Jaden Patrick

Dr. Damon Gray

Management Information Systems

30th November 2025

Governance Memo

The Help Desk Information System developed for this project relies on synthetic ticket data, but it is designed using the same principles that real companies use when handling sensitive operational information. Even though no actual customer data is stored, the system follows strong privacy, security, and governance standards to ensure it could support real-world use without exposing the organization to unnecessary risks. This memo explains the policies, safeguards, and technical practices used to protect the data, manage system access, prevent failures, and prepare the platform for future scaling using PostgreSQL.

A major part of maintaining a trustworthy system is ensuring strong data privacy. Even though the dataset for this project is fully synthetic, the database is structured as if it were storing live information. To protect users, the system avoids collecting any personally identifiable information (PII) such as emails, phone numbers, addresses, or financial details. Instead, it only stores general ticket fields like priority, status, timestamps, and randomly generated names. If the system were ever expanded to handle real ticket submissions, it would follow strict privacy rules: limiting the data collected to what is absolutely necessary, masking sensitive values in reports, encrypting data both at rest and in transit, and rotating access credentials regularly. These measures ensure that the help desk system minimizes privacy risks and maintains user trust. Another key part of governance is controlling who can access different parts of the system. This is where role-based access control (RBAC) becomes essential. In a real help desk environment, each type of user should only have access to the information required for their job. Regular customers should only be able to submit tickets and view their own cases. Support agents should have permission to update the tickets assigned to them but should not be allowed to delete records or modify system settings. Supervisors and team leads need a broader view so they can review performance, reassign tickets, and track SLA compliance across the whole team. System administrators require full access because they maintain the database, manage backups, and handle technical issues. This layered RBAC approach prevents unauthorized actions, reduces mistakes, and ensures accountability throughout the system.

Protecting data also requires a reliable and consistent backup strategy. Even small systems can lose valuable information if the database becomes corrupted or a device fails. For a lightweight SQLite database like the one used in this project, a practical backup plan includes making an automatic daily copy of the tickets.db file, creating a full weekly snapshot stored

separately, and keeping backups for at least 30 to 60 days. Each backup should be tested periodically to make sure the file can be opened and restored without errors. If data loss were ever to occur, the restore process is simple: stop the application, replace the broken database with the latest backup, restart the app, and verify that the data loads correctly. Although this project runs locally, real businesses would store backups in secure cloud locations and encrypt them to prevent unauthorized access. As the system grows and more features are added, change control becomes important to keep everything stable. Any updates to the database schema or application code should first be documented and tested in a safe environment rather than directly in production. Once the changes are confirmed to work correctly, they can be deployed during low-traffic hours to reduce downtime. Using version control tools like Git helps track changes, maintain clean history, and roll back to previous versions if something goes wrong. This structured approach to change control helps prevent crashes, mismatched tables, or broken dashboards.

In addition to backups and change control, proper logging and auditing keep the system secure and easier to troubleshoot. The help desk application should log when tickets are created, updated, closed, or reopened. It should also record which agent performed each action, as well as any failed login attempts or system errors. Storing these logs separately from the main data prevents tampering and provides a reliable record for resolving disputes or identifying misuse. As the organization grows, the help desk system may need to scale beyond SQLite. A clear migration path to PostgreSQL ensures the platform can handle more traffic, more agents, and more complex reporting in the future. The migration process is straightforward: first export the SQLite tables as CSV files, then create equivalent PostgreSQL tables using the project's schema. After that, import the CSV data into Postgres using bulk-loading commands, update the Streamlit app to connect using a Postgres driver like psycopg2 or SQLAlchemy, and test all dashboards and filters to make sure they work with the new engine. PostgreSQL offers stronger indexing, higher concurrency, and better security, making it a natural upgrade when the help desk system needs to support enterprise-level usage.

Overall, this governance framework ensures the Help Desk Information System is private, secure, reliable, and scalable. By applying strong privacy rules, a clear access model, a dependable backup routine, structured change control, proper logging, and a future-ready migration plan, the system aligns with real industry standards and can support both current needs and future growth.