Syllaby Book Writer Assignment Documentation

Assumptions

As there is no mention of UI in the ‘Coding Challenge’ document, I have assumed that only backend APIs need to be created.

Also, I am assuming that only the author can update the book name if required as there is no mention of such.

In regards to the roles ‘Author’ and ‘Collaborator’, instead of a role based ACL sort of implementation, I have created a book-collaborator mapper. An author can be a collaborator on some other author’s book, so for simplifying this task I created the above-mentioned mapper instead of an ACL mechanism. If the mapping record does not exist for a book-collaborator pair or the ‘access_revoked’ flag is true for some mapping, it means that the collaborator is not allowed to work on that book.

Selection of components

1. Framework:

   I chose Django Rest Framework for its ORM, Admin panel, easy validations, serializers and API template classes which make development lightning fast. Most of the trivial things are taken care of when using Django and we can focus on implementing the solution while saving time.

2. Database selection:

   Book content can have an infinite number of sections/subsections as per the requirement. To cater to this need, I have decided to use a JSON field that is supported by PostgreSQL as well as Django.

   To be honest, I feel that MongoDB fits best for this type of storage as it has high scalability and fast data retrieval capabilities. But I chose PostgreSQL for my familiarity of using it with Django and this assignment’s limited scope and time.

3. Authentication:

   Django Rest Framework’s Token authentication does not have token expiry by default and thus is not very versatile. Simple JWT library provides tokens which have configurable expiry. JWT authentication provides access and refresh tokens along with token blacklisting functionality if required. It also has tons of configuration options for customized authentication as per need.

4. Input Validation:

Django's default model serializers are enough for validating the usual database fields. But the Book table's 'Content' field is a JSON field, validating it with a custom validator is a challenge. However, Pydantic is one such library that is very easy to configure and use for validating such unstructured content. It makes the validation of the fields very easy with schema definitions while extending Django's default serializer validation method.