**EPAM Training Course**

**Designing a REST API**

**Polina Batova**

**Topic: Library**

Functional requirements:

* The system allows users to register, log in, log out and manage their profile
* Users’ roles management (users have different roles and depending on them have different rights)
* It is available to create, update and delete books for admins
* It is available to delete and update users for admins.
* The system must provide users with information about books
* Users must be able to borrow and return books
* Users must see what books they have borrowed earlier
* The system must track the loan status of books, including the loan date and due date.
* Support pagination for books
* Sorting books by various criteria

Nonfunctional requirements:

* Secure authorization using JWT tokens
* Methods should be concise and convenient, with pagination to improve ease of use
* API must 3 level according to the Richardson model and use HATEAOS
* Caching is implemented for better performance

**Model**

Entities

1. **Book**

|  |
| --- |
| BOOK |
| id |
| title |
| author |
| ISBN |
| year of publication. |
| category |
| available copies |
| total copies |
| description |

1. **User**

|  |
| --- |
| USER |
| id |
| username |
| password |
| email |
| full name |
| role |

1. **Loan**

|  |
| --- |
| LOAN |
| id |
| book |
| borrow date |
| return date |
| status |

Relationships

One User has many Loans **(One-to-Many)**.

One Book has many Loans **(One-to-Many)**.

**Operations**

The system must support CRUD operations for users, books and loans to follow functional requirements.

**REST API Description**

Collections

* **/books**

Operations:

1. **Create a Book**

**Method**: POST /books

**Request Body**: title, author, isbn, publication\_year, category, total\_copies, available\_copies, description

**Response**: 201 Created

**Roles**: admin

**Error Codes**:

* 400 Bad Request

|  |
| --- |
| { |
| "error": "Bad Request", |
| "message": "Invalid query parameters." |
| } |

*Example of response body*

* 401 Unauthorized

|  |
| --- |
| { |
| "error": "Unauthorized", |
| "message": "Authentication required." |
| } |

*Example of response body*

* 403 Forbidden

|  |
| --- |
| { |
| "error": "Forbidden", |
| "message": "You do not have permission to access this resource." |
| } |

*Example of response body*

* 409 Conflict (if ISBN or id already exists).

|  |
| --- |
| { |
| "error": "Conflict", |
| "message": "A book with the same ISBN or ID already exists." |
| } |

*Example of response body*

There are examples of interaction with the API further.

|  |
| --- |
| POST /books  Authorization: Bearer <admin-JWT-token> |
| Content-Type: application/json |
|  |
| { |
| "title": "War and Peace", |
| "author": "L. Tolstoy", |
| "isbn": "999999999999", |
| "publication\_year": 1869, |
| "category": "Novel", |
| "total\_copies": 10,  "available\_copies":10, |
| "description": "War and Peace is a sweeping and epic novel that explores themes of love, war, and the human condition." |
| } |

*Example of the request*

|  |
| --- |
| 201 Created |
| Content-Type: application/json |

*Example of the response*

1. **Get Books**

**Method**: GET /books

**Available parameters**: page (int), limit (int)

**Response**: 200 OK, 304 Not Modified (if the page is cached)

**Roles**: none

**Error Codes**: 400 Bad Request

* 400 Bad Request

|  |
| --- |
| { |
| "error": "Bad Request", |
| "message": "Invalid query parameters." |
| } |

*Example of response body*

There are examples of interaction with the API further.

|  |
| --- |
| GET /books?page=1&limit=10 |
| Accept: application/json |

*Example of the first request*

|  |
| --- |
| 200 OK |
| Content-Type: application/json |
| Cache-Control: max-age=600, public |
| ETag: "11111111111111111111111’" |
|  |
| { |
| "page": 1, |
| "limit": 10, |
| "total": 100, |
| "data": [ |
| { |
| "id": "123", |
| "title": "Alice's Adventures in Wonderland", |
| "author": "L. Carrol", |
| "isbn": "9000000000", |
| "publication\_year": 1928, |
| "category": "Portal fantasy" |
| }, |
| { |
| "id": "124", |
| "title": "1984", |
| "author": "G. Orwell", |
| "isbn": "100000000000", |
| "publication\_year": 1949, |
| "category": "Dystopian" |
| } |
| // other books |
| ], |
| "\_links": { |
| "self": "/books?page=1&limit=10", |
| "next": "/books?page=2&limit=10", |
| "prev": null |
| } |
| } |

*Example of the first response.*

“Cache-Control” and “ETag” are header that implement caching.

“Cache-Control: max-age=600, public” - specifies that this response can be cached for 600 seconds (10 minutes). Caching can be done by any intermediate proxies and clients.

“ETag: ‘11111111111111111111111’” is a unique version identifier for the resource that allows the client to check if the resource has changed using the If-None-Match header in subsequent requests.

|  |
| --- |
| GET /books?page=1&limit=10 |
| If-None-Match: "11111111111111111111111’" |

*Example of the request with cache*

If-None-Match header should contain ETag value that was got earlier.

|  |
| --- |
| 304 Not Modified |

*Example of the response*

304 Not Modified status, indicates that the data has not changed since the last request and the client can use the cached version.

Pagination is implemented using special query parameters “page” and “limit”.

Attribute “\_links” is used for more simple navigation and it implements HATEOAS.

1. **Get a specific Book**

**Method**: GET /books/id

**Request Body**:

**Response**: 200 OK

**Roles**: none

**Error Codes**:

* 400 Bad Request

|  |
| --- |
| { |
| "error": "Bad Request", |
| "message": "Invalid book ID format." |
| } |

*Example of response body*

* 404 Not Found.

|  |
| --- |
| { |
| "error": "Not Found", |
| "message": "Book not found with the provided ID." |
| } |

*Example of response body*

There are examples of interaction with the API further.

|  |
| --- |
| GET /books/126 |
| Accept: application/json |

*Example of the request*

|  |
| --- |
| 200 OK |
| Content-Type: application/json  Cache-Control: max-age=600, public  ETag: "22222222222222222222" |
| {  "id": "126",  "title": "To Kill a Mockingbird",  "author": "H. Lee",  "isbn": "9666666666",  "year\_of\_publication": 1960,  "category": "Fiction",  "available\_copies": 5,  "total\_copies": 10,  "description": "A novel about racial injustice in the Deep South.",  "\_links": {  "self": "/books/126",  "related": "/books?category=Fiction"  }  } |

*Example of the response*

Caching is not necessary here, because there is only one book in search. However, it can be added as it was done in the GET /books operation.

1. **Get Books by filter**

**Method**: GET /books

**Available parameters**: page (int), limit (int), author(string), category(string), publication year(int), isbn(string)

**Response**: 200 OK, 304 Not Modified (if the page is cached)

**Roles**: none

**Error Codes**: 400 Bad Request

* 400 Bad Request

|  |
| --- |
| { |
| "error": "Bad Request", |
| "message": "Invalid query parameters." |
| } |

*Example of response body*

**Example of interaction**:

|  |
| --- |
| GET /books?author=G.%20Orwell&category=Dystopian&page=1&limit=10 Accept: application/json |

*Example of the request*

|  |
| --- |
| **200 OK** **Content-Type:** application/json **Cache-Control:** max-age=600, public **ETag:** "33333333333333333333" |
|  |
| { |
| "page": 1, |
| "limit": 10, |
| "total": 2, |
| "data": [ |
| { |
| "id": "124", |
| "title": "1984", |
| "author": "G. Orwell", |
| "isbn": "100000000000", |
| "publication\_year": 1949, |
| "category": "Dystopian", |
| "available\_copies": 3, |
| "total\_copies": 10, |
| "description": "A dystopian novel set in a totalitarian society." |
| } |
| ], |
| "\_links": { |
| "self": "/books?author=G.%20Orwell&category=Dystopian&page=1&limit=10", |
| "next": null, |
| "prev": null |
| } |
| } |

*Example of response*

|  |
| --- |
| GET /books?author=G.%20Orwell&category=Dystopian&page=1&limit=10 If-None-Match: "33333333333333333333" |

*Example of request with caching*

|  |
| --- |
| 304 Not Modified |

*Example of response with caching*

1. **Update a Book**

**Method:** PUT /books/{id}  
**Request Body:** title, author, isbn, publication\_year, category, total\_copies, available\_copies, description  
**Response:** 200 OK  
**Roles:** admin  
**Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found

**Example of interaction:**

|  |
| --- |
| PUT /books/124 Authorization: Bearer <admin-JWT-token> Content-Type: application/json |
| { |
| "title": "1984", |
| "author": "G. Orwell", |
| "isbn": "100000000000", |
| "publication\_year": 1949, |
| "category": "Dystopian", |
| "total\_copies": 15, |
| "available\_copies": 5, |
| "description": "Updated description of the book." |
| } |

*Example of request*

|  |
| --- |
| 200 OK Content-Type: application/json |
| { |
| "id": "124", |
| "title": "1984", |
| "author": "G. Orwell", |
| "isbn": "100000000000", |
| "publication\_year": 1949, |
| "category": "Dystopian", |
| "total\_copies": 15, |
| "available\_copies": 5, |
| "description": "Updated description of the book.", |
| "\_links": { |
| "self": "/books/124", |
| "related": "/books?category=Dystopian" |
| } |
| } |

*Example of response*

1. **Delete a Book**

**Method:** DELETE /books/{id}  
**Response:** 204 No Content  
**Roles:** admin  
**Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found

**Examples of interaction:**

|  |
| --- |
| DELETE /books/124 Authorization: Bearer <admin-JWT-token> Accept: application/json |

*Example of request*

|  |
| --- |
| 204 No Content |

*Example of response*

* **/users**

Operations:

1. **Create user**

**Method:** POST /users  
**Request Body:** username, password, email, full\_name, role  
**Response:** 201 Created, User details in response body.  
**Roles:** admin  
**Error Codes**:

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 409 Conflict (if user with such username already exists)

**Examples of interaction:**

|  |
| --- |
| POST /users Authorization: Bearer <admin-JWT-token> Content-Type: application/json |
|  |
| { |
| "username": "ben10", |
| "password": "123456####Dv", |
| "email": "ben10@example.com", |
| "full\_name": "Ben Jack", |
| "role": "user" |
| } |

*Example of the request*

|  |
| --- |
| 201 Created Content-Type: application/json |
|  |
| { |
| "id": "1", |
| "username": "ben10", |
| "email": " ben10@example.com", |
| "full\_name": " Ben Jack", |
| "role": "user", |
| "\_links": { |
| "self": "/users/1" |
| } |
| } |

*Example of the response*

1. **Get users**

**Method:** GET /users  
**Available parameters:** page (int), limit (int)  
**Response:** 200 OK, 304 Not Modified (if the page is cached)  
**Roles:** admin  
**Error Codes:**

* Bad Request

**Examples of interaction:** examples are similar to the get requests from the books collection operations, only entities are different.

1. **Get a specific user**

**Method:** GET /users/{id}  
**Response:** 200 OK  
**Roles:** admin, user (users can see only their own page!)  
**Error Codes:**

* + 400 Bad Request
  + 401 Unauthorized
  + 403 Forbidden
  + 404 Not Found

**Example of interactions:**

examples are similar to the get requests from the books collection operations, only entities are different.

1. **Update user**

**Method:** PUT /users/{id}  
**Request Body:** username, email, full\_name, role  
**Response:** 200 OK  
**Roles:** admin

**Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found

**Example of interactions:** examples are similar to the get requests from the books collection operations, only entities are different.

1. **Delete user**

**Method**: DELETE /users/{id} **Response:** 204 No Content **Roles:** admin **Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found

**Example of interactions:** examples are similar to the get requests from the books collection operations, only entities are different.

* **/loans**

1. **Create loan**

**Method:** POST /loans  
**Request Body:** book\_id, user\_id, borrow\_date, return\_date  
**Response:** 201 Created, loan details in response body.  
**Roles:** user, admin  
**Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found (if the book with such id does not exist)

|  |
| --- |
| { |
| "error": "Not Found", |
| "message": "Book was not found with the provided ID." |
| } |

*Example of response*

* 409 Conflict (if there are no free books for loan)

|  |
| --- |
| { |
| "error": "Conflict", |
| "message": "The book is not available now." |
| } |

*Example of response*

**Examples of interaction:**

|  |
| --- |
| POST /loans Authorization: Bearer <user-JWT-token> Content-Type: application/json |
|  |
| { |
| "book\_id": "123", |
| "user \_id": "5", |
| "borrow\_date": "2024-08-17", |
| "return\_date": "2024-09-17" |
| } |

*Example of request*

|  |
| --- |
| 201 Created Content-Type: application/json |
|  |
| { |
| "id": "1", |
| "book\_id": "123", |
| "user\_id": "5", |
| "borrow\_date": "2024-08-17", |
| "return\_date": "2024-09-17", |
| "\_links": { |
| "self": "/loans/1", |
| "user": "/users/5", |
| "book": "/books/123" |
| } |
| } |

*Example of response*

1. **Get loans**

**Method:** GET /loans  
**Available parameters:** page (int), limit (int)  
**Response:** 200 OK, 304 Not Modified (if the page is cached)  
**Roles:** admin  
**Error Codes:**

* 400 Bad Request

**Examples of interaction:**

|  |
| --- |
| GET /loans?page=1&limit=10 Authorization: Bearer <admin-JWT-token> Accept: application/json |

*Example of request*

|  |
| --- |
| 200 OK Content-Type: application/json Cache-Control: max-age=600, public ETag: "66666666666666666666" |
|  |
| { |
| "page": 1, |
| "limit": 10, |
| "total": 50, |
| "data": [ |
| { |
| "id": "1", |
| "book\_id": "123", |
| "user\_id": "5", |
| "borrow\_date": "2024-08-17", |
| "return\_date": "2024-09-17" |
| }, |
| // other loans |
| ], |
| "\_links": { |
| "self": "/loans?page=1&limit=10", |
| "next": "/loans?page=2&limit=10", |
| "prev": null |
| } |
| } |

*Example of response*

|  |
| --- |
| GET /loans?page=1&limit=10 If-None-Match: "66666666666666666666" |

*Example of request with caching*

|  |
| --- |
| 304 Not Modified |

*Example of response with caching*

1. **Get a specific loan**

**Method:** GET /loans/{id}  
**Response:** 200 OK  
**Roles:** admin  
**Error Codes:**

* 400 Bad Request
* 404 Not Found

**Examples of interaction:**

|  |
| --- |
| GET /loans/1 Authorization: Bearer <admin-JWT-token> Accept: application/json |

*Example of request*

|  |
| --- |
| 200 OK Content-Type: application/json Cache-Control: max-age=600, public ETag: "77777777777777777777" |
|  |
| { |
| "id": "1", |
| "book\_id": "123", |
| "user\_id": "456", |
| "borrow\_date": "2024-08-17", |
| "return\_date": "2024-09-17", |
| "\_links": { |
| "self": "/loans/1", |
| "user": "/users/456", |
| "book": "/books/123" |
| } |
| } |

*Example of response*

1. **Get loans by user**

**Method:** GET /loans/user/{user\_id}  
**Available parameters:** page (int), limit (int)  
**Response:** 200 OK, 304 Not Modified (if the page is cached)  
**Roles:** admin, user (user can access only their own loans)  
**Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found

**Example of interaction:**

|  |
| --- |
| GET /loans/user/5?page=1&limit=10 Authorization: Bearer <user-JWT-token> Accept: application/json |

*Example of request*

|  |
| --- |
| 200 OK Content-Type: application/json Cache-Control: max-age=600, public ETag: "88888888888888888888" |
|  |
| { |
| "page": 1, |
| "limit": 10, |
| "total": 5, |
| "data": [ |
| { |
| "id": "1", |
| "book\_id": "123", |
| "borrow\_date": "2024-08-17", |
| "return\_date": "2024-09-17" |
| }, |
| { |
| "id": "2", |
| "book\_id": "124", |
| "borrow\_date": "2024-07-01", |
| "return\_date": "2024-07-31" |
| } |
| // other loans for this user |
| ], |
| "\_links": { |
| "self": "/loans/user/5?page=1&limit=10", |
| "next": null, |
| "prev": null |
| } |
| } |

*Example of response*

1. **Get loans by book**

**Method:** GET /loans/book/{book\_id}  
**Available parameters:** page (int), limit (int)  
**Response:** 200 OK, 304 Not Modified (if the page is cached)  
**Roles:** admin  
**Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found

**Examples of interaction:**

|  |
| --- |
| GET /loans/book/123?page=1&limit=10 Authorization: Bearer <admin-JWT-token> Accept: application/json |

*Example of request*

|  |
| --- |
| 200 OK Content-Type: application/json Cache-Control: max-age=600, public ETag: "99999999999999999999" |
| { |
| "page": 1, |
| "limit": 10, |
| "total": 2, |
| "data": [ |
| { |
| "id": "1", |
| "user\_id": "456", |
| "borrow\_date": "2024-08-17", |
| "return\_date": "2024-09-17" |
| }, |
| { |
| "id": "3", |
| "user\_id": "789", |
| "borrow\_date": "2024-06-01", |
| "return\_date": "2024-06-30" |
| } |
| // other loans for this book |
| ], |
| "\_links": { |
| "self": "/loans/book/123?page=1&limit=10", |
| "next": null, |
| "prev": null |
| } |
| } |

*Example of response*

1. **Update loan (prolong or finish)**

**Method:** PUT /loans/{id}  
**Request Body:** return\_date  
**Response:** 200 OK  
**Roles:** admin, user (only their own loan)  
**Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found

**Examples of interactions:** examples are similar to same operations in the book collection, only entities are different.

1. **Delete loan**

**Method:** DELETE /loans/{id}  
**Response:** 204 No Content  
**Roles:** admin  
**Error Codes:**

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found

**Examples of interactions:** examples are similar to same operations in the book collection, only entities are different.

**Comments:**

Filters are implemented in searching operations with books and loans using query parameters.

Pagination was implemented using the “page” and “limit” query parameters.

Authentication was realized with JWT. Special header (Authorization: Bearer <token>) was used in this purpose. That is why authentication errors were added (401 Unauthorized, 403 Forbidden).

Caching is implemented with special headers “Cache-Control” and “ETag”, so the 304 Not Modified response code was added where caching was used. Only some methods like updating, creating, deleting resources do not use caching because there is no need.

The methods are mapped not only to a URI but to HTTP methods as well. HATEOAS is used (links to other resources).