LibraryAPI  
DOCUMENTATION  
by Raghuraamm

This API aims to connect a user online to a DB(SQL-Only) on server using several packages in JAVA.  
The DB keeps track of users, the books that are available and the borrowing by each user.   
 **Future Expansions:**  
***1***.to implement stock availability <Planning to do that but failed due to time constraint>  
***2.***to add dates to borrowing to improve and manage the time and have a back-end response using assertions to notify users of the expiry in borrowing.  
***3.***to add more redundancy checks in API  
***4.***to improve on testing of the API

More information on future implementation:  
1. The stock implementation will work by updating stock when users borrow the specific book. This version does not have stock implemented hence has no effect on borrowing

**SQL Specifications:**  
This API requires 3 tables to be designed in SQL  
Users Table: <Use JDBC compatible SQL>

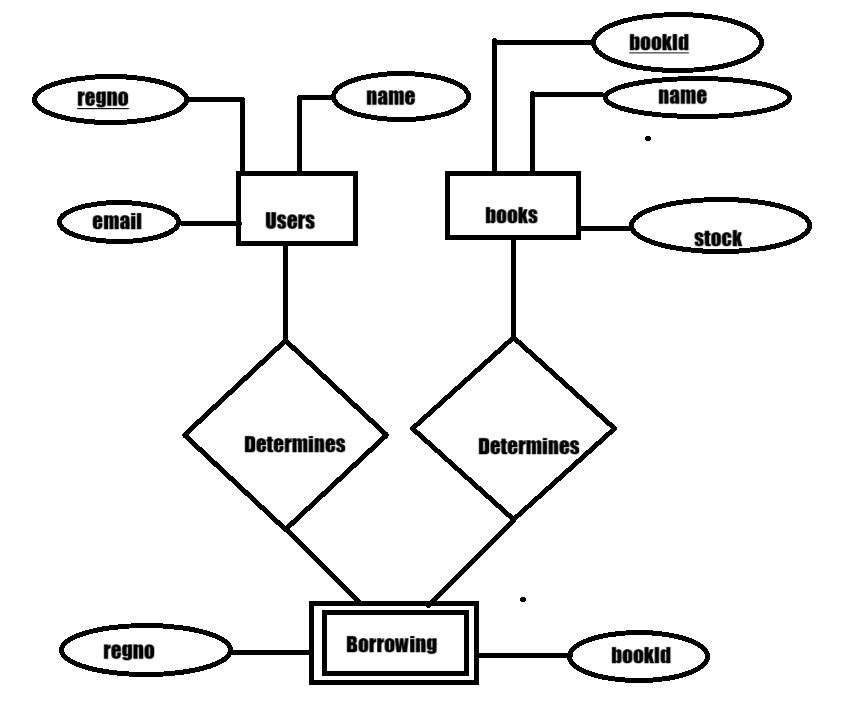
|  |  |  |
| --- | --- | --- |
| regno(int) | name(varchar) | email(varchar) |

books Table:

|  |  |  |
| --- | --- | --- |
| bookId(int) | name(varchar) | stock(varchar) |

borrowings Table:

|  |  |
| --- | --- |
| regno(int) | bookId(int) |

Redundancy is not managed in API so may cause   
spurious output  
Please set regno as primary key in Users table.  
Please set bookId as primary key in books table.  
Please set regno and bookId as foreign key in reference to  
Users Table and books table respectively.  
 **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*IMPORTANT\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Fill in Your SQL Details in the application properties  
\*The internal\_logon is for admin login for db can be removed if not necessary\*.  
  
  
  
ER diagram:  
  
  
SQL-Environment recreation code:  
CREATE TABLE Users ( regno INT PRIMARY KEY, name VARCHAR(100) NOT NULL, email VARCHAR(100) NOT NULL );**

**CREATE TABLE Books (bookid INT PRIMARY KEY, -- Book ID as the primary key name VARCHAR (100) NOT NULL, stock INT NOT NULL -- Number of books in stock); Borrowings table**

**CREATE TABLE Borrowings (regno INT, bookid INT, PRIMARY KEY (regno, bookid), CONSTRAINT fk\_user FOREIGN KEY (regno) REFERENCES Users(regno), CONSTRAINT fk\_book FOREIGN KEY (bookid) REFERENCES Books(bookid) );  
API Specifications:**This API uses SpringBoot as its Standard for RESTFUL implementation**.**The API has all the CRUD implementation using Mappings from SpringBoot   
**\*The context of the API is /libraryapi\***

**USER HANDLING:**

**For handling User registration to library:** Use curl Post **/user** and give the body as a Json   
in this format:  
{

        "regno": example-regno,

        "name": " example-name ",

        "email": "example-email"

}  
when successful returns inserted tuple.  
failure is managed by 400 bad request

**For handling User retrieval to library:** Use curl get **/user?regno=example-regno**   
when successful returns requested tuple.  
failure is managed by 404 error  
  
**For handling Bulk User retrieval (ALL) to library:**

Use curl get **/all**   
when successful returns requested tuples.  
failure is managed by 404 error

**For handling User update to library:** Use curl Put **/user?regno=example-regno** and give the body as a Json   
in this format:  
{

        "regno": example-updated-regno,

        "name": " example-updated-name ",

        "email": "example-updated-email"

}

when successful returns updated tuple.  
failure is managed by 400 bad request

**For handling User Deletion to library:** Use curl Put **/user?regno=example-regno**   
 when successful returns the Deleted tuple when successful  
failure is managed with 404 error  
  
  
   
   
  
  
  
  
**Books HANDLING:  
  
For handling Books registration to library:** Use curl Post **/book** and give the body as a Json   
in this format:  
{

        "bookId": example-bookId,

        "name": " example-name",

        "stock": "example-stock"

}  
when successful returns inserted tuple.  
failure is managed by 400 bad request

**For handling Book retrieval to library:** Use curl get **/book?bookId=example-bookId**  
when successful returns requested tuple.  
failure is managed by 404 error  
  
**For handling Bulk Book retrieval(ALL) to library:**

Use curl get **/books**  
when successful returns requested tuples.  
failure is managed by 404 error

**For handling Book update to library:** Use curl Put **/book?bookId=example-bookId** and give the body as a Json   
in this format:  
{

        "bookId": example-bookId,

        "name": " example-name",

        "stock": "example-stock"

}

when successful returns updated tuple.  
failure is managed by 400 bad request

**For handling Book Deletion to library:** Use curl Delete **/book?bookId=example-bookId**  
 when successful returns the Deleted tuple when successful  
failure is managed with 404 error  
  
**Borrowings HANDLING:  
For handling Borrowings registration to library:** Use curl Post **/borrowing** and give the body as a Json   
in this format:  
{  
 “regno”:example-regno,

        "bookId": example-bookId  
}  
when successful returns inserted tuple.  
failure is managed by 400 bad request

**For handling Book retrieval to library:** Use curl get **/ borrowing?regno=example-regno**  
when successful returns requested Hashmap  
with user details as key and list of borrowed books as values.  
failure is managed by 404 error  
  
**For handling Bulk Book retrieval(ALL) to library:**

Use curl get **/ borrowing/all**  
when successful returns requested tuples all user details encapsulated.  
failure is managed by 404 error

**For handling Borrowing update to library:** Use curl Put **/ borrowing?regno=example-regno** and give the body as a Json   
in this format:  
{  
 “regno”:example-updated-regno,

        "bookId": example-updated -bookId  
}  
when successful returns updated tuple.  
failure is managed by 400 bad request

**For handling Book Deletion to library:** Use curl Delete **/ borrowing?regno=example-regno**  
 when successful returns the Deleted tuple when successful  
failure is managed with 404 error