Project Report

Section 1: System Features

My Note taking application is a web-based application designed to efficiently organise and retrieve text and image based notes. The system implements the following key features:

The application provides comprehensive note management capabilities, allowing users to create, view, edit, and delete notes through an intuitive web interface. Each note contains structured data including title, content, associated URL, image reference, and category, providing rich and contextual information. The dynamic categorisation system automatically organises notes, creating a flexible index that evolves with user content.

A sophisticated search functionality enables users to quickly locate relevant notes using a smart algorithm that prioritises exact matches and sorts results by relevance. The system displays the top three most relevant results to maintain a clean interface while indicating when additional results are available.

The application follows a robust MVC architecture pattern with and CSV-based note storage ensuring data integrity. The responsive web interface provides consistent access across devices, with JSP files delivering content rendering.

Section 2: Design and Programming Process

My design approach centred on creating a maintainable, extensible system through a careful application of object-oriented principles. I adopted the Model-View-Controller architecture to clearly separate responsibilities and enhance system modularity.

The Model class forms the core of the application, encapsulating all data operations including reading, writing and searching for notes. I deliberately designed this class for centralising CSV reading/writing operations alongside specific methods like getNotesByCategory() and getAllCategories(). The Note class represents the fundamental data entity, implementing proper encapsulation through private fields with appropriate accessor methods for all fields.

I employed the Singleton pattern via the ModelFactory class to ensure consistent access to a single model instance throughout the application lifecycle. This design choice prevents data inconsistencies that might arise from multiple model instantiations while providing a clean API for controller components.

The servlet classes function as controllers, processing HTTP requests and coordinating between the model and views. Each servlet has a focused responsibility aligned with specific user operations. For example, the SearchServlet implements search algorithms that prioritise exact matches while providing fallback behaviour when no matches are found.

My implementation demonstrates strong adherence to OO principles through proper encapsulation, clean abstractions, and the single responsibility principle. Data validation occurs at appropriate layers, and error handling provides graceful degradation when exceptions occur.

The CSV-based mechanism balances simplicity with functionality, allowing for easy deployment while maintaining data integrity through careful file operations. I organised the codebase to support future extensibility, ensuring new features can be added with minimal changes to existing components.

In retrospect, while the system successfully fulfils its requirements with good OO design, future improvements could include implementing a more robust database solution for increased scalability, adding more comprehensive input validation, and expanding the search functionality to include full-text search of note content. Nevertheless, the current implementation represents a well-structured solution that effectively balances complexity with maintainability.