**Record Management System Report**

**Introduction**

The Record Management System is a collaborative project developed by a team of five beginners for a specialist travel agent. Built using Python and Tkinter, the application provides a graphical user interface (GUI) to manage Client, Airline, and Flight records through create, delete, update, and search (CRUD) operations. Data is stored in a JSON file, chosen for readability and ease of use, with version control managed via Git (Git SCM, 2025). The project includes unit tests to ensure reliability and meets accessibility standards under the Equality Act 2010 (UK Government, 2010). This report outlines the design, functionality, challenges, and compliance with requirements, reflecting our learning journey.

**Application Design**

The system’s structure is organised under a record\_management/ root folder, featuring src/ for source code, tests/ for unit tests, and docs/ for documentation. Within src/, main.py hosts the GUI, records.py handles CRUD logic, and storage.py manages file I/O, with record.json in src/record/ . The GUI is a 600x500 window with a dark gray background for high contrast, adhering to WCAG 2.2’s Level AA perceivable criterion (W3C, 2023). It includes a control panel with CRUD buttons, a form section with a dropdown for record types, and a Treeview display area. Accessibility is enhanced with keyboard shortcuts (e.g., Alt+C), tab navigation, and clear labels, meeting WCAG 2.2’s operable and understandable criteria (W3C, 2023).

**Functionality** The application supports three record types: Clients (e.g., Name, Address), Airlines (e.g., Company Name), and Flights (e.g., Date, Start City). The GUI’s "Create Record" button opens a pop-up form tailored to the type, with delete, update, and search functions fully implemented using records.py. Data persistence uses storage.py to save to src/record/record.json with JSON, employing context managers and UTF-8 encoding. Unit tests in tests/ validate records.py and storage.py, while a User Acceptance Test (UAT) confirmed usability for travel agents, ensuring all requirements are met.

**Challenges and Solutions**

As beginners, we encountered several challenges. Designing the Tkinter GUI to fit all Client fields (11) was tricky, resolved by a dynamic pop-up form. Git integration issues, like pathspec errors with files, were fixed by verifying paths (Git SCM, 2025). We initially used JSONL from the skeleton but switched to JSON for simplicity, updating storage.py with team input. Accessibility testing with NVDA confirmed keyboard navigation and high-contrast design meet WCAG 2.2 standards for motor- and visually-impaired users (W3C, 2023; NV Access, 2025). These solutions reflect our learning process.

**Conclusion**

This Record Management System meets the assignment’s requirements with a GUI for CRUD, JSON storage, unit tests, and Git version. The 500-word report details our design, from the modular structure to accessibility features, and addresses challenges like GUI layout and Git errors. The completed project, with integrated CRUD and tested accessibility, fulfills the requirements. Our collaborative effort, tracked in the meeting minutes, demonstrates our learning.

**References**

Git SCM. (2025) *Git Documentation*. Available at: <https://git-scm.com/doc> (Accessed: September 2025).

NV Access. (2025) *NVDA Documentation*. Available at: <https://www.nvaccess.org> (Accessed: September 2025).

UK Government. (2010) *Equality Act 2010*. Available at: <https://www.legislation.gov.uk/ukpga/2010/15/contents> (Accessed: September 2025).

W3C. (2023) *Web Content Accessibility Guidelines (WCAG) 2.2*. Available at: <https://www.w3.org/WAI/standards-guidelines/wcag/> (Accessed: September 2025).

**Word Count:** 465