COMP0004 Coursework-NotesApp

Section 1: Summary of Program Features

All 7 requirements completed. The `NotesApp` program includes following operations:

1. **Multiple types of notes**: This program supports three kinds of nots——`TextNote`(plain text note), `TodoListNote`(users can add tasks and mark them as done by ticking them off), `MultimediaNote`(Combination of images, hyperlinks and text)
2. **Subjects/Categories**: Users can create custom subjects for organizing their notes(e.g., an academic discipline, a university module, a meeting minutes, etc.) Selecting a subject from the sidebar will display all notes under that category.
3. **Search and Sorting**: Users can search for notes by `title` or `content`(To-Do List notes can be searched by `tasks`, while Multimedia Notes can be searched by `descriptions`.) and sort them by creation time, last modified time, or title in both ascending and descending order.
4. **Automatic Save**: When users create or edit a TextNote, any modifications, including changes to the title, subject, or content, are automatically saved and recorded in a local JSON file—no need to click a save button.
5. **The Recycling Bin System**: Users can move unused notes to the trash, hiding them from the note list (while keeping them stored locally). They can then choose to recover or permanently delete the notes from the trash.
6. **A clean frontend page**: A web page built with basic HTML, CSS, and JavaScript, offering an intuitive design, logical layout, and smooth user interaction.

Section 2: Design and Programming Process Evaluation

Design Process

Stage 1: Defined the Entities:

1.1 Defined `Note` as abstract class to encapsulate common attributes (e.g., `title`, `subject`, `modifiedTime`) and methods, then implemented `TextNote`, `TodoListNote`, and `MultimediaNote` as concrete implementations, each adding type-specific fields.

1.2 Defined enumeration `NoteType` to map note types to classes and suffixes, including utility methods (e.g., `fromNote`, `fromSuffix`) for type resolution. This reinforced polymorphism and type safety.

Stage 2: Defined the Model Layer and Utility Classes:

2.1 Implemented `ModelFactory` to provide a singleton `Model`, using a `HashMap<String, Note> (noteBook)` and a `List<String>` for subjects, then added basic methods like `createNote`, `updateNote`, `findNoteById`.

2.2 Defined `JsonUtil`, `TimeUtil`, `ImageUtil` classes for data persistence and completely isolated them from the usage scenarios, ensuring easier future use.

2.3 Developed auxiliary models, `FileModel` (for JSON persistence with `JsonUtil`), `TrashModel` (for `trash management`), and `QueryModel` (for sorting and searching) to further separate concerns and improve maintainability.

Stage 3: Implemented the Controller Layer:

3.1 Created an abstract `BaseServlet` with reflection-based URI mapping (e.g., `/crud/createNote` to `createNote`), abstracting request handling.

3.2 Based on different operations categories, implemented `NoteCrudServlet`(for creating and updating,

etc), `NoteTaskServlet`(for searching and sorting), `NoteLoadServlet`(for loading notes list and form pages), pre-defined the communication rules between the View layer and completed the part of interaction with the Model layer.

Stage 4: Developed the View Layer:

4.1 Created the basic form in `TestNote.jsp`, `todoListNote.jsp`, `multimediaNote.jsp`based on the data needed by the controller layer.

4.2 Coordinated the interaction between the Controller layer and the View layer, organized the webpages execution and routing logic.

4.3 Completed `NoteList.jsp`, `searchNote.jsp`, the part of the View layer that retrieves and displays data from the Controller layer.

4.4 Improved the page interaction logic using JavaScript and enhanced the content layout and appearance with CSS.

Evaluation of OOP DesignPractices

Abstraction

Robust abstraction through the `Note` and `BaseServlet` abstract classes, which provided a foundational framework for extensibility and enabled polymorphism and reduced code duplication. The enumeration `NoteType` further enhanced this by formalizing note types, while utility classes abstracted low-level operations such as JSON parsing and image saving. This layered abstraction allowed high-level logic in servlets to remain clean and focused, minimizing complexity in business logic implementation.

Encapsulation

Encapsulation is reinforced by `Note` subclasses hiding type-specific data and `Model` concealing persistence details behind a clean interface. The controller's process of encapsulating frontend data into a structured object (a `Note` instance) before passing it to the model shields the model from raw data complexity. Private fields like `noteBook` in `Model` and `invokeMethod` in `BaseServlet` safeguard internal state, promoting data integrity and providing controlled access through well-defined APIs.

Cohesion and Single Responsibility Principle (SRP)

Within `Model`, methods like `createNote` and `getAllNotes` work cohesively towards data management, ensuring internal elements are tightly related. Each class is designed for a single, well-defined purpose: `Model` centralizes data coordination, `JsonUtil` focuses on JSON parsing and writing, and `TimeUtil` defines timestamp formatting. The `Model` further supports SRP by delegating to auxiliary models like `FileModel` and `TrashModel`, preventing overburdening and ensuring a modular design.

Open-Closed Principle (OCP)

The application supports the Open-Closed Principle (OCP) by allowing new note types to extend `Note` and update `NoteType` without altering existing code. `BaseServlet` accommodates new endpoints via method addition, and utility classes are reusable.

Overall Quality

I worked diligently on this coursework and believe I achieved excellent results. I carefully reviewed the official Jakarta EE documentation on Servlets and JSP. Additionally, I self-studied some concepts like reflection, enumeration. In particular, I effectively implemented the aforementioned object-oriented programming design principles in the Model and Controller layers. The View layer's frontend code might be a bit messy, but it ultimately achieved a highly satisfactory outcome.