

## Pharmacy Portal Report

The first thing I did for the pharmacy portal project was to incorporate the starter code into vscode, startup XAMPP activate the apache and mySQL services, and create the database in phpmyadmin.

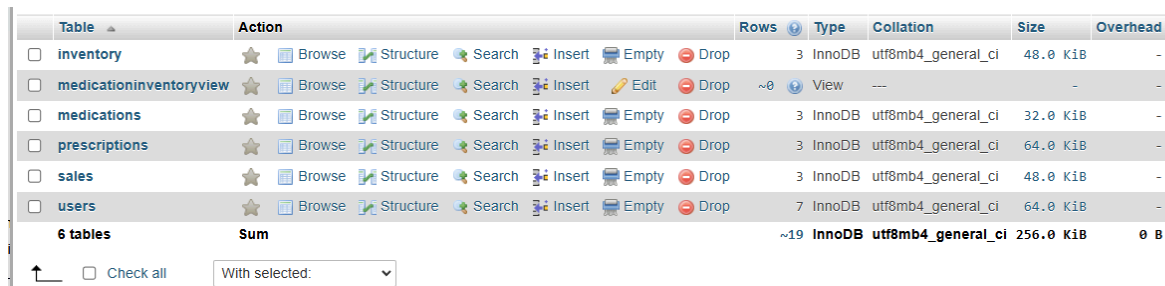


Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> inventory	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	48.0 KiB	-
<input checked="" type="checkbox"/> medicationinventoryview	★ Browse Structure Search Insert Edit Drop	~0	View	---	-	-
<input type="checkbox"/> medications	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> prescriptions	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	64.0 KiB	-
<input type="checkbox"/> sales	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	48.0 KiB	-
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop	7	InnoDB	utf8mb4_general_ci	64.0 KiB	-
6 tables	Sum	~19	InnoDB	utf8mb4_general_ci	256.0 KiB	0 B

☐ Check all With selected: ▾

I created the tables users, medications, prescriptions, sales, and inventory.

Then I created the stored procedure AddOrUpdateUser. This procedure made it easier to manage the Users table by allowing both insertions and updates depending on whether a userId or userName already existed. I included logic to check if a user was already in the system and, if not, it inserted them. If the user was found by either ID or name, the record would be updated with the latest contact info and user type. This was helpful especially during login and registration where user roles needed to be consistently managed.

Next, I developed the ProcessSale procedure to handle medication transactions. This procedure receives the prescription ID and quantity sold. It checks the current inventory, calculates the sale amount using a fixed price (for simplicity), updates the Inventory table by subtracting the sold amount, and inserts a new record into the Sales table. I made sure it used a transaction block (START TRANSACTION and COMMIT) to keep the database consistent in case any step failed. This stored procedure saved time and reduced human error when processing pharmacy sales.

I also created a SQL View called MedicationInventoryView. This view combines data from the Medications and Inventory tables to provide pharmacists with a quick snapshot of available medications, their dosage, manufacturer, and current stock levels. I used LEFT JOIN to ensure that even medications with no inventory data yet would still appear. This was useful during early testing and debugging.

Finally, I implemented a trigger named AfterPrescriptionInsert. This trigger activates automatically after a new prescription is inserted. It updates the Inventory by deducting the prescription quantity and logs the time it was updated. I also included a conditional block to raise a custom SQL error using SIGNAL if the inventory drops below a critical threshold (set to 5 units). This helped simulate real-world scenarios where pharmacy staff should be alerted of low stock.

I then populated each table with atleast three entries.

<input type="checkbox"/>				1	adan_ben	NULL	patient
<input type="checkbox"/>				2	michael_ko	mkocovic@gmail.com	pharmacist
<input type="checkbox"/>				3	elly_fam	eleassarf@gmail.com	patient
<input type="checkbox"/>				4	alice_wong	NULL	patient
<input type="checkbox"/>				5	steve_matthew	NULL	pharmacist
<input type="checkbox"/>				6	admin1	admin@gmail.com	pharmacist
<input type="checkbox"/>				7	test	testuser@gmail.com	patient

				medicationId	medicationName	dosage	manufacturer
<input type="checkbox"/>				1	Amoxicillin	500mg	GlaxoSmithKline
<input type="checkbox"/>				2	Ibuprofen	200mg	Johnson & Johnson
<input type="checkbox"/>				3	Paracetamol	650mg	Bayer

				prescriptionId	userId	medicationId	prescribedDate	dosageInstructions	quantity	refillCount
<input type="checkbox"/>				1	1	1	2025-05-01 09:00:00	Take one capsule every 8 hours	21	1
<input type="checkbox"/>				2	3	2	2025-05-02 13:30:00	Take two tablets after meals	30	2
<input type="checkbox"/>				3	2	3	2025-05-03 11:15:00	One tablet at bedtime	15	0

				inventoryId	medicationId	quantityAvailable	lastUpdated
<input type="checkbox"/>				1	1	120	2025-05-01 08:30:00
<input type="checkbox"/>				2	2	75	2025-05-02 14:45:00
<input type="checkbox"/>				3	3	200	2025-05-03 10:15:00

				saleId	prescriptionId	saleDate	quantitySold	saleAmount
<input type="checkbox"/>				1	1	2025-05-01 09:15:00	21	42.00
<input type="checkbox"/>				2	2	2025-05-02 13:45:00	30	15.00
<input type="checkbox"/>				3	3	2025-05-03 11:30:00	15	7.50

After setting up the database side of the project I was now ready to begin altering the Starter Code and begin with the php side of the project. I went into the PharmacyDatabase.php file to update the connection settings. I changed the values for the host, port, database name, username, and password to match the MySQL environment I had running on XAMPP which was no password so I set password = "". This was important because without the correct credentials and database name, none of the queries or functionality would work. Once I saved the changes, I tested the connection by refreshing the homepage, and after confirming there were no errors, I was ready to continue building out the PHP functionality.

```
<?php
class PharmacyDatabase {
    private $host = "localhost";
    private $port = "3306";
    private $database = "pharmacy_portal_db";
    private $user = "root";
    private $password = "";
    private $connection;
```

After updating the database connection details, I tested the platform by launching the project on my local PHP server using XAMPP. I opened the browser and navigated to <http://localhost/final-project/PharmacyServer.php> to check if the homepage would load without errors. Once I saw the homepage with the "Pharmacy Portal" title and navigation links, I knew the connection was working.

Then I clicked on the "Add Prescription" link to make sure it led to the form. At first, the page was blank, but after debugging and making sure the path to `addPrescription.php` was correct, it started working. I filled out the form and submitted a test prescription. This helped confirm that the `addPrescription` function in the `PharmacyDatabase.php` file was working as expected.

I also tested the "View Prescriptions" page. It originally showed a blank screen too, but I fixed it by passing the `$prescriptions` variable properly and making sure the data was being fetched with a working query. After these fixes, I was able to view a styled table of all prescriptions in the database.

After setting up the homepage and testing the platform routes, I implemented medication addition functionality within the `addPrescription()` method in the `PharmacyDatabase.php` file.

The code checks if the patient exists in the Users table using their username and verifies they're of userType 'patient'. If the patient is found, it proceeds to insert a new prescription entry into the Prescriptions table using the selected medication ID, dosage instructions, and quantity.

```
public function addPrescription($patientUserName, $medicationId, $dosageInstructions, $quantity) {
    $stmt = $this->connection->prepare(
        "SELECT userId FROM Users WHERE userName = ? AND userType = 'patient'"
    );
    $stmt->bind_param("s", $patientUserName);
    $stmt->execute();
    $stmt->bind_result($patientId);
    $stmt->fetch();
    $stmt->close();

    if ($patientId) {
        $stmt = $this->connection->prepare(
            "INSERT INTO prescriptions (userId, medicationId, dosageInstructions, quantity) VALUES (?, ?, ?, ?)"
        );
        $stmt->bind_param("iisi", $patientId, $medicationId, $dosageInstructions, $quantity);
        $stmt->execute();
        $stmt->close();
        echo "Prescription added successfully";
    } else {
        echo "failed to add prescription";
    }
}
```

After getting the main prescription features working, I moved on to user creation. I completed and implemented the `addUser()` method in the `PharmacyDatabase.php` file. This function lets me add new users into the system by passing in their username, contact info, user type, and optionally a password.

I used the stored procedure AddOrUpdateUser inside the method so that it would either add the user or update them if they already exist. Then, if a password was provided, I hashed it using PHP's password\_hash() and updated the user's record with that hash. This allowed the login system to verify users later on.

```
public function addUser($userName, $contactInfo, $userType, $password = null) {
    if ($password) {
        $hashed = password_hash($password, PASSWORD_DEFAULT);
    } else {
        $hashed = null;
    }

    $stmt = $this->connection->prepare("CALL AddOrUpdateUser(?, ?, ?, ?)");
    $null = null;
    $stmt->bind_param("iss", $null, $userName, $contactInfo, $userType);
    $stmt->execute();
    $stmt->close();

    if ($hashed) {
        // Store hashed password if it's a new user
        $stmt = $this->connection->prepare("UPDATE Users SET passwordHash = ? WHERE userName = ?");
        $stmt->bind_param("ss", $hashed, $userName);
        $stmt->execute();
        $stmt->close();
    }
}
```

Once the core pages were working, I went ahead and tested the MedicationInventoryView to make sure it was showing the correct data from the database. I first created a method called MedicationInventory() inside PharmacyDatabase.php that runs a query on the MedicationInventoryView. Then I added a viewInventory action in the PharmacyServer.php file that loads a table on the front end.

To test it, I clicked on the "View Inventory" link from the homepage and checked if the medications were showing their name, dosage, manufacturer, and current inventory count. The numbers matched the data I inserted earlier, which confirmed the view was working like it should.

```
public function MedicationInventory() {
    $result = $this->connection->query("SELECT * FROM MedicationInventoryView");
    return $result->fetch_all(MYSQLI_ASSOC);
}
```

After setting up login and registration, I needed a way to show user-specific info like their details and prescription history. So I completed and implemented the getUserDetails(\$userId) method in PharmacyDatabase.php. This function first pulls the user's basic data like their username, contact info, and user type from the Users table. Then I added a second query inside the same

function to also grab all prescriptions tied to that user, including the medication name and dosage instructions.

All of this data gets returned in a single array so I could use it on the profile page later. This method helped tie everything together so users can log in, view their own info, and see their medication history. I used this in the ?action=profile part of the platform to test that it worked.

To secure the platform, I set up a login system in login.php that works for both patients and pharmacists. I used password\_hash() when creating accounts to store the password securely, and then password\_verify() to check if the login password matched what was in the database. This made sure no one's actual password was stored in plain text.

After a user logs in, I used session\_start() to store their user ID, name, and user type so the platform knows who's logged in. I added role-based access so pharmacists can see everything, like adding prescriptions and checking inventory, while patients are only allowed to see "View My Prescriptions" and "View My Profile." I also made a logout.php file so users can safely end their session. This whole setup made the site more secure and let me control what each user type can do.

## Pharmacy Portal

[View Prescriptions](#)[View My Profile](#)[Logout](#)

## Pharmacy Portal

[Add Prescription](#)[View Prescriptions](#)[View Inventory](#)[View My Profile](#)[Logout](#)