Client Query Management System

NAME –: RAHUL RAJ

BATCH CODE –: DS-C-WE-E-B8

PROJECT-NAME –: CLIENT QUERY MANAGEMENT SYTEM

OVERFLOW OF PROJECT

1. INTRODUCTION	3-5
2. KEY FEATURES	6-7
3. TECHNOLOGY USED	8
4. WORKING PRINCIPLE	9-11

5. FEATURES IMPLEMENTED	12
6. IMPLEMENTATION STEPS	13
7. FUTURE SCOPE	14
8. CONCLUSION	15

1. Introduction

In the modern digital age, businesses and service-based organizations rely heavily on efficient communication systems to interact with customers and resolve their concerns. As the number of users increases, so does the volume of queries, complaints, and feedback they generate. Handling these manually becomes difficult, time-consuming, and error-prone. This challenge has created the need for automated systems that can collect, store, and manage customer queries in an organized and efficient manner. The Client Query Management System (CQMS**)** developed using Stream lit and MySQL aims to solve this problem by providing a simple, fast, and structured way for clients to submit queries and for admin to manage them effectively.

The main idea behind building this system is to create a smooth communication link between clients and the admin. Many organizations struggle with unstructured query collection, such as queries sent through WhatsApp messages, handwritten notes, emails, or phone calls. These methods are difficult to track and often lead to missed issues, delayed responses, and dissatisfied clients. A digital platform that centralizes all communication ensures that every query is recorded, tracked, and resolved on time. This project implements such a solution while keeping the interface simple enough for non-technical users.

The project uses Stream lit, a popular Python-based framework that helps developers quickly build web applications without needing advanced front-end development skills. Stream lit is known for its simplicity, real-time responsiveness, and ability to turn Python scripts into interactive websites. It allows the developer to focus on logic rather than UI complexity. For database operations, MySQL has been used due to its reliability, scalability, and widespread use in real-world applications. MySQL is a relational database that handles large amounts of data efficiently and supports complex queries, making it suitable for production-grade systems.

This Client Query Management System includes two major roles: Client and Admin. The client-side interface allows users to register themselves, log in securely, and submit queries along with necessary details such as name, email, mobile number, query heading, and description. Once submitted, the query is automatically saved in the MySQL database. This ensures that no information is lost and that each query is assigned a unique ID for tracking. The aim is to provide clients with a simple experience where they can raise concerns without any confusion.

The admin-side interface provides tools for the admin team to view all incoming queries in a structured table format. Each query is displayed along with important details including the client's identity, problem description, submission time, and current status. The admin team can then take appropriate action on each query. A special feature of the system is the "Close Query" button that allows admin staff to update the query status once the issue is resolved. This helps maintain transparency and clarity in the entire process. Having an automatically recorded 'closed time' helps for audit trails and performance evaluation.|

One primary motivation behind this project is to demonstrate the practical use of Python and database technologies in solving real-world problems. The system reflects the core principles of software engineering: simplicity, usability, and reliability. It also helps students and developers understand how to integrate a web framework with a backend database, manage user authentication, and build CQMS (Create Query Management System) functionalities. The inclusion of login and registration ensures that only authorized users can access sensitive data, adding a layer of security to the application.

Client Query Management System is also available on any browser of mobile phone, tab, laptop, and desktop with proper responsive manner with the help of GitHub and Stream lit app. SQLite is being used in the GitHub instead of SQL because If the client is using his desktop or laptop then his browser is not connected to any database so all the data is already available in SQLite which is uploaded by GitHub and when the internet connection of the client is on then it connects to that webpage and when the client fills his details then it gets displayed to the admin through SQLite. There is link through we have go to this webpage: https://rahul-rajminiproject-le68mdhn2sfhgqdwbau23c.streamlit.app/.

Another important aspect of this project is improving the user experience between clients and organizations. When clients raise a query, they want quick responses and clarity. Having a digital platform builds trust and professionalism. The system allows the admin team to handle multiple queries efficiently and ensures that none of them go unnoticed. It also helps in maintaining digital records, which can be used later for analytics, performance monitoring, and identifying recurring issues.

In conclusion, this project demonstrates how technology can simplify communication between clients and admin. It shows the practical implementation of Python, Stream lit, and MySQL in building a reliable and user-friendly web application. By automating the process of collecting and managing client queries, the system improves efficiency, reduces errors, enhances transparency, and increases customer satisfaction. This introduction lays the foundation for understanding the project's objectives, features, architecture, implementation, and future enhancements. The following sections of the report will go deeper into system design, workflow diagrams, database schema, code explanation, challenges faced, testing results, and final conclusions.

2. Key Features

• Client registration and login

A client registration and login system is a secure method to ensure that only verified users can access personalized features of an application. In this system, a client first registers by providing details such as name, email, mobile number, and password. The password is encrypted before being stored in the database to maintain security. During login, the system checks the entered email and verifies the password by comparing the encrypted form with the stored hash. After successful authentication, the client is granted access to their dashboard where they can submit queries and view their previous records. This system ensures proper data protection, prevents unauthorized access, and links each query accurately to the correct client.

• Form submission for queries (name, email, mobile, heading, description)

A user can perform their

Name – username.

Email – user personal email.

Mobile – user personal mobile number.

Heading – The heading of client query.

Description – The full details of client query.

• Secure storage of queries in MySQL

Secure storage of queries in MySQL ensures that all client-submitted information is safely recorded, organized, and protected from unauthorized access. When a client submits a query, the data is inserted into a structured table using parameterized SQL statements to prevent SQL injection attacks. MySQL assigns each query a unique ID, timestamps it, and stores details such as name, email, heading, and description. Proper indexing and access control help maintain data integrity and improve retrieval speed. Only authenticated admin users can view or update the query status. This secure storage process guarantees reliability, confidentiality, and smooth data management throughout the system.

• Admin login to view all queries

Admin login to view all queries allows authorized team members to securely access every client request stored in the system. When an admin user logs in with a valid username and password, the system verifies their credentials using hashed values in the database. After successful authentication, they are redirected to a dashboard that displays all submitted queries in an organized table format. Admin staff can filter, sort, or search through queries to respond efficiently. Access control ensures that only authorized personnel can view sensitive client information. This login system helps maintain privacy, improves workflow, and ensures that all client issues are handled quickly and professionally.

• Query status tracking: Open and Closed

Query status tracking with **Open** and **Closed** labels helps the system stay neat and easy to understand. When a client submits a new issue, it automatically enters the system as **Open**, signalling that action is needed. Admin staff can review the details, respond, and work toward a solution. Once the problem is fixed, the status is updated to **Closed**, showing that no further action is required. This simple two-state system keeps both clients and admin teams on the same page. It also helps avoid confusion, since every query clearly shows whether it still needs attention or has already been resolved.

• Close Query feature with timestamp

The Close Query feature adds clarity and accountability by attaching a timestamp whenever a query is marked as resolved. When admin staff finishes working on a client's issue, they simply update the query to **Closed**, and the system automatically records the exact date and time of closure. This helps create a clear history of each interaction, showing how long a query remained open and when it was completed. The timestamp also improves transparency, making it easier to evaluate response efficiency and maintain proper records. Clients can see when their issues were officially resolved, and the system becomes more reliable for future audits or reviews.

• Dashboard-like view for admin users

A dashboard-like view for admin users provides a clean, organized, and easy-to-read interface where all important information appears in one place. Instead of searching through multiple pages, admin staff can instantly see total queries, open cases, closed cases, client details, and recent activity. Tables, counters, and status indicators help them understand which issues need urgent attention. The dashboard acts like a command center that updates in real time, allowing the admin team to track workflow smoothly. With this centralized view, admin users can manage queries faster, prioritize tasks better, and improve their overall response quality while ensuring no client request gets missed.

3. Technologies Used

- Stream lit for front-end interface

In virtual environment of vs code with the help of interpreter and using pip install we connect the stream lit, hashlib, and my SQL connector. Stream lit offers a fast, Python-centric way to build the CQMS front end, enabling interactive forms, tables, and charts with minimal boilerplate. Developers write simple Python functions to define UI elements; stream lit handles the reactive rendering and state updates. This approach accelerates prototyping and provides a clean, consistent interface for clients and admin. Stream lit integrates easily with Python libraries for plotting and data manipulation, so dashboards and analytics are straightforward to add. For small teams or campus projects, Streamlet's low friction makes it ideal, delivering a responsive web application without the overhead of traditional front-end frameworks.

- Python for backend logic

Python serves as the backbone of the backend logic in the Client Query Management System, tying together the database, authentication, form processing, and dashboard operations. Because Python is both expressive and easy to maintain, it allows developers to build complex features with clean, readable code. In this system, Python handles user registration, securely hashes passwords, validates login details, and establishes stable connections to the MySQL database using connector libraries. Every query submitted by a client is processed through Python functions that sanitize inputs, execute parameterized SQL statements, and commit the data safely. For admin users, Python retrieves and filters records, updates query statuses, and logs timestamps. Stream lit relies on Python's backend operations to display tables, forms, and interactive components dynamically. Error handling, session management, and permission checks are all implemented using Python's built-in modules and custom functions. Because Python integrates seamlessly with data libraries like pandas and plotting tools, it also powers dashboards, analytics, and performance summaries. Its simplicity makes debugging easier, while its flexibility allows the system to scale with new features such as email alerts, file uploads, or advanced reporting. Overall, Python provides the reliability, structure, and adaptability needed for a polished and capable support system.

- MySQL for database storage

MySQL plays a central role in the Client Query Management System by acting as the secure, structured, and reliable storage layer for all client and admin data. It organizes records into well-defined tables such as client users, admin users, and client queries, ensuring that every piece of information is stored in a consistent format. MySQL's relational structure helps maintain strong data integrity, linking queries to the users who created or resolved them. Through features like primary keys, foreign keys, indexes, and constraints, the database ensures fast retrieval and prevents invalid or duplicate entries. All operations, including storing new client queries, validating login credentials, updating query statuses, and recording timestamps, are handled using SQL commands executed from Python. MySQL also supports parameterized queries, which reduces the risk of SQL injection and enhances the system's overall security. With its ability to handle large data volumes efficiently, MySQL keeps the application responsive even as the number of users or queries grows. Backup options and transaction support add extra layers of safety, protecting the system from data loss or inconsistent states. Overall, MySQL provides a dependable and scalable foundation that ensures the Client Query Management System runs smoothly, securely, and efficiently.

- MySQL. Connector for DB connection

MySQL. Connector library acts as the essential bridge between the Python backend and the MySQL database. It allows the application to open secure, reliable connections to the database, execute SQL commands, and retrieve query results. With this connector, Python can insert new client queries, validate login credentials, update statuses, and fetch data for the admin dashboard. It supports parameterized queries, which help prevent SQL injection and keep the system secure. The connector also provides proper error handling, connection closing, and transaction control, ensuring smooth communication between the stream lit interface and the MySQL server at all times.

4. Working Principle

```
+-----------------------+
|       CLIENT         |
| (Streamlit Web App UI) |
+----------+------------+
           |
           V
+----------+------------+
 |     BACKEND LOGIC     |
 | (app.py - Python Code) |
 | Handles Login/Register |
 | Query Submission, etc.  |
+----------+------------+
           |
           V
+----------+------------+
 |       DATABASE       |
 | (SQLite/MySQL)        |
 | Stores Client Queries, |
 | User Data, Status, etc. |
 +------------------------+
```

- Client logs in and submits a query

The system begins with the client login process, which acts as the entry gate for users who wish to submit issues, complaints, or inquiries. When a client enters their credentials, the stream lit interface passes the data to the backend, where Python verifies the information using a secure MySQL lookup. If the credentials match, the client is successfully logged in and redirected to the query submission panel. This panel contains structured input fields such as name, email, mobile number, query heading, and description. These fields ensure that every query carries enough detail for the support team to understand the issue clearly. Once the user fills out the form and submits it, Python collects the data and prepares it for storage. The application uses parameterized SQL queries to prevent unauthorized database access or SQL injection attacks. If all values are valid, the backend inserts the query into the client queries table in MySQL. A timestamp is also added automatically to record the exact moment the query was submitted. Throughout this journey, the client receives live feedback through stream lit notifications, confirming successful submissions or showing helpful warnings in case of mistakes. This entire process creates a smooth, professional, and secure way for clients to communicate their problems with the support team.

- Data is stored in the MySQL database.

After the client submits their query through the Stream lit interface, the system immediately prepares the information for safe and structured storage. MySQL acts as the central hub where all client data is maintained. Using MySQL. Connector, Python establishes a secure and stable connection to the database. Before inserting anything, the system checks whether the necessary table exists; if not, it automatically creates one. This dynamic approach ensures that the application remains functional even during fresh installations or migrations.

Once the connection is active, the data is inserted into the client queries table using parameterized SQL statements. This method protects the system from SQL injection and ensures only intended values enter the database. Each query is stored along with fields such as client name, email, mobile, heading, description, status, and timestamps. The status initially remains at “Open,” signalling that the support team needs to review it. MySQL’s strong indexing capabilities allow fast filtering and sorting of queries, even as the number of records grows. Additionally, timestamps stored automatically by MySQL help in long-term reporting, tracking response times, and improving customer support workflow. This organized storage strategy ensures reliability, accuracy, and long-term accessibility of all client data.

- Support user logs in to view client queries.

Support users play a vital role in handling client issues, and their workflow begins with a secure login process. The system verifies their credentials against the users table in MySQL to ensure that only authorized personnel gain access. Once logged in, the interface changes to display a dashboard-like environment tailored specifically for support tasks. Using Python’s connection to MySQL, the system fetches all available queries, typically ordered by creation time so the newest or most urgent issues appear at the top. The dashboard shows a detailed list of all queries, including client information, the issue description, and the current status. Support users can scroll through this list and quickly identify which queries are pending attention. This view helps support teams work efficiently, track unresolved issues, and maintain transparency in communication. The combination of Streamlet’s visual layout and MySQL’s structured data ensures every query is displayed accurately and updated instantly. By logging in through this system, support staff can manage a large number of client queries without confusion or delays. This structured approach creates an organized environment where each user has a clear role, helping maintain system security and improving response times.

- Support can close a query, updating its status and close time.

One of the most important features of the Client Query Management System is the ability for support users to close a query once it has been addressed. When a support user selects a query from the dashboard, they are shown a “Close Query” button if the query is still marked as “Open.” This button triggers a backend function that updates the query’s status and records the exact closing timestamp. When the support user clicks the button, Python connects to the MySQL database through the MySQL. Connector library and executes an UPDATE query. This query changes the status from “Open” to “Closed” and inserts the current date and time into the closed at field. This process ensures that each query has a complete lifecycle record, showing when it was created and when it was resolved. The timestamp helps measure response time, support efficiency, and overall system performance.

After updating the record, the interface immediately refreshes using St. Rerun () so the change becomes visible without restarting the application. The updated query is then clearly marked as Closed in the support dashboard, preventing duplicate handling or confusion. This smooth workflow helps support staff handle issues responsibly while maintaining accurate and traceable records that can be audited at any time.

- All data remains accessible for review at any time.

A major strength of this system is that all stored data remains accessible for future review, audits, or reports. Since every query and user action is recorded in the MySQL database, nothing gets lost, even over long periods. This allows administrators, support teams, or managers to analyse past records, review client histories, and improve the support workflow. Whether it is an open query or a closed one, the system can fetch it anytime through simple SQL queries. Stream lit enhances this process by displaying the information in a structured and readable format directly on the dashboard. Support users can review older queries to understand common issues or track client satisfaction levels. The stored timestamps also allow detailed reporting, such as calculating average response time or identifying delays. Since MySQL stores data in a reliable and consistent manner, the system can scale over time, keeping thousands of records without slowing down.

Having long-term access to these queries is extremely useful for performance evaluation, team training, and improving customer service strategies. It also supports transparency, as every action performed on a query is recorded. This makes the entire system dependable, future-proof, and suitable for real-world business use where historical data is crucial for decision-making.

5. Features Implemented

- User Registration and Login

The user registration and login module forms the foundation of the Client Query Management System, ensuring that every client is securely authenticated before accessing the platform. During registration, users provide essential information such as name, email, password, and user role (Client or Support). The password is not stored in plain text; instead, it is hashed using the SHA-256 algorithm to enhance security and prevent unauthorized access even if the database is compromised.

The registration process inserts this data into a user’s table in MySQL, ensuring uniqueness through email validation. If the email already exists, the system displays an error message, maintaining data consistency and avoiding duplicates.

For login, users input their email and password, which are validated against the stored credentials. Once authentication succeeds, the user’s session is stored in st.session_state, enabling personalized access without requiring repeated logins during the session. Clients are redirected to the query submission page, while admin users are directed to the dashboard.

This feature not only establishes secure entry points but also enforces role-based access control**,** ensuring that clients and admin staff have separate privileges. The robust authentication design maintains data privacy and integrity while providing a smooth and user-friendly login experience. This approach follows best security practices in web applications using Stream lit**,** Python**,** and MySQL, forming a secure base for all subsequent functionalities within the system.

- Submit New Queries

The Submit Query feature allows clients to communicate their concerns, issues, or feedback directly to the admin team through a simple and efficient form interface. Once a user logs in as a client, a dynamic form is displayed, prompting them to enter essential details such as their mobile number, query heading, and a detailed description of the issue.

The submission process is designed using Stream lit forms, ensuring that data is validated before being stored. If required fields are left empty, a warning is displayed, prompting the user to complete the form. Upon successful submission, the entered data is inserted into the client queries table within the MySQL database, linking it with the user’s unique ID. The system automatically records the creation timestamp and sets the default status to "Open".

This module ensures that all client queries are properly logged and retrievable by the admin team. The combination of frontend form validation and backend database handling guarantees accuracy and prevents data corruption. The intuitive design and minimal input requirements make the process fast and user-friendly. By implementing this feature, the system promotes transparency and efficiency in client–admin = communication, creating a reliable digital platform for issue tracking and resolution.

- Fields: name, email, mobile, heading, description

Name: Identifies the client submitting the query, used for personalization and record tracking.

Email: Serves as a unique identifier for login and communication between the client and admin team.

Mobile: Provides an alternative contact method for quick updates or verification purposes.

Heading: A short title summarizing the main issue or subject of the client’s query.

Description: A detailed explanation of the problem or request, helping admin staff understand and resolve it effectively.

- Queries Stored in MySQL

In this project, all the client queries are stored securely in a MySQL database, ensuring organized data management and easy retrieval. When a client submits a new query using the Stream lit web interface, the data — including the client's name, email, mobile number, heading, and description — is sent to the backend logic written in Python. From there, the data is inserted into a MySQL table named client queries. This storage process ensures that each query is permanently recorded with unique identification and time-stamped details. MySQL's structured format makes it easy to run SQL queries to retrieve, update, or analyse the data whenever needed. By storing data in a database rather than temporary memory, all queries remain available for review even after the application restarts. This reliable storage system provides data integrity, supports concurrent users, and forms the backbone for report generation and analytics in future improvements.

- Status Tracked: Open / Closed

Each query in the system is assigned a status — either Open *or* Closed — to indicate its current state in the support process. When a client submits a query, it is automatically marked as "Open." This shows that the issue still needs attention or is pending resolution. Once a support team member reviews and resolves the query, they can update its status to "Closed." The system also records the exact closed at timestamp, ensuring transparency and accountability in query handling. Tracking statuses helps categorize and manage multiple queries efficiently, making it easier for support staff to prioritize tasks. Clients can later check the status of their queries to confirm whether their issue has been resolved. This open/closed mechanism introduces a simple yet effective workflow that keeps the system organized and the communication between clients and support staff clear. It also provides valuable insights for measuring response times, identifying recurring issues, and ensuring overall customer satisfaction.

6. Implementation Steps

- Created Streamlit UI for Clients and Admin

The Streamlit framework was used to create an interactive and user-friendly interface for both clients and admin users. Clients can easily register, log in, and submit queries using simple form inputs like name, email, mobile, heading, and description. Admin users have a dedicated dashboard to view, manage, and close client queries. Streamlit's built-in components like st.form, st.dataframe, and st.button made it easy to design an intuitive UI without writing complex frontend code. The layout was organized with sidebars for navigation and forms for input, ensuring a clean and modern experience. The dynamic page rendering allows the UI to update in real time whenever new data is added or a query is closed, making the interface both responsive and visually appealing.

- Connected with MySQL using Python

The backend connection between Streamlit and MySQL was established using the MySQL. Connector library in Python. This allowed seamless communication between the web interface and the database. The connection parameters such as host, user, password, and database name were securely configured in the connection function. Through this integration, the app could insert, fetch, and update data stored in the MySQL database. MySQL was chosen for its reliability, scalability, and structured query language support. Using Python as the middleware ensured that database operations like INSERT, SELECT, and UPDATE could be executed dynamically based on user interactions. This connection provided the foundation for storing client details, managing queries, and tracking their statuses in real time.

- Designed Separate Login & Registration Modules

To ensure proper user management and security, separate login and registration modules were implemented for both clients and admin users. During registration, user details like name, email, password, and role (Client or Admin) are collected and stored in the users table after password hashing for security. For login, the system verifies the user's credentials against the database and checks their assigned role to redirect them to the correct interface. This separation ensures that clients can only submit queries, while admin users can only manage and respond to them. The use of hashed passwords prevents unauthorized access and protects sensitive information. These modules ensure structured access control, maintaining clear boundaries between clients and the admin team.

- Implemented Data Insertion, Query Fetching, and Updates

The application allows real-time interaction with the MySQL database for inserting, retrieving, and updating records. When a client submits a query, it is inserted into the client queries table using SQL INSERT commands. Admin users can fetch all client queries through SELECT queries and view them in a Stream lit table. Additionally, when an admin member resolves an issue, the query status is updated from "Open" to "Closed" using an UPDATE query, and a timestamp is recorded. This flow ensures data consistency and makes it easy to track query history. Each operation is handled securely using Python's parameterized SQL queries to prevent SQL injection and maintain database integrity.

- Added Session Management to Keep Users Logged In

Session management in Stream lit was implemented using st.session_state to maintain user information after login. Once a user successfully logs in, their credentials and role are stored in session variables. This allows the system to remember who is logged in and display the appropriate dashboard without requiring the user to re-enter credentials after every interaction. Session management ensures a smooth and continuous user experience across multiple app actions. It also prevents unauthorized access by clearing session data when the user logs out or refreshes the app. This technique is essential in web applications where persistent state management improves both usability and security.

- Displayed Client Queries in a Stream lit Table

For the admin team's dashboard, client queries are displayed in a dynamic table using Streamlit's st.dataframe () feature. The data fetched from the database includes important columns such as client name, email, mobile number, query heading, description, status, and timestamps. The table provides a clear overview of all submitted queries, allowing admin users to easily analyse and manage them. Each record can be expanded to view details, and open queries include a "Close Query" button for real-time updates. Once a query is closed, the table automatically refreshes to show the updated status. This visual representation enhances usability, helping the admin team efficiently track and resolve issues with transparency and speed.

7. Future Scope

In the future, integrating an email notification system can greatly improve communication between clients and the admin team. Whenever a client submits a query, an automated email confirmation can be sent to acknowledge receipt. Similarly, when the status of a query changes — for example, from “Open” to “Closed” — the client can be notified instantly. This can be implemented using Python’s smtplib or third-party APIs like Gmail API or Send Grid. Email notifications help maintain transparency and keep clients informed without requiring them to log in repeatedly. Admin users can also receive alerts for newly assigned or high-priority queries. Adding this feature enhances user engagement, builds trust, and improves response time. In professional systems, automated notifications are a standard feature that adds to the overall reliability and efficiency of query management platforms. Currently, the system allows a single admin user to manage all queries. In the future, a multi-admin user assignment system can be introduced to improve task distribution and efficiency. Each incoming client query can be automatically or manually assigned to a specific admin staff based on factors like workload, expertise, or priority level. This will require an additional database table for tracking assigned queries and admin user IDs. The feature can include role-based access, ensuring that only assigned users can modify or close specific queries. Implementing this system would enhance scalability and make the platform suitable for organizations with larger teams. It will also improve accountability, as each query will have a clear handler, ensuring faster and more organized resolutions.

A future enhancement could include building an analytics dashboard to visually represent the system’s data using charts and graphs. With the help of libraries like matplotlib, seaborn, or plot, the admin or admin staff can view insights such as the total number of queries, average response time, number of open vs closed cases, and performance per support agent. This visualization can be directly integrated into Streamlit using its chart components. The dashboard can also display monthly or weekly trends to identify recurring issues or customer pain points. Such analytical data will help the organization make data-driven decisions to improve customer service. Moreover, it adds a professional layer to the system, turning raw data into meaningful information. An analytics dashboard not only improves transparency but also demonstrates the effectiveness of the query management workflow. To enhance client admin, adding a file upload feature would allow users to attach screenshots, documents, or other supporting materials while submitting queries. Streamlit’s st.file_uploader () component can be used to handle file uploads. The uploaded files can then be stored in a server directory or cloud storage service such as AWS S3, Firebase, or Google Drive, with references saved in the database. This feature helps admin team better understand client issues by reviewing attached files, especially for technical or visual problems. It also adds professionalism and versatility to the system. Implementing secure file handling practices, such as limiting file size and validating file types, ensures data safety. This enhancement would significantly improve the accuracy and efficiency of issue resolution for both clients and the admin team.

Currently, the project may be running locally, but deploying it online using **Streamlit Cloud** will make it accessible globally through a shareable web link. Streamlit Cloud simplifies deployment by connecting directly to a GitHub repository and automatically running the app.py file. Once deployed, clients and admin staff can access the system anytime without installing dependencies. The database can be hosted on a cloud platform like MySQL on Google Cloud, AWS RDS, or a shared server for persistence. Online deployment ensures that updates pushed to GitHub are automatically reflected in the live version. This step transforms your project from a prototype into a fully functional, real-world application, making it usable for organizations, institutions, or start-ups. It enhances professionalism and demonstrates your ability to build, host, and maintain web-based software systems.

8. Conclusion

The Client Query Management System successfully fulfils all the functional and technical requirements that were established at the beginning of the project. The main goal of this system was to provide a digital solution that helps organizations or businesses efficiently manage client queries in a streamlined, organized, and responsive manner. Throughout the development process, the project demonstrated a complete understanding and implementation of full-stack development, integrating the frontend, backend, and database components into one cohesive system. The frontend of the project, built using Streamlit, offers an intuitive and interactive web interface that allows users to easily register, log in, and submit queries. This modern web-based interface ensures that clients can communicate their issues or questions in a user-friendly way without requiring any technical expertise.

On the backend, the project leverages the Python programming language to handle core functionalities such as user authentication, data validation, session management, and database communication. Python’s simplicity and readability made it easier to maintain a clean and structured codebase while also allowing for scalability and future improvements. The backend interacts with a database—implemented using either MySQL or SQLite, depending on deployment requirements—to store and retrieve client queries, user information, and query statuses. This database ensures that all information is securely saved, efficiently queried, and readily available when needed. Moreover, the project includes mechanisms to track each query’s status—whether it is ‘Open’ or ‘Closed’—which enhances transparency and provides real-time updates to both clients and admin staff.

In addition to demonstrating technical skills, this project embodies practical problem-solving ability and professional development practices. It showcases the ability to design a system that can be extended or integrated into larger enterprise solutions. The system architecture was carefully designed to maintain clear separation between the user interface, the processing logic, and the data storage, ensuring reliability and ease of maintenance. The integration of Streamlit makes the web application highly interactive and easy to deploy, even for non-technical users. Meanwhile, the SQL database connection illustrates an understanding of structured data management, including table creation, insertion, updating, and retrieval operations.

From a learning perspective, this project has proven to be an invaluable experience in understanding the real-world application of Python, Streamlit, and SQL. It provided an opportunity to explore the full cycle of software development—from designing and coding to testing and deployment. Furthermore, it reflects the growing importance of developing data-driven systems in today’s digital business environment. The Client Query Management System stands as a robust, scalable, and practical solution for managing client interactions, ultimately enhancing communication and efficiency for businesses. In conclusion, this project not only demonstrates technical competency but also highlights the capability to translate theoretical knowledge into a working, real-world application that adds value to both organizations and their clients.