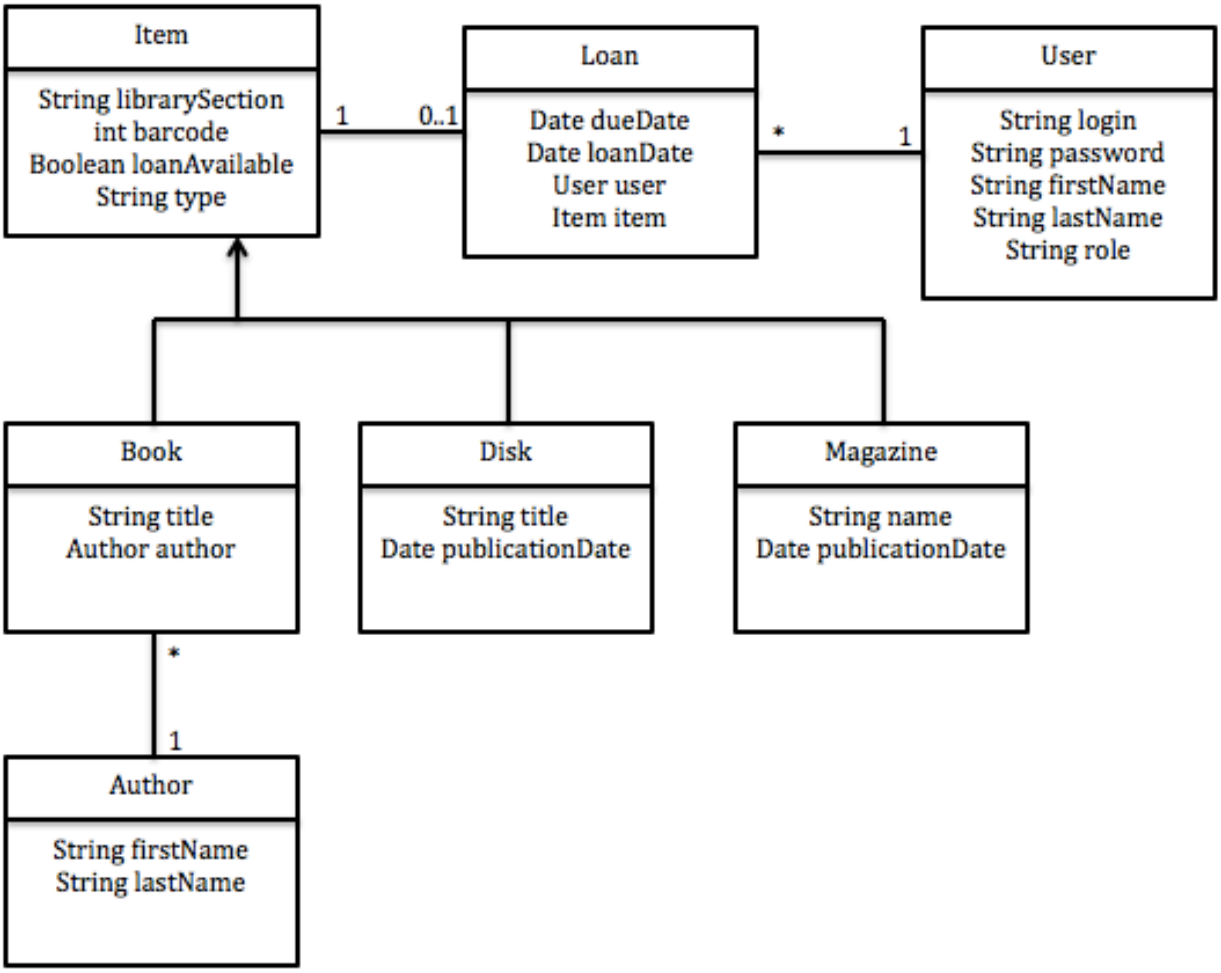
Design Paper

Assignment 5

Manon Chancereul: A20330164

I. MVC elements and their relationships



We can have seven domain classes: Item, Book, Disk, Magazine, Loan, User and Author.

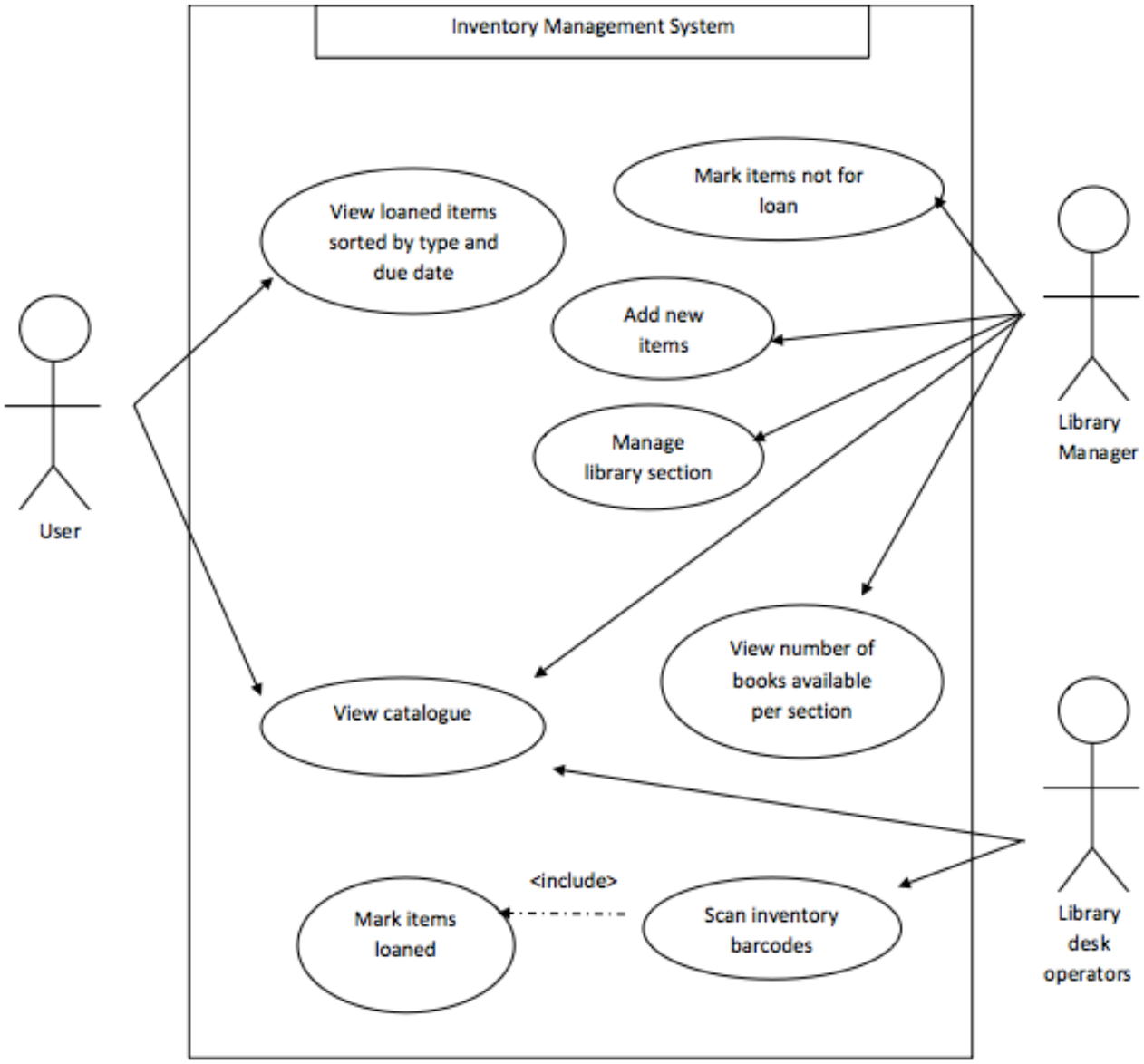
Item is a generalization for Book, Disk and Magazine. It will have four attributes: the library section, a unique barcode, a loan available Boolean to know if the item can be loaned and a type which is Book, Disk or Magazine. Each Disk will have a title and a publication date. Each Magazine will have a name and a publication date. Finally, each Book will have a title and will be associated to a unique Author. This Author will have two attributes: his first name and his last name.

An item can be associated to a loan. The loan has four attributes, which are the due date, the loan date, an item and a user. The loan is associated to a unique user and the user has five attributes: a unique login, a password, a first name, a last name and a role. The role could be user, library manager or desk operator.

A book has a unique author but an author can have written multiple books. A user can have multiple loans but a loan is associated to a unique user. You have one item per loan and a unique item can only be on one loan (several copies of an item will have different barcodes).

We will need a controller for each domain class, and a view for almost each controller, except the author and the user.

II. How to build and link the MVC elements in a working application



I choose to describe the project for an implementation on GRAILS.

In a GRAILS application, we will create seven domain classes. These classes will be done in the domain class folder. You have to create a package for all these classes. The simplest way to do that is to create a package the name of your project in GRAILS. The seven classes are the Author, the Book, the Disk, the Magazine, the Item, the Loan and the User. The Book, Disk and Magazine classes will extend the Item class. We will write in each class every attribute described in the section above. It can be interesting to write a toString method in each domain class if we have to display these classes later on.

To start the application, we could create new Controllers, one for each domain class. In the controller folder, create a package with the same name that the package of the domain classes. In that package, add the seven controllers in creating new controllers. To begin with, and see if our domain classes are good, we'll just write "def scaffold = true" in each controller. We can verify our application with the CRUD operations already implemented with the scaffolding.

For the views, we have a folder named layouts in the folder views and a file main.gsp. We have to modify this one to delete every grails layout from every page of our application. In the views folder, we will have a folder for each of our domain classes. In these folders, we will create pages concerning the domain class.

In the grails-app folder, in the assets folder there is a image folder. We can remove it. It is pictures, logos banners of GRAILS basic application.

III. Application code layout

To start the application, it will be interesting to modify the BootStrap.groovy in the conf folder to add some data in the local database. We should add three users: one with the role user, one manager and one desk operator. Then, we should create at least one item of each type, one author and two loans associated to the user with the role user.

To start the application, we need a welcome page that is the index page in the views folder. On this welcome page, we need a log in form for all types of user to log in. This form could be a template, rendered on this index page. On the welcome page, we should also have a navigation bar but with just a button to log in and the button Home that redirect to the same index page. Considering the role of user, we should add some links in the navigation bar. Whoever user is logged in, we will need a button to log out in the navigation bar.

We may have a link to register new users. If we do that, we will need a register form, which can be a partial page rendered on the welcome page or on the page that is redirected by the link register from the home page.

When the user logs in the application, I assume he can see the catalog of items. We need to add three links in the navigation bar to see all the Books, all the Disks and all the Magazines. These links should go to a "show all" function in the book controller, the disk controller and the magazine controller depending on the link clicked. These controllers will redirect to an index.gsp page in each Book, Disk and Magazine folder in the views folder. The user can also see the items he has loaned so we need another link in the navigation bar to go to this function. This function will be implemented in the loan controller and will show all the items loaned by the user logged in. It will redirect to the view named index in the loan folder in the views folder.

If a desk operator logs in, there will be the same navigation bar with the link to the Books, Disks and Magazines and another link to scan the barcodes. This link goes to the item controller, which directly redirect to the scanBarcodes.gsp view in the item folder in the views folder. On this view, there should be a form that receives the scan of barcodes from items and the user login, and goes to the item controller. This one verifies the identity of the user and put his item into loaned status. The desk operator has a message saying that the specific user has loaned the specific item.

Finally, when a library manager logs in, we add a link on the navigation bar to manage the library sections in addition to the links to see Books, Disks and Magazines. On the page to see the different items, the manager need another column to edit the item with a link on each row and after the table a button to add an item. The edit and add links goes to the specific item controller (ie : Book controller) that redirect to a page named edit.gsp or add.gsp in the specific item folder (ie :Book folder). These pages display a form with the lines needed. For the add form, there is a line for each attribute of the specific item. For the edit form, there only is a line for the library section and the loan availability. When you submit these forms, it goes back to the specific item controller in another closure, edit the information or add the specific item and redirect to the index page of the specific item. When the manager clicks on Manage library sections, it goes to the librarySection page in the item folder (redirect via the item controller) and display a search bar. In this search bar, the manager can enter a library section. The submission goes to the item controller in a search closure, which renders the list of items in this specific library section on the same page (librarySection page).

IV. Major visual layout considerations

For this application, we need a simple layout because very different users like the library manager, the desk operators and every client of the library will use the application.

In the CSS folder, in the web-app folder, we can add bootstrap style sheets. We can reference them in the beginning of the head and the end of the body in the main view in the layouts folder in the views folder.

We need a simple navigation bar that will have different links on it depending which sort of user is logged in.

To show the catalog of items: books, disks and magazines we will use a bootstrap table.

V. Security and access control

Each user will be defined by a role. Depending on this role, the user will see different things as explained above.

For the security of the application, we should test on every page that the user has the role expected, before displaying the content. For that, we can create a service that will give back the role of the user logged in and call the service in each function or closure in controllers.

The password should be at least hashed before registered in the database and may also be salted.

If we do a register form on the public page, this could automatically put the role to classic user for this new user.

VI. Deployment Strategy (technology and databases)

For this application, the technology used is GRAILS. We use the 2.4.4 version with a Java JDK version 1.8.

For the development of the application, we will use the memory database included in the GGTS software. When the development will be at the end, we could use an external database like the MySQL database and to do that we will have to change the JDBC string.

VII. Additional assumptions and questions you would need answered to continue

I assume in this project that a book had a unique author, but an author could be associated to multiple books. I suppose that the different copies of each item will have a different barcode.

One question is when a user return an item, should it be the same system like just a barcode and the system put this item in the status 'not loaned' or do we have to add a step of verification of the state of the item by a desk operator before putting the item in a status that can be loaned?

An assumption I have made for this project is that only a user which role user (not desk operator and manager) can loan an item.

Another question should be about the possibility of a user to register to the application. Can he do it on his own or do we need a step before done by the library manager or a desk operator to let someone register to the application, like giving him temporary credentials?

For now, we have no user that can delete completely items so there is no need to implement some cascade. But if someone could delete items, or users, we could implement a constraint belongsTo on the loan domain classes with the user attribute and another with the item attribute.