COMP0004 Coursework Report

Summary of Features:

This application satisfies requirements 1, 2, 3, 4 and 5.

Landing Page: Users begin on a homepage with navigation options to both the editor and the search page.

Editor: Users can create new notes, assigning them unique titles. The editor supports a simplified Markdown format:

- Multiple header levels (# Heading 1, ## Heading 2, etc.).
- Image embedding using ![Alt text](Web URL) on a new line.

Search & Filtering:

- The search page initially lists all saved notes.
- Users can filter results by searching for any string in the note content or the alt text of images.
- Sorting options include alphabetical order and modification date.

Viewing & Editing:

- Notes can be opened individually for editing.
- Users can modify content and save changes.
- Notes can also be deleted when no longer needed.

Design and implementation:

My design process focused on implementing a clean separation of concerns through the Model-View-Controller (MVC) pattern. I structured the application with distinct components that handle specific responsibilities, ensuring code maintainability and extensibility.

The core of my model consists of three primary classes, that is the Note Class, the FileIndex Class and the FileMiniIndex Class. The Note Class serves as the primary data model, responsible for reading text files and converting Markdown syntax into HTML for web display. This class encapsulates all note-specific functionality, including parsing different markdown elements like headers and image tags.

The FileIndex Class manages the collection of notes by loading files from the data directory. It contains the logic for searching across all notes and sorting the results based on user-selected criteria, that is either alphabetical or by modification date.

Finally the FileMiniIndex Class provides a simplified version of the FileIndex Class. It acts as a compact representation of notes for the note viewer page, offering a streamlined way to navigate between notes. I also have heavy use of the built-in File Class in Java.

The servlets are kept lightweight with minimal logic by delegating the business logic to these model classes. The servlets primarily handle HTTP requests, call appropriate methods on the Notes, FileIndex, FileMiniIndex and File Classes and forward to JSPs for rendering. This approach adheres to the MVC pattern by ensuring that each component has a single, well-defined responsibility.

For data persistence, I opted for a simple approach using plain text files (.txt) stored in a dedicated "data" folder. This solution avoids the complexity of database integration while still providing reliable storage. Each note is saved as an individual text file, making it easy to manage and maintain file integrity.

Overall, the application achieves a good balance between functionality and design quality, with a clear separation of concerns that makes the code maintainable and extensible for future enhancements. Although improvements can be made by further reducing the logic in the Servlets by passing the url parameters directly into the models, this approach began being more closely followed while implementing the search functionality. Furthermore, I would consider implementing interface-based programming to increase flexibility and potentially introduce a factory pattern for creating different types of notes.