COMP0004 Report

Section 1:

Upon launching my app designed for storing, searching, and viewing lists of information, users will be greeted with a screen with 2 options: create a list and view list. After either creating a list or pressing the view list button, the user will be redirected to a page where they can see all the lists currently stored in the application. For each list they can either rename the list, delete it, or create an item to add to the list. Types of items that can be created include URLs, plaintext, links to other lists, images, or a combined item which consist of an image, and one of the other aforementioned item types. Also from the viewLists page, users can search for either an item or a list by typing in the search box at the top of the page. To view the items in a list, the user must simply click on the list name. This will redirect them to a page where they can edit or delete the items in the list. Editing items in the list allows the user to edit the content of a particular item. Lists are automatically saved locally so lists and their content will be maintained upon restarting the app. This app successfully implements all 6 requirements as specified in the coursework prompt with the exception that item editing does not work for combined item types.

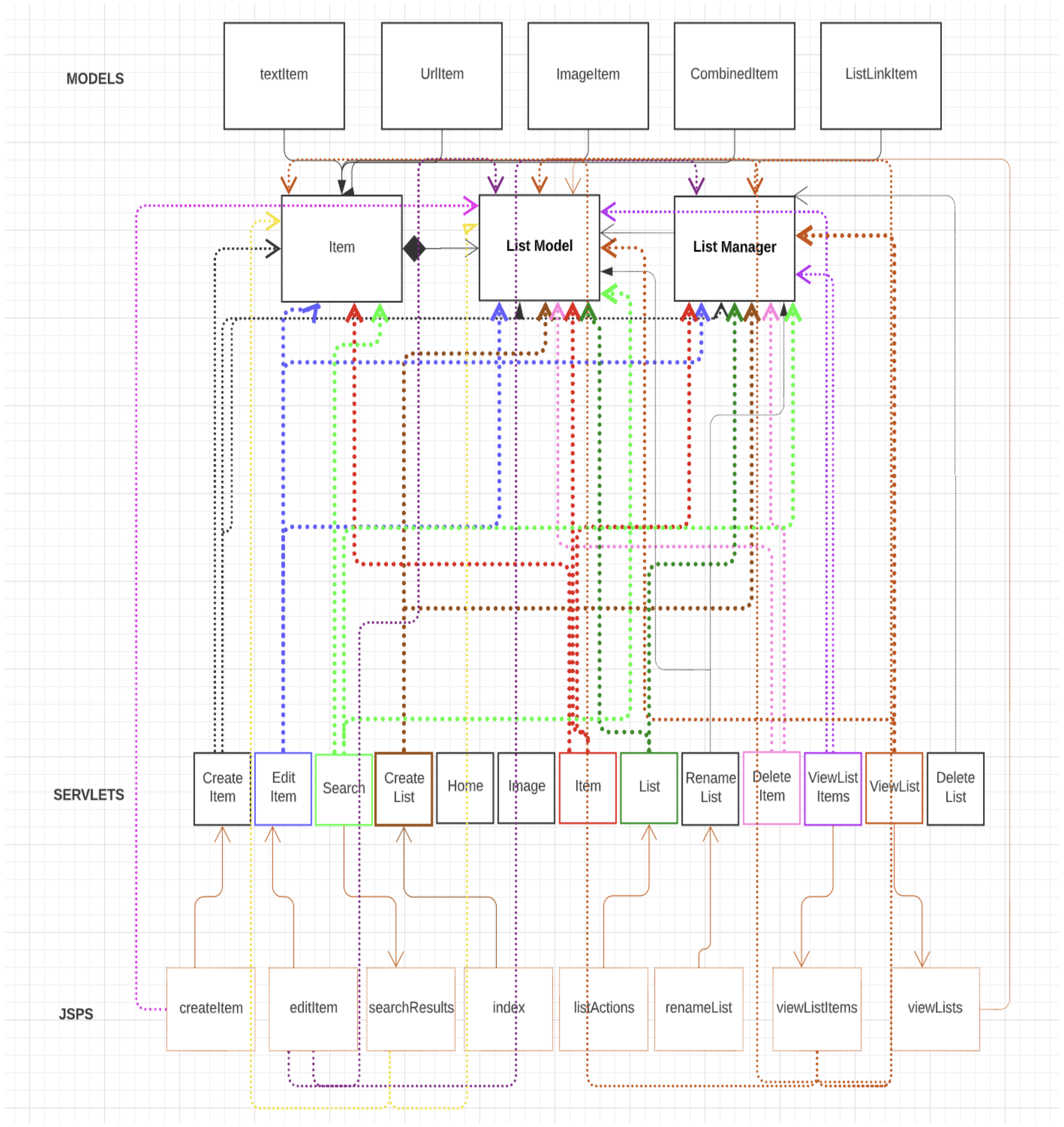
Section 3:

Regarding the design process of the application, I mainly focused on keeping the code well managed and separated. The application is designed around 3 core classes: ListManager, ListModel and the abstract class, Item. These classes separate the different aspects of the application into distinct classes, making the code more modular and cohesive.

The item class is an abstract class which represents individual items- within this class is the abstract method getContent(), which was later implemented for each of the individual item types. Making this method abstract not only made the code easier to read but also enabled easier debugging of specific implementations of getContent. The ListModel class represents a list which has a name, a unique ID, and a collection of items. These fields are all private and not accessible from outside the class, but the public methods such as addItem() etc. allow interaction from other parts of the codebase. This separation of the internal representation of the list and the public interface prevents direct manipulation of list data, reducing risk of errors that may arise from tight code coupling, and also makes errors easier to identify. Similar to how ListModel encapsulates methods regarding items as described above, ListManager encapsulates the logic for creating, retrieving and deleting lists in its methods, again making it easier to identify and resolve issues. Additionally, saving and loading of lists through serialization is also abstracted, meaning that changes to the storage system are less likely to affect other parts of the application and lead to new bugs.

In addition to the abstraction detailed above I tried to be mindful with regards to making my code readable, and also named methods and variables suitably. This, in tandem with my code being well organized led to an overall cohesive codebase which allowed me to more easily resolve many bugs that appeared thus leading to a smoother development process. Overall I am happy with the quality of my code, however I believe it could have been more heavily commented.

Section 2:



NOTES:
-To make an image item users must enter the name of a png file stored within the data file in the project directory
-To make a list link item they must enter the ID of the list they want to make a link to

-In the case of a URL item, the link must have “https://” i.e instead of “google.com” use “https://google.com”