COMP0004 Coursework Report

Features of my Java Web Application

Each note can:

- Have a name;
- Store text;
- Store a URL;
- Store the URL to an image on the internet;
- Once created be deleted, edited, and renamed.

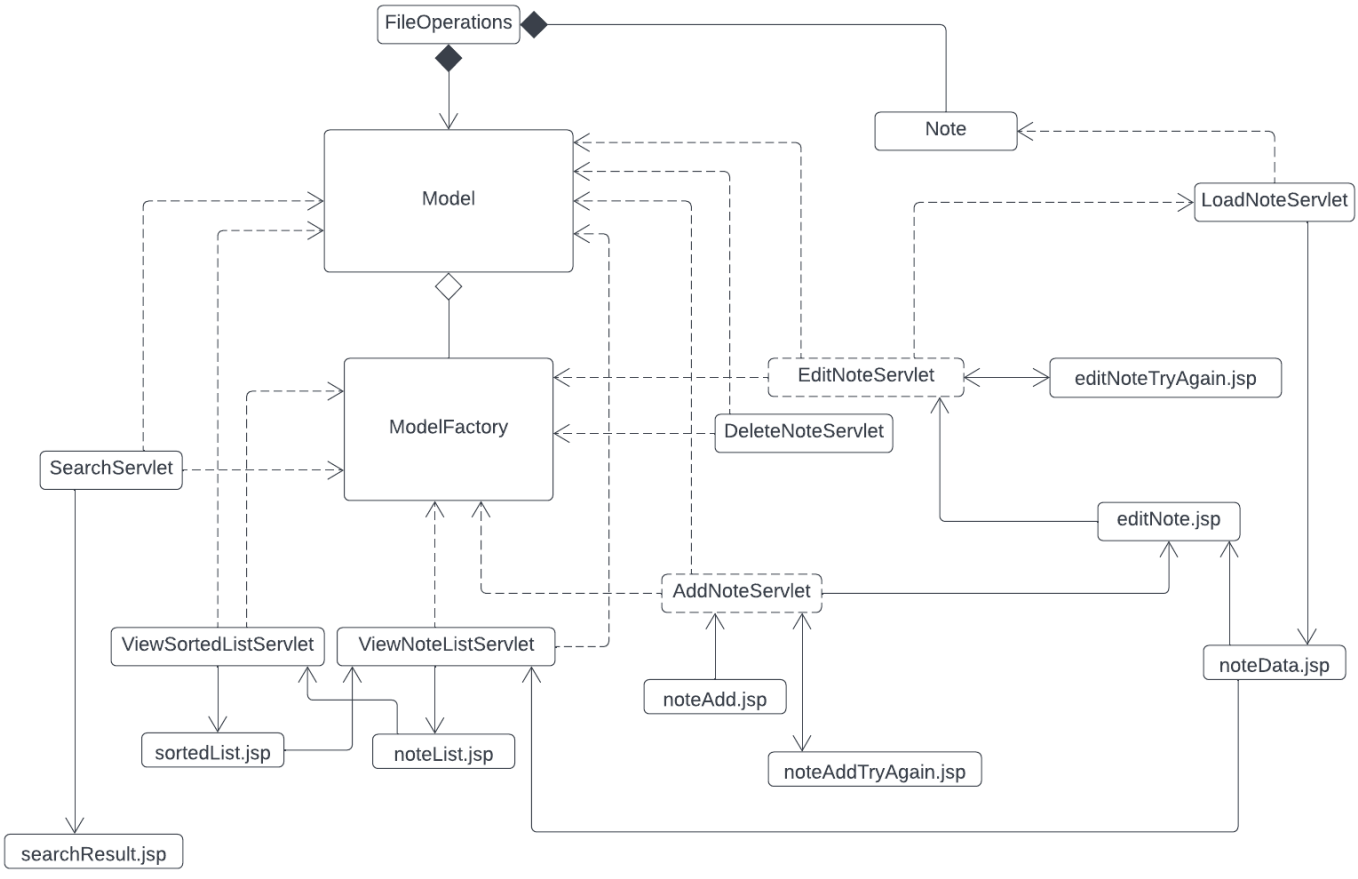
Each note is stored in a separate text file. The names of all the notes are stored in one text file called “noteNames.txt”.

The index holds the names of all the notes:

- In order of date and time added – from oldest to newest,
- And sorted by name in alphabetical order.

When the user is on the index page, they can click on a name to load the corresponding note on to their webpage and then they can choose to edit the contents of the note. The user can also use the search page to look for notes whose names match their search query which will load a new index page with the names of notes that match their search query.

UML Class Diagram



Description and Evaluation of design and programming process

All webpages will interact with only the servlets. Every servlet will then call the relevant methods from the Model class to carry out a given task and forward the results to a Java Server Page (JSP). Therefore, the program will follow the Model-View-Controller (MVC) pattern where the servlets act as the Controller.

However, every class that creates a Model class object will do so via the singleton pattern. This means a static method in the ModelFactory class will return the Model and restricts the program to only one instance of the Model class.

The users will only interact with the JSPs, the webpages, which is the only thing that is visible to them. The JSPs will invoke the servlets to interact with the Model. The Model includes the methods for data handling. Some of which are private and only need to be used by the Model class.

I have used the packages, main, model and servlets to organise my java classes. In main, we have the Main.java file to run my program as a web app. In model, we have Note, Model and FileOperations which are all responsible for directly modifying or reading the text files and act as the Model of the MVC pattern. Lastly, all servlet classes are placed in the package servlet. These all act as the controllers of the MVC pattern and take in input from the web pages the users interact with and forward the output data to the JSPs.

All servlets except for the LoadNoteServlet are related to both the Model and ModelFactory by dependency. This is because, some methods defined in the Model are required to be used in a method in these servlets. The LoadNoteServlet, however, is related to the Note class by dependency because it uses the methods defined in Note instead. The Model is related to the ModelFactory by aggregation because essentially a ModelFactory object is never made but does returns the same Model object. A FileOperations object is an attribute to Note and is required when creating a new Note object, therefore, a composition relationship is established. Similarly, a FilesOperations object is an attribute to the Model class. All JSPs are related to other JSPs or servlets by association.

In the FileOperations class, we have methods to open, create, delete, or write to text files. The Model class will utilise these methods to create new notes, delete notes or make amendments to the notes' contents. The Note class uses the FileOperations class to open a text file of a given name to retrieve the data of a note when a new Note instance variable is created. The LoadNoteServlet will use getter methods to retrieve data of a note to display a note on a webpage. All other servlet classes use the Model to modify each note or retrieve data for the index page.

Furthermore, each servlet carries out a distinct task as part of managing the notes. This keeps the code in the servlets short and precise. They often do not have code that overlaps with other servlets – there is very little repetition of code.

Moreover, I have used encapsulation in my model classes using private instance variables which can only be changed and retrieved using public setter and getter methods. This helps improve maintainability, flexibility and re-usability of my code because I can change the code without breaking other classes that use the code to get or set variables. What the other

parts of the code do, does not concern the other classes that use the getter and setter methods which, consequently, hides the implementation details of the code from other classes.

I designed the FileOperations class so that Model and Note can both use the methods in FileOperations to work with any of the text files where the data for each note are stored. The methods defined here are particularly important because the format I chose to save each of my notes in are text files. The Model class includes methods that concern the index pages to view the list of notes, modifying the notes' contents or searching for notes. The Model class does not have methods to retrieve data of each note because this the Note class is responsible for this. Consequently, the LoadNoteServlet can more easily access the data in each note using the getter methods in the Note class. The Model class will not need to communicate with the Note class because the Model does not need to know about the contents of any notes but only needs to overwrite the text files where the notes are stored.

Having a FileOperations class reduces the size of the Model class and a few of the methods in the Model class to make it more manageable. By more manageable, it will be easier to debug or modify if needed. It also means the Note class does not have to interact with the Model class but only with FileOperations. It also reduces the number of imports required for each class. For example, only FileOperations would need java.io.File. This improves cohesion in my classes which will consequently increase reusability especially of the FileOperations class.

On the other hand, to improve the design of my code I can create more abstractions. For example, I could add two more classes, splitting the Model class, where one class is responsible for the data on the index pages and the other class for updating the contents of a text file.

Overall, firstly, I have used good abstraction to keep my methods short. For example, defining methods to do with file handling in FileOperations means the same code does not need to be written in the each of the methods in Model. Furthermore, each class has methods that can be categorised together for cohesion. As mentioned earlier, FileOperations has methods for file handling, or the Note class has methods to retrieve data of each note. The model classes also make good use of encapsulation.

In terms of the layout and format of my code, I have used appropriate variable names whilst staying consistent with my variable names and the format at which I wrote my code. Throughout I have also used comments to explain the purpose of each method.

To assess the features implemented in my web app, I have met all 5 requirements defined in the coursework sheet. However, there are more features I could have added to improve if it weren't for the time constraints. This includes:

- More ways to view the index of notes e.g., reverse alphabetical order;
- Adding more than one image or URL to one note;
- Adding other forms of data to a note such as categorising notes using colours.