Jeffrey Gomez

CIT262

4/29/2025

**Abstract**

FixMate addresses the familiar challenge homeowners and renters face in organizing recurring and one-off maintenance tasks. With its user-friendly design, centralizing task creation, scheduling, cost tracking, and repair-guide references in a single web interface prevents overlooked upkeep, reduces user anxiety, and can save time and expenses associated with neglected home repairs. Key features include an intuitive dashboard, a context-sensitive YouTube search tool for “How to …” tutorials, and an optional Streamlit analytics dashboard for visualizing task progress and trends.

**Technologies Used**

The backend is implemented in Python 3.x with the Flask microframework (v2.x–3.x), leveraging Werkzeug’s generate\_password\_hash and check\_password\_hash for secure credential handling. Data persistence uses SQLite via Python’s built-in SQLite3 module. Dynamic HTML generation relies on the Jinja2 templating engine, while Bootstrap 5.3 (via CDN) and custom CSS ensure responsive, modern styling. User sessions utilize Flask’s secure client-side mechanism. Version control is managed through Git and GitHub. An optional analytics module employs Streamlit (v1.x) and Pandas to produce interactive charts from the same SQLite database. The entire application runs locally on Flask’s development server.

**Backend & Frontend Design Overview**

FixMate follows a three-tier architecture. The Presentation Layer comprises six Flask-driven pages—Login, Register, Dashboard, Add/Edit Task Form, Task Detail, and Analytics—each extending a shared base.html template that contains the navigation bar, flash-message area, and footer. A YouTube search button dynamically appears when a user enters a task title, opening a new tab with results for “How to ‹title›.” Completed tasks are visually distinguished and moved to a separate section.

The Application Layer (Flask) handles URL routing via @app.route decorators, enforces authentication with a custom @login\_required decorator, and orchestrates CRUD operations on the tasks table. Each task record includes fields for id, user\_id (foreign key with cascade delete), title, category, due\_date, frequency, cost, estimated\_time, guide, video\_url, completed (0/1), and a timestamp. A context processor injects the current year into all templates.

The Data Layer consists of a single SQLite database (fixmate.db) with two primary tables:

* users (id PK, username TEXT UNIQUE, password\_hash TEXT)
* tasks (id PK, user\_id FK, plus task-specific columns as above)

Initialization is automated via a Flask CLI command (flask init-db), and seed data can be loaded with flask seed-db. The optional Streamlit dashboard (analytics\_dashboard.py) connects to the same database, filters by user\_id passed as a URL query parameter and displays user-specific metrics (total vs. completed tasks, category breakdowns, cost summaries).

**Challenges & Solutions**

Early on, missing route definitions triggered BuildError exceptions when linking to task detail pages. Defining @app.route('/task/<int:id>') and matching url\_for('task\_detail', id=…) resolved this. Implementing secure authentication required migrating from plain-text passwords to hashed storage: during registration, generate\_password\_hash populates the password\_hash column; during login, check\_password\_hash verifies credentials. Inconsistent database schemas (“no such column: password\_hash”) were handled by deleting the outdated .db file and rerunning flask init-db to enforce the correct structure. UI refinements—such as reducing the gap beneath the navbar and consolidating incomplete/completed tasks into a toggled view—were accomplished via custom CSS and a ?view= query parameter in the Dashboard route. Ensuring the Streamlit analytics page respected user boundaries involved passing user\_id in the URL and filtering SQL queries accordingly.

**Conclusion & Future Work**

With its core functionality complete, FixMate can be enhanced by adding exciting features such as notification features (email or in-app reminders for due or overdue tasks) and expanding analytics to include time-series trend analysis of costs and completion rates. These enhancements will significantly enrich the user experience and the potential of the application. Introducing task prioritization, advanced filtering, and file attachments (receipts, photos, manuals) would further enrich the user experience. Exposing a RESTful API could enable third-party integrations or a dedicated mobile app. Finally, containerizing the application with Docker, deploying it via a production WSGI server (Gunicorn), and hosting it on a cloud platform would prepare FixMate for real-world, multi-user scenarios.