MIT WORLD PEACE UNIVERSITY

Full Stack Development
Third Year B.Tech, Semester 1

Design REST API using Node and Express

ASSIGNMENT 6

Prepared By A1−02. Mayur Behere
Batch A

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 (Representational State Transfer) API is an architectural style used for designing networked applications. It is an approach that defines a set of constraints to create web services that are well-structured, scalable, and easily maintainable.

Key principles of a RESTful API include:

- Stateless Communication: Each request from a client to the server must contain all the necessary information to understand and process that request. The server doesn't store any client state between requests, improving scalability and reliability.
- Client-Server Architecture: Separation of concerns between the client and server. The client is responsible for the user interface and user experience, while the server handles data storage, processing, and management.
- Uniform Interface: It specifies a uniform and standardized way to interact with resources using well-defined operations (HTTP methods) like GET, POST, PUT, DELETE, etc. Resources are identified by URIs (Uniform Resource Identifiers).
- Resource-Based: In a RESTful API, resources (data objects or representations) are the key abstraction. These resources can be manipulated using standard HTTP methods.
- Statelessness: REST APIs should be stateless, meaning the server should not retain any information about the client's state between

requests. Each request from a client must contain all the necessary information.

- Hypermedia as the Engine of Application State (HATEOAS): It allows the server to provide information to the client about what actions the client can perform next based on the current state of the application. This makes the API self-descriptive and discoverable.
- Cacheability: Responses from the server can be explicitly or implicitly marked as cacheable or non-cacheable. This helps in improving performance and reducing server load by allowing clients to cache responses

RESTful APIs are widely used in web development, allowing systems to communicate over the internet in a standardized and interoperable manner. They enable different systems and services to interact with each other, facilitating the exchange of data and functionalities across diverse platforms and technologies.

2] Main Purpose of REST API.

=>

The main purpose of a RESTful API (Representational State Transfer API) is to facilitate the interaction and communication between systems over the internet. Its primary goals include:

- Interoperability: REST APIs enable different systems and software applications to communicate and interact with each other, regardless of their underlying technologies, languages, or platforms. They provide a standardized way to exchange data and functionality, promoting interoperability between diverse systems.
- Scalability: RESTful APIs are designed to be scalable, allowing for easy integration and handling of increased demand. They are stateless, meaning they don't store client information between requests, facilitating better scalability and load distribution.
- Simplicity and Ease of Use: REST APIs are relatively simple and easy to understand due to their usage of standard HTTP methods (GET, POST, PUT, DELETE, etc.) and resources identified by URIs. This simplicity makes them accessible for developers to implement and use.
- Flexibility and Extensibility: They offer flexibility in terms of data formats (JSON, XML, etc.) and can support various data types. REST APIs can also be extended to accommodate evolving requirements

without disrupting existing functionality, providing adaptability for future changes.

- Statelessness: RESTful APIs do not retain any client information between requests, which promotes reliability and simplifies server-side logic. Each request contains all the necessary information for the server to process it independently.
- Support for Multiple Platforms and Devices: RESTful APIs enable the development of applications that can run on various platforms (web, mobile, IoT, etc.) and devices. They facilitate cross-platform compatibility and accessibility.
- Promotes Decoupling and Modularity: RESTful APIs promote a decoupled architecture, allowing for separation between the client and server. This separation allows components to be modified or updated independently without affecting other parts of the system.

Overall, the main purpose of a RESTful API is to provide a standardized, efficient, and flexible way for systems to communicate, exchange data, and access functionality across different applications, platforms, and devices over the internet.

.

FAQ :

1] What are HTTP Request types?

=>

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

- GET: Used to request data from a specified resource. It retrieves information without modifying the server's state or data. It's commonly used for fetching data like web pages, images, files, etc.
- POST: Submits data to be processed to a specified resource. It sends data to the server to create or update a resource, often used in form submissions and when uploading files.
- PUT: Sends data to the server to update or replace the entire resource at the specified URI. It updates an existing resource or creates a new resource if it does not exist at the given URI.

- DELETE: Requests the removal of a specific resource from the server. It deletes the resource identified by the URI.
- PATCH: Used to partially update a resource on the server. It applies modifications to a resource while leaving the rest of the resource unchanged.
- HEAD: Similar to GET, but it retrieves only the headers of the response without the actual body content. It's used to obtain metadata about a resource without transferring the entire content.
- OPTIONS: Requests information about the communication options available for a resource or server. It retrieves the HTTP methods supported by the server for a particular resource.
- TRACE: Echoes back the received request to the client, allowing clients to see what changes or additions have been made by intermediate servers.

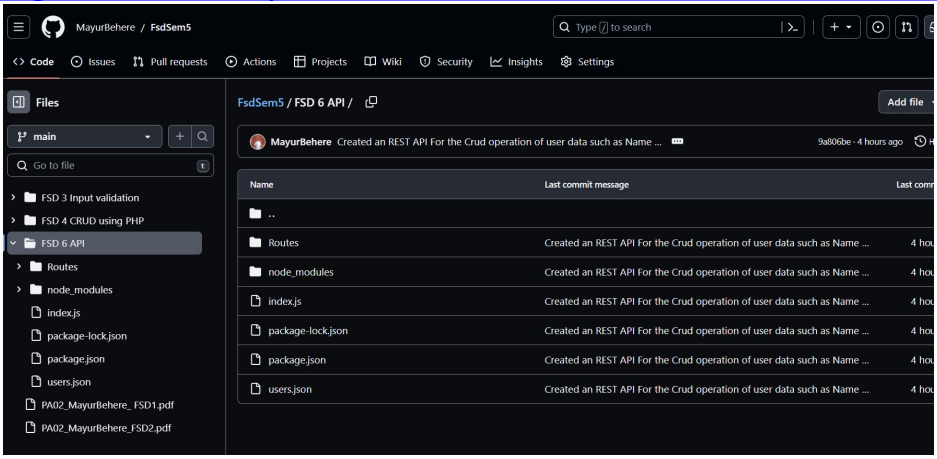
Each HTTP request method has specific semantics and purposes, and they enable various interactions between clients (such as web browsers or applications) and servers to perform actions like retrieving data, creating, updating, or deleting resources, and retrieving information about server capabilities.

Platform :

Browser : Chrome (localhost)
App : Vscode , POSTMAN

Output & SRC :

Github Repo :
https://github.com/MayurBehere/FsdSem5/tree/main/FSD%206%20API

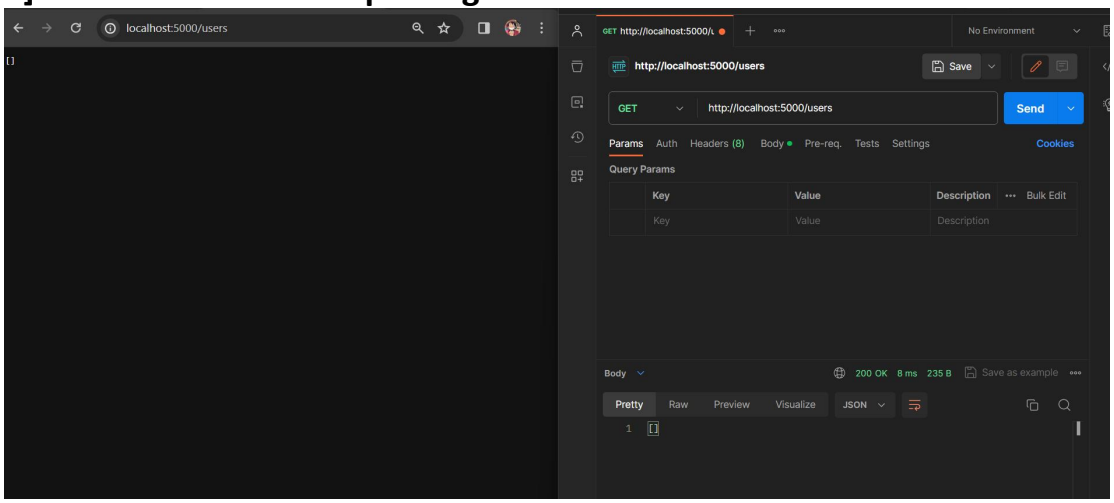


API Working :

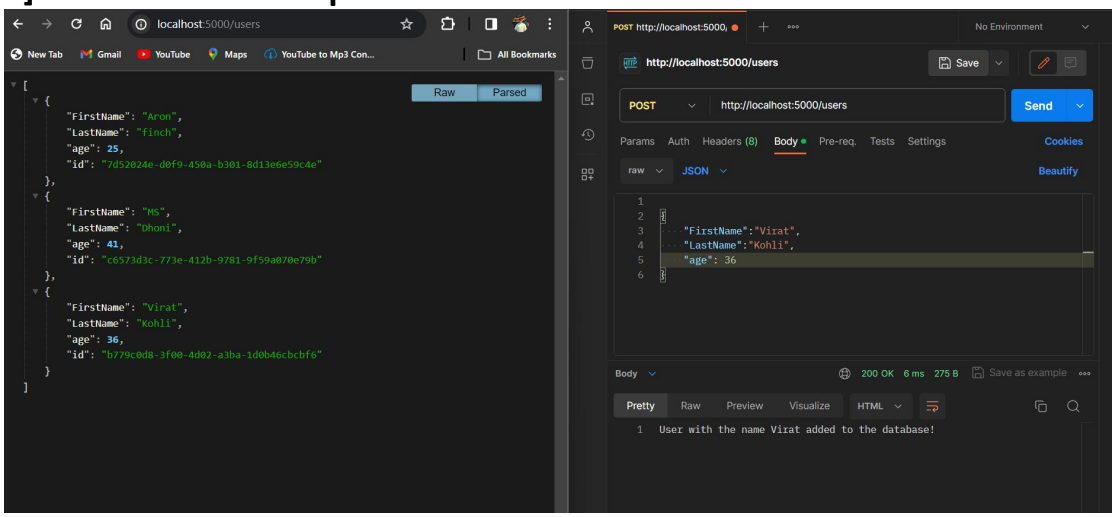
1] Home page

Hello User, Welcome to the Home Page of the API Server

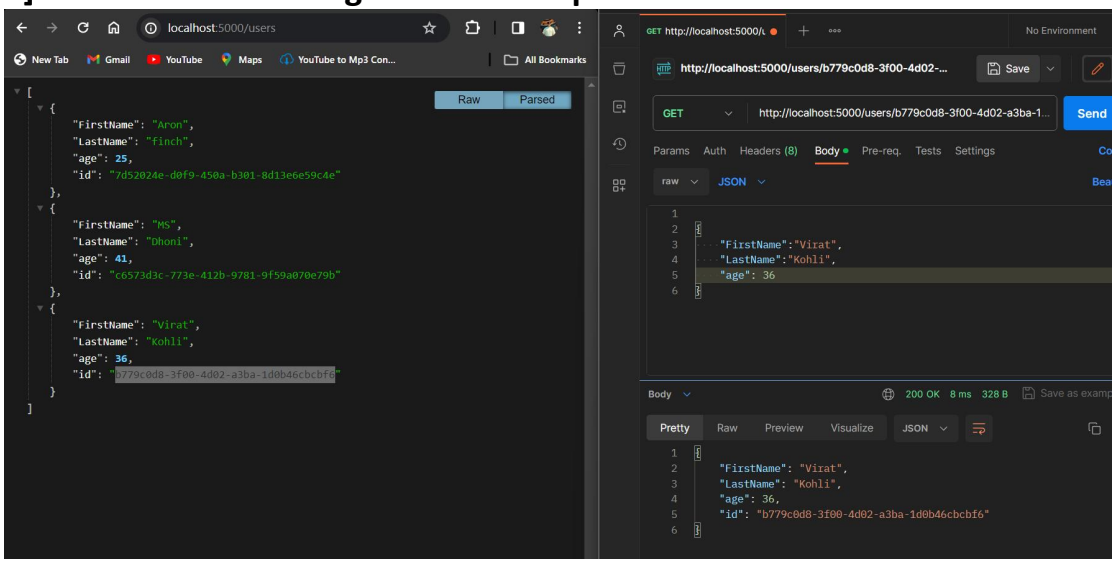
2] Started Postman for posting data into Database



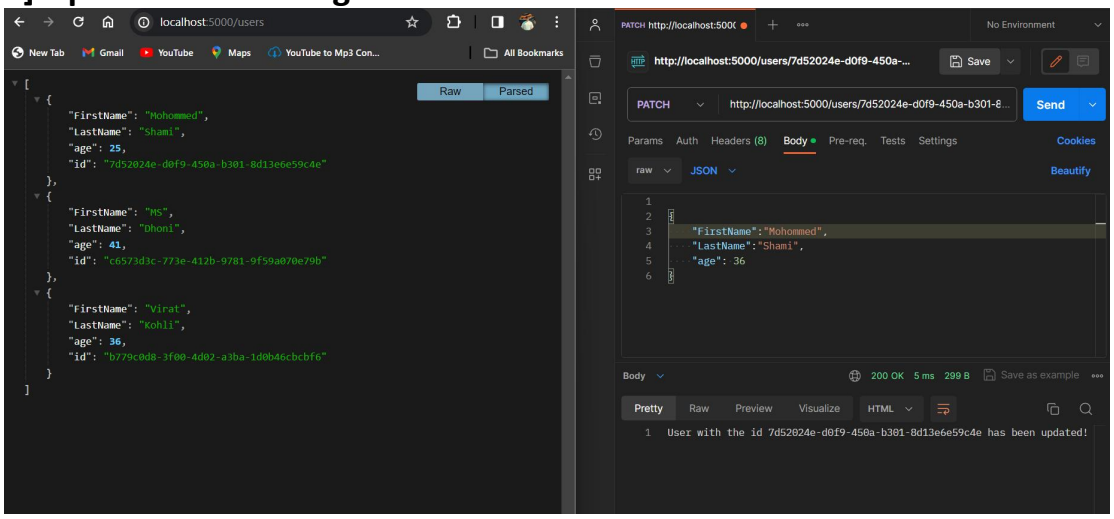
3] Performed POST operation into DB



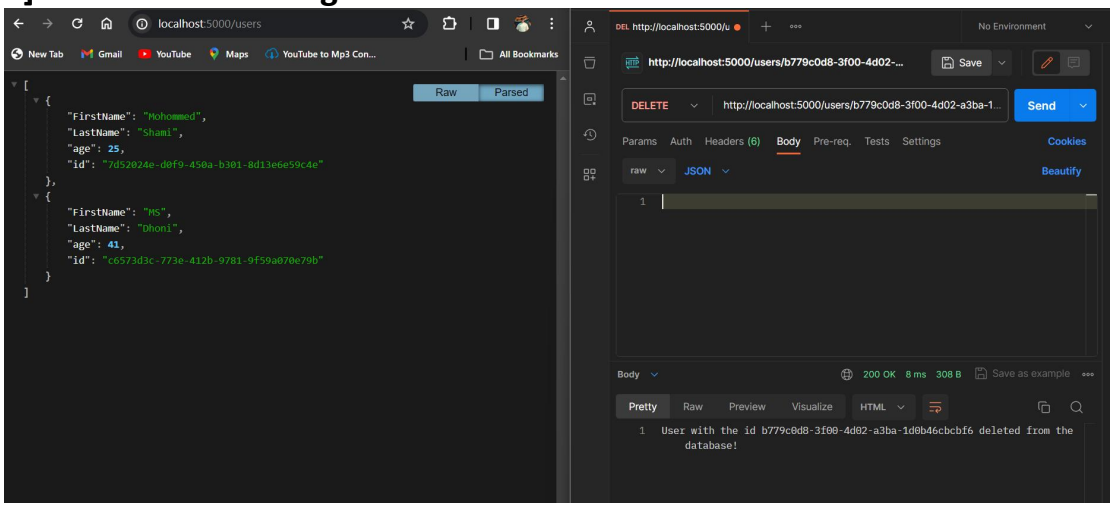
4] Retrieved data using GET and Unique ID



5] Updated Data using PATCH



6] Deleted Data using DATA Function




```
import express from "express";
import {v4 as uuidv4} from "uuid";
uuidv4();

const router = express.Router();

let users = [ ];

router.get("/", (req, res) => {
    console.log(users);
    res.send(users);
    console.log("GET ROUTE REACHED");
});

router.post("/", (req, res) => {
    console.log("POST ROUTE REACHED");
    const NewUser = req.body;
    const UserWithID = {...NewUser, id: uuidv4()};
    users.push(UserWithID);
    res.send(`User with the name ${NewUser.FirstName} added to the
database!`);
});

router.get("/:id", (req, res) => {
    const {id}= req.params;
    const FoundUser = users.find((user) => user.id === id);
    res.send(FoundUser);
});

router.delete("/:id", (req, res) => {
    const {id}= req.params;
    users = users.filter((user) => user.id !== id);
    res.send(`User with the id ${id} deleted from the database!`);
});

router.patch("/:id", (req, res) => {
    const {id}= req.params;
    const {FirstName, LastName, Age} = req.body;
    const user = users.find((user) => user.id === id);
    if(FirstName) user.FirstName = FirstName;
    if(LastName) user.LastName = LastName;
    if(Age) user.Age = Age;
    res.send(`User with the id ${id} has been updated!`);
});
export default router;
```

Result :

Created a REST API for CRUD operation using Express, Node , Nodemon and POSTMAN.