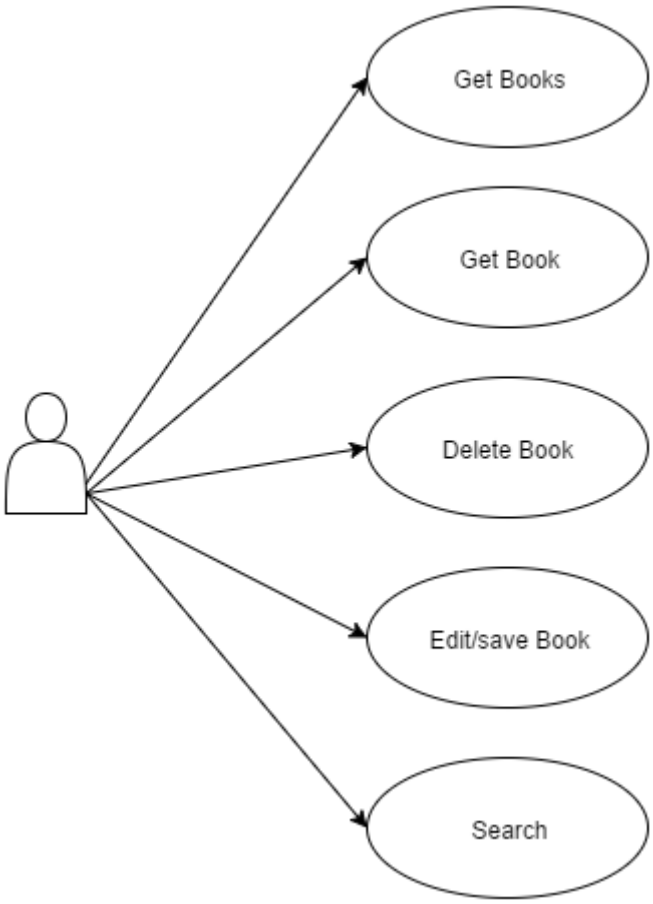
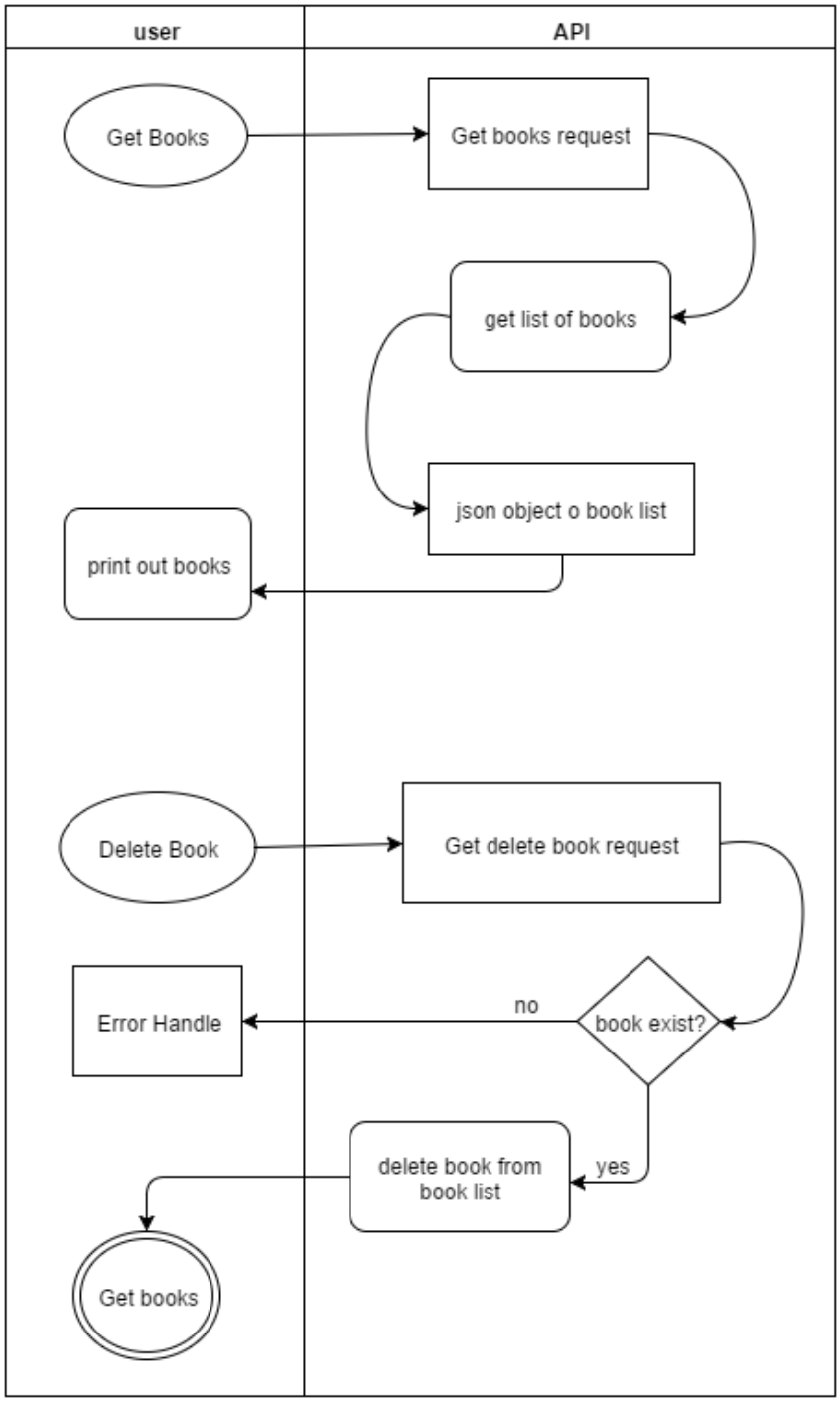
Task 1

Subtask a

The different use cases that is used in the system.



Two specific use cases (Get Books and Delete Book):



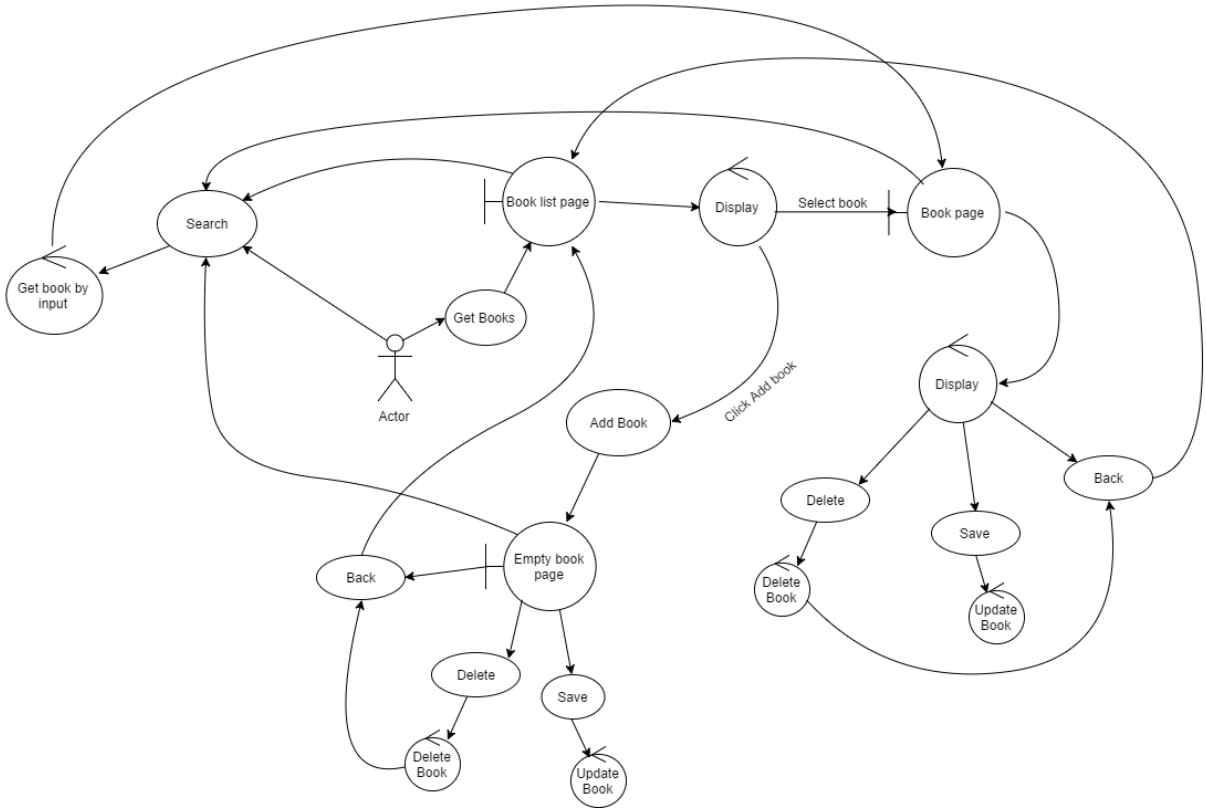
Reflections on subtask a

This UML was not hard to make nor was it very time consuming either but understanding how to make it was the big issue. It was hard because it was all quite new, sure in the lectures we have got some uml to look at, but it did not teach us how to make it only how to use it and why. Purhaps this task was the learning on how to make it. I did not expect the uml part to take very long, but the learning to how to actually create it I was counting on taking quite a time, I was right.

A quick note, I have absolutely no idea if this is even right.

Subtask b

The robustness diagram:



How it works:

User have two choices in the beginning, get books or search. When getting the books, the booklist will be displayed and the user can either click on a specific book or add a book. Clicking on a book will take you to the book page were it will give you the options to delete, go back and save. Adding a book will take you to the same page but no information will be displayed (empty book).

And when searching it will display the requested book (if match).

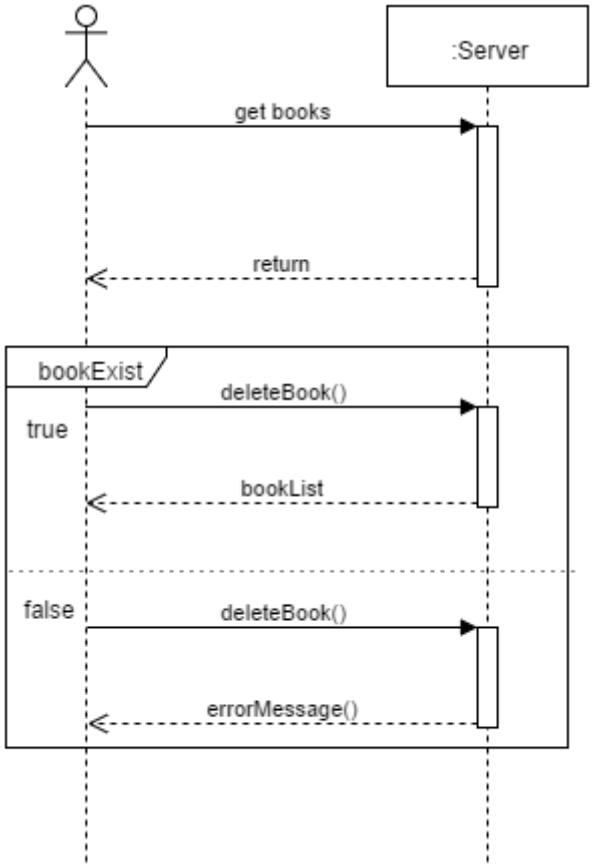
In all cases you can search for a book.

Reflection on subtask b

This task went little faster than subtask a, but the task itself was a bit unclear. I do hope this is correct there is so much more that is going on e.g. when getting a specific book, it has to select it from the list and if it's not there an error or something should be returned. But because of the simplicity and the fact that robustness diagram is supposed to be simple I choose to not include that sort of stuff.

Subtask c

Sequence diagram: get books and delete book



Reflection on subtask c

First of all I want to explain this as I don't think it's correct. The two cases should maybe be on two diagrams, also when deleting a book I wanted to make a check if book exist (that's why the frame). When true the updated booklist is returned so user can print out the rest of the books and when false the user receives an error.

Once again this task was hard on the point that I did not know anything about sequence diagram (I still don't) I've been looking up how sequence diagrams look, what certain things do and all but that doesn't help that much. If I were to do this all over I would know better and probably produce better diagrams.

Task 2

If I got the task correct my goal is to show a way to get the content of the XML, add it as books in the booklist. It says I should convert into objects and then json, but my book object already does that part so I see no reason why I would convert to json after making them into book objects.

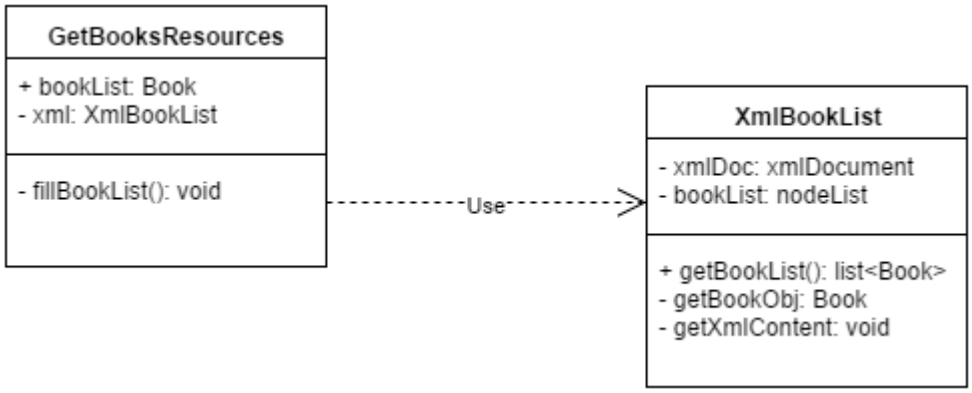
In order to make this I have to first show a way to get the xml file then the content and lastly convert it into book objects. This only in UML i.e. no code.

How I would have code it:

1. In a method call the url and store the data in a xmlDocument (xmlDoc)
2. Make a list of book objects
3. Get the nodes from the xmlDoc
4. Iterate the nodes
5. Making a book object from the current node
6. Adding the book into the list
7. Returning the list

Then just when starting the server up, only call this method and adding the list to the main booklist

The question is how to explain this in UML, by making it into a whole class itself and show how the system would use it?



Reflection on task 2

Once again, this whole thing is really unclear to me, should I explain how the get xml content work (in a simplified way) or how it's integrated in the system. I tried to do both, but it doesn't really say much. Feel like I should perhaps learn uml better to make these tasks but this assignment is maybe it.

Task 3

Reflection

I had a problem at first, not being able to get the id of a book but once I made the xml part it got fixed (had to do how I implemented the “auto” id before). Once the xml part was done the remove book was easy to make, just read in the xml, modify (remove) and write it back to the xml file. If I were to make it better I would have the xml data stored in a globally making it able to get the xml directly without first having to read the hole file then write it back. The only thing that took time was the problem fix i.e. the id of books. The code part was kind of straight forward.