Designing a REST API (catalogue of a library)

Functional requirements:

- Allow the creation of new users.
- Allow the librarian to remove old users.
- Allow the user to search for books based on title, publication date, author, etc., and find their location in the library.
- Users can request, reserve a book.
- Librarian can add and manage the books.
- The system should notify the user and librarian about the overdue books.
- The system calculates the fine for overdue books on their return.

Non- Functional requirements:

- Usability is the main non-functional requirement for a library management system. The UI should be simple enough for everyone to understand and get the relevant information without any special training.
- Accuracy is another important non-functional requirement for the library management system. The data stored about the books and the fines calculated should be correct, consistent, and reliable.
- The System should be available for the duration when the library operates and must be recovered within an hour or less if it fails. The system should respond to the requests within two seconds or less.
- The software should be easily maintainable and adding new features and making changes to the software must be as simple as possible.

Use cases

- For the user to get registered, registration forms are available that is needed to be fulfilled by the user.
- After getting the registered, a new book can be requested by the user as per there requirement.
- The user can search for a book using the web page or application filtering by name, author, language etc.
- When the user had found the book, he will receive the position of the book in the library and a request to reserve the book until a certain date specify by the rules of the library.

- If the user somehow forgets to return the book before the due date, then the user pays fine. Or if the user forgets to renew the book till the due date, then the book will be overdue and the user pays fine.
- Librarian has a key role in this system. Librarian deletes the record of a particular student if the student leaves the college or passed out from the college. If the book no longer exists in the library, then the record of the particular book is also deleted.
- adding the books database is another important role of Librarian.

USER:

*** Creating a user**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/user.

* URI Design - http://library.execute-api.com/user

```
{
  "id": 1005,
  "firstName": "John",
  "lastName": "James",
  "email": "john@email.com",
  "password": "12345",
  "phone": "123458354",
}
```

…

* Content type - `application/json`

* HTTP method – POST

* HTTP status code - `200` - `OK`, `405`-invalid input, 503-service not available

*** User log in**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/user/login

* URI Design - http://library.execute-api.com/user/login

```
{
 "email":  "john@email.com",
 "password": "12345"
}
```

* Content type - `application/json`

* HTTP method – GET

* HTTP status code - `200` - `OK`, `400`-Invalid username/password supplied

*** User log out**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/user/logout

* URI Design - http://library.execute-api.com/user/logout?userId={userId}

* Content type - `application/json`

* Request parameters: userId(int)- unique Id to log out.

* HTTP method – GET

* HTTP status code - `200` - `OK`.

*** Updating user**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/user.

* URI Design - http://library.execute-api.com/user/{userId}

```
{
 "firstName": "John",
```

"lastName": "James",

"email": "john@email.com",

"password": "12345",

"phone": "31453879",

}

* Content type - `application/json`

* HTTP method – PUT

* HTTP status code - `200` - `OK`, 400-Invalid user supplied, 404-user not found.

*** Deleting a user**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/admin/user

* URI Design - http://library.execute-api.com/admin/user/delete/{userId}

{

"id": 1005,

"email": "john@email.com".

}

* Content type - `application/json`

* HTTP method – DELETE

* HTTP header- Authorization: {token}

* HTTP status code – 204-no content, 400-Invalid username supplied, 404-user not found, 401-Unauthorized

-Authentication:Basic {base64 encoded string username:password}

*Example: Authorization: Basic ZGVtbzpwQDU1dzByZA==

BOOK:

*** Creating a book**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/admin/books/add

* URI Design - http://library.execute-api.com/ admin/books/add

{

"id": 10985,

"Name": "what is hiding in the deep?",

"Author": "Linda Stevenson",

"Genre": "adventure, Fantasy",

"Description": "stan is a kid who is always dreaming about the things in the deep sea until one he had a met which changed his life...",

"Idiom": "English",

"Release Date": "12/04/1956",

"Position": "A-124".

“status”:” available”

}

…..

* Content type - `application/json`

* HTTP method – POST

* HTTP header- Authorization: {token}

* HTTP status code - `200` - `OK`, `405``-invalid input, 503-service not available, 401-Unauthorized

-Authentication:Basic {base64 encoded string username:password}

*Example: Authorization: Basic ZGVtbzpwQDU1dzByZA==

-**Note for caching**: it is not necessary to put the cache since the information needs to be returned to the user with the information in real time, otherwise it could generate errors when using the services, for example a user is going to reserve a book that has already been reserved.

*** Deleting a book**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/admin/books/delete

* URI Design - http://library.execute-api.com/admin/books/delete/{bookId}/

{

"id": 10985

}

* Content type - `application/json`

* HTTP method – DELETE

* HTTP header- Authorization: {token}

* HTTP status code - 204, 400-Invalid user supplied, 404-user not found, 401-Unauthorized

-Authentication:Basic {base64 encoded string username:password}

*Example: Authorization: Basic ZGVtbzpwQDU1dzByZA==

-**Note for caching**: it is not necessary to put the cache since the information needs to be returned to the user with the information in real time, otherwise it could generate errors when using the services, for example a user is going to reserve a book that has already been reserved.

*** Updating a book**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/admin/books

* URI Design - http://library.execute-api.com/admin/books /{bookId}

{

"Name": "what is hiding in the deep?",

"Author": "Linda Stevenson",

"Genre": "adventure, Fantasy",

"Description": "stan is a kid who is always dreaming about the things in the deep sea until one he had a met which changed his life...",

"Idiom": "English",

"Release Date": "12/04/1956",

"Position": "C-134".

“status”:” reserved”

}

* Content type - `application/json`

* HTTP method – PUT

* HTTP header- Authorization: {token}

* HTTP status code - `200` - `OK`, 400-Invalid user supplied, 405-invalid input, 401-Unauthorized

-Authentication:Basic {base64 encoded string username:password}

*Example: Authorization: Basic ZGVtbzpwQDU1dzByZA==

-**Note for caching**: it is not necessary to put the cache since the information needs to be returned to the user with the information in real time, otherwise it could generate errors when using the services, for example a user is going to reserve a book that has already been reserved.

*** Search for a book**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/user/books/findby

* URI Design - http://library.execute-api.com/user/books/findby?limit=<limit>.

```
[
 {
   "filters": [
    {
{
 "Author": "Linda Stevenson",
 "Genre": "adventure, Fantasy",
 "Idiom": "English",
 "Release Date": "12/04/1956",
    }
   ],
"response": [
{
 "id": 10985,
 "Name": "what is hiding in the deep?",
 "Author": "Linda Stevenson",
 "Genre": "adventure, Fantasy",
 "Description": "stan is a kid who is always dreaming about the things in the deep sea until one he had a met which changed his life...",
 "Idiom": "English",
 "Release Date": "12/04/1956",
   "position": "c-132"
    "status": "available"
 },
{
 "id": 10904,
 "Name": "A fairy in the woods",
 "Author": "Linda Stevenson",
```

"Genre": "adventure, Fantasy",

"Description": two brothers will change their lives when one day while playing in the forest found a fairy and since that day…"

"Idiom": "English",

"Release Date": "12/04/1956",

"position": "c-145"

"status": "reserved"

}

……..

"links": [

{

"rel": "borrow",

"href": - http://library.execute-api.com/user/books/borrow,

"method": "POST"

},

{

"rel": "return",

"href": http://library.execute-api.com/user/books/return,

"method": "POST"

},

]

}

}

* Content type - `application/json`

* HTTP method – GET

* HTTP status code - `200` - `OK`, 405-invalid input, 503-service not available.

-**Note for caching**: it is not necessary to put the cache since the information needs to be returned to the user with the information in real time, otherwise it could generate errors when using the services, for example a user is going to reserve a book that has already been reserved.

*** Borrowing a book**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/user/books/borrow

* URI Design - http://library.execute-api.com/user/books/borrow?limit=<limit>.

{

“id”: 10904,

"Borrowing days": 7

}

* Content type - `application/json`

* HTTP method – POST

* HTTP status code - `200` - `OK`, `405`-invalid input, 503-service not available

*** Returning a book**

* Protocol - `HTTP`

* End Point - http://library.execute-api.com/user/books/return

* URI Design - http://library.execute-api.com/user/books/return/{bookId}

* Content type - `application/json`

*Response:

{

“Total to pay”: 0$

}

* HTTP method – POST

* HTTP status code - `200` - `OK`, 404-book not found,503-service not available.

-**Note for caching**: it is not necessary to put the cache since the information needs to be returned to the user with the information in real time, otherwise it could generate errors when using the services, for example a user is going to reserve a book that has already been reserved.