Davey Anguiano, Dawson Hill, Matt Morrison

MyHealth Team #2

Pharmacist & Provider Portal

Final report

Table of Contents

[Summary 2](#_Toc71224874)

[Implementation 2](#_Toc71224875)

[Web Design Implementation 2](#_Toc71224876)

[Implementation 1: Server Connection 2](#_Toc71224877)

[Implementation 2: Establish Session 3](#_Toc71224878)

[Implementation 3: Login Backend 3](#_Toc71224879)

[Implementation 4: Signup Backend 4](#_Toc71224880)

[Implementation 5: Logout Backend 5](#_Toc71224881)

[Styling: Main Concept 5](#_Toc71224882)

[Web Portal Implementation 6](#_Toc71224883)

[Function 1: index.php 6](#_Toc71224884)

[Function 2: signup.php 6](#_Toc71224885)

[Function 3 & 4: doctorHome.php and pharmacyHome.php 7](#_Toc71224886)

[Function 5: pharmHistory.php 7](#_Toc71224887)

[Function 6: pharmSell.php 8](#_Toc71224888)

[Function 7: docDiagnose.php 8](#_Toc71224889)

[Function 8: docPatients.php 9](#_Toc71224890)

[Database Implementation 10](#_Toc71224891)

[Table 1: Drugs 11](#_Toc71224892)

[Table 2: Insurance 11](#_Toc71224893)

[Table 3: Interactions 12](#_Toc71224894)

[Table 4: Patients 12](#_Toc71224895)

[Table 5: Pharmacy 12](#_Toc71224896)

[Table 6: Provider 13](#_Toc71224897)

[Table 7: Purchases 13](#_Toc71224898)

[Table 8: Referral 14](#_Toc71224899)

[Tools Used 14](#_Toc71224900)

[GitHub 14](#_Toc71224901)

[Discord 15](#_Toc71224902)

[Mockaroo 15](#_Toc71224903)

[How to Use - GitHub Link 16](#_Toc71224904)

[Challenges 16](#_Toc71224905)

[Implementation & Web Design Challenges 16](#_Toc71224906)

[Functionality & Connectivity Challenges 16](#_Toc71224907)

[Database Challenges 17](#_Toc71224908)

[Web Server Hosting Challenges 17](#_Toc71224909)

[Reflection and Conclusion 17](#_Toc71224910)

[Web Design Reflection 17](#_Toc71224911)

[Web Page Functionality Reflection 17](#_Toc71224912)

[Database Reflection 17](#_Toc71224913)

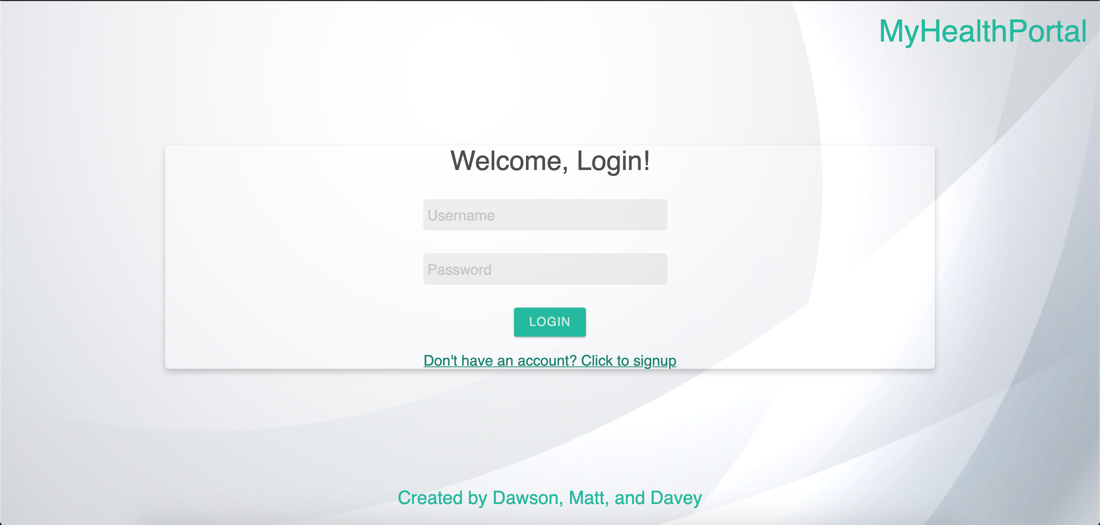
# Summary

Our team constructed a database that would interface with both the Pharmacist and Provider. Specifically, creating an interface for the provider to generate referrals and pharmacist to see all purchases and see patient history with certain drug interactions provided that the patient has given the pharmacy permission to view history. This is accomplished by using a web-based portal that identifies whether the username is associated with a pharmacist or a provider. Once that has been identified the user is redirected to their proper homepage to execute usage of their choice. Finally, this portal is handled by a database system that consists of tables that let the provider and pharmacist to access purchase history, insurance info, patient info, list of available drugs, and referrals generated for controlled substances or other drug types.

# Implementation

## Web Design Implementation

Implementation with the log-in and sign-up functions were first setup to run on a localhost server. Tinkering around with the connection files, from creating our own to utilizing the provided dbconn.php file; we experimented with multiple connection types to interact with the database. As for the graphical interface, most of the elements were inspired by Google’s Material Design standard; with every element and animation being made with pure CSS (no external libraries or JavaScript).



### Implementation 1: Server Connection

As shown in the image below, we insert this snippet to the top of every file establishing a connection with the database. First, we include out connection and login verification (functions.php) files. Then created a variable that establishes a connection with the database. Lastly, we have a function to check if the connection was successful. The dbconn.php file stores our variables for database access (i.e., server name, port number, socket number, etc.).



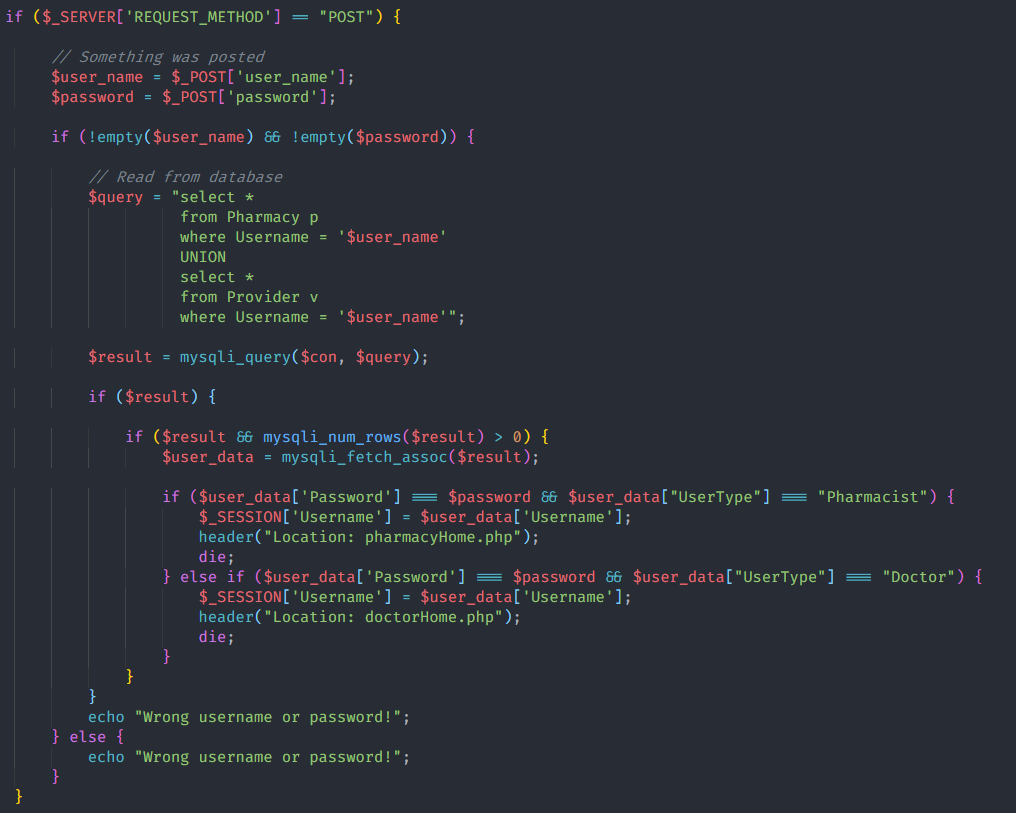
### Implementation 2: Establish Session

The verification to login is a simple SQL query to begin the session. If the data provided is not in the database or there was no account found, you’ll get redirected back to the login screen. To search both our user database tables we joined them using the UNION keyword, to create a super table of all the accounts to procced with the lookup. This function if successful returns the data of the user for use in other queries.



### Implementation 3: Login Backend

Upon logging in, the user’s input is looked up in the unionized table. Once, a user is found, the data is retrieved and compared to the username and password input. To determine if the inputted data is correct, we compare it with the credentials from the database to see if they are identical. If entered correctly, the user is redirected to its proper home page, determined by their user type we retrieved earlier. If entered incorrectly, the user receives an error message and can try again.



### Implementation 4: Signup Backend

When a new user is signing up, they are greeted with input fields for them to enter their information, once submitted they are redirected to the login screen. On the backend side, we store the data inputted as temporary variables, to later easily record their data into the database. We begin by checking if any of the fields are empty, if they are the user is prompted to input more information. After all the entries are satisfied, we check the user type selected, to properly input their data in the correct table.



### Implementation 5: Logout Backend

As the user completes their tasks for the day, they have the option to logout of the portal using the “LOGOUT” button on the bottom left. This feature is implemented by verifying a session is started and a username for the session is set; in other words, the function determines for a user is logged in. If the condition is true, we unset the session username and kill the session, then redirect back to the login screen.

Text

Description automatically generated

### Styling: Main Concept

Deciding on a design standard to follow, was a bit overwhelming, considering there are an infinite number of options. However, our main clients/users will be in a professional setting. Allowing up to filter all the heavily animated, color distracting, and style cues; after further evaluation Google’s Material Design seemed to be the perfect fit. Not only does it give a professional look and feel, but it also allows for some subtle animations to bring life to the page. Paired with a trustworthy font like Helvetica and some responsive scaling using flexbox properties; the final result brings an intuitive experience, while being easy on the eyes.

Text

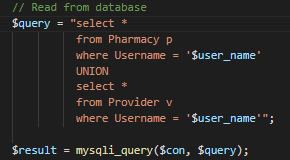
Description automatically generated Text

Description automatically generated

## Web Portal Implementation

### Function 1: index.php

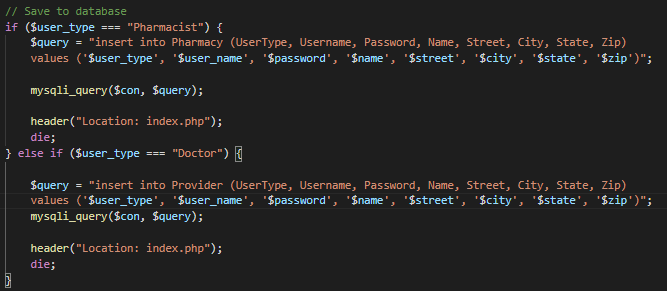
This portal will search both the Provider and Pharmacy tables to find matching username and password information. It sends the following query to determine where the username and password exist:



Depending on which user type is associated with the information found the user will be redirected to either the pharmacyHome.php page or the doctorHome.php page. If no matching records are found the system will echo an error message.

### Function 2: signup.php

This page allows a new doctor or pharmacy to sign up for the service. The page asks for the user’s profession, a username and password, their organization’s name, as well as their address. The password is required to be 8 characters long. Once the user has filled in all the fields, they can press the signup button and if there are no empty fields, one of the following queries will run:

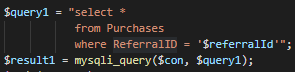


### Function 3 & 4: doctorHome.php and pharmacyHome.php

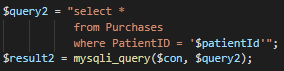
These pages are essentially identical to each other functionality wise. Both pages allow the user to navigate between their respective pages. The doctor has the option to lookup patient information or diagnose and write a prescription for the patient. The pharmacist has the option to either view prescriptions from doctors and record transactions or view their organization’s transaction history.

### Function 5: pharmHistory.php

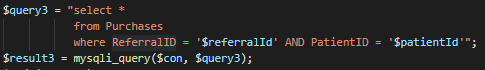
This page allows the pharmacy to see their transaction history. They can input either the referral ID, patient ID or both. This provides the option for three different use cases. Only filling in the referral ID will cause this query to run:



This makes it so that a pharmacist can view the unique referral without needing the patient ID. The information is then outputted into a table for easy viewing. If just the patient ID is filled in, the following query will run:



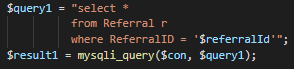
This query will allow the pharmacist to view the specific referral history of a given patient and will output the information into a table for easy viewing. If both the referral ID and the patient ID are filled in, the following query will run:

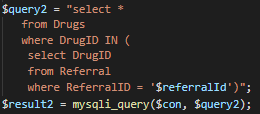


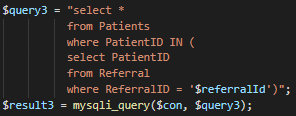
This should only bring up one result and is most useful when a unique patient wants to view a single referral of theirs. The result is displayed nicely in a table.

### Function 6: pharmSell.php

This page allows the pharmacist to view results from the Referral table and complete a transaction when the patient arrives to pick up their prescription. The page takes in a single referral ID and will run three different SQL queries based on the referral ID to display information from three different tables. These are the queries that will run as long as the referral ID field is not empty and the preceding query completes successfully:



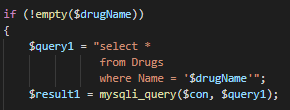




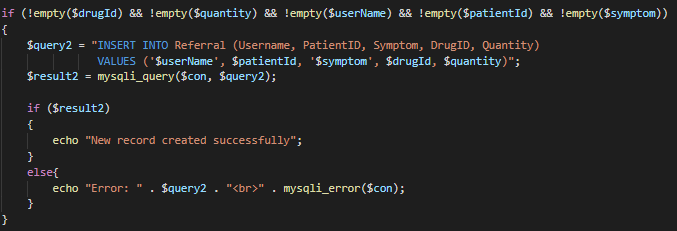
The page then takes all this data and displays it nicely for the pharmacist to view. When the pharmacist has verified the information with the patient, they can press the “Record Transaction” button, which will then run the following query and place the information into the Purchases table.

### Function 7: docDiagnose.php

This page allows the doctor to diagnose the patient and then make a referral for the pharmacy based on that diagnosis. The doctor will presumably know the name of the drug, but since the drug is not the primary key and there are multiple instances of different drugs in the Drugs table, they will first have to search the table using the drug ID. This query will produce the desired results:



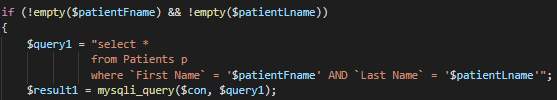
The data will be displayed nicely in a table so that the doctor can choose which version of the drug to prescribe. From there, he will input the drug ID, quantity, his own username, the patient ID, and the symptoms the patient was experiencing. As long as the fields are all filled, the following query will insert the information into the referral table, with the referral ID automatically incrementing:



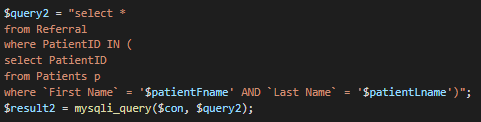
A pop-up message will be displayed if the insertion is successful. This will allow the pharmacy to view the referral upon the arrival of the patient.

### Function 8: docPatients.php

This page allows the doctor to input the patient’s first and last name. From there, the page will check if both input boxes were filled. The first query that will run will gather the patient's data from the patient table, and display it nicely in a table:



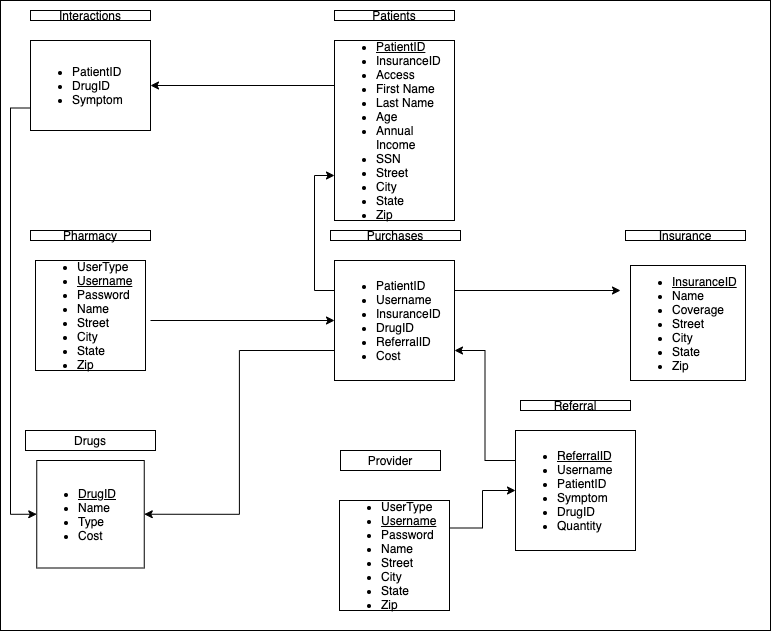
The second query will then run and gather the patient’s referral history for easy reference by the doctor. This will allow the doctor to make the right decisions when it comes to prescribing medicine. This data will also be displayed nicely in a table:



This functionality will allow the doctor to determine the best route forward to ensuring the patient is properly taken care of and that the process is as efficient as possible.

## Database Implementation

The database structure certainly has room for more complexity. However, given the time constraints and instruction we provided a basic structure focusing on the provider, pharmacist, confidential accessibility to patient history, referrals needed for certain purchases, cost associated with certain purchases regarding insurance, and a list of drugs for several tables to reference.



### Table 1: Drugs

The drug table consists of a list of drugs available for referral and purchase at the pharmacies. The primary key is the DrugID which is the unique index number assigned to the drug. The name of the drug. The type of drug which is either Prescription, Over the Counter, Controlled Substance, or a Device with prescription drugs and controlled substances needing a referral to purchase. Lastly, the cost per unit of the drug.

Table

Description automatically generated

### Table 2: Insurance

The insurance table consists of some basic information of the insurance company used to make the purchase at the pharmacy as well as the percentage of coverage. The primary key used is the InsuranceID which is a unique index number. Next, is the name of the insurance company, coverage of the patient under insurance company. We decided not to make plans for the insurance company given the focus was on provider and pharmacist. Finally, we have address information.

Table

Description automatically generated

### Table 3: Interactions

The Interactions table is protected by the patient table which is determined access by the bitwise access column located in the patient table. Once, the provider or pharmacist has access to the table they may view the patient’s history with certain drugs. These are referenced by the PatientID, the DrugID they had an interaction with, and whether their symptoms were mild moderate or severe.

Table

Description automatically generated

### Table 4: Patients

The patient table controls access to the interaction table as well as detailed information on the patient’s identity should the provider or pharmacist need to reference it. Each patient is assigned a unique PatientID which is the primary key. Next, we have the InsuranceID which is the current insurance that the patient has. The access column defines whether the patient has allowed access to their history being viewed by pharmacists or providers. Finally, we have several columns dedicated to listing the patient’s personal info such as first & last name, age, annual income, SSN, and address.

Table

Description automatically generated with low confidence

### Table 5: Pharmacy

The Pharmacy table contains data about the pharmacy. Also, it defines the usertype which determines the usertype associated with the username for the web interface. Next, it contains a unique username for each pharmacy as well as a password for security. Then, it continues to list the name of the pharmacy and its address.

Table

Description automatically generated

### Table 6: Provider

The provider table has the same construction as the pharmacy table with some key differences. First, the usertype is defined as doctor which is needed for the web portal to determine which homepage the user goes too. Also, instead of a company name it gives the doctors last name.

Graphical user interface, application

Description automatically generated

### Table 7: Purchases

The pharmacy table is a record of all pharmaceutical purchases within this system. However, the pharmacy accessing the purchases can only view purchases made by their pharmacy (i.e., their username in the username column). Within those select purchases the pharmacy can see the patient that made the purchase, which insurance they used for the purchases, the referral they had to purchase the drug, and the total cost for the transaction.

Table

Description automatically generated

### Table 8: Referral

The referral table is record of all referrals distributed within the system. Each referral is referenced by a unique index number its primary key. Next is the username which works similar to the purchases table in that the provider can only see their referrals given out. Next, we can see the PatientID to reference the patient given the referral. Symptoms are given as a justification to the referral given out along with the DrugID and Quantity prescribed.

Table

Description automatically generated with medium confidence

# Tools Used

## GitHub

Our team used GitHub for our code repository. This was decided a few key reasons; we each had experience in prior projects using this software, the GitHub app made uploading code very easy and therefore kept the team updated on the latest updates to the project, lastly GitHub had integration with the Discord app so not only was code updated quickly we were all informed.

Graphical user interface, text

Description automatically generated

## Discord

Discord was used as our primary communication channel. The entire team was very familiar with the software which gave us no technical difficulties when hosting meetings or conveying other information. Also, discord has an added screenshare feature which we used during conferences. This allowed us to see a team members screen so that we could see any information that would normally be harder to convey.

A screenshot of a computer

Description automatically generated with medium confidence

## Mockaroo

Mockaroo was used to implement fake data into our database. Mockaroo proved to be the simplest and time saving method for this process. After we had designed the tables, we could give them the appropriate names and use one of many data sets that Mockaroo contained. After that it was moved in our DB\_Build file to insert into the newly created tables.

A screen shot of a calculator

Description automatically generated with medium confidence

# How to Use - GitHub Link

Upon, opening the website (index.php), you will be prompt to login into your account or signup as a new member. Logging in only requires your specified username and password. Signing up requires you to input your information, also to choose your username and password. Once logged on, as a Pharmacist you can view your organization’s transaction history and input referral IDs from patients to ultimately record the transaction. As a doctor you can view a patient’s information and personal referral history. You can also diagnose illnesses and record referrals to the referral table for the pharmacist to view when the patient arrives to pick up their prescription. Logging out is possible by pressing the logout button in the bottom left.

**GitHub Repo:** <https://github.com/extremedawson/CS360-DB-Project>

# Challenges

## Implementation & Web Design Challenges

During the implementation phase, some challenges we encountered were figuring out how to get php to talk with the database. Also, adding an entry when someone signs up and verifying the inputted data to be correct. To add on, figuring out how to interact with the html for styling purposes, was a time eater too. As for styling, the only issues we have ran into were aligning all the elements in some sort of grid. Using the grid property would have been best for this problem but we wanted a little responsiveness to our website. Making us opt for the flexbox route and include a little bit of grid properties for data presentation. Lastly, figuring out how to style php echo statements, we found inline styling would be the best option.

## Functionality & Connectivity Challenges

Some challenges that Dawson encountered during development involved learning and implementing PHP, as well as using server hosting with the UI server and XAMPP (for local hosting). There was a significant learning curve that had to be endured to be able to develop at a faster pace. Once some of the initial hurdles were overcome, the development became much more streamlined.

## Database Challenges

A couple challenges were presented when designing and implementing the database, such as, determining how to implement the primary keys. We believed the best way was to assign the provider and pharmacy unique index numbers but the proved to be not the safest way for data integrity. Therefore, we implemented a unique username for each of the pharmacies and providers. We also had trouble making sure that the usernames we created were present in both the purchases and referrals table. While no concrete solution was thought out, we did isolate the usernames from the original tables and implement them as a custom list in the Mockaroo software.

## Web Server Hosting Challenges

Another challenge that we faced was that our file permissions within the UI course server would randomly be changed so that only one person could edit the files. We were able to work around this by having that person reassign the permissions, but this could at times cause some delay in development.

# Reflection and Conclusion

## Web Design Reflection

After, experience first-hand with all the quirks CSS brings, we realized that front-end developers are closer to full-stack developers than an artist who likes to code. They not only have to have the eye for design but also to understand and read functions provided by their team members. We also had to “go back” to art class and learn the color wheel again, in order to provide a harsh less experience. This project has taught us to apricate all the webpages with a high level of design and implementation, while also understanding the user comes first.

## Web Page Functionality Reflection

Implementation of PHP using SQL queries is no doubt a useful skill to have. We realized how useful the two can be when seamlessly integrated, especially when paired with properly styled webpages. It was incredibly useful to learn all these skills, and certainly opened our eyes to the possibilities of creating very useful tools to help people. Overall, the implementation of all these pieces went smooth near the end. It took a hard learning curve to understand how PHP and SQL can be used in tandem. At first development was shaky, but once we were able to implement working products, it made it much easier to build more features into the final product. There might be some improvements that can be made to some of the queries and how they are run procedurally. One way to maybe improve in the future is to use object-oriented PHP instead of procedural, however this reality may have manifested itself a little too late into the development process. It was ultimately a very useful learning experience.

## Database Reflection

In total this project demonstrated a lot of real-life practicality to database systems. We learned how something that seems simple such as columns or primary keys can have a major impact on user security. Also, the direction of how you implement the table flow can help regulate sensitive information. For example, not connecting the purchases table and the interactions table together but having the interaction route through the patient table allows the patient to allow or deny access to their medical history.