**FSD Laboratory 06**

**Name: Om Patel**

**Panel: F**

**Roll No: 16**

**Subject:- Full Stack Development**

**Submitted to: Prof. Vitthal Gutte**

**Aim:** Develop a set of REST API using Express and Node.

**Objectives:**

1. To define HTTP GET and POST operations.
2. To understand and make use of ‘REST’, ‘a REST endpoint’, ‘API Integration’, and ‘API Invocation’
3. To understand the use of a REST Client to make POST and GET requests to an API.

**Theory:**

1. What is REST API?

REST, which stands for Representational State Transfer, is an architectural style for designing networked applications. REST API (Application Programming Interface) is a set of rules and conventions for building and interacting with web services that adhere to the principles of REST.

Here are some key characteristics of a REST API:

1. Statelessness: Each request from a client to a server must contain all the information needed to understand and fulfill the request. The server should not store any information about the client's state between requests.

2. Client-Server Architecture: The client and the server are separate entities that communicate over a network. The client is responsible for the user interface and user experience, while the server is responsible for processing requests and managing resources.

3. Uniform Interface: A uniform and consistent way to interact with resources. This is typically achieved through a set of standard conventions for resource identification (using URIs/URLs), resource manipulation through representations (often in JSON or XML format), and the use of standard HTTP methods (GET, POST, PUT, DELETE).

4. Resource-Based: Resources, such as data objects or services, are identified by URIs. These resources can be manipulated using standard HTTP methods.

5. Stateless Communication: Each request from a client to a server must contain all the information needed to understand and fulfill the request. The server should not store any information about the client's state between requests.

6. Cacheability: Responses from the server can be explicitly marked as cacheable or non-cacheable, improving performance and reducing the load on the server.

7. Layered System: A client can interact with a system through a proxy or other intermediaries without knowing the details of the underlying implementation.

RESTful APIs are widely used in web development because of their simplicity, scalability, and ease of integration. They are commonly used in conjunction with the HTTP protocol, making use of standard HTTP methods (GET, POST, PUT, DELETE) for performing CRUD (Create, Read, Update, Delete) operations on resources.

1. Main purpose of REST API.

The main purpose of a REST API is to enable communication and data exchange between different software systems over the web. Here are some key purposes of REST APIs:

Interoperability: REST APIs provide a standardized way for different systems to communicate with each other. By adhering to a set of conventions and standards, REST APIs enable interoperability between software applications and services, regardless of the programming languages or technologies they are built with.

Simplicity and Ease of Use: REST APIs are designed to be simple and easy to understand. They use standard HTTP methods (GET, POST, PUT, DELETE) and are often based on widely adopted data formats like JSON or XML. This simplicity makes it easy for developers to integrate and work with APIs.

Scalability: REST APIs are well-suited for scalable architectures. The stateless nature of RESTful communication means that each request from a client to a server contains all the information needed, and the server does not need to store any client state between requests. This makes it easier to scale the system horizontally by adding more servers.

Resource Management: REST APIs are based on the concept of resources, which can represent data objects, services, or entities. Resources are identified by URIs, and the standard HTTP methods are used to perform operations on these resources. This resource-based approach simplifies the design and management of APIs.

Client-Server Architecture: REST APIs follow a client-server architecture, where the client is responsible for the user interface and user experience, and the server is responsible for processing requests and managing resources. This separation allows for independent development and scalability of client and server components.

**FAQ:**

1. What are HTTP Request types?

HTTP (Hypertext Transfer Protocol) defines several request methods, also known as HTTP request types or HTTP verbs. These methods indicate the desired action to be performed on a resource identified by a URI (Uniform Resource Identifier). The common HTTP request types include:

GET: Requests data from a specified resource. GET requests should only retrieve data and not have any other effect on the data.

POST: Submits data to be processed to a specified resource. The data is included in the body of the request. POST requests can create new resources or update existing ones.

PUT: Updates a resource or creates a new resource if it does not exist. The entire representation of the resource is replaced with the data sent in the request.

DELETE: Requests the removal of a resource identified by the URI. After a successful DELETE request, the resource is typically no longer available.

PATCH: Applies partial modifications to a resource. The data in the request body specifies the modifications to be applied.

**Output: Screenshots of the output to be attached.**

**Sample Problem Statements:**

**Creating and adding new book records in the book database using REST API.**

**Book-api.js**

const express = require('express')

const bodyParser = require('body-parser');

const cors = require('cors');

const app = express()

const port = 3000

let books = [{

    "isbn": "9781593275846",

    "title": "Eloquent JavaScript, Second Edition",

    "author": "Marijn Haverbeke",

    "publish\_date": "2014-12-14",

    "publisher": "No Starch Press",

    "numOfPages": 472,

},

{

    "isbn": "9781449331818",

    "title": "Learning JavaScript Design Patterns",

    "author": "Addy Osmani",

    "publish\_date": "2012-07-01",

    "publisher": "O'Reilly Media",

    "numOfPages": 254,

},

{

    "isbn": "9781449365035",

    "title": "Speaking JavaScript",

    "author": "Axel Rauschmayer",

    "publish\_date": "2014-02-01",

    "publisher": "O'Reilly Media",

    "numOfPages": 460,

}];

app.use(cors());

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json());

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

    const book = req.body;

    // output the book to the console for debugging

    console.log(book);

    books.push(book);

    res.send('Book is added to the database');

});

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

    res.json(books);

});

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

    // reading isbn from the URL

    const isbn = req.params.isbn;

    // searching books for the isbn

    for (let book of books) {

        if (book.isbn === isbn) {

            res.json(book);

            return;

        }

    }

    // sending 404 when not found something is a good practice

    res.status(404).send('Book not found');

});

app.delete('/book/:isbn', (req, res) => {

    // reading isbn from the URL

    const isbn = req.params.isbn;

    // remove item from the books array

    books = books.filter(i => {

        if (i.isbn !== isbn) {

            return true;

        }

        return false;

    });

    // sending 404 when not found something is a good practice

    res.send('Book is deleted');

});

app.post('/book/:isbn', (req, res) => {

    // reading isbn from the URL

    const isbn = req.params.isbn;

    const newBook = req.body;

    // remove item from the books array

    for (let i = 0; i < books.length; i++) {

        let book = books[i]

        if (book.isbn === isbn) {

            books[i] = newBook;

        }

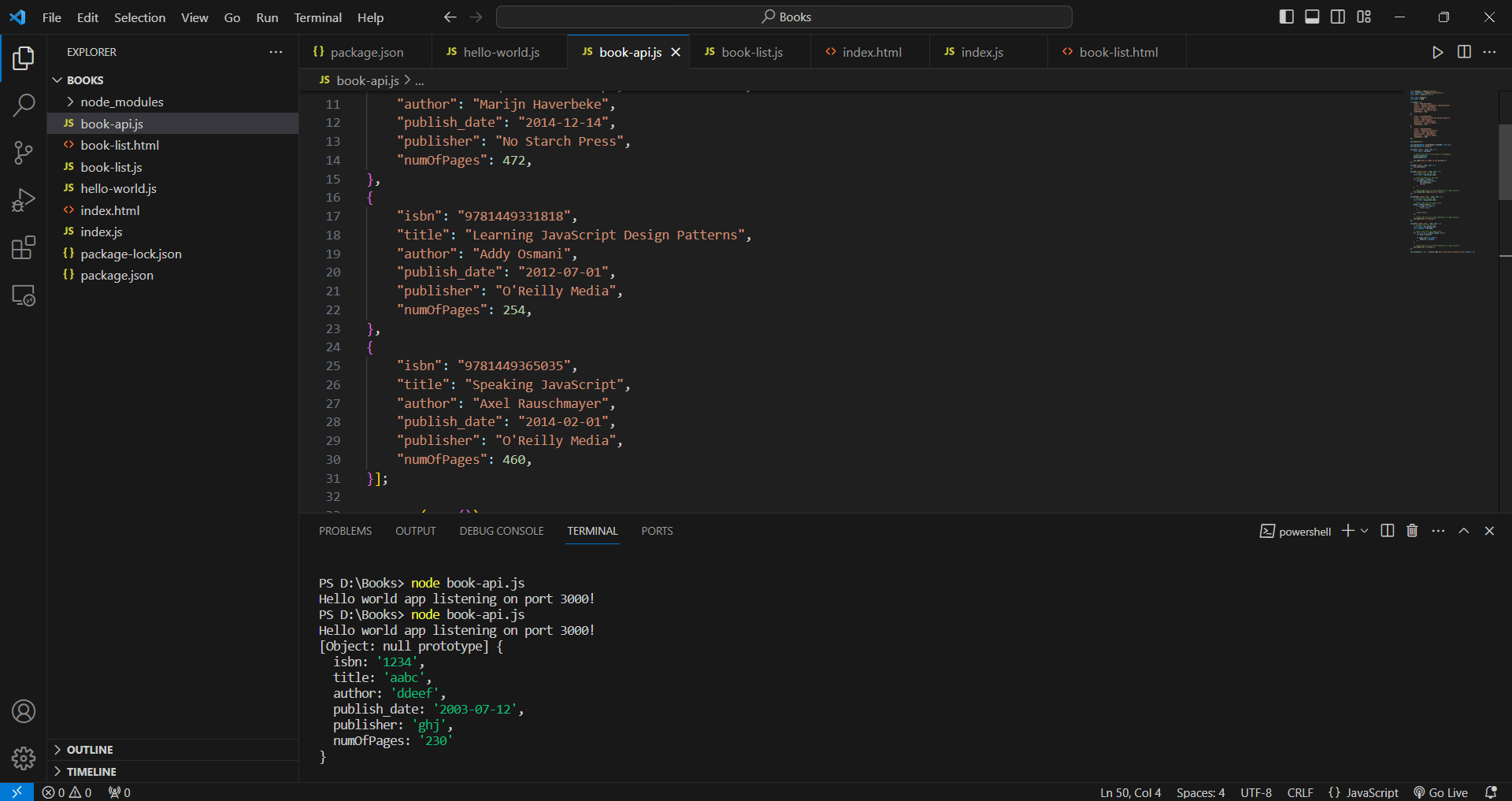
    }

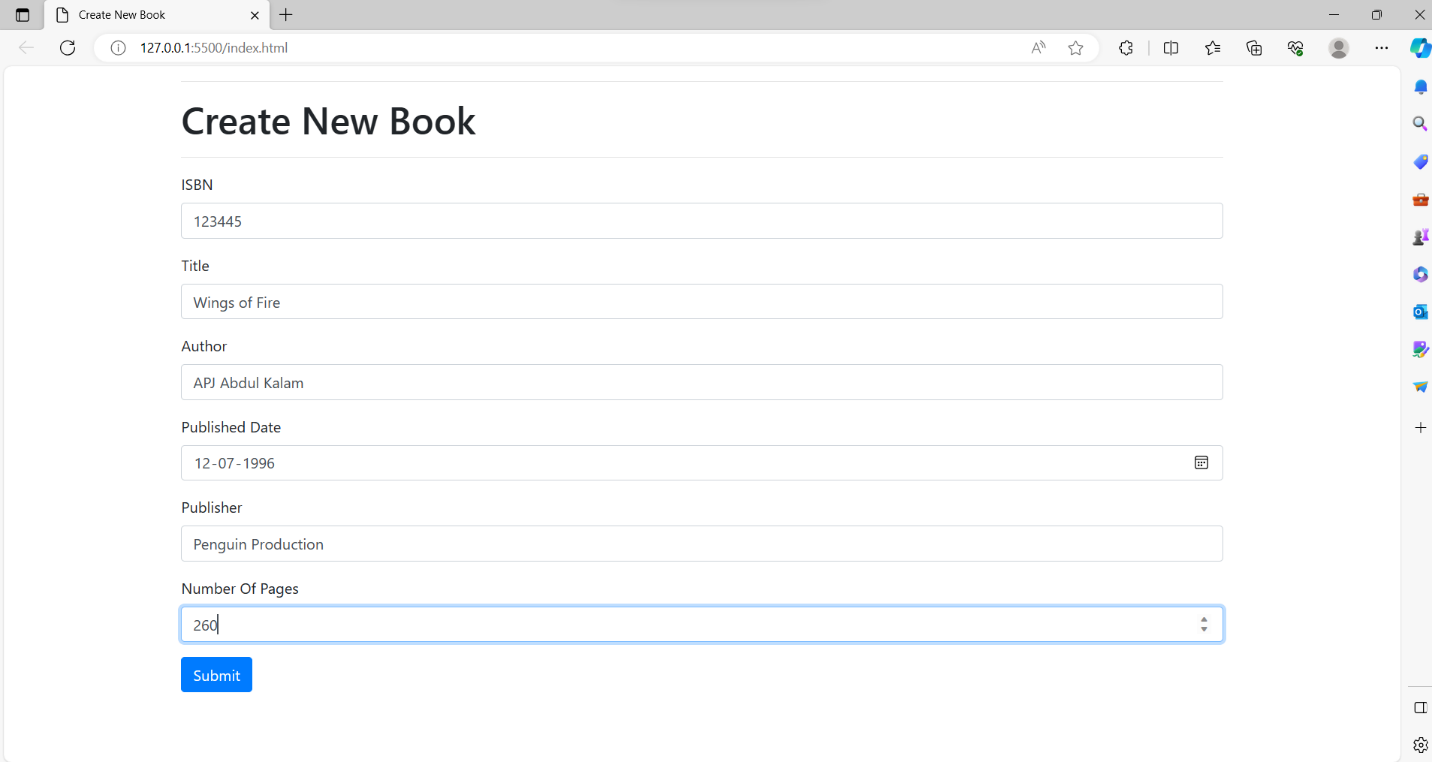
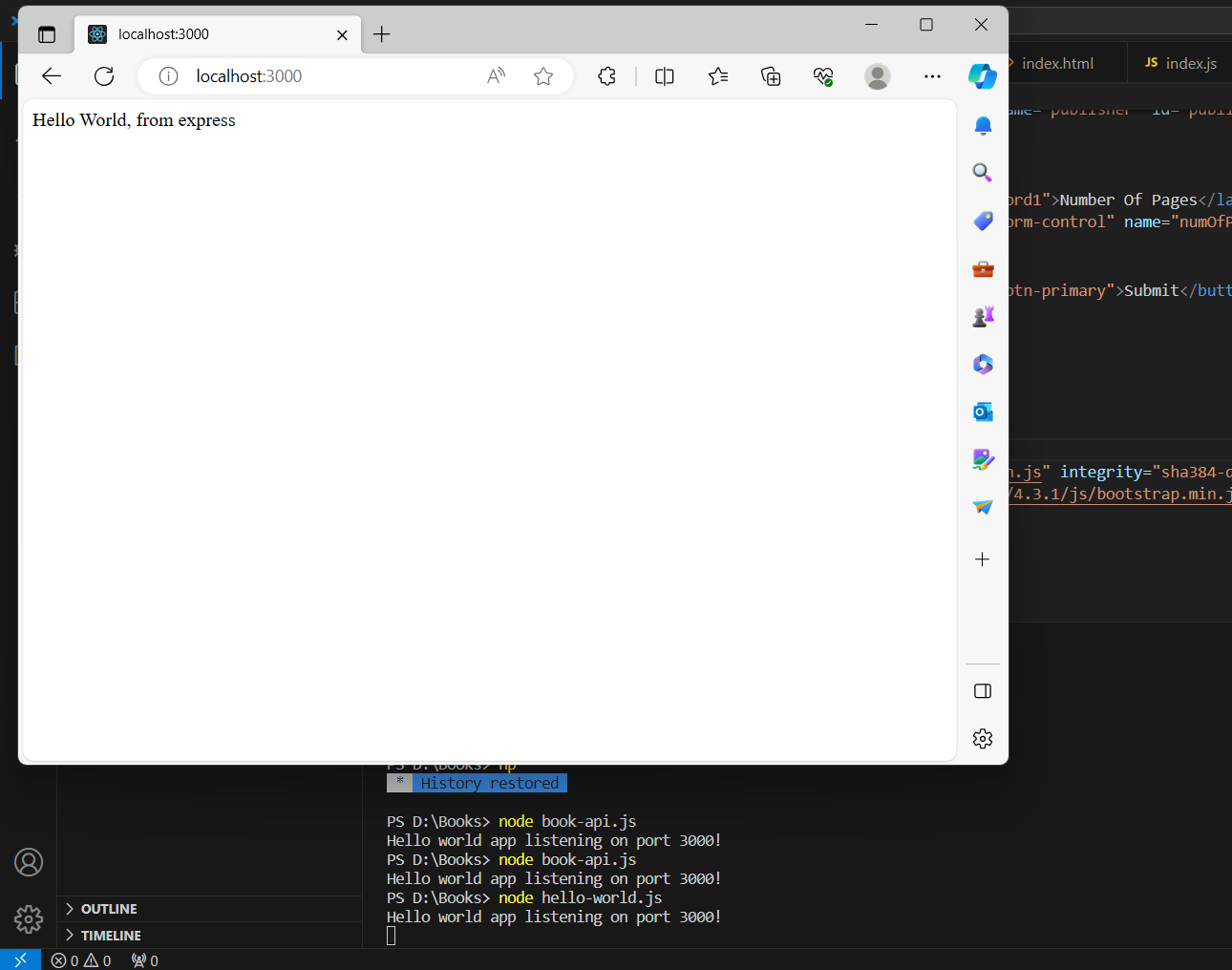
    // sending 404 when not found something is a good practice

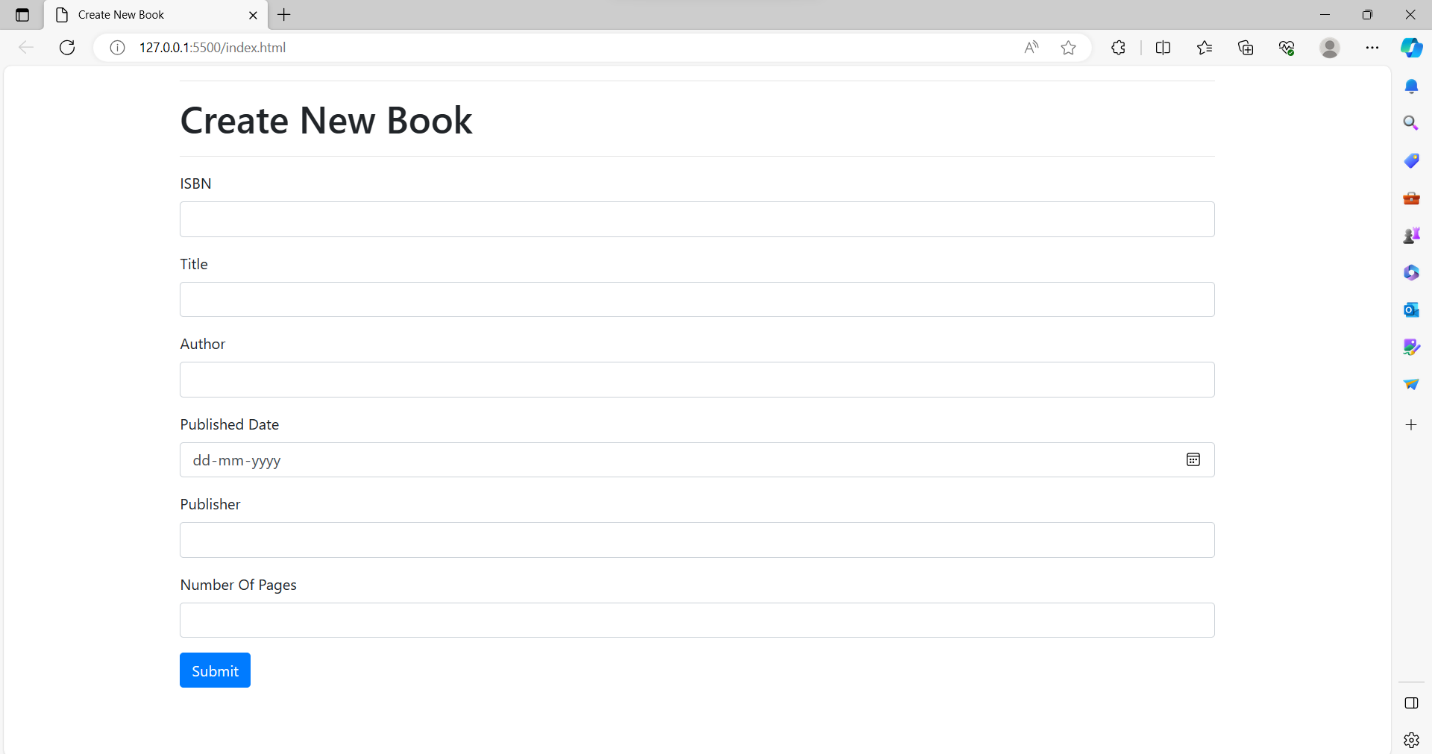
    res.send('Book is edited');

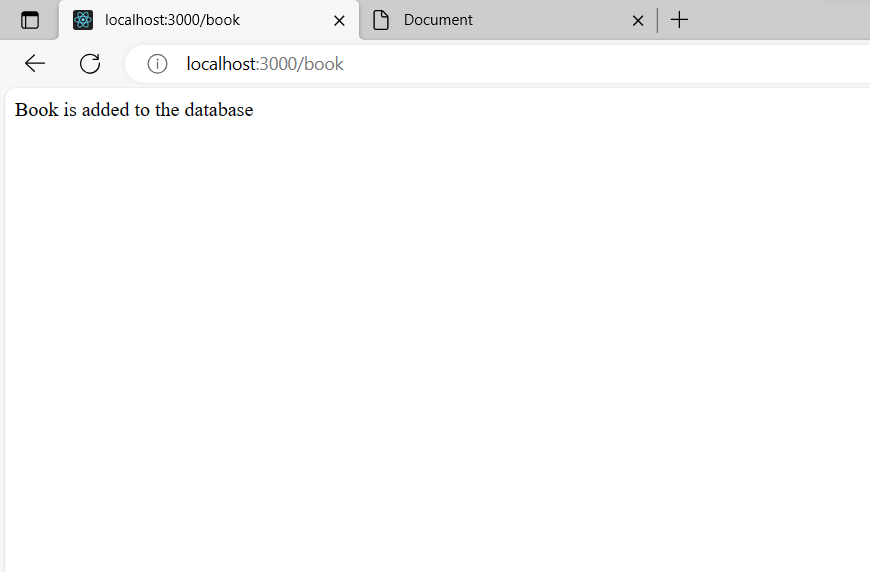
});

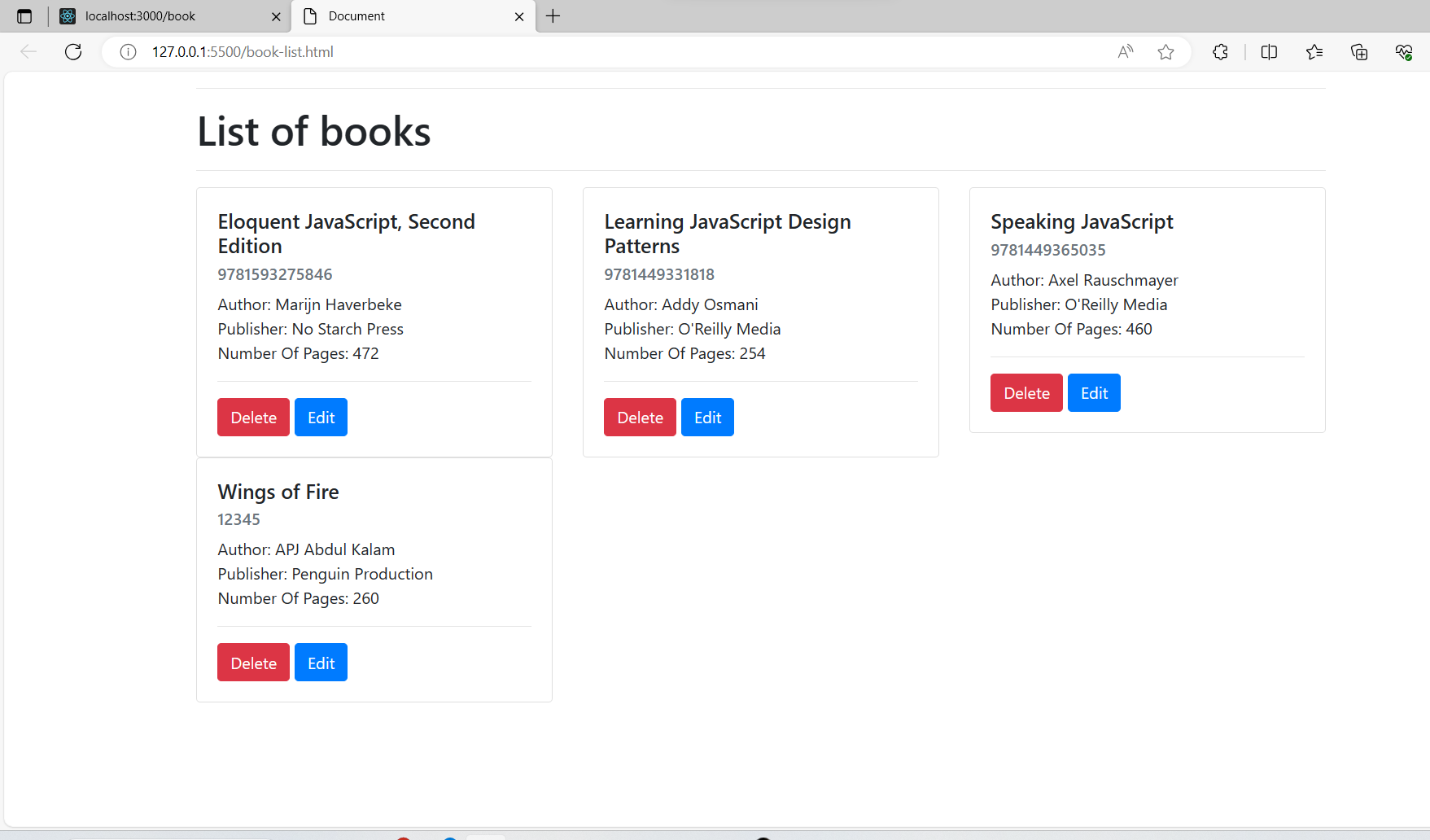
app.listen(port, () => console.log(`Hello world app listening on port ${port}!`));

****

****

****

****

****

Help Link:

https://stackabuse.com/building-a-rest-api-with-node-and-express/