access both the normal user and admin privileges

email: [admin@utibu.com](mailto:admin@utibu.com)

password: Jocteve1@

<https://utibu-frontend-56d24be6acf1.herokuapp.com>

**Introduction**

This is an application meant for Utibu Health facility. The solution entails a hybrid mobile application. The Backend is built/developed using Python Flask while the Frontend is built using React and converted to a mobile app using React Native. This approach was chosen to enable users/clients to access the services from their comfort without having to necessarily install the application (the application exists should they need to download it anyway).

The code has been clearly documented using concise and straightforward comments for clarity. In the sections that follow I describe the technologies used, the project overview where I show briefly how the application functions to achieve its goals and finally delve in to the performance and scalability measures put in place.

**Technical architecture**

In the technical architecture of the application, consideration has been given to integrating the online ordering system with the existing legacy database utilized by the pharmacist's inventory system. This integration ensures seamless coordination between online system and face-to-face sales, allowing for efficient management of medication inventory, customer orders, and sales transactions.

The integration involves configuring both systems to periodically synchronize data between the two database systems over the reliable internet connection at the health facility. When an order is placed online the stored data can be send to the **legacy database** through the synchronization. When a face-to-face sale occurs the record can be send to the cloud database either via the synchronization or the admin API endpoints.

By adopting this approach, the pharmacist can leverage the existing infrastructure and workflow established by the legacy inventory system while incorporating the convenience of online ordering for customers. This integration ensures that all orders, regardless of their origin, are accurately recorded in the legacy database, facilitating streamlined inventory management and customer service.

**Flask**

Flask is a lightweight and flexible web framework for Python used to develop the backend RESTful API endpoints.

**Flask\_Bcrypt**

Flask\_Bcrypt is a Flask extension that provides bcrypt hashing utilities for securing user passwords. It has been utilized to ensure that user passwords are hashed before storage.

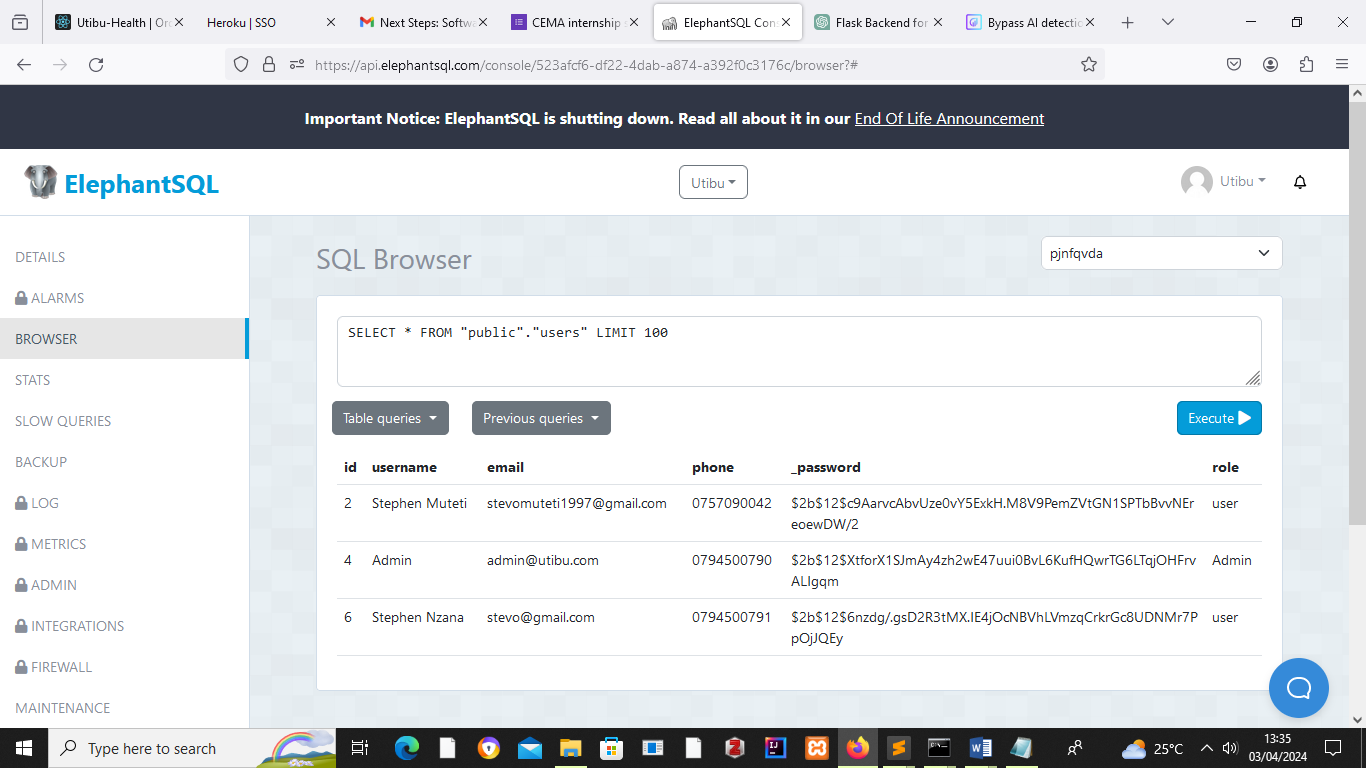


Figure 1: Hashed user passwords

**SQLAlchemy**

SQLAlchemy has been utilized in this application/system to simplify working with database entities as Python Objects, abstracting the complexity of SQL queries.

**CORS (Cross-Origin Resource Sharing)**

CORS is a mechanism that allows web servers to specify which origins are permitted to access resources. It has used to enable secure cross-origin requests between the frontend and backend components of the application.

**JWT (JSON Web Tokens)**

JWT is a compact, URL-safe means of representing claims to be transferred between two parties. It has been used for authentication and authorization purposes, allowing users to securely access protected endpoints by providing a token with each request.

**AWS PostgreSQL database**

The application utilizes an AWS PostgreSQL database for storing persistent data, such as user information, medication details, orders, and payment records. PostgreSQL is a powerful, open-source relational database management system known for its reliability, performance, and advanced features. Hosting the database on AWS ensures scalability, reliability, and accessibility, while PostgreSQL provides robust data management capabilities.

**Project Overview**

Let us walk through what each of the user roles is capable of doing on the application.

1. **User/Client**

This is the role for a normal user.

1. **User Registration and Authentication**

Users can register for an account and authenticate themselves securely using JWT (JSON WEB TOKENS). Passwords are encrypted for enhanced security. The role of a user is set in the backend to evade security leaks.

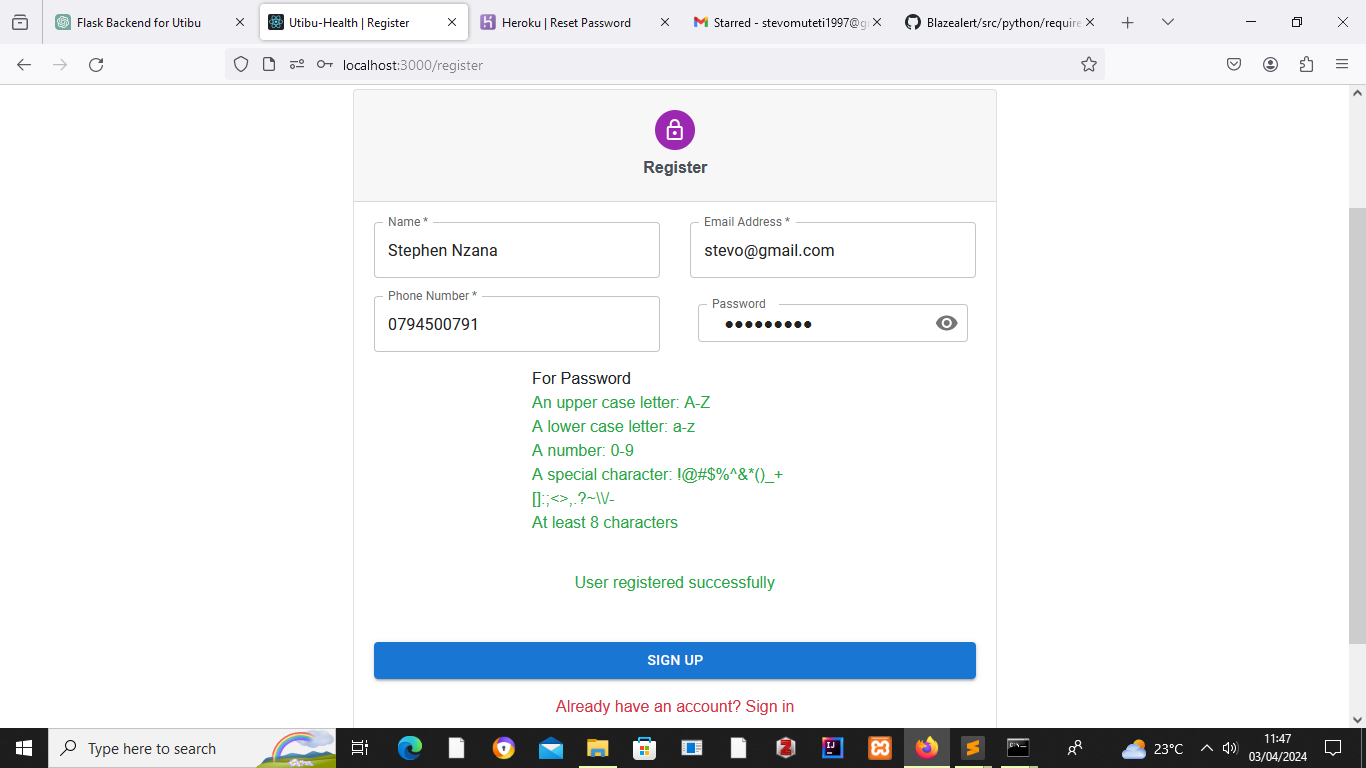


Figure 2: A screengrab for a successful user registration

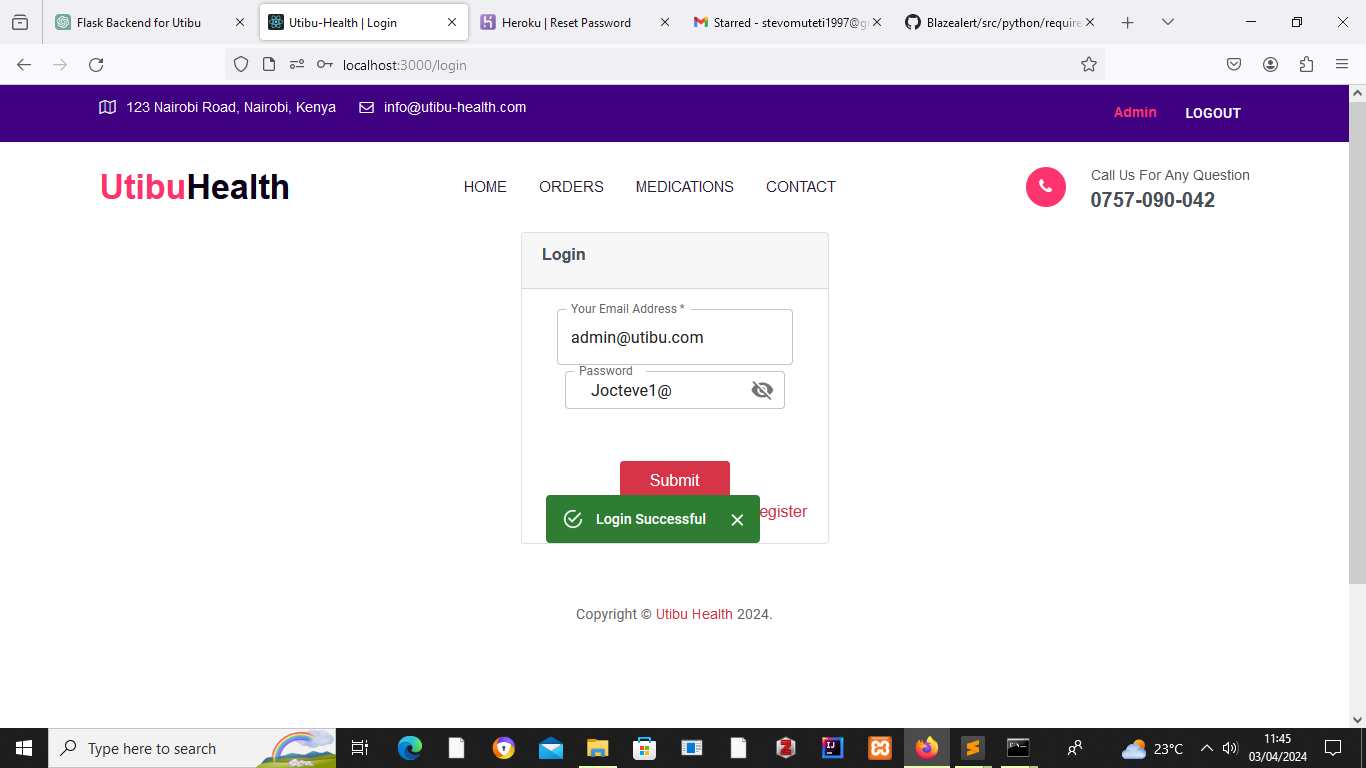


Figure 3: Successful User Login

1. **Medications Page**

On the medications tab/page for the normal user, a table displays the medications available in the store/pharmacy. The medication details listed are id, name and price. Alongside the table is a form which can also be used for order placement. The medications page enables the client to get the medication Ids for the medications they wish to order.

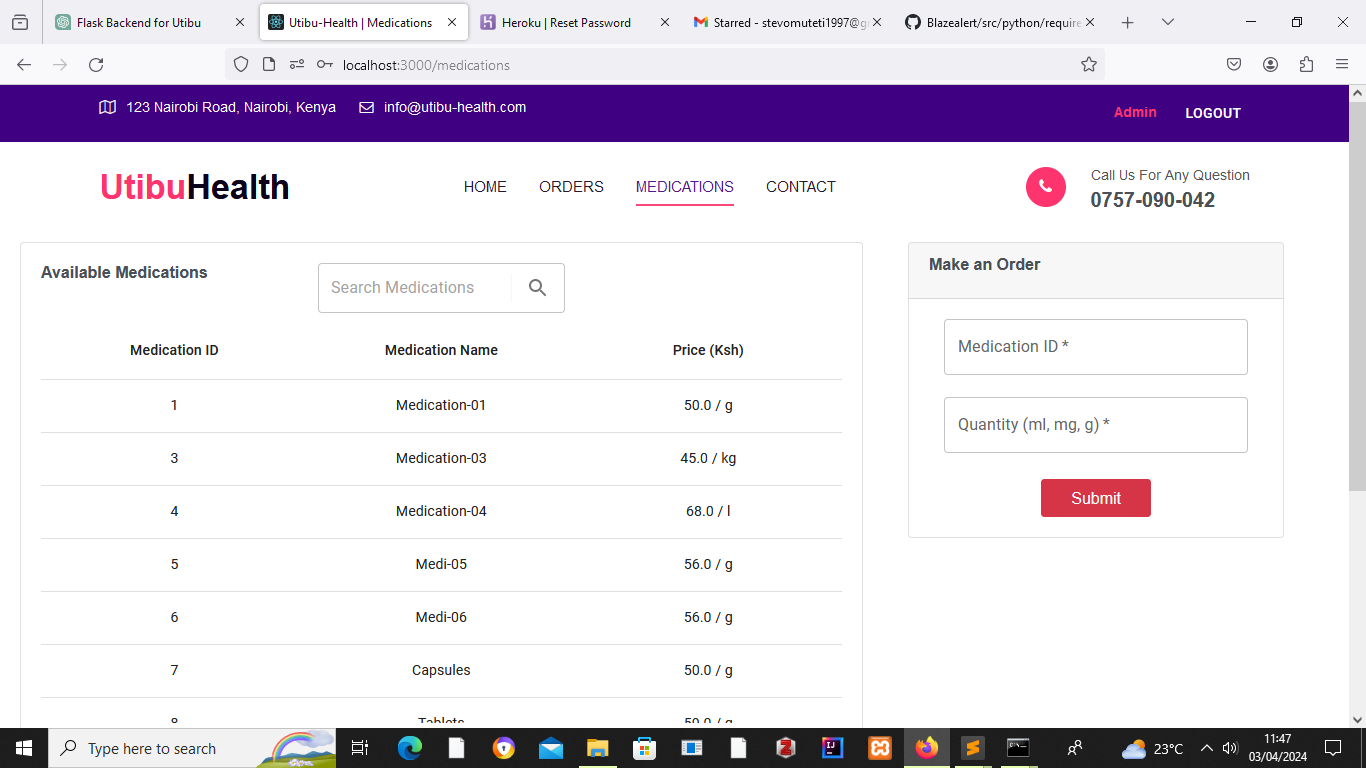


Figure 4: All medications

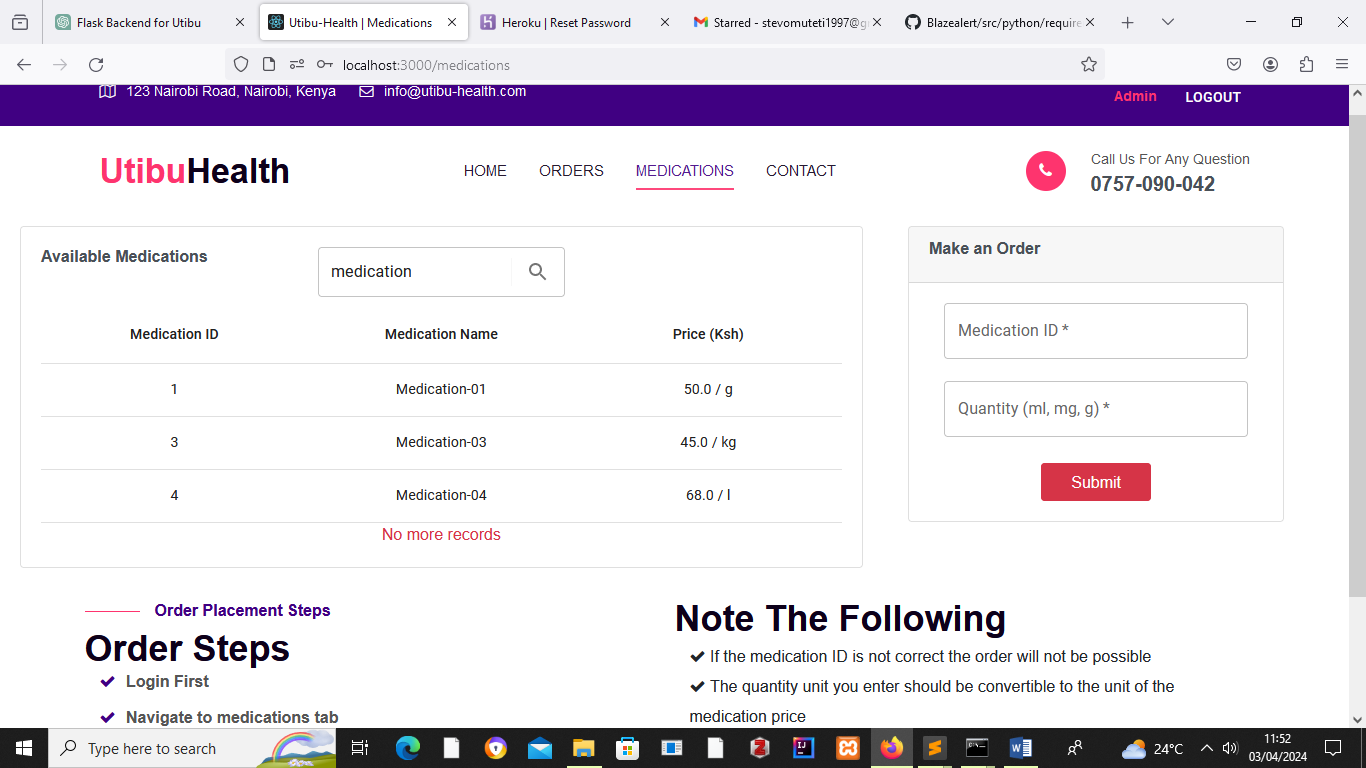


Figure 5: Filtered Medications

1. **Order Management**

Authenticated users can place, edit and cancel their own orders. For any actions to be performed on this section the user needs be logged in.

**Orders Table**

The previous orders by the currently authenticated user are displayed cleverly on a table. The details for a single order record/row include id, medication name, medication quantity, total price, order status, payment indicating whether the order has been paid for or not and actions which provides the cancel option for a pending order. The rendering of the orders on the User Interface(UI) is streamlined by pagination and the user can always load more orders at the click of a button. Users can also filter orders based on the date they were placed, their status or/and the medication’s name. All filters can be applied simultaneously for enhanced filtering/sieving.

It should be noted that users can only cancel orders whose status is pending. Once an order has been confirmed by the Admin(pharmacist) the client can only alter it by contacting the Admin(pharmacist).

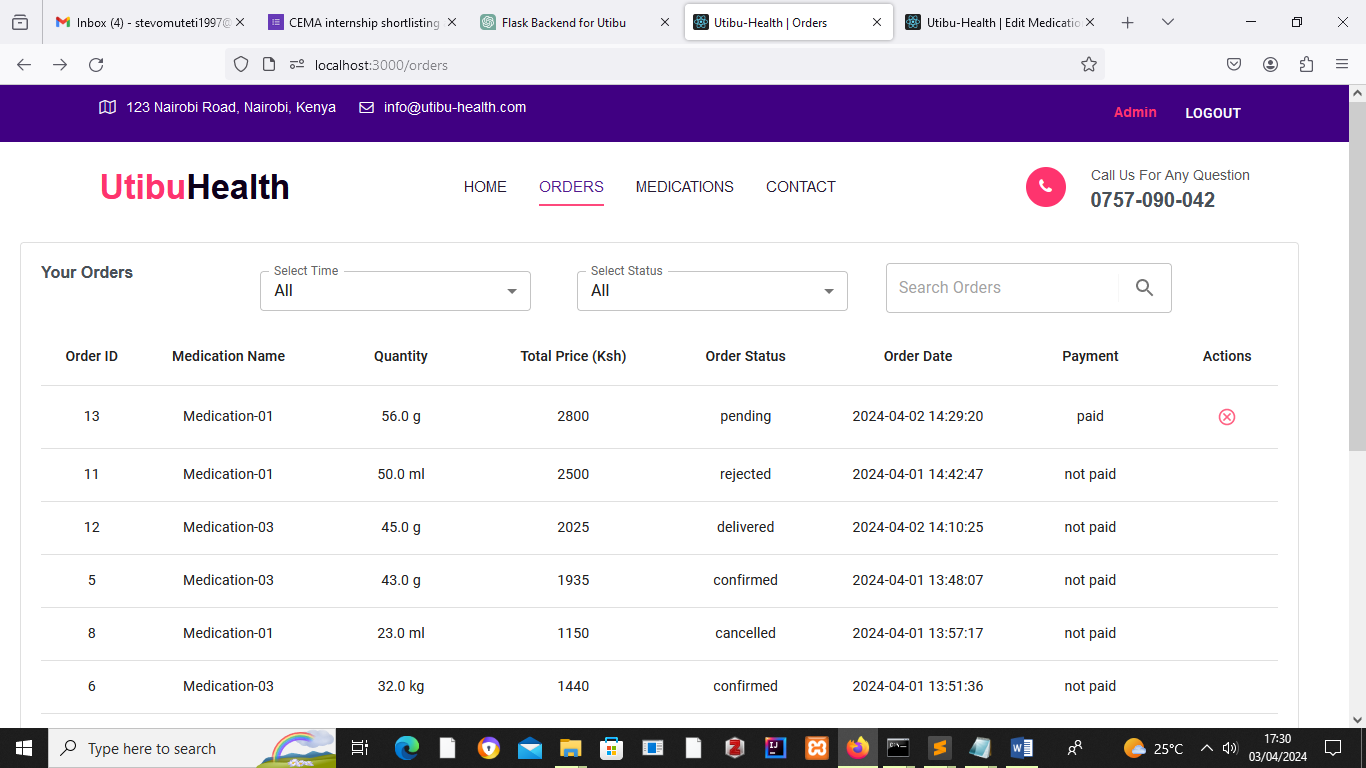


Figure 6: All orders

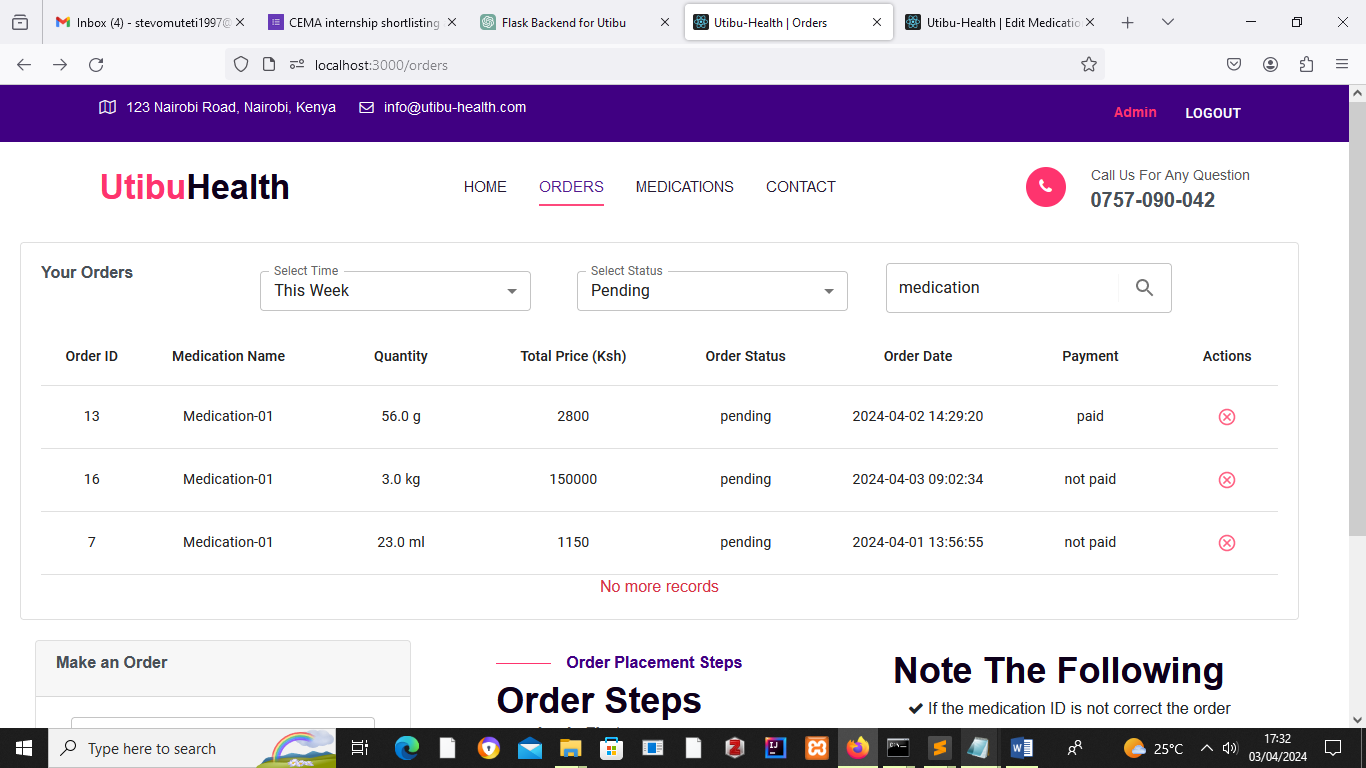


Figure 7: Filtered orders

**Order Form**

Clients place orders for medications they need. The medication’s id and the quantity **MUST** be entered correctly. The client has to ensure that the order quantity unit is convertible to the price unit of the particular medication they are trying to order. For example, if a medication’s price is Ksh. 50.00/g the client is only allowed to enter quantity in mass units. From the foregoing the following hold for the quantity field:

Quantity = 30ml is invalid since ml cannot be converted to g.

Quantity = 30mg is a valid value (mg can be converted to g)

Quantity = 1kg is valid because kg can be converted to g

The parsing of the quantity values has allowed the flexibility of entering the quantity and units as a single entity leaving the parsing to the backend. We save the client(user) the hurdle of strict input format by disregarding spaces during the parsing process. The following quantity field value formats will be accepted:

45 g, 45g, 45 g

When a correct order is submitted the total price is calculated based on the order medication quantity and medication price. Let’s walk through an example, suppose we enter a quantity of 3kg for a medication whose price is Ksh. 50.00/g. The price is calculated as in the steps below:

The order medication quantity is compared to the medication quantity in store and if the order quantity can be supplied using the currently available quantity it’s status is stored as confirmed and its order quantity subtracted from the medication’s store quantity and the remainder updated as the medication’s quantity. If the order quantity is more than the current medication’s quantity in the store its status is left as pending until the admin probably restocks the medication and confirms the order or rejecting the order if they are not in a position to service the order.

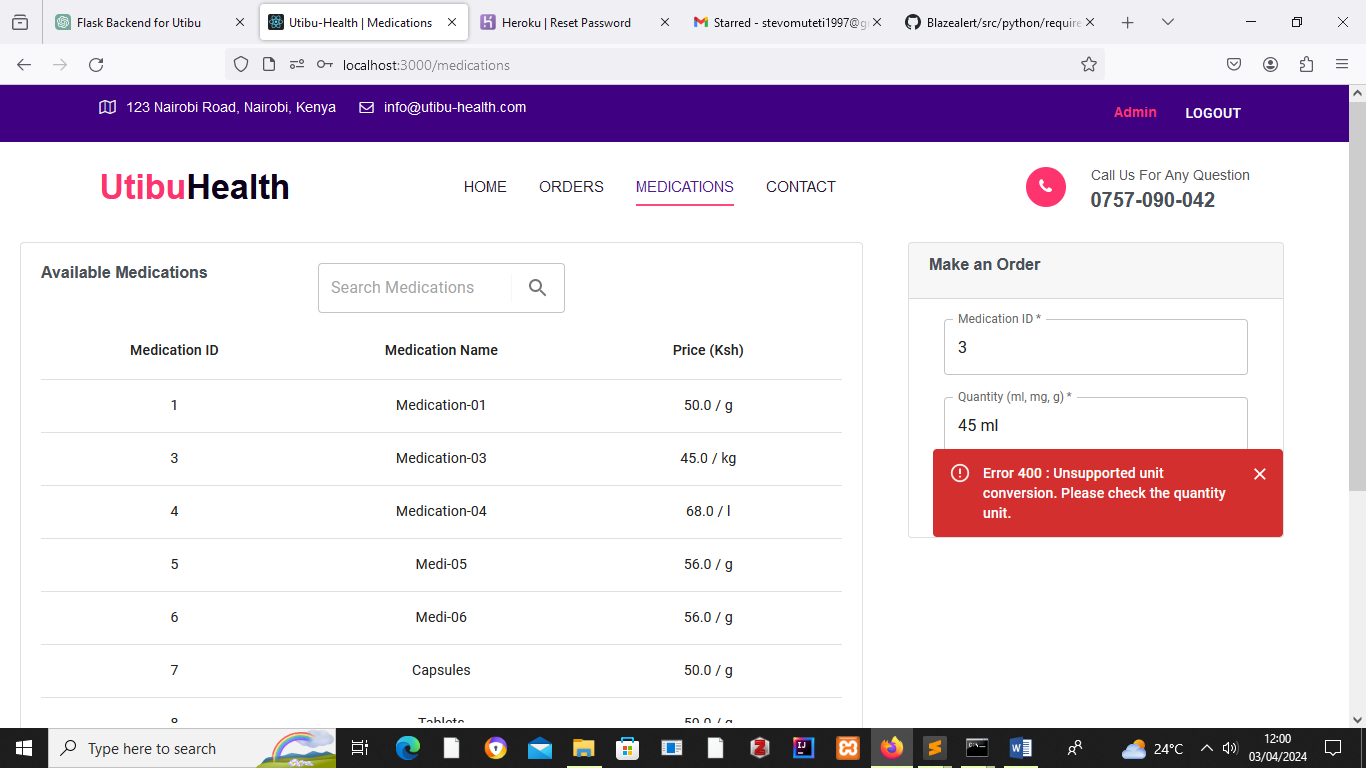


Figure 8: Wrong units for medication ID 3 Provided

1. **Checkout**

Upon a successful order placement, the application redirects to the checkout page. The order details are passed to the Receipt Form and filled appropriately. The filled form includes the order Id, the Medication Name, the Order Placement Date (Date for the same day), the Client’s Name, the quantity ordered and the total price of the order as calculated from the order quantity and medication price.

Here the client will be required to enter their phone number to finalize on the payment for the order. The phone number is validated for correct format (0 or +254 prefixes and 9 digits thereafter). They receive an STK push to their phone for confirmation. More payment options can also be considered.

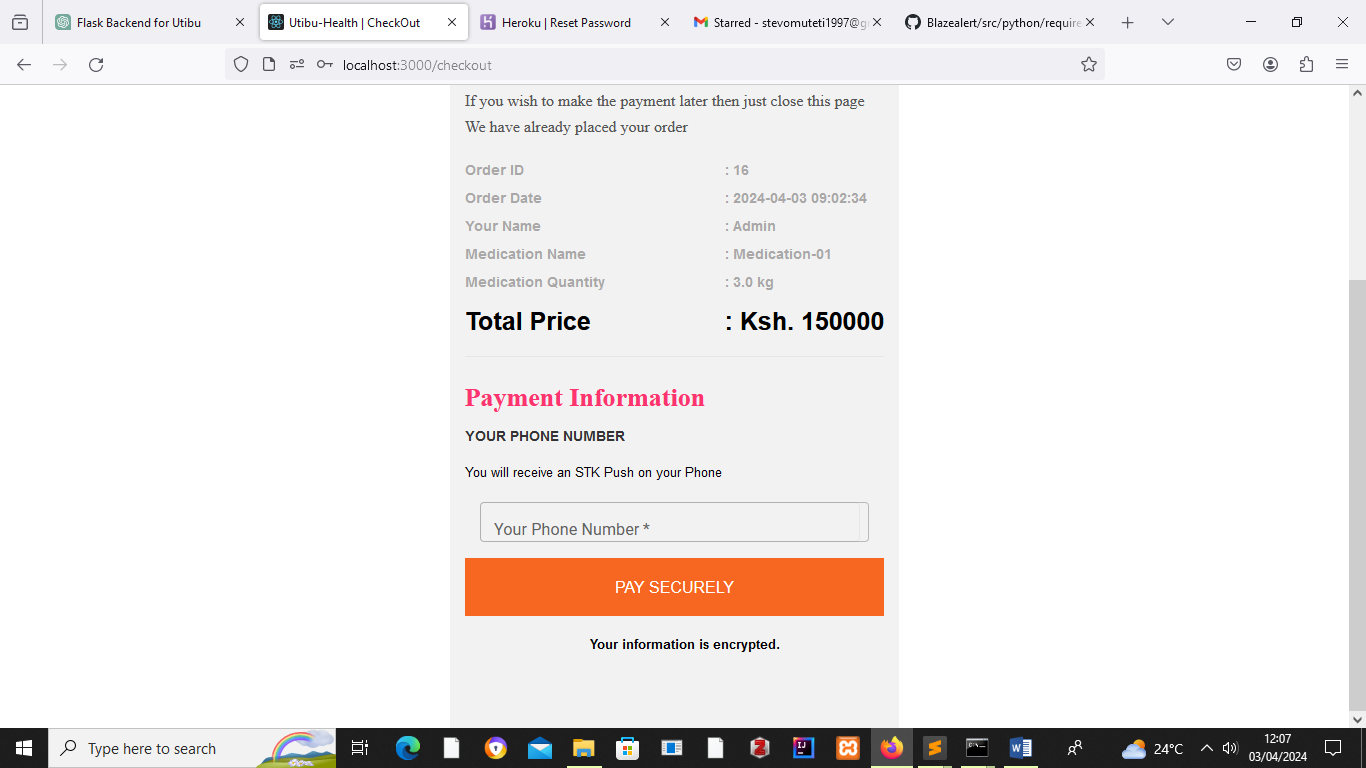


Figure 9: Checkout Step

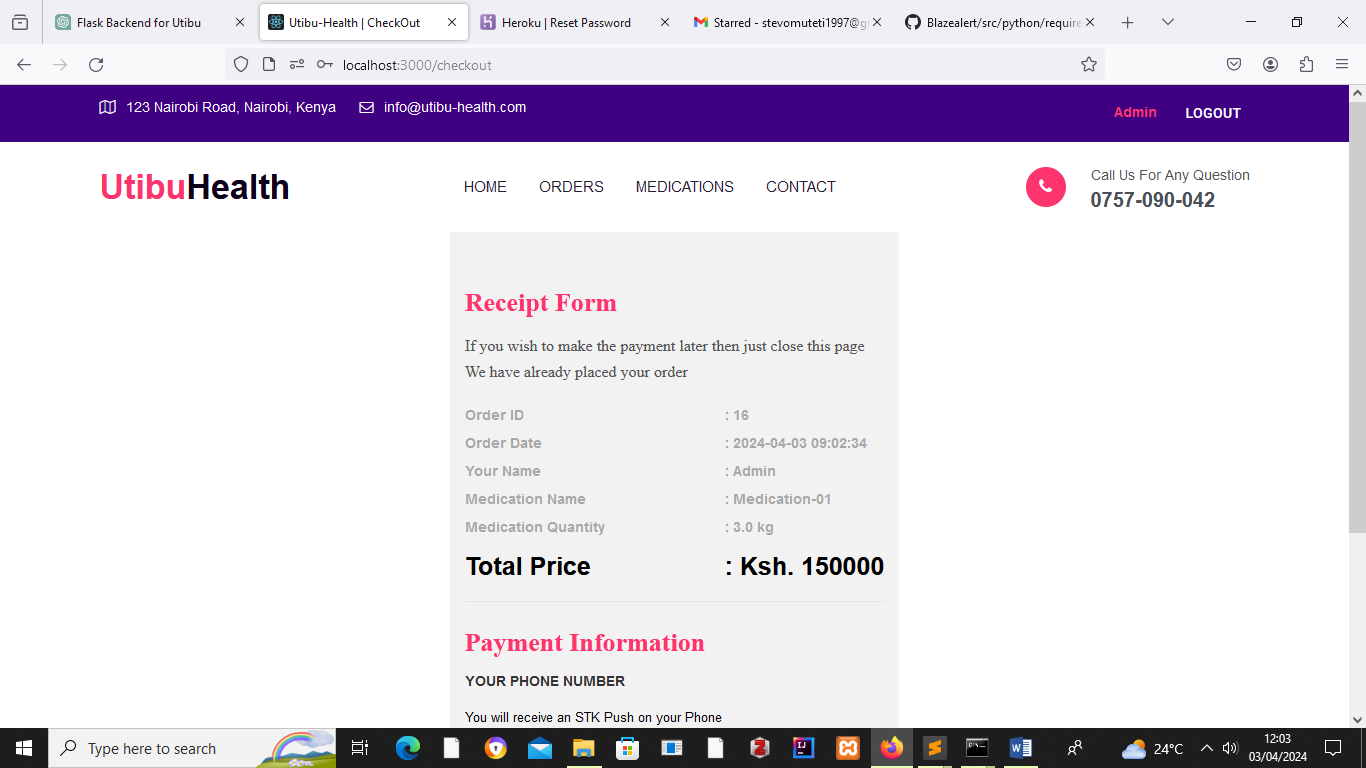


Figure 10: When 3kg is ordered for a medication(id 1) with a price of Ksh. 50.0/g

1. **Admin/Pharmacist**

The admin can access both the normal user and admin functionalities. Since we have already looked into what the normal user is capable of, lets delve into the admin privileges in this section. All the admin endpoints have been securely wrapped to require an authenticated admin. If this criterion is not met the application completely denies access to the particular resource.

1. **Medication Management**

The admin can add, edit and/or delete a medication.

**Medication Table**

This table displays all the medications already available in the application database and consequently in the pharmacy. The displayed details for any record are id, name, quantity, price and actions which provides the admin the edit and deletion privileges for a medication. The display is streamlined by a pagination of 10 records with more records available at the click of a button. The medications can be filtered based on the medication id, and/or availability in store.

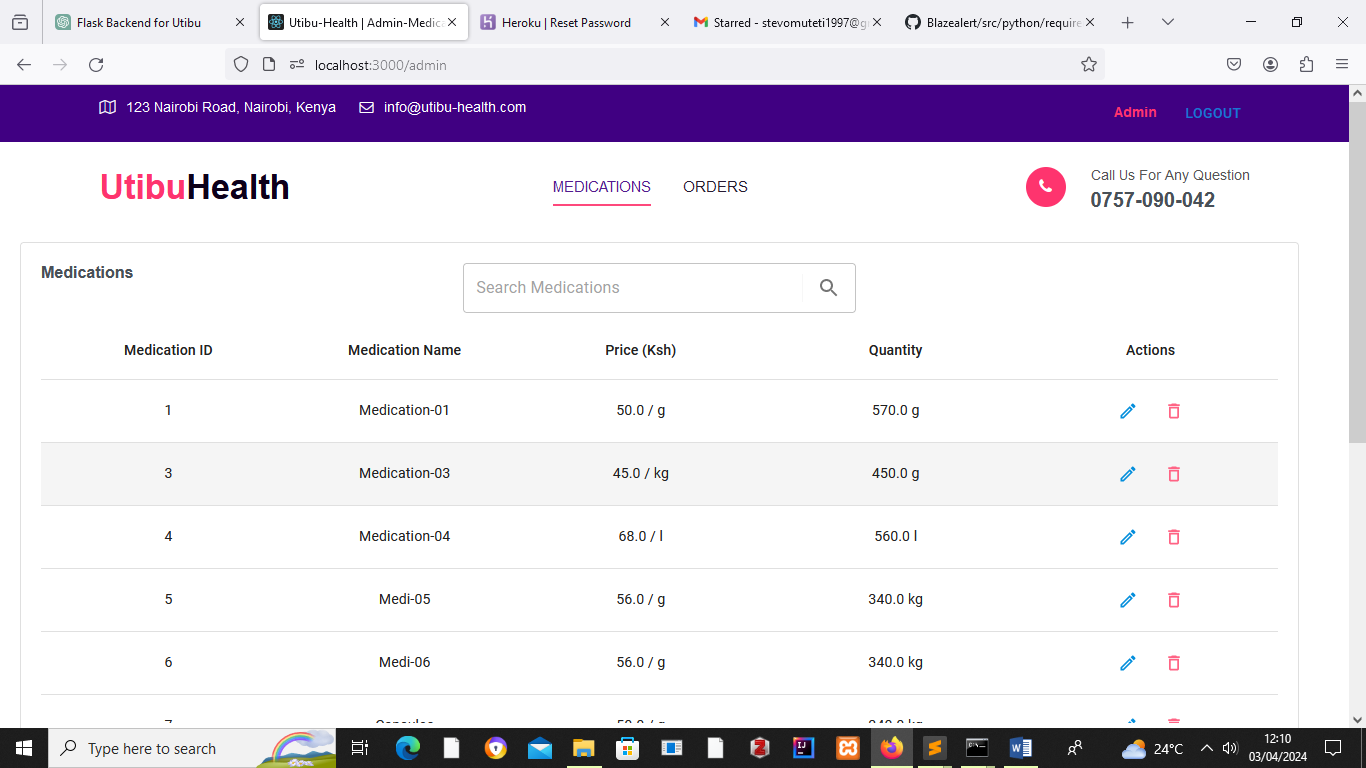


Figure 11: All medications

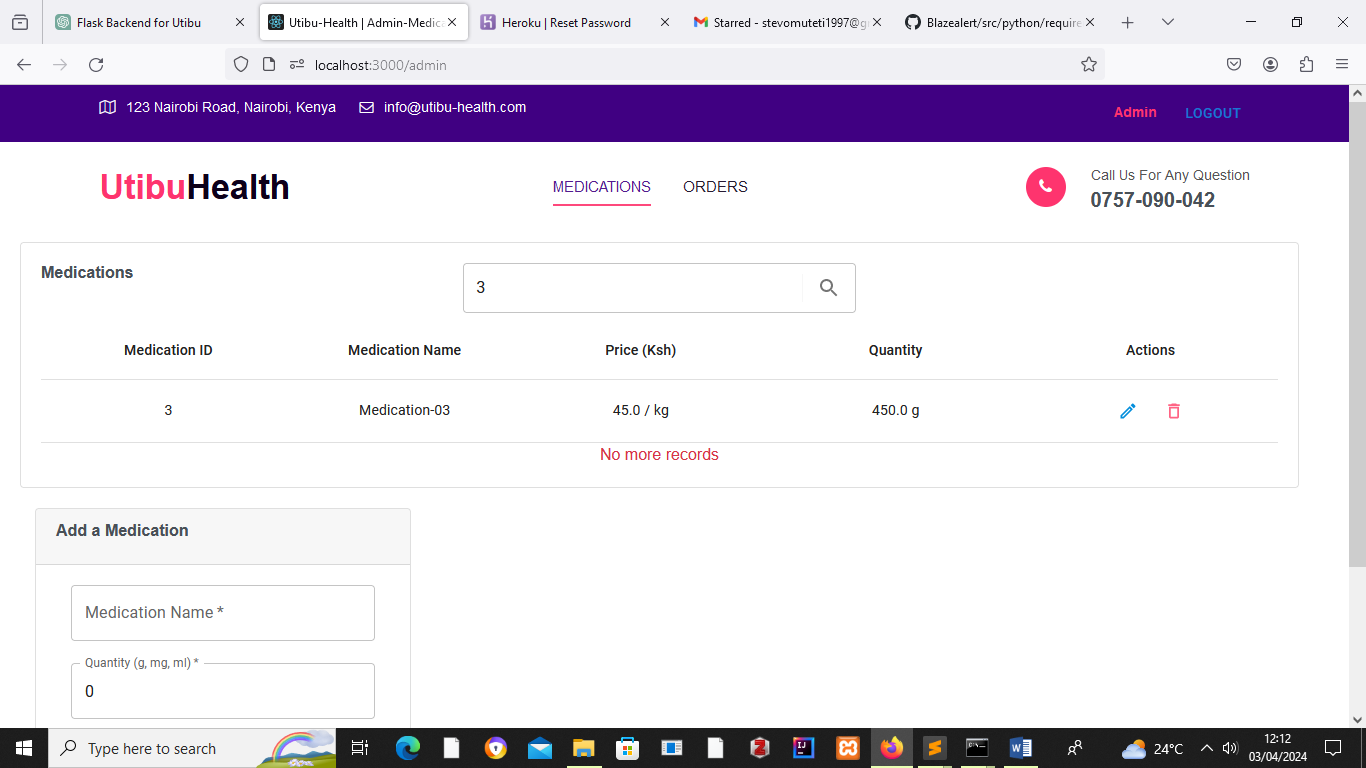


Figure 12: Medication with ID 3 filtered

**Medication Form**

This is the form the admin(pharmacist) uses in adding new medications to the system database. The admin is only required to provide the medications name, the current quantity and the price. Although the admin has been provided the flexibility of entering the values luxuriously, there are certain rules just to ensure data integrity within the system. The quantity should be entered correctly in the format `value unit`. The following examples are correct.

30kg, 45ml, 45 ml

Spaces do not matter as the parsing technique is smart enough to ignore any invalid spaces between the value and the unit. However, spaces within any of them does matter as this may completely alter the meaning of the data.

Similarly, the price should be entered in the format `value/unit`. Spaces immediately before or after the sign do not matter. Valid examples include the following:

30/g, 30/ g, 30 / g etc.

The backend will always separate these and store them as separate attributes. That is either of quantity or price it stores the value and unit separately.

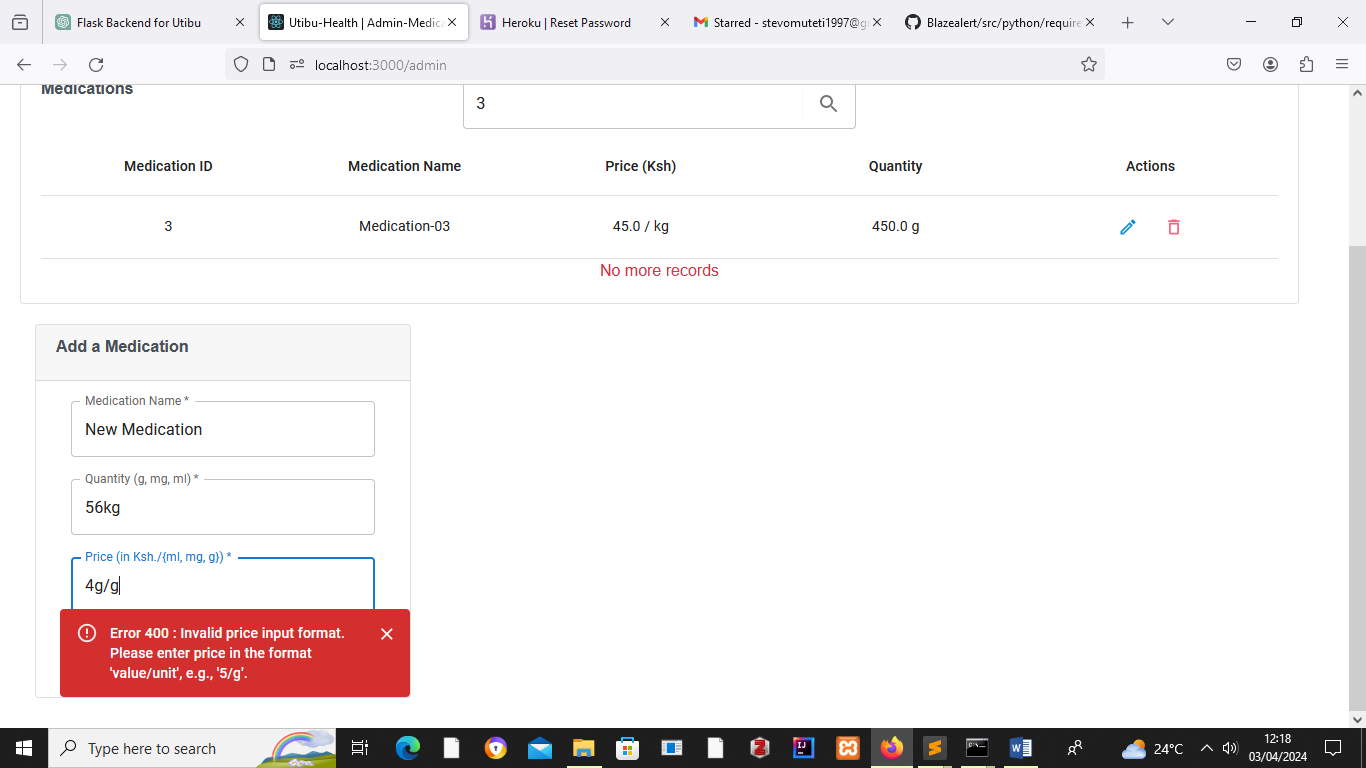
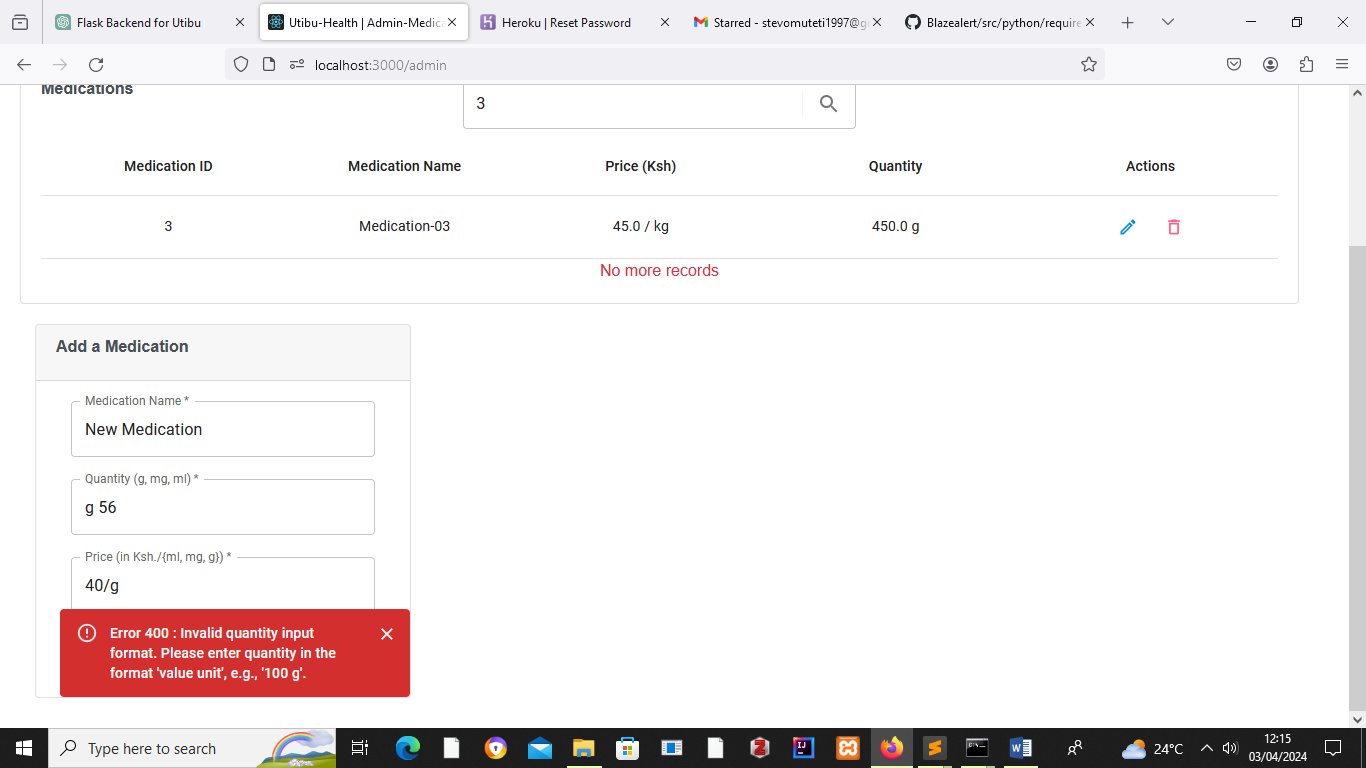


Figure 13: Wrong quantity and price formats provided

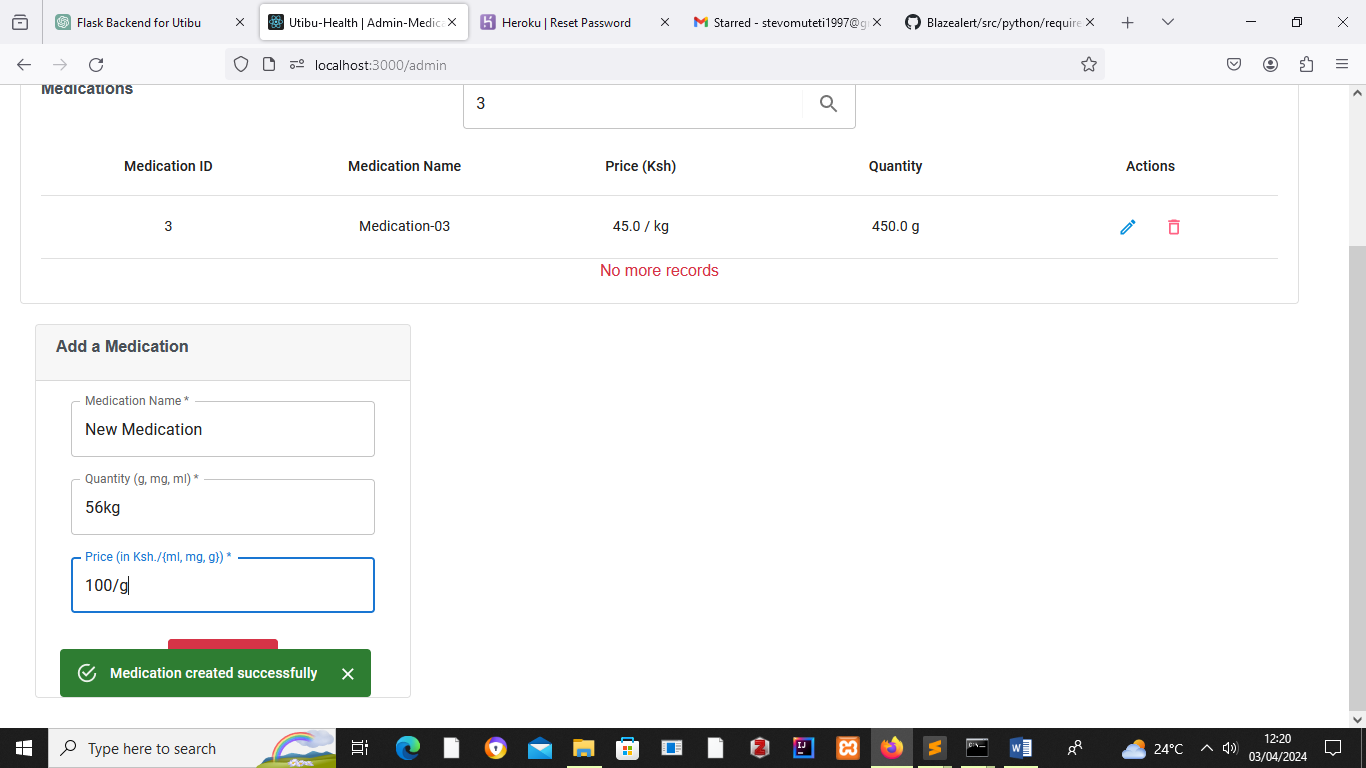


Figure 14: A successful medication addition

**Medication Edit Form**

For this the values should be altered and conformity to the formats stated above ensured. Let’s edit the newly added medication and check its record in the medications table for verification. We set the new values as in the screenshot below.

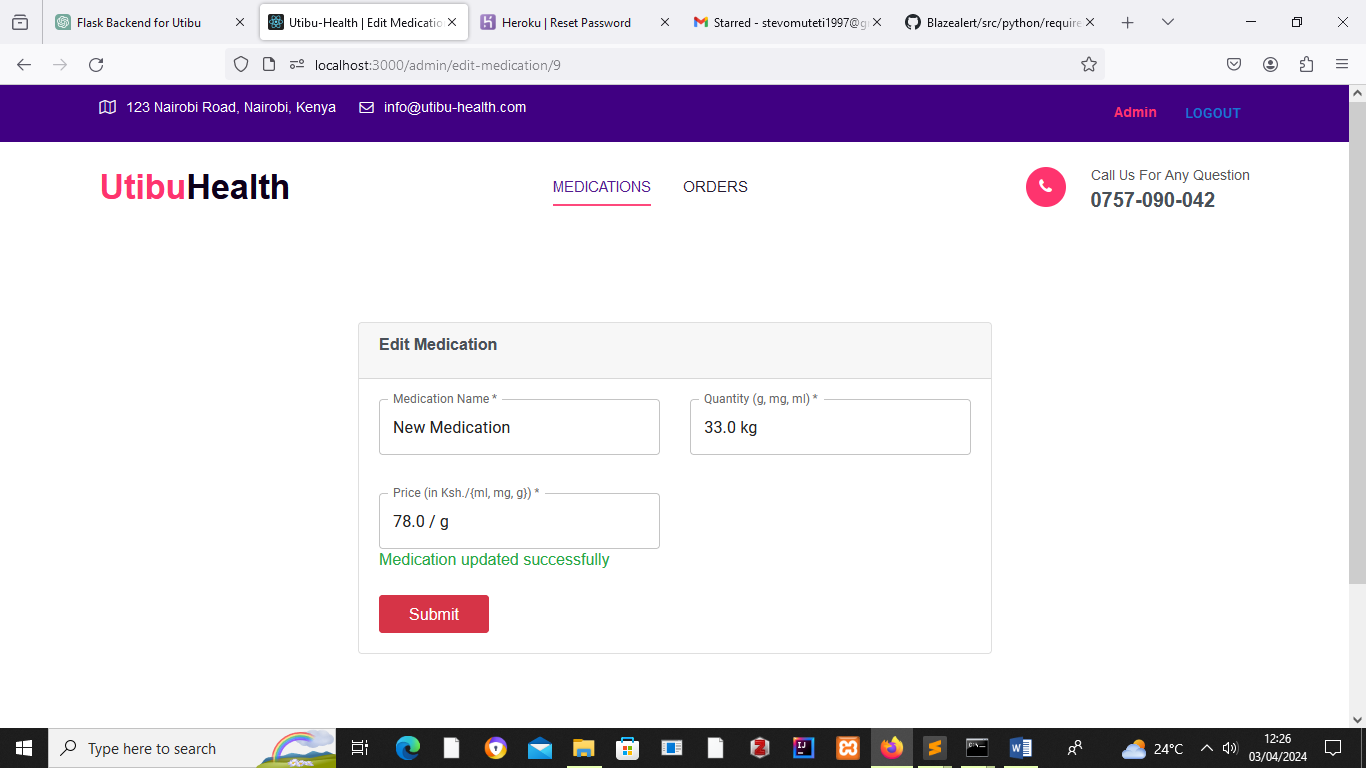


Figure 15: Medication edit

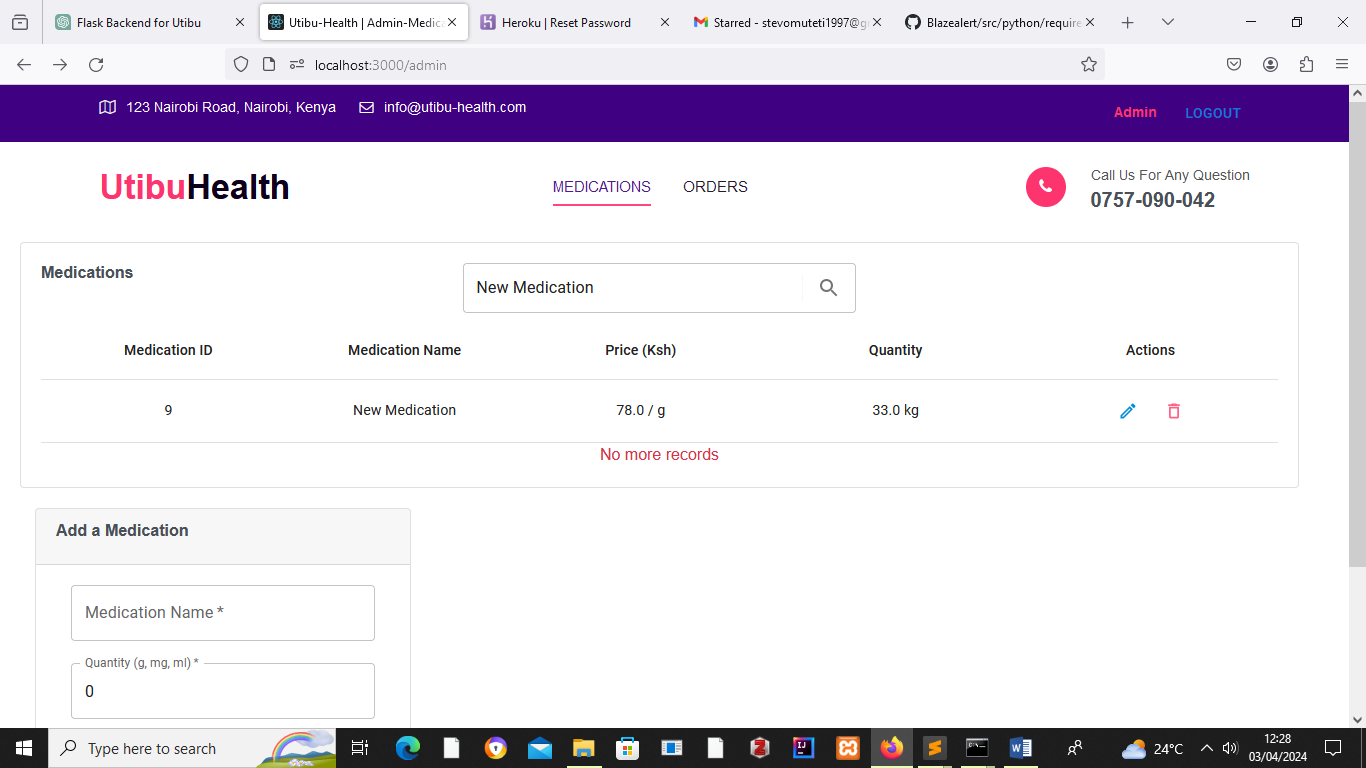


Figure 16: Verifying that the edit was actually successful

1. **Order Management**

This section constitutes a table displaying all the records for orders by different users/clients. Pagination is also implemented to minimize initial load times for a large number of order records. The order records can be filtered based on the order id, the placement date, the status, the client’s name and the medication name. Alongside the order status value is a dropdown enabling the admin to update the order status. The status update option is not available for rejected, cancelled or delivered orders.

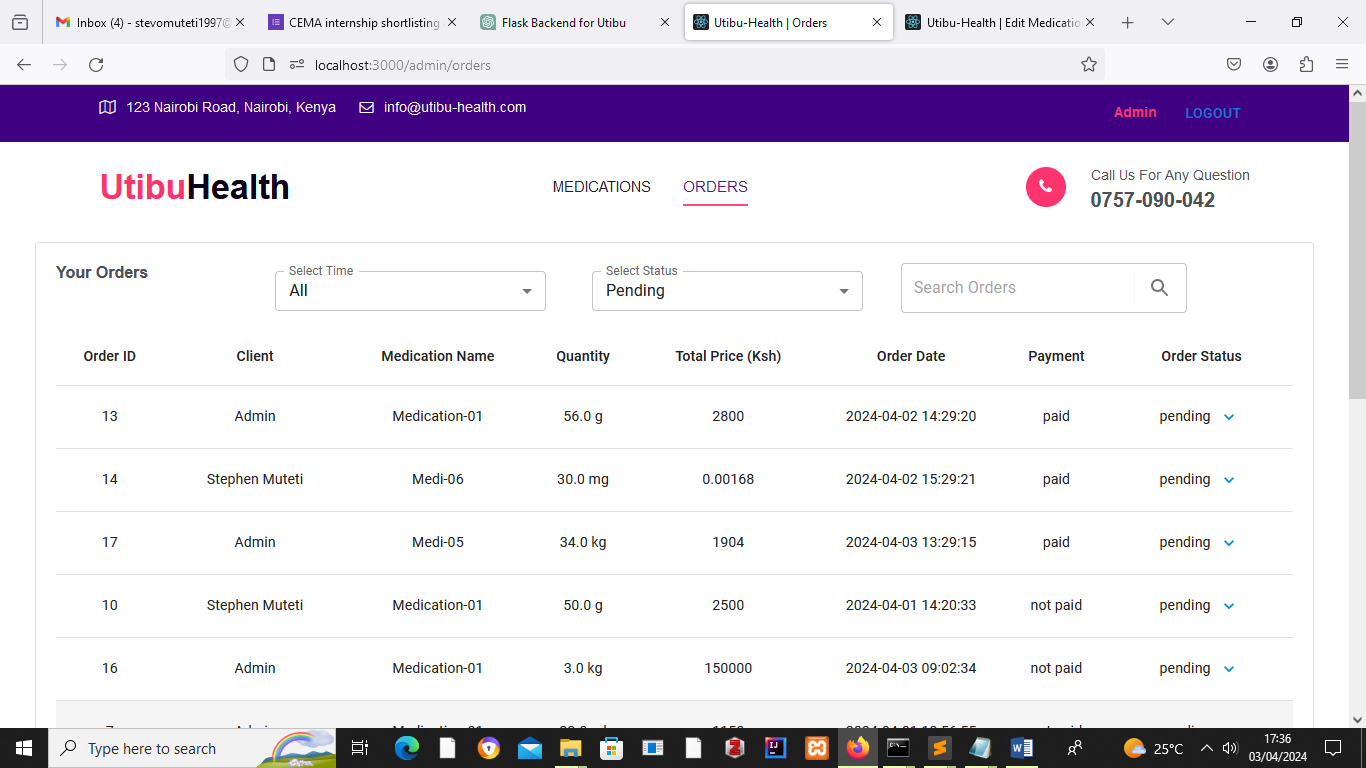


Figure 17: Showing all pending orders

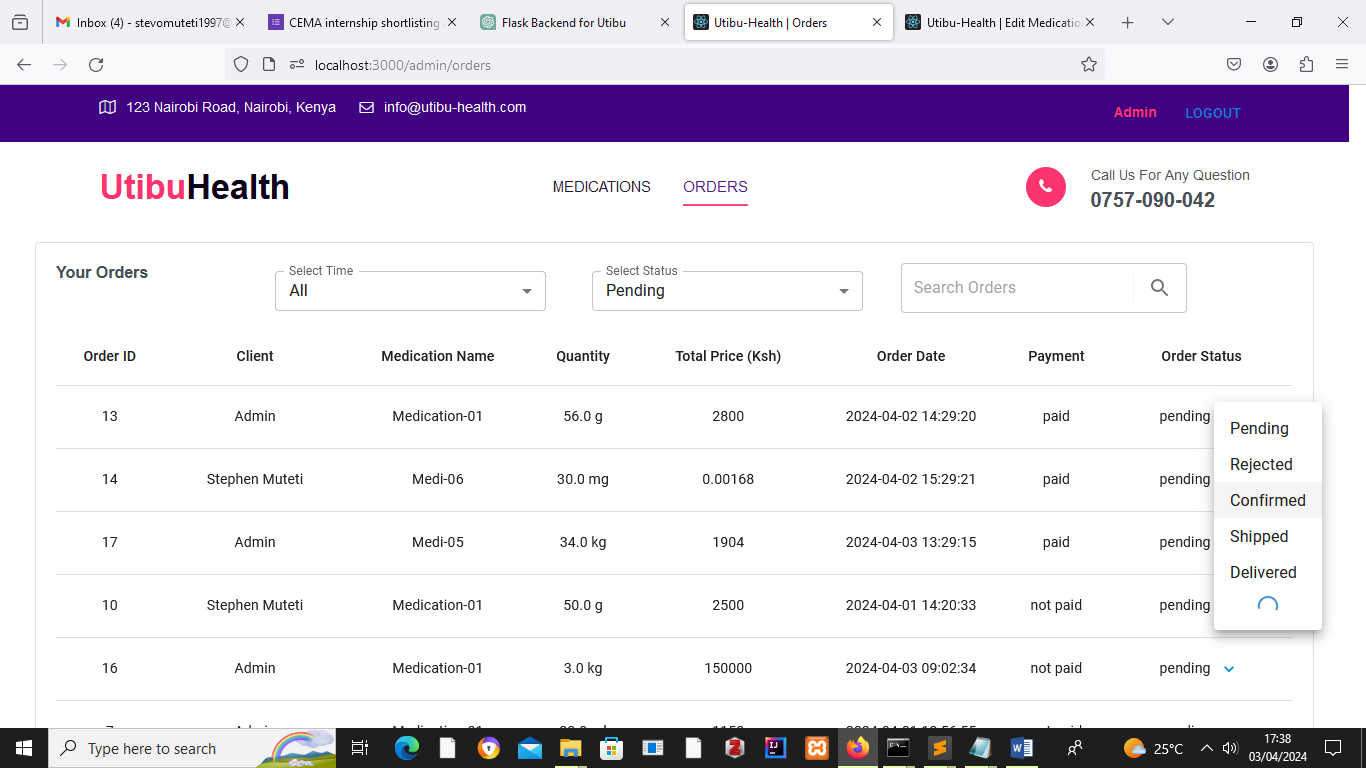


Figure 18: Showing the status change menu and updating an order (notice the progressbar)

1. **Payments Management**

This is a table displaying all the payments by clients for different orders. The details displayed include the order id, the amount and the payment date. The records are paginated and the payment for a particular order can be filtered by use of the order id.

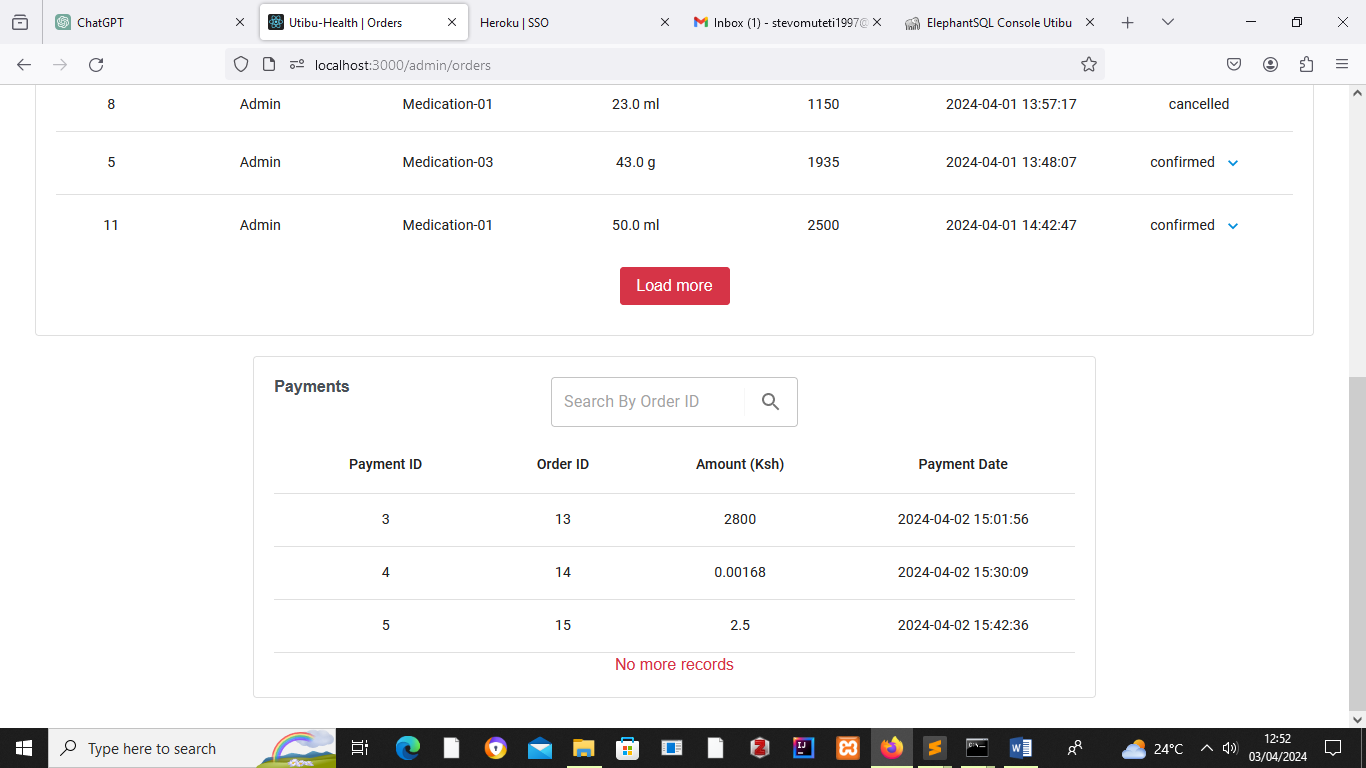


Figure 19: payments table

**performance and sacalability considerations**

**Database Optimization**

SQLAlchemy not only simplifies database interactions but also enhances performance through features such as database connection pooling. By leveraging connection pooling, the application mitigates the overhead of repeatedly opening and closing database connections, thereby improving overall database performance and resource utilization.

**AWS PostgreSQL Database**

Leveraging a cloud-based PostgreSQL database hosted on Amazon Web Services (AWS) provided scalability and reliability benefits. AWS offers scalable infrastructure resources, allowing the database to dynamically adjust to varying workloads. This ensured that the application could accommodate growing data volumes and increasing user traffic without compromising performance.

**Flask Framework**

Flask's lightweight and modular design enabled the development of a streamlined backend API. This minimized processing overhead and resource consumption, contributing to improved application performance. Additionally, Flask's asynchronous request handling capabilities further enhanced scalability by efficiently managing concurrent requests.

**CORS Configuration**

Implementing Cross-Origin Resource Sharing (CORS) allowed the frontend client to securely access resources from the backend server hosted on a different domain. Proper CORS configuration ensured efficient communication between the client and server, enhancing overall system performance and responsiveness.

**JWT Authentication**

JSON Web Token (JWT) authentication provided a secure and efficient mechanism for user authentication. JWT tokens are lightweight and stateless, reducing server overhead during authentication processes. This contributed to faster response times and improved system scalability, particularly during user login and authorization operations.