**Batch: T6**

**Practical No. 8**

**Title of Assignment: Study and implementation of node.js**

**Student Name: Parshwa Herwade**

**Student PRN: 22510064**

**Problem Statement 1: Introduction to Node.js**

1. **What is Node.js, and how does it differ from traditional server-side platforms like Apache or PHP?**
   * **Answer:** Node.js is a JavaScript runtime built on Chrome's V8 engine that allows for executing JavaScript code server-side. Unlike traditional server-side platforms like Apache or PHP, which use a blocking request-response model, Node.js employs a non-blocking, event-driven architecture that can handle many connections simultaneously.
2. **What is the purpose of the V8 engine in Node.js?**
   * **Answer:** The V8 engine is an open-source JavaScript engine developed by Google that compiles JavaScript code into machine code, enabling high-performance execution. It is responsible for executing JavaScript in Node.js.
3. **Explain the single-threaded, event-driven architecture of Node.js.**
   * **Answer:** Node.js uses a single-threaded architecture to handle multiple requests without creating new threads for each request. It relies on an event loop that allows it to manage asynchronous operations, enabling the server to continue processing other tasks while waiting for I/O operations to complete.
4. **Why is Node.js considered non-blocking?**
   * **Answer:** Node.js is considered non-blocking because it processes I/O operations asynchronously. When an I/O operation is initiated, Node.js continues executing other code instead of waiting for the operation to complete, thus not blocking the event loop.
5. **What is npm, and how is it used in Node.js?**
   * **Answer:** npm (Node Package Manager) is the default package manager for Node.js. It allows developers to install, share, and manage packages and libraries for Node.js applications.
6. **What is a module in Node.js? How do you export and import modules?**
   * **Answer:** A module in Node.js is a reusable block of code that encapsulates related functionalities. You can export a module using module.exports and import it using require().

Example:

const add = (a, b) => a + b;

module.exports = { add };

const math = require('./math');

console.log(math.add(2, 3)); // Outputs: 5

7. **What is the difference between require() and import in Node.js?**

* **Answer:** require() is part of the CommonJS module system, allowing for synchronous module loading, while import is part of the ES6 module system, enabling asynchronous module loading and providing a more flexible syntax.

8. **How can you create a custom module in Node.js?**

* **Answer:** You can create a custom module by defining functions or objects in a JavaScript file and exporting them using module.exports.

**Example:**

const myFunction = () => 'Hello from my module!';

module.exports = myFunction;

const myModule = require('./myModule');

console.log(myModule());

9. **What is the role of the package.json file in a Node.js project?**

* **Answer:** The package.json file contains metadata about the project, including its dependencies, scripts, and configurations. It is essential for managing packages and their versions in the project.

10. **How do you install a package globally and locally using npm?**

* **Answer:** To install a package locally, run npm install package-name, which installs it in the node\_modules directory. To install a package globally, use npm install -g package-name, making it available across the system.

11. **What is the difference between asynchronous and synchronous programming in Node.js?**

* **Answer:** Asynchronous programming allows tasks to run without blocking the execution of other code, whereas synchronous programming blocks the execution until a task is completed. Node.js favors asynchronous programming to handle multiple connections efficiently.

12. **How do you create an HTTP server in Node.js?**

* **Answer:** You can create an HTTP server using the built-in 'http' module.

**Example:**  
const http = require('http');

const server = http.createServer((req, res) => {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('Hello World!');

});

server.listen(3000, () => {

console.log('Server running at http://localhost:3000/');

});

13. **What is the difference between http.createServer() and using frameworks like Express.js?**

* **Answer:** http.createServer() creates a basic HTTP server, while frameworks like Express.js provide additional functionalities such as routing, middleware support, and easier handling of requests, making it simpler to develop web applications.

14. **How do you handle GET and POST requests in Node.js?**

* **Answer:** You can handle GET and POST requests using the 'http' module to check the request method.

Example:  
const http = require('http');

const server = http.createServer((req, res) => {

if (req.method === 'GET') {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('GET request received');

} else if (req.method === 'POST') {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('POST request received');

}

});

server.listen(3000);

**Technologies Used**

1. Node.js
2. V8 Engine
3. npm
4. Express.js

**Problem Statement 2: Middleware (Express.js)**

1. **What is middleware in Node.js, particularly in the context of Express.js?**
   * **Answer:** Middleware in Express.js is a function that has access to the request and response objects and the next middleware function in the application's request-response cycle. Middleware can modify the request or response, terminate the request-response cycle, and call the next middleware function.
2. **How do you create custom middleware in Express.js?**
   * **Answer:** Custom middleware can be created by defining a function that takes req, res, and next parameters.

const express = require('express');

const app = express();

const myMiddleware = (req, res, next) => {

console.log('Middleware executed');

next();

};

app.use(myMiddleware);

app.get('/', (req, res) => {

res.send('Hello World!');

});

app.listen(3000);

1. **Explain how middleware is executed in order in an Express.js application.**

* **Answer:** Middleware functions are executed in the order they are defined in the application. When a request is received, Express processes each middleware function sequentially until it reaches the end or a function ends the response.

**Technologies Used**

1. Node.js
2. Express.js

**Problem Statement 3: File System (fs) Module**

1. **How do you read and write files using the fs module in Node.js?**
   * **Answer:** You can use fs.readFile() to read files and fs.writeFile() to write files.

Example:  
const fs = require('fs');

fs.readFile('example.txt', 'utf8', (err, data) => {

if (err) throw err;

console.log(data);

});

fs.writeFile('example.txt', 'Hello World!', (err) => {

if (err) throw err;

console.log('File written!');

});

2. **What is the difference between fs.readFile() and fs.readFileSync()?**

* **Answer:** fs.readFile() is asynchronous and does not block the execution of the program, while fs.readFileSync() is synchronous and blocks the execution until the file is read.

3. **How can you check if a file or directory exists in Node.js?**

* **Answer:** You can check if a file or directory exists using fs.existsSync(path).

const fs = require('fs');

if (fs.existsSync('example.txt')) {

console.log('File exists!');

} else {

console.log('File does not exist!');

}

1. **How do you handle file operations in an asynchronous manner?**

* **Answer:** File operations can be handled asynchronously using callback functions or Promises.
* **Example:**  
  const fs = require('fs').promises;

fs.readFile('example.txt', 'utf8')

.then(data => console.log(data))

.catch(err => console.error(err));

**Technologies Used**

1. Node.js
2. fs Module

**Problem Statement 4: Database Connectivity**

1. **How do you connect to a MongoDB database from a Node.js application?**
   * **Answer:** To connect to a SQL or Oracle database from a Node.js application, you can use specific libraries like mysql2 for SQL databases or oracledb for Oracle databases.

**🡪npm install mysql2**

Create connection:  
const mysql = require('mysql2');

const connection = mysql.createConnection({

host: 'localhost',

user: 'yourUsername',

password: 'yourPassword',

database: 'yourDatabaseName'

});

connection.connect((err) => {

if (err) {

console.error('Error connecting to the database:', err);

return;

}

console.log('Connected to the MySQL database');

});

execute some quereies and end the connection when done.

2. **What is the purpose of the mongoose library in Node.js?**

* **Answer:** Mongoose is an ODM (Object Document Mapper) library that provides a schema-based solution for modeling application data with MongoDB, simplifying CRUD operations and schema validation.

3. **Explain how you would perform basic CRUD operations (Create, Read, Update, Delete) using MySQL and Node.js.**

**Example:  
Create a connection as above:**

**Insert data:  
const insertQuery = `INSERT INTO users (name, age, email) VALUES ('John Doe', 30, 'john@example.com')`;**

**connection.query(insertQuery, (err, result) => {**

**if (err) throw err;**

**console.log('Data inserted successfully:', result.insertId);**

**});**

**Read:  
const selectQuery = `SELECT \* FROM users`;**

**connection.query(selectQuery, (err, results) => {**

**if (err) throw err;**

**console.log('Data retrieved:', results);**

**});  
  
Update:  
const updateQuery = `UPDATE users SET age = 31 WHERE name = 'John Doe'`;**

**connection.query(updateQuery, (err, result) => {**

**if (err) throw err;**

**console.log('Data updated successfully:', result.affectedRows);**

**});**

**Delete:**

**const deleteQuery = `DELETE FROM users WHERE name = 'John Doe'`;**

**connection.query(deleteQuery, (err, result) => {**

**if (err) throw err;**

**console.log('Data deleted successfully:', result.affectedRows);**

**});**

**End connection:  
connection.end();**

**Technologies Used**

1. Node.js
2. MySQL

Problem Statement 5: Building a RESTful API:

**Technologies Used**

1. **Node.js**
   * **Description:** A JavaScript runtime built on Chrome's V8 JavaScript engine, allowing developers to build scalable and fast network applications.
   * **Theoretical Concepts:** Event-driven architecture, non-blocking I/O operations, single-threaded event loop.
2. **Express.js**
   * **Description:** A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
   * **Theoretical Concepts:** Middleware functions, routing, RESTful API design principles, request-response cycle.
3. **MySQL2: Driver for connecting Node.js to MySQL**
4. **Multer**
   * **Description:** A middleware for handling multipart/form-data, primarily used for uploading files.
   * **Theoretical Concepts:** Handling file uploads, managing storage options (disk vs. memory), file validation and sanitization.
5. **JWT (JSON Web Tokens)**
   * **Description:** A compact, URL-safe means of representing claims to be transferred between two parties, commonly used for authentication and information exchange.
   * **Theoretical Concepts:** Stateless authentication, token-based access control, encoding and decoding JWT, expiration and refresh tokens.
6. **Postman or Curl**
   * **Description:** Tools used for testing APIs by sending HTTP requests to your server and viewing responses.
   * **Theoretical Concepts:** API testing, understanding HTTP methods (GET, POST, PUT, DELETE), status codes.

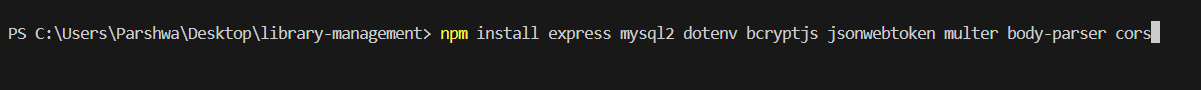
**Building a RESTful API**

1. **REST Principles:**
   * **Statelessness:** Each request from client to server must contain all the information the server needs to fulfill that request. The server does not store client context.
   * **Resource-Based:** REST APIs should be structured around resources (e.g., books) and use standard HTTP methods to perform actions on these resources.
   * **Representations:** Resources can be represented in multiple formats, typically JSON or XML. JSON is commonly used for its ease of use with JavaScript.
   * **Use of Standard HTTP Methods:**
     + **GET:** Retrieve data from the server.
     + **POST:** Send data to the server to create a new resource.
     + **PUT:** Update an existing resource.
     + **DELETE:** Remove a resource from the server.
   * **Status Codes:** HTTP status codes provide feedback on the outcome of the API request (e.g., 200 OK, 201 Created, 404 Not Found, 500 Internal Server Error).
2. **Authentication and Authorization:**
   * **User Authentication:** Verify a user's identity (e.g., using JWT) before allowing access to certain routes.
   * **Authorization:** Determine what authenticated users are allowed to do (e.g., only allowing admin users to delete books).

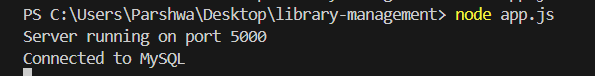
**File Upload and Management System**

1. **File Handling:**
   * **File Uploads:** Accepting files from users and storing them in a server directory or cloud storage.
   * **File Types:** Supporting various file formats (images, PDFs, etc.) and validating the file type to prevent malicious uploads.
2. **CRUD Operations for Files:**
   * **Create:** Uploading new files to the server.
   * **Read:** Fetching the list of uploaded files and displaying them to the user.
   * **Update:** (Optional) Modifying existing files if needed (e.g., replacing an uploaded file).
   * **Delete:** Allowing users to remove files they uploaded.
3. **Security Considerations:**
   * **File Validation:** Ensuring that uploaded files are safe and of the expected type/size.
   * **Access Control:** Restricting upload and management capabilities to authenticated users to prevent unauthorized access.
4. **Storage Management:**
   * **Local Storage vs. Cloud Storage:** Deciding whether to store files on the server’s local disk or in cloud storage solutions like AWS S3, Google Cloud Storage, etc.

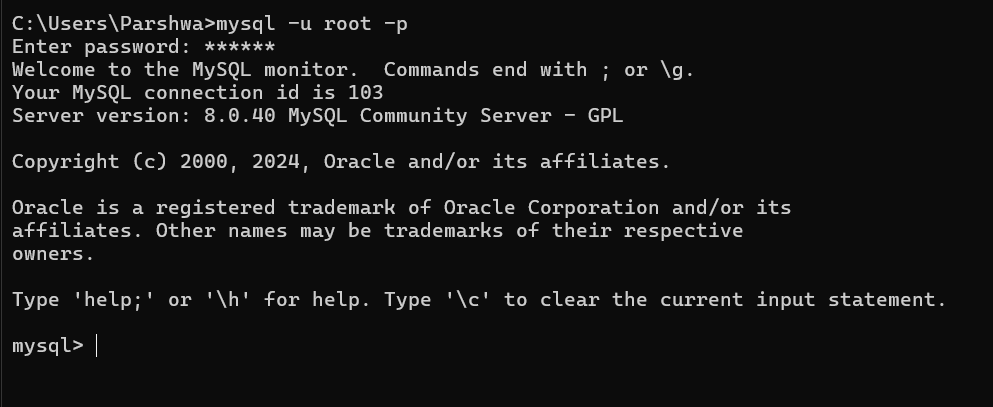
**Steps:**  





DATABASE:MYSQL2

* DOWNLOAD MYSQL WORKBENCH:
* LOGIN: 

CREATING TABLE AND STRUCTURES:

USER DATABSES:  
CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);  
  
Book Schema:  
CREATE TABLE books (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

author VARCHAR(255) NOT NULL,

genre VARCHAR(255),

year\_of\_publication INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

File Schema:  
CREATE TABLE files (

id INT AUTO\_INCREMENT PRIMARY KEY,

filename VARCHAR(255) NOT NULL,

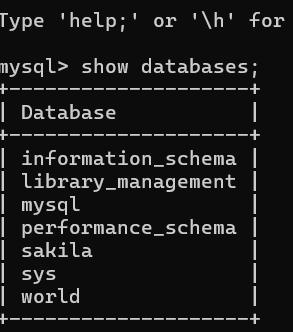
path VARCHAR(255) NOT NULL,

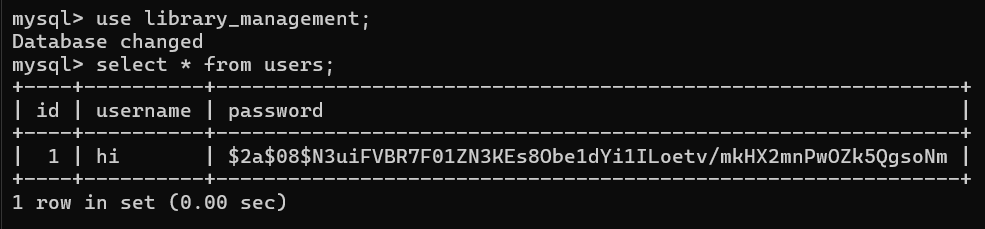
uploaded\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

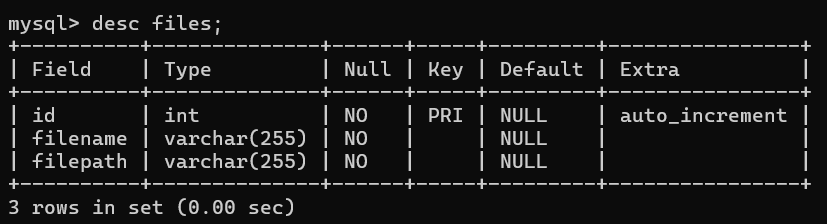
user\_id INT,

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

);

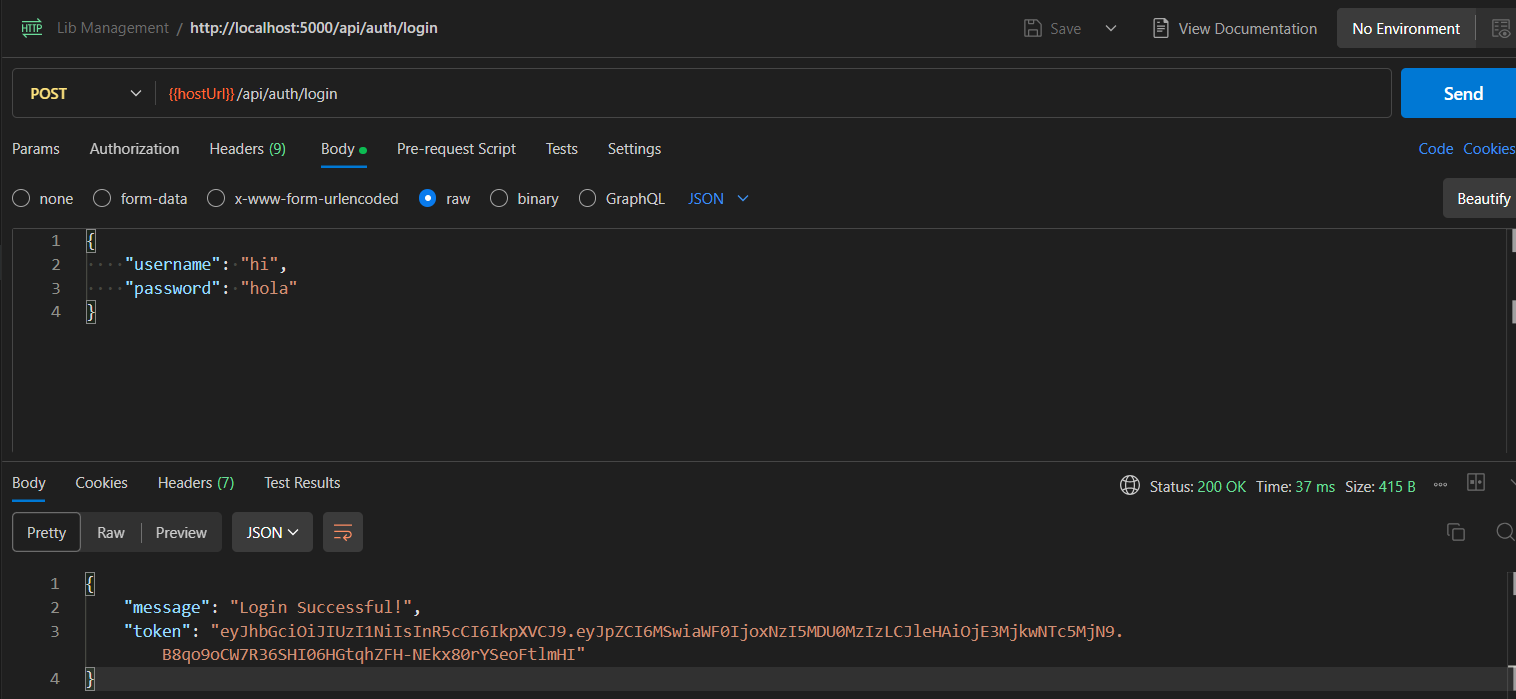




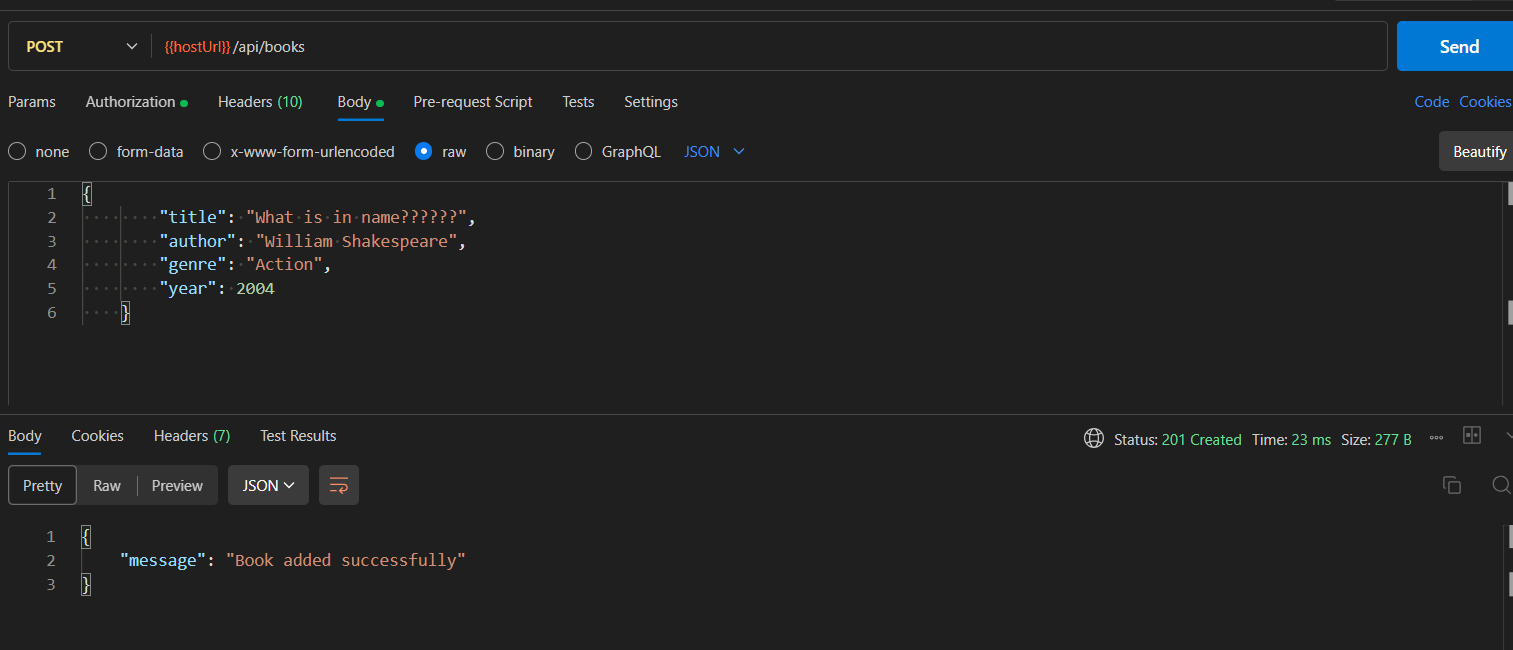


Postman for requests:

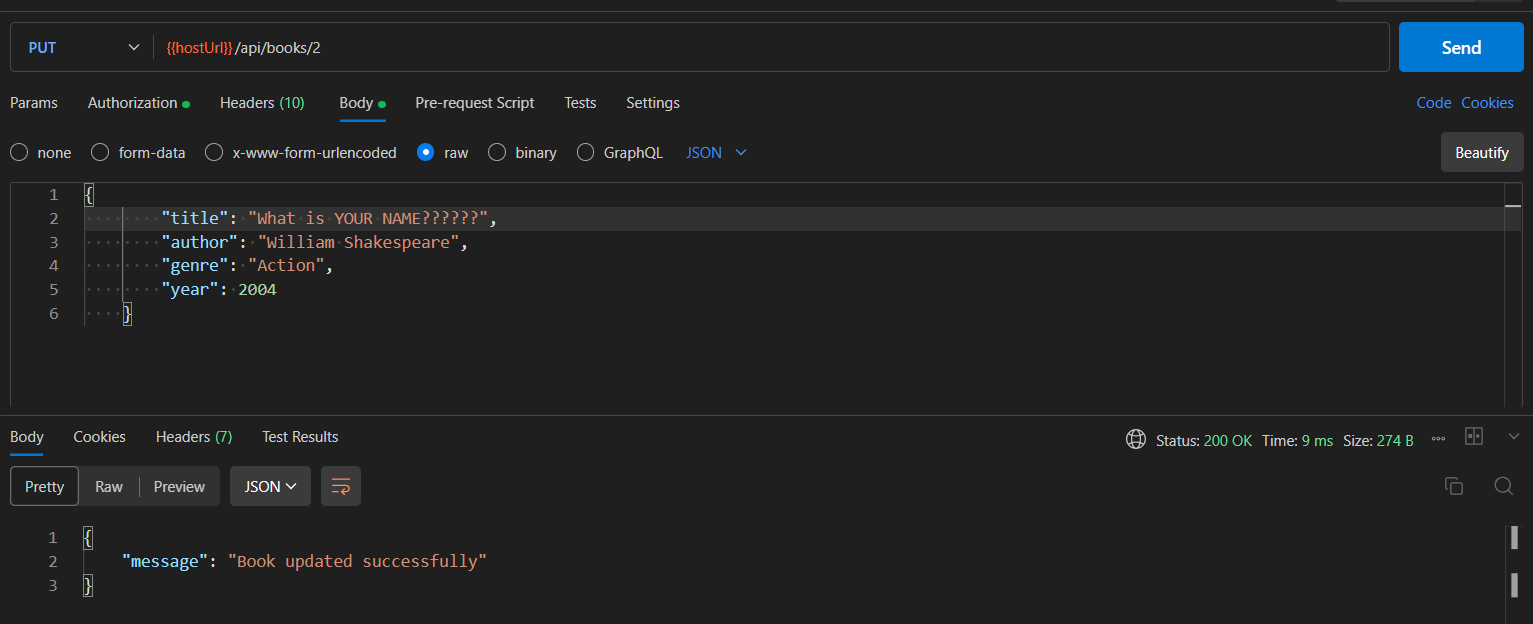
LOGIN:



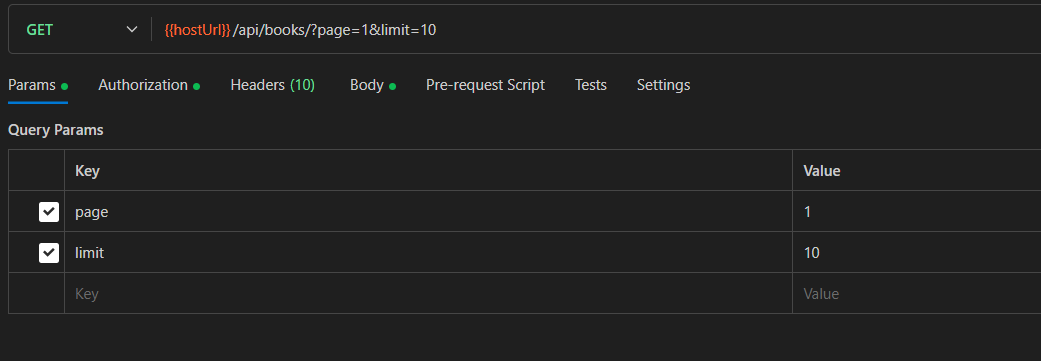
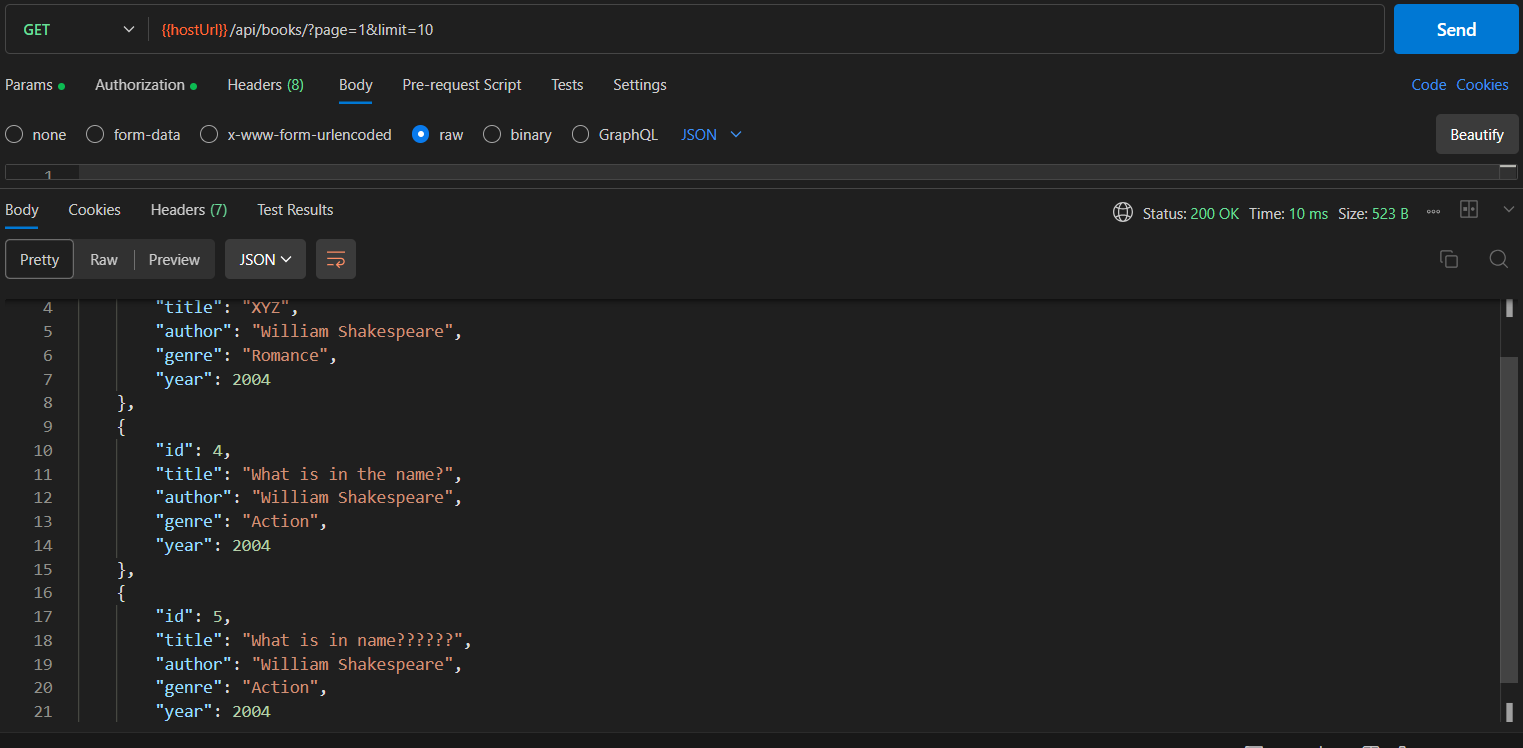
ADD BOOK:



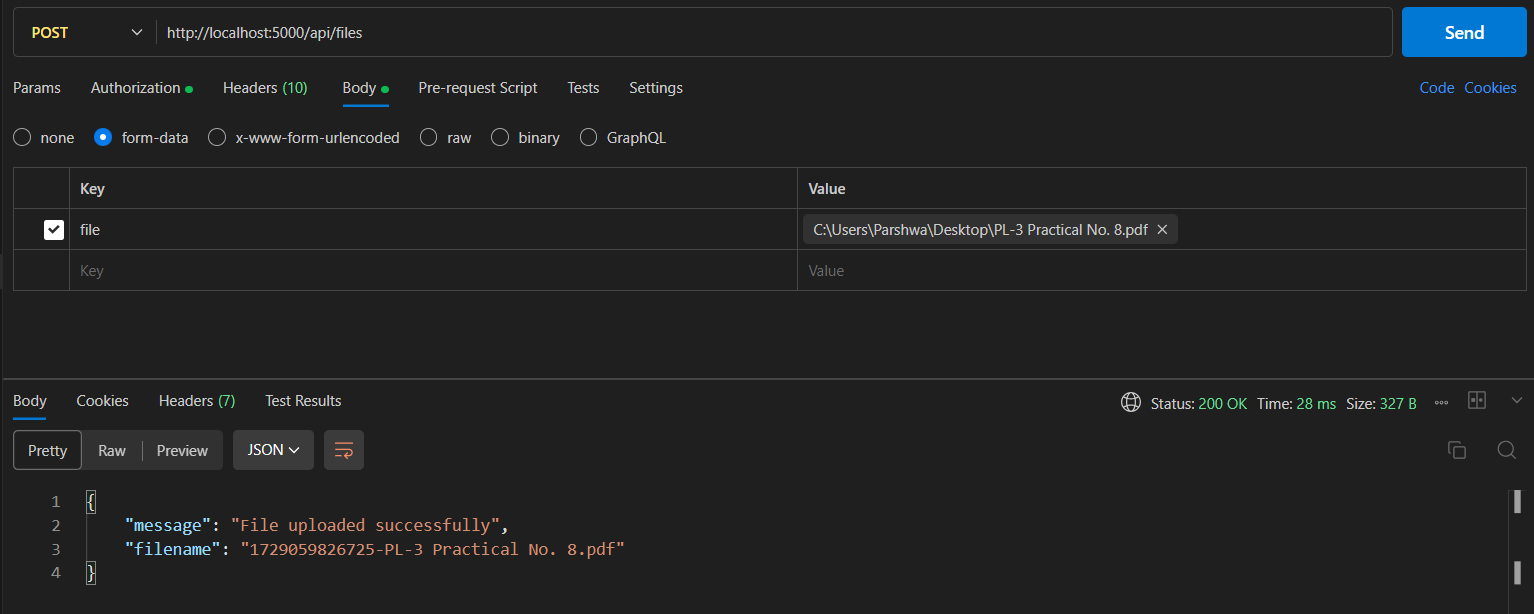
UPDATE BOOK:



GET:  
PAGINATION:



FILE UPLOAD:



GET FILE:  
