Introduction to Programming course project
code review

Overview

The project is implemented in Python language with Django framework. The piece of code for the review describes models of Documents and Copies. If you are not familiar with the framework, you may find this link useful to get the idea of what is a model and how it works in Django.

The link to the GitHub repository with the code is provided below the class diagrams.

Goals

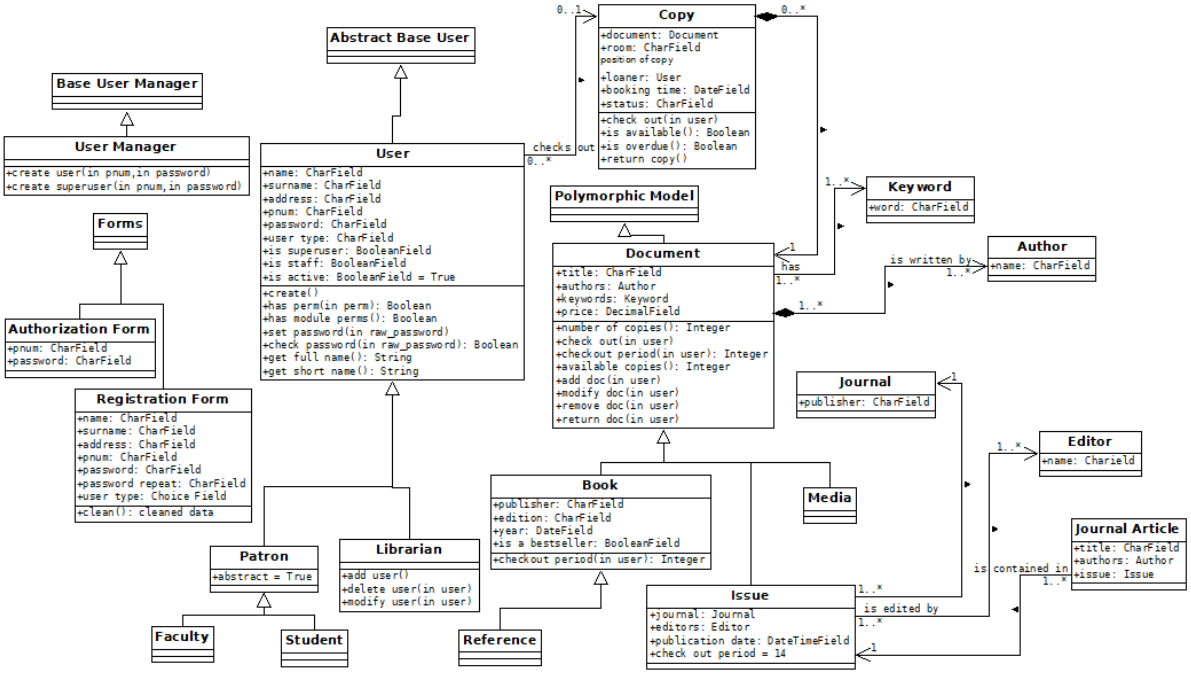
To suggest how to improve the piece of code (try to find and correct bugs, architecture or code style)

Usage Scenarios

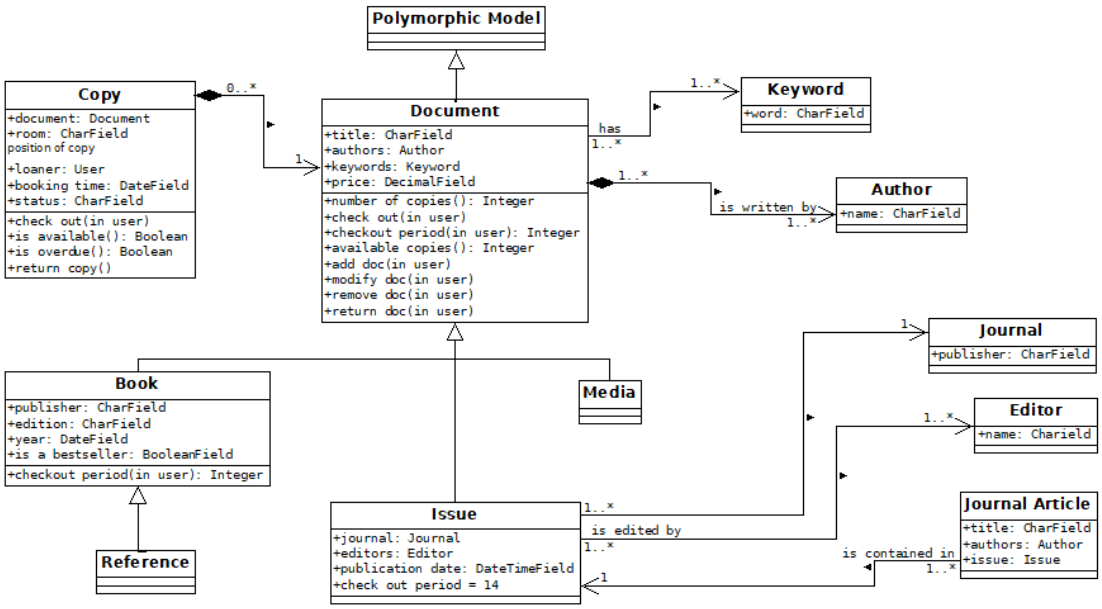
The models contain features like add/delete/modify a document, check out/return a copy

Class Diagrams

The whole system diagram for better understanding Of the models' interaction



Documents and Copies models



Code

Categories for review

1. Design decisions

Example:

[1.1] NameTTT: You should use HASH_TABLE for 'yyy' instead of ARRAY.

[1.1.1] Owner: the decision of using ARRAY is based on the fact that ARRAYs are lighter than HASH_TABLES

[1.1.2] NameTTT: That might be true, but you are compromising execution time, 'yyy' is used to perform several searches. Having a HASH_TABLE one can have constant complexity in searching

[1.1.3] Owner: Ok, I will change the design and implementation. [CLOSED]

2. *API design*
3. Architecture (including inheritance hierarchy, if any)
4. Implementation techniques, in particular choice of data structures and algorithms
5. Contracts
6. Programming style, names
7. Comments and documentation

1. Design decisions

[1.1] Reviewer1: You should define constants in one place, and then use these values when you need them. Hard-coding them all over is not that good.

[1.1.1] Owner: Good point, we will change it. [TO BE IMPLEMENTED]

[1.2] Reviewer1: Revisit the folder/file structure. I think that all the source code should be in the /src directory.

[1.2.1] Owner: The thing is, when a new application in the project is created, Django automatically makes these python files: __init__, admin, apps, models, tests, urls and views inside the application folder. The source code itself can be found in models and views, while the rest of the files primarily take care of the interaction of the project components and settings. [TO BE DISCUSSED] -> TO BE IMPLEMENTED
[1.3] Reviewer1: documents.py is too bulky. Consider moving classes to separate files and folders.
[1.3.1] Owner: Good advice, I think we'll change it. [TO BE FIXED]
[1.4] Reviewer1: waiting_list is created all over again, by calling get_waiting_list. As you are using OOP, consider retaining its state by keeping an additional field.
[1.4.1] Owner: I see your point here but according to the specification a waiting list for a document must be removed every time there is an outstanding request for the document. This is why we have implemented a separate method to update the waiting list. [TO BE DISCUSSED] -> TO BE FIXED
[1.5] Reviewer2: Probably it would be a nice idea to put all the important functions like counting particular timedelta in separate file
[1.5.1] Owner: I agree, will implement it. [TO BE IMPLEMENTED]

2. API design

[2.1] Reviewer3: you can apply the concept of polymorphism: define *check_out_period* in class Document and override it in class Book (if it is possible by language).
[2.1.1] Owner: in fact, this is what we have already done: we have defined the method *check_out_period* in class Document and overridden it in classes Book, Issue and Media. The thing is, the method is declared but not implemented in Document, the implementation is left for the descendants of Document, this is probably the reason why it could be unnoticed. [CLOSED]

3. Architecture (including inheritance hierarchy, if any)

[3.1] Reviewer1: code_review/document_manager/models/author.py should be refactored. Classes Author and Editor are exactly the same.
[3.1.1] Yes, they have the same fields but I do not think we need them to be different and have some other fields besides their names. [TO BE DISCUSSED] -> TO BE IMPLEMENTED
[3.2] Reviewer1: class Reference has nothing, and is exactly the same as the Book class. This may be a bad decision because Document has all the methods for checking out and so on. In check_out you are just checking if the document is reference, and if it is, false is returned. Make a Reference class which would not allow clients to call check_out method from it.
[3.2.1] Owner: [TO BE DISCUSSED] -> TO BE IMPLEMENTED

4. Implementation techniques, in particular choice of data structures and algorithms

[4.1] Reviewer1: waiting_list is created every time get_waiting_list is called. Consider making a separate data structure: Priority Queue implemented as a Heap to reduce overall time complexity.
[4.1.1] Owner: I agree, it would be better, we should follow this advice. [TO BE IMPLEMENTED]

5. Exceptions handling - Contracts

[5.1] Reviewer1: check_out procedure just returns False when preconditions are not met. Consider transforming the code so it throws Exceptions in such cases.

[5.1.1] Owner: ok, we will consider doing so. [TO BE IMPLEMENTED]

[5.2] Reviewer1: as checking if the user is Librarian is a precondition, as often used (`if isinstance(user, Librarian)`), you should consider making a method which checks that. Also, throw an Exception if condition not met, and return 403 forbidden HTTP response.

[5.2.1] Owner: Good advice because we really check if the user is a librarian quite often.

[5.3] Reviewer1: preconditions should be handled better. Make helper functions for all if conditions, and call them as middleware. If these preconditions are not met, throw HTTP responses accordingly.

[5.3.1] Owner: I think you are right, it would be better. [TO BE IMPLEMENTED]

[5.4] Reviewer1: remove_copy should throw an Exception rather than returning false. Also, try/catch block should be avoided.

[5.4.1] Owner: I think you are right, though I am not sure about the reason to avoid try/catch block. [TO BE DISCUSSED] -> [TO BE IMPLEMENTED]

6. Programming style, names

[6.1] Reviewer3: I think it's good to use class static variables instead exact numbers in *check_out_period* method, because most documents have the same period.

[6.1.1] Owner: yes, this is a good point not to use hard-coded constants. [TO BE IMPLEMENTED]

[6.2] Reviewer1: Having ('c', 'a', 'r') in Loan Status map to ('checked-out', 'available','renewed') makes it unclear in other files what do these letters represent. Consider naming them fully (e.g., 'checked-out','available','renewed').

[6.2.1] Owner: Each field takes a certain set of field-specific arguments. For example, CharField (and its subclasses) require a max_length argument which specifies the size of the VARCHAR database field used to store the data. Here the arguments this field can take are specified by the choices ('c', 'a', 'r') ('checked-out', 'available','renewed') accordingly. So, the contractions are used to be stored in database because it saves space.

[6.3] Reviewer1: follow PEP8 naming conventions. Delete redundant new lines at the end of the file, and use __ before any private field.

[6.3.1] Owner: Ok, I agree with the comment, we should change it.

[6.4] Reviewer2: It would be nice to mention what code style u use because now its like combination of PEP8 and your own style

[6.4.1] Owner: Good advice, we will take it into consideration

[6.5] Reviewer4: Leave the same amount of empty lines between functions.

[6.5.1] Owner: Ok, we will follow your advice.

7. Comments and documentation

[7.1] Reviewer3: line 227, I suppose comment should be "Check out document".

[7.1.1] Owner: yes, sure, it is a typo.

[7.2] Reviewer1: be more descriptive/verbose when you are describing classes. Have a multi line/documentation comment which describes the purpose of the class, and which classes is it related to and how.

[7.2.1] Owner: yes, it would be better to describe the classes in the code, however, we were asked to provide a pdf documentation with these data, so the connection between classes and their description can be found there.
[7.3] Reviewer1: most of the one-line comments in the code make no sense. They just repeat what is written beneath them. Consider making more meaningful comments and removing redundant ones.
[7.3.1] Owner: yes, these comments were primarily for ourselves to
[7.4] Reviewer2: many functions have no description, probably it would be good to describe them somehow
[7.4.1] Owner: yes, it is a good practice to write comments to the functions, however, most of them are named in such a way that their names explain what they do. You are right, there are some that can be not so clear though, so we will describe them.