Designing a REST API

Topic (Catalogue (a library – books, authors, categories; etc.)

Entities

Books:

Attributes: ID, Title, Author(s), Category, Publication Year, ISBN, etc.

Operations:

Create a new book (POST /books)
Get a list of all books (GET /books)
Get information about a specific book (GET /books/{bookID})
Update information about a book (PUT /books/{bookID})
Delete a book (DELETE /books/{bookID})
Search books by author (GET /books?author={authorName})
Search books by category (GET /books?category={categoryName})
Pagination of results (GET /books?page={pageNumber}&limit={pageSize})

Authors:

Attributes: ID, First Name, Last Name, Date of Birth, Biography, etc.

Operations:

Create a new author (POST /authors)
Get a list of all authors (GET /authors)
Get information about a specific author (GET /authors/{authorID})
Update information about an author (PUT /authors/{authorID})
Delete an author (DELETE /authors/{authorID})
Search authors by name (GET /authors?name={authorName})
Pagination of results (GET /authors?page={pageNumber}&limit={pageSize})

Categories:

Attributes: ID, Name, Description, etc.

Operations:

Create a new category (POST /categories)
Get a list of all categories (GET /categories)
Get information about a specific category (GET /categories/{categoryID})
Update information about a category (PUT /categories/{categoryID})
Delete a category (DELETE /categories/{categoryID})
Search categories by name (GET /categories?name={categoryName})
Pagination of results (GET /categories?page={pageNumber}&limit={pageSize})

Publishers:

Attributes: ID, Name, Address, Contact Information, etc.

Operations:

Create a new publisher (POST /publishers)
Get a list of all publishers (GET /publishers)
Get information about a specific publisher (GET /publishers/{publisherID})
Update information about a publisher (PUT /publishers/{publisherID})

Delete a publisher (DELETE /publishers/{publisherID})
Search publishers by name (GET /publishers?name={publisherName})
Pagination of results (GET /publishers?page={pageNumber}&limit={pageSize})

Users:
Attributes: ID, Username, Email, Password (hashed), etc.
Operations:
Create a new user account (POST /users)
Get a list of all users (GET /users)
Get information about a specific user (GET /users/{userID})
Update user information (PUT /users/{userID})
Delete a user account (DELETE /users/{userID})
Search users by username or email (GET /users?query={usernameOrEmail})
Pagination of results (GET /users?page={pageNumber}&limit={pageSize})

REST API Description:

This design includes collections (e.g., books, authors, categories, publishers, users), filters for searching (by author, category, name), pagination support, and standard HTTP methods for CRUD operations. It adheres to the Richardson Maturity Model and includes meaningful status codes, error handling, authentication, and caching considerations.

Books

Create a Book:
Method: POST /books
Request Body: JSON with new book data
Response: Code 201 Created and data of the created book

```
// Example response for successful book creation
{
  "id": 123,
  "title": "Sample Book",
  "author": "Author Name",
  "category": "Sample Category",
  "year": 2022,
  "isbn": "1234567890"
}
```

Get a List of All Books:
Method: GET /books
Request Parameters: page, limit (optional)
Response: Code 200 OK and a list of books in JSON format

```
// Example response for getting a list of all books:
{
  "books": [
    {
      "id": 1,
      "title": "Book One",
      "author": "Author A",
      "category": "Category X",
      "year": 2020,
      "isbn": "1111111111"
    },
    {
      "id": 2,
      "title": "Book Two",
      "author": "Author B",
      "category": "Category Y",
      "year": 2021,
      "isbn": "2222222222"
    },
    // ... more books
  ]
}
```

Get Information about a Book:
Method: GET /books/{bookID}
Response: Code 200 OK and book data in JSON format

```
// Example response for getting information about a book:
{
  "id": 123,
  "title": "Примерная Книга",
  "author": "Имя Автора",
  "category": "Примерная Категория",
  "year": 2022,
  "isbn": "1234567890"
}
```

Update Information about a Book:
Method: PUT /books/{bookID}
Request Body: JSON with updated book data
Response: Code 200 OK and data about the updated book

```
// Example response for successfully updating book information:
{
  "id": 123,
  "title": "Updated Book Title",
  "author": "Updated Author Name",
  "category": "Updated Category",
  "year": 2022,
  "isbn": "1234567890"
}
```

Delete a Book:
Method: DELETE /books/{bookID}
Response: Code 204 No Content

```
// Example response for successful book deletion:
// (Status code 204 No Content)
```

Search Books by Author:
Method: GET /books?author={authorName}
Response: Code 200 OK and a list of books in JSON format

```
// Example response for searching books by author:
{
  "books": [
    {
      "id": 1,
      "title": "Book One",
      "author": "Author A",
      "category": "Category X",
      "year": 2020,
      "isbn": "1111111111"
    },
    {
      "id": 2,
      "title": "Book Two",
      "author": "Author A",
      "category": "Category Y",
      "year": 2021,
      "isbn": "2222222222"
    },
    // ... more books by the author
  ]
}
```

Search Books by Category:
Method: GET /books?category={categoryName}
Response: Code 200 OK and a list of books in JSON format

```
// Example response for searching books by category:
{
  "books": [
    {
      "id": 1,
      "title": "Book One",
      "author": "Author A",
      "category": "Category X",
      "year": 2020,
      "isbn": "1111111111"
    },
    {
      "id": 3,
      "title": "Book Three",
      "author": "Author B",
      "category": "Category X",
      "year": 2022,
      "isbn": "3333333333"
    },
    // ... more books in the category
  ]
}
```

Pagination of Results:

Use parameters page and limit in requests

```
// Example response for paginating results:
{
  "page": 1,
  "limit": 10,
  "totalPages": 5,
  "books": [
    // ... books for the first page
  ]
}
```

Authors

Create an Author:

Method: POST /authors

Request Body: JSON with new author data

Response: Code 201 Created and data of the created author

Get a List of All Authors:

Method: GET /authors

Request Parameters: page, limit (optional)

Response: Code 200 OK and a list of authors in JSON format

Get Information about an Author:
Method: GET /authors/{authorID}
Response: Code 200 OK and author data in JSON format

Update Information about an Author:
Method: PUT /authors/{authorID}
Request Body: JSON with updated author data
Response: Code 200 OK and data about the updated author

Delete an Author:
Method: DELETE /authors/{authorID}
Response: Code 204 No Content

Search Authors by Name:
Method: GET /authors?name={authorName}
Response: Code 200 OK and a list of authors in JSON format

Pagination of Results:
Use parameters page and limit in requests

Categories

Create a Category:
Method: POST /categories
Request Body: JSON with new category data
Response: Code 201 Created and data of the created category

Get a List of All Categories:
Method: GET /categories
Request Parameters: page, limit (optional)
Response: Code 200 OK and a list of categories in JSON format

Get Information about a Category:
Method: GET /categories/{categoryID}
Response: Code 200 OK and category data in JSON format

Update Information about a Category:
Method: PUT /categories/{categoryID}
Request Body: JSON with updated category data
Response: Code 200 OK and data about the updated category

Delete a Category:
Method: DELETE /categories/{categoryID}
Response: Code 204 No Content

Search Categories by Name:
Method: GET /categories?name={categoryName}
Response: Code 200 OK and a list of categories in JSON format

Pagination of Results:
Use parameters page and limit in requests

Users

Create a User:
Method: POST /users
Request Body: JSON with new user data
Response: Code 201 Created and data of the created user

Get a List of All Users:
Method: GET /users
Request Parameters: page, limit (optional)
Response: Code 200 OK and a list of users in JSON format

Get Information about a User:
Method: GET /users/{userID}
Response: Code 200 OK and user data in JSON format

Update User Information:
Method: PUT /users/{userID}
Request Body: JSON with updated user data
Response: Code 200 OK and data about the updated user

Delete a User:
Method: DELETE /users/{userID}
Response: Code 204 No Content

Search Users by Username or Email:
Method: GET /users?query={usernameOrEmail}
Response: Code 200 OK and a list of users in JSON format

Pagination of Results:
Use parameters page and limit in requests

Publishers

Create a Publisher:
Method: POST /publishers
Request Body: JSON with new publisher data
Response: Code 201 Created and data of the created publisher

Get a List of All Publishers:

Method: GET /publishers
Request Parameters: page, limit (optional)
Response: Code 200 OK and a list of publishers in JSON format

Get Information about a Publisher:
Method: GET /publishers/{publisherID}
Response: Code 200 OK and publisher data in JSON format

Update Information about a Publisher:
Method: PUT /publishers/{publisherID}
Request Body: JSON with updated publisher data
Response: Code 200 OK and data about the updated publisher

Delete a Publisher:
Method: DELETE /publishers/{publisherID}
Response: Code 204 No Content

Search Publishers by Name:
Method: GET /publishers?name={publisherName}
Response: Code 200 OK and a list of publishers in JSON format

Pagination of Results:
Use parameters page and limit in requests

Functional and Non-functional Requirements:

Authentication:
Description: Authentication requirement. All API requests must be accompanied by a valid authentication token implemented using JWT (JSON Web Token) technology. This ensures API security by limiting access and identifying users.
Example: Before performing an operation, the user must provide a valid JWT token in the request header.

Errors:
Description: Error handling requirement. In the case of unsuccessful requests, the API should return detailed error descriptions in JSON format. This helps developers understand and diagnose issues when using the API.
Example: Attempting to access a non-existent book should return JSON with an error code and explanation.

Status Codes:
Description: Use of standard HTTP status codes to convey information about the result of a request. Each status code should be accompanied by a detailed description, facilitating comprehension and processing of the response by the client.

Example: A successful request might return a status of 200 OK, while an unsuccessful one might return 404 Not Found.

Caching:

Description: Utilization of the HTTP Cache-Control header to manage data caching. This optimizes performance by reducing the number of redundant server requests for the same data.
Example: Setting the Cache-Control: max-age=3600 header to allow clients to cache data for one hour.

Richardson Maturity Model:

Description: Application of levels 2-3 in the Richardson Maturity Model, signifying the use of resources, standard HTTP methods, and stateless communication on the server side during interaction.
Example: Using distinct URLs for accessing resources (level 1) and providing hyperlinks for navigation between related resources (level 3).

HTTPS Support:

Description: Requirement for using the secure data transmission protocol - HTTPS. All API requests must go through a secure connection to prevent unauthorized access or data alteration.
Example: All API URLs should begin with https:// rather than http://.

Versioning:

Description: Inclusion of the API version in the URL to ensure compatibility and support for different API versions. This eases the process of making changes to the API without disrupting the functionality of client applications.
Example: Instead of /books, use /v1/books for version 1 of the API.