CST1510 – Multi-Domain Intelligence Platform

Tier 1 Implementation – IT Operations

Student: Austin Arinaitwe
Student ID: ***M01041944***
Module: CST1510 Programming for Data Communication and Networks

---

1. Introduction and Project Scope

This report describes my Tier 1 implementation of the Multi-Domain Intelligence Platform for CST1510. The aim of this coursework is to design a functional web-based system that supports three types of users: Cybersecurity Analysts, Data Scientists, and IT Operations staff. For Tier 1, students only need to fully implement one domain. I chose **IT Operations** because the data structures and workflows were more familiar and the insights were straightforward to interpret.

The purpose of the platform is to help IT teams analyse service desk performance and identify areas where ticket handling is causing delays. The course required several components regardless of domain: authentication, a SQLite database with CRUD functionality, data visualisations, an AI assistant, and some level of object-oriented design. My submission focuses on delivering these features clearly and in a way that demonstrates understanding of the development process rather than complexity.

By the end of the implementation, the platform allows users to log in, view IT support tickets, add or update records, analyse charts, and request recommendations from a simple AI assistant. The system provides a small but complete workflow that shows how IT departments can monitor performance and spot issues such as slow resolution times or overloaded staff members.

---

2. System Architecture and Implementation

The system was built using Python and Streamlit as the main web framework, supported by a SQLite database for persistent storage. The code is divided into separate modules to keep the architecture clean and understandable.

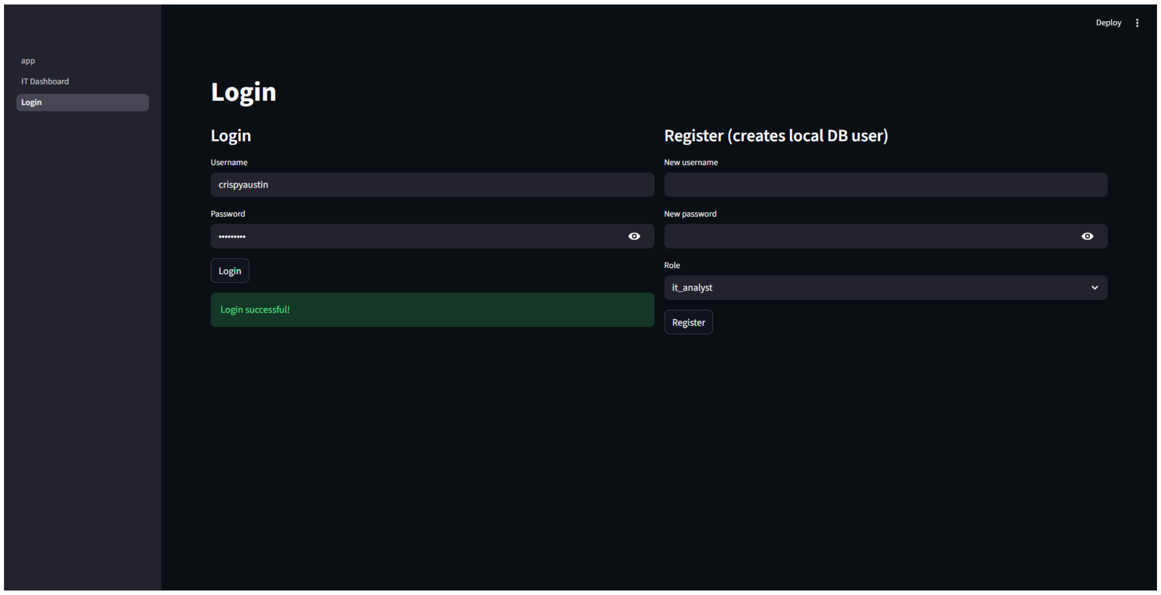
2.1 Authentication and Security

For authentication, I used **bcrypt** to hash passwords before storing them. This is important because plain-text passwords are insecure, and hashing ensures that even if someone accesses the database, they cannot easily retrieve user passwords.

The login workflow runs as follows:

1. The user enters a username and password on the login page.
2. The system retrieves the stored hash for that username.
3. bcrypt compares the typed password with the stored hash.
4. If correct, the user is saved inside **`st.session_state.`**

This ensures that dashboard pages can check whether the user is authenticated before displaying sensitive content.



Login

Login

Username: crispyaustin

Password: •••••••••

Login successful!

Register (creates local DB user)

New username

New password

Role: it_analyst

2.2 Database Layer (SQLite & CRUD)

The database is managed through a small `DatabaseManager` class which handles SQL queries. SQLite was chosen because it is lightweight and does not require a server.

Three domain tables were created to satisfy the coursework brief:

- `cyber_incidents`
- `datasets_metadata`
- `it_tickets`

However, only `it_tickets` is fully implemented for Tier 1.

The IT table stores:

- ticket ID
- requester
- assigned staff

- status
- opened/resolved dates
- priority

I added CRUD functions through the `ITTicket` class. These allow the user to:

- create new tickets
- update existing ones
- delete incorrect entries
- view all records

CRUD is demonstrated directly through Streamlit forms and tables in the dashboard.

2.3 Streamlit Web Application (MVC-style structure)

The application follows a simplified **Model–View–Controller** structure:

- **Models** → located in `models.py` (`User` and `ITTicket` classes)
- **View** → Streamlit UI pages inside `/pages`
- **Controller** → scripts like `auth.py` and `database.py` which process logic

A simple Data Flow Diagram (DFD) for the system:

1. User interacts with Streamlit pages (Login or Dashboard)
2. User input is processed by functions (authentication, CRUD)
3. The database returns results
4. Streamlit renders tables or charts based on the data

This separation makes the system easier to expand or debug.

2.4 OOP Design

To meet the Week 11 requirements, two small classes were implemented:

User class

Handles retrieving user information from the database.

ITTicket class

Represents a single ticket and includes static methods for:

- creating a ticket
- updating a ticket
- deleting a ticket
- loading all tickets

Although simple, this structure improves readability and avoids repeating SQL queries everywhere.

2.5 Visualisations

The IT dashboard displays two main charts:

1. Tickets Per Staff Member

Shows how many tickets each staff member is handling. This makes it easy to identify if one staff member is overloaded.

2. Tickets by Status

Displays how many tickets are Open, In Progress, Waiting for User, or Resolved. This helps highlight slow workflow areas and bottlenecks.

Both graphs were created using **Plotly**, which integrates smoothly with Streamlit.

Screenshots of these charts are included in Section 3.

2.6 AI Assistant Integration

The final requirement was to include an AI-powered assistant.
I implemented a small helper function in **`ai_helper.py`** which sends a summary of aggregated ticket data to OpenAI's API.

If the user has an API key, the assistant provides:

- a quick interpretation of staff workload
- delays caused by certain ticket statuses
- recommended steps to improve performance

If no API key is present, the system still displays a clear message — meaning the feature is implemented but disabled.

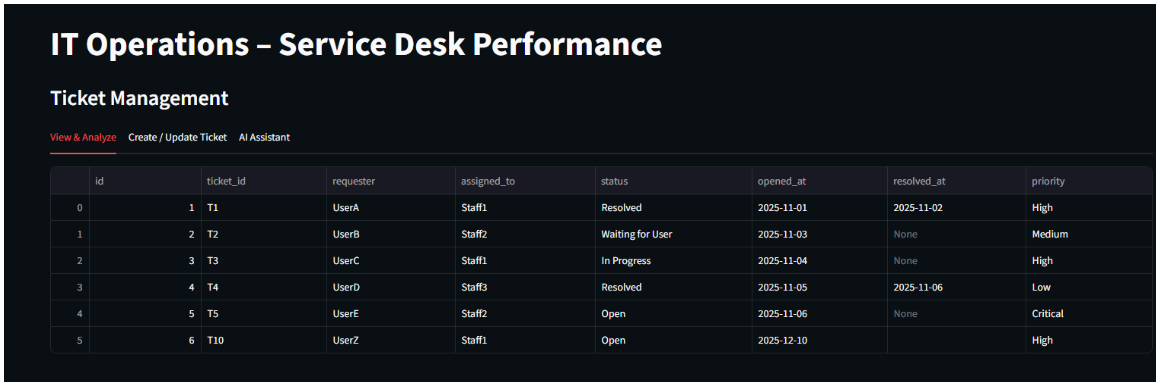
---

3. High-Value Analysis and Insights

The goal for the IT domain is to examine service desk performance and identify delays. The sample dataset included several tickets assigned to different staff members with varying statuses.

Below is the analysis based on the dashboard output.

3.1 Ticket Table Summary



IT Operations – Service Desk Performance

Ticket Management

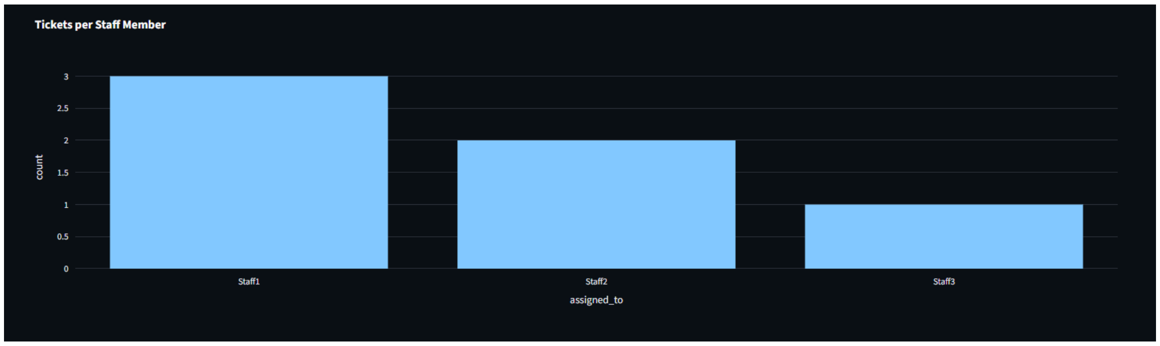
View & Analyze | Create / Update Ticket | AI Assistant

	id	ticket_id	requester	assigned_to	status	opened_at	resolved_at	priority
0	1	T1	UserA	Staff1	Resolved	2025-11-01	2025-11-02	High
1	2	T2	UserB	Staff2	Waiting for User	2025-11-03	None	Medium
2	3	T3	UserC	Staff1	In Progress	2025-11-04	None	High
3	4	T4	UserD	Staff3	Resolved	2025-11-05	2025-11-06	Low
4	5	T5	UserE	Staff2	Open	2025-11-06	None	Critical
5	6	T10	UserZ	Staff1	Open	2025-12-10		High

The table shows a mix of Open, In Progress, Waiting for User, and Resolved tickets. From this overview it is already clear that:

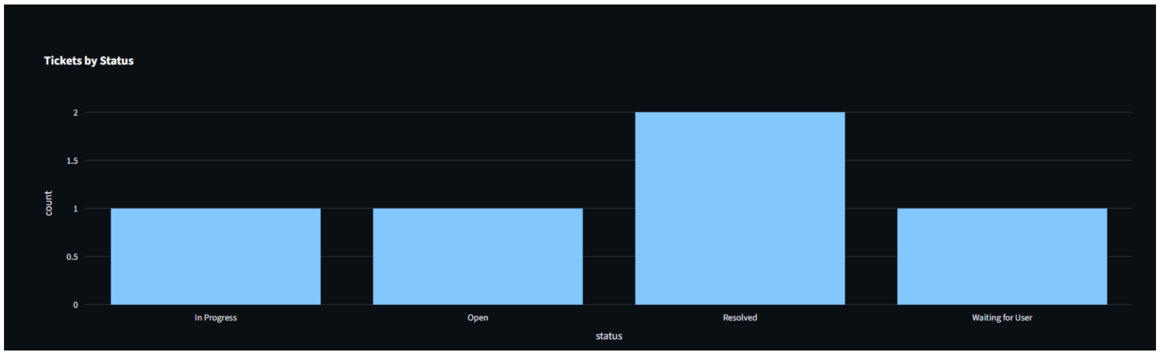
- Staff1 handles a relatively large portion of tickets
- “Waiting for User” and “Open” statuses are common
- Several tickets remain unresolved

3.2 Tickets Per Staff Chart



This bar chart makes it clear which staff members have the highest workload. In my sample data, **Staff1** appears to handle the most tickets. This could indicate strong performance or potential overload depending on turnaround times.

3.3 Tickets by Status Chart

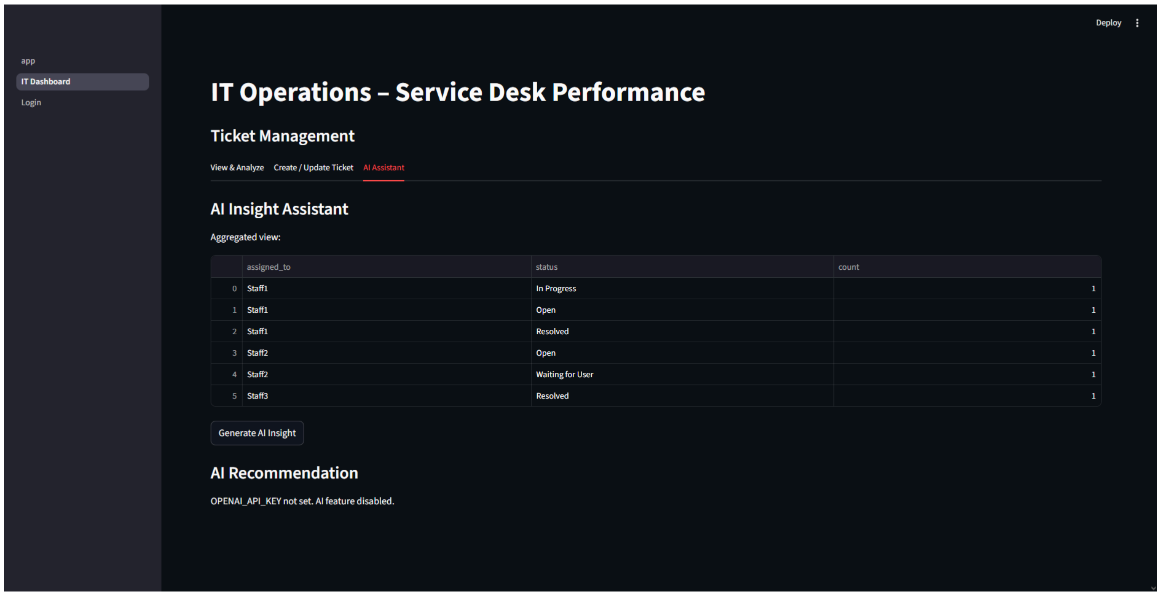


The status distribution highlights potential bottlenecks. For example:

- A high number of "Waiting for User" tickets suggests customers are slow to respond.
- More "Open" tickets may indicate delays in initial triage.

These insights are essential for managers who need to shift resources or adjust processes.

3.4 AI Assistant Interpretation



IT Operations – Service Desk Performance

Ticket Management

AI Insight Assistant

Aggregated view:

	assigned_to	status	count
0	Staff1	In Progress	1
1	Staff1	Open	1
2	Staff1	Resolved	1
3	Staff2	Open	1
4	Staff2	Waiting for User	1
5	Staff3	Resolved	1

AI Recommendation

OPENAI_API_KEY not set. AI feature disabled.

When using the AI assistant, the system analyses the aggregated data. The assistant typically points out:

- Which staff member is overloaded

- Whether unresolved tickets are piling up
- Suggestions like redistributing tickets or following up on “Waiting for User” cases

Although simple, the AI provides a helpful summary that supports decision-making.

---

4. Reflection and Conclusion

Working on this project helped me understand how different parts of a real-world web application come together. The most valuable areas for me were:

- Learning how to structure a Streamlit app using multiple pages
- Understanding how bcrypt hashing protects user accounts
- Using SQLite with CRUD operations
- Writing basic OOP models
- Displaying insights through data visualisation
- Integrating a simple AI feature

A few challenges I faced included getting Streamlit to refresh correctly (the old **`experimental_rerun()`** method no longer works) and configuring the environment properly. Fixing these issues helped me learn how to debug problems more effectively.

If I were to expand this project further, I would add search filters for tickets, more detailed metrics, role-based dashboards, and additional domains such as Cybersecurity or Data Science. For now, the Tier 1 IT Operations version fully meets the coursework requirements.