Portoflio Project: API Spec

CS 493: Cloud Application Development

Spring 2023

Oregon State University

Project URL: https://cs493-portfolio-f.ue.r.appspot.com

Change log

Version	Change	Date
1.0	Initial version.	Oct 6, 2022
1.1	Portfolio version.	June 8, 2023

Data Model

The app stores three kinds of entities in Datastore, Users, Libraries, and Books.

User

Property	Data Type	Notes
Id	Int	Generated by datastore.
Name	String	Name of the User
Jwt_Id	String	JWT ID String

The id is generated by datastore and can be used to look or get that value, but the payload JWT ‘sub’ is saved as a unique identifier along with name to keep track of the users in addition to this id for datastore. The server will take care of this when decoding the login attempt for viewing in Postman. In addition, the user is considered an owner for both books and libraries. They are assigned by the server, thus making authentication required for most endpoints.

Library

Property	Data Type	Notes
id	Integer	The id of the Library. Datastore automatically generates it. Don't add it yourself as a property of the entity.
name	String	Name of the Library. Required.
description	String	Description of Library or its contents. Required.
categories	String	Categories for libraries listed in a String. Required.
public	Boolean	If the Library should be accessible private only or publicly. Required.
owner	String	JWT ID String (Sub). Assigned automatically via the Authorization Token in the request.
Books	List	List of Entities that are books assigned to the library. Generated by the server.
Self	String	URL address to the item in question. Generated automatically by the server.

Books

Property	Data Type	Notes
id	Integer	The id of the Book. Datastore automatically generates it. Don't add it yourself as a property of the entity.
name	String	The name of the book. Required.
Author	String	The author(s) of the book. Required.
isbn	Integer	The ISBN number for the book. Required.
public	Boolean	If the Book should be accessible privately or publicly. Required.
owner	String	JWT ID String (Sub). Assigned automatically bia the Authorization Token by the server. Do not assign this.
library	Entity	Library entity that the book is assigned to. A single book can only be assigned to one library. Server will generate this.
self	String	URL address to the book item in question. Generated automatically by the server.

Relationship

A book can be placed in a Library, and a Library contains books. Book entities are placed in a list within a Library, and the Library id is saved in the Book entity to show that it belongs to that library.

*Note – All requests with a response that returns data must accept application/json

Create a Library

Allows you to create a new library.

POST /libraries

Request

Path Parameters

None

Request Headers

Authorization: JWT Token after login

Required as authorization for the creation of the library, and to be able to assign an owner to the library.

Request Body

Required

Request Body Format

JSON

Request JSON Attributes

Name	Description	Required?
name	Name of the library.	Yes
description	Description for the library.	Yes
categories	Categories associated with library. String, comma delimited.	Yes
public	True/False. If True, will be publicly accessible.	Yes

Request Body Example

```
{
   "name": "library1",
   "description": "test library 1",
   "categories": "tests",
   "public": true
}
```

Response

Response Body Format

JSON

Response Statuses

Outcome	Status Code	Notes
Success	201 Created	
Failure	400 Bad Request	
Failure	406 Not Acceptable	Client must accept application/json

Response Examples

Success

Status: 201 Created (Extra attributes are added by the server)


```
{
   "books": [],
   "categories": "tests",
   "description": "test library 1",
   "id": 5759318150348800,
   "name": "library1",
   "owner": "auth0|6483e81f7c26c2b64e037f4b",
   "public": true,
   "self": "https://127.0.0.1/libraries/5759318150348800"
}
```


Failure

Status: 400 Bad Request


```
{
"Error":  "The request object is missing at least one of the required
attributes"
}
```


Status: 406 Not Acceptable


```
{
   "Error": "Not acceptable"
}
```

Get a Library

Allows you to get an existing library

GET /libraries/:library_id

Request

Path Parameters

Name	Description
library_id	ID of the library

Request Headers

Authorization: Token (JWT)

Single library requests must always be authorized before they will work. Only lists of libraries are available to public non-authorized requests.

Request Body

None

Response

Response Body Format

JSON

Response Statuses

Outcome	Status Code	Notes
Success	200 OK	
Failure	404 Not Found	Not Found
Failure	406 Not Acceptable	Client must accept application/json

Response Examples

Success

```
Status: 200 OK

{
   "books": [],
   "categories": "tests",
   "description": "test library 1",
   "id": 5759318150348800,
   "name": "library1",
   "owner": "auth0|6483e81f7c26c2b64e037f4b",
   "public": true,
   "self": "https://127.0.0.1/libraries/5759318150348800"
}
```

Failure

Status: 404 Not Found

```
{
"Error":  "Not Found"
}
```

Status: 406 Not Acceptable

```
{
   "Error": "Not acceptable"
}
```

List all Libraries

List all the libraries.

GET /libraries

Request

Path Parameters

None

Request Headers

Authorization: JWT Token

Required for all non-public libraries. Excluding the Authorization header will only return the public Library entries. If there are non, it will return an empty list.

Request Body

None

Response

Response Body Format

JSON

Response Statuses

Outcome	Status Code	Notes
Success	200 OK	
Failure	406 Not Acceptable	Client must accept application/json

Response Examples

Success

Status: 200 OK (Example of a public request)


```
[
  {
    "books": [],
    "categories": "tests",
    "description": "test library 1",
    "id": 5655374346584064,
    "name": "library1",
    "owner": "auth0|648662c8ae870301b179774d",
    "public": true,
    "self": "https://127.0.0.1/libraries/5655374346584064"
  },
  {
    "books": [],
    "categories": "tests",
```

```
        "description": "test library 1",
        "id": 5718746756808704,
        "name": "library1",
        "owner": "auth0|648662c8ae870301b179774d",
        "public": true,
        "self": "https://127.0.0.1/libraries/5718746756808704"
    }
]
```


Status: 406 Not Acceptable


```
{
    "Error": "Not acceptable"
}
```

Edit a Library

Allows you to edit a library.

```
PATCH /libraries/:library_id
PUT /libraries/:library_id
```

Request

Path Parameters

Name	Description
library_id	ID of the library to edit

Request Body

Required

Request Body Format

JSON

Request JSON Attributes

Name	Description	Required?
name	Name of the library.	Yes (PUT)
description	Description for the library.	Yes (PUT)
categories	Categories associated with library. String, comma delimited.	Yes (PUT)
public	If a library is public or private.	YES (PUT)

*Note, a PATCH request requires one of these at a time, and will only update the given attribute. PUT requests will require all of them to be included.

Request Body Example

```
{
  "name": "library1",
  "description": "updated test library 1",
  "categories": "tests",
  "public": true
}
```

Response

Response Body Format

JSON

Response Statuses

Outcome	Status Code	Notes
Success	200 OK	
Failure	400 Bad Request	A PUT request must contain all the named categories above, but a patch only requires one of the four.
Failure	403 Forbidden	Must have valid token in Authorization header
Failure	404 Not Found	No library with given library_id exists.
Failure	406 Not Acceptable	Client must accept application/json

Response Examples

Success

Status: 200 OK


```
{
   "books": [],
   "categories": "tests",
   "description": "updated test library 1",
   "id": 5759318150348800,
   "name": "library1",
   "owner": "auth0|6483e81f7c26c2b64e037f4b",
   "public": true
}
```


Failure

Status: 400 Bad Request


```
{
"Error": "The request object is missing at least one of the required attributes"
}
```


Status: 403 Forbidden


```
{
    "Error": "Forbidden"
}
```

Status: 404 Not Found

```
{
    "Error": "Not Found"
}
```

Status: 406 Not Acceptable

```
{
   "Error": "Not acceptable"
}
```

Delete a Library

Allows you to delete a library. Deleting a library will remove all books from the library, but will still keep them as loose, unassigned books.

DELETE /libraries/:library_id

Request

Path Parameters

Name	Description
library_id	ID of the library

Request Header

Authorization: Token (JWT) after login required

Request Body

None

Response

No body

Response Body Format

Success: No body

Failure: JSON

Response Statuses

Outcome	Status Code	Notes
Success	204 No Content	
Failure	403 Forbidden	Must have valid token in Authorization header
Failure	404 Not Found	No library with the given library_id exists

Response Examples

Success

Status: 204 No Content

Failure

Status: 403 Forbidden

```
{
    "Error": "Forbidden"
}
```

Status: 404 Not Found


```
{
    "Error": "Not Found"
}
```

Create a Book

Allows you to create a new book.

```
POST /books
```

Request

Path Parameters

None

Request Headers

Authorization: Token (JWT)

Requires a token after login (JWT) in the Authorization header to create a book. The created book can be either public or private.

Request Body

Required

Request Body Format

JSON

Request JSON Attributes

Name	Description	Required?
name	The name of the book.	Yes
author	The author of the book.	Yes
isbn	The isbn of the book, must be int.	Yes
public	If the book is private or public, must be True or False	Yes

Request Body Example

```
{
  "name": "The Best Book",
  "author": "The Worst",
  "isbn": 1231254152,
  "public": true
}
```

Response

Response Body Format

JSON

Response Statuses

Outcome	Status Code	Notes
Success	201 Created	
Failure	400 Bad Request	
Failure	403 Forbidden	Must have valid token in Authorization header
Failure	404 Not Found	No book with given book_id exists.

Failure	406 Not Acceptable	Client must accept application/json

Success

Status: 201 Created


```
{
   "author": "The Worst",
   "id": 5647975057457152,
   "isbn": 1231254152,
   "library": null,
   "name": "The Best Book",
   "owner": "auth0|6483e81f7c26c2b64e037f4b",
   "public": true,
   "self": "https://127.0.0.1/books/5647975057457152"
}
```


Failure

Status: 400 Bad Request


```
{
    "Error": "The request object is missing at least one of the required
attributes"
}
```


Status: 403 Forbidden


```
{
    "Error": "Forbidden"
}
```


Status: 404 Not Found


```
{
    "Error": "Not Found"
}
```


Status: 406 Not Acceptable


```
{
   "Error": "Not acceptable"
}
```

Get a Book

Allows you to get an existing book.

GET /books/:book_id

Request

Path Parameters

Name	Description
book_id	ID of the book

Request Headers

Authorization: Token (JWT) required

Must be logged in to get a specific book, and have a valid token.

Request Body

None

Response

Response Body Format

JSON

Response Statuses

Outcome	Status Code	Notes
Success	200 OK	
Failure	404 Not Found	No book with this book_id exists

Response Examples

Success

Status: 200 OK


```
{
  "books": [],
  "categories": "tests",
  "description": "updated test library 1",
  "id": 5759318150348800,
  "name": "library1",
  "owner": "auth0|6483e81f7c26c2b64e037f4b",
  "public": true,
  "self": "https://127.0.0.1/libraries/5759318150348800"
}
```

Failure

Status: 404 Not Found

```
{
    "Error": "Not Found"
}
```

List all Books

List all the books. All for a user if given an authorization token, or all public books available in the database.

GET /books

Request

Path Parameters

None

Request Headers

Authorization: JWT Token (Optional)

Getting private books requires a token, but public books does not require a public token.

Request Body

None

Response

Response Body Format

JSON

Response Statuses

Outcome	Status Code	Notes
Success	200 OK	

Response Examples

Success

```
Status: 200 OK

[
  {
    "author": "The Worst",
    "id": 5647975057457152,
    "isbn": 1231254152,
    "library": null,
    "name": "The Best Book",
    "owner": "auth0|6483e81f7c26c2b64e037f4b",
    "public": true,
    "self": "https://127.0.0.1/books/5647975057457152"
  },
  {
    "author": "The Worst",
    "id": 5670706842959872,
    "isbn": 1231254152,
```

```
        "library": null,
        "name": "The Best Book",
        "owner": "auth0|6483e81f7c26c2b64e037f4b",
        "public": true,
        "self": "https://127.0.0.1/books/5670706842959872"
    }
]
```

Delete a Book

Allows you to delete a book. If the book is deleted, the library containing it will also have its book list edited to show the change.

DELETE /books/:book_id

Request

Path Parameters

Name	Description
book_id	ID of the book

Request Headers

Authorization: JWT Token after login required

Deleting both public and private tokens must be done by the owner of the book, and requires a valid token to allow for a book deletion.

Request Body

None

Response

No body

Response Body Format

Success: No body

Failure: JSON

Response Statuses

Outcome	Status Code	Notes
Success	204 No Content	
Failure	403 Forbidden	Must have valid Authorization header with token
Failure	404 Not Found	No book with this book_id exists

Response Examples

Success

Status: 204 No Content

Failure

Status: 403 Forbidden

```
{
    "Error": "Forbidden"
}
```

Status: 404 Not Found

```
{
    "Error": "Not Found"
}
```

Book Placed in Library

Book is placed in a library.

PUT /libraries/:library_id/:book_id

Request

Path Parameters

Name	Description
library_id	ID of the library
book_id	ID of the book

Request Headers

Authorization: JWT Token after login required

Request Body

None

Note: Set Content-Length to 0 in your request when calling out to this endpoint.

Response

No body

Response Body Format

Success: No body

Failure: JSON

Response Statuses

Outcome	Status Code	Notes
Success	204 No Content	Succeeds if both library and book from given ids exist, and are able to be added.
Failure	403 Forbidden	Must have authorization header with token
Failure	404 Not Found	The given library or book did not exist

Response Examples

Success

Status: 204 No Content

Failure

Status: 403 Forbidden

```
{
    "Error": "Forbidden"
}
```

Status: 404 Not Found

```
{
    "Error": "Not Found"
}
```

Book Removed from Library

The given book will be removed from the given library using their ids.

DELETE /books/:library_id/:book_id

Request

Path Parameters

Name	Description
library_id	ID of a library
book_id	ID of a book

Request Headers

Authorization: JWT Token after login required

Request Body

None

Response

No body

Response Body Format

Success: No body

Failure: JSON

Response Statuses

Outcome	Status Code	Notes
Success	204 No Content	Succeeds when book is found inside a library.
Failure	403 Forbidden	Must have authorization header with token
Failure	404 Not Found	No library or book found with given ids.

Response Examples

Success

Status: 204 No Content

Failure

Status: 403 Forbidden

```
{
    "Error": "Forbidden"
}
```

Status: 404 Not Found

```
{
    "Error": "Not Found"
}
```

List all Users

List all users

GET /users

Request

Path Parameters

None

Request Body

None

Response

Response Body Format

JSON

Response Statuses

Outcome	Status Code	Notes
Success	200 OK	
Failure	406 Not Acceptable	Must accept application/json

Response Examples

Success

Status: 200 OK


```
[
  {
    "id": 5658442790338560,
    "jwt_id": "auth0|648662c8ae870301b179774d",
    "name": "user2@dempsjam.com",
    "self": "https://127.0.0.1/users/5658442790338560"
  },
  {
    "id": 5721207605297152,
    "jwt_id": "auth0|6483e81f7c26c2b64e037f4b",
    "name": "user1@dempsjam.com",
    "self": "https://127.0.0.1/users/5721207605297152"
  }
]
```

Failure

```
Status: 406 Not Acceptable

{
   "Error": "Not acceptable"
}
```

Unmet Requirements

Pagination

I ran into major issues when trying to paginate. At first, I was manually implementing pagination with parameters passed in the url, but this often didn't work, and as far as I could tell, was likely due to the new pagination cursors that are preferred in datastore. If I managed to get a version working, I would integrate it into my get() method in the DatastoreDatabase.py file, where I place all the interactions with datastore (and data management).

When I tried pagination with the preferred method, I was having a difficult time keeping track of where I was for the required self link. Which would lead to issues when attempting to use the self link to get to the next part. With the cursor, I was also having issues actually picking the correct values to start from (and how many items to get from datastore). Every time I attempted, it would break the program for requests that didn't involve pagination.

Honestly, I think one of the major issues I had was over-complication. In an attempt to make an easier way to interact with datastore, I ended up increasing the complexity of those functions. When those functions were created without pagination in mind, it would require a lot more work to integrate that. So I would need to simplify everything in the future If I wanted to make pagination easier to get working. In particular, I was converting the Entities into dictionaries so I can more easily manipulate and display them as JSON. In most cases, I could refactor the code to exclude that need.

I also think I need to take some more time with understanding pagination in datastore. The other part of the issue was the documentation. I kept confusing myself with the current way to do it, and the previously deprecated examples they had available. I spent too long without determining what I would use before moving on, thus giving me too little time to make the needed changes to get everything to work.

Creation of New Users

The following are the users I created, also available in the provided postman environment, in addition to the decode function that will show the JWTs.

User1: email – user1@dempsjam.com password - User1Pass	User2: email – user2@dempsjam.com password - User2Pass