Library Management System Documentation

For: Maids.cc

Content List

- **Data Base relations**
- **Authentication Controller**
- **Book Controller**
- **Patron Controller**
- **Borrowing Records Controller**
- **Security**
- **Examples For End Points**
- **Aspect Oriented Programming**
- **Unit Testing**
- **Transactional**
- **Swagger Documentation**
- **Caching**
- **Error Handling & Validations**

By : Omar Mohamed Elaraby

- **Data Base Relations:**

  -Database my sql has been used for the project

  -It is called library and it has 4 tables:

    - Book
    - Patron
    - Borrow Book Record
    - User

-Book has 5 columns:

- Id (**Primary Key**)
- Author (**Not null**)
- Title (**Not null**)
- Publication Year (**Not null**)
- ISBN (**Not null & Unique**)

-Patron has 4 columns:

- Id (**Primary Key**)
- Name (**Not null**)
- Mobile (**Not null**)
- Email (**Not null & Unique**)

- Borrow Book Record has 5 columns:

- Id (**Primary Key**)
- Book_Id (**Not null**)
- Patron_Id (**Not null**)
- Borrow Date (**Not null**)
- Return Date
- Book_Id , Patron_Id , Borrow Date , Return Date (**Unique**)

-User has 5 columns:

- Id (**Primary Key**)
- First name (**Not null**)
- Last name (**Not null**)
- Email (**Not null , Unique**)
- Password (**Not null**)

- Borrow Book Record has many to one relationship with Books and

many to one relationship with Patrons

-User table is created for authentication and authorization so it has no relationships with other table.

- **Authentication Controller:**

End Point Name	Request Type	Path	Request Model	Response Model
Add New User	Post	/api/auth/register	RegisterReqModel	AuthenticationResModel
Authenticate User	Post	/api/auth/authenticate	AuthenticationReqModel	AuthenticationResModel

- **Book Controller:**

End Point Name	Request Type	Path	Request Model	Response Model
Get All Books	Get	/api/books?PageNo=&PageSize=	----------	BookResModel
Create New Book	Post	/api/books	BookReqModel	Integer
Get Book By Id	Get	/api/books/{id}	----------	BookResModel
Update Book	Put	/api/books/{id}	BookReqModel	BookResModel
Delete Book	Delete	/api/books/{id}	----------	----------

- **Patron Controller:**

End Point Name	Request Type	Path	Request Model	Response Model
Get All Patrons	Get	/api/Patrons? PageNo=&PageSize=	----------	PatronsResModel
Create New Patron	Post	/api/ Patrons	PatronsReqModel	Integer
Get Patron By Id	Get	/api/ Patrons /{id}	----------	PatronsResModel
Update Patron	Put	/api/ Patrons /{id}	PatronsReqModel	PatronsResModel
Delete Patron	Delete	/api/ Patrons /{id}	----------	----------

- **Borrowing Records Controller:**

End Point Name	Request Type	Path	Request Model	Response Model
Create New Borrowing Book Record	Post	/api/borrow/{bookId} /patron/{patronId}	BorrowReqModel	Integer
Create New Book Return Record	Put	/api/borrow/{bookId} /patron/{patronId}	Return ReqModel	Void

- **Security:**

  -Creating authentication and authorization using **JWT bearer token**.

  -The dependency "**io.jsonwebtoken**" used for implementing security for the project

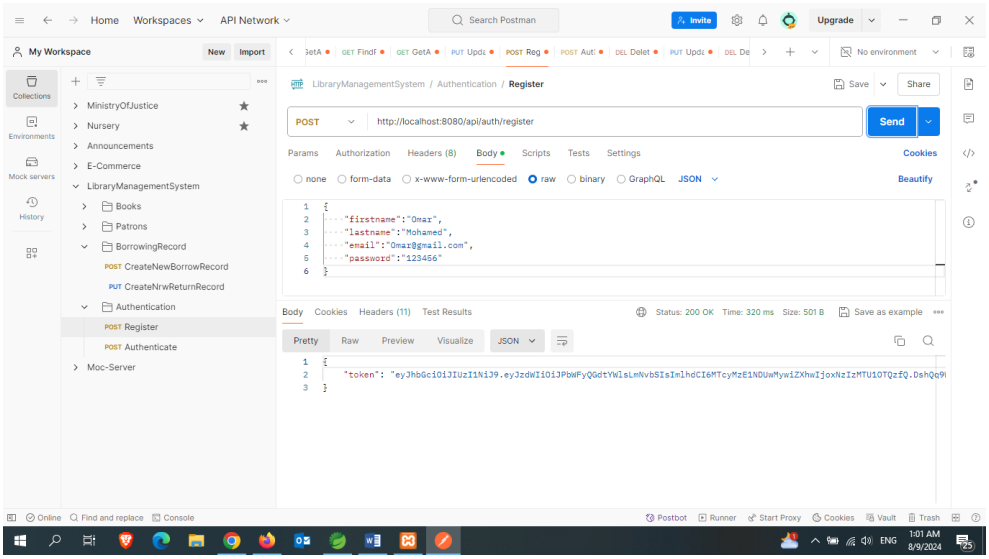
  -it is divided in to 2 steps:

    - Register user (**Sign up**)
    - Authenticate user (**Log in**)

  -The authorization is only for 1 user so, can access for all methods.

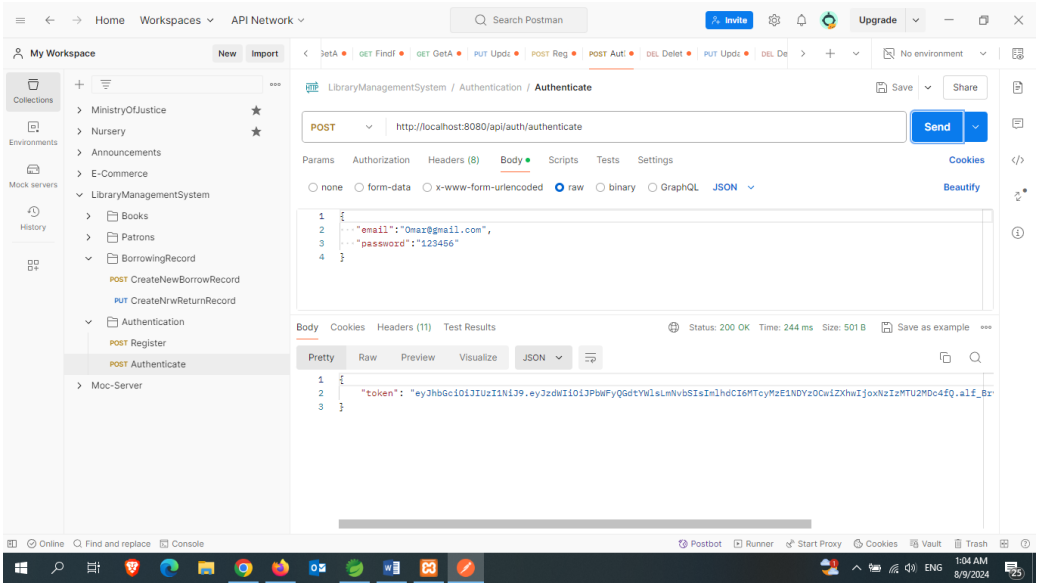
- **Examples For End Points:**

1. Register user



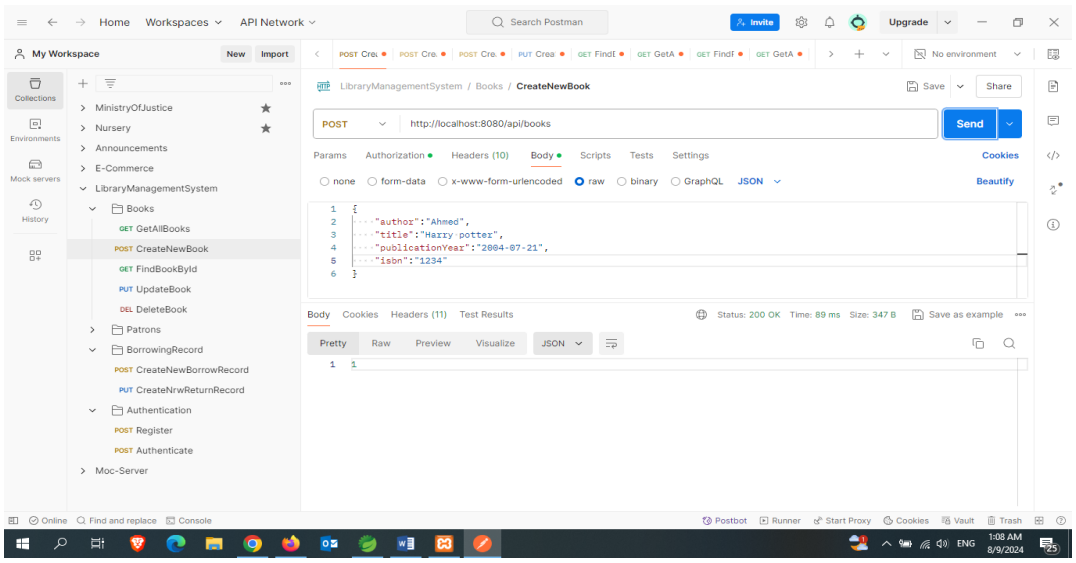
-This token is not used for authentication but it represents that the user registered successfully

2. Authenticate user

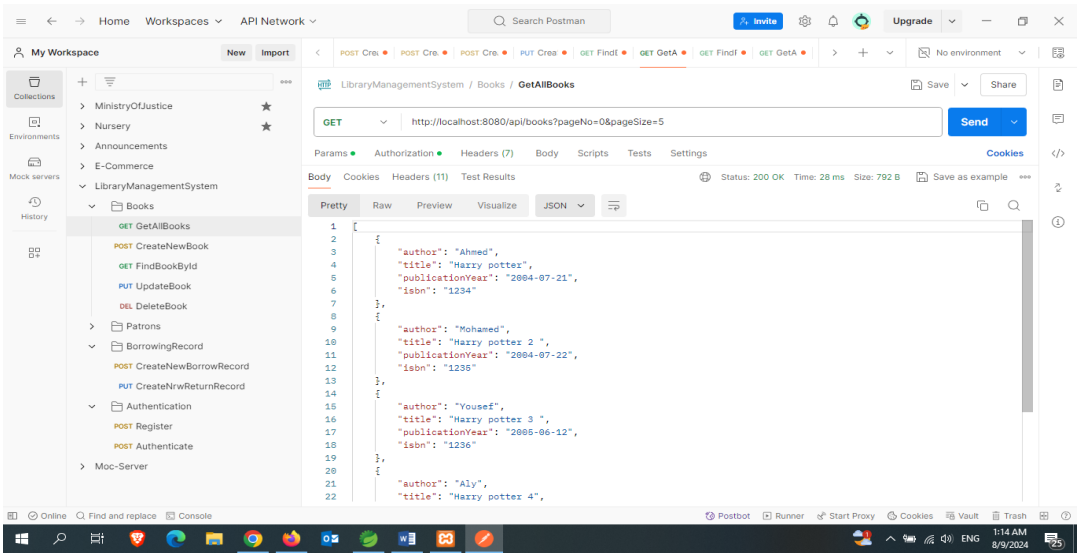


-This token is used for user authentication. It will be put in authorization tab in postman as bearer token.

3. Add Book

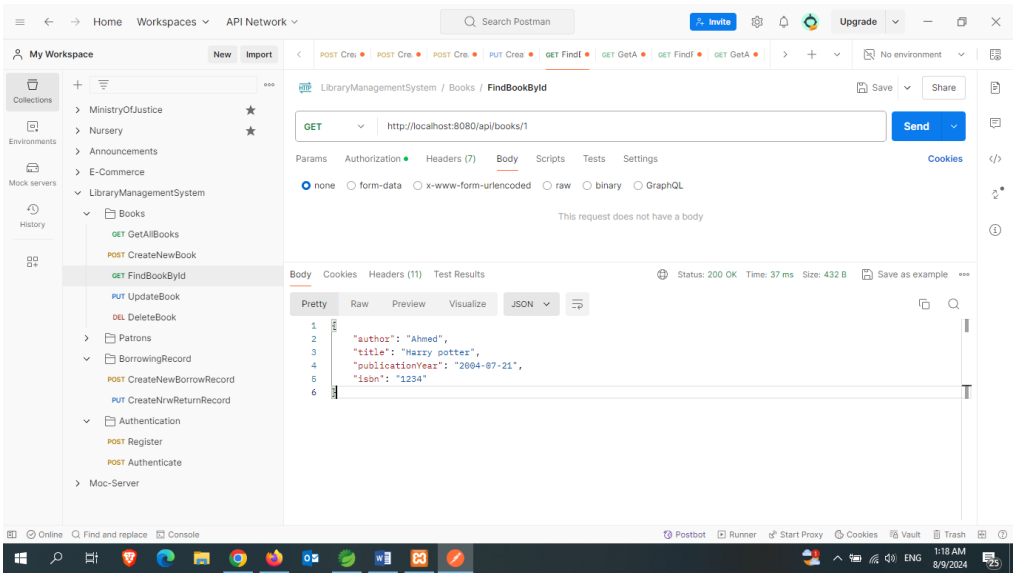


4. Get All Books

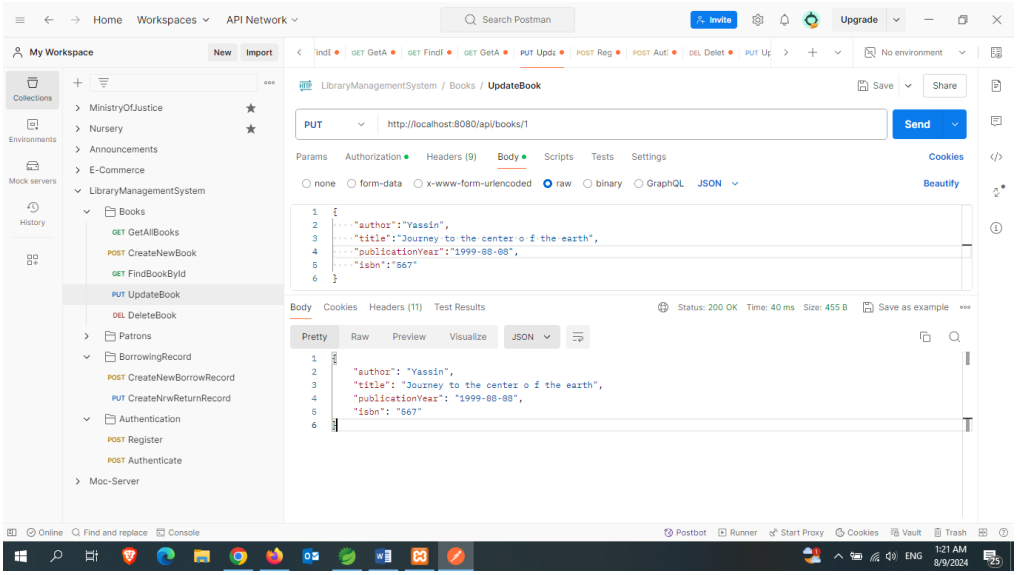


-Using of pagination and page size to control number of data

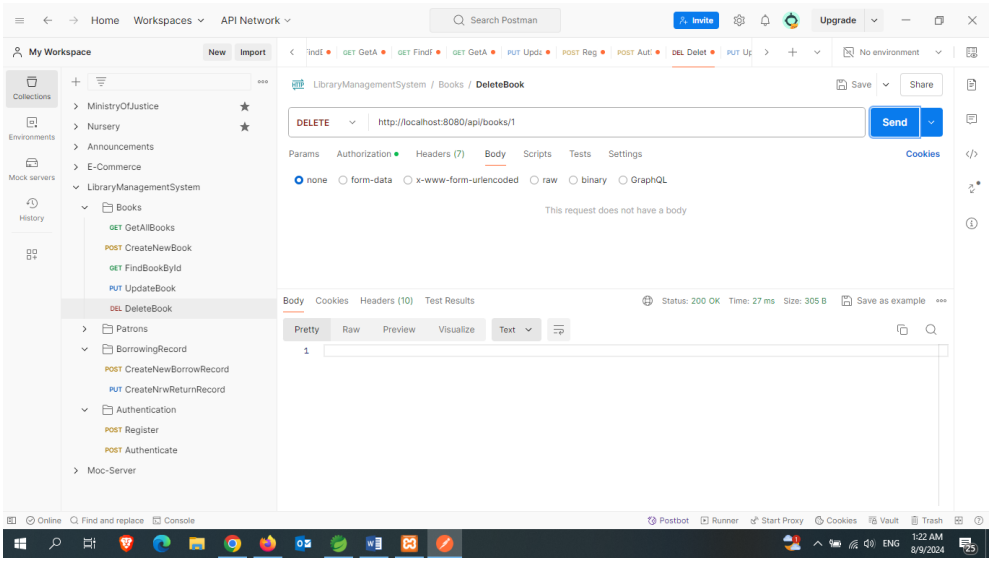
5. Get Book By Id



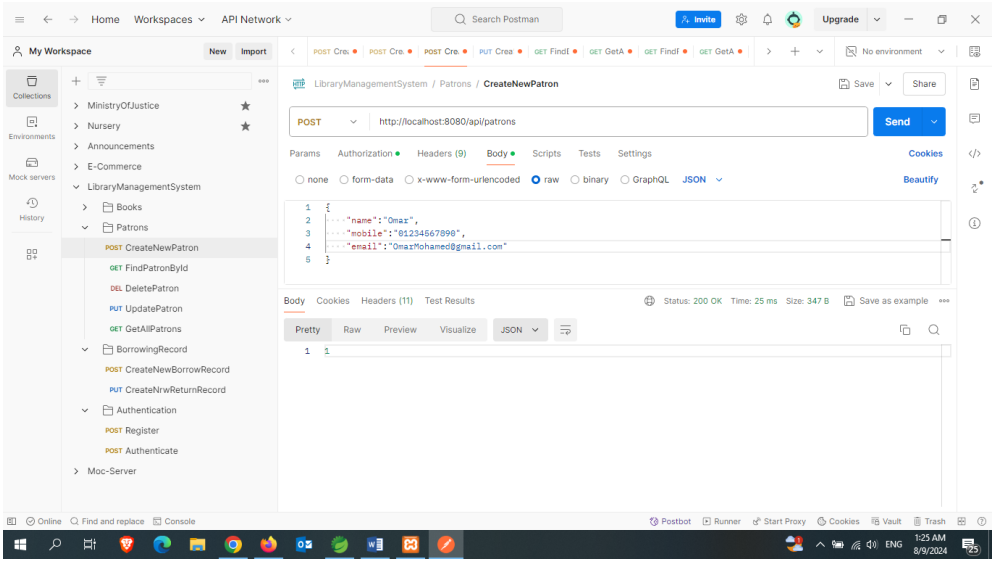
6. Update Book



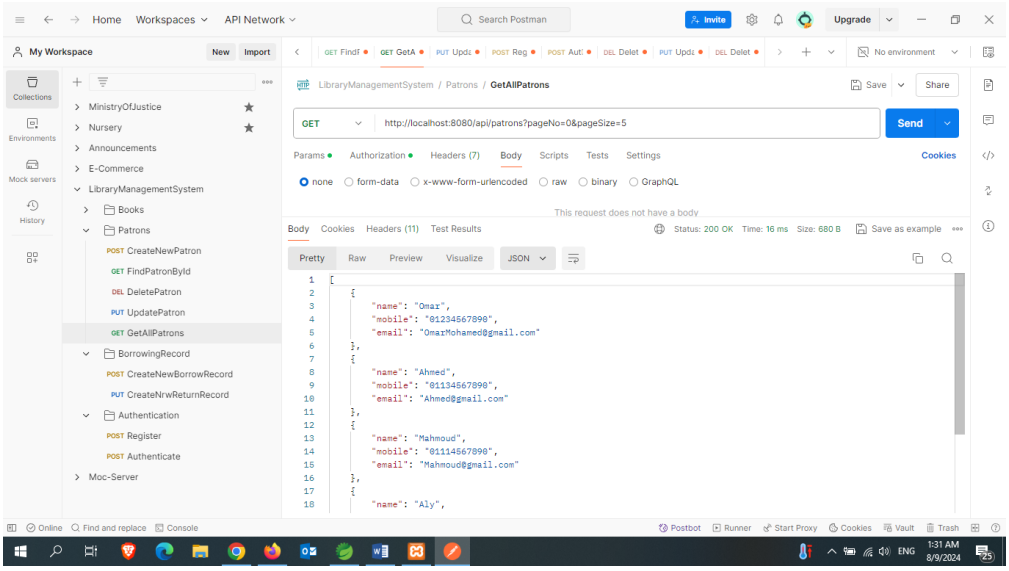
7. Delete Book



8. Add Patron

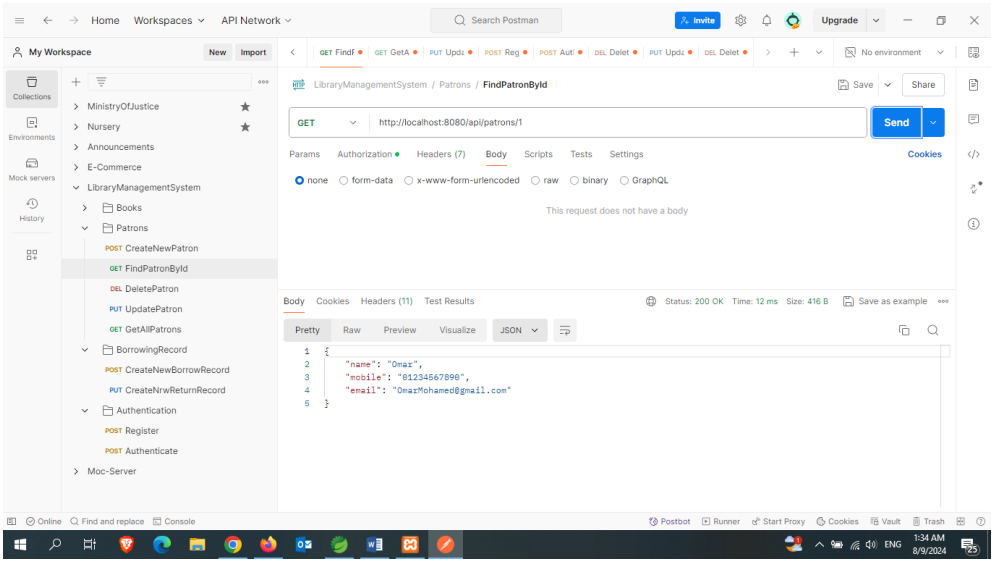


9. Get All Patrons

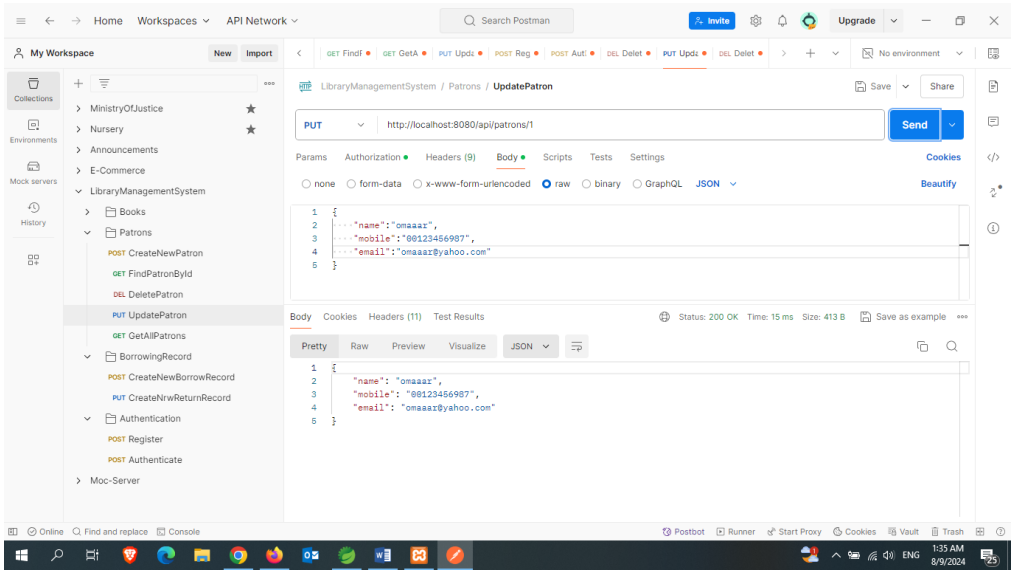


-Using of pagination and page size to control number of data

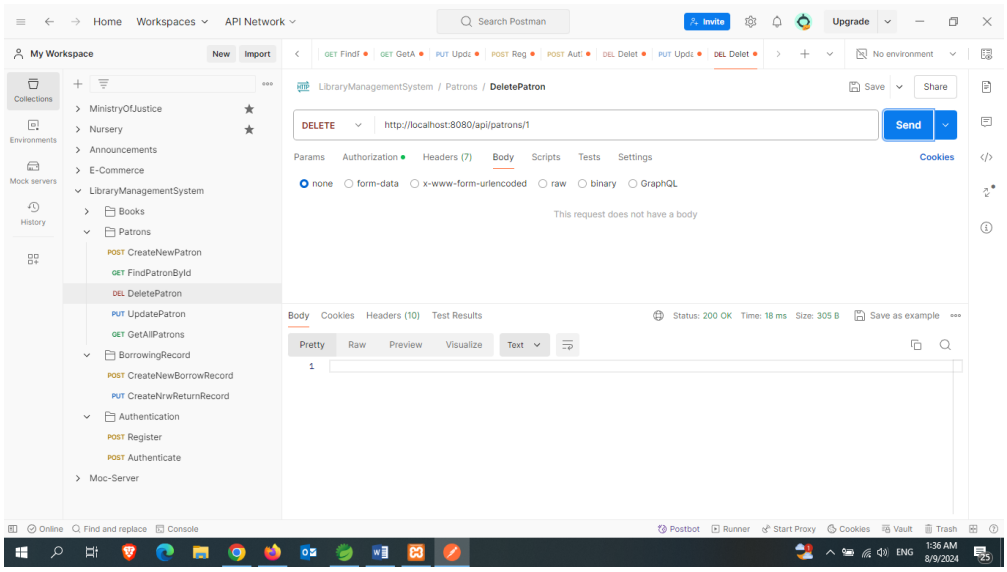
10. Get Patron By Id



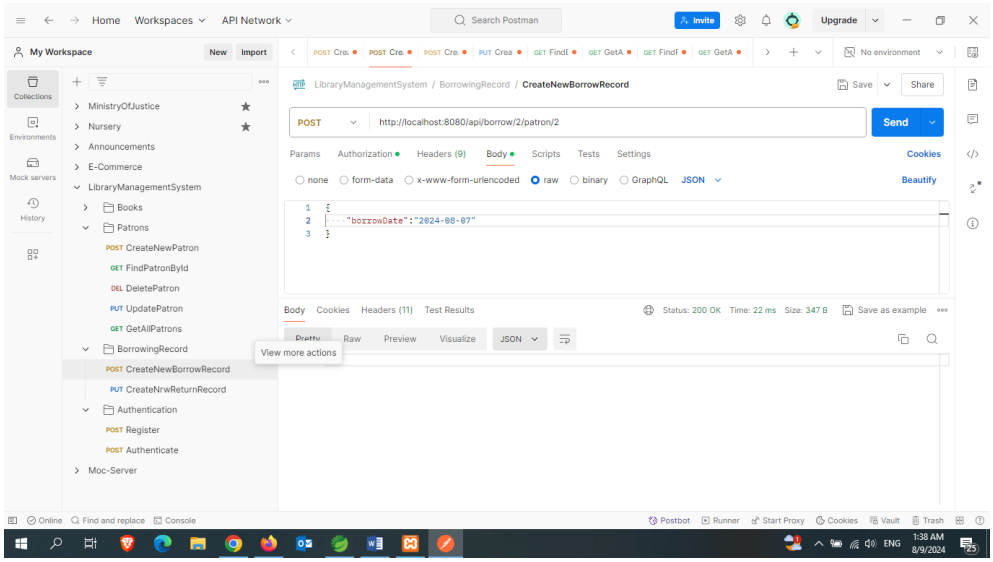
11.Update Patron



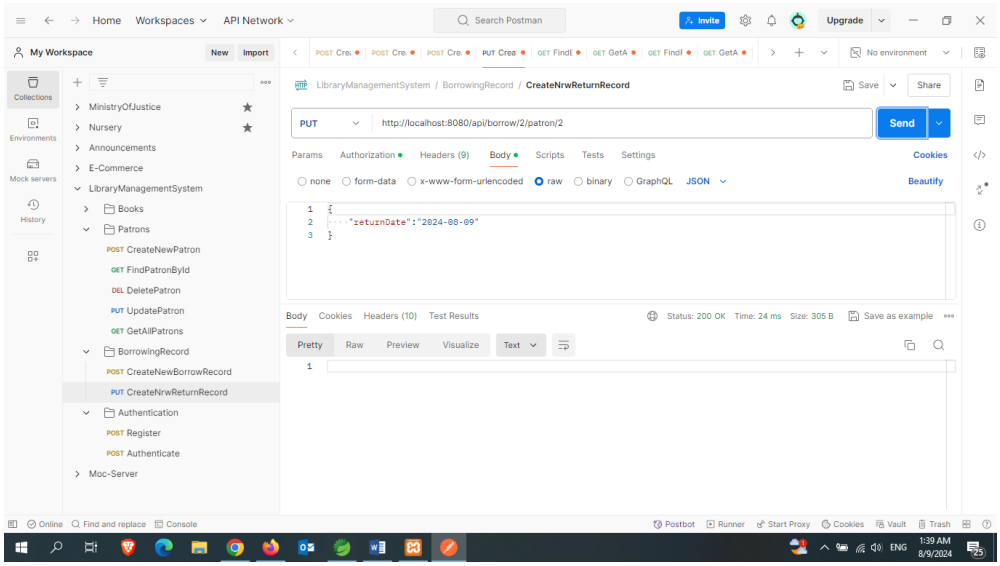
12.Delete Patron



13.Create New Borrow Record



14.Create New Return Record



- **Aspect Oriented Programming:**
  -Creating a class acts as an aspect using @aspect annotation to log the method data As its name , service and execution time at join point "**serviceImple**" package.

  -using dependency "**spring-boot-starter-aop**" to implement this method

- **Unit testing:**
  -Creating unit testing for many test cases for each end point using "**Springboot test"** and "**Mockito**" with dependency "**spring-boot-starter-test**"

- **Transactional:**
  - Using "**@Transactional**" annotation Spring manages the transaction automatically, ensuring that the method runs within a transaction context. If an exception occurs during the method execution, the transaction is rolled back; otherwise, it's committed.

- **Swagger Documentation:**
  -Using "**Swagger**" for the visual documentation for the project by fetching this url http://localhost:8080/swagger-ui/index.html

  -Using dependency "**org.springdoc**" to implement swagger documentation

- **Caching:**
  -Using springboot caching with "**@Cacheable**" annotation to improve end point performance with dependency "**spring-boot-starter-cache**"

- **Error Handling And Validations:**
  -Using springboot validations like **@NotNull** and **@NotBlank** to validate end points data and handle the project errors with obvious message for the error. this helps to now the reason of error rapidly

  -Using dependency "**spring-boot-starter-validation**" and " **javax.validation**"