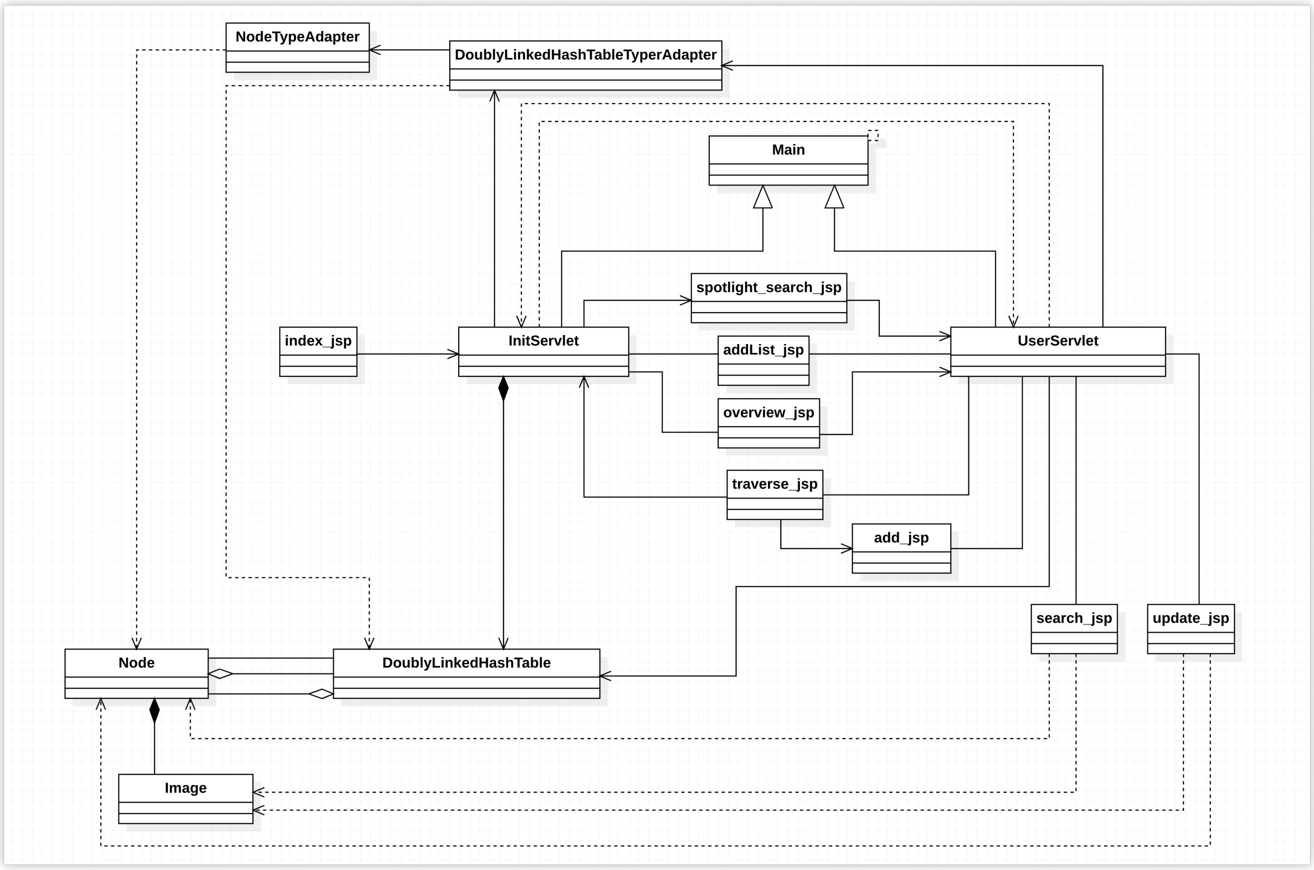
DEVELOPMENT REPORT

SUMMARY OF FEATURES

The program is capable of storing multiple data types in a project, such as text, numbers, images, URLs, and links to other lists. Each item is given a name and supports viewing, adding, deleting, and editing functions. To implement a list with O(1) time complexity for insertion, deletion, and search operations, a data structure named DoublyLinkedHashTable is utilised. The list can be located through its unique name and supports various functions, including viewing, searching by name, inserting an item, deleting an item, renaming a list, creating a new list, deleting a list, and modifying a list. Additionally, the program features the spotlight_search function, which quickly locates items with the same name in all lists by searching for item names in the main interface. Furthermore, the program supports data storage through JSON files and real-time updates. Engaging interfaces were created to visualise the program's features, using InitServlet to initialise the web page, read JSON files, and UserServlet to recognise and execute user operations, redirecting them to other pages accordingly. The web pages were created using JSP files, written in JSTL, including add, addList, index, overview, search, spotlight_search, traverse, and update.
To represent images, an Image Class was constructed, supporting input of local addresses or URLs and reading their contents. To enable custom data to be written to a JSON file, the write and read methods of two custom TypeAdapters were rewritten to support the writing and reading of DoublyLinkedHashTable and Node.

UML CLASS DIAGRAM



(Drawn on StarUML)

EVALUATION

Design

Overview:
The project design combines several packages to store the different programs. The core package includes the Java package for backend operations and the resource package for user data, while the webapp package stores the web interface. This facilitates the maintenance and modification of the project. Within the Java package, there are three different packages: Entities, Main, and Servlets, which store the design data types, initialise the project and handle web interactions, classified according to the type of backend operations. And Maven is also used to manage the versioning of imported external JAR packages.

Classes:

- **Image**: The design of the image class seems appropriate and follows good OO design conventions. The class has clear responsibilities for handling image data and well-defined methods for setting up and retrieving image sources, image data and encoding images. The image source and image (stored as BufferedImage) are encapsulated as a whole thus remedying the problem of not being able to fetch the image source using BufferedImage alone. At the same time the class provides several easy to understand interfaces that allow the programmer to create and use objects of the class without knowing the internal implementation. This class uses several exception handling to deal with errors that may occur when reading image data from a file or URL. In addition, the choice to encode images using Base64 encoding simplifies the sending and receiving of image data.
- **Node**:The Node class appears to be a generic class that can store multiple objects of different types as its content. It uses an ArrayList to store multiple objects of the same type. The class has a constructor that initialises the content with a given object and opens up some of the interfaces for adding content to a Node, getting and setting the name of a Node, getting the size of the content, and getting and setting the next and previous Node in the list. flexible in terms of the types of objects it can store, but also makes it less type-safe, which can lead to run-time errors. This restricts the user to URLs, Images, Doubles, Lists and Strings, rather than storing whatever data they want to store.
- **DoublyLinkedHashTable**: The class implements a doubly linked hash table data structure. By adding before and after pointers to the elements stored in the hash table, the unordered nature of hash table storage and the secondary traversal problem when deleting elements using a single linked table are optimised, resulting in a data structure with an O(1) time complexity for most operations. Optimising the speed of operations is necessary for applications with network transfers. The DoublyLinkedHashTable class uses good OO design practices such as abstraction and encapsulation. The class provides partially explicitly defined methods to perform various operations on the data structure, such as insertion, deletion and modification. These methods make use of lookup methods to locate nodes in hash maps and chained tables. And it provides methods for updating files to read and write double-linked hash tables into a JSON file. This functionality provides persistence to the data structure, allowing the user to store and retrieve data even after the program has terminated. The class also has appropriate exception handling to handle file input/output operations.
- **NodeTypeAdapter& DoublyLinkedHashTableTypeAdapter**: These classes is a GSON type adapter that converts correspond objects to JSON format or JSON to correspond objects by overriding the write() and read() methods in the GSON type adapter. The design uses appropriate

abstraction, cohesion and polymorphism, and adds exception handling to deal with cases where a file does not open properly.

- **Main**: This class is mainly responsible for starting the tomcat server and deploying resources to wait for requests from users. By abstracting it away rather than putting it together with the Servlet implementation it makes the project easier to maintain and reduces duplicate code and avoids the problem of not finding the server when starting the Servlet. It also provides a runnable class to ensure that all requests and responses are made in order.
- **InitServlet & UserServlet**: The InitServlet generates the corresponding doubly linked hash tables by reading the JSON file and passing their names to the interface for the user to select. The UserServlet reads the corresponding list from the database with the key passed in and sets it as one of the properties for subsequent user operations. They receive requests from users and sends them to the corresponding JSP file. The code uses several interfaces, such as HttpServlet, as well as abstract classes and the generic type TypeToken, making the code flexible and extensible. However, for the InitServlet, when the database is empty, examples are created to write files for the user to read, which prevents the user from creating a completely empty database and introduces additional overhead. For the UserServlet, the DoublyLinkedHashTable class used in the UserServlet class may cause thread safety issues. If a user uses multiple pages to manipulate the same list at the same time, this can lead to data consistency issues. Therefore, this class needs to be changed to support multi-threaded access.
- **JSPs:** These JSP files read using the JavaServer Pages Tag Library (JSTL) and EL expressions, simplifying the code and making the pages easier to read. They do not have methods for changing specific Nodes and Doubly Linked Hash Tables, but can only be changed by passing the parameters back to the servlet, keeping the data safe and preventing the overhead of frequent database changes.

Programming Process

I first analysed the data storage of the web application and searched the internet for a data structure that met the requirements. Having established that operation time was important, I chose and implemented a double-linked hash table as my 'list'. I also designed the Node class to store the different data types. After debugging the data structure, I drew concise UML diagrams based on some UML diagrams of web applications on the internet to represent my architecture as well as requests and respond. after that I started to build the tomcat server and Main class to facilitate subsequent debugging work. I then created the JSP file for the basic functionality first then added the processing for the corresponding JSP request in the Servlet at the same time. After debugging all the code and pages required for a 'list' to operate, I used the same process to create and debug multiple 'list' access and switching functions. After all the functionality was complete, I changed the front-end UI, added input prompts and finalised the design.

For time reasons, I did not optimise each operation, I just did some unit tests and modified the bugs, I did not test the design of the database security and stability and the specific performance of the operation. In future development, performance testing of code will need to be added as part of development loop, for example using the Apache Bench to simulate user requests and measure response times, and then optimising fixed methods to improve performance depending on where the program is executed. And add encryption as required to prevent hackers from maliciously destroying or stealing private data from the database. This ensures that the application is secure and stable.

Overall quality of work

Overall, the project achieves most of the requirements and uses a good OO design, but some of the code optimisation and development process needs work. Self-evaluation: 80/100.