List Manipulation Web App
Gabriel Bor

Feature summary

I have developed a web application that enables users to create lists of items. A list has a name and can only contain item objects. Each item must have a name and in addition, it can have text, an URL, an image, or link to another list. Users can view all lists via the "Data" button on the navigation bar, which shows a table containing all the data of each item grouped by the list they are in. Clicking on a URL takes you to that website, while clicking on an image displays a larger version of it. Clicking on a link to another list shows data in that list, while clicking on the name of an item allows you to edit the item, and the name of a list shows the data of that list only.

The "Editor" button on the navigation bar leads you to a page where you can edit lists by adding or deleting them, as well as changing their names. By clicking on the name of a list, you can view the names of the items in that list, delete items, or add more items. Clicking on the name of an item leads you to the item editor, where you can change its name, add or remove text, link it to another pre-existing list or reset it to link to none, and add or remove a URL or image. By having an item in a list link to another list of items you can have lists of lists. The changes you make get saved upon clicking the submit button or cleared using the clear or reset button.

The web app also features a search bar in the top right corner of the navigation bar that enables you to search all list and item names, displaying any that contain the entered query. Clicking on any of the search results brings you to that list or item.

Program design

Regarding the application itself, I have designed it in an efficient and effective manner. I have utilized appropriate classes and good object-oriented design practices. All web pages are generated by Java Server Pages (JSPs), while Java Servlets handle requests from the web page. Requests are received whenever a button is pressed, a link is clicked, or a form is submitted by the user on the web page. The servlet then calls methods on the model to carry out a specific action before forwarding the results of that action to the JSP to display the new update. The servlet can reference the Model by requesting the model object from the model factory using the static method ModelFactory.getModel(). This allows the same Model object to be referenced each time.

Once the servlet has a reference to the model, it can use the methods in the model class to manipulate the data and the Item and AList classes. There are two primary packages: the alist package and the item package. Within the alist package, there are two classes: the AList class, which represents a list, and the static class ListFilesEditor, which helps to synchronize the data stored with the list and item objects. Each AList object has a name and a list of item objects. The methods in the AList class allow you to get and set the name of the list, as well as get the items and add an item to the list.

The other package is the item package, which contains two classes: the Item class, which represents an item, and the ItemFilesEditor, which helps to synchronize the data stored with the list and item objects. Each Item object has a name and can also have text, a URL, an image, or a link to another list. The methods within the Item class allow you to set, get, and manipulate each of the local variables within the item.

There are also two static classes within the fileIio package. One is called FileHelpers, which helps to delete a directory, make a new file, or read the content of a file. The other static class is the GetFileData class, which allows the data to be stored in list and item objects when the web app begins to run.

I used Maven to manage, build, and run the application. The Main class uses the Apache Tomcat server to run a web application located in the "src/main/webapp/" directory on port 8080. It also adds an additional directory of classes to be loaded by the web application. The program starts the server and waits for it to exit. The ModelFactory allows the servlets to access the same instance of the Model, and the Model class has an array list of all the lists and methods that allow it to manipulate the lists or items. There are also methods to search through the lists and items within the Model class.

There are many different servlets – one for each function that the user might want to perform. Servlets handle requests and responses by accepting requests from the client side, processing the requests, generating dynamic responses, and sending them back to the client side. The data is dispatched to the Java Server Page where an HTML page is generated and sent to the client's browser. The JSP handles the presentation of the data. I also used CSS and Bootstrap to style the web pages, making them look cleaner and more presentable to the user.

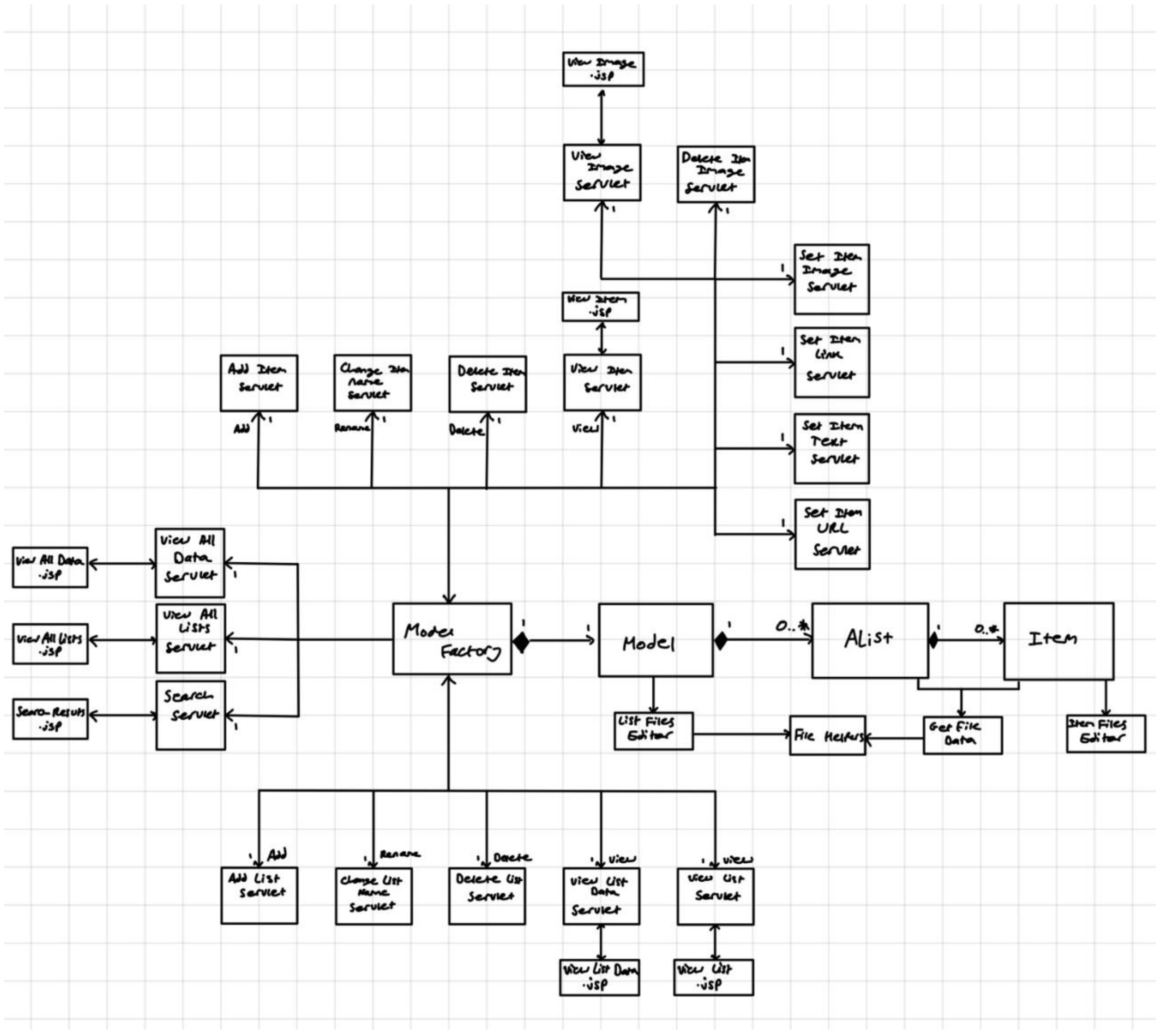
I chose to use directories to store the data. Each list is its own directory and has the name of the directory as the name of the list. Within the list directory, there are directories for each item with the name of the directory being the same as the name of the item. Within the item directory, there is a text file for the text, one for the URL, and one for the name of the other list that the item may link to. If the item has an image, it is stored as a JPEG file within that directory too. I chose this method of storing data due to its ease of use both in manipulating and reading the data.

In terms of future development, there are a few areas where the application could be improved. Firstly, the search feature could be expanded to include searching for specific attributes of items, such as URLs or images. Secondly, the application could be made more user-friendly by adding features such as drag and drop to rearrange objects within an item or allow for an item to have multiple images or multiple URLs. Additionally, users could be allowed to customize the appearance of their lists and items, such as changing font sizes or colours.

Overall, the List Manipulation Web App is a useful tool for creating and managing lists of items. It has efficient design, uses appropriate classes, and uses good object-oriented design practice (abstraction and static classes). It has a very user-friendly interface, robust functionality and runs smoothly.

UML class diagram

Here is the UML class diagram for the web app. As you can see there is a 1-to-1 relationship between the Model Factory and the Model then a dependency relationship of 1-to-many with the AList class. There is only one Model which can be referenced and so this is used to manage all operations between the client side and the backend. A list object can't exist without the model but there can be many list objects for the one model. The same is true with the relationship between the AList class and the Item. There can be many items in a list, but an item cannot exist without being in a list. The servlets all reference the Model Factory to retrieve the single Model in use. Each servlet either invokes a change in the list or item classes or updates the data through the connections with the model, item, or list classes. Some servlets invoke new JSP files which become classes at runtime.


View Image .jsp
View Image Servlet
Delete Item Image Servlet
Set Item Image Servlet
View Item .jsp
Set Item Link Servlet
Add Item Servlet
Change Item Name Servlet
Delete Item Servlet
View Item Servlet
Set Item Text Servlet
Set Item URL Servlet
Add
Rename
Delete
View
1
View All Data .jsp
View All Data Servlet
View All Lists .jsp
View All Lists Servlet
Search Results .jsp
Search Servlet
Model Factory
Model
AList
Item
0..*
List Files Editor
File Helpers
Get File Data
Item Files Editor
Add List Servlet
Change List Name Servlet
Delete List Servlet
View List Data Servlet
View List Servlet
View List Data .jsp
View List .jsp