**Batch -** T5

**Practical No. -** 8

**Title –** Study and implementation of Node.js

**Student Name -** Sharaneshwar Bharat Punjal

**Student PRN -** 23520011

**Perform following problem statements using Node.Js**

**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?**

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine, allowing JavaScript to be run on the server. Unlike traditional platforms like Apache or PHP, Node.js is event-driven and non-blocking, making it ideal for handling multiple concurrent requests. It uses a single-threaded architecture, while Apache and PHP typically follow a multi-threaded model.

1. **What is the purpose of the V8 engine in Node.js?**

The V8 engine is responsible for executing JavaScript code in Node.js. It compiles JavaScript to machine code for faster execution, which makes Node.js highly performant.

1. **Explain the single-threaded, event-driven architecture of Node.js.**

Node.js operates on a single thread, but it's event-driven, meaning it uses an event loop to manage asynchronous operations. When a task is completed, it triggers a callback, avoiding blocking the main thread and efficiently managing many concurrent operations.

1. **Why is Node.js considered non-blocking?**

Node.js is considered non-blocking because it handles I/O operations asynchronously. Instead of waiting for a task to finish, Node.js moves to the next task, allowing the system to handle multiple operations simultaneously without being blocked by slow tasks.

1. **What is npm, and how is it used in Node.js?**

npm (Node Package Manager) is the default package manager for Node.js. It is used to install, manage, and share packages (libraries) that developers can use to add functionality to their Node.js applications.

1. **What is a module in Node.js? How do you export and import modules?**

A module in Node.js is a reusable block of code that can be exported from one file and imported into another. You can export a module using module.exports = <value> and import it using const moduleName = require('./module').

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

require() is the CommonJS way of importing modules in Node.js, which is synchronous. import is the ES6 module syntax and can be used asynchronously. require() is traditionally used in Node.js, while import is more common in modern JavaScript and requires a specific configuration in Node.js.

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

You can create a custom module by writing JavaScript code in a file and exporting it using module.exports. For example:

// myModule.js

module.exports = function() {

console.log('Hello from myModule!');

};

// Import in another file

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

myModule();

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

package.json is a configuration file in Node.js that contains metadata about the project, such as the name, version, and dependencies. It is essential for managing dependencies and scripts within the project.

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

To install a package globally: npm install -g <package-name>. To install a package locally (within the project): npm install <package-name>.

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

In synchronous programming, code is executed line-by-line, and each operation waits for the previous one to complete. In asynchronous programming, operations are executed in the background, allowing the program to continue without waiting for the current task to finish.

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

You can create an HTTP server using the http module:

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 on port 3000');

});

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

http.createServer() is a low-level way to create an HTTP server in Node.js, handling basic routing and responses manually. Express.js is a framework built on top of http that simplifies server creation by providing more robust routing, middleware support, and convenience methods.

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

In native Node.js:

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') {

let body = '';

req.on('data', chunk => { body += chunk.toString(); });

req.on('end', () => { res.end('POST data received: ' + body); });

}

});

server.listen(3000);

In Express.js, you can use app.get() and app.post() for GET and POST requests respectively.

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

1. **What is middleware in Node.js, particularly in the context of Express.js?**

Middleware in Express.js is a function that processes the request object (req), response object (res), and a next() function. Middleware functions can modify the request, response, or terminate the request-response cycle.

1. **How do you create custom middleware in Express.js?**

You can create custom middleware by defining a function that accepts req, res, and next:

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

console.log('Custom middleware called');

next(); // Pass control to the next middleware

};

app.use(customMiddleware);

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

Middleware in Express.js is executed in the order it is defined. When a request is made, Express processes the middleware functions in the order they are registered using app.use() or specific route handlers.

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

1. **How do you read and write files using the fs module in Node.js?**

You can use fs.readFile() to read a file and fs.writeFile() to write to a file:

const fs = require('fs');

// Reading a file

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

if (err) throw err;

console.log(data);

});

// Writing to a file

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

if (err) throw err;

console.log('File written');

});

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

fs.readFile() is asynchronous and non-blocking, while fs.readFileSync() is synchronous and blocks the execution of further code until the operation is complete.

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

You can use fs.existsSync() to check if a file or directory exists:

const fs = require('fs');

if (fs.existsSync('path/to/file')) {

console.log('File exists');

} else {

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

}

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

File operations can be handled asynchronously using callbacks or Promises. For example, using fs.promises:

const fs = require('fs').promises;

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

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

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

**Problem Statement 4: Database Connectivity**

**1. How do you connect to a SQL or Oracle database from a Node.js application?**

To connect to a SQL or Oracle database from a Node.js application, you need to use a library that supports database connectivity. For MySQL, a common choice is the mysql2 library, while for Oracle databases, the oracledb library is used. These libraries allow you to establish a connection by providing the database host, user credentials, and database name. Once the connection is established, queries can be executed to interact with the database.

**2. What is the purpose of the mysql2 library in Node.js?**

The mysql2 library is used in Node.js to connect to MySQL databases. It provides an API to execute SQL queries and perform database operations like reading, writing, updating, and deleting records. It also supports both callbacks and Promises for handling asynchronous operations, making it easier to work with queries in modern JavaScript.

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

* Create: Inserting new records into the database using SQL INSERT statements.
* Read: Retrieving records from the database using SQL SELECT statements.
* Update: Modifying existing records using SQL UPDATE statements.
* Delete: Removing records from the database using SQL DELETE statements.

Each operation requires executing the respective SQL query within the Node.js application using a database library like mysql2.

**Problem Statement 5: Building a RESTful API**

Develop a RESTful API using Node.js and Express.js for a library management system. The

system should allow users to:

• Add new books (title, author, genre, year of publication).

• Update book details.

• Delete books from the collection.

• Fetch a list of books with pagination and filtering by genre and author.

• Add a user authentication system to restrict access to certain API routes.

File Upload and Management System

Build a file upload and management system using Node.js and Multer (or any other file

upload middleware). The system should allow users to:

• Upload files (images, PDFs, etc.).

• View the list of uploaded files.

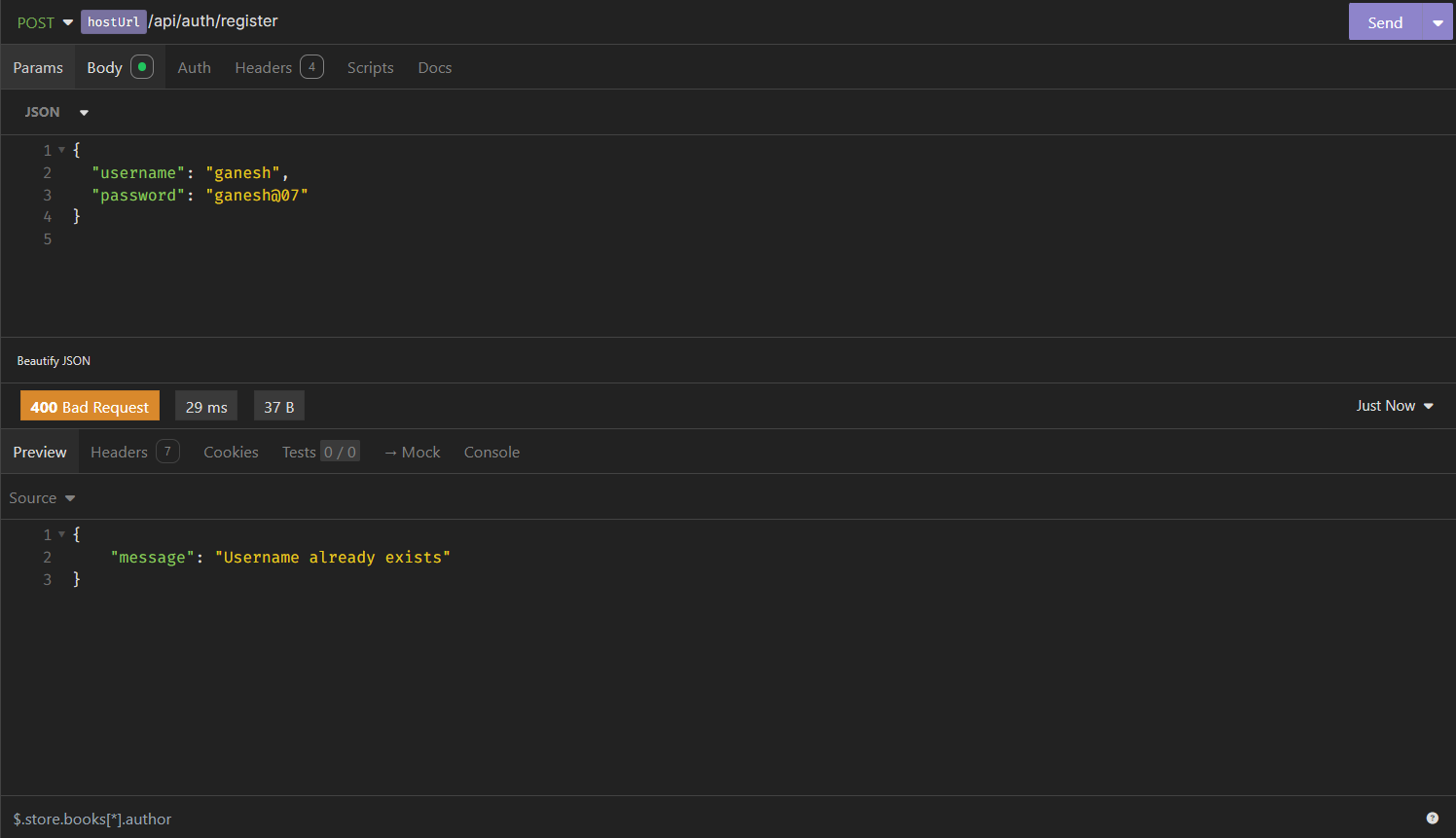
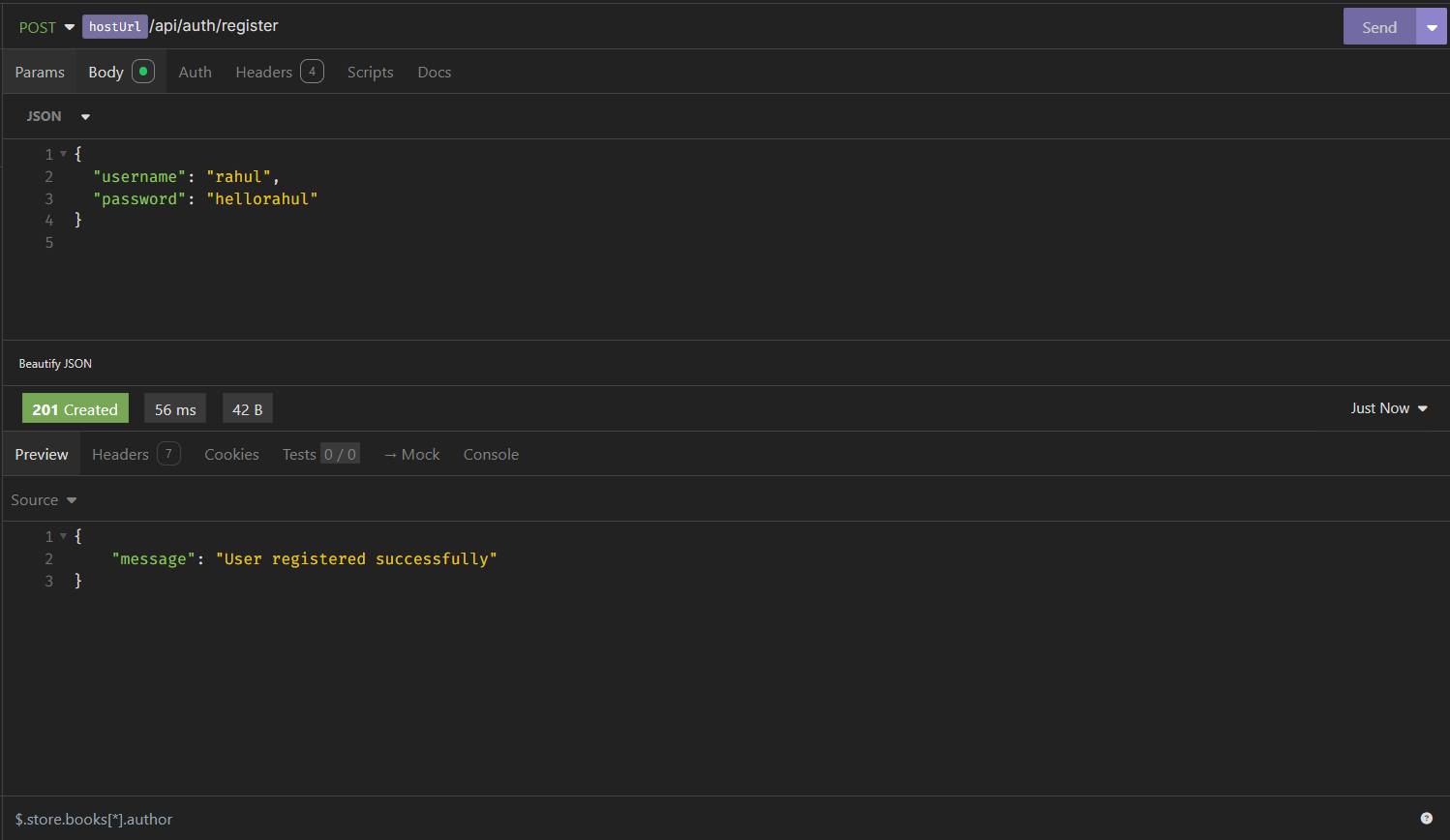
• Download or delete specific files.

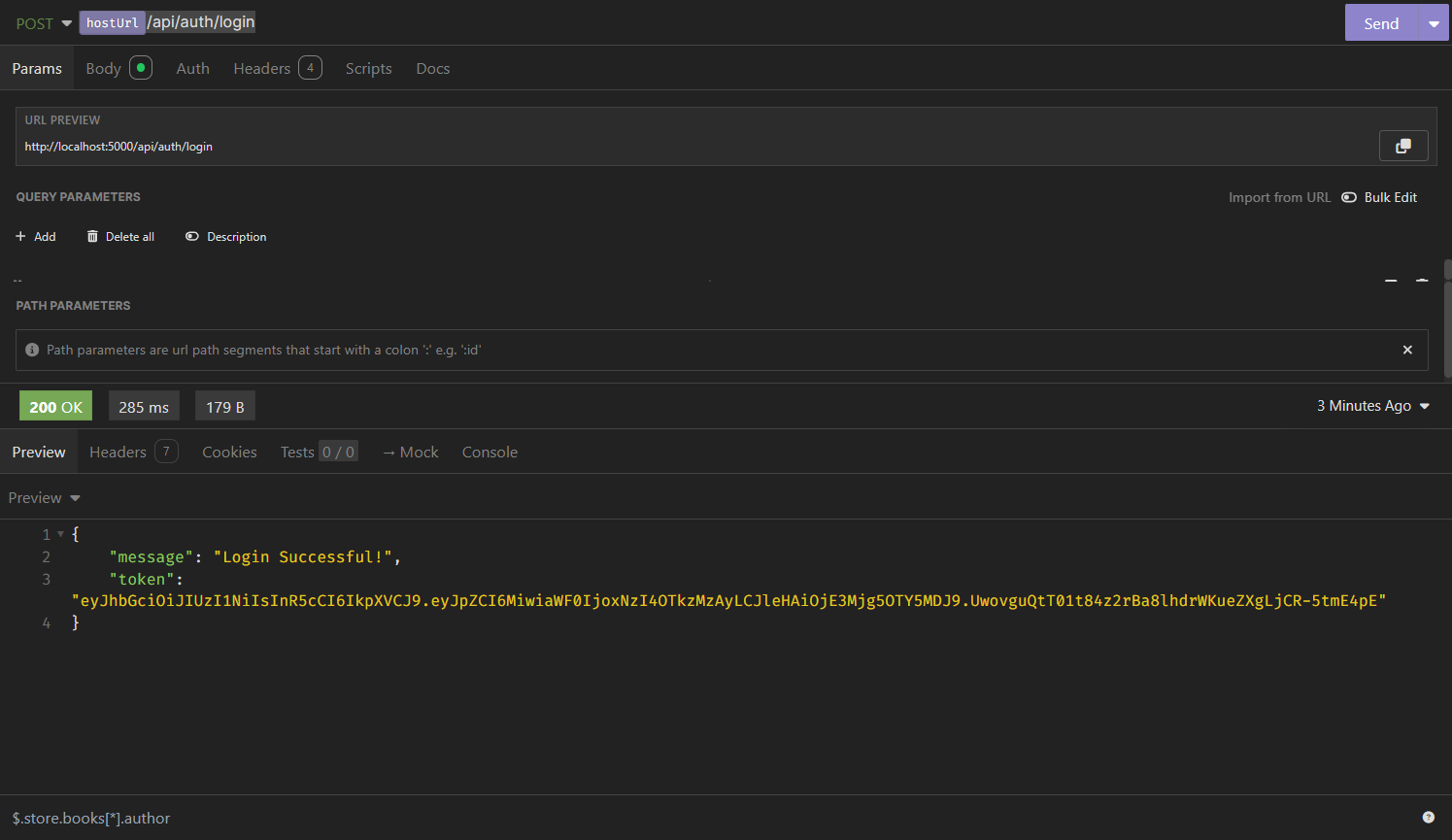
• Implement user authentication so that only authorized users can upload and

manage files.

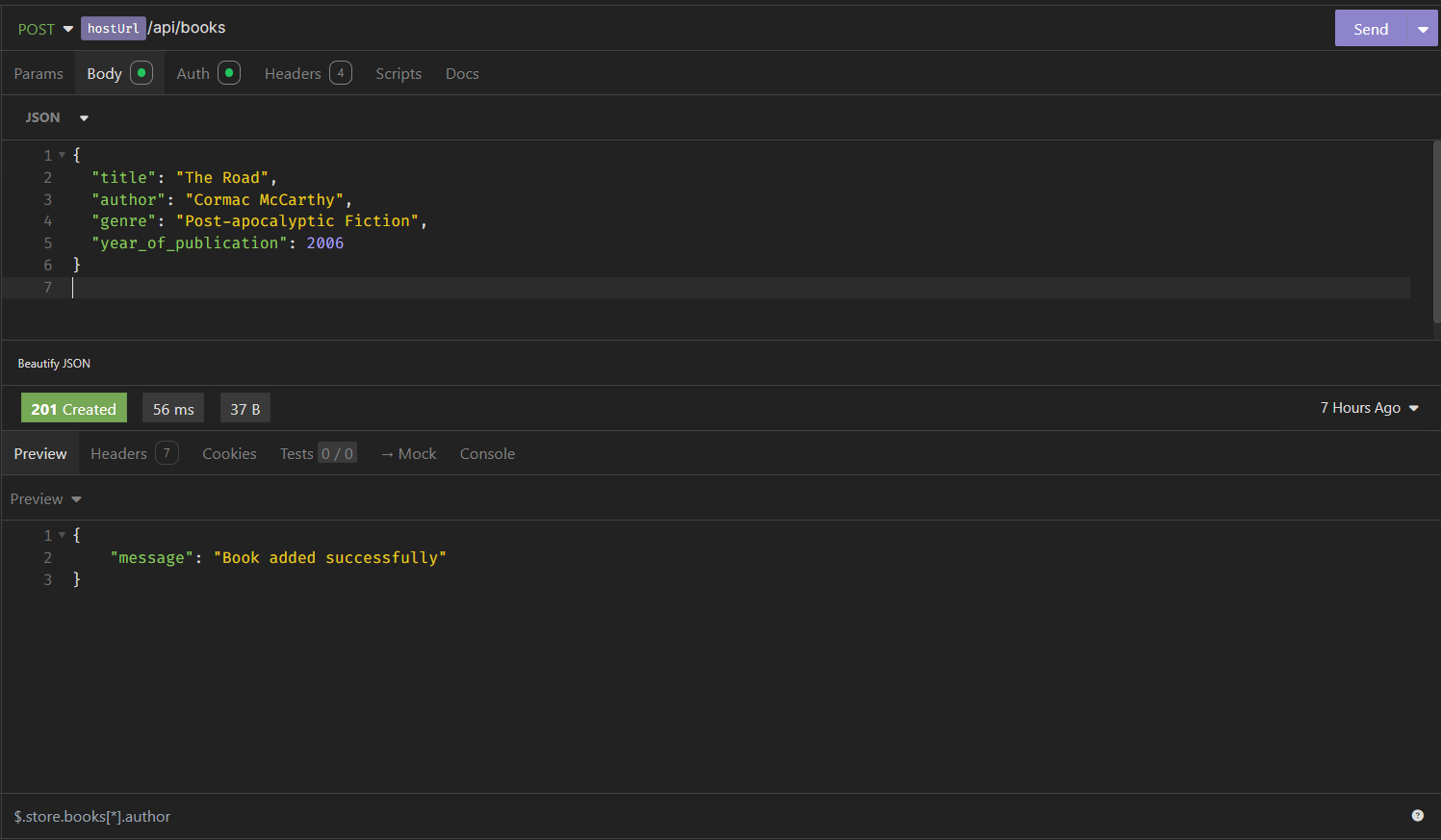
**Screenshots –**

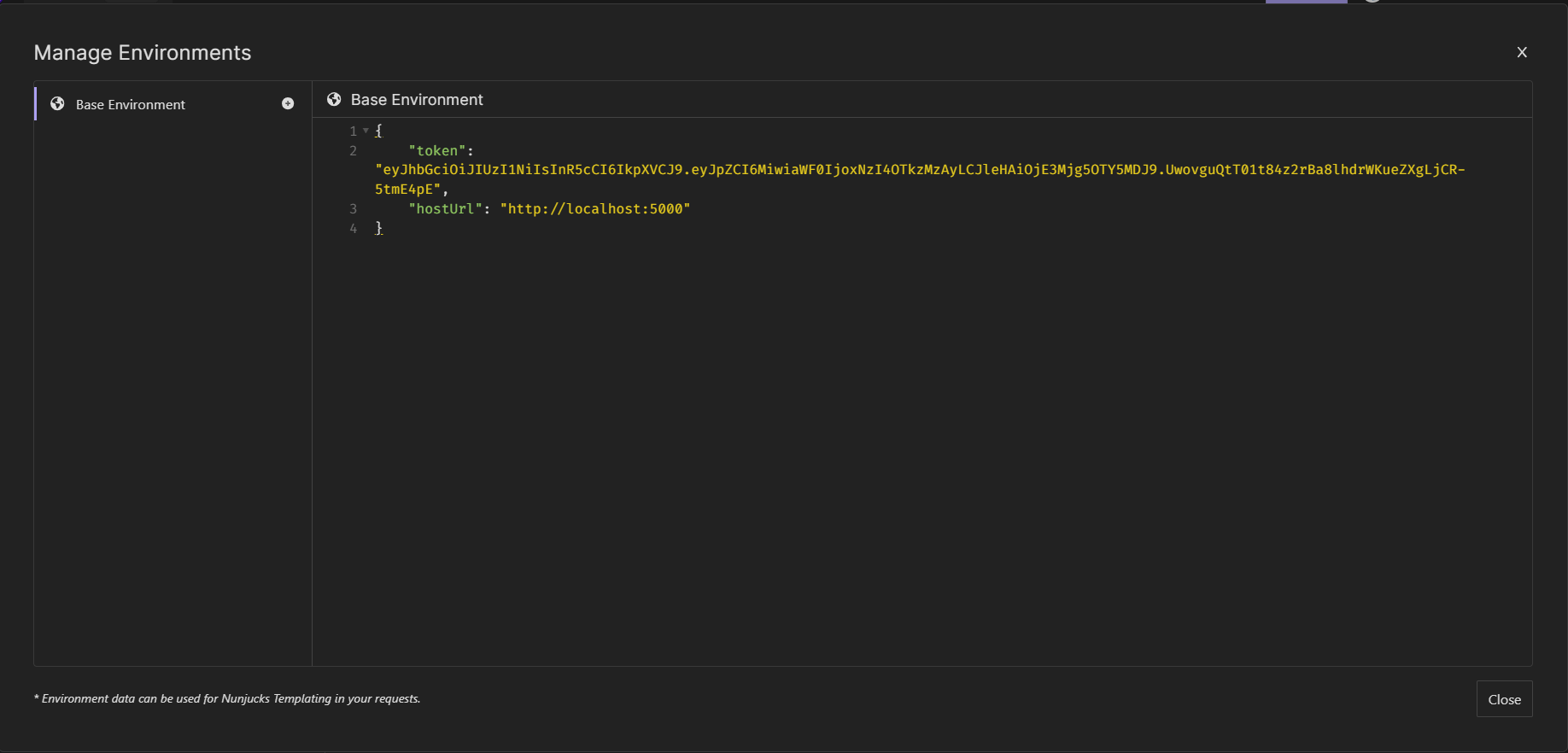
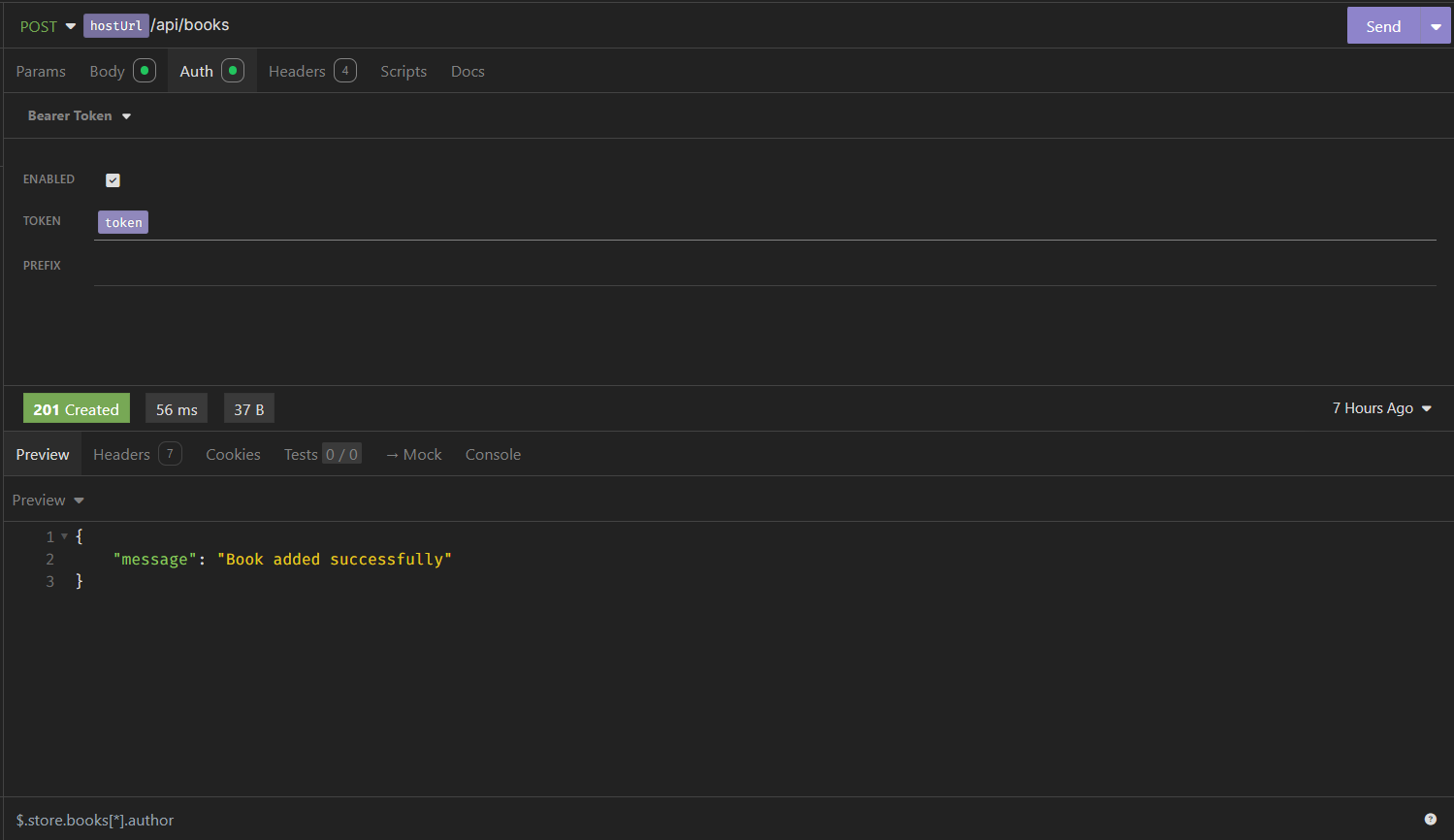
User Registration and Login:



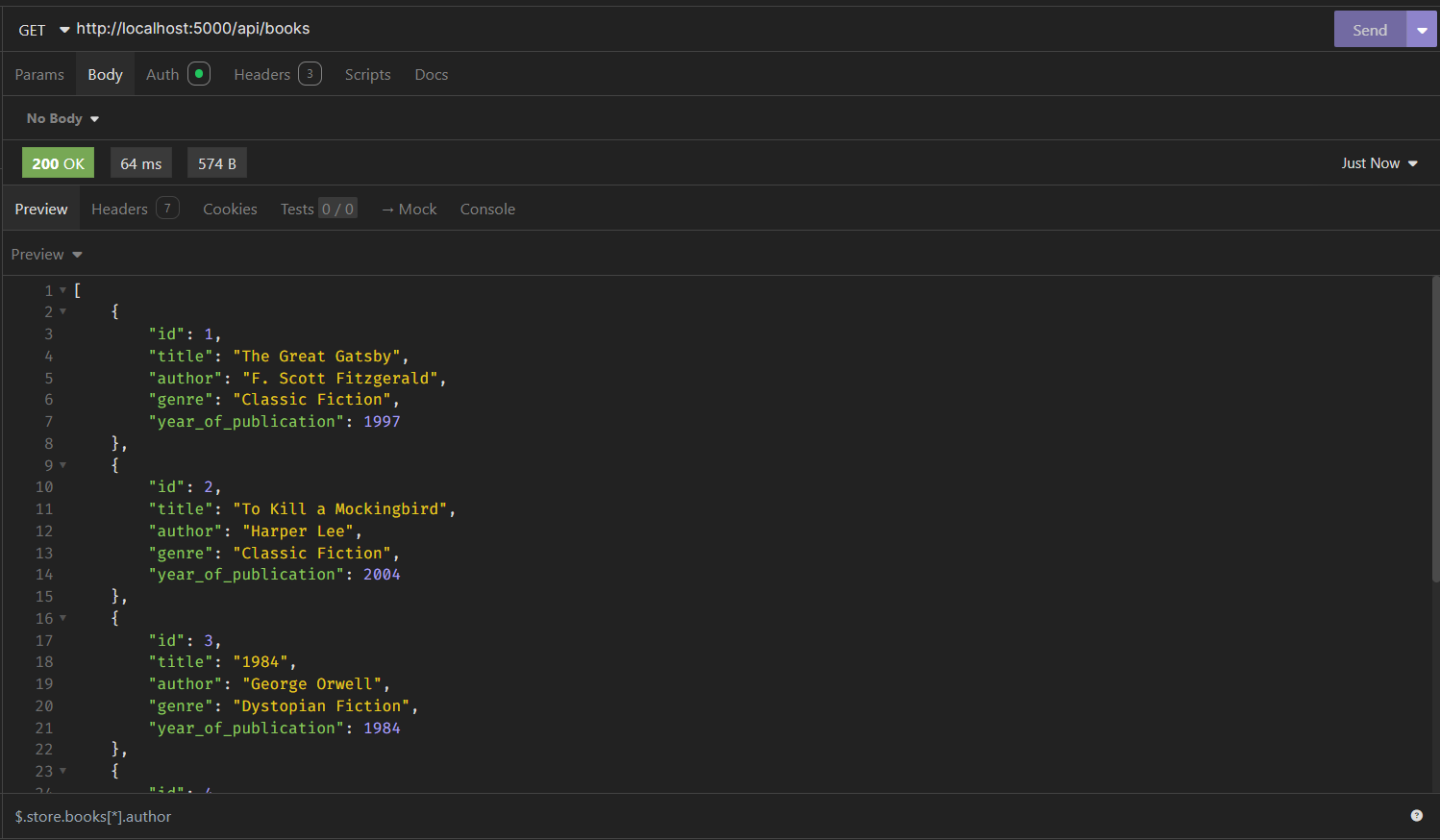


Add New Book:

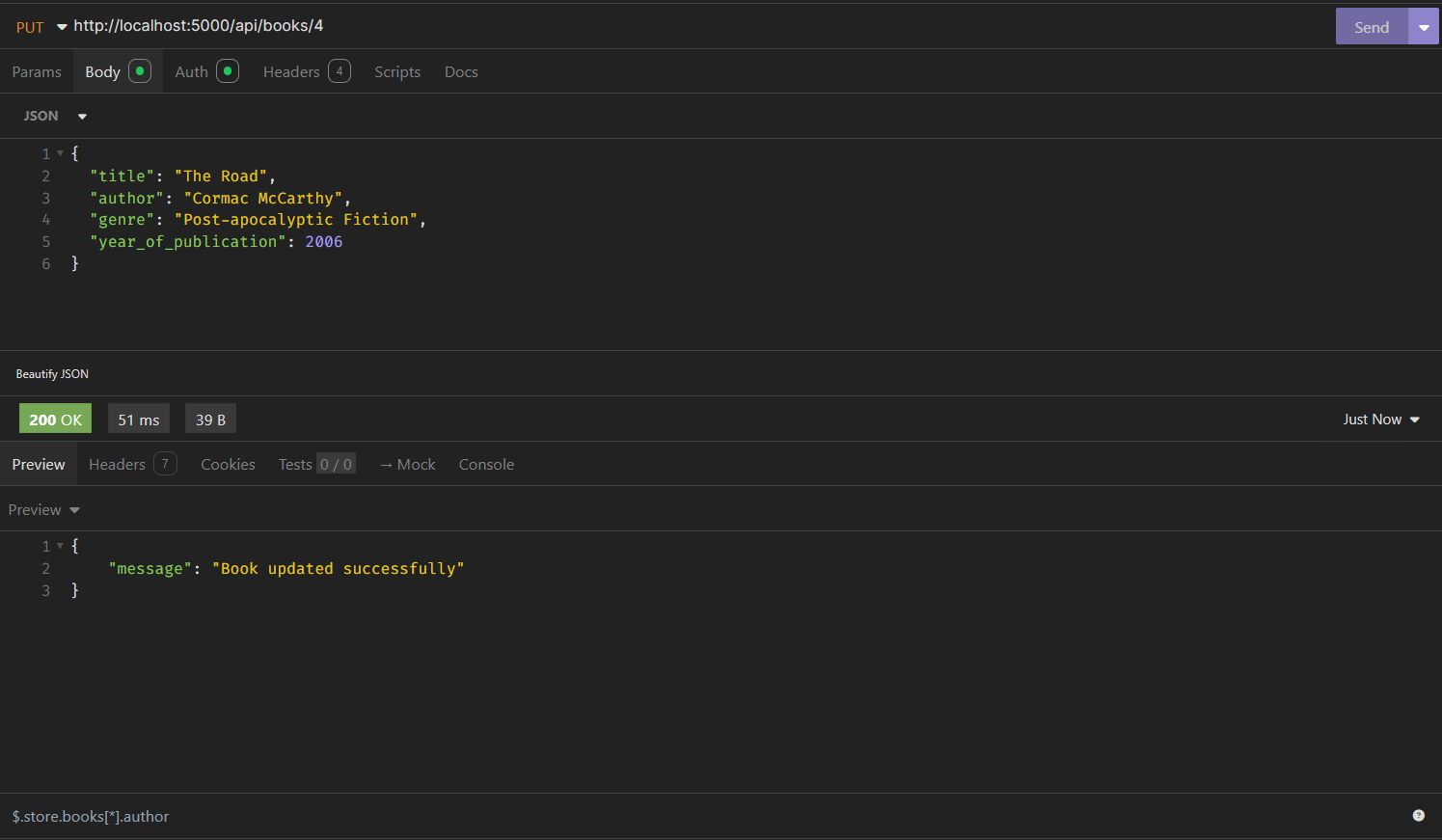




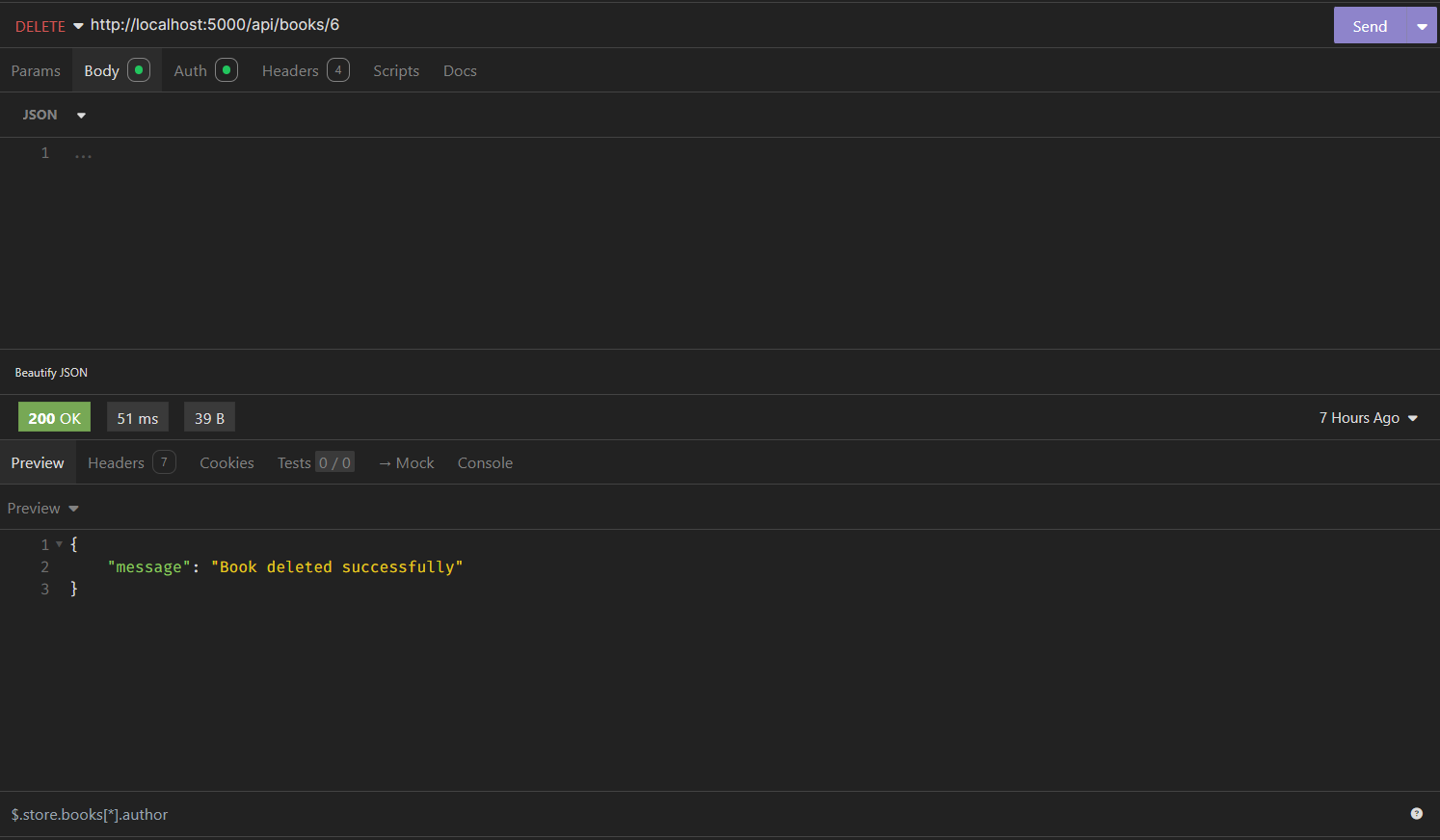
Retrieve Books -



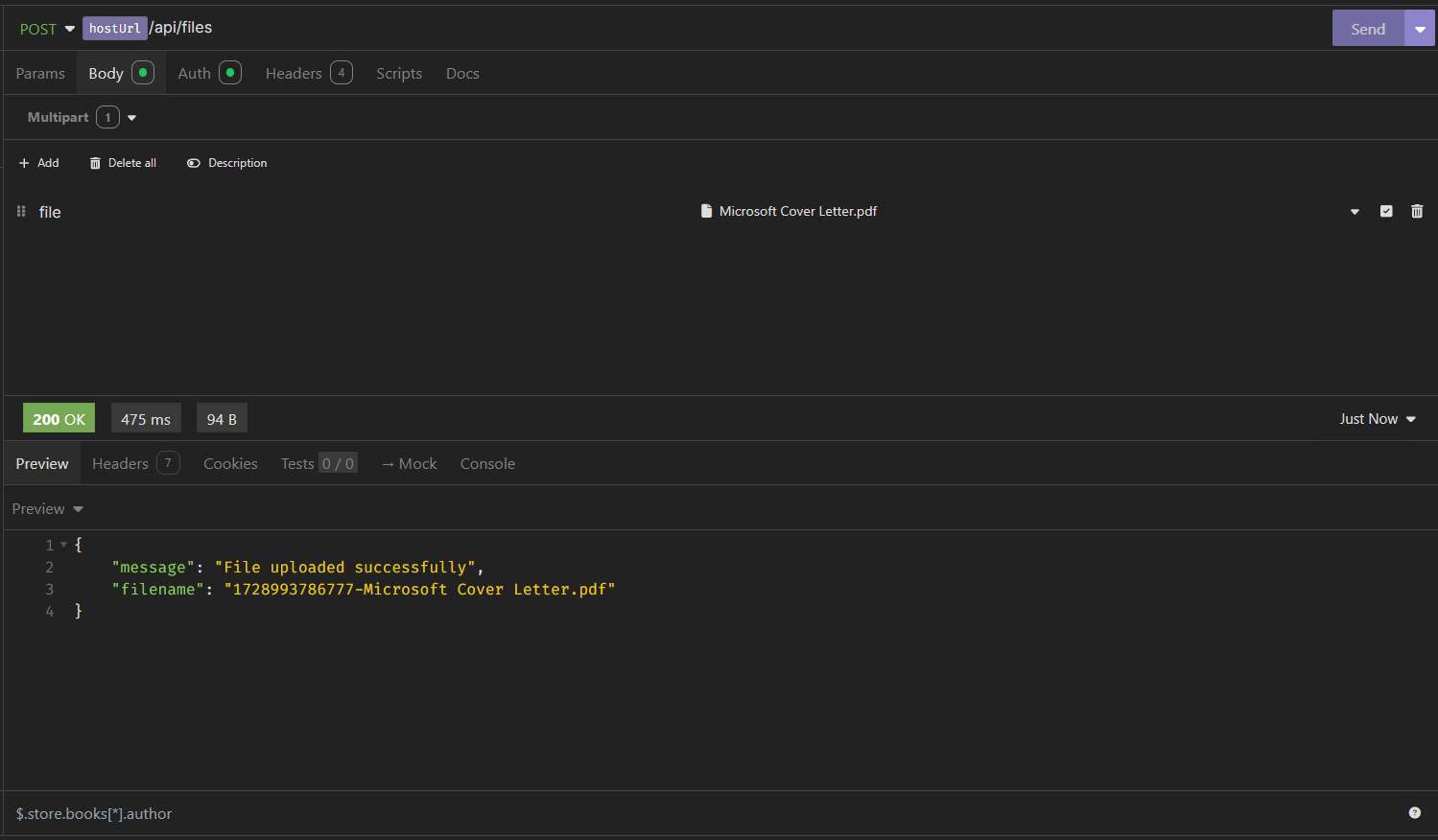
Update Book -



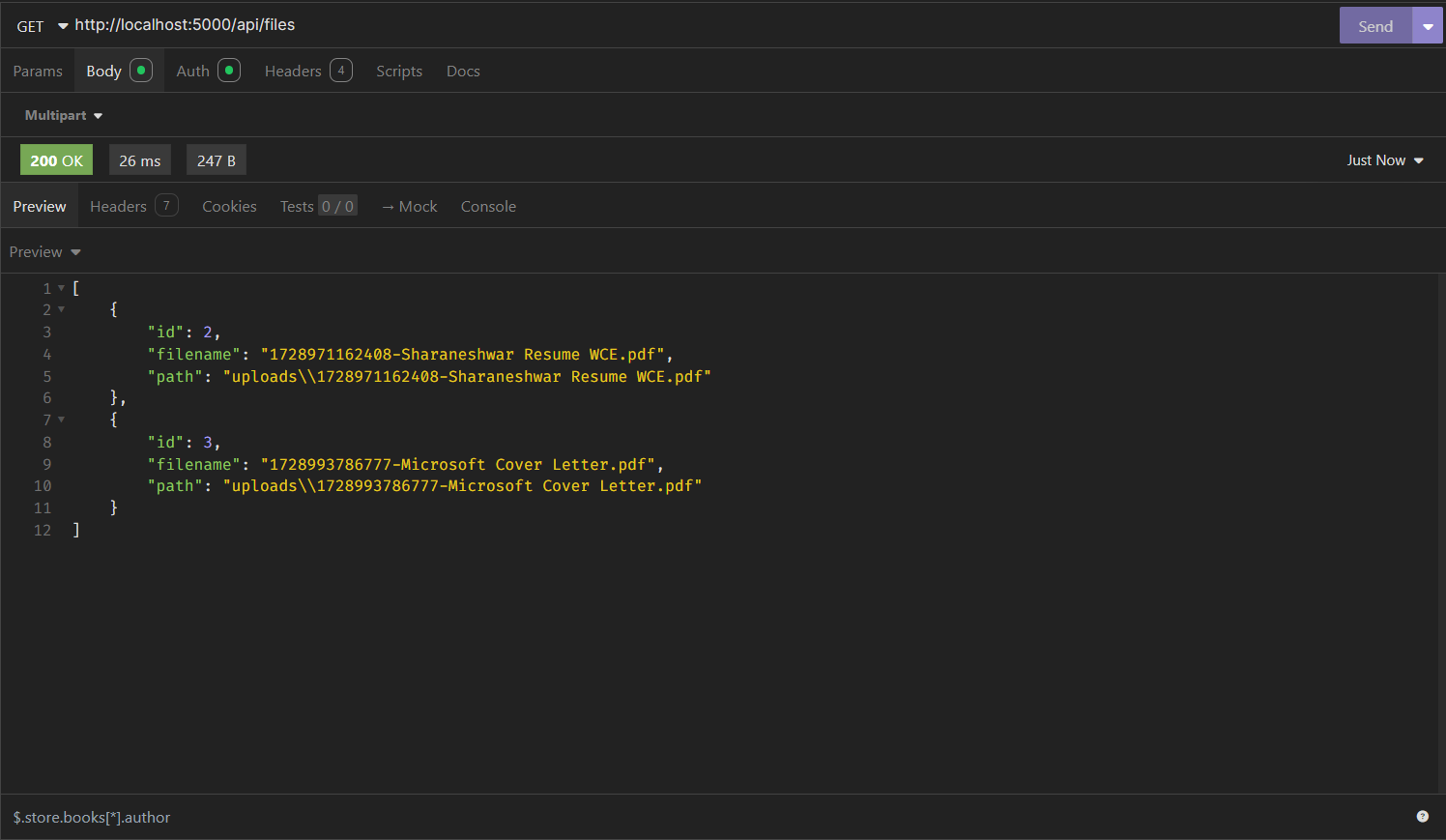
Delete Book -



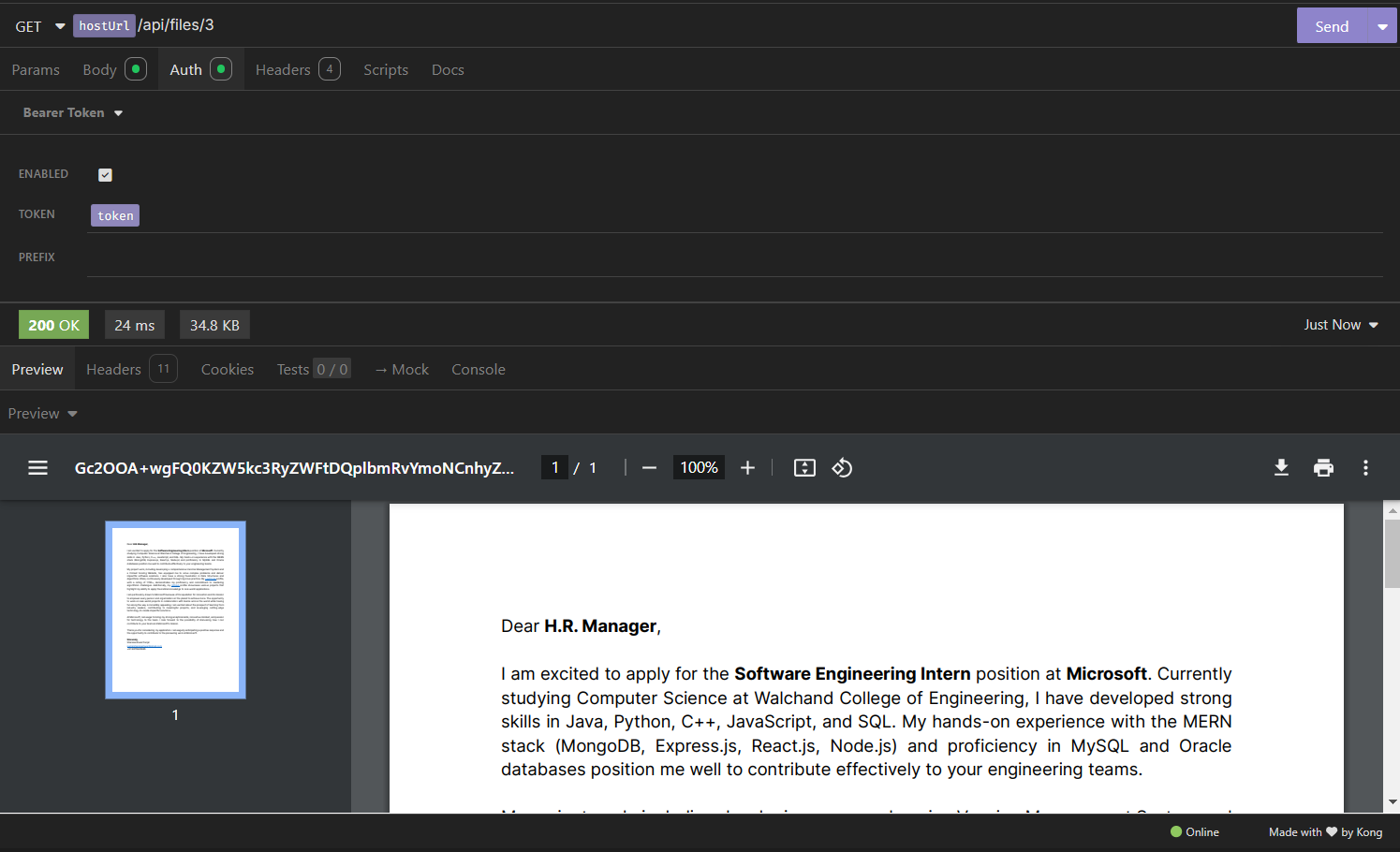
File Upload –



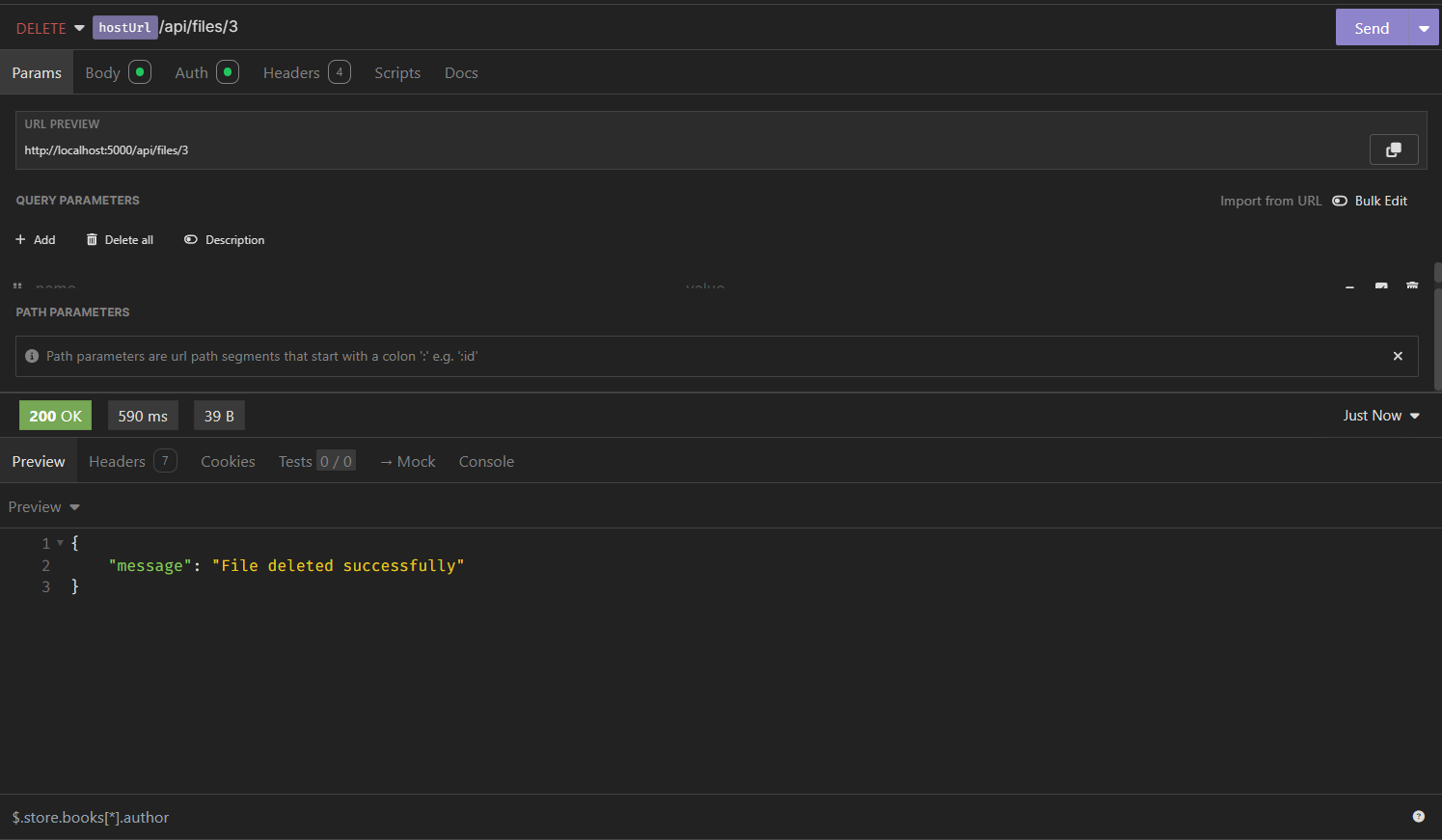
Retrieve All Files -



Download a File –



Delete a File -



MySQL Database –

