LIBRARY MANAGEMENT SYSTEM
API DOCUMENTATION

This is a sample API documentation for Library Management System. This documentation will provide all details related to the operations performed in a library.

Introduction:

Welcome to the API documentation for the Library Management System (LMS) built with Flask. We have designed LMS seamlessly for managing operations in a library including different resources, such as books, users, borrowing/issuing and returning processes.

Key Features:

1. **User Management:** Register new users and manage user accounts and their accessibility to different resources.
2. **Book Management:** Create, view, update and delete books in the library catalog.
3. **Borrowing/Issuing and Returning:** Facilitates borrowing/issuing and returning books by the users.
4. **Authentication:** Secure endpoints with authentication mechanisms to ensure data privacy and access control.
5. **Authorization:** Secure endpoints with authorization mechanisms for ensuring that only authorized users can perform certain actions.

Purpose:

The Library Management System (LMS) API is designed to provide programmatic access to the core features and the data of the library management application.

The key objectives are depicted below:

- **Resource Management:** Enable CRUD (Create, Read, Update and Delete) operations on various resources, such as users, books inventory and all transactions.
- **Authentication and Authorization:** Implement secure authentication mechanisms for better data privacy and access control, ensuring that only authorized users can perform certain actions.
- **Transaction Handling:** Facilitates the borrowing/issuing and returning of books by the users, as well as the management of overdue items and calculating fines.

Target Audience:

1. **Developers:** Software developers looking forward to integrating the code into their applications or building new solutions for library management systems.
2. **Admins:** Admins will have access to the main endpoints such as registering users, creating roles and permissions, and user’s roles and permissions as well as creating and deleting the books inventory in the library system.
3. **Librarians:** Librarians will have access to the endpoints such as registering students, issuing books to them and initiating return requests, as well as updating the books inventory.
4. **Students:** Students can access their own details regarding issued books, fine and as well as view the books inventory.
5. **Guest Users:** Guest users can only view the books' inventory, i.e., search for available books in the system.

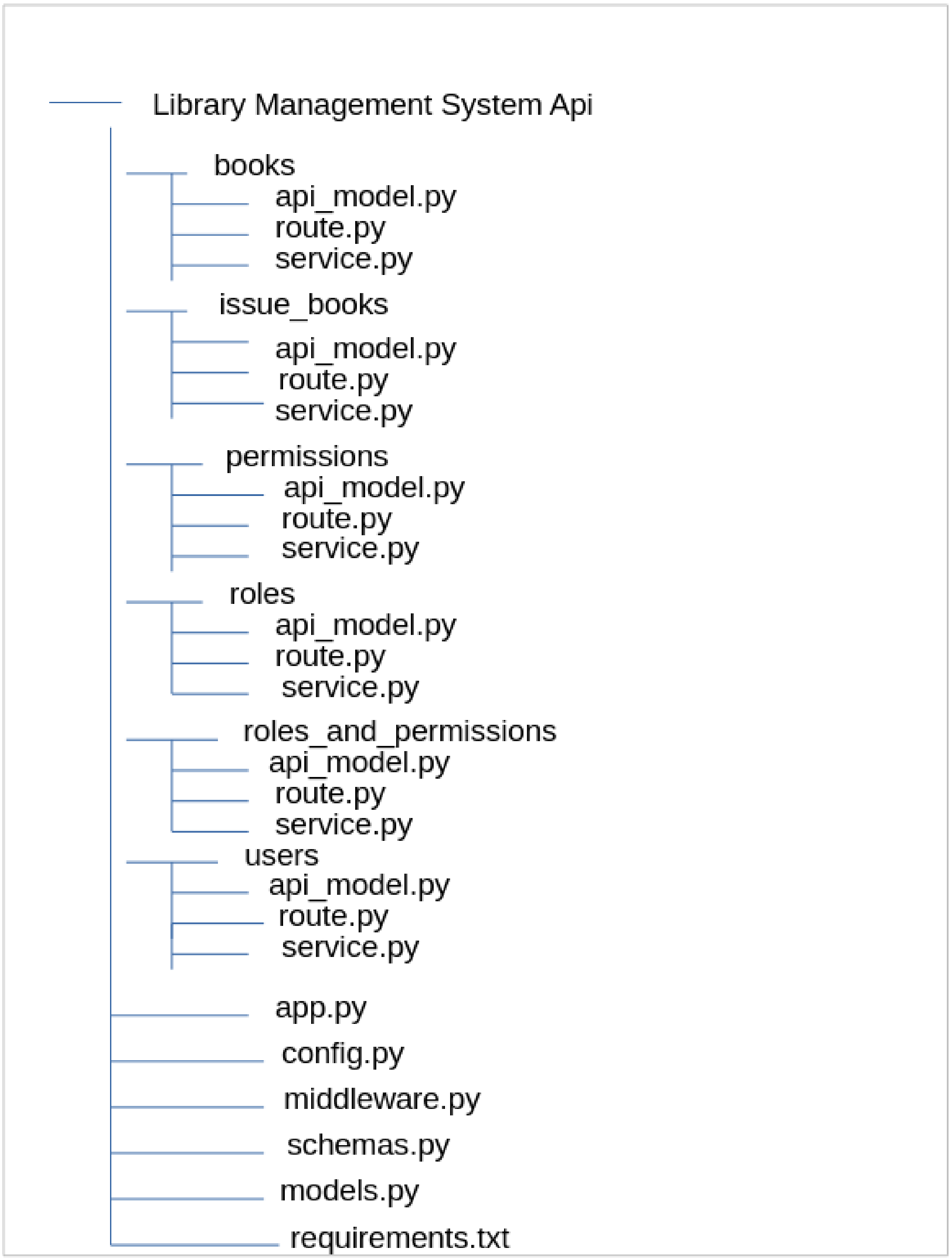
Technologies Used:

The technologies used in the library management system API are listed in the **requirements.txt** file -

- **Flask:** The lightweight and extensible Python web framework used to develop the API.
- **Flask-SQLAlchemy :** The SQL toolkit and Object-Relational Mapping (ORM) library for interacting with the database.
- **Flask-RESTX:** An extension for Flask that adds support for quickly building REST APIs with minimal setup.
- **PyMySQL :** A Pure-Python MySQL client library, which means it is a Python package that creates an API interface to access MySQL relational databases.
- **Flask-Migrate:** An extension that handles SQLAlchemy database migrations for Flask applications using Alembic.
- **Flask-Restful:** An extension for Flask that adds support for building REST APIs in Python using Flask as a backend.
- **Flask-JWT-Extended:** It adds support for using JSON Web Tokens (JWT) to Flask for protecting routes, but also many helpful (also optional) features built in to make working with JSON Web Tokens easier.
- **PyJWT :** A Python library which allows us to encode and decode JSON Web Tokens (JWT).
- **Flask-Marshmallow:** A thin integration layer for Flask (A Python web framework) and marshmallow (an object serialization/deserialization) library that adds additional features, such as URL and Hyperlinks fields.
- **Marshmallow-sqlalchemy :** Marshmallow-sqlalchemy is an integration of Flask-Marshmallow with SQLAlchemy. Marshmallow deserializes dicts to SQLAlchemy models or serializes SQLAlchemy models to dicts. SQLAlchemy maps database schema (tables) and data to Python objects. The two packages complement each other.

Target Api Structure -

The structure of files is depicted below: -



```
Library Management System Api
    books
        api_model.py
        route.py
        service.py
    issue_books
        api_model.py
        route.py
        service.py
    permissions
        api_model.py
        route.py
        service.py
    roles
        api_model.py
        route.py
        service.py
    roles_and_permissions
        api_model.py
        route.py
        service.py
    users
        api_model.py
        route.py
        service.py
    app.py
    config.py
    middleware.py
    schemas.py
    models.py
    requirements.txt
```

GitHub Link:

The Library Management System repository - **https://github.com/AishniNarain/Library-API**

Database Used: MySQL

Middleware:

- Middlewares are created in Flask by creating a decorator; a function can have multiple middlewares, and the order matters a lot.
- We need to add a secret key to our application for implementing authentication.
- I have used middleware in my application for authenticating users and authorizing them based on their access as per their desired roles and permissions.
- The function is simply a decorator function. Inside this function, you check if there is an Authorization field in the headers part of the request; if this is missing, you return an authorization error.
- If everything goes fine, then the view function is called. We return f (*args, **kwargs).
- We have added this middleware (**@access_required([*roles], [*permissions]**)) to every function, we want authenticated users only to access.

Authentication:

- Authentication is a process of verifying that an entity is who they claim to be.
- For example, a user might authenticate by providing a username/email and password. If the username/email and password are valid, the system will check if the user can access the resource. After the system checks the user's details against its database and if the details are valid, the user is thus authenticated and can access available resources.

Authorization:

- Authorization is the process of specifying and enforcing access rights of users to resources.
- It is implemented using middleware based on different roles and permissions.
- The function is simply a decorator function. Inside the function, you check if there is an Authorization field in the headers part of the request; if this is missing, you return an authorization error – **"Unauthorized to access this endpoint"**.

Roles:

- Roles are predefined categories assigned to users based on their job title or other criteria.
- In LMS, users can have three roles in the system. They are-
    - Admin
    - Librarians
    - Students
    - Guest Users

- Role-based Access Control (RBAC) – RBAC is used to manage user roles and permissions. Role information is stored in a table in database and has many-to-many relationships with users.
- Here is a breakdown of different roles assigned to users in the library management system:

1. **Admin:**
   - **Description –**
     - Admins have full control over the library management system.
     - They can register new users, view, update, delete, block and unblock users, view and create books inventory.

   - **Example API Endpoints -**

     - **/api/v1/users/register:** Endpoint for registering new users.
     - **/api/v1/users/{id}:** Endpoint for viewing, updating and deleting users.
     - **/api/v1/users/{id}/(block/unblock):** Endpoint for blocking/unblocking users.
     - **/api/v1/books:** Endpoint for creating and viewing books inventory.

2. **Librarians:**
   - **Description -**
     - Librarians have privileges to manage books inventory and user activities.
     - They can register students, view their/all students' issued book details, delete, block and unblock students, view and update books inventory, issue books, view issued book details and initiate return book requests.
     - Library Management System can have multiple librarians.

   - **Example API Endpoints -**

     - **/api/v1/users/{id}:** Endpoint for viewing their/all student's details and deleting students.
     - **/api/v1/users/{id}/(block/unblock):** Endpoint for blocking/unblocking students.
     - **/api/v1/books:** Endpoint for viewing and updating books inventory.
     - **api/v1/librarian/issue_books :** Endpoint for issuing books to the students.
     - **api/v1/librarian/return_book/<int:id>:** Endpoint for initiating return books request to the students
     - **api/v1/librarian/issued_books :** Endpoint for viewing their own student's issued book details.
     - **api/v1/librarian/all_students/issued_books :** Endpoint for viewing all students' issued book details.
     - **api/v1/librarian/issued_books/history:** Endpoint for viewing issued book history details of the students.

3. Students:
   - **Description -**
     - Students are regular users of the library management system.

- They can search for available books, borrow books and manage their own account details.

- **Example API Endpoints -**

  - /api/v1/books: Endpoint for viewing the books inventory.
  -/api/v1/librarian/issued_books : Endpoint for viewing their own issued book details.

4. Guest Users:

- **Description -**
  - Guest users are irregular users of the library management system.
  - They can only view the books' inventory, i.e., search for available books only.

- **Example API Endpoints -**

  - /api/v1/books: Endpoint for viewing the books inventory.

Permissions:

- Permissions refer to the level of access or rights that a client application or user has to interact with the API's resources or perform certain actions.
- These permissions are enforced by mechanisms such as authentication and authorization.
- **Role-based Permissions:** API's implement Role-based Access Control (RBAC) , where permissions are granted based on the roles of the users or the client application.
- APIs typically require clients to authenticate themselves to enforce permissions. This authentication process verifies the identity of the client (e.g., using API keys, JWT, or other authentication mechanisms) and determines the level of access they are granted based on their permissions.
- Users or applications are assigned roles, and each role has a predefined set of permissions associated with it.
- **Permissions' information is stored in a table in database and has many-to-many relationships with users and their associated roles.**
- In LMS, users have the following permissions -

1. **Register Users:** This permission allows new users to the system to register whether they are librarians or students.

2. **View Users:** This permission allows to view the user details in the library system.

3. **Update Users:** This permission allows us to update the user details in the library system.

4. **Delete Users:** This permission allows you to delete the user details in the library system.

5. **Block Users:** This permission allows to block the users in the library system.

6. **Unblock Users:** This permission allows you to unblock the blocked users in the library system.

7. **View Inventory:** This permission allows to view the books inventory regarding various details like titles, authors, publishers, total and available copies of the books.

8. **Create Inventory:** This permission is used to create the books' inventory.

9. **Update Inventory:** This permission is used to update the details of the book's inventory such as book's title, author, publisher and total copies.

10. **Delete Inventory:** This permission is used to delete the books inventory.
11. **Issue Books:** This permission is used to issue the books to the students in the library system.

12. **View Issued Books:** This permission is used to view the details of the issued books to different students in the library system.

13. **Return Books:** This permission is used to initiate the return of books request in the library system.
14. **View Issued Books History:** This permission is used to view the history of the issued book details of the students in the library system.

JWT (JSON Web Tokens):

- A secure method for authenticating and authorizing users accessing the API endpoints.
- JWT is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- JWT's consist of three parts separated by periods: **header, payload, and signature.**
- For authentication, JWT's are included in the **'Authorization'** header.

- **JWT Generation** -

  - When a user logs in to the library management system, upon successful authentication, JWT's are generated.
  - An **'access token'** and **'refresh token'** are generated using the methods **'create_access_token'** and **'create_refresh_token'** respectively.
  - Access tokens are used for accessing protected resources, while refresh tokens are used to obtain new access tokens once they expire.
  - The identity of the user is embedded within the JWT payload as a claim (e.g., `sub` for subject).
  - This claim typically contains the user's unique identifier, such as a username/email address or a password.
  - When the server receives a request with a JWT, it extracts the identity from the token's payload to identify the user associated with the request.

- JWT’s should be transmitted over HTTPS to encrypt the communication and prevent eavesdropping or man-in-the-middle attacks.

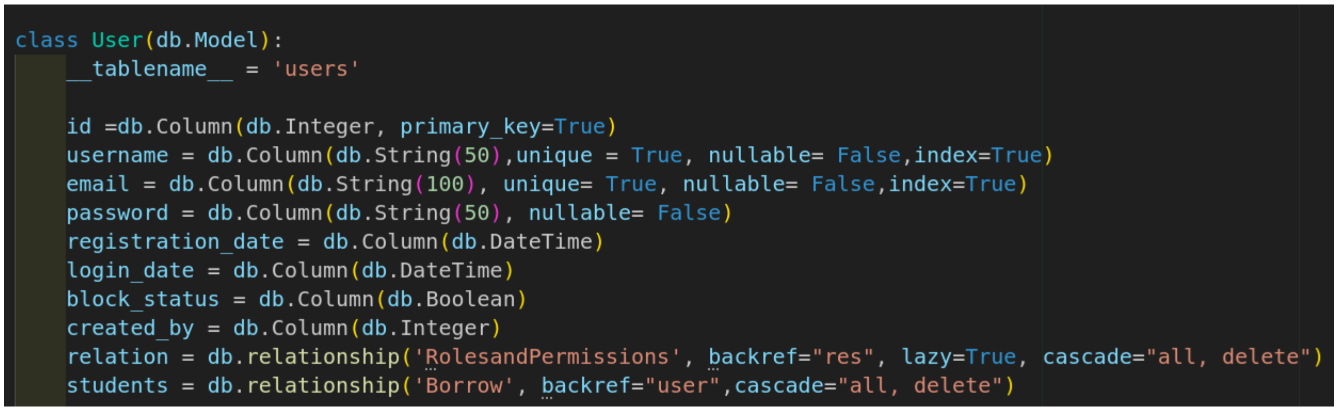
- **JWT’s for Authentication -**

    - JWT’s are generated using cryptographic algorithms, typically asymmetric algorithms like RSA or HMAC (Hash-based Message Authentication Code).
    - Upon successful authentication, the server generates a JWT containing information about the user (known as claims), signs it with a secret key, and sends it back to the client.
    - The client includes this JWT in subsequent requests to authenticate itself with the server.
    - The server verifies the JWT's signature to ensure its authenticity and extracts the user's identity from the token to grant access to protected resources.

Models:

- Models define the structure of data being exchanged (i.e. data being sent or received) in the API.
- They are typically represented as Python classes in Flask application.
- Models represent the data structures exchanged between the client (like a web or a mobile application) and the server (your Flask application).
- We have used following models in our Library Management System (LMS) :

1. User Model -



```
class User(db.Model):
    __tablename__ = 'users'

    id =db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50),unique = True, nullable= False,index=True)
    email = db.Column(db.String(100), unique= True, nullable= False,index=True)
    password = db.Column(db.String(50), nullable= False)
    registration_date = db.Column(db.DateTime)
    login_date = db.Column(db.DateTime)
    block_status = db.Column(db.Boolean)
    created_by = db.Column(db.Integer)
    relation = db.relationship('RolesandPermissions', backref="res", lazy=True, cascade="all, delete")
    students = db.relationship('Borrow', backref="user",cascade="all, delete")
```

- **Description:** User Model is used for representing individual users in the LMS system.
  This model is used for user authentication, registration and other information.
- **Attributes:**

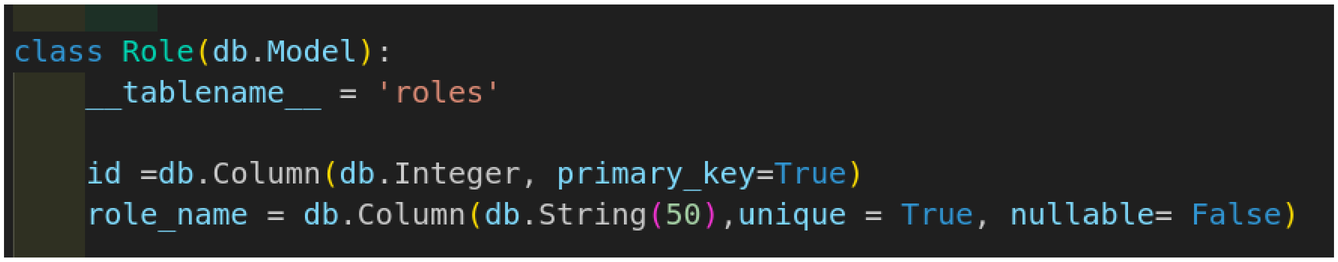
  - id: The unique identifier for each user. It is used as a primary key and is auto-incremented.

- **username:** A string representing the username for each user. It must be unique and is indexed.
- **email:** A string representing the email address for each user. It must be unique and is indexed.
- **password:** An encrypted string for securely storing the password for each user. It is used for authenticating users during login.
- **registration_date :** Date and time when the user was registered within the system.
- **login_date :** Date and time of the user's last login to the system.
- **block_status :** Boolean data representing the status of the user, i.e. either blocked or active.
- **created_by :** The id of the user who created this particular user.

- **Relationships:**

  - **relation:** One-to-many relationship with the `RolesandPermissions` model. This relationship allows us to associate various roles and permissions with each user.
  - **students:** One-to-many relationship with the `Borrow` model. Represents the users who have borrowed items. Cascade delete ensures that associated borrow records are removed when a user is deleted.

2. **Role Model -**



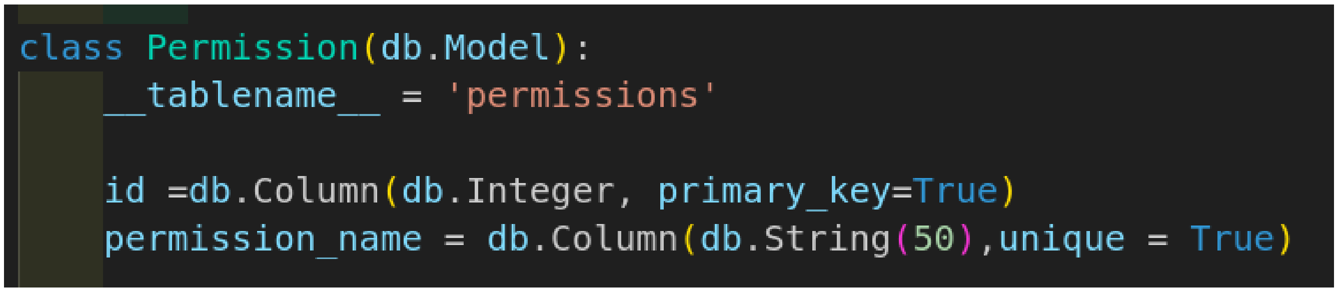
```
class Role(db.Model):
    __tablename__ = 'roles'

    id =db.Column(db.Integer, primary_key=True)
    role_name = db.Column(db.String(50),unique = True, nullable= False)
```

- **Description:** Role model is used for representing the various roles assigned to the users in the system for accessing different endpoints.

- **Attributes:**

  - **id:** The unique identifier for each user representing their role id. It is used as a primary key and is auto incremented.
  - **role_name :** A string representing the role name for each user. It must be unique and indexed.

3. **Permission Model -**

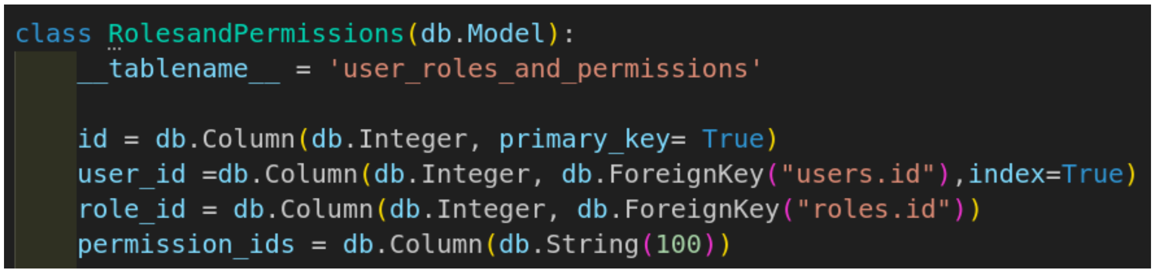

```
class Permission(db.Model):
    __tablename__ = 'permissions'

    id =db.Column(db.Integer, primary_key=True)
    permission_name = db.Column(db.String(50),unique = True)
```

- **Description:** Permission model is used for representing the various permissions assigned to the users in the system for accessing different endpoints.

- **Attributes:**

  - id: The unique identifier for each user representing their permission id. It is used as a primary key and is auto incremented.
  - permission_name : A string representing the permission name for each user. It must be unique and indexed.

4. **RolesandPermissions Model -**


```
class RolesandPermissions(db.Model):
    __tablename__ = 'user_roles_and_permissions'

    id = db.Column(db.Integer, primary_key= True)
    user_id =db.Column(db.Integer, db.ForeignKey("users.id"),index=True)
    role_id = db.Column(db.Integer, db.ForeignKey("roles.id"))
    permission_ids = db.Column(db.String(100))
```

- **Description:** RolesandPermissions model is used for representing the association between users, roles and permissions.

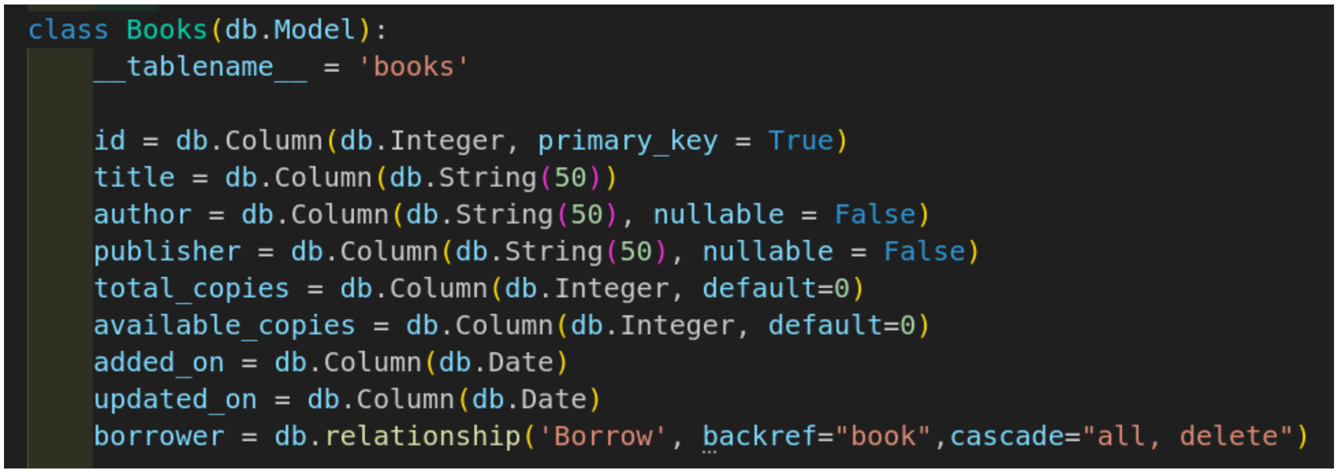
- **Attributes:**

  - id: The unique identifier for the roles and permissions association for each user. It is a primary key and is auto incremented.
  - user_id : The identifier linking to the associated user. It is a foreign key and is indexed.
  - role_id : The identifier linking to the associated role. It is a foreign key.
  - permission_ids : A string containing a list of permission identifiers associated with the roles for each user.

- **Relationships:**

  - user_id : One-to-one relationship with the 'User' model.
  - role_id : One-to-one relationship with the 'Role' model.

5. **Books Model -**



```
class Books(db.Model):
    __tablename__ = 'books'

    id = db.Column(db.Integer, primary_key = True)
    title = db.Column(db.String(50))
    author = db.Column(db.String(50), nullable = False)
    publisher = db.Column(db.String(50), nullable = False)
    total_copies = db.Column(db.Integer, default=0)
    available_copies = db.Column(db.Integer, default=0)
    added_on = db.Column(db.Date)
    updated_on = db.Column(db.Date)
    borrower = db.relationship('Borrow', backref="book",cascade="all, delete")
```

- **Description:** The Books model is used for representing the books inventory available in the Library Management System (LMS).
  This model is used for storing the general information about all the books in the system.

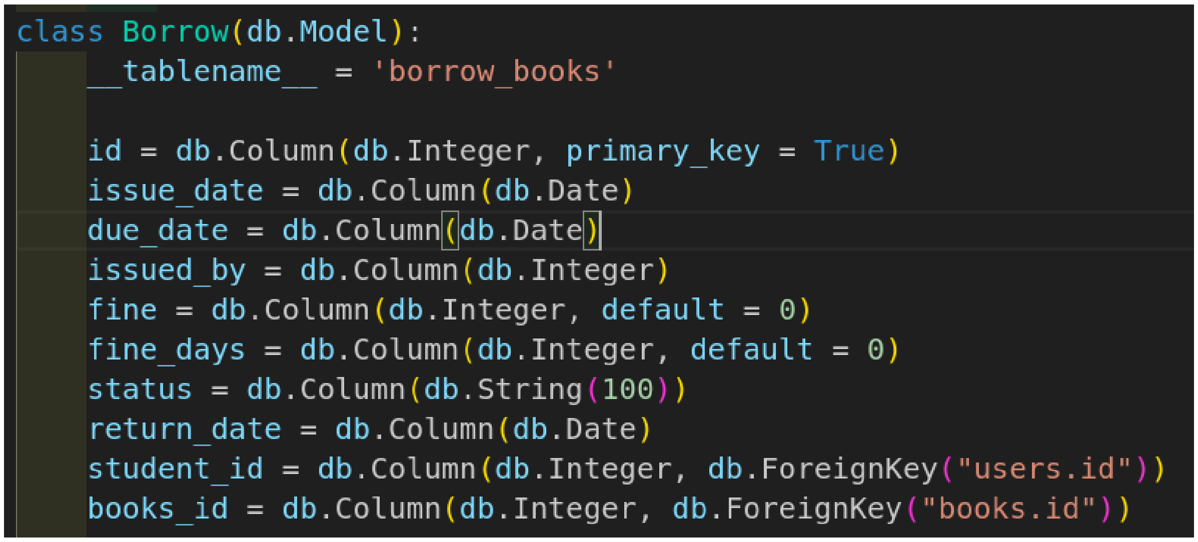
- **Attributes:**

  - id: The unique identifier for each book in the system. It is a primary key and is indexed.
  - title: A string representing the title of the book.
  - author: A string representing the author of the book.
  - publisher: A string representing the publisher of the book.
  - total_copies : The total number of copies available for a particular book.
  - available_copies : The number of copies for a book currently available for borrowing.
  - added_on : The date when the book was added to the system.
  - updated_on : The date when the book's information was last updated.

- **Relationship:**

  - borrower: One-to-many relationship with the 'Borrow' model.

6. **Borrow Model -**



```python
class Borrow(db.Model):
    __tablename__ = 'borrow_books'

    id = db.Column(db.Integer, primary_key = True)
    issue_date = db.Column(db.Date)
    due_date = db.Column(db.Date)
    issued_by = db.Column(db.Integer)
    fine = db.Column(db.Integer, default = 0)
    fine_days = db.Column(db.Integer, default = 0)
    status = db.Column(db.String(100))
    return_date = db.Column(db.Date)
    student_id = db.Column(db.Integer, db.ForeignKey("users.id"))
    books_id = db.Column(db.Integer, db.ForeignKey("books.id"))
```

- **Description:** The Borrow Model is used for representing the borrowing instance when a book is issued to the student in the LMS system.

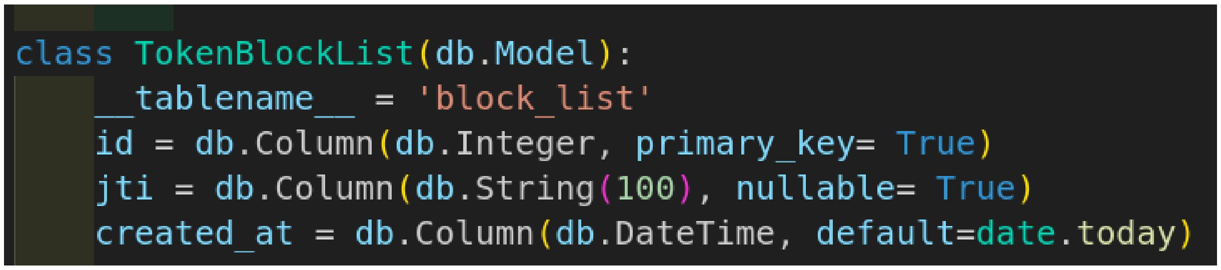
- **Attributes:**

  - **id:** The unique identifier for the borrowing instance for each student. It is a primary key and is auto incremented.
  - **issue_date :** The date when the book was issued to the student.
  - **due_date :** The expected date to return the book.
  - **issued_by :** The id of the librarian who issued the book to the student.
  - **fine:** The fine amount associated with the borrowing instance.
  - **fine_days :** The number of days for which the fine was incurred.
  - **status:** The status of the borrowing instance, (i.e., No Fine, Fine Pending and Return Request Initiated).
  - **return_date :** The return date of the book.
  - **student_id :** The id of the student to whom the book was issued.
  - **books_id :** The id of the book which was issued to the student.

- **Relationships:**

  - **student_id :** One-to-one relationship with the ‘User’ model.
  - **books_id :** One-to-one relationship with the ‘Books’ model.

7. **TokenBlockList Model -**



```
class TokenBlockList(db.Model):
    __tablename__ = 'block_list'
    id = db.Column(db.Integer, primary_key= True)
    jti = db.Column(db.String(100), nullable= True)
    created_at = db.Column(db.DateTime, default=date.today)
```

- **Description:** The TokenBlockList model is used for representing the list of blocked tokens in the system.
  When a user has been logged out of the system, the token is added to the blocked list and cannot be used for authentication or authorization.

- **Attributes:**

  - **id:** The unique identifier for each entry in the token block list.
  - **jti :** The JWT (JSON Web Token) identifier associated with the blocked token.
  - **created_at :** The timestamp when the token was added to the blocked list.

Schemas:

- A schema is used to define the structure and data types of the values in a database or an object.
- Schemas are used for data validation to ensure if the incoming requests follow the desired structure or format.
- We have used Marshmallow to define and enforce schemas, making it easier to work with complex data structures and perform serialization/deserialization tasks.

- Need for schemas:

  1. **Input Validation:** Schemas are used for validating input data by specifying fields, their types and any validation rules, like required fields or minimum/maximum lengths.

  2. **Error Handling:** When a request fails validation against a schema, Flask handles the errors regarding them.

  3. **Security:** Validating input data ensures better security of the application.
  4. **Consistency:** Schemas are used for maintaining data consistency across the API endpoints.

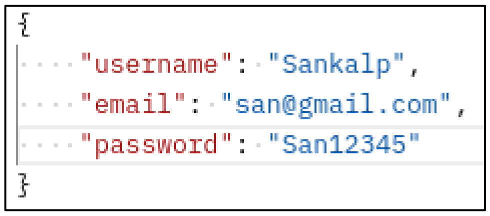
Endpoints -

- **api/v1** namespace is added to every endpoint.

1. Endpoint: /users/register

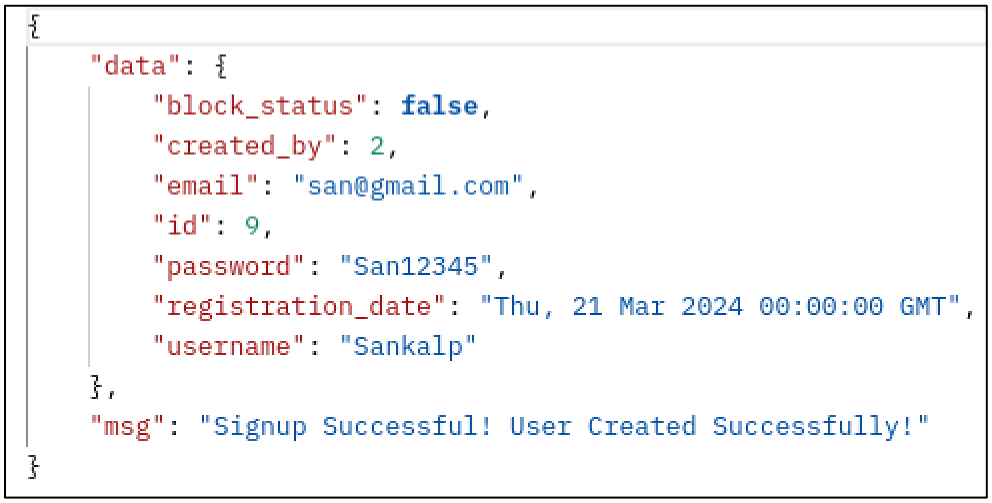
- **Description:** This endpoint is used for registering new users to the system.
- **HTTP Method:** POST
- **URL:** http://127.0.0.1:5000/api/v1/users/register

Example -
Request: POST http://127.0.0.1:5000/api/v1/users/register
Body:



```
{
    "username": "Sankalp",
    "email": "san@gmail.com",
    "password": "San12345"
}
```

Response:
Status: 200 OK
Body:



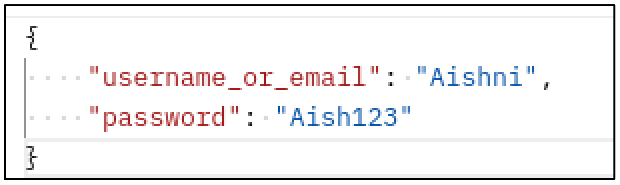
```
{
    "data": {
        "block_status": false,
        "created_by": 2,
        "email": "san@gmail.com",
        "id": 9,
        "password": "San12345",
        "registration_date": "Thu, 21 Mar 2024 00:00:00 GMT",
        "username": "Sankalp"
    },
    "msg": "Signup Successful! User Created Successfully!"
}
```

2. Endpoint: /users/login

- **Description:** This endpoint is used for authenticating users who have registered to the system. We have used **flask_jwt_extended** package to create access and refresh tokens with the help of methods **create_access_token** and **create_refresh_token** respectively.

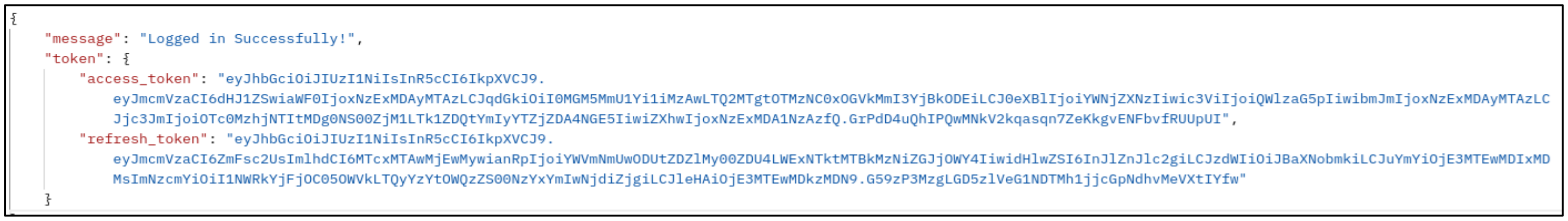
- **HTTP Method:** POST
- **URL:** http://127.0.0.1:5000/api/v1/users/login

Example -
Request: POST http://127.0.0.1:5000/api/v1/users/login
Body:



```
{
    "username_or_email": "Aishni",
    "password": "Aish123"
}
```

Response:
Status: 200 OK
Body:



```
{
    "message": "Logged in Successfully!",
    "token": {
        "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
            eyJmcmVzaCI6dHJ1ZSwiaWF0IjoxNzExMDAyMTAzLCJqdGkiOiI0MGM5MmU1Yi1iMzAwLTQ2MTgtOTMzNC0xOGVkMmI3YjBkODEiLCJ0eXBlIjoiYWNjZXNzIiwic3ViIjoiQWlzaG5pIiwibmJmIjoxNzExMDAyMTAzLC
            Jjc3JmIjoiOTc0MzhjNTItMDg0NS00ZjM1LTk1ZDQtYmIyYTZjZDA4NGE5IiwiZXhwIjoxNzExMDA1NzAzfQ.GrPdD4uQhIPQwMNkV2kqasqn7ZeKkgvENFbvfRUUpUI",
        "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
            eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTcxMTAwMjEwMywianRpIjoiYWVmNmUwODUtZDZlMy00ZDU4LWExNTktMTBkMzNiZGJjOWY4IiwidHlwZSI6InJlZnJlc2giLCJzdWIiOiJBaXNobmkiLCJuYmYiOjE3MTEwMDIxMD
            MsImNzcmYiOiI1NWRkYjFjOC05OWVkLTQyYzYtOWQzZS00NzYxYmIwNjdiZjgiLCJleHAiOjE3MTEwMDkzMDN9.G59zP3MzgLGD5zlVeG1NDTMh1jjcGpNdhvMeVXtIYfw"
    }
}
```

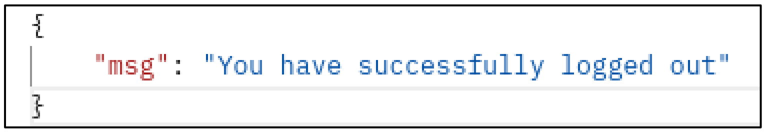
3. Endpoint: /users/logout

- **Description:** This endpoint is used for logging out users from the system, revoking the access token and adding the respective token to the blocked list.
  This ensures that the access token is not valid anymore, which ensures better security for users.

- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/users/logout

Example -
Request: GET http://127.0.0.1:5000/api/v1/users/logout

Response:
Status: 200 OK
Body:



```
{
    "msg": "You have successfully logged out"
}
```

4. Endpoint: /users/profile

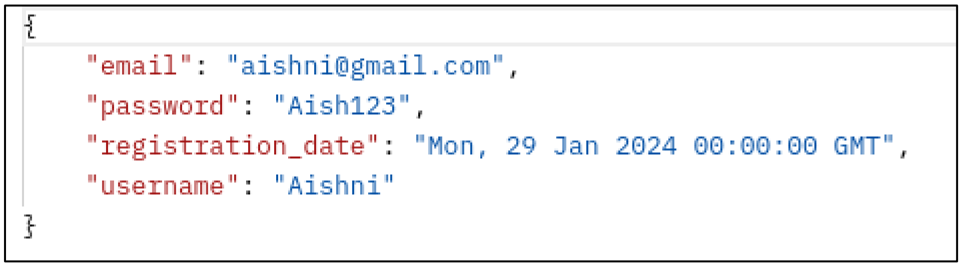
- **Description:** This endpoint is used to display the profile of the user who is currently logged in to the system.

- **HTTP Method:** GET

- **URL:** http://127.0.0.1:5000/api/v1/users/profile

Example -
Request: GET http://127.0.0.1:5000/api/v1/users/profile

Response:
Status: 200 OK
Body:



```
{
    "email": "aishni@gmail.com",
    "password": "Aish123",
    "registration_date": "Mon, 29 Jan 2024 00:00:00 GMT",
    "username": "Aishni"
}
```

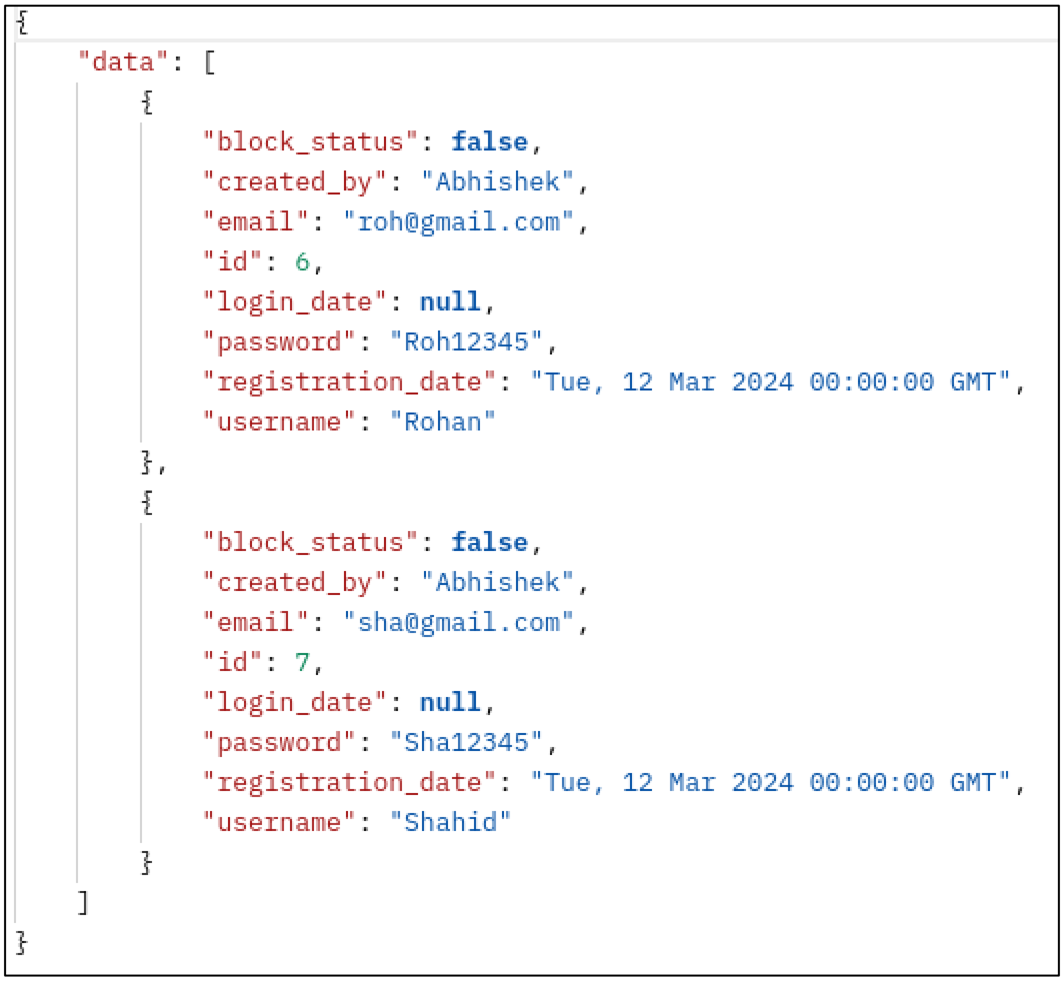
5. Endpoint: /users

- **Description:** This endpoint is used for displaying all the users who have been registered/created by the user who is currently logged in to the system.
  This helps the user view the details of users created by him/her.

- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/users

Example -
Request: GET http://127.0.0.1:5000/api/v1/users

Response:
Status: 200 OK
Body:



```
{
    "data": [
        {
            "block_status": false,
            "created_by": "Abhishek",
            "email": "roh@gmail.com",
            "id": 6,
            "login_date": null,
            "password": "Roh12345",
            "registration_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "username": "Rohan"
        },
        {
            "block_status": false,
            "created_by": "Abhishek",
            "email": "sha@gmail.com",
            "id": 7,
            "login_date": null,
            "password": "Sha12345",
            "registration_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "username": "Shahid"
        }
    ]
}
```

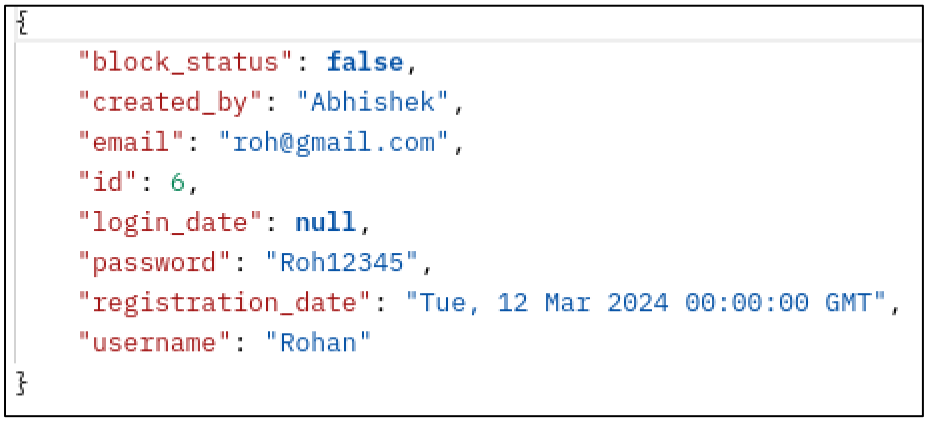
6. Endpoint: /users/<int:id>

- **Description:** This endpoint is used to display the details of a particular user by their unique id (specified as path parameter) who has been registered/created by the user who is currently logged in to the system.

- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/users/{id}

Path Parameters:
{id} (required, integer): The unique identifier of the user.

Example -
Request: GET http://127.0.0.1:5000/api/v1/users/6
Response:
Status: 200 OK
Body:



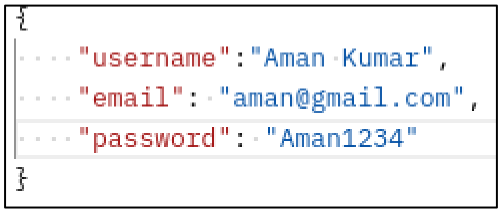
```json
{
    "block_status": false,
    "created_by": "Abhishek",
    "email": "roh@gmail.com",
    "id": 6,
    "login_date": null,
    "password": "Roh12345",
    "registration_date": "Tue, 12 Mar 2024 00:00:00 GMT",
    "username": "Rohan"
}
```

7. Endpoint: /users/<int:id>

- **Description:** This endpoint updates the user by their unique id (specified as path parameter) and passing the parameters in the request body which must be updated.
- **HTTP Method:** PUT
- **URL:** http://127.0.0.1:5000/api/v1/users/{id}

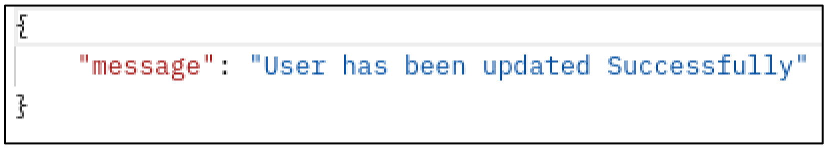
Path Parameters:
{id} (required, integer): The unique identifier of the user.

Example -
Request: PUT http://127.0.0.1:5000/api/v1/users/2
Body:



```json
{
    "username":"Aman Kumar",
    "email": "aman@gmail.com",
    "password": "Aman1234"
}
```

Response:
Status: 200 OK
Body:



```json
{
    "message": "User has been updated Successfully"
}
```

8. Endpoint: /users/<int:id>

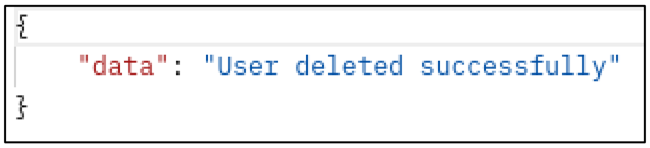
- **Description:** This endpoint deletes the user by their unique id (specified as path parameter) except those who have been issued books.

- **HTTP Method:** DELETE
- **URL:** http://127.0.0.1:5000/api/v1/users/{id}

Path Parameters:
{id} (required, integer): The unique identifier of the user.

Example -
Request: DELETE http://127.0.0.1:5000/api/v1/users/8

Response:
Status: 200 OK
Body:



```
{
    "data": "User deleted successfully"
}
```

9. Endpoint: /users/block/<int:id>

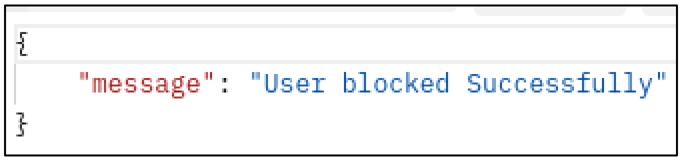
- **Description:** This endpoint blocks the user by their unique id (specified as path parameter) except those who have been issued books.

- **HTTP Method:** PATCH
- **URL:** http://127.0.0.1:5000/api/v1/users/{id}/block

Path Parameters:
{id} (required, integer): The unique identifier of the user.

Example -
Request: PATCH http://127.0.0.1:5000/api/v1/users/block/2

Response:
Status: 200 OK
Body:



```
{
    "message": "User blocked Successfully"
}
```

10. Endpoint: /users/unblock/<int:id>

- **Description:** This endpoint is used to unblock the particular user by their unique id (specified as path parameter).

- **HTTP Method:** PATCH

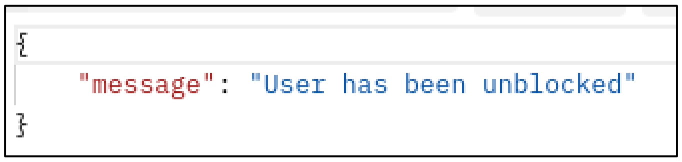
- **URL:** http://127.0.0.1:5000/api/v1/users/{id}/unblock

  Path Parameters:

  {id} (required, integer): The unique identifier of the user.

Example -
Request: PATCH http://127.0.0.1:5000/api/v1/users/unblock/2

Response:
Status: 200 OK
Body:



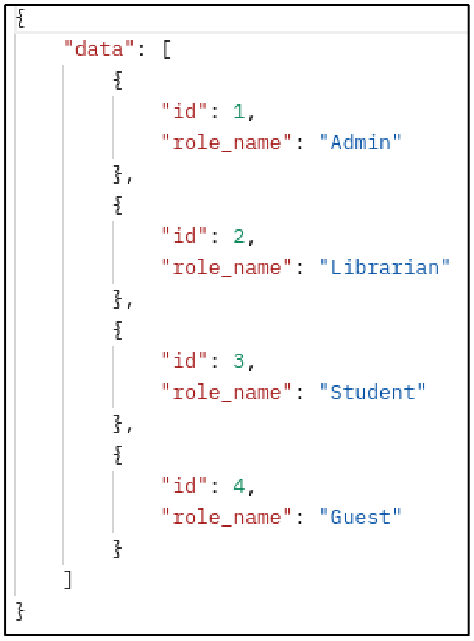
```
{
    "message": "User has been unblocked"
}
```

11. Endpoint: /roles

- **Description:** This endpoint displays the roles which will be assigned to different users.
- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/roles

Example -
Request: GET http://127.0.0.1:5000/api/v1/roles

Response:
Status: 200 OK
Body:



```
{
    "data": [
        {
            "id": 1,
            "role_name": "Admin"
        },
        {
            "id": 2,
            "role_name": "Librarian"
        },
        {
            "id": 3,
            "role_name": "Student"
        },
        {
            "id": 4,
            "role_name": "Guest"
        }
    ]
}
```

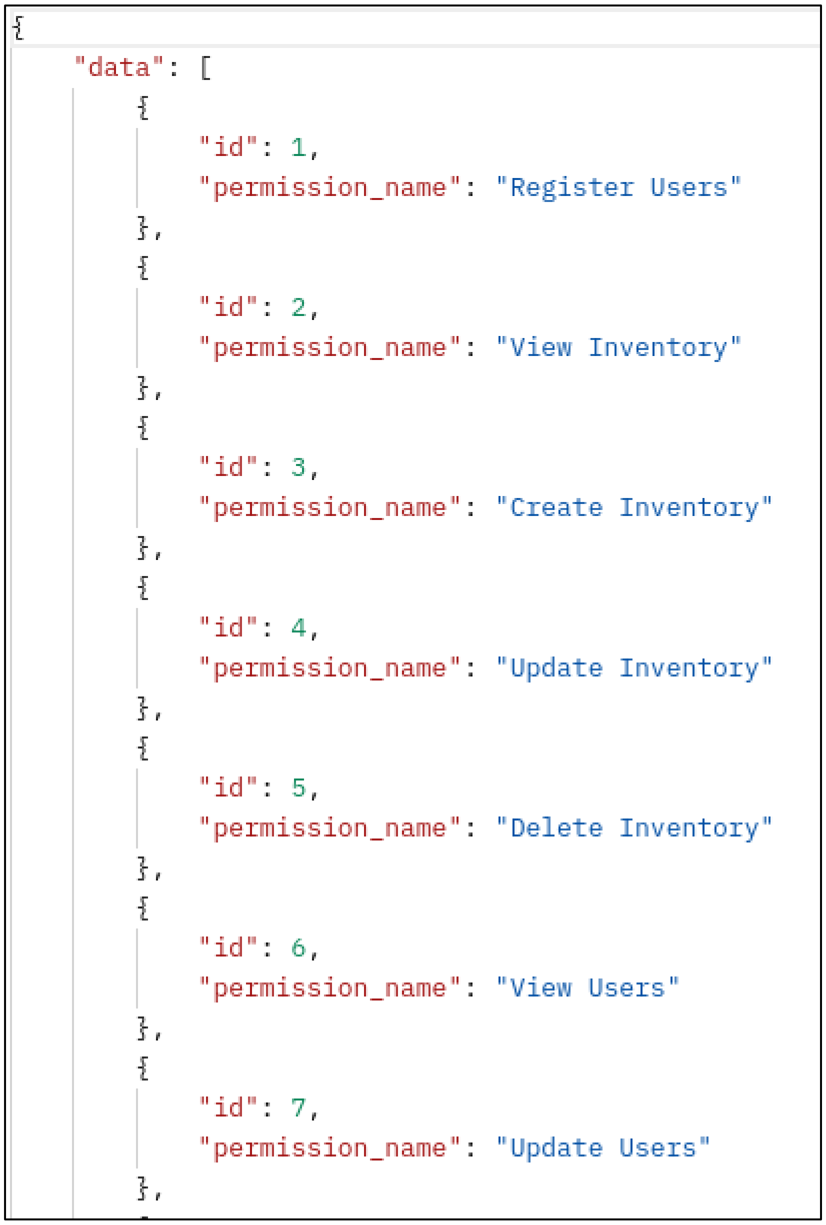
12. Endpoint: /permissions

- **Description:** This endpoint displays the permissions which will be assigned to different users.

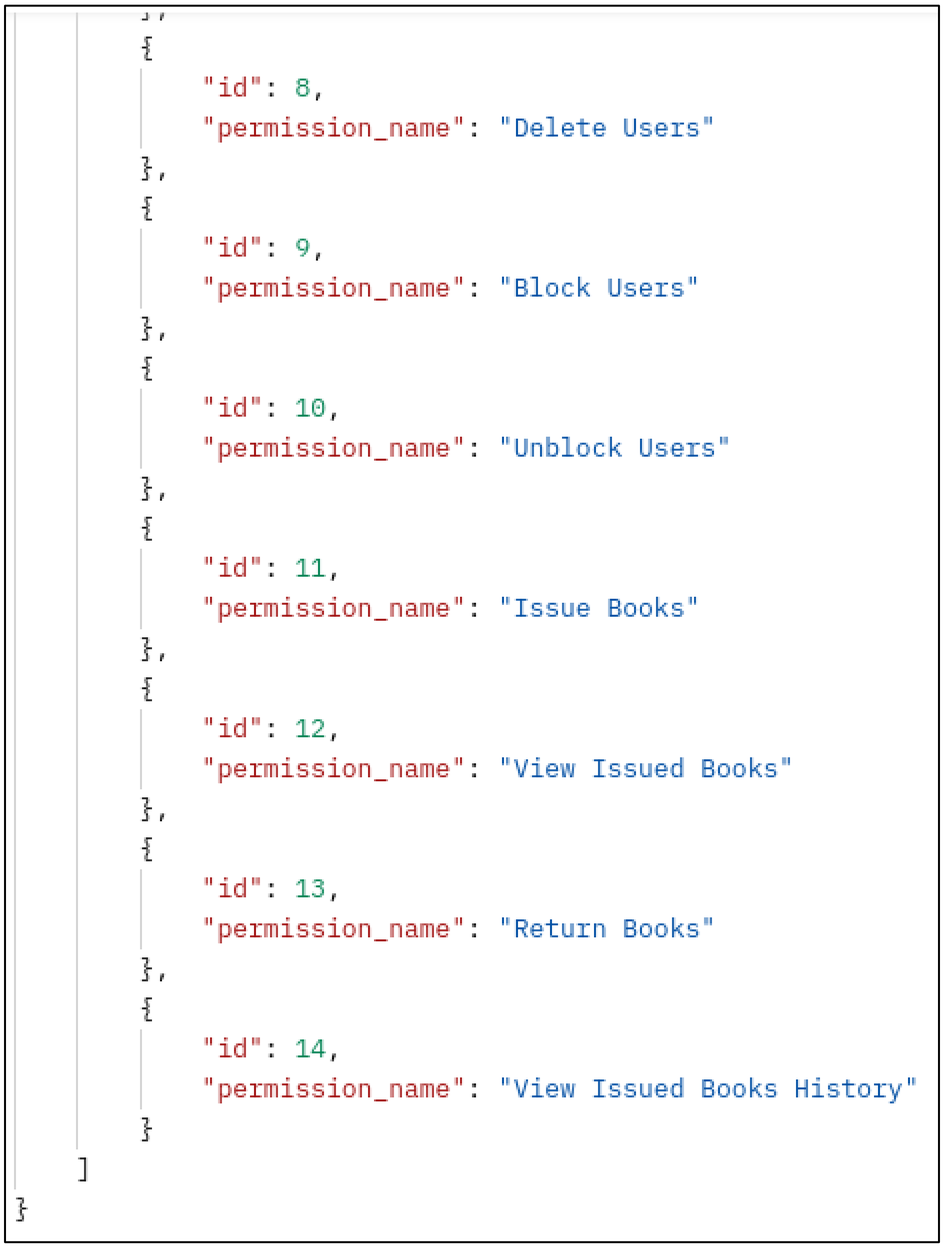
- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/permissions

Example -
Request: GET http://127.0.0.1:5000/api/v1/permissions

Response:
Status: 200 OK
Body :



```
{
    "data": [
        {
            "id": 1,
            "permission_name": "Register Users"
        },
        {
            "id": 2,
            "permission_name": "View Inventory"
        },
        {
            "id": 3,
            "permission_name": "Create Inventory"
        },
        {
            "id": 4,
            "permission_name": "Update Inventory"
        },
        {
            "id": 5,
            "permission_name": "Delete Inventory"
        },
        {
            "id": 6,
            "permission_name": "View Users"
        },
        {
            "id": 7,
            "permission_name": "Update Users"
        },
```



```
        {
            "id": 8,
            "permission_name": "Delete Users"
        },
        {
            "id": 9,
            "permission_name": "Block Users"
        },
        {
            "id": 10,
            "permission_name": "Unblock Users"
        },
        {
            "id": 11,
            "permission_name": "Issue Books"
        },
        {
            "id": 12,
            "permission_name": "View Issued Books"
        },
        {
            "id": 13,
            "permission_name": "Return Books"
        },
        {
            "id": 14,
            "permission_name": "View Issued Books History"
        }
    ]
}
```

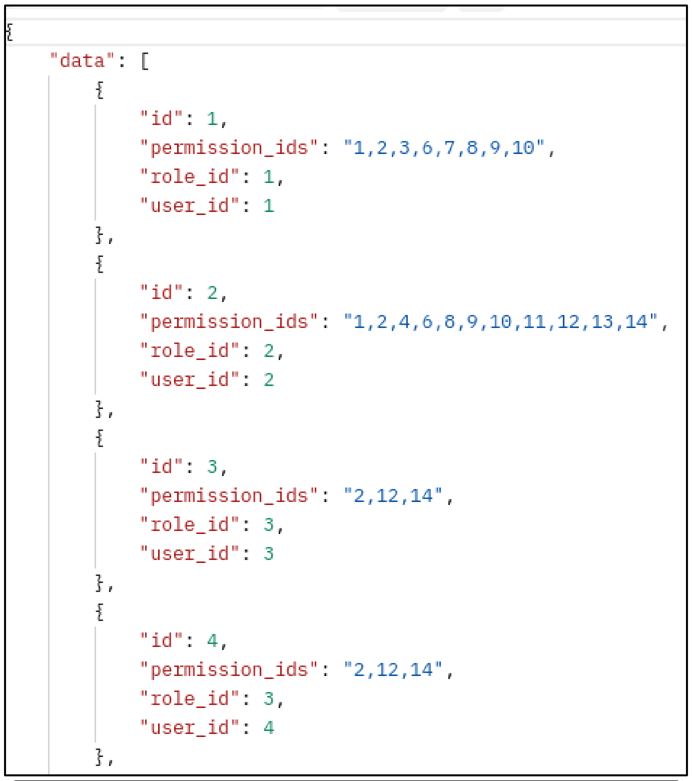
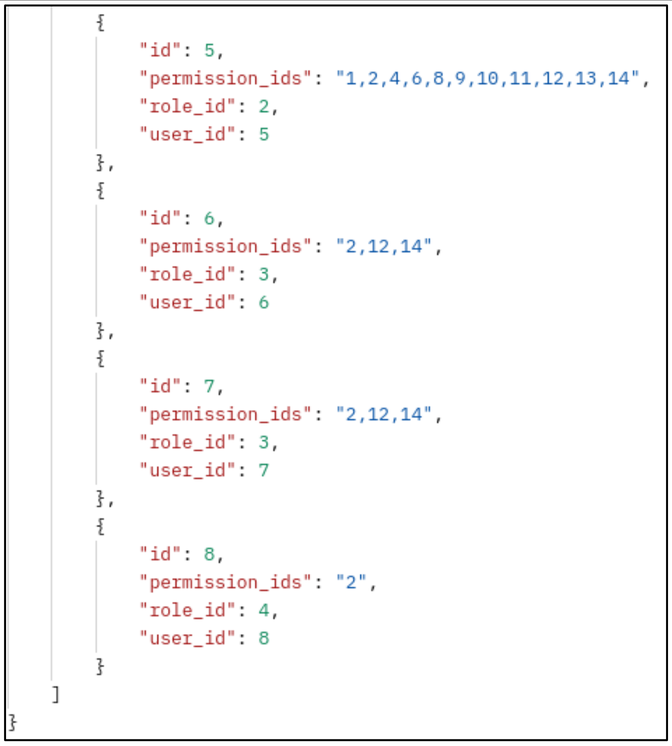
13. Endpoint: /user_rolesandpermissions

- **Description:** This endpoint is used to display the roles and permissions assigned to different users.
- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/user_rolesandpermissions

Example -
Request: GET http://127.0.0.1:5000/api/v1/user_rolesandpermissions

Response:
Status: 200 OK

Body:

```
{
    "data": [
        {
            "id": 1,
            "permission_ids": "1,2,3,6,7,8,9,10",
            "role_id": 1,
            "user_id": 1
        },
        {
            "id": 2,
            "permission_ids": "1,2,4,6,8,9,10,11,12,13,14",
            "role_id": 2,
            "user_id": 2
        },
        {
            "id": 3,
            "permission_ids": "2,12,14",
            "role_id": 3,
            "user_id": 3
        },
        {
            "id": 4,
            "permission_ids": "2,12,14",
            "role_id": 3,
            "user_id": 4
        },
        {
            "id": 5,
            "permission_ids": "1,2,4,6,8,9,10,11,12,13,14",
            "role_id": 2,
            "user_id": 5
        },
        {
            "id": 6,
            "permission_ids": "2,12,14",
            "role_id": 3,
            "user_id": 6
        },
        {
            "id": 7,
            "permission_ids": "2,12,14",
            "role_id": 3,
            "user_id": 7
        },
        {
            "id": 8,
            "permission_ids": "2",
            "role_id": 4,
            "user_id": 8
        }
    ]
}
```

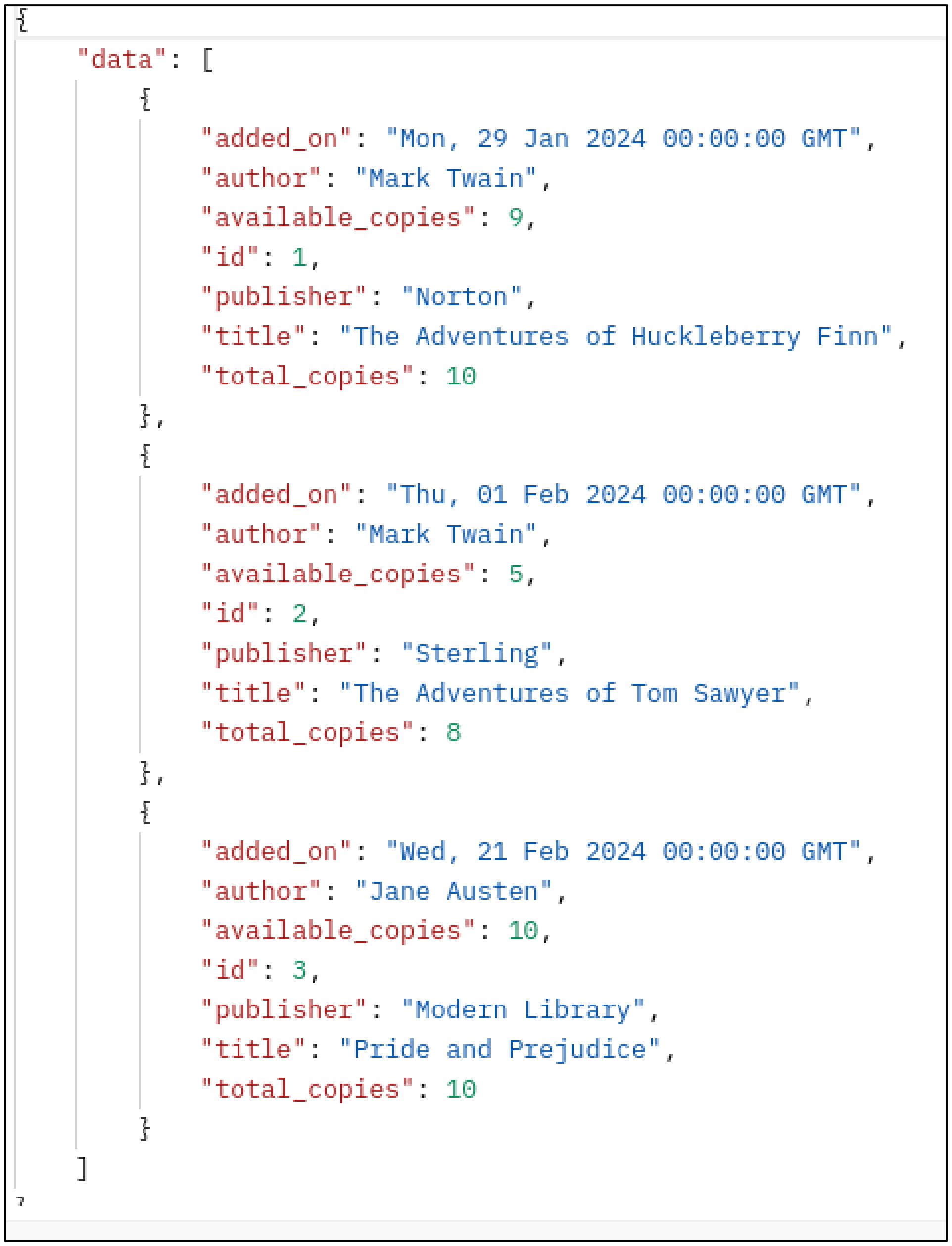
14. Endpoint: /books

- **Description:** This endpoint is used to display all the books available in the books inventory.
- **HTTP Method:** GET

- **URL:** http://127.0.0.1:5000/api/v1/books

Example -
Request: GET http://127.0.0.1:5000/api/v1/books

Response:
Status: 200 OK
Body :



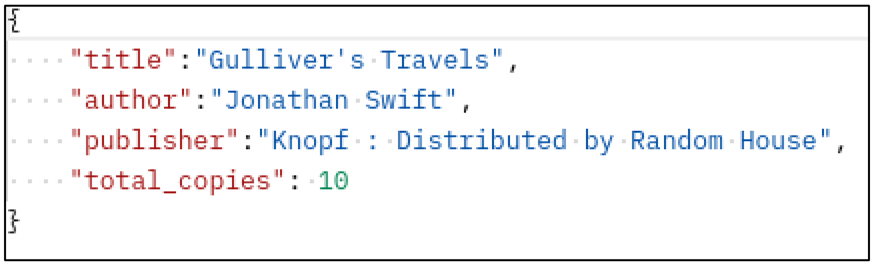
```
{
    "data": [
        {
            "added_on": "Mon, 29 Jan 2024 00:00:00 GMT",
            "author": "Mark Twain",
            "available_copies": 9,
            "id": 1,
            "publisher": "Norton",
            "title": "The Adventures of Huckleberry Finn",
            "total_copies": 10
        },
        {
            "added_on": "Thu, 01 Feb 2024 00:00:00 GMT",
            "author": "Mark Twain",
            "available_copies": 5,
            "id": 2,
            "publisher": "Sterling",
            "title": "The Adventures of Tom Sawyer",
            "total_copies": 8
        },
        {
            "added_on": "Wed, 21 Feb 2024 00:00:00 GMT",
            "author": "Jane Austen",
            "available_copies": 10,
            "id": 3,
            "publisher": "Modern Library",
            "title": "Pride and Prejudice",
            "total_copies": 10
        }
    ]
}
```

15. Endpoint: /books

- **Description:** This endpoint is used to create new books by passing parameters in the request body.

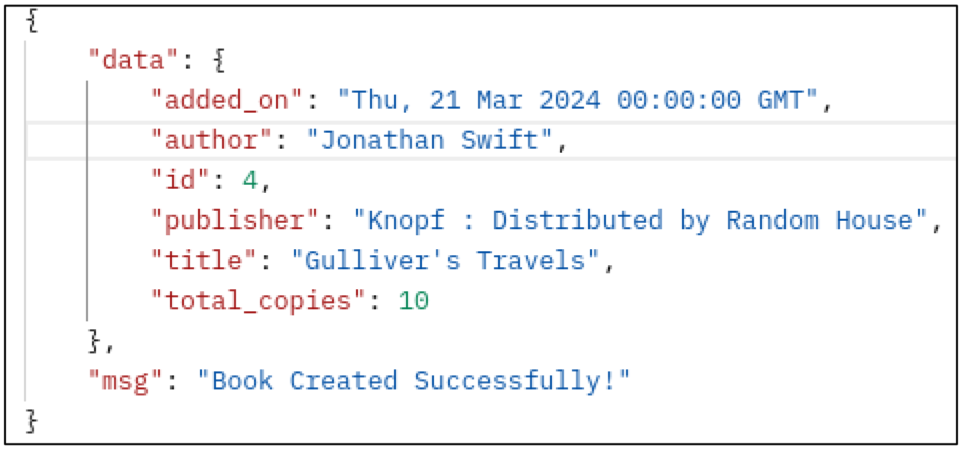
- **HTTP Method:** POST
- **URL:** http://127.0.0.1:5000/api/v1/books

Example -
Request: POST http://127.0.0.1:5000/api/v1/books
Body:



```
{
    "title":"Gulliver's Travels",
    "author":"Jonathan Swift",
    "publisher":"Knopf : Distributed by Random House",
    "total_copies": 10
}
```

Response:
Status: 200 OK
Body :



```
{
    "data": {
        "added_on": "Thu, 21 Mar 2024 00:00:00 GMT",
        "author": "Jonathan Swift",
        "id": 4,
        "publisher": "Knopf : Distributed by Random House",
        "title": "Gulliver's Travels",
        "total_copies": 10
    },
    "msg": "Book Created Successfully!"
}
```

16. Endpoint: /books/<int:id>

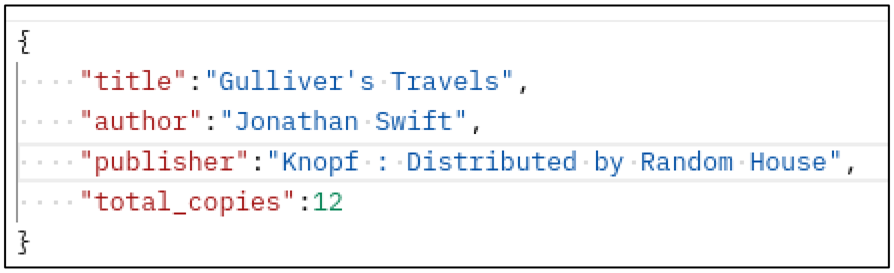
- **Description:** This endpoint is used to update the particular book by its unique id (specified as path parameter) and passing the parameters in the request body which are required to be updated.

- **HTTP Method:** PUT
- **URL:** http://127.0.0.1:5000/api/v1/books/{id}

Path Parameters:

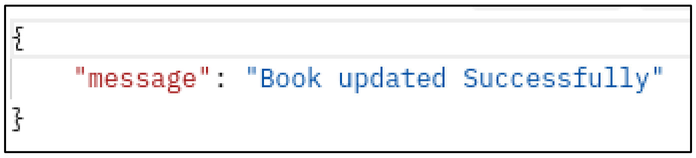
{id} (required, integer): The unique identifier of the user.

Example -
Request: PUT http://127.0.0.1:5000/api/v1/books/4
Body:



```
{
    "title":"Gulliver's Travels",
    "author":"Jonathan Swift",
    "publisher":"Knopf : Distributed by Random House",
    "total_copies":12
}
```

Response:
Status: 200 OK
Body:



```
{
    "message": "Book updated Successfully"
}
```

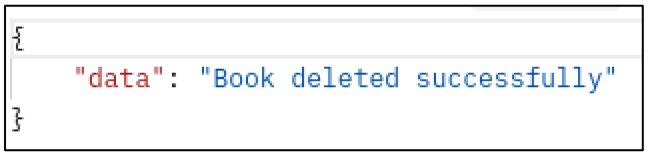
17. Endpoint: /books/<int:id>

- **Description:** This endpoint deletes the book by its unique id (specified as path parameter).
- **HTTP Method:** DELETE
- **URL:** http://127.0.0.1:5000/api/v1/books/{id}

Path Parameters:
{id} (required, integer): The unique identifier of the user.

Example -
Request: DELETE http://127.0.0.1:5000/api/v1/books/4

Response:
Status: 200 OK
Body:

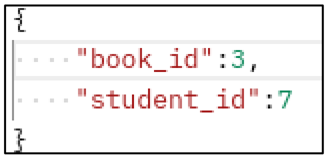


```
{
    "data": "Book deleted successfully"
}
```

18. Endpoint: /librarian/issue_books

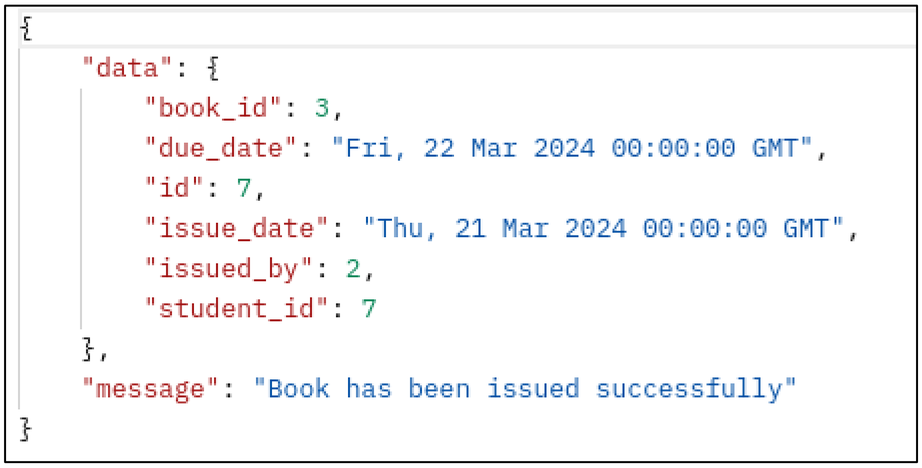
- **Description:** This endpoint is used to issue the books to the students by the librarian.
- **HTTP Method:** POST
- **URL:** http://127.0.0.1:5000/api/v1/librarian/issue_books

Example -
Request: POST http://127.0.0.1:5000/api/v1/librarian/issue_books
Body:



```
{
    "book_id":3,
    "student_id":7
}
```

Response:
Status: 200 OK
Body:



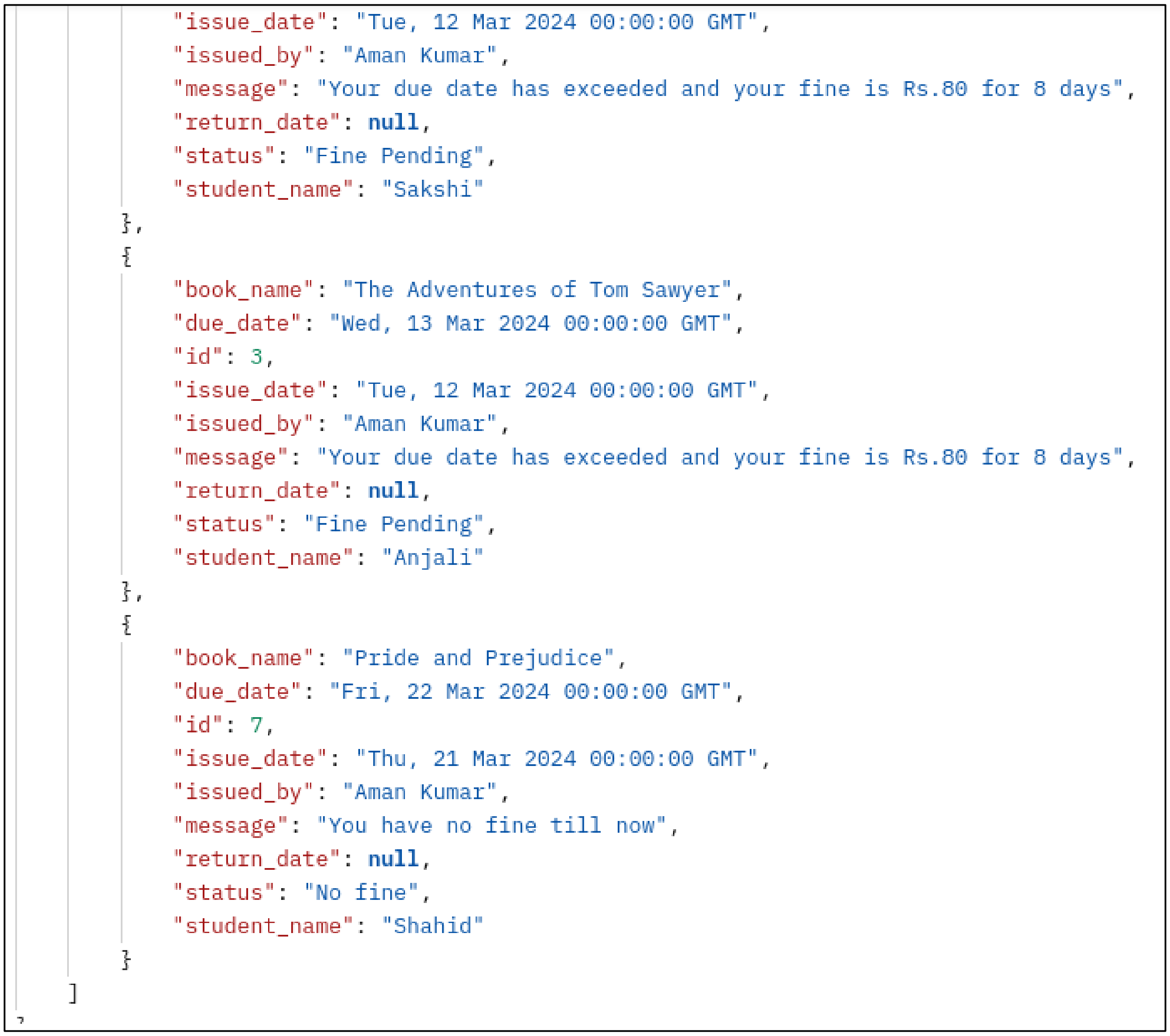
```
{
    "data": {
        "book_id": 3,
        "due_date": "Fri, 22 Mar 2024 00:00:00 GMT",
        "id": 7,
        "issue_date": "Thu, 21 Mar 2024 00:00:00 GMT",
        "issued_by": 2,
        "student_id": 7
    },
    "message": "Book has been issued successfully"
}
```

19. **Endpoint:** /librarian/issued_books

- **Description:** This endpoint is used to display the issued book details to the librarians (who have issued books) and students (to whom the books are issued).
- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/librarian/issued_books

Example -
Request: GET http://127.0.0.1:5000/api/v1/librarian/issued_books

Response:
Status: 200 OK
Body:



```
            "issue_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "issued_by": "Aman Kumar",
            "message": "Your due date has exceeded and your fine is Rs.80 for 8 days",
            "return_date": null,
            "status": "Fine Pending",
            "student_name": "Sakshi"
        },
        {
            "book_name": "The Adventures of Tom Sawyer",
            "due_date": "Wed, 13 Mar 2024 00:00:00 GMT",
            "id": 3,
            "issue_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "issued_by": "Aman Kumar",
            "message": "Your due date has exceeded and your fine is Rs.80 for 8 days",
            "return_date": null,
            "status": "Fine Pending",
            "student_name": "Anjali"
        },
        {
            "book_name": "Pride and Prejudice",
            "due_date": "Fri, 22 Mar 2024 00:00:00 GMT",
            "id": 7,
            "issue_date": "Thu, 21 Mar 2024 00:00:00 GMT",
            "issued_by": "Aman Kumar",
            "message": "You have no fine till now",
            "return_date": null,
            "status": "No fine",
            "student_name": "Shahid"
        }
    ]
,
```

20. Endpoint: /librarian/return_book/<int:id>

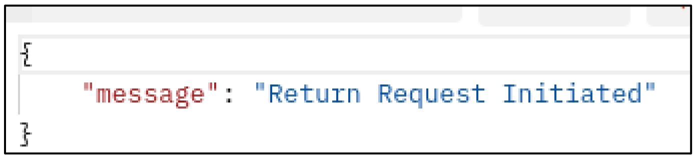
- **Description:** This endpoint is used to return the book of the particular student by its unique id (specified as path parameter) by the librarian.
- **HTTP Method:** PATCH
- **URL:** http://127.0.0.1:5000/api/v1/librarian/return_book/{id}

Path Parameters:
{id} (required, integer): The unique identifier of the user.

Example -
Request: PATCH http://127.0.0.1:5000/api/v1/librarian/issued_books

Response:

Status: 200 OK
Body:



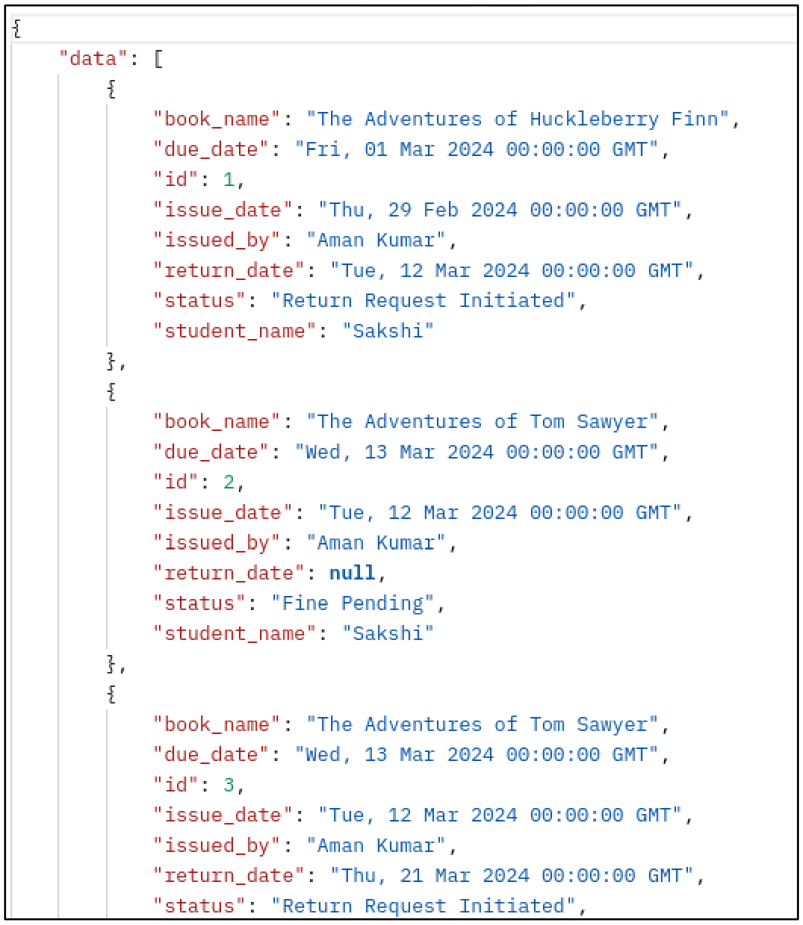
```
{
    "message": "Return Request Initiated"
}
```

21. Endpoint: /librarian/all_students/issued_books

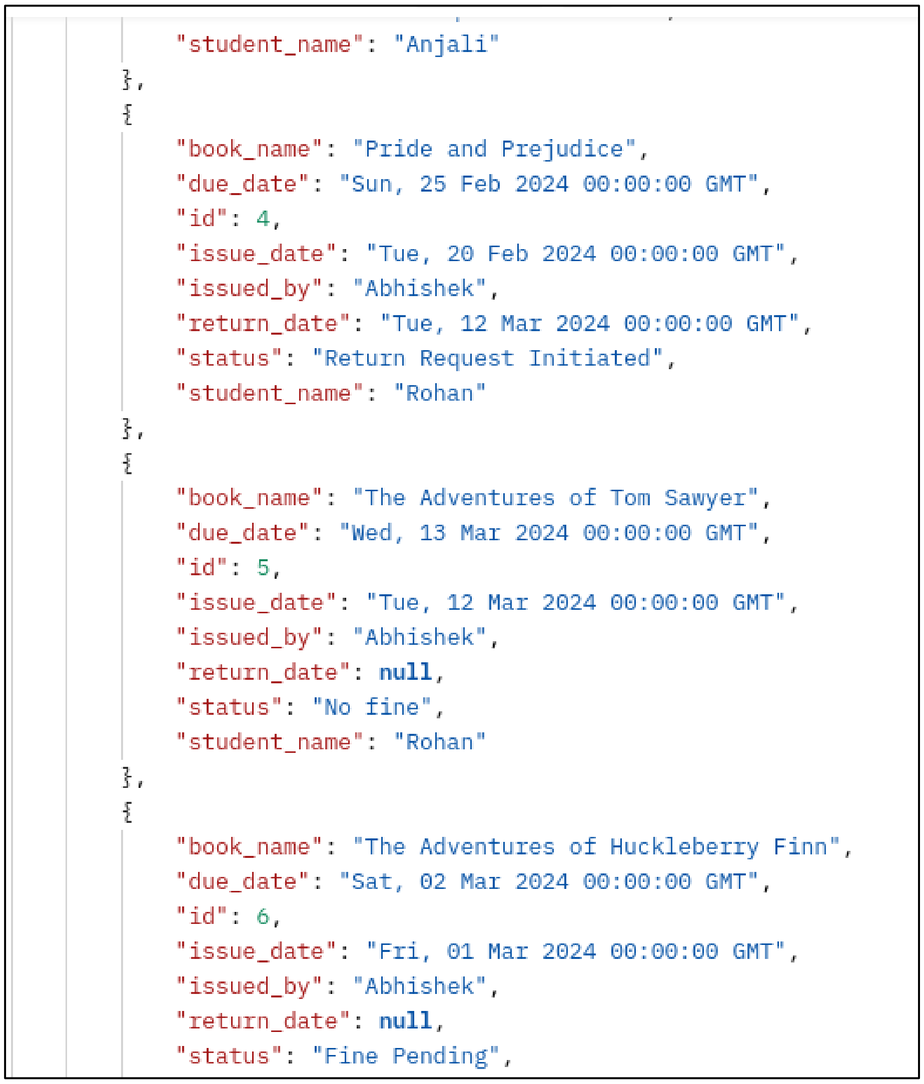
- **Description:** This endpoint is used to display the issued book details of all students to all the librarians in the system.
- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/librarian/all_students/issued_books

Example -
Request: GET http://127.0.0.1:5000/api/v1/librarian/all_students/issued_books

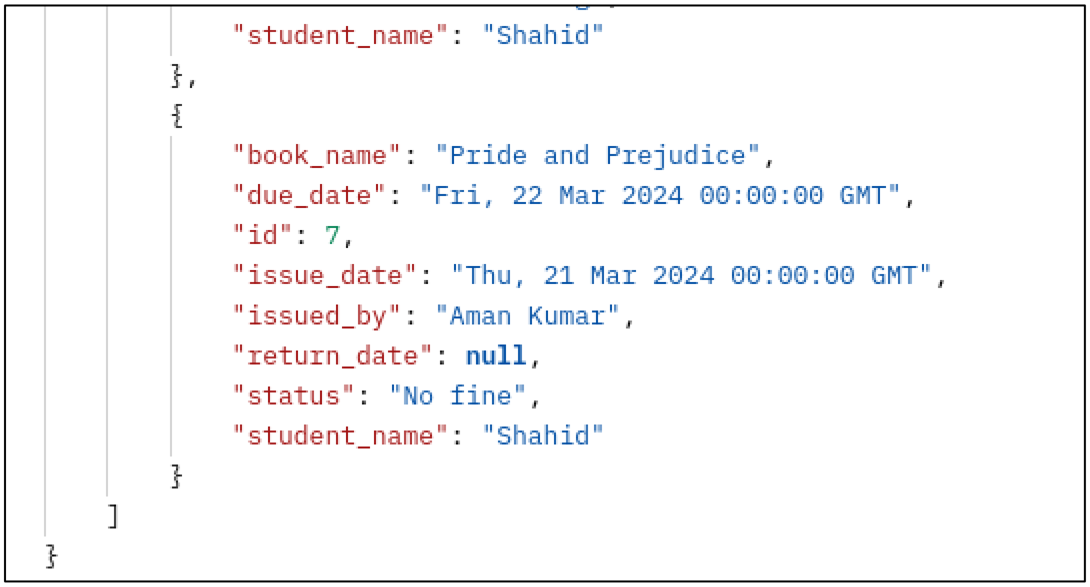
Response:
Status: 200 OK
Body:



```
{
    "data": [
        {
            "book_name": "The Adventures of Huckleberry Finn",
            "due_date": "Fri, 01 Mar 2024 00:00:00 GMT",
            "id": 1,
            "issue_date": "Thu, 29 Feb 2024 00:00:00 GMT",
            "issued_by": "Aman Kumar",
            "return_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "status": "Return Request Initiated",
            "student_name": "Sakshi"
        },
        {
            "book_name": "The Adventures of Tom Sawyer",
            "due_date": "Wed, 13 Mar 2024 00:00:00 GMT",
            "id": 2,
            "issue_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "issued_by": "Aman Kumar",
            "return_date": null,
            "status": "Fine Pending",
            "student_name": "Sakshi"
        },
        {
            "book_name": "The Adventures of Tom Sawyer",
            "due_date": "Wed, 13 Mar 2024 00:00:00 GMT",
            "id": 3,
            "issue_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "issued_by": "Aman Kumar",
            "return_date": "Thu, 21 Mar 2024 00:00:00 GMT",
            "status": "Return Request Initiated",
```



```
            "student_name": "Anjali"
        },
        {
            "book_name": "Pride and Prejudice",
            "due_date": "Sun, 25 Feb 2024 00:00:00 GMT",
            "id": 4,
            "issue_date": "Tue, 20 Feb 2024 00:00:00 GMT",
            "issued_by": "Abhishek",
            "return_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "status": "Return Request Initiated",
            "student_name": "Rohan"
        },
        {
            "book_name": "The Adventures of Tom Sawyer",
            "due_date": "Wed, 13 Mar 2024 00:00:00 GMT",
            "id": 5,
            "issue_date": "Tue, 12 Mar 2024 00:00:00 GMT",
            "issued_by": "Abhishek",
            "return_date": null,
            "status": "No fine",
            "student_name": "Rohan"
        },
        {
            "book_name": "The Adventures of Huckleberry Finn",
            "due_date": "Sat, 02 Mar 2024 00:00:00 GMT",
            "id": 6,
            "issue_date": "Fri, 01 Mar 2024 00:00:00 GMT",
            "issued_by": "Abhishek",
            "return_date": null,
            "status": "Fine Pending",
```



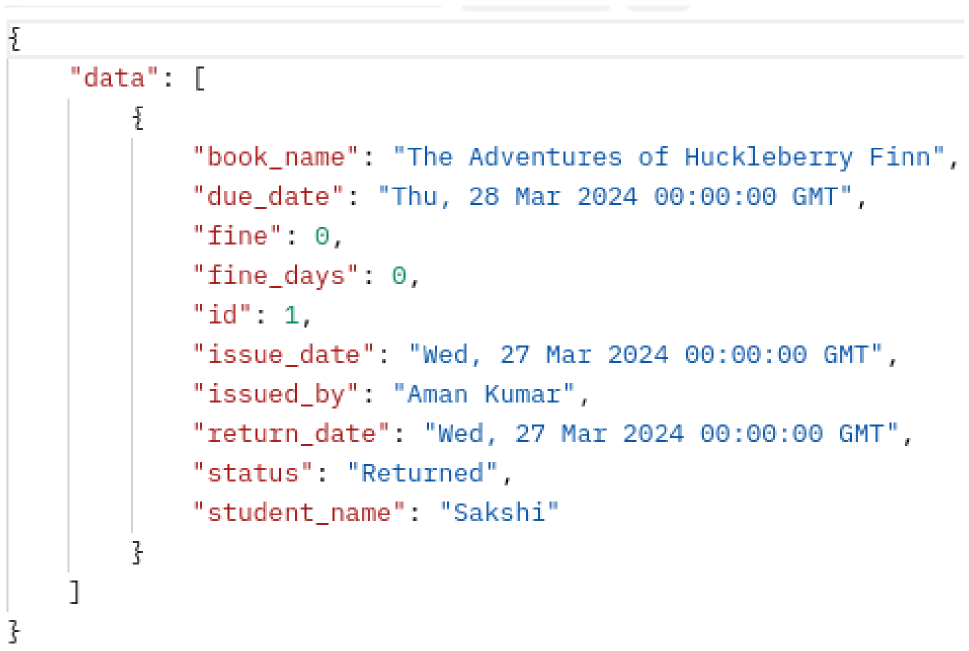
```
            "student_name": "Shahid"
        },
        {
            "book_name": "Pride and Prejudice",
            "due_date": "Fri, 22 Mar 2024 00:00:00 GMT",
            "id": 7,
            "issue_date": "Thu, 21 Mar 2024 00:00:00 GMT",
            "issued_by": "Aman Kumar",
            "return_date": null,
            "status": "No fine",
            "student_name": "Shahid"
        }
    ]
}
```

22. Endpoint: /issued_books/history

- **Description:** This endpoint is used to see the historical details of issued books of particular students by the librarians.
- **HTTP Method:** GET
- **URL:** http://127.0.0.1:5000/api/v1/issued_books/history

Example -
Request: GET http://127.0.0.1:5000/api/v1/issued_books/history

Response:
Status: 200 OK
Body:



```
{
    "data": [
        {
            "book_name": "The Adventures of Huckleberry Finn",
            "due_date": "Thu, 28 Mar 2024 00:00:00 GMT",
            "fine": 0,
            "fine_days": 0,
            "id": 1,
            "issue_date": "Wed, 27 Mar 2024 00:00:00 GMT",
            "issued_by": "Aman Kumar",
            "return_date": "Wed, 27 Mar 2024 00:00:00 GMT",
            "status": "Returned",
            "student_name": "Sakshi"
        }
    ]
}
```