API Documentation

INTRODUCTION

This API will help the user or developer to easily access the book library and its data and easily alter the data present in its Database.
This project also helps the user to add the books in their favourite list and remove books from their favourite list.
The API defined in the project is very easy to call.
The user without authentication can’t alter the books and their favourite list.

Allowed HTTPs requests:

GET- get the resource

POST- update resource

Description Of Usual Server Responses:

- 200 OK - the request was successful (some API calls may return 201 instead)
- 201 Created - the request was successful and a resource was created
- 401 Unauthorized - authentication failed or user doesn't have permissions for requested operation.
- 400 Bad Request - the request could not be understood or was missing required parameters.
- 404 Not Found - resource was not found.
- 409 Conflict -Already update or exist.
- 422 Invalid Input-Unprocessable Entity response

User for login-

UserID - demo@gmail.com
User_password- demo

APIs

get_book(bookID) -

This API is used for get the book by book ID and return rhe response.It return the book data like title,authors,price,rating etc

Argument-

It takes only one value as an argument and returns the response. If book ID exists with 200 HTTP code and if book does not exist it returns HTTP 404 code.

Method-

Method used in this API is HTTP GET method

User login-

To get the book by ID there is no need to login the user first

Example-

http://127.0.0.1:5000/get_book/123

get_favourite(user_id) -

This API is used to get the user's favourite list by user ID and return the response. It returns the book data like title,authors,price,rating etc of the user's favourite books. The user only can access only your favourite list, not anyone else.

Argument-

It takes only one value as an argument and returns the response.If the user doesn't have any books it returns an empty response otherwise list of favourite books.

Method-

Method used in this API is HTTP GET method

User login-

To get the favourite list by ID have to login the userID and password

Example-

http://127.0.0.1:5000/get_favourite/demo@gmail.com

add_favourite(user_id,book_id) -

This API is used to update the user's favourite list. and return the response . It returns a status message that list is updated successfully or not. The user only can access only your favourite list, not anyone else. If the book ID does not exist then it returns HTTP 404 code.

Argument-

It takes only two values as an argument book ID and user ID and returns the response.

Method-

Method used in this API is HTTP GET method

User login-

To add the favourite list by user ID and book ID user have to login the userID and password

Example-

http://127.0.0.1:5000/add_favourite/123/demo@gmail.com

remove_favourite(user_id,book_id) -

This API is used to update the user's favourite list. and return the response . It returns a status message that list is updated successfully or not. The user only can access only your favourite list, not anyone else. If the book ID does not exist in the user favourite list it returns HTTP code 409.

Argument-

It takes only two values as an argument book ID and user ID and returns the response.

Method-

Method used in this API is HTTP GET method

User login-

To update the favourite list by user ID and book ID user have to login the userID and password

Example-

http://127.0.0.1:5000/remove_favourite/123/demo@gmail.com

get_books(filter,start_page,page_size) -

This API is used to filter the books according to the filter condition and return the response as a list of books. It skips the n page according to the values of start_page and page_size. Divide the rest filters books in the given page size called list and list size can’t exceed the page size.

Argument-

It takes only three values as an argument filter condition and start_page and page_size and returns the response.

Method-

Method used in this API is HTTP POST method

User login-

To get books by filter condition the user doesn't need to login the userID and password.

Example-

http://127.0.0.1:5000/get_books

add_books(book_id,params) -

This API is used to add the book on a book data table and return the response. Its second argument must be in json format otherwise it will generate HTTP 422 error code. If the book ID already exists it will 409 HTTP code and the book is successfully inserted it will respond with HTTP 200 code.

Argument-

It takes only two values as an argument book ID and params and param must be in json format.

Method-

Method used in this API is HTTP POST method.

User login-

To add book user have to login with user ID and password otherwise it will response with HTTP 401 code.

Example-

http://127.0.0.1:5000/add_book

update_books(book_id,params) -

This API is used to update the book on a book data table and return the response. Its second argument must be in json format otherwise it will generate HTTP 422 error code. If the book ID already exists it will 409 HTTP code and the book is successfully inserted it will respond with HTTP 200 code.

Argument-

It takes only two values as an argument book ID and params and param must be in json format.

Method-

Method used in this API is the HTTP POST method.

User login-

To update books users have to login with user ID and password otherwise it will respond with HTTP 401 code.

Example-

http://127.0.0.1:5000/update_book

Code sample

```
from flask import *
import jwt
import datetime
from functools import wraps
import boto3
from boto3.dynamodb.conditions import Key,Attr

app = Flask(__name__) #creating the Flask class object
app.config['SECRET_KEY']='TheSecretKeyCanBeAnyThingAsYouWis
h'


def vaild_token():
    try:
        if session['uname']:
            # token=session['token']
            try:
                token=session['token']

data=jwt.decode(token,app.config['SECRET_KEY'],algorithms=['HS25
6'])
            except Exception as e:
                session.pop('uname')
                session.pop('userID')
    except:
        pass


def token_required(f):
    @wraps(f)
    def decorated(*args,**kwargs):

        if 'token' not in session:
            return render_template('login.html',data={'error_msg':"Invalid
Username or password"}),401
        token=session['token']
        try:

data=jwt.decode(token,app.config['SECRET_KEY'],algorithms=['HS25
6'])
        except Exception as e:
            # print(e,token)
            # return jsonify({'response:'token is invalid'}),403
            # session.pop('uname')
            # print('pop out/N fire')
            return render_template('login.html',data={'error_msg':"Invalid
Username or password"}),401
        return f(*args,**kwargs)

    return decorated


############# Form ##################
@app.route('/login_form')
def login_form():
    return render_template('login.html')

@app.route('/add_book_form')
def add_book_form():
    return render_template('add_book_form.html')
```

```
@app.route('/update_book_form')
def update_book_form():

    return render_template('update_book_form.html')

@app.route('/filter_form')
def filter_form():
    return render_template('filter_form.html')

################# FORM #################







@app.route('/check',methods=['POST'])
def check(): #checking login Creadetial
    uname = request.form['uname']
    passwd = request.form['passwd']
    table_name='users'
    ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table(table_name)
    response =
ddb.query(KeyConditionExpression=Key('userID').eq(uname))
    response=response['Items']
    # print(response)
    if len(response)==0:
        return  jsonify({'response':'user ID or password is invalid.'}),401
    if passwd ==str(response[0]['password']):
        # print('if tree')

token=jwt.encode({'user':uname,'exp':datetime.datetime.utcnow()+dat
etime.timedelta(minutes=30)},app.config['SECRET_KEY'])
        # print(token)
        session['token']=token
        session['uname']=str(response[0]['full_name'])
        session['userID']=response[0]['userID']
        return redirect('/'),200
    return  jsonify({'response':'user ID or password is invalid.'}),401


@app.route('/')
def home():
    import boto3
    from boto3.dynamodb.conditions import Key,Attr
    table_name='data'
    ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table('data')
    response= ddb.scan(FilterExpression=Attr('title').begins_with("A") |
Attr('title').begins_with("Harry Potter"))
    data=response['Items']
    vaild_token()
    return render_template('home.html',data=data),200




@app.route('/get_books',methods=['GET','POST'])
def get_books():
    key=request.form['condition']
    value=request.form['value']
    s_page=request.form['start_page']
    page_size=request.form['page_size']
    table_name='data'
    ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table(table_name)
    if key=='title':
        # if len(key)<=2 or key.a
        response= ddb.scan(FilterExpression=Attr('title').contains(value))

        # print(response['Items'])

    elif key=='authors':
        response=
ddb.scan(FilterExpression=Attr('authors').contains(value))
        # print(response['Items'])


    elif key=='average_rating':
        response=
ddb.scan(FilterExpression=Attr('average_rating').begins_with(value))
        # print('yup')
        # print(response['Items'])


    elif key=='isbn':
        response= ddb.scan(FilterExpression=Attr('isbn').eq(value))
        # print(response['Items'])


    elif key=='language_code':
        response=
ddb.scan(FilterExpression=Attr('language_code').eq(value))
        # print(response['Items'])


    elif key=='rating_count':
        response=
ddb.scan(FilterExpression=Attr('ratings_count').eq(value))
        # print(response['Items'])




    if len(s_page)==0:s_page=0
    else:s_page=int(s_page)

    if len(page_size)==0:page_size=0
    else:page_size=int(page_size)

    if s_page>0 :skip=(s_page-1)*page_size
    else:skip=0
    count=0

    if s_page==0 and page_size==0 or page_size==0:
        # print('gttt')
        main_dict={}
        l1=[]
        for item in response['Items']:

d1={"bookID":str(item['bookID']),"title":str(item['title']),"authors":str(item[
'authors']),"average_rating":str(item['average_rating']),"isbn":str(item['is
bn']),"language_code":str(item['language_code']),"ratings_count":str(it
em['ratings_count']),"price":str(item['price'])}
            l1.append(d1)
        main_dict[1]=l1


    else:
        main_dict={}
        current_size=0
        l1=[]
        l1_count=0
        for item in response['Items']:
            count+=1
            if count>=skip:
                if current_size==page_size:
                    # print(l1)
                    l1_count+=1
                    main_dict[l1_count]=l1
                    # print(l1_count)
                    current_size=0
                    l1=[]

d1={"bookID":str(item['bookID']),"title":str(item['title']),"authors":str(item[
'authors']),"average_rating":str(item['average_rating']),"isbn":str(item['is
bn']),"language_code":str(item['language_code']),"ratings_count":str(it
em['ratings_count']),"price":str(item['price'])}

                l1.append(d1)
                current_size+=1
        if len(l1)>0:main_dict[l1_count+1]=l1
    try:
        return jsonify({'response':main_dict}),200
    except Exception as e:
        return jsonify({'response':e}),500


@app.route('/get_book/<string:bookid>',methods=["GET", "POST"])
def getbook(bookid):
    data={}
    try:
        table_name='data'
        ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table(table_name)
        response =
ddb.query(KeyConditionExpression=Key('bookID').eq(bookid))
        if len(response['Items'])==0:
            return jsonify({'response':'Book is not exist.'}),404
        data=response['Items']
    except Exception as e:
        return jsonify({'response':e}),500
    return jsonify({'response':data}),200




@app.route('/get_favourite/<string:userid>')
@token_required
def get_favourite(userid):
    if session['userID']!=userid:
        return jsonify({'response':f'You can access only your favourites
list not other one and your user ID is {session["userID"]}'})
    try:
        table_name='users'
        ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table(table_name)
        response =
ddb.query(KeyConditionExpression=Key('userID').eq(userid))
        fav_list=response['Items'][0]['favourites']

        table_name='data'
        ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table(table_name)
        data=[]
        if len(fav_list)==0 or len(fav_list[0])==0:
            return jsonify({'response': 'No books in your favourite list'}),200
        for id in fav_list:
            print(id)
            id=str(id)
            response =
ddb.query(KeyConditionExpression=Key('bookID').eq(id))
            response=response['Items'][0]
            # print(response)
            data.append(response)
    except Exception as e:
        return jsonify({'response': e}),500
    return jsonify({'response': data}),200




@app.route('/add_favourite/<string:bookID>/<string:userID>')
@token_required
def add_favourite(bookID,userID):
    if session['userID']!=userID:
        return jsonify({'response':f'You can add only your favourites list
not other one and your user ID is {session["userID"]}'})

    try:
        userID="".join(userID.split())
        bookID=str(bookID)
        ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table('data')
        response =
ddb.query(KeyConditionExpression=Key('bookID').eq(bookID))
        if len(response['Items'])==0:
            return jsonify({'response': 'Book is not exist'}),404

        ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table('users')
        response =
ddb.query(KeyConditionExpression=Key('userID').eq(userID))
        if len(response['Items'])!=0:
            fav_list=response['Items'][0]['favourites']
            if bookID in fav_list:
                return jsonify({'response': 'Already in your wish list'}),200
            fav_list.append(bookID)

        key='favourites'
        value=fav_list
        ddb.update_item(Key={'userID':userID},UpdateExpression=f'SET
{key} = :val1',ExpressionAttributeValues={':val1': value})


    except Exception as e:
        return jsonify({'response': e}),402
    return jsonify({'response': 'Successfully added.'}),200




@app.route('/remove_favourite/<string:bookID>/<string:userID>')
@token_required
def remove_favourite(bookID,userID):
    table_name='users'
    userID="".join(userID.split())
    if session['userID']!=userID:
        return jsonify({'response':f'You can remove only from your
favourites list not other one and your user ID is {session["userID"]}'})

    ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table(table_name)
    response =
ddb.query(KeyConditionExpression=Key('userID').eq(userID))


    if len(response['Items'])!=0:
        fav_list=response['Items'][0]['favourites']
        if bookID not in fav_list:
            return jsonify({'response': 'Already NOT in your wish list'}),409

        fav_list.remove(bookID)
        value=fav_list
        key='favourites'
        try:
```

```
ddb.update_item(Key={'userID':userID},UpdateExpression=f'SET
{key} = :val1',ExpressionAttributeValues={':val1': value})
          #
ddb.update_item(key={'userID':userID},UpdateExpression='SET
favourites = :val1',ExpressionAttributeValues={':val1': fav_list})

       except Exception as e:
          return jsonify({'response': e}),500

    return jsonify({'response': 'Successfully Removed.'}),200



@app.route('/add_book',methods=['POST']) #decorator drfines the
@token_required
def add_book():
    bookid=request.form['bookid']
    params=request.form['params']

    table_name='data'
    ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table(table_name)
    response =
ddb.query(KeyConditionExpression=Key('bookID').eq(bookid))

    if len(response['Items'])!=0:
       return jsonify({'response':bookid+' Already Exsist.'}),409

    ######## checking the JSON validation #####
    if len(params)==0:
       # print('yes')
```

```
        params='{"bookID":"'+bookid+'"}'
        # print(params)
        ddb.put_item(Item=json.loads(params))

        return jsonify({'response':bookid+' Successfully inserted.'}),201


    try:
      json.loads(params)
    except:
        return  jsonify({'response':' Invalid Json Syantax'}),422

    params='{"bookID":"'+bookid+'",'+params[1:]
    ddb.put_item(Item=json.loads(params))

    return jsonify({'response':bookid+' Successfully inserted.'}),201

@app.route('/update_book',methods=['POST']) #decorator drfines the
@token_required
def update_book():
    bookid=request.form['bookid']
    params=request.form['params']

    table_name='data'
    ddb = boto3.resource('dynamodb',
endpoint_url='http://localhost:8000').Table(table_name)
    response =
ddb.query(KeyConditionExpression=Key('bookID').eq(bookid))

    if len(response['Items'])==0:
        return jsonify({'response':bookid+'not exsist.'}),409
    if len(params)==0:return jsonify({'response':'book ID can not change
params is empty'}),400


    try:
        print(params)
        params=json.loads(params)
        print(params)
    except:
        return  jsonify({'response':' Invalid Json Syantax'}),422

    for key,value in params.items():

ddb.update_item(Key={'bookID':bookid},UpdateExpression=f'SET
{key} = :val1',ExpressionAttributeValues={':val1': value})


    return jsonify({'response':bookid+' Successfully Updated.'}),201

if __name__ =='__main__':
    app.run(debug = True)
```