Library Books

By Eduardo M. Ramírez Blauvelt

FUNCTION REQUIREMENTS

- The library catalogue should be able to connect have the possibility to read the database where all the books are stored, should be able to accept new entries as new books are acquired by the library and should allow for book deletion from the catalogue if something happens to a book that makes it permanently unavailable to the public.

- The database should store information from the books (name, author, publisher, edition and other relevant data) and can be searched for any of the above purposes based on the ISBN, name, author and category.

- To modify the database, the user requires a special login that gives privileged access, these modifications generate a log.

- Registered users can log in to the catalogue with their username and password and the can search the entire catalogue of books.

- List all books

- update

NON-FUNCTIONAL REQUIREMENTS

- The catalogue should be available 24/7 through the internet (in library search would be also on the web)

USE CASES

Title: Search for a book or books
Actor: Registered user

Scenario: The user accesses the catalogue looking for a book, in order to find it he can search by name, category or author. The search will return the book or books associated with the related search. Results will be paginated.

Title: Delete a book
Actor: Librarian with special access
Scenario: The librarian with special access logs in to the system and searches for a book that is going to be retired from general access, upon finding it and making sure it's the right one, he proceeds to delete it.

Title: Insert a book
Actor: Librarian with special access
Scenario: The librarian with special access logs in to the system and inserts the information about the new book (filling up all the required and hopefully all the non-required fields)

Title: update a book
Actor: Librarian with special access
Scenario: The librarian with special access logs in to the system and updates the information about the book (filling up all the required and hopefully all the non-required fields)

Title: Log in
Actor: Librarian with special access or registered user
Scenario: The librarian with special access or registered user should input its username and password to access the system and therefore modify the database.

LOG IN

- Protocol – HTTPS
- Endpoint – https://servername.com/login
- URI design – https://servername.com/login
- Representation
    - JSON

Request
{"username": "name",
"password": "1234567890"}

// Header
{
"alg": "HS256",
"typ": "JWT"
}

// Payload
{
"sub": "1234567890",
"name": "name",
"iat": 1516239022
}

HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),

your-256-bit-secret

)

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Im5hbWUiLCJpYXQiOjE1MTYyMzkwMjJ9.3ZEDmhWNZWDbJDPDlZX_I3oaalNYXdoT-bKLxIxQK4U

Response
{“session id”: “XXXX”,
“status”: “XXX”}

- Content type – application/json
- Content type – application/xml
- HTTP Methods – POST
- HTTP Status code – 200 (OK), 400 (Bad Request), 404 (User name or Password is invalid)

INSERTING A BOOK

- Protocol – HTTPS
- Endpoint – https://servername.com/catalogue
- URI design – https://servername.com/catalogue
- Representation
    - JSON
      Request
      {“ISBN”: “XXX”,
      “name”: “XXX”,
      “author”: “XXX”,
      “category”: “XXX”,
      “ publisher”: “XXX”}

      Response

{“id”: “XXX”,
“ISBN”: “XXX”,
“name”: “XXX”,
“author”: “XXX”,
“category”: “XXX”,
“ publisher”: “XXX”}

- XML

<book>
<id = XXX / >
<ISBN = XXX />
<name = XXX / >
<author = XXX />
<category = XXX / >
<publisher = XXX / >
</book>

- Content type – application/json
- Content type – application/xml
- HTTP Methods – POST
- HTTP Status code – 201 (OK), 400 (Bad request), 409 (Book already exists), 401 (Not authorized, please log in), 403 (User not allowed to perform operation)

DELETING A BOOK

- Protocol – HTTPS
- Endpoint – https://servername.com/catalogue
- URI design – https://servername.com/catalogue/{id}
- Content type – application/json
- path variable – int -> book id
- HTTP Methods – DELETE
- HTTP Status code – 204 (OK), 404 (Book not found), 401 (Not authorized, please log in), 403 (User not allowed to perform operation)

UPDATING A BOOK

- Protocol – HTTPS
- Endpoint – https://servername.com/catalogue
- URI design – https://servername.com/catalogue/{id}
- Representation
  - JSON
    Request
    {“ISBN”: “XXX”,

“name”: “XXX”,
“author”: “XXX”,
“category”: “XXX”,
“ publisher”: “XXX”}

Response
{“id”: “XXX”,
“ISBN”: “XXX”,
“name”: “XXX”,
“author”: “XXX”,
“category”: “XXX,
“publisher”: “XXX”}

- XML
  <book>
  <id = XXX / >
  <ISBN = XXX />
  <name = XXX / >
  <author = XXX />
  <category = XXX / >
  <publisher = XXX / >
  </book>

- Content type – application/json
- path variable – int -> book id
- HTTP Methods – PUT
- HTTP Status code – 200 (OK), 404 (Book not found), 401 (Not authorized, please log in), 403 (User not allowed to perform operation)

SEARCHING FOR A BOOK

- Protocol – HTTP
- Endpoint – http://servername.com/catalogue
- URI design – http://servername.com/catalogue?page=<page>&size=<size>&sort=<asc>&isbn=<isbn>&name=<name>&author=<author>
- Representation
  - JSON
    Response
    [{“id”: “XXX”,
    “ISBN”: “XXX”,
    “name”: “XXX”,

"author": "XXX",
"category": "XXX",
"publisher": "XXX",
"url": http://servername.com/catalogue/XXX}]

- XML

<books>
<book>
<id = XXX / >
<ISBN = XXX />
<name = XXX / >
<author = XXX />
<category = XXX / >
<publisher = XXX / >
<link url = http://servername.com/catalogue/XXX />
</book>
</books>

- Content type – application/json
- Query parameter – page int, size int, sort string, ISBN long, name string, author string
- HTTP Methods – GET
- HTTP Status code – 200 (OK), 404 (search returned no results), 401 (Not authorized, please log in), 403 (User not allowed to perform operation)
- Caching: Keep the latest books or last five searches from user