GIT Repo URL: https://github.com/prateekp1304/FSD

Roll No: 08
Name: Prateek Patel
PRN: 1032210532

FSD Laboratory 06

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 APIs (Application Programming Interfaces) are a set of rules and conventions for building and interacting with web services that adhere to the principles of REST.

   Key characteristics of a REST API include:

   - Statelessness: Each request from a client to a server contains all the information needed to understand and fulfill that request. The server does not store any information about the client's state between requests.
   - Resource-Based: Resources, such as data or services, are identified by URIs (Uniform Resource Identifiers), and they are manipulated using standard HTTP methods (GET, POST, PUT, DELETE, etc.).
   - Representation: Resources can have different representations (e.g., JSON, XML), and clients interact with these representations.

   In a RESTful API, communication is typically done over HTTP, and the API is designed to be simple, scalable, and stateless.

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

   - Interoperability: REST APIs enable interoperability between different systems and platforms. They allow diverse applications to communicate and share data regardless of their underlying technologies.

- Simplicity and Scalability: REST APIs are designed to be simple, with a small set of well-defined principles. This simplicity makes them easy to understand and implement. Additionally, RESTful services can scale well since they are stateless, and each request contains all the information needed to process it.

- Standardization: REST APIs use standard HTTP methods (GET, POST, PUT, DELETE), making it easy to leverage existing infrastructure and tools. This standardization simplifies development and integration processes.

- Resource Management: REST APIs are resource-centric, with resources identified by unique URIs. This makes it easy to manage and manipulate resources using the standard HTTP methods. For example, to retrieve data, you use a GET request, and to update data, you use a PUT or POST request.

- Statelessness: The stateless nature of REST APIs means that each request from a client to a server is independent. The server does not store any information about the client's state between requests. This simplifies server implementation and enhances reliability.

- Scalability: RESTful architectures can be highly scalable due to their stateless nature and the use of standard protocols. This is crucial for applications that need to handle a large number of simultaneous requests.

Overall, REST APIs provide a standard and efficient way for systems to communicate over the web, making them a widely adopted choice for building web services and APIs.

FAQ:

1. What are HTTP Request types?

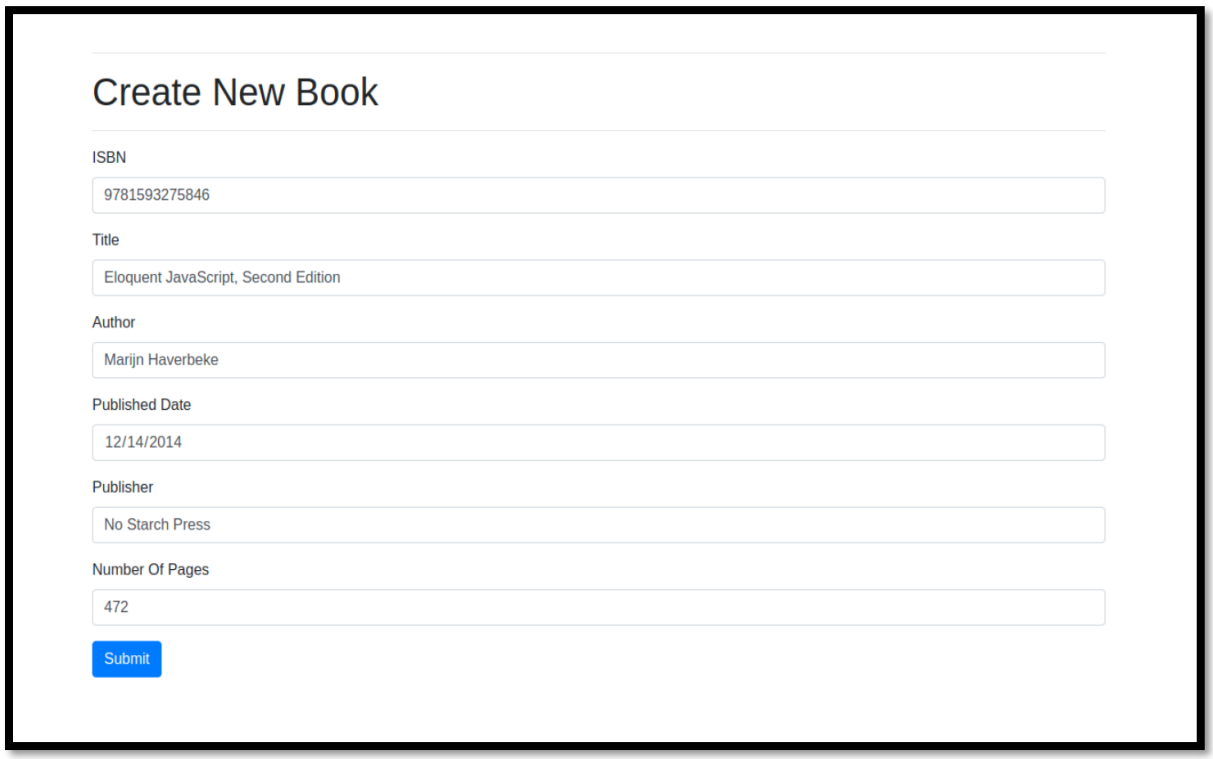
Ans: HTTP (Hypertext Transfer Protocol) defines several request methods or types that indicate the desired action to be performed for a given resource. Each HTTP request type has a specific purpose and is associated with a particular operation. The common HTTP request methods include:

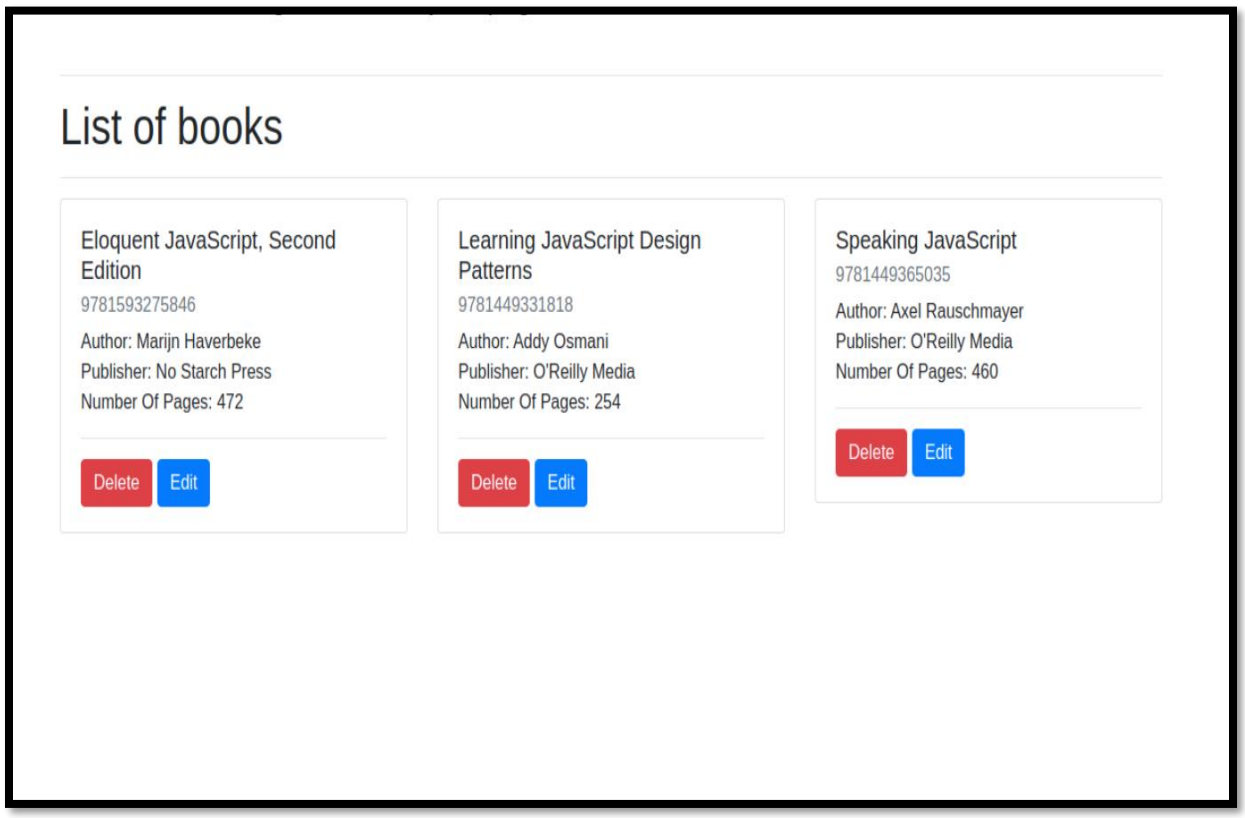
1. GET:
   - Purpose: Retrieve data from a specified resource.
   - Example: Fetching the contents of a web page, retrieving user information, etc.
   - Safe and Idempotent: Generally considered safe (does not modify the resource) and idempotent (repeating the request has the same effect as making it once).
2. POST:
   - Purpose: Submit data to be processed to a specified resource.
   - Example: Submitting form data, uploading a file, etc.
   - Not Idempotent: Not considered idempotent, as repeating the request may have different effects.
3. PUT:
   - Purpose: Update a resource or create a new resource if it does not exist.
   - Example: Updating user details, uploading a resource to a specific URI, etc.
   - Idempotent: Considered idempotent; repeating the request has the same effect.

4. PATCH:
    - Purpose: Partially update a resource.
    - Example: Updating only certain fields of a resource without modifying the entire resource.
    - Idempotent: Not guaranteed to be idempotent, but it can be in some cases.
5. DELETE:
    - Purpose: Remove a resource.
    - Example: Deleting a user account, removing a file, etc.
    - Idempotent: Considered idempotent; repeating the request has the same effect.
6. HEAD:
    - Purpose: Retrieve the headers of a resource without the actual data.
    - Example: Checking if a resource has been modified without downloading the entire content.
    - Safe and Idempotent: Generally considered safe and idempotent.
7. OPTIONS:
    - Purpose: Get information about the communication options available for a resource.
    - Example: Checking the allowed methods for a resource, understanding CORS (Cross-Origin Resource Sharing) headers.
    - Safe and Idempotent: Generally considered safe and idempotent.
8. TRACE:
    - Purpose: Echo back the received request to the client. (Primarily used for diagnostic purposes.)
    - Example: Checking how a request changes as it passes through various proxies.
    - Safe and Idempotent: Generally considered safe and idempotent.
9. CONNECT:
    - Purpose: Establish a tunnel to the server identified by the target resource.
    - Example: Used for setting up a secure communication channel via a proxy.
    - Not Safe or Idempotent: Typically not considered safe or idempotent.

These HTTP request methods provide a standardized way for clients to interact with web servers, specifying the desired action for a given resource. The appropriate use of these methods contributes to the efficiency, security, and reliability of web communications.

Output Screenshots:





Sample Problem Statements:

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

Help Link:

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