COMP0004 Coursework Report

The design and programming of the Notes Web Application occurred as such:

- Creating an outline of classes and methods needed to meet Requirements 1, 2, and 3 as described in the assignment brief, and then expanding from there.
- Writing and testing classes in parallel with their respective Servlets and JSPs

Using the outline as a guide, classes were written in the following order:

1) Note Class

This contained the main functionality for managing the user's notes. Methods performed actions for adding new notes retrieving, editing, and deleting existing notes. Once it was decided that images would be supported in notes, methods to save an image and get its file extension were also added, allowing images to be created, replaced, or deleted. URLs are stored within the "text" field of the Note but are identified through regex in the JSP and then embedded in hyperlinks on the webpage. Additional free-form text search methods were implemented later in the development stage.

2) Index Class

The main purpose of this class is to get the index of notes, which is simply done by parsing through the "name" field of notes in "notes.json", meaning the index itself automatically updates upon changes to "notes. A later search method was added to allow the index to search for notes within it by name. Though this class was basic, keeping it separate from Note helped maintain high cohesion within both classes.

3) Sort Class

This implemented optional sorting functionality for the user to utilize when viewing notes. Though the sorting methods were at first placed in the Note class, upon the realization that they weren't really dependent on Note, these methods were given their class to maintain low coupling.

4) Category Class

Created to meet requirement 6 of adding categories to the Notes App. This contains methods allowing users to create and delete categories, add/remove notes to/from categories.

5) DataHandler Class

Having realized that the Category and Note classes shared similar instance variables and methods, it was decided to make them subclasses of a new DataHandler superclass.

Since retrieval/updating functionality was present in all Note and Category methods, the getArray method was created in DataHandler to abstract the process of calling a new ArrayNode every time. This method is called by Model in its methods, and then the ArrayNode is passed to the Note and Category methods in Model as an argument.

Additionally, as it was necessary to match a note or category's id, which is passed from the View to the respective note and then retrieve data from it, a matchById function was made to abstract this process. Similar abstractions were made by creating an updateFile method to update the "notes.json" file after every change made to the ArrayNode of either notes or categories.

This class was a satisfactory use of inheritance and allowed for significant abstraction across several classes.

Evaluation of the Design and Programming Process

Writing the Java classes and their respective servlets and JSPs together was an effective design decision. It allowed errors to be isolated when testing individual components. By building the MVC environment in parallel, later developments could utilize reliable methods, servlets, and JSPs, speeding up the development of new features.

The Model initializes instance objects of Note and Category, allowing it to call their methods within its methods, which are called by servlets. Index objects are created within Model methods as well if required. Though the same data is being passed from the JSP, to the servlet, to the Model, and then to an object, this was useful as it helped isolate errors by identifying which section the data failed to pass through.

Servlets with different purposes remained separate to achieve low coupling. Though they should ideally be limited to receiving requests, calling methods, and forwarding results to JSPs, some servlets e.g. IndexServlet, could receive similar requests from notes and categories. This meant they contained conditional statements to decide what methods to call depending on where the request came from. This was a convenient feature for development but could be viewed as undesirable behaviour for a servlet.

All note and category data is stored in a JSON file called "notes.json" in the "data" directory, except for image files, which are stored in the "resources" directory. Automatic saves to changes were achieved easily due to the use of JSON. As many of the Note/Category methods update their respective ArrayNode, simply setting the "notes" / "categories" field of the "notes.json" file with this new ArrayNode allows the new data to be written to the file upon changes.

Overall, the quality of work was good as classes were well structured and organized. Some OOP methods like inheritance, low coupling, and high cohesion were achieved, and the opportunity for abstractions was used throughout the design process, though better implementation could have occurred.