Library Management System API Documentation

Base URLs

Development	http://localhost:3002/api
Production	N/A

Modules Overview

Modules
Book Management Module
Author Management Module
User Management Module
Loan Management Module
Book Review Management Module

1. Book Management Module

These APIs provide CRUD operations for managing books. It is built using Node JS and MongoDB.

Endpoints Overview

Endpoint	Method	Description

/books	GET	Retrieve all books
/books/:id	GET	Retrieve a book by ID
/books	POST	Add a new book
/books/:id	PUT	Update book details
/books/:id	DELETE	Remove a book
/books/:id/upload-cover	POST	Upload a book's cover image

Endpoints Details

1. Retrieve All Books

Method : **GET**

Endpoint : **/books**

Description : **Fetches a list of all available books. Supports optional query parameters for filtering and limiting results.**

Query Parameters

Parameter	Type	Description
`title`	string	The title of the book
`author`	string	The name of the author
`status`	string	Indicates the current status of the book: "Available" → Book is available for borrowing. "Checked Out" → Someone has borrowed the book.

		"Reserved" → The book is reserved by a member.
`limit`	number	The maximum number of authors to return. Default is `0` (no limit).

Request Example

```
GET /books?author=John&status=available&limit=5
```

Response Example

- Status: `200 OK`
- Body:


```
{
    "message": "All Available Books",
    "data": [
        {
            "_id": "67c9a4d36467652d0864aa8c",
            "title": "The Alchemist",
            "author": "Paulo Coelho",
            "publishedYear": 1998,
            "pages": 207,
            "copiesAvailable": 5,
            "status": "Available",
            "createdAt": "2025-06-03T13:36:19.000Z",
            "updatedAt": "2025-03-06T13:53:00.183Z",
            "__v": 0
        }
    ]
```

```
}
```

---

2. Retrieve a Book by ID

Method : **GET**

Endpoint : **/books/:id**

Description : **Fetches details of a single book based on the provided book ID.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the book.

Response Example

- Status: `200 OK`
- Body:

```
{
    "message": "Get book by ID",
    "data": [
        {
            "_id": "67c9a4d36467652d0864aa8c",
            "title": "The Alchemist",
            "author": {
                "_id": "67ce9baf1963cc876ad3c2e6",
                "name": "Demo Author",
                "bio": "An Italian Author.",
                "nationality": "Italian",
                "books": [],
```

```
            "createdAt": "2025-03-10T07:58:39.623Z",
            "updatedAt": "2025-03-10T07:58:39.623Z",
            "__v": 0
        },
        "publishedYear": 1998,
        "pages": 207,
        "copiesAvailable": 5,
        "status": "Available",
        "createdAt": "2025-06-03T13:36:19.000Z",
        "updatedAt": "2025-03-06T13:53:00.183Z",
        "__v": 0
    }
]
}
```


- Status: `404 Not Found` (if entity does not exist)


```
{
    "error": "Book not found"
}
```


- Status: `400 Bad Request` (if entity id is invalid)


```
{
    "error": "Book id invalid"
}
```


---

3. Add a New Book

Method : **POST**

Endpoint : **/books**

Description : **Creates a new book entry in the database.**

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description
`title`	string *	Title of the book
`author`	string *	The Author ID of the book
`publishedYear`	number *	Year the book was published
`pages`	number *	The total no. of pages
`copiesAvailable`	number	The count of copies of book
`status`	string	Indicates the current status of the book: "Available" → Book is available for borrowing. "Checked Out" → Someone has borrowed the book. "Reserved" → The book is reserved by a member. Default status is "Available".

Request Example

```
{
  "title": "The Alchemist",
  "author": "67ce9baf1963cc876ad3c2e6",
  "publishedYear": 1988,
  "pages": 208,
  "copiesAvailable": 5,
  "status": "Available"
}
```

Response Example

- Status: `201 Created`
- Body:

```
{
    "message": "Book Created",
    "data": [
        {
            "title": "The Alchemist",
            "author": "67ce9baf1963cc876ad3c2e6",
            "publishedYear": 1998,
            "pages": 207,
            "copiesAvailable": 5,
            "status": "Available",
            "_id": "67c9ad7499e5b3611c0fb4b0",
            "createdAt": "2025-03-06T14:13:08.934Z",
            "updatedAt": "2025-03-06T14:13:08.934Z",
            "__v": 0
```

```
        }
    ]
}
```

- Status: `400 bad request` (if entity is not unique)

```
{
    "error": "Book already exists"
}
```

- Status: `400 Bad Request` (if entity's body is missing)

```
{
    "error": "\"title\" is required"
}
```

---

4. Update Book Details

Method : **PUT**

Endpoint : **/books/:id**

Description : **Updates details of a book based on the provided book ID.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the book

Request Body

Header	Type	Description
Content-Type	string *	application/json

Parameter	Type	Description
id	string *	The ID of the book
title	string *	Updated title of the book
author	string *	Updated author name
publishedYear	number *	Updated publication year
pages	number *	The total no. of pages
copiesAvailable	number	The count of copies of book
status	string	Indicates the current status of the book: "Available" → Book is available for borrowing. "Checked Out" → Someone has borrowed the book. "Reserved" → The book is reserved by a member. Default status is "Available".

Request Example

```
{
  "title": "The Alchemist Updated Title",
  "author": "67ce9baf1963cc876ad3c2e6",
  "publishedYear": 1988,
```

```
  "pages": 208,
  "copiesAvailable": 5,
  }
```

Response Example

- Status: `200 OK`
- Body:

```
{
    "message": "Book Updated",
    "data": [
        {
            "_id": "67c9ad7499e5b3611c0fb4b0",
            "title": "The Alchemist Updated Title",
            "author": "67ce9baf1963cc876ad3c2e6",
            "publishedYear": 1998,
            "pages": 207,
            "copiesAvailable": 5,
            "status": "Available",
            "createdAt": "2025-03-06T14:13:08.934Z",
            "updatedAt": "2025-03-06T14:40:18.880Z",
            "__v": 0
        }
    ]
}
```

- Status: `404 Not Found` (if entity does not exist)

```
{
    "error": "Book not found"
}
```

- Status: `400 bad request (if entity id invalid)`

```
{
    "error": "Book id invalid"
}
```

- Status: `400 bad request` (if entity 's body is missing)

```
{
    "error": "\"title\" is required"
}
```

---

5. Remove a Book

Method : **DELETE**

Endpoint : **/books/:id**

Description : **Removes a book from the database based on the provided book ID.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the book

Response Example

- Status: `200 OK`
- Body:

```
{
    "message": "Book deleted",
    "data": [
        {
            "_id": "67c9a4d36467652d0864aa8c",
            "title": "The Alchemist",
            "author": "67ce9baf1963cc876ad3c2e6",
            "publishedYear": 1998,
            "pages": 207,
            "copiesAvailable": 5,
            "status": "Available",
            "createdAt": "2025-06-03T13:36:19.000Z",
            "updatedAt": "2025-03-06T13:53:00.183Z",
            "__v": 0
        }
    ]
}
```

- Status: `404 Not Found` (if entity does not exist)

```
{
    "error": "Book not found"
}
```

- Status: `400 bad request (if entity id invalid)`

```
{
    "error": "Book id invalid"
}
```

---

6. Upload a book's cover image

Method : **POST**

Endpoint : **/books/:id/upload-cover**

Description : **Uploads a new cover image of an existing book in the database.**

Request Parameter

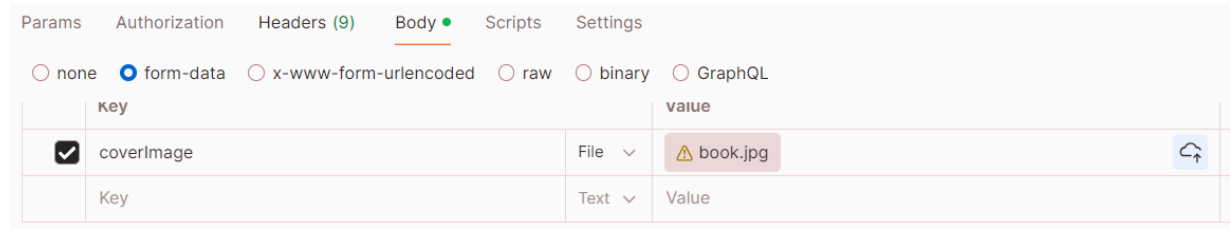
Parameter	Type	Description
`id`	string *	The ID of the book.

Request Body

Header	Type	Description
`Content-Type`	multipart/form-data	Specifies that the request contains file data.

Parameter	Type	Description
`coverImage`	file	The uploaded file for the book's cover image.

Request Example



Response Example

- Status: `200 Ok`
- Body:

```
{
    "message": "Book cover uploaded successfully",
    "data": [
        {
            "_id": "67c9a4d36467652d0864aa8c",
            "title": "The Alchemist",
            "author": "67ce9baf1963cc876ad3c2e6",
            "publishedYear": 1998,
            "pages": 207,
            "copiesAvailable": 5,
            "status": "Available",
            "coverImage": "uploads\\BookCover\\10-Mar-2025-05-29-45-
            book.jpg",
            "reviews": [],
            "createdAt": "2025-03-10T11:52:59.855Z",
            "updatedAt": "2025-03-10T11:59:45.932Z",
            "__v": 0
        }
    ]
```

```
}
```

- Status: `400 Bad Request` (if file is not uploaded)

```
{
    "error": "Upload a cover image for book"
}
```

- Status: `400 Bad Request` (if file exceed the size limit)

```
{
    "error": "File is too large, file size limit 1MB"
}
```

---

2. Author Management Module

These APIs provide CRUD operations for managing authors. It is built using Node JS and MongoDB.

Endpoints Overview

Endpoint	Method	Description
/authors	GET	Retrieve all authors
/authors/:id	GET	Retrieve a author by ID
/authors	POST	Add a new author
/authors/:id	PUT	Update author details
/authors/:id	DELETE	Remove a author

Endpoints Details

1. Retrieve All Authors

Method : **GET**

Endpoint : **/authors**

Description : **Fetches a list of all available authors. Supports optional query parameters for filtering and limiting results.**

Query Parameters

Parameter	Type	Description
`name`	string	The name of the author
`nationality`	string	The nationality of the author
`limit`	number	The maximum number of authors to return. Default is `0` (no limit).

Request Example

```
GET /authors
```

```
GET /authors?name=Jhon&nationality=USA&limit=1
```

Response Example

- Status: `200 OK`
- Body:

```
{
```

```
    "message": "All Available Authors",
    "data": [
        {
            "_id": "67ce9253eab5672e99759cd3",
            "name": "Demo Author",
            "bio": "An Italian Author.",
            "nationality": "Italian",
            "books": [],
            "createdAt": "2025-03-10T07:18:43.508Z",
            "updatedAt": "2025-03-10T07:24:24.931Z",
            "__v": 0
        }
    ]
}
```


---

2. Retrieve a Author by ID

Method : **GET**

Endpoint : **/authors/:id**

Description : **Fetches details of a single author based on the provided author ID.**

Request Parameter

Parameter	Type	Description
id	string *	The ID of the author.

Request Example

```
GET /authors/67ce9253eab5672e99759cd3
```

Response Example

- Status: `200 OK`
- Body:


```
{
    "message": "Get author by ID",
    "data": [
        {
            "_id": "67ce9253eab5672e99759cd3",
            "name": "Demo Author",
            "bio": "An Italian Author.",
            "nationality": "Italian",
            "books": [],
            "createdAt": "2025-03-10T07:18:43.508Z",
            "updatedAt": "2025-03-10T07:24:24.931Z",
            "__v": 0
        }
    ]
}
```


- Status: `404 Not Found` (if entity does not exist)


```
{
    "error": "Author not found"
}
```


- Status: `400 Bad Request` (if entity id is invalid)

```
{
    "error": "Author id invalid"
}
```

---

3. Add a New Author

Method : **POST**

Endpoint : **/authors**

Description : **Creates a new author entry in the database.**

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description
`name`	string *	The unique name of the author
`bio`	string	A brief biography of the author
`nationality`	string *	The nationality of the author
`books`	array	A list of books written by the author

Request Example

```
{
```

```
    "name": "Demo Author",
    "nationality": "Indian"
}
```

Response Example

- Status: `201 Created`
- Body:

```
{
    "message": "Author Created",
    "data": [
        {
            "name": "Demo Author",
            "bio": "",
            "nationality": "Indian",
            "books": [],
            "_id": "67ce9253eab5672e99759cd3",
            "createdAt": "2025-03-10T07:18:43.508Z",
            "updatedAt": "2025-03-10T07:18:43.508Z",
            "__v": 0
        }
    ]
}
```

- Status: `400 bad request` (if entity is not unique)

```
{
    "error": "Author already exists"
```

```
}
```

- Status: `400 bad request` (if entity 's body is missing)

```
{
    "error": "\"name\" is required"
}
```

---

4. Update Author Details

Method : **PUT**

Endpoint : **/authors/:id**

Description : **Updates an existing author's details.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the author.

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description
`name`	string *	The unique name of the author

`bio`	string	A brief biography of the author
`nationality`	string *	The nationality of the author
`books`	array	A list of books written by the author

Request Example

```
{
    "name": "Demo Author",
    "nationality": "Italian",
    "bio": "An Italian Author."
}
```

Response Example

- Status: `200 OK`
- Body:

```
{
    "message": "Author Updated",
    "data": [
        {
            "_id": "67ce9253eab5672e99759cd3",
            "name": "Demo Author",
            "bio": "An Italian Author.",
            "nationality": "Italian",
            "books": [],
            "createdAt": "2025-03-10T07:18:43.508Z",
            "updatedAt": "2025-03-10T07:24:24.931Z",
```

```
            "__v": 0
        }
    ]
}
```

- Status: `404 Not Found` (if entity does not exist)

```
{
    "error": "Author not found"
}
```

- Status: `400 Bad Request (if entity id invalid)`

```
{
    "error": "Author id invalid"
}
```

- Status: `400 bad request` (if entity's body is missing)

```
{
    "error": "\"title\" is required"
}
```

---

5. Remove a Author

Method : **DELETE**

Endpoint : **/authors/:id**

Description : **Deletes an author from the database.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the author.

Request Example

```
DELETE /authors/67ce9253eab5672e99759cd3
```

Response Example

- Status: `200 OK`
- Body:


```
{
    "message": "Author deleted",
    "data": [
        {
            "_id": "67ce9253eab5672e99759cd3",
            "name": "Demo Author",
            "bio": "An Italian Author.",
            "nationality": "Italian",
            "books": [],
            "createdAt": "2025-03-10T07:18:43.508Z",
            "updatedAt": "2025-03-10T07:24:24.931Z",
            "__v": 0
        }
    ]
}
```

- Status: `404 Not Found` (if entity does not exist)

```
{
    "error": "Author not found"
}
```

- Status: `400 Bad Request (if entity id invalid)`

```
{
    "error": "Author id invalid"
}
```

---

3. User Management Module

These APIs provide CRUD operations for managing users. It is built using Node JS and MongoDB.

Endpoints Overview

Endpoint	Method	Description
/users	GET	Retrieve all users
/users/:id	GET	Retrieve a user by ID
/users	POST	Add a new user
/users/:id	PUT	Update user details
/users/:id	DELETE	Remove a user
/users/:id/upload-profile-picture	POST	Upload a user's profile picture

Endpoints Details

1. Retrieve All Users

Method : **GET**

Endpoint : **/users**

Description : **Fetches a list of all available users. Supports optional query parameters for filtering and limiting results.**

Query Parameters

Parameter	Type	Description
`name`	string	The name of the user
`email`	string	The unique email of the user
`role`	string	The role of the user, can be "member", "librarian", or "admin"
`limit`	number	The maximum number of authors to return. Default is `0` (no limit).

Request Example

```
GET /users
```

```
GET /users?name=Jhon&email=jhon@gmail.com&role=member
```

Response Example

- Status: `200 OK`
- Body:

```
{
    "message": "All Available Users",
    "data": [
        {
            "_id": "67cd317f99192ba8535c6b8c",
            "name": "Demo User",
            "email": "demouser2@gmail.com",
            "role": "librarian",
            "profilePicture": "",
            "createdAt": "2025-03-09T06:13:19.932Z",
            "updatedAt": "2025-03-09T06:13:19.932Z",
            "__v": 0
        },
        {
            "_id": "67ce9bf71963cc876ad3c2e9",
            "name": "Demo User",
            "email": "demo.user@gmail.com",
            "role": "member",
            "profilePicture": "",
            "createdAt": "2025-03-10T07:59:51.674Z",
            "updatedAt": "2025-03-10T07:59:51.674Z",
            "__v": 0
        }
    ]
}
```

2. Retrieve a User by ID

Method : **GET**

Endpoint : **/users/:id**

Description : **Fetches details of a single user based on the provided user ID.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the user.

Request Example

```
GET /users/67ce9253eab5672e99759cd3
```

Response Example

- Status: `200 OK`
- Body:


```
{
    "message": "Get author by ID",
    "data": [
        {
            "_id": "67ce9253eab5672e99759cd3",
            "name": "Demo Author",
            "bio": "An Italian Author.",
            "nationality": "Italian",
            "books": [],
            "createdAt": "2025-03-10T07:18:43.508Z",
            "updatedAt": "2025-03-10T07:24:24.931Z",
            "__v": 0
        }
```

```
    ]
}
```

- Status: `404 Not Found` (if entity does not exist)

```
{
    "error": "User not found"
}
```

- Status: `400 Bad Request` (if entity id is invalid)

```
{
    "error": "User ID invalid"
}
```

---

3. Add a New User

Method : **POST**

Endpoint : **/users**

Description : **Creates a new user entry in the database.**

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description

`name`	string *	The name of the user
`email`	string *	The unique email of the user, in lowercase
`role`	string	The role of the user, can be "member", "librarian", or "admin" (default: "member").
`profilePicture`	string	URL or path to the user's profile picture (default: empty string).

Request Example

```
{
    "name": "Demo User",
    "email": "demo.user@gmail.com"
}
```

Response Example

- Status: `201 Created`
- Body:

```
{
    "message": "User Created",
    "data": [
        {
            "name": "Demo User",
            "email": "demo.user@gmail.com",
            "role": "member",
            "profilePicture": "",
```

```
            "_id": "67ce9bf71963cc876ad3c2e9",
            "createdAt": "2025-03-10T07:59:51.674Z",
            "updatedAt": "2025-03-10T07:59:51.674Z",
            "__v": 0
        }
    ]
}
```


- Status: `400 Bad Request` (if entity is not unique)


```
{
    "error": "User already exists"
}
```


- Status: `400 Bad Request` (if entity 's body is missing)


```
{
    "error": "\"name\" is required"
}
```


---

4. Update User Details

Method : **PUT**

Endpoint : **/users/:id**

Description : **Updates an existing user's details.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the user.

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description
`name`	string *	The name of the user
`email`	string *	The unique email of the user, in lowercase
`role`	string	The role of the user, can be "member", "librarian", or "admin" (default: "member").
`profilePicture`	string	URL or path to the user's profile picture (default: empty string).

Request Example

```
{
    "name": "Demo Author",
    "nationality": "Italian",
    "bio": "An Italian Author."
}
```

Response Example

- Status: `200 OK`

- Body:


```
{
    "message": "Author Updated",
    "data": [
        {
            "_id": "67ce9253eab5672e99759cd3",
            "name": "Demo Author",
            "bio": "An Italian Author.",
            "nationality": "Italian",
            "books": [],
            "createdAt": "2025-03-10T07:18:43.508Z",
            "updatedAt": "2025-03-10T07:24:24.931Z",
            "__v": 0
        }
    ]
}
```


- Status: `404 Not Found` (if entity does not exist)


```
{
    "error": "User not found"
}
```


- Status: `400 Bad Request (if entity id invalid)`


```
{
    "error": "User ID invalid"
```

```
}
```

- Status: `400 bad request` (if entity's body is missing)

```
{
    "error": "\"title\" is required"
}
```

---

5. Remove a User

Method : **DELETE**

Endpoint : **/users/:id**

Description : **Deletes an user from the database.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the user.

Request Example

```
DELETE /users/67ce9253eab5672e99759cd3
```

Response Example

- Status: `200 OK`
- Body:

```
{
    "message": "User deleted",
```

```
    "data": [
        {
            "_id": "67cd317f99192ba8535c6b8c",
            "name": "Demo User",
            "email": "demouser2@gmail.com",
            "role": "librarian",
            "profilePicture": "",
            "createdAt": "2025-03-09T06:13:19.932Z",
            "updatedAt": "2025-03-09T06:13:19.932Z",
            "__v": 0
        }
    ]
}
```


- Status: `404 Not Found` (if entity does not exist)


```
{
    "error": "User not found"
}
```


- Status: `400 Bad Request (if entity id invalid)`


```
{
    "error": "User ID invalid"
}
```


---

6. Upload a user's profile picture

Method : **POST**

Endpoint : **/users/:id/upload-profile-picture**

Description : **Uploads a new profile picture of an existing user in the database.**

Request Parameter

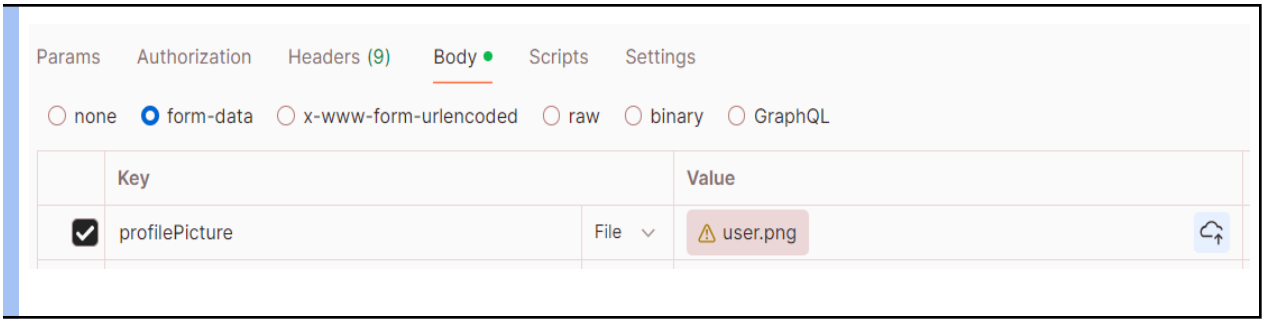
Parameter	Type	Description
`id`	string *	The ID of the user.

Request Body

Header	Type	Description
`Content-Type`	multipart/form-data	Specifies that the request contains file data.

Parameter	Type	Description
`profilePicture`	file	The uploaded file for the user's profile picture.

Request Example



Response Example

- Status: `200 Ok`
- Body:

```
{
    "message": "User profile picture uploaded successfully",
    "data": [
        {
            "_id": "67ce9bf71963cc876ad3c2e9",
            "name": "Demo User",
            "email": "demo.user@gmail.com",
            "role": "member",
            "profilePicture":"uploads\\UserProfile\\10-Mar-2025-02-40-24-
            user.png",
            "createdAt": "2025-03-10T07:59:51.674Z",
            "updatedAt": "2025-03-10T09:10:24.388Z",
            "__v": 0
        }
    ]
}
```


- Status: `400 Bad Request` (if file is not uploaded)


```
{
    "error": "Upload a profile photo for user"
}
```


- Status: `400 Bad Request` (if file exceed the size limit)


```
{
    "error": "File is too large, file size limit 1MB"
```

```
}
```

---

4. Loan Management Module

These APIs provide CRUD operations for managing loans. It is built using Node JS and MongoDB.

Endpoints Overview

Endpoint	Method	Description
/loans	GET	Retrieve all loans
/loans	POST	Add a new loan
/loans/:id	PUT	Update loan details
/loans/:id	DELETE	Remove a loan

Endpoints Details

1. Retrieve All Loans

Method : **GET**

Endpoint : **/loans**

Description : **Fetches a list of all available loans.**

Request Example

```
GET /loans
```

Response Example

- Status: `200 OK`
- Body:


```
{
    "message": "All Available Loans",
    "data": [
        {
            "_id": "67ced7b2a61aac1e339207ab",
            "user": {
                "_id": "67ce9bf71963cc876ad3c2e9",
                "name": "Demo User",
                "email": "demo.user@gmail.com",
                "role": "member",
                "profilePicture":
"uploads\\UserProfile\\10-Mar-2025-02-40-24-user.png",
                "createdAt": "2025-03-10T07:59:51.674Z",
                "updatedAt": "2025-03-10T09:10:24.388Z",
                "__v": 0
            },
            "book": {
                "_id": "67ced29b4d3726bee7913f84",
                "title": "Demo Book",
                "author": "67ce9baf1963cc876ad3c2e6",
                "publishedYear": 2023,
                "pages": 120,
                "copiesAvailable": 1,
                "status": "Available",
                "coverImage":"uploads\\BookCover\\10-Mar-2025-05-29-45
```

```
                -book.jpg",
                "reviews": [],
                "createdAt": "2025-03-10T11:52:59.855Z",
                "updatedAt": "2025-03-10T11:59:45.932Z",
                "__v": 0
            },
            "borrowDate": "2025-03-10T11:59:43.711Z",
            "isReturned": false,
            "returnDate": "2025-03-17T12:14:42.890Z",
            "createdAt": "2025-03-10T12:14:42.891Z",
            "updatedAt": "2025-03-10T12:14:42.891Z",
            "__v": 0
        }
    ]
}
```


---

2. Add a New Loan

Method : **POST**

Endpoint : **/loans**

Description : **Creates a new loan entry in the database.**

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description
`user`	string *	The ID of the user who borrowed the book.
`book`	string *	The ID of the book that is borrowed.
`borrowDate`	date(string)	The date when the book was borrowed. Defaults to the current date.
`returnDate`	date(string)	The expected return date (7 days from the borrow date by default).
`isReturned`	boolean	Indicates whether the book has been returned. Defaults to false.

Request Example

```
{
    "user": "67ce9bf71963cc876ad3c2e9",
    "book": "67ced29b4d3726bee7913f84"
}
```

Response Example

- Status: `201 Created`
- Body:

```
{
    "message": "Loan Created",
    "data": [
        {
```

```
            "user": "67ce9bf71963cc876ad3c2e9",
            "book": "67ced29b4d3726bee7913f84",
            "borrowDate": "2025-03-10T11:59:43.711Z",
            "isReturned": false,
            "_id": "67ced7b2a61aac1e339207ab",
            "returnDate": "2025-03-17T12:14:42.890Z",
            "createdAt": "2025-03-10T12:14:42.891Z",
            "updatedAt": "2025-03-10T12:14:42.891Z",
            "__v": 0
        }
    ]
}
```


- Status: `400 Bad Request` (if entity is not unique)


```
{
    "error": "Loan already exists"
}
```


- Status: `400 Bad Request` (if entity ’s body is missing)


```
{
    "error": "\"user\" is required"
}
```


---

3. Update Loan Details

Method : **PUT**

Endpoint : **/loans/:id**

Description : **Updates an existing loan's details.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the loan.

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description
`user`	string *	The ID of the user who borrowed the book.
`book`	string *	The ID of the book that is borrowed.
`borrowDate`	date(string)	The date when the book was borrowed. Defaults to the current date.
`returnDate`	date(string)	The expected return date (7 days from the borrow date by default).

Request Example

```
{
```

```
    "user": "67ce9bf71963cc876ad3c2e9",
    "book": "67ced29b4d3726bee7913f84",
    "returnDate": "2025-03-24T12:14:42.890Z"
}
```

Response Example

- Status: `200 OK`
- Body:

```
{
    "message": "Loan Updated",
    "data": [
        {
            "_id": "67ced7b2a61aac1e339207ab",
            "user": "67ce9bf71963cc876ad3c2e9",
            "book": "67ced29b4d3726bee7913f84",
            "borrowDate": "2025-03-10T11:59:43.711Z",
            "isReturned": false,
            "returnDate": "2025-03-24T12:14:42.890Z",
            "createdAt": "2025-03-10T12:14:42.891Z",
            "updatedAt": "2025-03-10T12:20:12.909Z",
            "__v": 0
        }
    ]
}
```

- Status: `404 Not Found` (if entity does not exist)

```
{
    "error": "Loan not found"
}
```

- Status: `400 Bad Request (if entity id invalid)`

```
{
    "error": "Loan ID invalid"
}
```

- Status: `400 bad request` (if entity's body is missing)

```
{
    "error": "\"user\" is required"
}
```

---

4. Remove a Loan

Method : **DELETE**

Endpoint : **/loans/:id**

Description : **Deletes a loan from the database.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the loan.

Request Example

```
DELETE /users/67ced7b2a61aac1e339207ab
```

Response Example

- Status: `200 OK`
- Body:


```
{
    "message": "Loan deleted",
    "data": [
        {
            "_id": "67ced7b2a61aac1e339207ab",
            "user": "67ce9bf71963cc876ad3c2e9",
            "book": "67ced29b4d3726bee7913f84",
            "borrowDate": "2025-03-10T11:59:43.711Z",
            "isReturned": true,
            "returnDate": "2025-03-24T12:14:42.890Z",
            "createdAt": "2025-03-10T12:14:42.891Z",
            "updatedAt": "2025-03-10T12:22:29.031Z",
            "__v": 0
        }
    ]
}
```


- Status: `404 Not Found` (if entity does not exist)


```
{
    "error": "Loan not found"
}
```

- Status: `400 Bad Request (if entity id invalid)`

```
{
    "error": "Loan ID invalid"
}
```

---

5. Book Review Management Module

These APIs provide CRUD operations for managing reviews on a book. It is built using Node JS and MongoDB.

Endpoints Overview

Endpoint	Method	Description
/books/:id/reviews	GET	Retrieve all review on a book
/books/:id/reviews	POST	Add a new review on a book
/books/:id/reviews/:reviewId	PUT	Update review details
/books/:id/reviews/:reviewId	DELETE	Remove a review

Endpoints Details

1. Retrieve All Review on a Book

Method : **GET**

Endpoint : **/books/:id/reviews**

Description : **Fetches a list of all available reviews on a book.**

Request Example

```
GET /books/:id/reviews
```

Response Example

- Status: `200 OK`
- Body:


```
{
    "message": "All Reviews",
    "data": {
        "_id": "67ced29b4d3726bee7913f84",
        "title": "Demo Book",
        "author": "67ce9baf1963cc876ad3c2e6",
        "publishedYear": 2023,
        "pages": 120,
        "copiesAvailable": 1,
        "status": "Available",
        "coverImage": "uploads\\BookCover\\10-Mar-2025-05-29-45-book.jpg",
        "reviews": [
            {
                "_id": "67cee49a37d22efa9c2c5101",
                "user": {
                    "_id": "67ce9bf71963cc876ad3c2e9",
                    "name": "Demo User",
                    "email": "demo.user@gmail.com",
                    "role": "member",
                    "profilePicture":
"uploads\\UserProfile\\10-Mar-2025-02-40-24-user.png",
```

```
                    "createdAt": "2025-03-10T07:59:51.674Z",
                    "updatedAt": "2025-03-10T09:10:24.388Z",
                    "__v": 0
                },
                "rating": 4.5,
                "comment": "Demo comment",
                "createdAt": "2025-03-10T13:09:46.650Z",
                "updatedAt": "2025-03-10T13:09:46.650Z",
                "__v": 0
            },
            {
                "_id": "67cee55037d22efa9c2c5108",
                "user": {
                    "_id": "67ce9bf71963cc876ad3c2e9",
                    "name": "Demo User",
                    "email": "demo.user@gmail.com",
                    "role": "member",
                    "profilePicture":
"uploads\\UserProfile\\10-Mar-2025-02-40-24-user.png",
                    "createdAt": "2025-03-10T07:59:51.674Z",
                    "updatedAt": "2025-03-10T09:10:24.388Z",
                    "__v": 0
                },
                "rating": 5,
                "comment": "",
                "createdAt": "2025-03-10T13:12:48.196Z",
                "updatedAt": "2025-03-10T13:12:48.196Z",
                "__v": 0
```

```
            }
        ],
        "createdAt": "2025-03-10T11:52:59.855Z",
        "updatedAt": "2025-03-10T13:12:48.203Z",
        "__v": 0
    }
}
```


---

2. Add a New Review on a Book

Method : **POST**

Endpoint : **/books/:id/reviews**

Description : **Creates a new review entry of a book in the database.**

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description
`user`	string *	The ID of the user who submitted the review.
`rating`	number *	The rating given by the user (minimum: 1, maximum: 5).
`comment`	string	Comment provided by the user.

Request Example

```
{
    "user": "67ce9bf71963cc876ad3c2e9",
    "rating": 4.5,
    "comment": "Demo comment"
}
```

Response Example

- Status: `201 Created`
- Body:

```
{
    "message": "Review Created",
    "data": [
        {
            "user": "67ce9bf71963cc876ad3c2e9",
            "rating": 4.5,
            "comment": "Demo comment",
            "_id": "67cee49a37d22efa9c2c5101",
            "createdAt": "2025-03-10T13:09:46.650Z",
            "updatedAt": "2025-03-10T13:09:46.650Z",
            "__v": 0
        }
    ]
}
```

- Status: `400 Bad Request` (if entity 's body is missing)

```
{
    "error": "\"user\" is required"
}
```

- Status: `400 Bad Request` (if entity 's id is invalid format)

```
{
    "error": "Invalid User ID format"
}
```

---

3. Update Review Details

Method : **PUT**

Endpoint : **/books/:id/reviews/:reviewId**

Description : **Updates an existing review details of a book.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the book.
`reviewId`	string *	The Review ID of a book.

Request Body

Header	Type	Description
`Content-Type`	string	`application/json`

Parameter	Type	Description
`user`	string *	The ID of the user who submitted the review.
`rating`	number *	The rating given by the user (minimum: 1, maximum: 5).
`comment`	string	Comment provided by the user.

Request Example

```
{
    "user": "67ce9bf71963cc876ad3c2e9",
    "rating": 5,
    "comment": "Nice Book"
}
```

Response Example

- Status: `200 OK`
- Body:

```
{
    "message": "Review Updated",
    "data": [
        {
            "_id": "67cee49a37d22efa9c2c5101",
            "user": "67ce9bf71963cc876ad3c2e9",
            "rating": 5,
            "comment": "Nice Book",
            "createdAt": "2025-03-10T13:09:46.650Z",
```

```
            "updatedAt": "2025-03-10T13:25:11.883Z",
            "__v": 0
        }
    ]
}
```

- Status: `404 Not Found` (if entity does not exist)

```
{
    "error": "Review not found"
}
```

- Status: `400 Bad Request (if entity id invalid)`

```
{
    "error": "Review ID invalid"
}
```

- Status: `400 bad request` (if entity's body is missing)

```
{
    "error": "\"user\" is required"
}
```

---

4. Remove a Review

Method : **DELETE**

Endpoint : **/books/:id/reviews/:reviewId**

Description : **Deletes a review of a book from the database.**

Request Parameter

Parameter	Type	Description
`id`	string *	The ID of the book.
`reviewId`	string *	The Review ID of a book.

Request Example

```
DELETE /users/67cee49a37d22efa9c2c5101
```

Response Example

- Status: `200 OK`
- Body:


```
{
    "message": "Review deleted",
    "data": [
        {
            "_id": "67cee49a37d22efa9c2c5101",
            "user": "67ce9bf71963cc876ad3c2e9",
            "rating": 5,
            "comment": "Nice Book",
            "createdAt": "2025-03-10T13:09:46.650Z",
            "updatedAt": "2025-03-10T13:25:11.883Z",
            "__v": 0
        }
    ]
}
```

- Status: `404 Not Found` (if entity does not exist)

```
{
    "error": "Review not found"
}
```

- Status: `400 Bad Request (if entity id invalid)`

```
{
    "error": "Review ID invalid"
}
```

Notes

- Ensure MongoDB is running before using the API.
- Use a tool like Postman to test the API endpoints.
- All responses are in JSON format.