FINAL REPORT

COMP4150

ADVANCED AND PRACTICAL DATABASE SYSTEMS

Mental Health Database Application

Group 4:
Dennis Dao, Tyler Johnston, Andrew Simon

Submitted to:
Dr. Pooya Moradian Zadeh

November 29, 2023

Mental Health Database Application

Dennis Dao
School of Computer Science
University of Windsor
Windsor, ON, Canada
dao114@uwindsor.ca

Tyler Johnston
School of Computer Science
University of Windsor
Windsor, ON, Canada
hedgesjt@uwindsor.ca

Andrew Simon
School of Computer Science
University of Windsor
Windsor, ON, Canada
simon31@uwindsor.ca

***Abstract*—This document outlines the specifications for a Flask application designed to manage mental health information within a secure and user-friendly framework. The system encompasses various components, including a well-defined database schema, API documentation, user interface design, and robust security measures. User stories depict the interactions between clinicians, patients, and administrators, ensuring a comprehensive understanding of system functionality. Emphasis is placed on data security, encryption protocols, and compliance with healthcare privacy regulations. The testing plan covers unit, integration, and user acceptance testing, ensuring the reliability and accuracy of mental health data handling. Deployment strategies, documentation requirements, and considerations for scalability round out the specifications, providing a comprehensive guide for the development, deployment, and maintenance of the application. The document aims to address the unique challenges posed by the sensitivity and confidentiality of mental health information, ensuring that the system meets high standards of security and usability.**

***Index Terms*—Flask, database, mental health information, web application, front-end, back-end, database scheme, API, authentication, authorization, data encryption, user roles, deployment**

Project Overview

We have created a functional mental health database along with a GUI for navigation. This was done by implementing a Flask server and creating a SQL database. The motivation for this is to be able to aid specialized health professionals around the world in mental health research as well as counseling/therapy statistics for them to help their clients better. This is an important issue to solve because a lot of time is wasted in counseling when trying to figure out what a patient is looking for. This application will help the doctor create a list of most likely methods to try first.

Andrew Simon worked on the database itself and the Python script to put more data into the database. Tyler Johnston worked on creating the PowerPoint presentation and parts of the final report. Dennis Dao worked on the application communicating with the database, ensuring it was secure and correct.

Project Details and Methodology

Development for the database took place in DB Browser for SQLite while the development for the Flask application took place in VSCode using Python.

For the database, we created a data standard, enabled complex queries, built a schema from scratch to connect the tables together, and we made sure to focus on the relations.

For the application, it acts as the layer for end-users to interact with the database after being authenticated. It was designed with security in mind and envisioning how an end-user would try to interact with the system (front-to-back).

Definitions

- Flask: A lightweight and web framework for Python. It is designed to be simple and easy to use, making it suitable for developing web applications, including those handling mental health information.
- Database: A structured collection of data that is organized and stored for efficient retrieval and management. In the context of your application, it would store and manage mental health information.
- Mental Health Information: Refers to data related to individuals' mental well-being, including assessments, diagnoses, treatment plans, and other relevant details.
- Web Application: A software application that runs on a web server and is accessed through a web browser.
- Front-end: The user interface and user experience layer of a web application, including the visual elements that users interact with.
- Back-end: The server-side of a web application responsible for processing requests, managing databases, and performing other server-side operations.
- Database Schema: A blueprint that defines the structure of a database, including tables, fields, relationships, and constraints.
- API (Application Programming Interface): A set of rules and protocols that allows different software applications to communicate with each other. In your case, it may include endpoints for interacting with the mental health database.
- Authentication: The process of verifying the identity of a user or system. It ensures that only authorized individuals have access to sensitive mental health information.
- Authorization: Determines the permissions and access levels granted to authenticated users, controlling what actions users are allowed to perform within the application.
- Data Encryption: The process of converting data into a coded format to secure it from unauthorized access. It's

crucial for protecting the confidentiality of mental health information.

- User Roles: Defines the specific permissions and responsibilities assigned to different types of users within the application. For example, clinicians may have different roles than patients.
- Deployment: Refers to the process of making a web application accessible to users. It includes activities such as configuring servers, setting up databases, and ensuring the application is secure and performant.

Specifications

Flask is a web application framework written in Python that was designed to be easily learned by those who have experience in HTML and CSS. SQLite is a relational database system that is written in C. SQLite is used over SQL because it is server-less, so the database is integrated into the software.

We use the bcrypt module to encrypt passwords, re (regular expressions) module to validate input, the datetime module to validate login time, and sqlite3 modules from Python to complete this application.

As mentioned in the overview, time can be wasted in counselling when trying to find a treatment option. The application uses the Flask framework and a SQLite database of mental health information around the world to help pinpoint a list of treatment options for doctors, no matter their location. Users should be able to do so simply by logging in and selecting the option they know they need to view and/or use the search function to narrow down their options. Figure 1 displays a sample use case of searching for a condition.

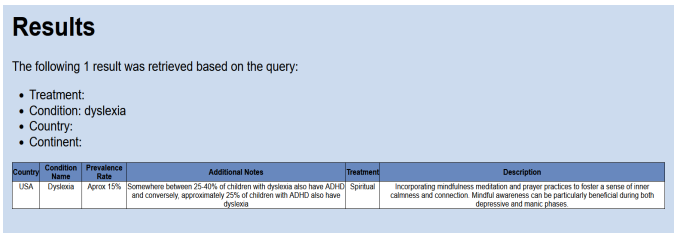

Figure 1. Search Use Case; Source: from design

Thus, this application should reduce the amount of time required for doctors to determine treatment options for their patients.

Architecture

The application is on a 2-tier architecture, where the Flask application server acts as a layer for clients to communicate with the SQLite database. The application server handles all the data management while the database side has the DBMS, leaving possible scalability in the future.

Platform

The database is hosted on a Flask platform with an application to interact with aspects of the database. Flask was chosen due to the lightweight nature for an application and the compatibility with using Python and the SQLite3 module. The database itself is a SQLite database, which is a simple and lightweight database. We use SQLite since it works well in websites/web applications as mentioned in [1], which is compatible with our web application.

Design

We designed the database to have five main tables - *MentalHealthConditions*, *Countries*, *CountryMentalHealthData*, *TreatmentApproaches*, and *Users*. Figure 2 displays the schema of the database for the first four tables since the Users table can stand on its own.

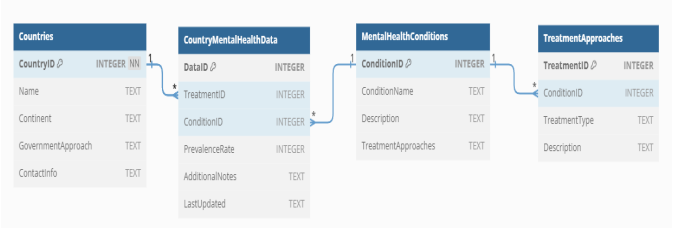

Figure 2. Database Schema; Source: from design

A country can have many amounts of data on a mental health condition. A single mental health condition can be treated with many approaches and can be in many countries' datasets. Users can access this data and if marked as an admin, they can also manage more of the database contents. The account data to login users is encrypted for security purposes. Queries are prepared statements (ie. the parameter is passed later) to protect the database from unauthorized and unintended access.

We designed our UI with Flask to be intuitive, prevent unauthorized users from accessing potentially sensitive information, protect any data the user puts in, and allow the user to look at the information in the database in a quick and easy way.

Implementation and Experimental Setup

Our implementation setup uses the VSCode editor and DB Browser for SQLite [2] to create the application. VSCode provided tools and space to create the Flask application in Python while DB Browser allowed data in the database to be changed quickly, allowing for efficient testing with different sets of data.

Implementation Details

The Flask application was implemented in Python and the basic data input was initially done with DB Browser for SQLite. Data input transferred to the application itself once it was implemented to further test the functionality.

Application development began with starting at the homepage and slowly building up by thinking in the perspective of the end-user, moving from page to page. Upon logging in, a session variable will be set to the user's browser to indicate they are logged in. With a user session variable, end-users can view information, With an admin session variable, end-users can view and manage information. A search page was available to narrow down options for users to view. Lastly,

the admin page was given options to manage contents in the database, creating, updating, and deleting records.

TESTING

To test the database application, the forms to input data were given valid and invalid data for the application to validate before running the specified operations on the database. Example use cases include registering (creating) an account and viewing access. Strategies used include unit, integration, and user acceptance testing. DB Browser was also used to test use cases under different data sets in the database.

When creating an account, fields have various specifications. Usernames should be between 4-25 characters with at least one number and no special characters, passwords should be at least 12 characters and include at least one uppercase, one lowercase, one number, and one symbol. Other fields require input to be accepted. Therefore, we test with input data that both passes and fails the criteria (including inputs that could compromise the application or database, such as SQL injection). Figures 3, 4 and 5 show sample inputs where users input data to the database, one with a failed SQL injection attack, one with a safely completed registration, and one with errors displayed.

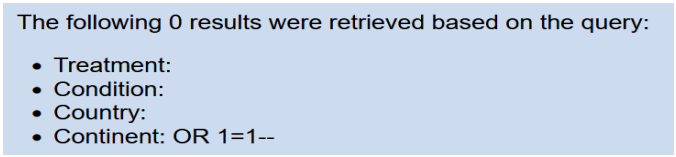

Figure 3. Attempted SQL Injection; Source: from design

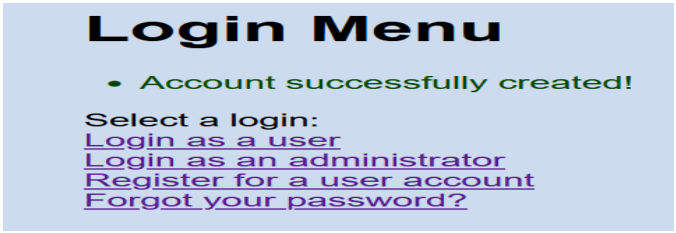

Figure 4. Successful Registration; Source: from design

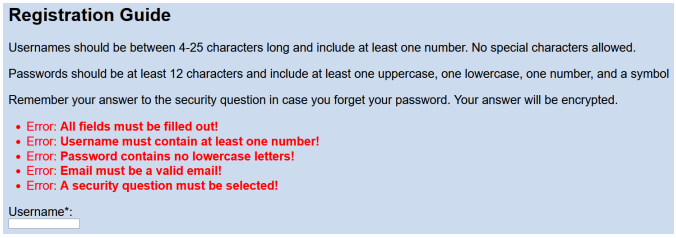

Figure 5. Registration Error; Source: from design

If the user views a page that requires an active login session but does not have one, they should be given an error that they must sign in to view the content. When logging in, the username and password should exist, otherwise a generic error message is given. Additionally, the login time should not be earlier than what is recorded in the database (ex. manual change computer date to July 5, 2023, but the last login was November 22, 2023. Unit testing was performed on each module to ensure the operation was complete and secure. Once that was complete, integration testing was done to perform operations on modules in sequence one after the other, ensuring they could work together. Lastly, we do user acceptance testing to make sure nothing has been overlooked and no bugs remain. This testing allows the application and the database to be complete, secure, and correct.

DISCUSSION

Achievements

- Created database to store various information including mental conditions, treatments, prevalence by country and others.
- Created secure and efficient Flask application to interact with database.

Challenges

- Standardization of data input due to different sources having different data in varying formats, as well as how those sources collected that data, ensuring that all information in the database reaches a consistent format. Presently, we format contact info through name then phone/email. For dates, we format as YYYY-MM-DD.
- Separating what details on mental health was more important than others, getting the right balance between having extra information and overloading the database and not having enough related information for it to be useful to the users. Presently, this depends on how much information users supply when entering the field.
- Finding appropriate ways for dealing with mental illnesses within the many different categories, dyslexia cannot be "cured" for example, however those with dyslexia may suffer symptoms that can be dealt with through medication, such as anxiety or symptoms like ADHD making stimulants effective. We solve this by providing a search function to search for treatments by attributes such as condition.
- Initially figuring out how to work with Flask - it was our first time working with the framework. Application development relied on prior experience with PHP but required figuring out the equivalent functions/procedures required in Python. Researching documentation in [3] and [4] allowed us to determine the equivalent procedures.

CONCLUSION

The application was able to be successfully implemented by using a Flask server, and a SQLite server. In the future, our application can be expanded on by adding functions such as tele-health, getting more professionals involved that are experienced in mental health-related fields, as well as collecting more data for more accurate results.

REFERENCES

[1] "Appropriate Uses For SQLite". *SQLite*. https://www.sqlite.org/whentouse.html (accessed Oct 25, 2023).

[2] "DB Browser for SQLite". *DB Browser for SQLite*. https://sqlitebrowser.org/ (accessed Oct 25, 2023).

[3] "Flask | Read the Docs". *Read the Docs*. https://readthedocs.org/projects/flask/ (accessed Nov 15, 2023).

[4] "Welcome to Flask - Flask Documentation (3.0.x)". *Flask*. https://flask.palletsprojects.com/en/3.0.x/ (accessed Nov 15, 2023).