COMP0004 Coursework – Eima Miyasaka

Java Notes Web Application Report

Section 1: Summary of Implemented Features

The Notes Application is a Java web application built using the Model-View-Controller (MVC) structure that allows the user to create, view and search notes. Users can create notes with titles, content, and assigned categories, enabling organisation and storage of text. The application presents an index of all notes, with each entry being clickable to view the full content. A key feature is the search functionality that matches queries against note titles, content and categories, presenting results on a separate page. The system implements persistent storage through JSON files, automatically saving and loading notes without requiring manual intervention from users. Notes will therefore continue to exist when the server is restarted. All notes can be assigned to multiple categories to facilitate organisation. The application satisfies the core requirements outlined in the assignment specification.

Section 2: Design and Programming Process

The design and implementation of the Notes web application followed a structured approach founded on object-oriented principles and the Model-View-Controller (MVC) structure. This analysis examines the design decisions, class structure, and adherence to OO design practices.

The application architecture revolves around three principal components: model classes representing data entities, servlet controllers handling HTTP requests, and JSP views for rendering the user interface. This separation ensures modular development and maintenance.

The core model consists of two primary classes. The 'Note' class implements the entity representation with appropriate encapsulation of attributes (title, content, categories) and behaviour. The 'NoteIndex' class implements the Singleton pattern to provide a centralised collection manager with persistence capabilities. This design adheres to the Single Responsibility Principle as each class maintains a well-defined, cohesive purpose. The 'Note' class focuses exclusively on representing note data and attributes, while 'NoteIndex' handles collection management and persistence operations.

The application incorporates several design patterns. The Singleton Pattern is employed in the 'NoteIndex' class through a private constructor and static getInstance() method ensuring a single, globally accessible instance. This approach prevents potential data inconsistencies that could arise from multiple collections managing the same notes concurrently. The MVC Pattern creates a strict separation between model (data), view (JSPs), and controller (servlets), resulting in a maintainable architecture where components can evolve independently. Controllers process user inputs, interact with models, and select appropriate views without mixing presentation concerns.

The implementation demonstrates appropriate abstraction levels. Data members in the ‘Note’ class are properly encapsulated as private fields with public accessor/mutator methods, preventing unauthorised direct access. The ‘NoteIndex’ class abstracts away the complexity of persistence operations, presenting a clean API for note management. ‘NoteServlet’ and ‘SearchServlet’ abstract HTTP request handling, translating between web protocols and application methods.

The class design exhibits high cohesion and low coupling. Each class performs a well-defined set of related functions. For example, the ‘NoteIndex’ class manages all aspects of collection operations (add, delete, find, search) without assuming unrelated responsibilities. Dependencies between components are minimised through appropriate abstractions. Servlets depend on the model but not vice versa, creating a unidirectional dependency flow that simplifies maintenance.

The application implements a straightforward JSON-based persistence strategy using Gson for serialisation. This approach offers several advantages: human-readable storage format, schema flexibility for future enhancements, no database configuration required, and simple implementation with minimal dependencies. However, this design trades scalability for simplicity, which I believe is an acceptable compromise given the application's scope.

Despite the overall sound design, several improvements could be made to the architecture. Interface-based design could further reduce coupling between components. A more robust error handling framework would improve resilience. Input validation could be centralised through a dedicated validation layer. The persistence layer could benefit from abstraction to support multiple storage backends.

In conclusion, the Notes application demonstrates solid adherence to OO design principles with appropriate class decomposition, encapsulation, and pattern application. While more requirements could be fulfilled in the future, this application demonstrates the core necessities of a notes storage website with minimal complexity in its architecture.