The problem here is that given an user id and a query string, we want to return a list of books to recommend the user to read.

1. The structure of the recommendation system is explained below.

First, based on the query string, there might be some books that exactly match the query string. For example, if the user types in an ISBN number, then we only need to return a particular book. Another example might be that if an user typed in an author name, we may only want to recommend books that are written by this particular author. So, we will first try to match the query string exactly if possible. If this exact match process is possible, then we will get a list of candidate books based on the exact match. Then, we will sort the candidate books by a collaborative filtering system. We will explain the building of the collaborative filtering system later. The collaborative filtering will then sort those candidate books. Finally, we remove any books that the user has already read from the sorted candidate book list and return this cleaned and sorted list of books as the final recommendation. We think a user has read a book when the user has already given a rating about a particular book.

If the exact query string match process is not possible, then we will use tf-idf in NLP to compute a similarity score between the query string and string features of each book. The string features of each book are preprocessed. A string feature of a book is a string concatenation of author names, original title, title, language code and all tags associated with a book. (The exact implementation detail is in the notebook titled "preprocess book string features"). Now, we have a query string similarity score for every book in the database. Also, we use the collaborative filtering system to check the query user id against every book in the database. Even though we are using the collaborative filtering system against every book, this process is fast because the collaborative filtering is based on trained neural networks. Now, for every book, we have two scores. One is based on the tf-idf feature and the other is based on collaborative filtering. Then, we combine the two scores and generate a recommendation book list based on the combination of those two scores. Finally, we remove any books the user has already read from this recommendation book list before showing the final book recommendation.

2. The collaborative filtering system.

The collaborative filtering system is based on a neural network learning latent features about each user and each book. For now, we arbitrarily choose 10 as the dimensionality of the latent features for both books and users. Then, for each actual (user_id,book_id) rating pair, we can think of it as the inner product between the user_id latent feature vector and the book latent feature vector. To train a neural network to learn those latent features properly, we have training data, validation data, and testing data. The keras package will use the adam optimizer to reduce the mean squared error on the training data. We will terminate the optimization process and avoid overfitting by early stopping. This is done by measuring the mean mean squared error on the validation data. Finally, we measure the quality of the neural network by computing the mean squared error on the testing data.

3. Further extension.

One question that is ignored here is scalability. We have a neural network learning the latent features of every book and every user. This might not be possible in a real world production environment as there are too many users and too many books. A simple solution might be that we do a layer of classification first on users and books. That is, we group users into several groups and group books into several groups and we only learn the latent features of those groups of people and groups of books.
Other considerations might be comparing the current recommendation system against other types of recommendation system, such as using SVD to factor the utility matrix.