A library

1. Functional requirements: The system must

- allow users to create a books, authors and categories.
- allow users to retrieve a books, authors and categories.
- allow users to update a books, authors and categories.
- allow users to delete a books, authors and categories.
- allow listing categories with optional filtering

2. Non-functional requirements: The system should be

- scalable to accommodate increasing numbers of books, authors, categories.
- intuitive to use.
- available 24/7.
- updatable.

3. What entities it must use.

Book: id, title, authorId, categoryId, publicationDate.

Author: id, fullName, birthdate.

Category: id, name, description.

4. What operations with these functions the API must support.

- Operation with Book entities:

Create a Book:

Request header: POST /books

Request body (Example):

```
{
    "title": "The Great Gatsby",
    "authorId": 1,
    "categoryId": 2,
    "publicationDate": "1925-04-10"
}
```

Retrieve a Book:

Request header: GET /books/{id}

Update a Book:

Request header: PUT /books/{id}

Request body (Example):

```
{
    "title": "The Great Gatsby",
    "authorId": 1,
    "categoryId": 2,
    "publicationDate": "1925-04-10"
}
```

Delete a Book:

Requst: DELETE /books/{id}

List Books:

Request header: GET /books

Query parameters: authorId, categoryId, title, publicationDateFrom.

Response: JSON array of books

- Operations with Author entities:

Create an Author:

Request header: POST /authors

Request body (Example):

```
{
	"id": 1,
	"fullName": "F. Scott Fitzgerald",
	"birthdate": "1896-09-24"
}
```

Retrieve an Author:

Request header: GET /authors/{id}

Update an Author:

Request header: PUT /authors/{id}

Request body (Example):

```
{
	"id": 1,
	"fullName": "F. Scott Fitzgerald",
	"birthdate": "1896-09-24"
}
```

Delete an Author:

Requst: DELETE /authors/{id}

List Authors:

Request header: GET authors

Query parameters: id, fullName, birthdate

Response: JSON array of authors

- Operations with Category entities:

Create a Category:

Request header: POST /categories

Request body (Example):

```
{
    "id": 1,
    "name": "Fiction",
    "description": "A category for fictional works."
}
```

Retrieve a Category:

Request header: GET /categories/{id}

Update a Category:

Request header: PUT /categories/{id}

Request body (Example):

```
{
    "id": 1,
    "name": "Fiction",
    "description": "A category for fictional works."
}
```

Delete a Category:

Requst: DELETE /categories/{id}

List Category:

Request header: GET /categories

Query parameters: id, name, description.

Response: JSON array of categories

5. A REST API including collections, filters, pagination, etc.

Filtering and Pagination. For listing resources, support query parameters for filtering (e.g., authorId, categoryId, title, etc.) and pagination (e.g., page, size).

Error Handling. Return appropriate HTTP status codes and error messages for invalid requests, not found entities, etc.

Sorting. Allow sorting by attributes (e.g., sort books by title, publicationDate).

Status codes which can be thrown:

Create: “201 Creates” – successfully created a new entity,

“400 Bad Request” – invalid input data,

“500 Internal Server Error” – server error.

Retrieve: “200 OK” – successfully retrieved the entity,

“404 Not Found” – entity with this id doesn`t found,

“500 Internal Server Error” – server error.

Update: “200 OK” – successfully updated the entity,

“400 Bad Request” – invalid input data,

“404 Not Found” – entity with this id doesn`t found,

“500 Internal Server Error” – server error.

Delete: “204 No Content” – successfully deleted the entity,

“404 Not Found” – entity with this id doesn`t found,

“500 Internal Server Error” – server error.

List: “200 OK” – successfully retrieved the entity,

“400 Bad Request” – invalid query parameters,

“500 Internal Server Error” – server error.

Cached Methods:

List Books, List Authors, List Categories.

This endpoint is ideal for caching because it retrieves a potentially large list of entities, and caching can prevent repeated expensive queries. Cache entries should be invalidated when new entitiesare added or existing ones are updated or deleted.

Non-Cached Methods:

Create (book. author, category). This operation modifies the database and should not be cached because it changes the state of the data.

Retrieve (book. author, category). Individual entities retrievals are generally less frequent. Can be cached based on the author ID.

Update (book. author, category). This operation modifies entity details and should invalidate any existing cache for the entity.

Delete (book. author, category). This operation should invalidate the cache for the deleted entity.