`

Project Final Report: PharmFetch

University of Calgary
Course: CPSC 471

Lisa Tatarnikov (30227087) lisa.tatarnikov@ucalgary.ca
Dipti Kumar (30213859) dipti.kumar@ucalgary.ca
Jahnissi Nwakanma (30174827) jahnissi.nwakanma@ucalgary.ca

CPSC 471: Project Report

Abstract

Accessing prescription and over the counter medication can be challenging for many individuals, especially for the elderly and infirm. Despite the availability of prescription delivery services, over the counter medications delivery remains limited and unreliable, often coupled with inflated cost and no guarantee. Our proposed solution is to create a smooth delivery system that integrates prescription and over the counter delivery. This addresses the gaps in the current system by offering a convenient and efficient way for users to access essential medications without leaving their homes. This project is motivated by the need to reduce the burden on individuals who may be too ill to visit a pharmacy, helping them recover in the comfort of their home. Using a MySQL database hosted on Azure, the app allows for customers, delivery drivers, and pharmacies to create accounts and place, prepare, or deliver orders to ease the delivery of medication to those who need them.

Introduction

The convenience of access to prescription and over-the-counter medication is an issue that most will encounter at some point in their lives. Elderly, disabled, or ill individuals often struggle to get to a pharmacy due to mobility issues, time constraints, or cannot withstand a trip due to their health. There do exist pharmacies where one can have their prescription delivered. However, many people do not only take prescription medications, but also over the counter medication or supplements. The only option for delivery over-the-counter medication are services like UberEats and DoorDash. However, both services offer a limited variety of over-the-counter medication at an inflated cost. Further, the delivery of any ordered over-the-counter medication is not guaranteed since the shopper provided by the service may be unable to find the ordered product, so the customer is refunded instead. Current systems for medication delivery only partially address these issues. Consequently, individuals who need prescriptions and/or over-the-counter medication delivered are left dissatisfied, especially those in urgent situations like illness or recovery.

Project Design + ER Diagram

PharmFetch was built to reflect how a real-world pharmacy delivery system would function, with a focus on keeping the experience clear, organized, and accessible for everyone involved. The system supports five main user types: customers, doctors, pharmacies, delivery drivers, and admins. Each user has their own dedicated portal with specific permissions and tools based on what they actually need to do. Customers can register, log in, and place orders for either prescription or over-the-counter medication. To keep the order flow simple and manageable, customers can only order one type of medication per order. They can also upload prescriptions as needed and view the status of current and past orders.

- Doctors can upload new prescriptions for registered customers, which are automatically linked to the appropriate customer profile. Doctors also have access to a dashboard where they can view all the prescriptions they have submitted and filter them by customer. This makes it easy to track prescription history and follow up if needed.

- Pharmacies receive and review new orders placed by customers. A pharmacy must accept an order before it becomes available to drivers for pickup. This step ensures that only verified and accepted orders are delivered. Pharmacies can also view their in-progress orders as well as completed deliveries to keep track of their full order history.

- Drivers can view a list of accepted orders that are ready for pickup. Once they accept an order, it moves to their in-progress deliveries. Drivers are able to view all current and past orders they have picked up and are responsible for marking an order as delivered once it has reached the customer.

- Admins have access to a simple backend dashboard where they can manage the system's users, including customers, drivers, and pharmacies. Their main role is to keep things running smoothly by updating or removing user records when necessary, whether that means correcting information, handling issues, or cleaning up inactive accounts.

Each interface was designed to be role specific, so users only see what they actually need to interact with. This keeps the experience clear and prevents confusion. Customers aren't dealing with delivery tools, and drivers aren't exposed to prescription uploads. The system is also structured to scale efficiently. Whether it is used by a single pharmacy or expanded to serve multiple locations, the architecture holds up without requiring major changes.

Implementation

The relational model for PharmFetch was created by following the standard ER-to-relational schema conversion process. Each entity in the EERD was translated into a table, with appropriate primary keys and foreign keys used to maintain relationships. For many-to-many relationships, we created separate tables to represent the connections clearly. For example, Includes links medications to customer orders, and Stocks tracks which pharmacies carry which medications and in what quantity.

For weak entities like Prescription, we used a composite primary key made up of prescription_id, doc_id, and customer_id. This ensures that each prescription is uniquely tied to both the doctor who issued it and the customer it was issued for. We also created a Prescription_References table to store the specific medications associated with a given prescription, which keeps the data organized and prevents duplication.

<u>Changes from the EERD</u>

- Admin table added username and password as attributes

- In customer, Priority is now a subtype of regular customer

- Driver table added username, password and admin id as attribute
- Pharmacy added username and password
- Medication was renamed to med
- Doctor was renamed to Doc and added username and password as attribute
- Doctor added username and password as attributes
- Order was renamed to Customer order, and added prescription id as attribute
- Prescription added doc_id and customer id as part of primary key
- Payment table was renamed as transaction table
- A new relationship was formed from customer order to prescription, where a customer order can directly reference the prescription record of that customer

Design Decisions

One of the most important implementation choices we made was how to handle over-the-counter medications. Because these do not require a prescription, we made the prescription_id in the Customer_Order table optional. This allows customers to place orders for non-prescription medications without having to attach a prescription, while still supporting full prescription records for orders that do need them.

To manage data cleanly across different parts of the system, we created separate relationship tables like Includes and Stocks. Includes records which medications are part of each order, while Stocks links medications to the pharmacies that carry them. This structure helped organize the data without redundancy and kept the relationships between entities clear, especially as we built out the backend logic and connected the database to the app's functionality.

These design choices helped keep the schema organized, intuitive, and aligned with how the system functions in practice. It supports the full process of placing orders, verifying prescriptions, and managing deliveries without unnecessary complexity.

Database Management System

We chose to use MariaDB because it integrates smoothly with our backend and supports all the features we needed for this project. It allowed us to define foreign key constraints, use composite keys where necessary, and run complex queries reliably. Hosting the database on Azure also gave us consistent access to test and use live data during development. The system performed well and allowed us to implement the full functionality we designed without limitations.

SQL Statements

```
CREATE DATABASE PharmFetch;

CREATE TABLE Admin (
  AdminId INT NOT NULL,
  Fname VARCHAR(255),
  Lname VARCHAR(255),
  PhoneNum VARCHAR(20),
  Email VARCHAR(255),
  PRIMARY KEY (AdminId)
);

CREATE TABLE Driver (
  DriverId INT NOT NULL AUTO_INCREMENT,
  Username VARCHAR(255),
  Password VARCHAR(255),
  LicenseNumber VARCHAR(100),
  Fname VARCHAR(255),
  Lname VARCHAR(255),
  Email VARCHAR(255),
  PhoneNum VARCHAR(20),
  Availability VARCHAR(255),
  AdminID INT,
  PRIMARY KEY (DriverId),
  FOREIGN KEY (AdminID) REFERENCES Admin(AdminID)
);

CREATE TABLE RegularCustomer (
  CustomerId INT NOT NULL AUTO_INCREMENT,
  Username VARCHAR(255),
  Password VARCHAR(255),
  Fname VARCHAR(255),
  Lname VARCHAR(255),
  Email VARCHAR(255),
  Address VARCHAR(255),
  PhoneNum VARCHAR(20),
  AdminID INT,
  PRIMARY KEY (CustomerId),
  FOREIGN KEY (AdminID) REFERENCES Admin(AdminID)
);

CREATE TABLE PriorityCustomer (
```

```
  CustomerId INT NOT NULL,
  MembershipId INT NOT NULL,
  Fname VARCHAR(255),
  Lname VARCHAR(255),
  Email VARCHAR(255),
  Address VARCHAR(255),
  AdminID INT,
  CONSTRAINT PriorityCustomerID_pk PRIMARY KEY (CustomerId, MembershipId),
  FOREIGN KEY (AdminID) REFERENCES Admin(AdminID)
);

CREATE TABLE Pharmacy (
  PharmacyId INT NOT NULL AUTO_INCREMENT,
  Username VARCHAR(255),
  Password VARCHAR(255),
  Name VARCHAR(255),
  Address VARCHAR(255),
  Email VARCHAR(255),
  PhoneNum VARCHAR(20),
  OperatingHours VARCHAR(255),
  AdminID INT,
  PRIMARY KEY (PharmacyId),
  FOREIGN KEY (AdminId) REFERENCES Admin(AdminId)
);

CREATE TABLE Medication (
  MedicationId INT NOT NULL AUTO_INCREMENT,
  Name VARCHAR(255),
  Description VARCHAR(255),
  Price DECIMAL(10,2),
  Quantity INT,
  PRIMARY KEY (MedicationId)
);

CREATE TABLE Doctor (
  DoctorId INT NOT NULL AUTO_INCREMENT,
  Username VARCHAR(255),
  Password VARCHAR(255),
  Fname VARCHAR(255),
  Lname VARCHAR(255),
  PRIMARY KEY (DoctorId)
);

CREATE TABLE Prescription (
```

```
  PrescriptionId INT NOT NULL AUTO_INCREMENT,
  DoctorId INT,
  CustomerId INT,
  MedicationId INT,
  Dosage VARCHAR(255),
  Instructions VARCHAR(255),
  PRIMARY KEY (PrescriptionId),
  FOREIGN KEY (DoctorId) REFERENCES Doctor(DoctorId),
  FOREIGN KEY (CustomerId) REFERENCES RegularCustomer(CustomerId),
  FOREIGN KEY (MedicationId) REFERENCES Medication(MedicationId)
);

CREATE TABLE Customer_Order (
  OrderId INT NOT NULL AUTO_INCREMENT,
  OrderDate DATE NOT NULL,
  DriverId INT,
  CustomerId INT NOT NULL,
  PharmacyId INT NOT NULL,
  PrescriptionId INT,
  MedicationId INT,
  OrderStatus VARCHAR(255),
  FOREIGN KEY (DriverId) REFERENCES Driver(DriverId),
  FOREIGN KEY (CustomerId) REFERENCES RegularCustomer(CustomerId),
  FOREIGN KEY (PharmacyId) REFERENCES Pharmacy(PharmacyId),
  FOREIGN KEY (PrescriptionId) REFERENCES Prescription(PrescriptionId),
  FOREIGN KEY (MedicationId) REFERENCES Medication(MedicationId),
  PRIMARY KEY (OrderId)
);

CREATE TABLE Stocks (
  PharmacyId INT NOT NULL,
  MedicationId INT NOT NULL,
  FOREIGN KEY (PharmacyId) REFERENCES Pharmacy(PharmacyId),
  FOREIGN KEY (MedicationId) REFERENCES Medication(MedicationId),
  PRIMARY KEY (PharmacyId, MedicationId)
);
```

<u>Customers</u>

1. Add a new customer

```
INSERT INTO RegularCustomer (Username, Password, Fname, Lname, Email, Address, PhoneNum, AdminId) VALUES ('exampleUser', 'pass123', 'Example', 'Example', 'example@gmail.com', '123 Example St', '1234567890', 1);
```

2. Edit customer information

UPDATE RegularCustomer SET Email = 'newExample@gmail.com' WHERE CustomerId = 1111;

3. Search for a customer

SELECT * FROM RegularCustomer WHERE CustomerId = 1111;

4. Delete a customer

DELETE FROM RegularCustomer WHERE CustomerId = 1111;

Doctors

1. Add a new doctor

INSERT INTO Doctor (Username, Password, Fname, Lname) VALUES ('docUser', 'docpass', 'First', 'Last');

2. Search for a doctor

SELECT * FROM Doctor WHERE DoctorId = 1234;

3. Edit doctor information

UPDATE Doctor SET Lname = 'NewLastName' WHERE DoctorId = 1234;

4. Delete doctor

DELETE FROM Doctor WHERE DoctorId = 1234;

Drivers

1. Add a new driver

INSERT INTO Driver (Username, Password, LicenseNumber, Fname, Lname, Email, PhoneNum, Availability, AdminId) VALUES ('driver1', 'pass', 'LIC123', 'First', 'Last', 'driver@gmail.com', '1234567890', 'Available', 1);

2. Search for a driver

SELECT * FROM Driver WHERE DriverId = 1234;

3. Edit driver information

UPDATE Driver SET Availability = 'Unavailable' WHERE DriverId = 1234;

4. Delete driver

DELETE FROM Driver WHERE DriverId = 1234;

Pharmacy

1. Add new pharmacy

INSERT INTO Pharmacy (Username, Password, Name, Address, Email, PhoneNum, OperatingHours, AdminId) VALUES ('pharm1', 'pharmpass', 'PharmaPlace', '123 Health St', 'pharma@example.com', '9876543210', '9AM–9PM', 1);

2. Edit pharmacy information

```
UPDATE Pharmacy SET Email = 'updated@pharmacy.com' WHERE PharmacyId = 123;
```

3. Search for a pharmacy

```
SELECT * FROM Pharmacy WHERE PharmacyId = 123;
```

4. Delete pharmacy

```
DELETE FROM Pharmacy WHERE PharmacyId = 123;
```

Medication

1. Add medication

```
INSERT INTO Medication (Name, Description, Price, Quantity) VALUES ('Advil', 'Pain Reliever', 7.99, 100);
```

2. Edit medication

```
UPDATE Medication SET Quantity = 90 WHERE MedicationId = 1;
```

3. Search medication

```
SELECT * FROM Medication WHERE Description LIKE '%Pain%';
```

4. Delete medication

```
DELETE FROM Medication WHERE Quantity = 0;
```

Orders

1. Place order for prescription or over-the-counter medication

```
INSERT INTO Customer_Order (OrderDate, CustomerId, PharmacyId, MedicationId, OrderStatus) VALUES (CURDATE(), 111, 123, 2, 'PENDING');
```

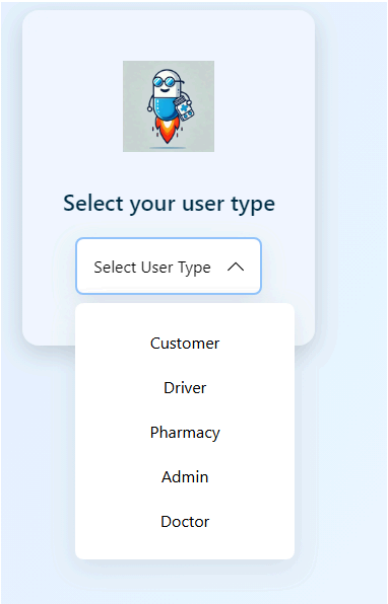
2. Assign order to driver

```
UPDATE Customer_Order SET DriverId = 456 WHERE OrderId = 789;
```

3. Group orders by pharmacy

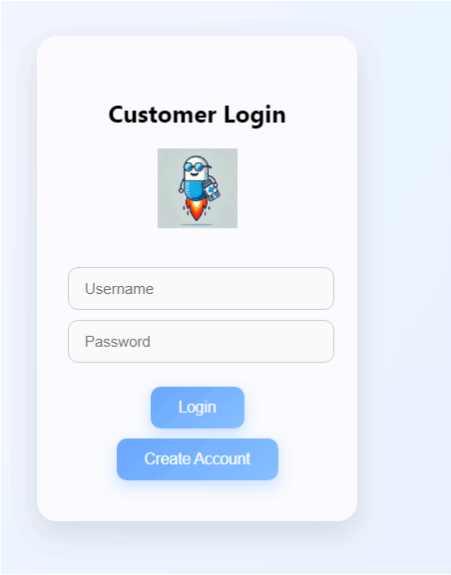
```
SELECT PharmacyId, COUNT(*) AS TotalOrders FROM Customer_Order GROUP BY PharmacyId;
```

Interface



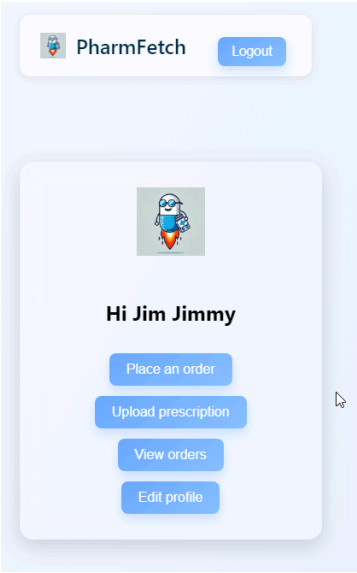
User Type Selection Page

This is the first screen users see when opening the app. They choose their role as Customer, Driver, Pharmacy, Admin, or Doctor. Based on their selection, they are directed to the appropriate login and dashboard. This ensures that users only interact with the features relevant to them and avoids showing unnecessary options.



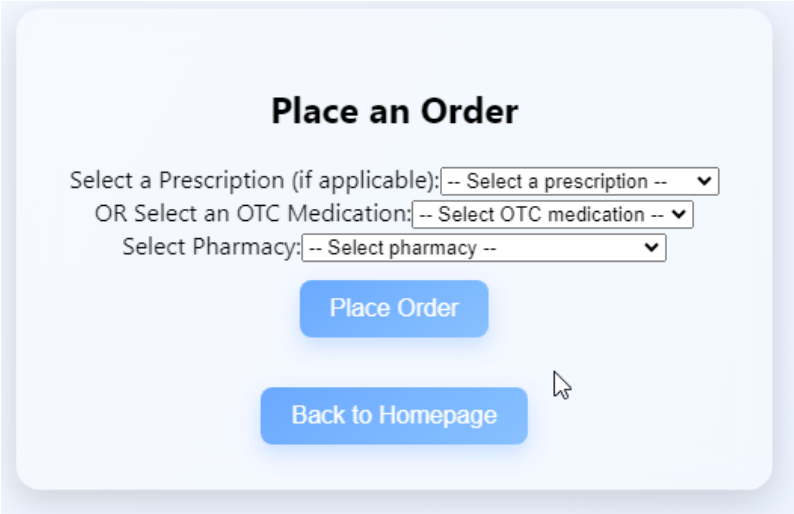
Customer Login Page

After selecting the customer role, users are taken to the login screen. Returning users can log in using their username and password, while new users have the option to create an account. This helps manage access to the customer portal and keeps the login process simple.



Customer Home Page

Once logged in, customers are welcomed by name and presented with clear options. They can place a new order, upload a prescription, view their past orders, or edit their profile. The layout makes it easy for customers to complete the actions they need without confusion. The interface also features a logout button, to allow users to easily leave their homepage once they are finished interacting with the app.



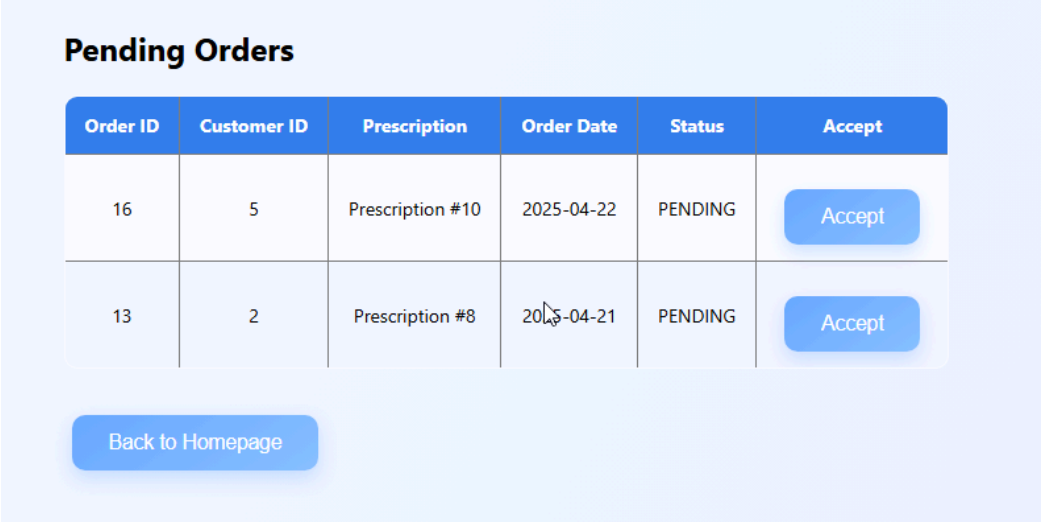
Place an Order Page

This page allows customers to place a new order by selecting either a prescription or an over-the-counter medication along with the pharmacy fulfilling the request. To prevent invalid combinations, the form is set up so that only one of the two fields, prescription or OTC, can be selected at a time. This helps maintain clean data and ensures that each order is tied to a single type of medication. The process is straightforward and guides customers through each step before submitting their order. When a customer

places an order, the app automatically records the current date to store with the order, minimizing user input.

Customer Orders Page

This page shows a logged-in customer their complete order history. Each order is listed with its ID, prescription type, pharmacy, driver, delivery status, and date. Orders that don't require a prescription are labeled as "Over-the-counter," while others reference the linked prescription. This view gives customers an easy way to track the progress of past and active orders all in one place.

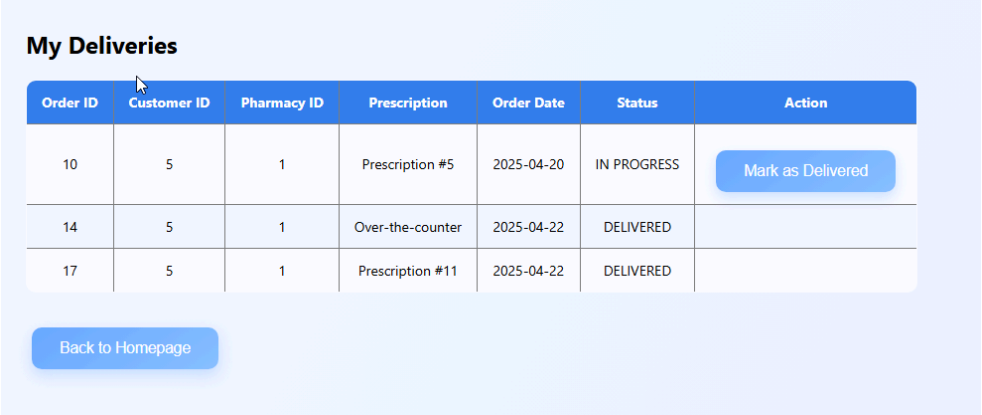


Pharmacy Accept Order Page

This page shows pharmacies a list of orders that have been placed by customers but not yet accepted. Each row displays the order ID, customer ID, prescription information, date, and current status. Pharmacies must manually accept an order before it becomes available for a driver to pick up. This step ensures that only confirmed and prepared orders move forward in the delivery process.

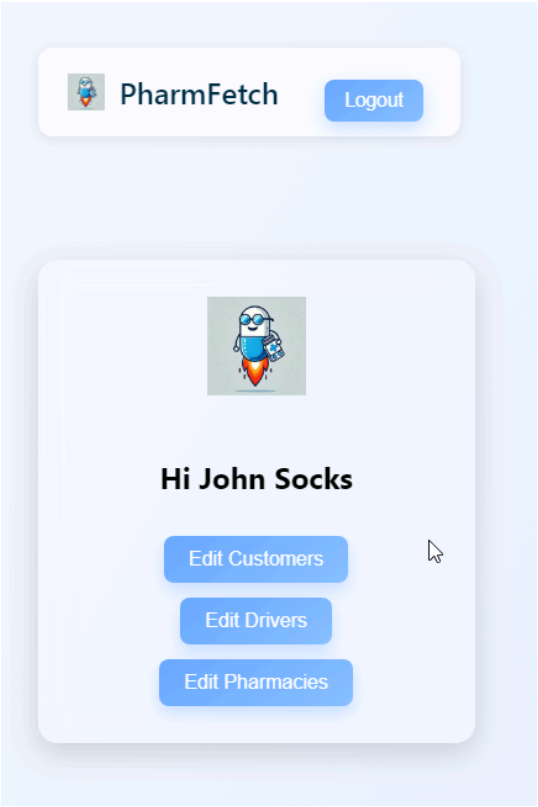
Driver Pick-up Orders Page

This page allows drivers to pick up orders they would like to deliver. It lists all accepted but unassigned deliveries, allowing drivers to review basic order details and claim the ones they want to deliver. This makes it easy for drivers to manage their workload and helps ensure no orders are missed.



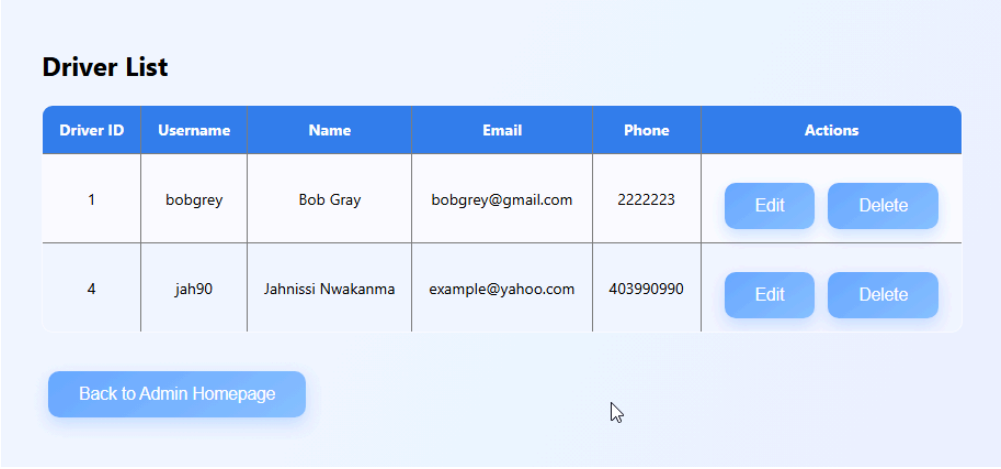
Driver Delivery Dashboard

This page displays all orders that a driver has already picked up. For each order, the driver can see details like the customer ID, pharmacy ID, medication type, order date, and current status. Once the driver is finished with the order, they can use the button provided to mark it as “delivered”.



Admin Homepage

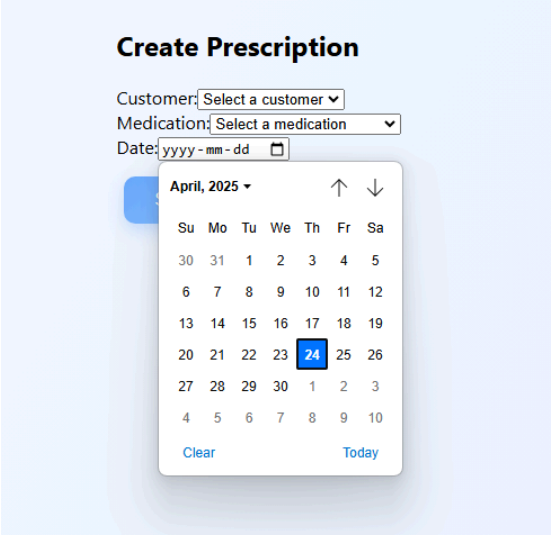
After logging in, admins are taken to this main menu where they can choose to manage customer, driver, or pharmacy accounts. Each button leads to a separate section where they can view, edit, or remove records. This setup gives admins quick access to the core parts of the system they are responsible for maintaining.



Manage Drivers Page

This page shows a list of registered drivers along with their usernames, names, contact details, and an actions column. Admins can choose to edit or delete any driver directly from this view. The layout used

here is consistent across all admin management pages, which helps keep the workflow clear and predictable when managing different user types.

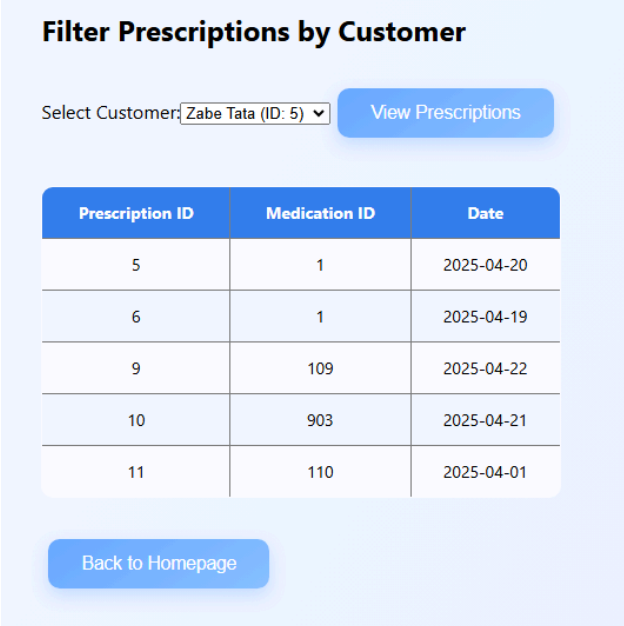


Create a Prescription Page

Doctors use this form to create and submit new prescriptions. They select a customer, choose the appropriate medication, and set the issue date using the date picker. Once submitted, the prescription is linked to both the doctor and the customer.

Doctor's Prescription History Page

This view shows all prescriptions created by the currently logged-in doctor. Each entry includes the prescription ID, customer ID, medication ID, and the date it was issued. This gives doctors a full overview of their prescription activity and supports easy reference across multiple patients.



Filter Prescriptions by Customer Page

This page allows doctors to filter and view prescriptions they've written for a specific customer. After selecting a customer from the dropdown, a list of all prescriptions associated with that person is displayed. This feature helps doctors keep track of past prescriptions and follow up if needed.

Conclusion

Overall, the app was designed to be clear and easy to use for every type of user. Each dashboard only shows the information and actions that are relevant to that specific role, which helps avoid confusion and keeps the experience simple. Navigation is straightforward, forms are direct, and users are guided through key tasks like placing orders or uploading prescriptions without unnecessary steps. The goal was to make the system accessible, even for users who may not be familiar with similar platforms.

User Guide

The application is currently accessible locally, yet the database is hosted online using Azure.

Steps to run the project:

1. Clone the git repository on an IDE that allows React projects and its corresponding files.
2. Open two separate terminals within the project directory 'pharmfetch.'
3. In terminal 1, navigate into 'backend' and enter 'npm start' to start the server, which queries the database. If any errors occur, run 'npm install' and after that, if issues persist, ensure that node is installed on the computer that is running the project.

4. In terminal 2, navigate into ‘frontend’ and enter ‘npm start’ to start the project, at which point the application will be hosted on localhost:3000. If any errors occur, run ‘npm install’ and after that, if issues persist, ensure that node is installed on the computer that is running the project.

5. Then, on the homescreen, users can pick what type of user they are (customer ordering medication, delivery driver, pharmacy, doctor entering prescriptions for customers, or admins managing the users and data).

6. All users can then login if they are pre-existing users, or can then create accounts if they are customers or drivers.