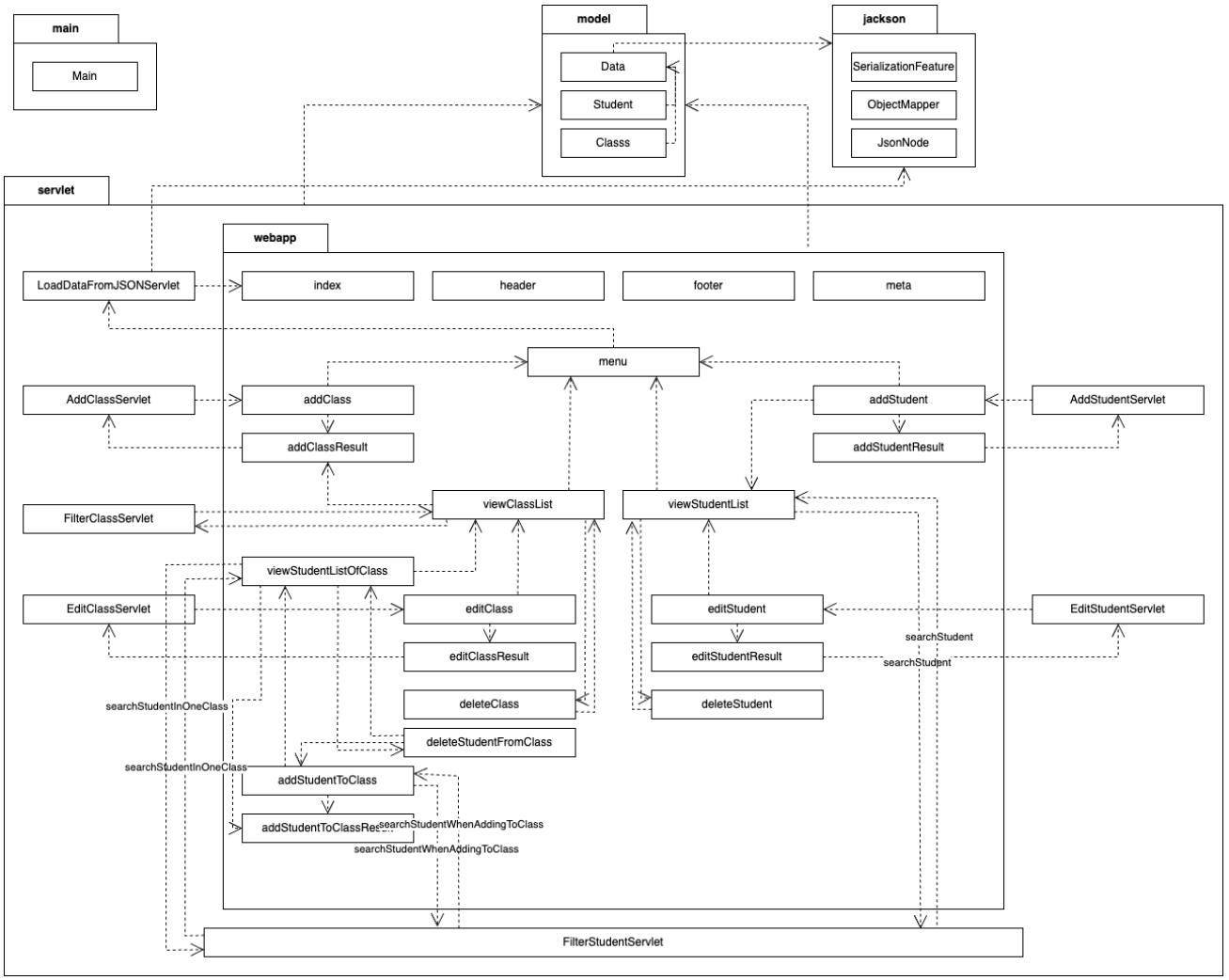
COMP0004 Coursework Report

Section 1

The program, School Administration App is a list management web application written in Java and uses Java Servlet Pages (JSPs). It has functionality for storing, searching, and viewing lists, namely a list of classes and its students in a hypothetical school setting. It fulfils all 6 requirements in the design specification as follows. To start the program, class and student data from JSON files are loaded automatically when the user enters the main menu (Requirement 6). The program stores user-created lists as classes with unique names, while user-created students are the list items (Requirement 2), making it a list of students within each class in a list of classes (Requirement 4). Each student has information on their first name, last name, age, gender, the class they are in, unique student ID and an image URL (Requirement 1). Classes and students can be searched, viewed, renamed and deleted in different JSPs (Requirement 2/ 3), with functionality on search filters based on the different fields listed above (Requirement 5). Every time a change is made, the JSON files are updated automatically (Requirement 6). Built-in images are stored in the assets folder and JSON files are stored in the data folder.

Section 2



Section 3

Overview:

The School Administration App follows the standard model-view-controller pattern. I chose this structure because. We were taught about it in lectures and Referring to the above UML diagram, the model package acts as the model, containing static methods and objects related to data. The servlet package acts as the controller, containing servlets to carry out specific tasks with data from JSPs and forwards to a JSP for the results needed. The webapp package acts as the view, containing JSP files which are converted to Java classes in run-time. The web pages in the app are generated using JSPs. I made sure to separate the three groups of classes into different directories to avoid confusion.

Model:

The program has 3 model classes, one for the Student data type and one for the Classs data type, since the program is about storing and managing information of students in a school. The name “Classs” is used to avoid conflict with the Java keyword “class” and is not a typo. There are the basic getter and setter methods in each class. Naming my type “Classs” turned out to be a design mistake, since there are inconsistencies on whether this program-specific “classs” is used such as in variable type declarations, and whether the english meaning of “class” is used, such as the variable “className” that is used in many of my servlets and JSPs. I will be aware of naming in future projects. The third one, Data class handles all the data storage and JSON file updating. There are methods for adding, deleting and searching Student and Classs objects, along with a update JSON method that is called every time a change to the two global ArrayLists, Data.classes and Data.students are made. ArrayList is used since it is simple to implement and is dynamic in size as the user can periodically add Classses and Students. Global variables are used so all other classes in the program can access and manipulate the two main lists. The former list stores information on classes and the list of students they contain, and the latter stores a list of all students. There is only one instance of the Data class so there is no issue of having multiple main lists. At the beginning, I wanted to code functionality for the students first, which is why Data.students was created. Further into the coding process, I noticed that Data.students was not necessary, since Data.classes already has a complete list of students. However, refactoring code to accommodate for the removal of Data.students was not made as there was not enough time. I think this change can save memory which is something I should have done.

The jackson library was chosen for the JSON file functionalities since it was recommended by other programmers online and I have never handled JSON files before. Only ObjectMapper, SerializationFeature and JsonNode were imported as that was enough for what I needed. This directly ties into how I should write code for reading and writing data, such as telling the program which key to map to which data type. Therefore, I have to decide on the types of instance variables of Student and Classs class objects, since they are the ones being stored in JSON files.

View:

This part was quite messy because there are so many of them. I started with a menu page then expanded from there as I added more functionality. There is also some similar looking code between pages as I coded more of them, so some of my JSPs can probably be merged into ones that can display multiple functionalities on the same page for different conditions using more parameters and attributes. I intentionally split viewing, editing, deleting and adding into separate pages, one of each for Student and Classs and a results page for some of them to show if the action has failed for being invalid or not. I am not sure if having my pages perform one function only is good or bad. On one hand it is clear what the page does but on the other hand there are so many pages. I have no web development experience so I honestly do not know. For the looks of the program I tried to make something that looks neat and simplistic. I know aesthetics are not graded but I felt like making my app look presentable anyway. Figuring out the different html features was pretty fun. I encountered problems with passing around data between classes. For the same page, data is sometimes passed from another JSP and sometimes from a servlet, so there was some confusion in using parameters for pages and attributes for servlets. There were a lot of null pointer exceptions when I was coding since data was not passed correctly and became null and I should make this more consistent. For navigation between pages, I tried to make it user-friendly by adding things like a back button, naming the <href> tags with enough information for the user and images as visual aid.

Controller:

I like how systematic this part works. The most confusing one for me was FilterStudentServlet used in searching because I tried to make it generate a filtered list of students under 3 different conditions: searching from all Students, from Students of a specific Classs and from Students not in a specific Classs when adding Students to a Classs. It took me some debugging to get the parameter names right since I kept directing myself to pages with wrong parameters. The LoadDataFromJSONServlet was a weird one because at the beginning I added it to load in data from my JSON files, so there are a bunch of helper methods other than doGet. I did not really know how servlets function at the start and should just put the other methods in Data where data handling is done anyway, which I did at the end for consistency. Other than searching, I made servlets for adding and editing. There was none for deleting since it is only a few lines of code for the way I implemented it so I just included those in the corresponding JSPs.

Conclusion:

As the first web application I made, I think this was an alright result. I was able to implement all the requirements but the implementation was quite messy and nowhere near well-polished code. There also were not many object-oriented-programming concepts such as inheritance even though my app functions without them. I think inheritance is not necessary in the app since there are no two or more classes that have the same method names to inherit from.