BOOK MANAGEMENT API

Submitted in partial fulfillment of the

requirement for the award of the degree of

B.Tech Computer Science Engineering

Under The Supervision of
Dr. Avneesh Kumar
Associate Professor

Submitted By:

Name of Student: PALLAV RAJ
Admission Number: 19SCSE1010442
Application Number: 19021011620
Section: 03
Semester: 05

&

Name of Student: RAVI SHANKAR KUMAR
Admission Number: 19SCSE1010675
Application Number: 19021011826
Section: 03
Semester: 05

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERINGGALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA MONTH, YEAR

Abstract

There are lots and lots of digital books (e-book) available in the market. If a person tries to read a book, they have to find their respective books on their official site and download it. The biggest problem arises when a person wants to find a book by its genre/category they have to do many research and find the book best fit for him/her. One more problem is that for a single category there are many books available and people generally gets confuse, which book they choose.

The solution is Book Management API in this we will available the e-books by permission of the publishers or a link to the download page of the books official site. The API solo purpose will be to search for a book and deliver it to you, by this searching a book will be very easy. In this API we will categories all the books by their genre and rank them. If a book is read or downloaded by many people then the rank of the book will be higher, like wise if a book is not so popular the rank will be lower. This will help people choose their books faster.

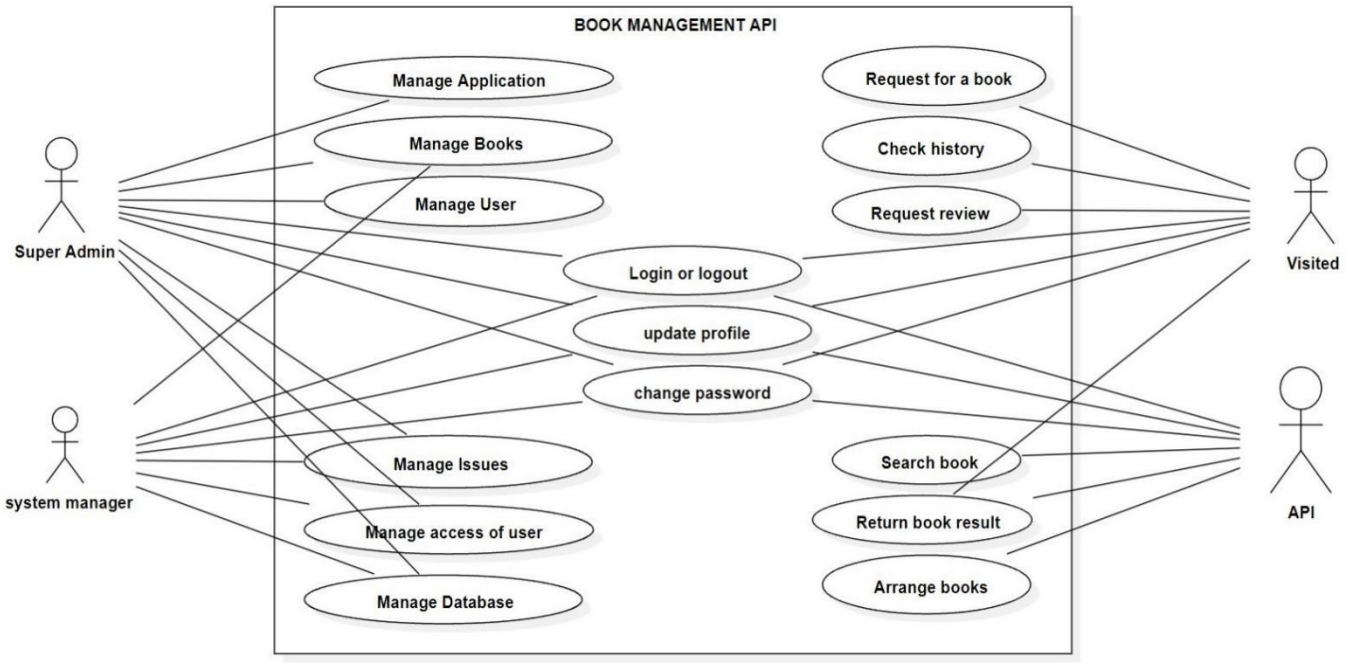
There are many tools and method to build an API. We choose nodeJS which is very popular now a days because it uses javascript to build any web page or API's.
With nodeJS we have used many tools

- Express: popular framework of nodeJS.
- Html / Css / JavaScript: We need some place where we can show data so we choose website because it can be accessed by any device.
- Any Database: We will use mongoDB database for this project to store data / books / username / password etc.

Result: After completing the API and deploying it if a person goes to our website people can able to search for any book in only one click and download it or read it on our website.

Conclusion and future scope: As you can see the fastest way to get a book is to use an API. Which is very popular now and remain popular in future because we can use

List of Figures



List of Figures

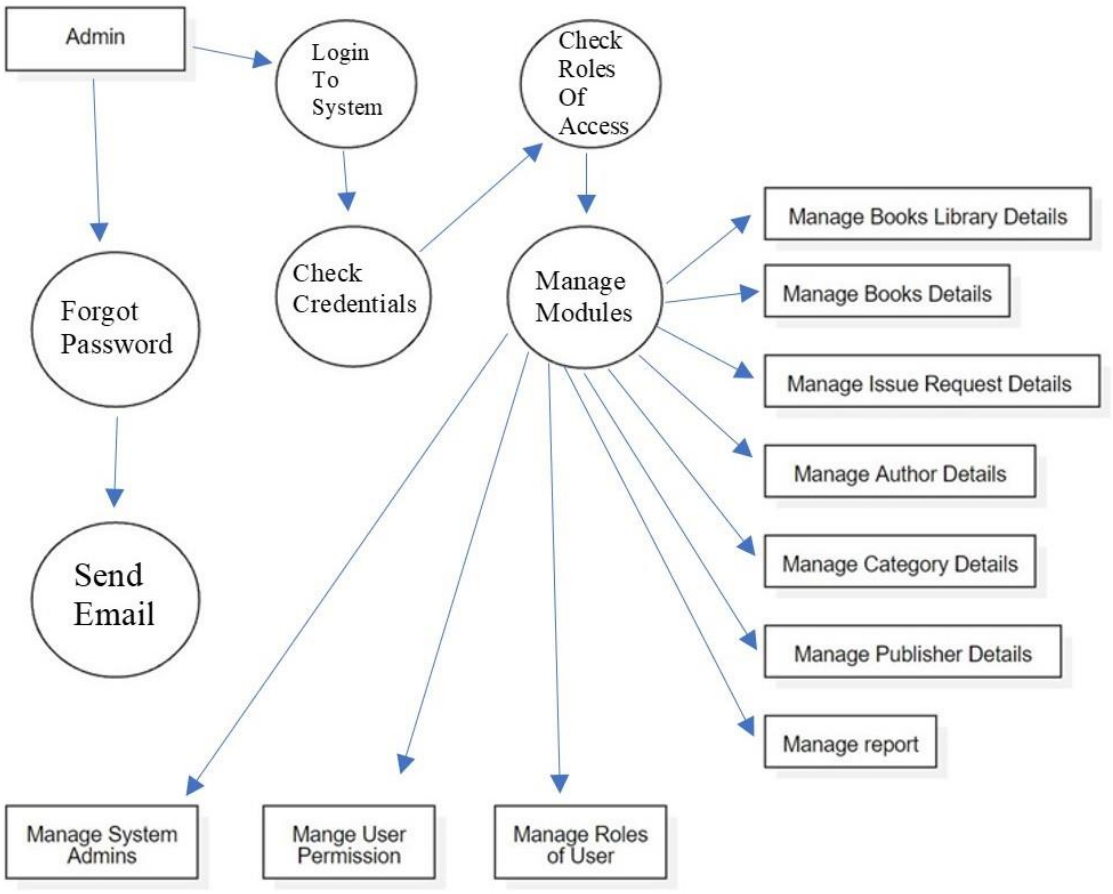


Table of Contents

CHAPTER-1
Introduction

1. Introduction

The Books API is a way to search and access that content, as well as to create and view personalization around that content.

1.1 Modules:

Admin login: Admin is the one who administers the system by adding or removing e-books into and from the system respectively.
User login: Users have to register themselves into the system to create an account. After registering successfully, they can then login into the system by entering 10 digit mobile number and their email id.
Add and Update Books: The admin can add books to the system by entering the details of the books and can even update the details.
Search option: Admin and User can even search for books by entering the name of the book.
View Request of a book-The admin can see which book is requested the most.
Charge money- If the book is not free admin can charge money through any payment gateway.

1.2 Purpose / Working of an API:

- Produce a list of all the books with their authors and genres.
- Give details of each and every book with its author and the genres.
- Provide the support for creating, updating and deleting a book.
- Produce a list of all the authors with their books published.
- Details of each and every author with their published books.

CHAPTER-1
Introduction

- Provide the support for creating, updating and deleting an author.
- Produce a list of all the genres and the respective books in each genre.
- Each genre and the books that fall into this genre.
- Provide the support for creating, updating and deleting a genre.

CHAPTER-2
Literature Survey

2.1 Models

First, we will create the models. There will be 3 models Author, Genre and Book. Book has a Foreign Key field to an Author, and a Many to Many fields to Genre.

2.1.1 Create Sample data

We completed the model now Migrate the models into the database and create some demo records in each table to make sure everything's is working fine.

Created sample records of Book, Genre and Author here. First, we will create instance of Author and save in data base. Genre instance is created and then book is added according to genre.

Till now we have made an API which can receive the request sent by clients to the server and response with all the requested resources to the client. The resources could be in any format maybe JSON or maybe in XML.

In our case, a user/client will ask for a book by its name, or by genres or by the author's name. To send a response to the client, we need to make our response data into appropriate format. In this I will choose JSON.

2.2 . Serialization

REST Framework provides serializing and deserializing functionalities, which converts the instances of the model to Primitive Data Types and then to JSON, or vice-versa.

CHAPTER-2
Literature Survey

The fields option in the Meta class contains all those fields which we want to serialize. The Model used for serialization is available in the model option in the meta class. The serializer above can serialize an instance of a Book, its price, its name and its description.

2.2.1 Serializing single resources

Till now we have defined serializer usage of the book. Structure of the book model is printed when we call the serializer. Serializing the first Book instance serializes only the fields mentioned in the fields option in the Meta class of the serializer. serializer.data gives us the JSON formatted output, the **JSON Representation of the Book Resource**.

2.2.2 Serializing Multiple Resources

We will repeat the serialization process on all the Book objects. The input to the serializer here is a whole queryset, instead of a single record. Converting this to JSON produces error, as it tries to search the fields in the queryset (It will get the name field in a single instance, but not on the queryset). So we have to tell the serializer that this is a queryset with multiple such Book records. That's when it understands and works correctly.

2.3. Views

There's been some changes in writing the views with NodeJS REST framework. We will see these changes one by one.

2.3.1 REST Request

CHAPTER-2
Literature Survey

The request object in NodeJS REST is a wrapper around HTTP Request object with some better functionalities.

REST request provides parsing mechanisms which allows to treat each request object as a JSON media type. It's similar to HTTPRequest.POST, but also parses data from PUT and PATCH methods.I will used Postman to demonstrate the working of the APIs.

Following are the sample views used for the demonstration below. The one at the top uses the default View provided in NodeJS, which uses HTTP methods for request and response. The one at the bottom uses the REST framework to create the request and responses. The request object is debugged using pdb.

HTTP request.body produces the byte string when accessed, whereas to get values as key-value pairs we use request.POST. With REST request.data produces the values as key-value pairs just like request.POST. This is fine, but let's see what we get if the request is raw JSON data.

With JSON data, HTTP request.body gives the data in byte format again, and with request.POST doesn't have access to it. Whereas, with REST request.data, we do see that the data can be accessed like a key-value pair just like a POST request in HTTP. This is what differentiates both of them. No use of request.POST is required here if you are using REST APIs.

With HTTP request, if any parameter is sent over the URL, then it only can be accessed via request.GET, even though it might be POST request. Addressing this, REST request gives request.query_params which gets those parameters from the URL for any type of request.

request.query_params work with any type of requests. It brings whatever we pass on the URL.

2.3.2 Content Negotiation

REST Request provides certain properties which allows us to know the content type finalized during the Content Negotiation process. These properties are request.accepted_renderer and request.accepted_media_type.

2.3.3 REST Response

REST allows us to return the Response with the content type negotiated during the content negotiation stage. It keeps the context of the request and its content type and uses it to create the Response.

This is where we are comparing the HTTP Response to REST Response. With HTTP, the default Response content type is “text/html; charset=utf-8”. So even if the request accepts JSON rendering with JSON media type, the response will be rendered in HTML format. We have to manually change the content type in HttpResponse like the second case. Whereas, the REST Response keeps the context of the request and uses it to create the response.

REST allows us to create a Response without any rendered content. We can pass the data of Python primitive data types to the Response object while creating.

Knowing that we cannot pass any user defined data or a model record to the response, as it won’t understand the format. To send in the model records, we need to serialize

them using the Serializers defined above and then use them to create the Response object.

So here goes the View to fetch the list of Books. And below is a screenshot when we hit the API on Postman. We can see the name, price and description of a list of Books that can be retrieved by the GET request.

The response we get is in JSON representation. And these are the books that are currently available in the database. So this is the current state.

Now, We need to create APIs for Authors and Genres. Also the APIs should support updating, creating and deleting the respective records.