Christopher Blanchard
CS 670 Final Project

**Introduction:**

Migraine disease is a devasting illness that affects over 10% of the US population. While there are varying degrees of severity, the consequences of a migraine attack can be devesting to the individual experiencing it. This ranges from severe pains levels in the skull, to cognitive disorders such as slurred speech, to visual disturbances including light sensitivity and temporary loss of sight. While the disturbances pass, the unintended repercussions of these attacks can result in absence of important events, loss of employment, and subsequent depression and anxiety. While there is no cure, the purpose of this application was to help the user find patterns in these occurrences, to avoid stimuli that may cause these attacks to occur. My idea for this project occurred many years ago, as a loved one suffers from this disease. My thought was if there was a way for an individual to track all of this information, and then analyze the data to look for re-occurring patterns, then maybe one could discover a common specific item on all of the attacks. This could therefore help the user avoid these triggers to then lower their migraine attacks. There are many options currently available for such a use like N1-Headache [1], Migraine Insight[2], and an excellent app on the itunes app store named Migraine Buddy[3]. My solution of the application for this project – MIO (Migraine Information Organizer) does a fraction of what Migraine Buddy can do. With Migraine Buddy, you can actually choose an image of the skull in which the pain originates, to view at a later time. It displays Pressure Variation in the forecast, summary reports, and a way to track your relief from different medications. The application does quite a bit, but can almost be overwhelming, especially to non-technology proficient people.

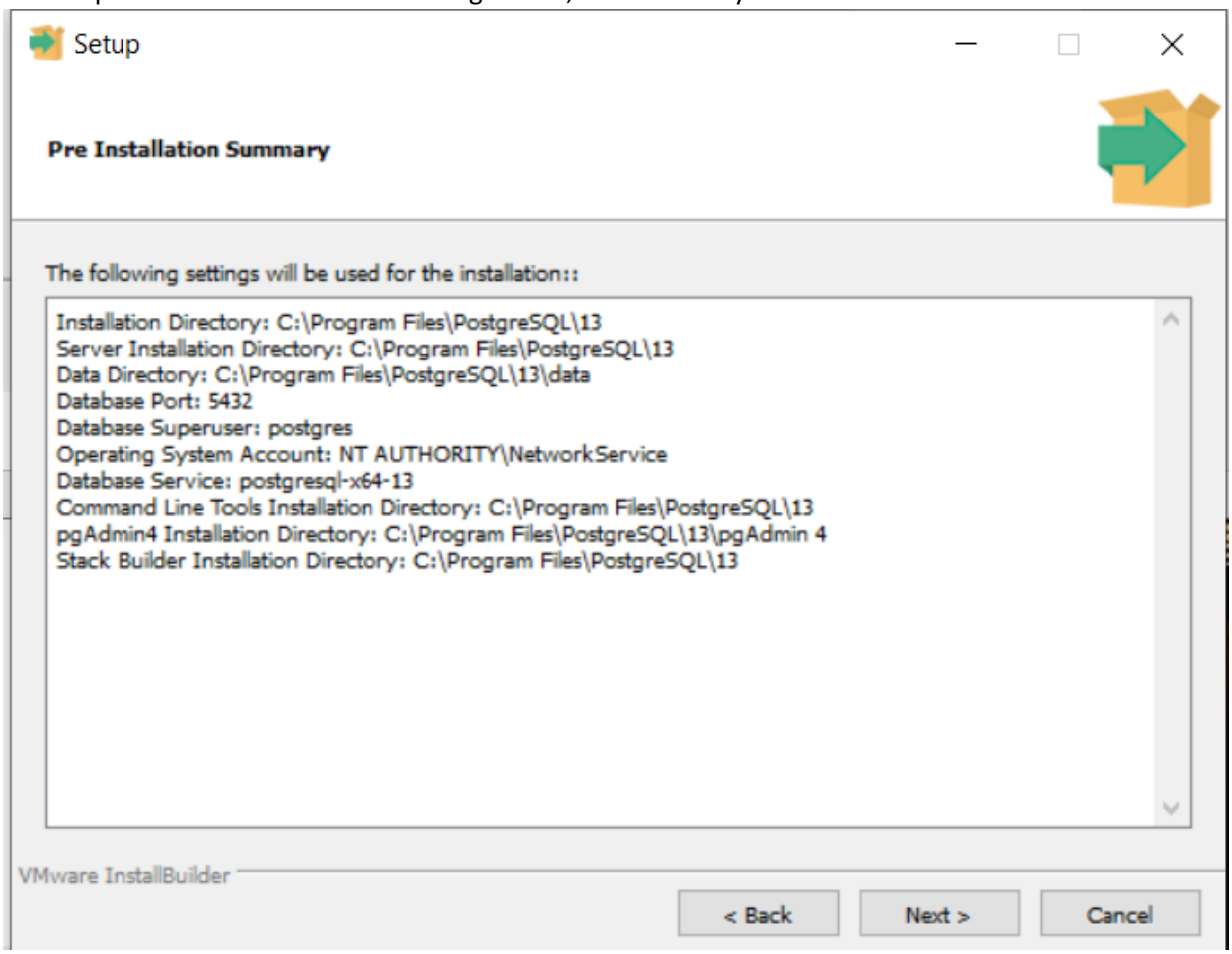
**Platform:**

I chose to create this application with Visual Studio Code and run on my local server, with a thought of potentially purchasing a domain and deploying to a production environment. After researching potential avenues for deploying applications, using Python for an OOP language and Flask as a web framework seemed like the best solution. My original idea of storing user information for retrieval at a later time required the use of a database, so the back-end used PostgreSQL and pgAdmin to view the database tables. SQLAlchemy which is an open-source SQL toolkit and object-relational mapper was used in creating the database models for the application.

**Project:**

My front-end design began by viewing Youtube videos creating basic web applications with Flask. With no prior experience in creating web applications, but a basic knowledge of Python and creating small programs in local IDEs, this was the most helpful in constructing the basic fundamental structure. So the video which I decided to follow to help to complete the core aspects of the project was Build & Deploy A Python Web App | Flask, Postgres & Heroku - YouTube, by Traversy Media. The first item I needed to accomplish was installing Visual Studio Code, which I did not have on my PC nor did I have prior experience with. In my courses at SHU, I only had experience with C++ in Visual Studio, which is a bit different than VS Code. Once installed, I had no history with installing extensions, so Python needed to be in installed as an extension, as well as Sublime text editor. Once VS Code was installed, PostgreSQL also required installation. I followed How to Install PostgreSQL & pgAdmin 4 on Windows 10 [ 2021

Update ] Step by Step Installation guide - YouTube as a guide in how to install the database. Again, I have no prior experience installing a databases other than using an older version of Oracle for creating a database for CS603 Database Design at SHU, however that was specifically for running from the command line and did not contain as many features as PostgreSQL. One issue I encountered in this process was during installation, pgAdmin4 was installed, which is a graphical interface user administration tool used with PostgreSQL, however the version installed was the newest version. This caused an error where the screen would continuously load in a loop. After researching, to resolve this issue, I installed an earlier version. I currently have pgAdmin v 4.9 while the most recent is version 5.0. The notes I found in StackOverflow pgadmin 4 v4.28 keeps loading - Stack Overflow, discusses how pgAdmin uses some additional security features, to prevent issues caused by content sniffing. As a result, some Windows systems (which I use Windows 10 as my OS), are misconfigured and can cause Javascript issues which results in loading failure, which was my issue.




*[Image 1]*

Once Visual Studio Code and PostgreSQL/pgAdmin were installed, the coding began. Again, the video proved extremely helpful in setting up the libraries needed to achieve my goal. I used a virtual environment in creating the application, which again I had no prior experience with or knowledge of. Apparently after researching this topic, the benefit of using a virtual environment is that it contains the libraries you use for your task in a singular container, and not globally, therefore eliminating issues creating multiple projects and all therefore affected by global library dependencies. This is utilized

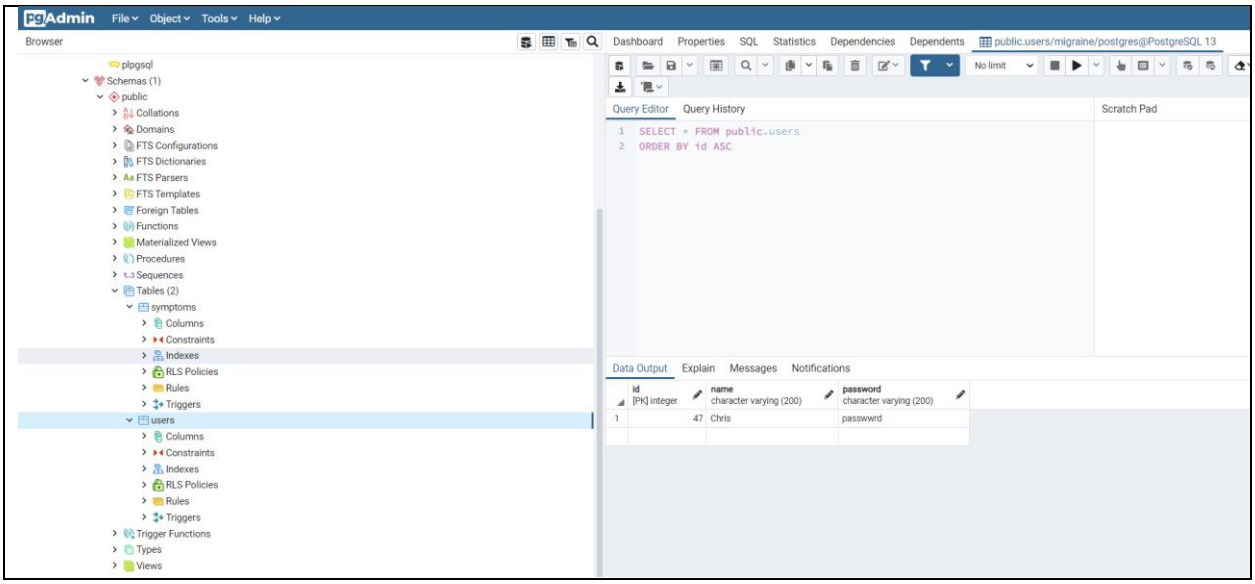
through the command line of Visual Studio Code, by installing the pip package manager. pip is a package management system written in Python used to install and manage software packages. Once pip was installed the following packages were then installed as well: flask, which is our web framework, psycopg2, which is a database adapter for PostgreSQL, and flask-sqlalchemy, which is used to construct our database models. The main python file, app.py was creating and I began importing the libraries, initializing the app, and creating our development environment. The application, also had to be configured to the database via an app config command and include the password used in creating PostgreSQL and logging into pgAdmin. Once this step was completed, the database models were created. My project consisted of only two models and tables, a User model and Symptom model. After the models were created, the following needed to be run from the VS Code command line:
CNTRL C stopped the server
Then type:  python
from app import db
db.create_all()
exit()
python app.py
This would then create the database models.

```python
#creating a new table for login and password storing
class User(db.Model):
  __tablename__='users'
  id = db.Column(db.Integer, primary_key = True)
  name = db.Column(db.String(200), nullable=False)
  password = db.Column(db.String(200), nullable=False)
  children = db.relationship("Symptom")
```

*[Image 2]*

```python
#SQLAlchemy uses models, like Mongoose or SQLize. Create in form of a class
#creating table columns 'fields'
class Symptom(db.Model):
  __tablename__='symptoms'
  id = db.Column(db.Integer, primary_key = True)
  symptom = db.Column(db.String(200))
  trigger = db.Column(db.String(200))
  rating = db.Column(db.Integer)
  comments = db.Column(db.Text())
  symptomtime = db.Column(db.String(200))
  parent_id = db.Column(db.Integer, ForeignKey('users.id'))
```

*[Image 3]*




*[Image 4]*

After the models were created, Flask routes needed to be created and assigned to the proper paths. The accompanying html files then need to be created and linked. In my first run through, one of my major issues was essentially having the submit button that was used at the login page, to submit the symptoms to their applicable tables. The submit button simply was not working. It was only after many attempts that I found that the submit form action needs to be assigned in the html file if you plan on using a submit button for other functions. Because this was only assigned once, the second submit button did not do as I needed. My front-end knowledge is extremely limited, so after many frustrating attempts and quite a bit of internet research, I was able to discover this necessity.

In viewing the video, the use of Flask routes was the main emphasis. The route needed a corresponding path associated with it for example, so the first path was routed to the index html page:

```
@app.route('/', methods=['POST', 'GET'])
def index():
  session['username'] = ""
  session['id'] = ""
  if 'username' in session:
    username = session['username']
    userid = session['id']
    name = ''
    password = ''
```

```
  if request.method == 'POST':
    name = request.form['name']
    password = request.form['password']
    if name == '' or password == '':
      return render_template("index.html", message
='Please enter a username and password.')
    elif db.session.query(User).filter(User.name =
= name).count() == 0:
      data = User(name=name, password=password)
      print(data)
      db.session.add(data)
      db.session.commit()
      session['username']=request.form['name']
      username = session['username']
      return render_template("profile.html", name=
name, username=username,message='Welcome to MIO:')

    session['username']=request.form['name']
    username = session['username']
    print("UserID:", username)
    print("UserID Test:", userid)
    print("Testing username:",username)
    return render_template("profile.html", name=na
me, message='Welcome back:')
  return render_template('/index.html')
```

*[Image 5]*

Below is the accompanying index html page. You will notice the code uses {} quite often. This is Jinja [Jinja — Jinja Documentation (2.11.x) (palletsprojects.com)](), a modern and designer-friendly templating

language for Python, modelled after Django's templates. I had never heard of this prior to this project, and was very impressed how it works as this was the method for displaying values passed from the Flask routes. Furthermore, I used block content, to apply a navbar for quick navigation through the page. I discovered this feature by viewing Codemy on Youtube: [(1) Templates, Bootstrap Navbars, and Links - Flask Fridays #4 - YouTube]. This video really opened my eyes to some extremely useful tips in using CSS and Javascript. I plan on utilizing this much more once I finally correct the functionality of the database users and the one-to-many relationship I was looking for.

```
{% extends 'base.html' %}
{% block content %}
{% with messages = get_flashed_messages() %}
  {% if messages %}
    {% for message in messages %}
      <p>{{ message }}</p>
    {% endfor %}
  {% endif %}
{%endwith%}

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-
width, initial-scale=1.0">
  <meta http-equiv="X-UA-
Compatible" content="ie=edge">
  <link rel="stylesheet" href="../static/style.css
">
</head>
<div class="form-group">
<body>
```

```
  <div class = "container">
    <form action = "/" method="POST">
      {% if message %}
      <p class="message">{{ message | safe}}
      </p>
      {% endif %}

    <h1>Welcome to MIO - Migraine Information Orga
nizer</h1>
    <h3> Please enter a username and password to b
egin</h3>
    <p>Username:</p>
    <p><input type = "text" name = "name"/></p>
    <p>Password:</p>
    <p><input type = "password" name ="password" t
ype = "password"/></p>
    <p><input type = "submit" value = "Submit" for
matcion = "POST" class = "btn" /></p>
    </form>
  </div>
</div>

</body>
</html>
</body>
</html>
{% endblock %}
```

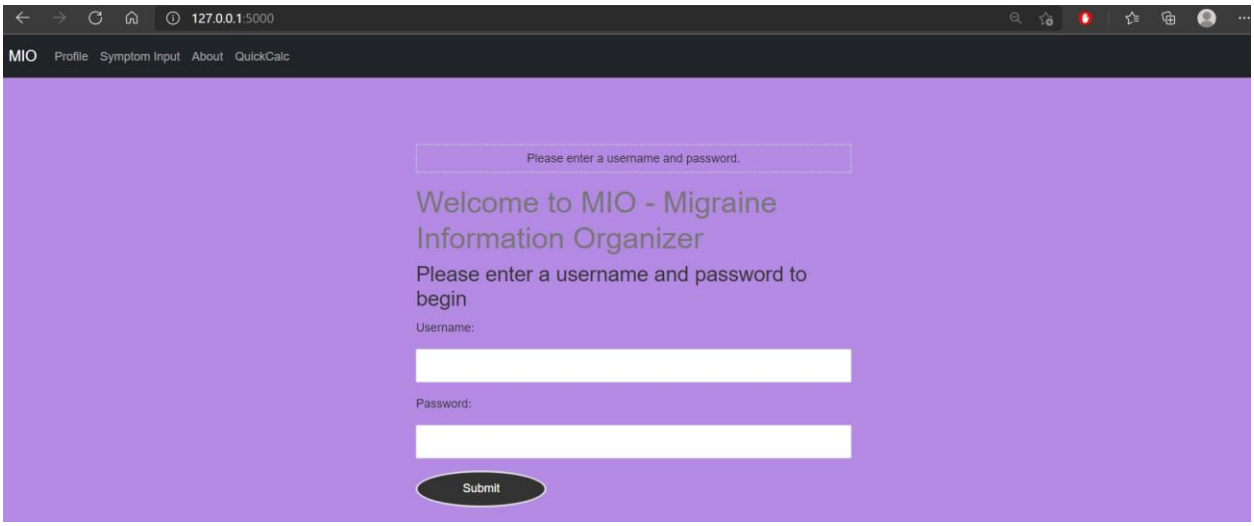
*[Image 6]*

I did not construct database tables ERD diagrams, pseudo code, or flow charts. In retrospect I should have completed this, but in my mind the main concept of the application was simple, to have input/output with two tables. I wanted the user to input key metrics of their specific symptom, a rating, comments, and what they believed was the trigger. I wanted the application to give the user a summary based on what their most numerous triggers were, and as an indicator these show a pattern and should inform the user they should be avoided. I knew a foreign key would need to be included in the symptoms table, to link to a corresponding user table. But given how I basically learned almost all of the technologies as I progressed, I missed one of the most important concepts of the project, by not being able to create a foreign key based on the user table primary key. As a result, my data was not linked and my queries were not functioning properly. Toward the end of the project, I spent hours trying to determine how to input a foreign key, and began watching videos and scouring StackOverflow on information for creating user sessions and ways to track the user once they logged into the application throughout the different forms. I had no idea how tedious the process was, and what began as a simple project in theory ballooned into a frustrating attempt of deleting database models and creating new tables in an effort to: 1) capture the information from the forms correctly and 2) to retrieve it correctly but also based on the user's primary key. Throughout my code, you will find links to libraries like WTForms, which I would like to attempt to utilize in continuation of this project beyond my educational pursuits. Apparently WTForms: WTForms — WTForms Documentation (2.3.x) is a flexible forms validation and rendering library for Python web development, working with whatever web framework and template engine you choose, and supports data validation and CSRF(Cross-site request forgery, which apparently is very important) protection. Just the mere process of logging in a user is a course by itself. I found myself completely overwhelmed, and once I attempted to use this functionality, I only made the project more convoluted and broken. This feature needed a class, which did not work correctly with my current index route, so I was basically splitting the project up yet again in terms of which functionality I chose to embrace. In the end, I chose to revert back to the original functionality and sticking to Flask routes and corresponding html forms.

**Results**

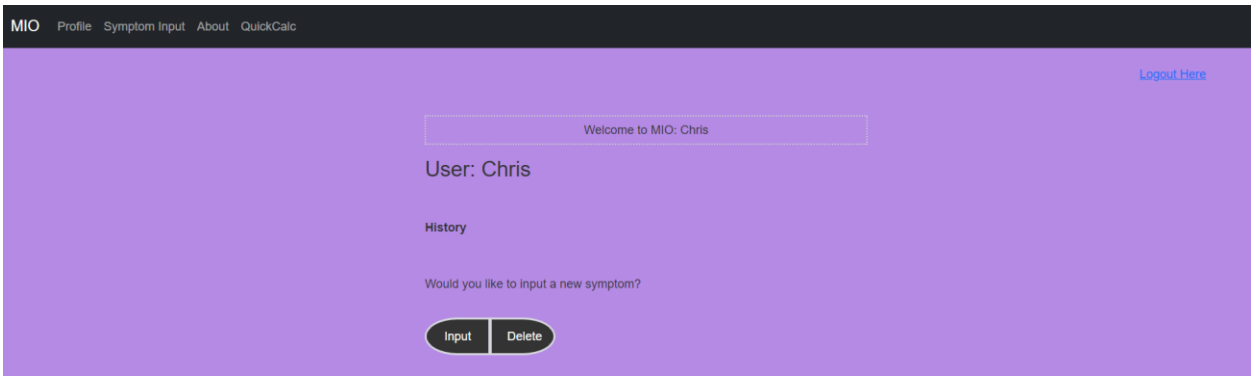
My final outcome fell short of what I intended, but I was able to accomplish two points: 1) to input data into a database from the web application and 2) to provide some sort of output back to the user to summarize their input.

The first page starts plainly enough, by prompting the user for username and password. If no information is present, a pop-up asks them to include the missing information. It will accept any username and password, and if new, creates a welcome pop-up on a profile summary screen. If they are returning users, it checks the database for a duplicate name (I know this is not at all a useful process but for the purpose of this project I chose this path), and if it is a duplicate, prompts the user with a "Welcome back" message.



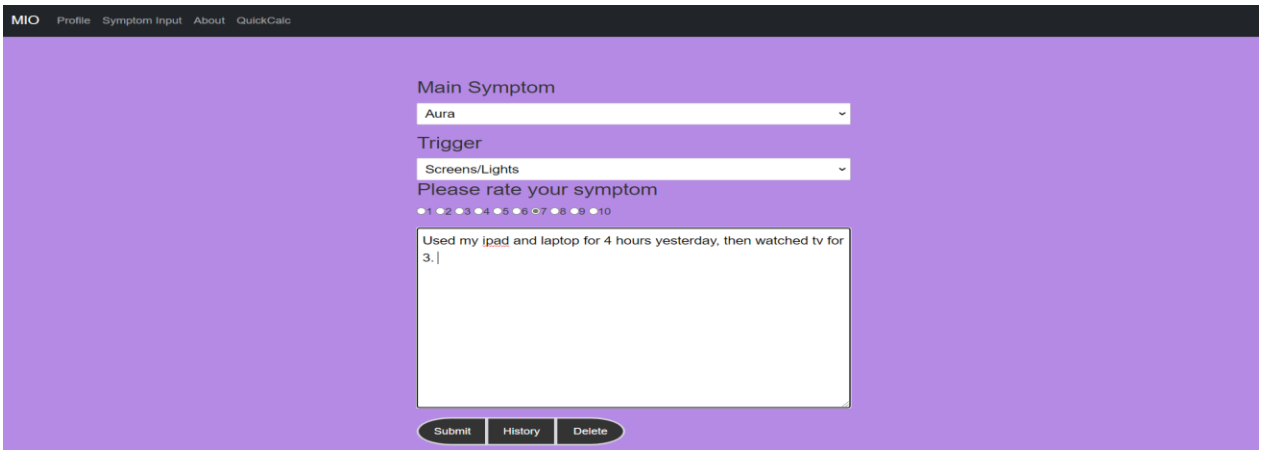

*[Image 7]*

In this specific case I was already a user, so I was welcomed back. At this point the symptoms should appear under the History header, but because I was not able to link the two tables based on user, it does not. I also provided a “Logout here” link, which brings the user back to the index page, and assigns the user session to None.



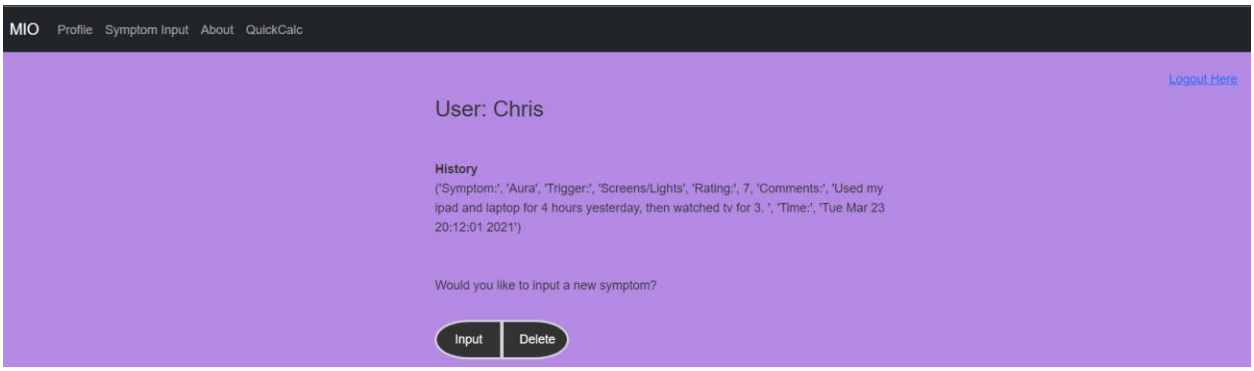

*[Image 8]*

By selecting input, this would bring the user to the Symptom route and html page. This would then allow the user to input their symptom, trigger, a rating, and additional notes for storage.



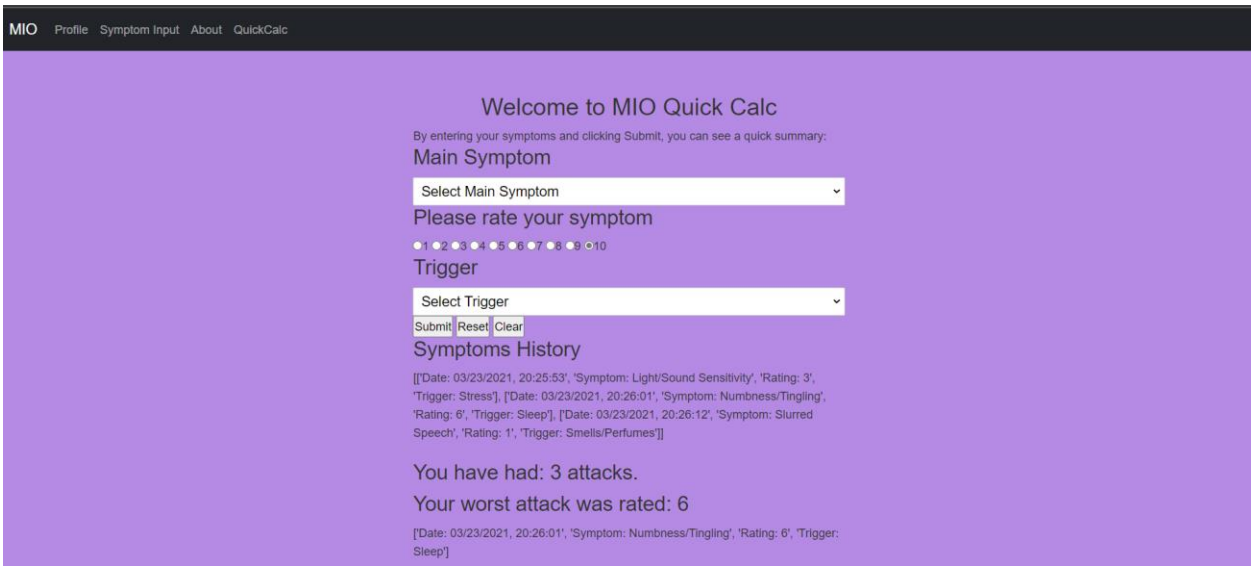

*[Image 9]*

Once submit was clicked, this would bring the user back to the profile page, with their data summarized in a list. The plan was for this to then give the user feedback on their most numerous triggers and symptoms. I was not successful in creating the logic to do this, as I spent most of my time troubleshooting first my html submit button issue, then constructing the database and trying to link the child table to the parent table. I then essentially became lost through a lot of information on best practices tracking a user through form navigation.




*[Image 10]*

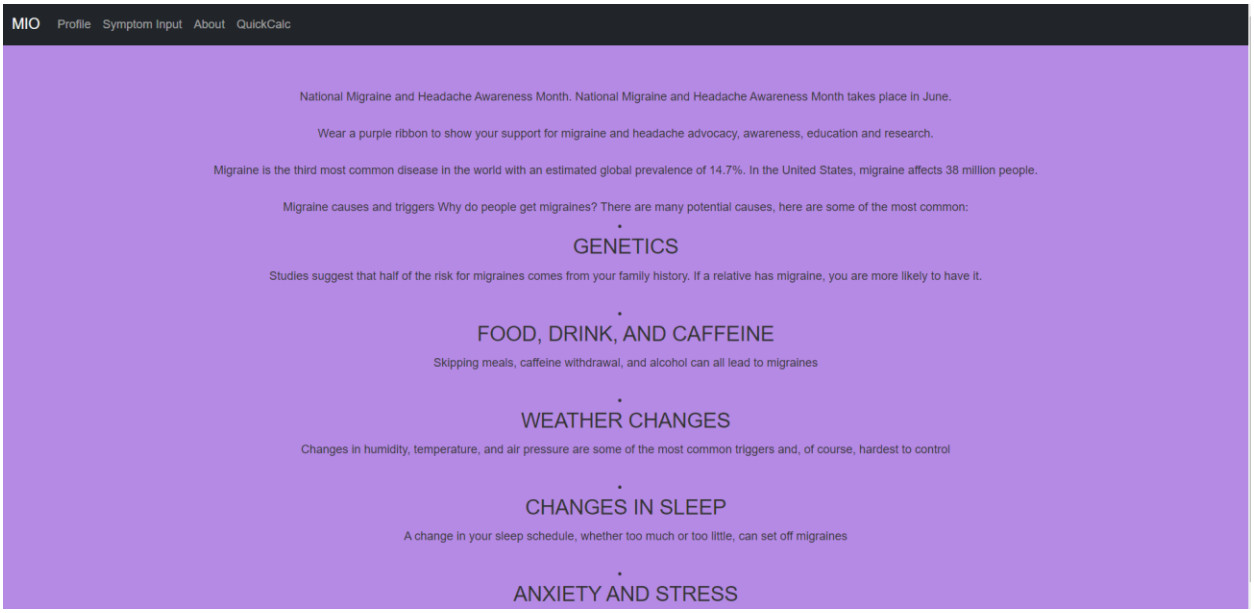
I also added a QuickCalc tab in the navbar, to allow the user to essentially complete the main concept of the application for a quick summary:




*[Image 11]*

This does not store any information in a database but rather appends to a list in the uses a counter and max value to summarize the information for the user, so in this particular case the worst symptom of the migraine was numbness/tingling and was brought on by what the user can define as sleep related. This functionality is what I wanted my profile to display to the user when they logged into the application, and allowed them to add/modify/delete these values and provide a summary for them to track their potential most impactful culprits and help them avoid.

Finally, I added an About page, with some basic information for the user about migraines. It would be nice to expand upon this, and provide useful information regarding medical research, treatments, etc.




*[Image 12]*

**Conclusion**

In my development phase, I encountered numerous obstacles. I did not fully grasp the complexity of gathering user credentials and tracking input to a database based on the user. In my Database Development course at SHU, I created a database and input values, but not in this nature. The project quickly morphed into many problems to troubleshoot and I simply ran out of time. The last two weeks specifically I spent an enormous amount of time trying to utilize different concepts I found to my question of correctly assigning a foreign key based on a primary key value, but would make a change and break the application. In the future, I would like to post this to Github, which is useful for essentially keep all code in a static setting, and updates made would be to new versions, rather than making changes and risk breaking my application completely when trying new ideas. While I was ultimately disappointed in the final outcome of my project, I learned quite a lot in short amount of time, I believe the project itself brings awareness to this disease, and I have a great foundation to complete this application in its entirety and how I intened in the future.

**References**


1. "N1-Headache", N1-Headache™ Pricing (n1-headache.com), https://n1-headache.com/patients
2. "Migraine Insight", Migraine Tracking App - Fix migraines | Migraine Insight, https://migraineinsight.com
3. "Migraine Buddy", Migraine Buddy |#1 Migraine Monitoring App & Headache Tracker App, https://migrainebuddy.com