INFSCI 1500 – Final Project Report

Digital Pharmacy Management System

Authors: Nolan Le, Payton Lin, Jeremy Matloub

Github Repository: https://github.com/paytonlin1/Online-Pharmacy-DEMO2

This project implements a digital pharmacy managing system that is designed to facilitate the issuing, tracking, and pickup of prescriptions within a secure, role-based environment. There are three distinct user bases interacting with the system. Each user base's associated capabilities are listed as follows:

Patients

- View prescriptions
- View scheduled refills
- View personal medical history

Doctors

- Issue new prescriptions
- Modify and view patient medical information
- Review patient pickup history
- View patient prescription history

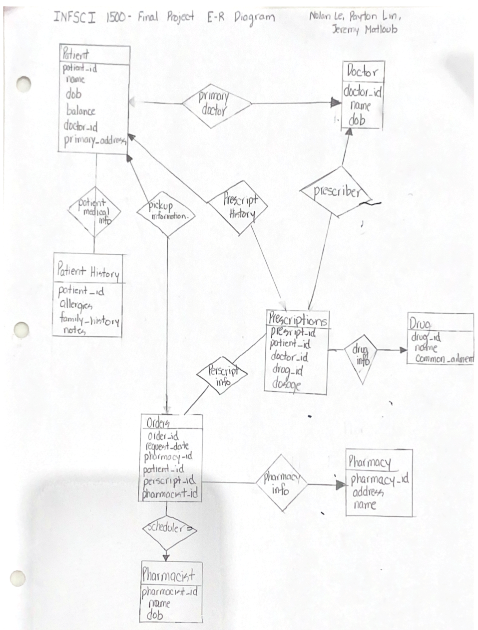
Pharmacists

- Mark orders as completed or cancelled
- View patient prescription history

Regardless of type, all users access the system through a unified home page. Upon initialization, the designated user selects their role (Patient, Doctor, Pharmacist) from a navigation menu. Access to system features is strictly role-based.

The database backend stores all relevant core information: patients, doctors, pharmacists, prescriptions, pharmacies, orders/pickups, and patient histories. The system ensures the integrity of the data in using foreign keys and a normalized schema design. The system is designed for real-world application, taking into account several key assumptions.

- Users only use the dashboard that corresponds to their role.
- *All* doctors and pharmacists in the system are accordingly verified employees
- Each patient is associated with exactly *one* primary doctor.
- Each patient has *one* medical history record.



- A doctor may oversee *many* patients and issue *many* prescriptions.
- A pharmacist may schedule *many* pickups.
- Each order corresponds to exactly *one* prescription.
- All user inputs are via drop-down menus, preventing malformed submissions.

There is no implementation of:

- Billing/insurance logic
- Refill/renewal workflows
- Prescription expiration
- Pharmacy inventory tracking
- Drug–drug interaction or allergen alerts
- Pickup scheduling

Several of the features listed, and not implemented, were omitted as "simplifications" as to best focus on the core functionality of the database and prescription workflows, the two of which being most integral to the purpose the system is designed to fulfill.

The graphical schema of the database is depicted above in the digital scan of our E-R. We have additionally included a brief description of each entity set, relationship set, and their corresponding attributes.

Entity Sets

Patient

- patient_id (PK)
- name
- dob
- doctor_id (FK → Doctor.doctor_id)
- Primary_address

Represents registered patients and their demographic and billing information.

PatientHistory

- patient_id (PK/FK → Patient.patient_id)
- allergies
- family_history
- Notes

Stores medical background for each patient.

Doctor

- doctor_id (PK)
- name
- dob

Represents physicians who issue prescriptions.

Pharmacist

- pharmacist_id (PK)
- name
- dob

Represents pharmacists who schedule and process prescription pickups.

Drug

- drug_id (PK)
- name

- common_ailment

Represents available medications and their primary uses.

Pharmacy

- pharmacy_id (PK)
- name
- address

Represents pharmacy locations.

Prescriptions

- prescript_id (PK)
- patient_id (FK)
- doctor_id (FK)
- drug_id (FK)
- dosage

Represents specific prescriptions issued by doctors.

Pickups (Orders)

- order_id (PK)
- request_date
- pharmacy_id (FK)
- patient_id (FK)
- prescript_id (FK)
- pharmacist_id (FK)
- status (Scheduled, Completed, Cancelled)

Represents the pickup scheduling and fulfillment process.

Relationship Sets

- A doctor can oversee many patients
- A patient can have many prescription entries
- A doctor can have issued many prescriptions
- A pharmacist can schedule many pickups
- A patient can have many pickups
- A patient can only have 1 patient history entry
- A drug can be listed under many prescriptions
- A pharmacy may be listed as the pickup address for many prescriptions
- An order can only have 1 corresponding prescription

We then derived a formal relational schema from the E-R Diagram as follows:

Patient

- patient_id - primary
- name
- dob
- doctor_id – foreign from doctor
- primary_address

Patient_id -> (name, dob)

PatientHistory

- patient_id - primary, foreign from Patient
- Allergies
- Family history
- Special Notes

patient_id->(allergies, family_history)

Prescriptions

- prescript_id - primary
- patient_id – foreign from patient
- doctor_id - foreign from doctor
- drug_id - foreign from drug
- dosage

prescript_id, patient_id->drug_id, dosage

Pharmacy

- pharmacy_id - primary
- Address
- Informal Name

pharmacy_id->address, name

Orders

- order_id - primary
- request_date
- pharmacy_id - foreign from Pharmacy
- patient-id - foreign from Patient
- prescript_id - foreign from Prescriptions
- Pharmacist-id - foreign from pharmacist
- status

order_id->request_date, patient_id, prescript_id

Drug

- drug_id - primary
- name
- common_ailment

drug_id->name, common_ailment

Doctor

- doctor_id - primary
- name
- dob

doctor_id->name

Pharmacist

- pharmacist_id - primary

- name
- dob

ID -> name

As contained in our SQL DDL (Final_Project_User_Actions.sql) file in the github repository, all tables are defined in BCNF (Boyce-Codd Normal Form). They are reiterated within this document as follows:

```
USE pharmacy_testing;

CREATE table doctor

    (doctor_id INT UNIQUE,

    name VARCHAR(30),

    dob DATE,

    PRIMARY KEY(doctor_id));
```

```
CREATE table pharmacist

    (pharmacist_id INT UNIQUE,

    name VARCHAR(30),

    dob DATE,

    PRIMARY KEY(pharmacist_id));
```

```
CREATE table Patient

    (patient_id INT UNIQUE,

    name VARCHAR(30),

    dob date,
```

```
        doctor_id INT,
        primary_address VARCHAR(100),
        PRIMARY KEY(patient_id),
        FOREIGN KEY(doctor_id)
                references doctor);
```

```
CREATE table PatientHistory
        (patient_id INT UNIQUE,
        allergies VARCHAR(200),
        family_history VARCHAR(200),
        notes VARCHAR(200),
        PRIMARY KEY(patient_id),
  FOREIGN KEY(patient_id)
                references Patient);
```

```
CREATE table prescriptions
        (prescript_id INT UNIQUE,
        patient_id INT,
        doctor_id INT,
        drug_id INT,
        dosage INT,
        PRIMARY KEY(prescript_id),
```

```
    FOREIGN KEY(patient_id)
        references Patient,
    FOREIGN KEY(doctor_id)
        references doctor);
```

```
CREATE table drug
    (drug_id INT UNIQUE,
    name VARCHAR(50),
    common_ailment VARCHAR(200));
```

```
CREATE table pharmacy
    (pharmacy_id INT UNIQUE,
    address VARCHAR(100),
    name VARCHAR(100),
    PRIMARY KEY(pharmacy_id));
```

```
CREATE table orders
    (order_id INT UNIQUE,
    request_date DATE,
    pharmacy_id INT,
    patient_id INT,
    prescript_id INT,
```

```
        pharmacist_id INT,

        status ENUM("Cancelled", "Scheduled", "Completed"),

    PRIMARY KEY(order_id),

    FOREIGN KEY(pharmacy_id)

            references pharmacy,

    FOREIGN KEY(patient_id)

            references Patient,

    FOREIGN KEY(prescript_id)

            references prescriptions,

    FOREIGN KEY(pharmacist_id)

            references pharmacist);
```

The schema is in BCNF, as referenced previously, as it abides by all of the required conditions. Each table has a single-attribute primary key, with all non-key attributes being dependent solely on the given primary key. Additionally, there are no transitive dependencies or composite keys, eliminating any partial dependencies. Every functional dependency has a left hand side in tandem, serving as a superkey. As such, the schema not only satisfies the requirements of BCNF, it also satisfies 1NF, 2NF, and 3NF respectively.

Design & Connection

The front end of the system is designed using Bootstrap, which offers a clean, responsive interface across the four primary HTML templates. These templates correspond to the major flask routes and user roles:

- **Home Page** → User selection and page navigation

- **Patient Dashboard** → Displays prescription history, refill scheduling, and current medications
- **Doctor Dashboard** → Prescription issuance tools and patient lookup
- **Pharmacist Dashboard** → Pickup completion workflow

Bootstrap components such as forms, navigation bars, buttons, and tables were used to ensure an intuitive user experience. Each interface page uses Jinja2 templating to dynamically insert data returned from Flask routes.

Integration

The system's back-end is implemented in Flask, routing user interactions to Python functions to subsequently execute the required database operations. By utilizing SQLAlchemy, the application connects to a MySQL database that manages queries, inserts, and updates, all the while preserving any referenced information, respectively.

Forging this connection was one of the more challenging aspects of the project. Establishing communication between Flask and MySQL Workbench distinctly required configuration and handling of environmental variables. Having never implemented SQL user queries with a legitimate SQL database, this proved to be more adversarious than anticipated. Upon connecting, however, SQLAlchemy made the translation from Python to SQL significantly more fluid.

One of the most difficult aspects of the back-end implementation was writing the SQL queries in of itself, as well as creating a distinct and concise schema to avoid redundancy. Proper syntax was required to ensure all foreign key dependencies were aligned with said schema, as it was logistically essential for the foreign keys to match the prescription and orders tables, respectively.

System Implementation Overview

There are three key components of the system's implementation. The attached images are depictions of the user interface designed to demonstrate the working implementation.

Database Tables: Implemented in SQL, representing Patients, Doctors, Pharmacists, Prescriptions, Drugs, Pharmacies, and Pickups.

HTML/Bootstrap Templates: The user-facing pages where forms, tables, and results are shown.

Flask Routes: Logic connecting forms to Python, then to SQL statements.

Order of Operations

1. Initialization
2. User selects role & logs in
3. Flask verifies identity and loads the respective dashboard
4. The user interacts with forms:
    - *Issuing* prescriptions
    - *Scheduling* pickups
    - *Querying* patient history
5. Flask:
    - *Receives* form data
    - *Constructs* SQLAlchemy queries
    - *Updates* MySQL.
6. Updated data is displayed through rendered Jinja2 templates.

Call: 1-800-PHARMACY info@pharmacy.com Mon-Sat: 8AM - 8PM

Online Pharmacy [DEMO] Home Doctor Portal Pharmacist Portal Patient Portal

Your Health, Our Priority

Connect with doctors, manage prescriptions, and get medications delivered to your door. Experience seamless healthcare management.

Patient Portal Doctor Portal Reset Demo

15	11	14	10
Active Patients	Qualified Doctors	Prescriptions Filled	Licensed Pharmacists

Why Choose Us?

Secure & Safe

Your health data is protected with enterprise-level security and HIPAA compliance.

24/7 Access

Manage your prescriptions and connect with healthcare providers anytime, anywhere.

Fast Delivery

Get your medications delivered to your doorstep quickly and reliably.

Access Your Portal

Patient Portal

View prescriptions, track orders, and manage your health.

Access Portal

Doctor Portal

Manage patients, prescribe medications, and track appointments.

Access Portal

Pharmacist Portal

Process prescriptions, manage inventory, and fulfill orders.

Access Portal

Online Pharmacy

Your trusted online pharmacy connecting doctors, pharmacists, and patients.

Quick Links

About Us
Contact
Privacy Policy
Terms & Conditions

Contact Info

123 Medical Plaza, Health City
1-800-PHARMACY
info@pharmacy.com

Online Pharmacy [DEMO] Home Doctor Portal Pharmacist Portal Patient Portal

Doctor Dashboard

Welcome back, Hiro Tanaka

New Prescription

2
Total Patients

riptions This Month

Recent Prescriptions

Patient
Nolan Le
Jeremy Matloub

History
History

Patient History - Jeremy Matloub

Allergies

N/A

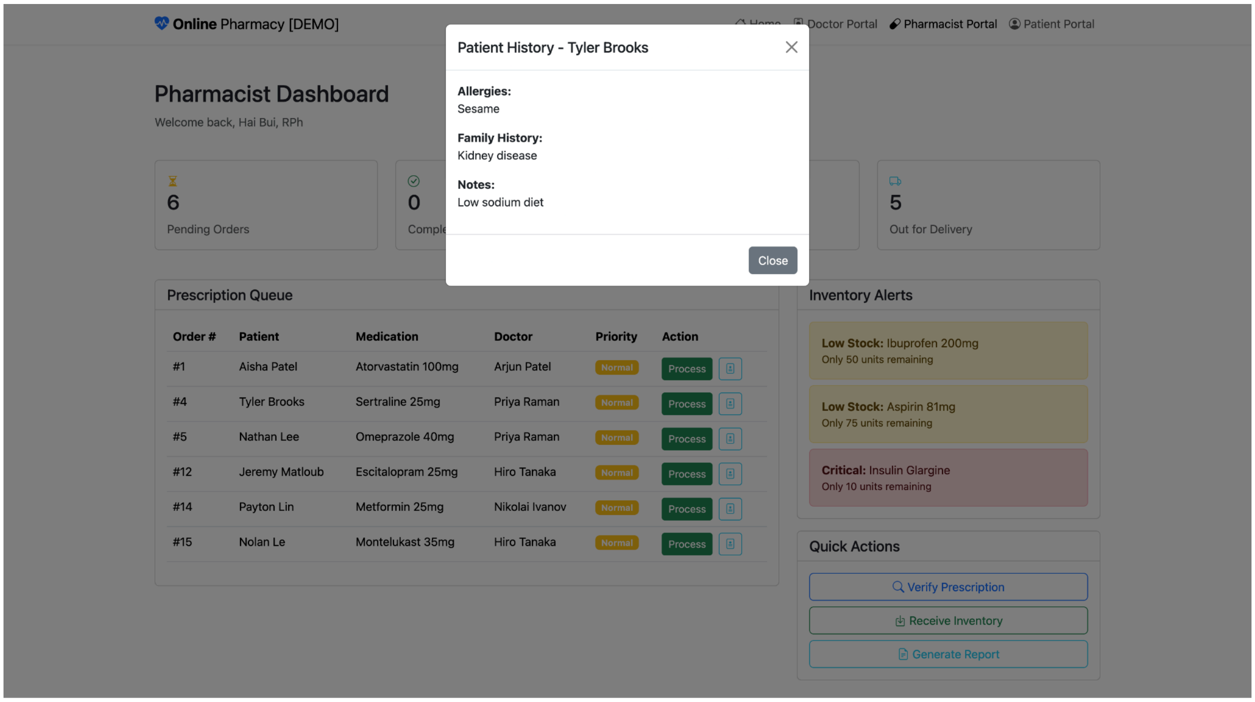
Family History

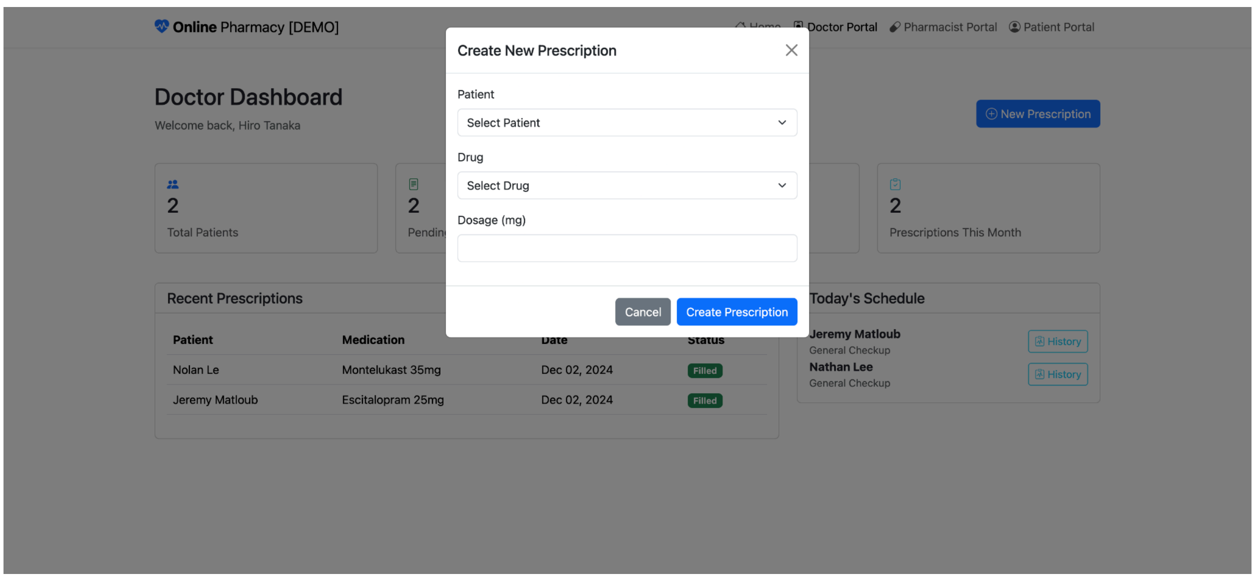
N/A

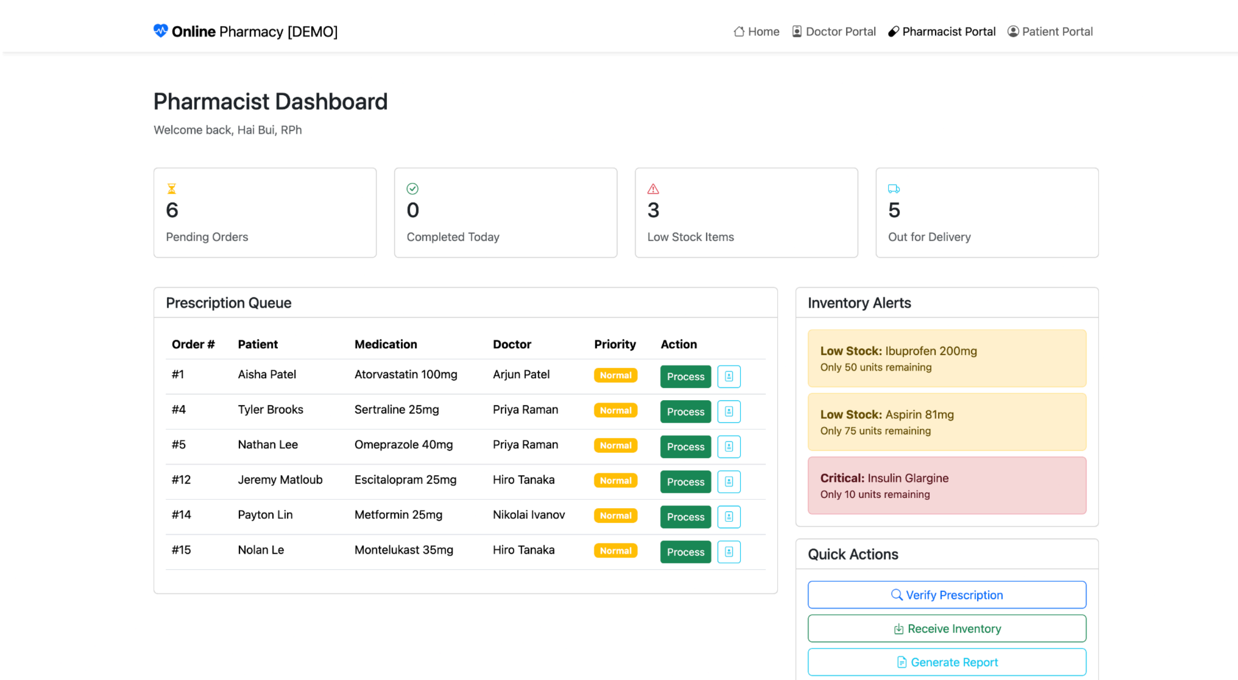
Notes

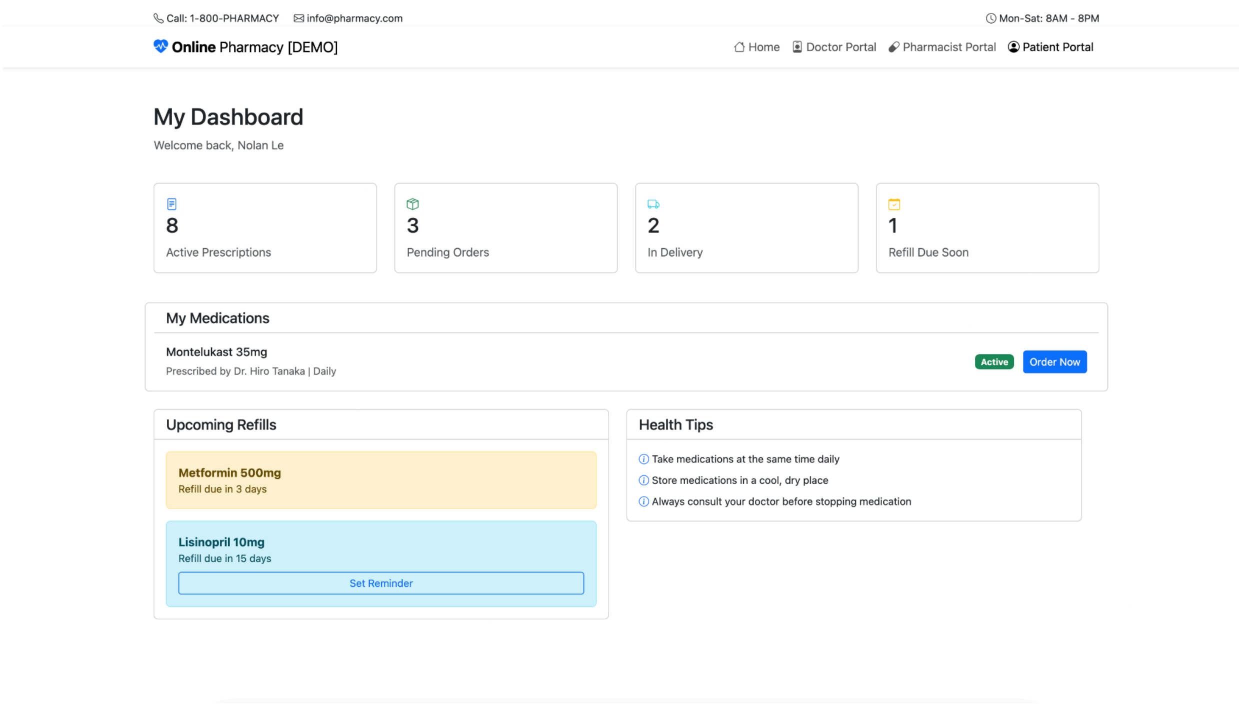
N/A

Close Save Changes


Online Pharmacy [DEMO]
Home
Doctor Portal
Pharmacist Portal
Patient Portal
Patient History - Tyler Brooks
Allergies:
Sesame
Family History:
Kidney disease
Notes:
Low sodium diet
Close
Pharmacist Dashboard
Welcome back, Hai Bui, RPh
6
Pending Orders
0
5
Out for Delivery
Prescription Queue
Order # Patient Medication Doctor Priority Action
#1 Aisha Patel Atorvastatin 100mg Arjun Patel Normal Process
#4 Tyler Brooks Sertraline 25mg Priya Raman Normal Process
#5 Nathan Lee Omeprazole 40mg Priya Raman Normal Process
#12 Jeremy Matloub Escitalopram 25mg Hiro Tanaka Normal Process
#14 Payton Lin Metformin 25mg Nikolai Ivanov Normal Process
#15 Nolan Le Montelukast 35mg Hiro Tanaka Normal Process
Inventory Alerts
Low Stock: Ibuprofen 200mg
Only 50 units remaining
Low Stock: Aspirin 81mg
Only 75 units remaining
Critical: Insulin Glargine
Only 10 units remaining
Quick Actions
Verify Prescription
Receive Inventory
Generate Report


Online Pharmacy [DEMO]
Doctor Portal
Pharmacist Portal
Patient Portal
Create New Prescription
Patient
Select Patient
Drug
Select Drug
Dosage (mg)
Cancel
Create Prescription
Doctor Dashboard
Welcome back, Hiro Tanaka
New Prescription
2
Total Patients
2
2
Prescriptions This Month
Recent Prescriptions
Patient Medication Date Status
Nolan Le Montelukast 35mg Dec 02, 2024 Filled
Jeremy Matloub Escitalopram 25mg Dec 02, 2024 Filled
Today's Schedule
Jeremy Matloub
General Checkup
History
Nathan Lee
General Checkup
History


Online Pharmacy [DEMO]
Home
Doctor Portal
Pharmacist Portal
Patient Portal
Pharmacist Dashboard
Welcome back, Hai Bui, RPh
6
Pending Orders
0
Completed Today
3
Low Stock Items
5
Out for Delivery
Prescription Queue
Order # Patient Medication Doctor Priority Action
#1 Aisha Patel Atorvastatin 100mg Arjun Patel Normal Process
#4 Tyler Brooks Sertraline 25mg Priya Raman Normal Process
#5 Nathan Lee Omeprazole 40mg Priya Raman Normal Process
#12 Jeremy Matloub Escitalopram 25mg Hiro Tanaka Normal Process
#14 Payton Lin Metformin 25mg Nikolai Ivanov Normal Process
#15 Nolan Le Montelukast 35mg Hiro Tanaka Normal Process
Inventory Alerts
Low Stock: Ibuprofen 200mg
Only 50 units remaining
Low Stock: Aspirin 81mg
Only 75 units remaining
Critical: Insulin Glargine
Only 10 units remaining
Quick Actions
Verify Prescription
Receive Inventory
Generate Report



Testing Efforts

Testing efforts included validating both the SQL schema and Flask application logic to ensure the system operated as intended for all user roles. Each set of testing proved to possess their own unique challenges.

SQL Testing

- Verified each DDL statement to ensure proper formatting and syntax.
- Inserted dummy test data to confirm if data types and constraints matched the schema
- Ensured foreign keys referenced valid tables and that logical relationships made sense
  - Prescriptions correctly link doctors, patients, and drugs
  - Orders correctly link pharmacies and pharmacists
- Checked that no tables contained redundant data or violated BCNF assumptions as stated previously.

Creating SQL queries was one of the most difficult aspects of development. Ensuring each query joined the correct tables, referenced the proper columns, and returned consistent results required significant time, testing, and iteration.

Flask Testing

- Tested every form submission route to verify valid handling of user input.
- Ensured SQLAlchemy queries had:
  - Properly construction
  - No errors when connecting to MySQL
- Confirmed logical restrictions:
  - Patients cannot issue prescriptions
  - Doctors cannot confirm pickups
  - Pharmacists cannot alter medical history

Error Handling

The system is capable of detecting and handling three primary types of errors, in turn dramatically improving the user experience and reducing malfunctions in tandem. Each given case addresses the errors listed below it, respectively.

reset_demo()
- Insert/create statements

create_prescription()
- No doctor in database
- No drug in database
- No patient in database

get_patient_history()
- Missing history in database

System Limitations

While the system successfully implements core digital pharmacy functionality, several limitations are sustained. Certain limitations were acknowledged and accounted for feasibility purposes, the primary of which being arbitrary prescription prices and a lack of support for insurance claims, co-pays, or billing workflows. The backend cannot verify if the user is of the role they claim to be, creating a system vulnerable to exploitation. Additionally, we do not take into account pharmacy inventory, drug expiration dates, or refill limits due to the exceeding complexity.

The integration process of connecting Flask to MySQL and implementing the attached queries proved challenging due to inexperience with a live SQL environment. It is likely additional optimization is possible on the back-end.

System Improvements

The collection of features we omitted would all prove to be positive system improvements for the future. By integrating real-world drug prices, insurance and billing support, internal pickup scheduling, inventory tracking and management, the system would better support the needs of the user. An additional feature could include automated allergy and drug interaction alerts tied to any given patient's prescription history.

This digital pharmacy management system successfully facilitates the issuing, tracking, and processing of prescriptions within a role-based environment, as intended. While there are supplementary features needed for a real-world implementation, the core functionality of the system is sound and reliable for each intended user base.