FSD Laboratory 06

Arin Khopkar, TY B.Tech CSE, Panel I, I2, 62, 1032221073

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?
2. Main purpose of REST API.

REST API (Representational State Transfer Application Programming Interface) is a standardized architectural style for creating web services. It allows different systems to communicate with each other over the web using HTTP requests.

Key Concepts of REST:

1. **Client-Server Architecture**: REST separates the client (user interface) from the server (data storage), allowing each part to evolve independently.
2. **Stateless**: Each request from a client to a server must contain all the information needed to understand and process the request. The server doesn't store the client's state between requests.
3. **Uniform Interface**: REST APIs have consistent, well-defined methods (like GET, POST, PUT, DELETE) for different types of operations.
4. **Resource-Based**: Data and functionality are considered resources (like user data, articles, etc.), each with its own unique URL.
5. **Representations**: Resources are typically represented in formats like JSON or XML.
6. **Cacheability**: Responses from the server can be labeled as cacheable or non-cacheable to improve performance.

Purpose of REST API:

The **main purpose** of a REST API is to provide a standardized way for different software applications or systems to interact and exchange data over the web. Some specific purposes include:

1. **Interoperability**: Enable communication between various systems regardless of the platform or language.
2. **Scalability**: REST APIs allow systems to scale efficiently by separating concerns between client and server.
3. **Flexibility**: It supports multiple types of clients, from web browsers to mobile apps.
4. **Simplicity**: The uniform interface and stateless nature make REST APIs easier to develop, maintain, and understand.

5. **Modularity**: Different services can be developed and deployed independently, promoting microservices architecture.

FAQ:

1. What are HTTP Request types?

A1) HTTP request types, also known as **HTTP methods** or **verbs**, define the action the client wants to perform on a given resource in a RESTful API. Each method is used for a specific type of operation. Here are the most common HTTP request types:

1. GET

- **Purpose**: Retrieve data from the server.
- **Usage**: Used to fetch or read data from a resource (like a database or file system).
- **Example**: Fetch a list of users or details about a specific user.
  - **Request**: `GET /users/1`
- **Idempotent**: Yes (multiple identical requests result in the same response).

2. POST

- **Purpose**: Send data to the server to create a new resource.
- **Usage**: Used for creating a new resource, such as adding a new user or submitting a form.
- **Example**: Create a new user.
  - **Request**: `POST /users` (with the user details in the request body).
- **Idempotent**: No (multiple identical requests could create multiple resources).

3. PUT

- **Purpose**: Update or replace an existing resource.
- **Usage**: Used to fully replace or update a resource with new data. If the resource does not exist, it can create it.
- **Example**: Update a user's details.
  - **Request**: `PUT /users/1` (with the new user details in the request body).
- **Idempotent**: Yes (multiple identical requests result in the same resource state).

4. PATCH

- **Purpose**: Partially update an existing resource.
- **Usage**: Used to modify or update specific fields of a resource without replacing the entire resource.
- **Example**: Update only the email address of a user.
  - **Request**: `PATCH /users/1` (with the changes in the request body).
- **Idempotent**: Yes.

5. DELETE

- **Purpose**: Remove a resource from the server.

- **Usage**: Used to delete a resource, such as removing a user or deleting a file.
- **Example**: Delete a user.
  - **Request**: `DELETE /users/1`
- **Idempotent**: Yes (deleting a resource that is already deleted does not cause further action).

6. OPTIONS

- **Purpose**: Describe the communication options for a given resource.
- **Usage**: Used to get information about the communication methods supported by the server for a particular resource.
- **Example**: Check what HTTP methods are allowed for a specific resource.
  - **Request**: `OPTIONS /users`
- **Idempotent**: Yes.

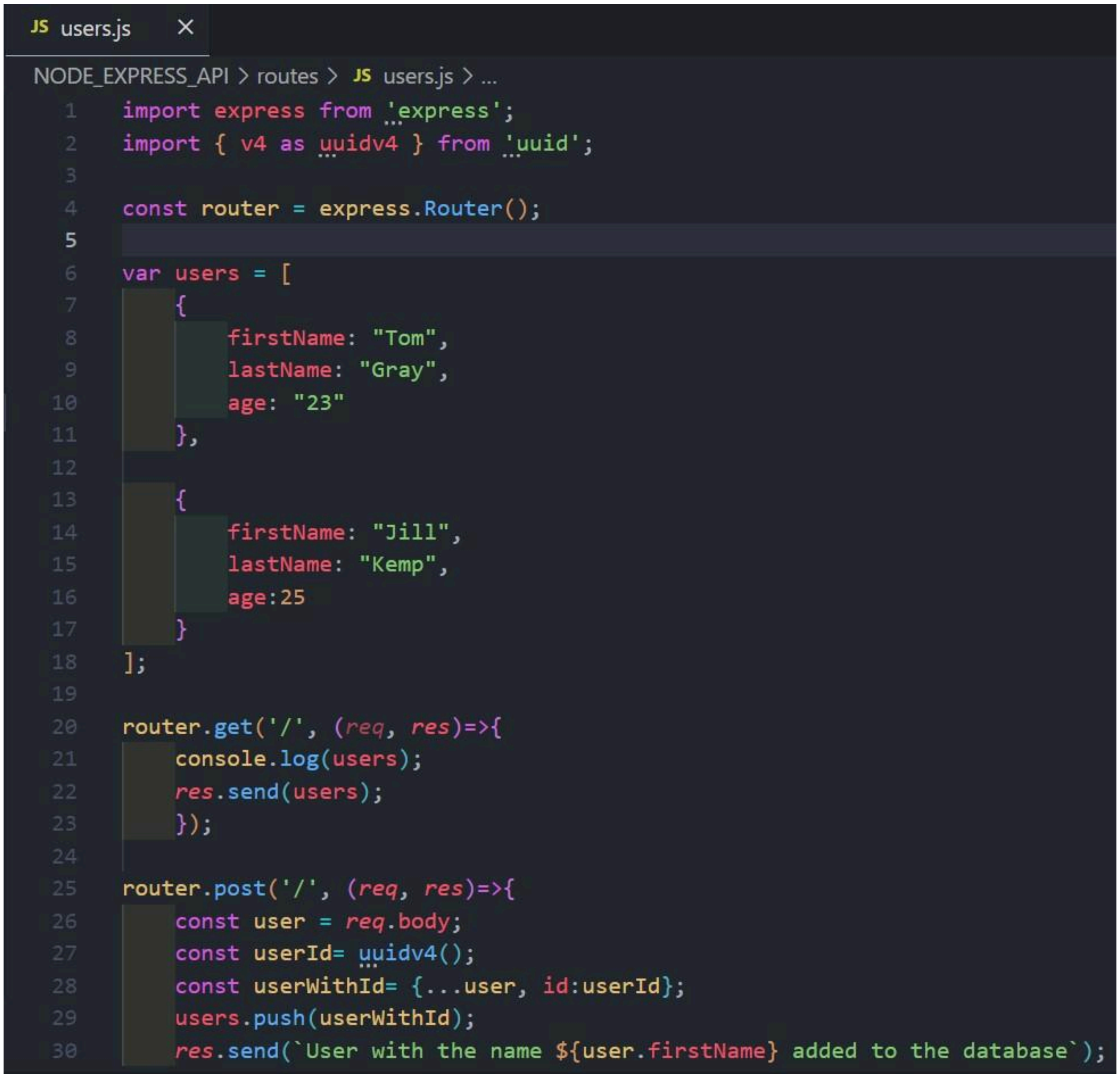
7. HEAD

- **Purpose**: Retrieve the headers of a resource, without the actual body content.
- **Usage**: Used to check meta-information about a resource (e.g., last modified time, content type) without transferring the entire resource.
- **Example**: Check if a resource exists or get metadata without fetching the full resource.
  - **Request**: `HEAD /users/1`
- **Idempotent**: Yes.

Less Common Methods:

- **TRACE**: Used for debugging; echoes back the received request so that the client can see what changes have been made by intermediary servers.
- **CONNECT**: Used for creating a network connection, commonly for proxy tunneling.

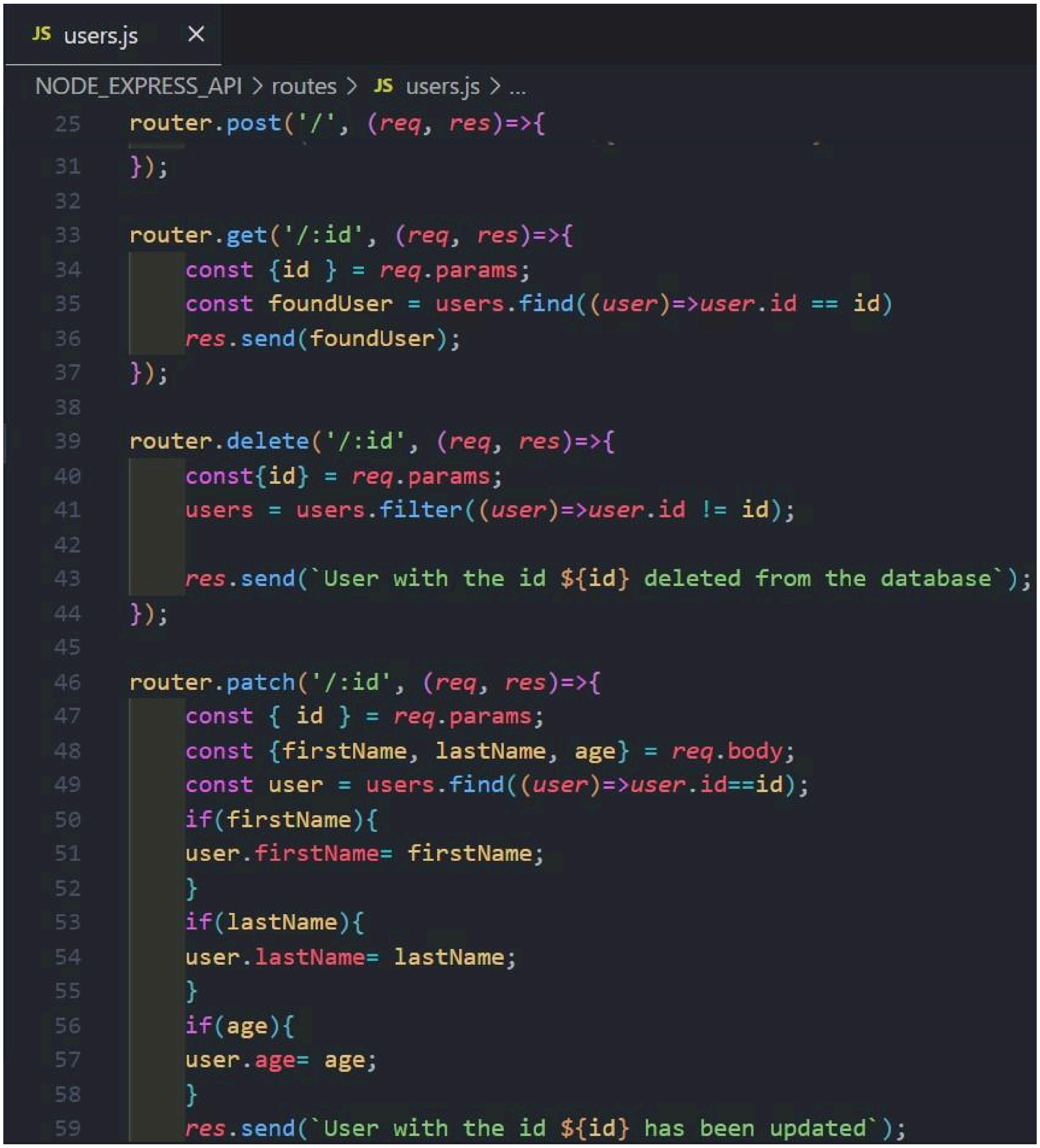
Code:



users.js

NODE_EXPRESS_API > routes > users.js > ...

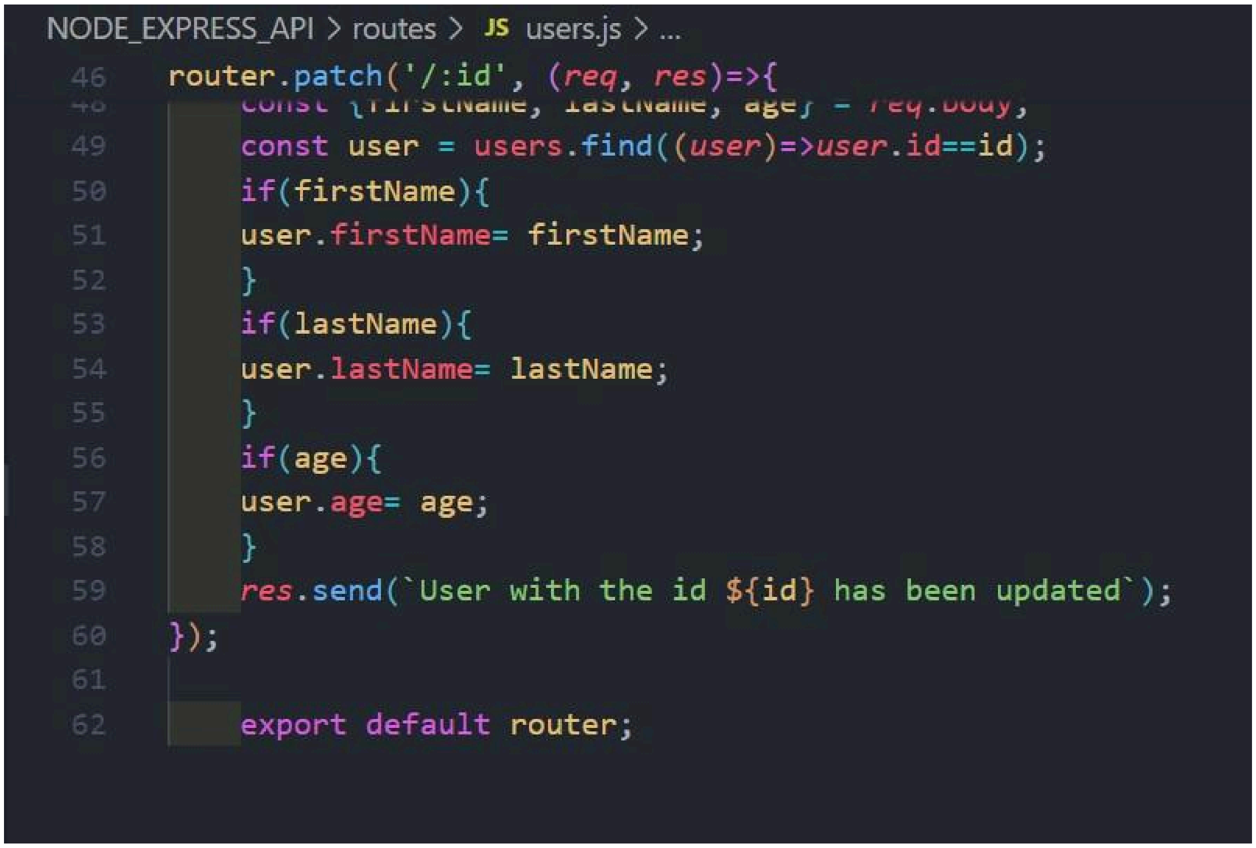
```
import express from 'express';
import { v4 as uuidv4 } from 'uuid';

const router = express.Router();

var users = [
    {
        firstName: "Tom",
        lastName: "Gray",
        age: "23"
    },

    {
        firstName: "Jill",
        lastName: "Kemp",
        age:25
    }
];

router.get('/', (req, res)=>{
    console.log(users);
    res.send(users);
    });

router.post('/', (req, res)=>{
    const user = req.body;
    const userId= uuidv4();
    const userWithId= {...user, id:userId};
    users.push(userWithId);
    res.send(`User with the name ${user.firstName} added to the database`);
```



users.js

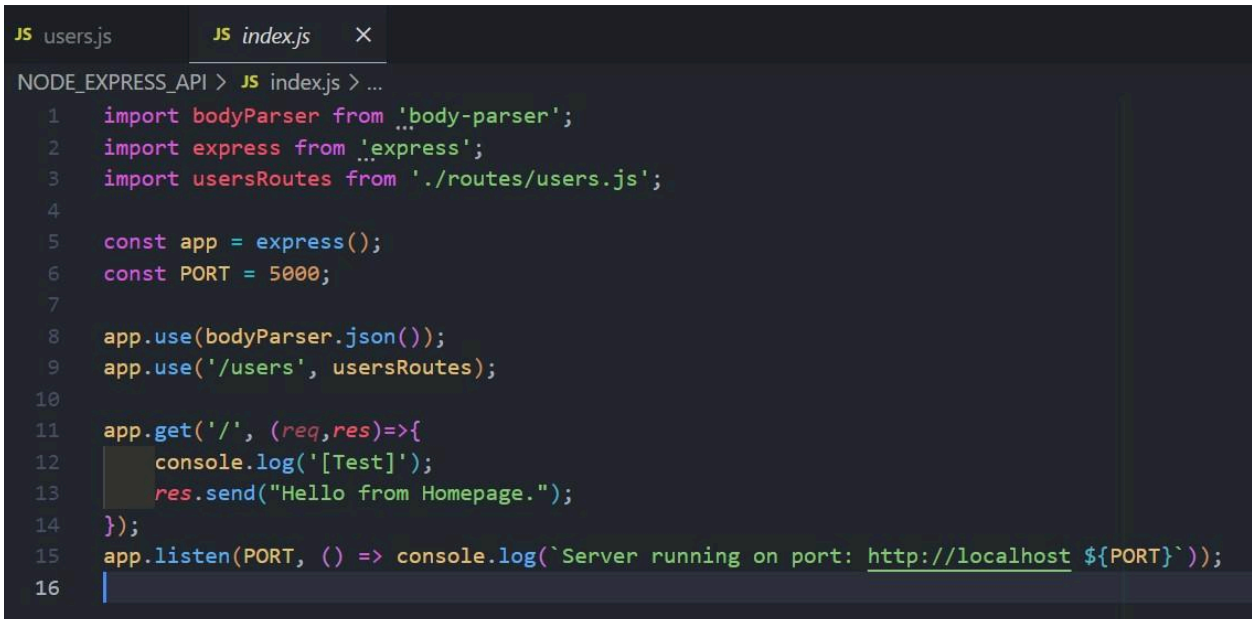
NODE_EXPRESS_API > routes > users.js > ...

```
router.post('/', (req, res)=>{
});

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`);
```



NODE_EXPRESS_API > routes > JS users.js > ...

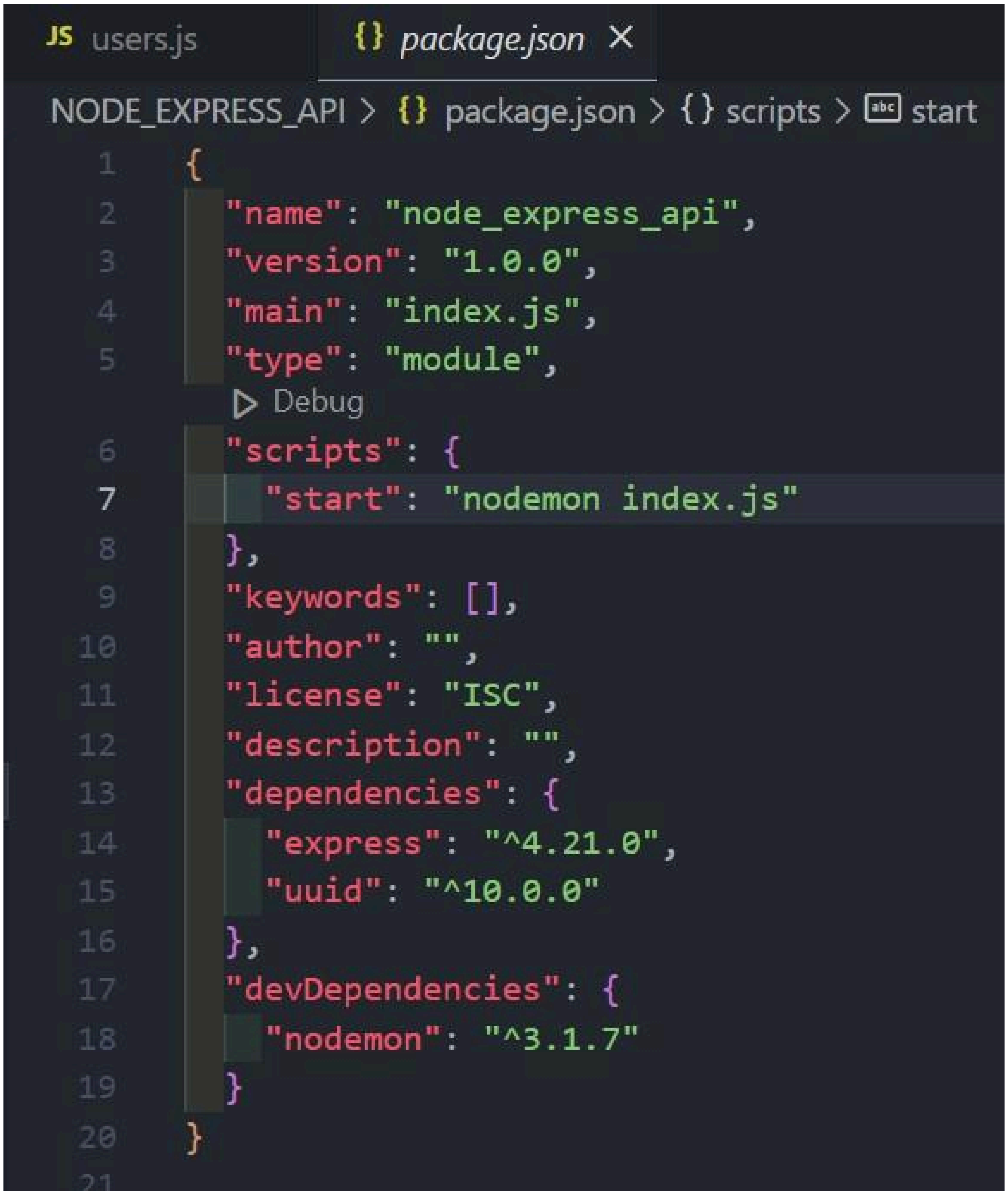
```
router.patch('/:id', (req, res)=>{
    const {firstName, lastName, age} = req.body;
    const user = users.find((user)=>user.id==id);
    if(firstName){
    user.firstName= firstName;
    }
    if(lastName){
    user.lastName= lastName;
    }
    if(age){
    user.age= age;
    }
    res.send(`User with the id ${id} has been updated`);
});

    export default router;
```



JS users.js   JS index.js ×

NODE_EXPRESS_API > JS index.js > ...

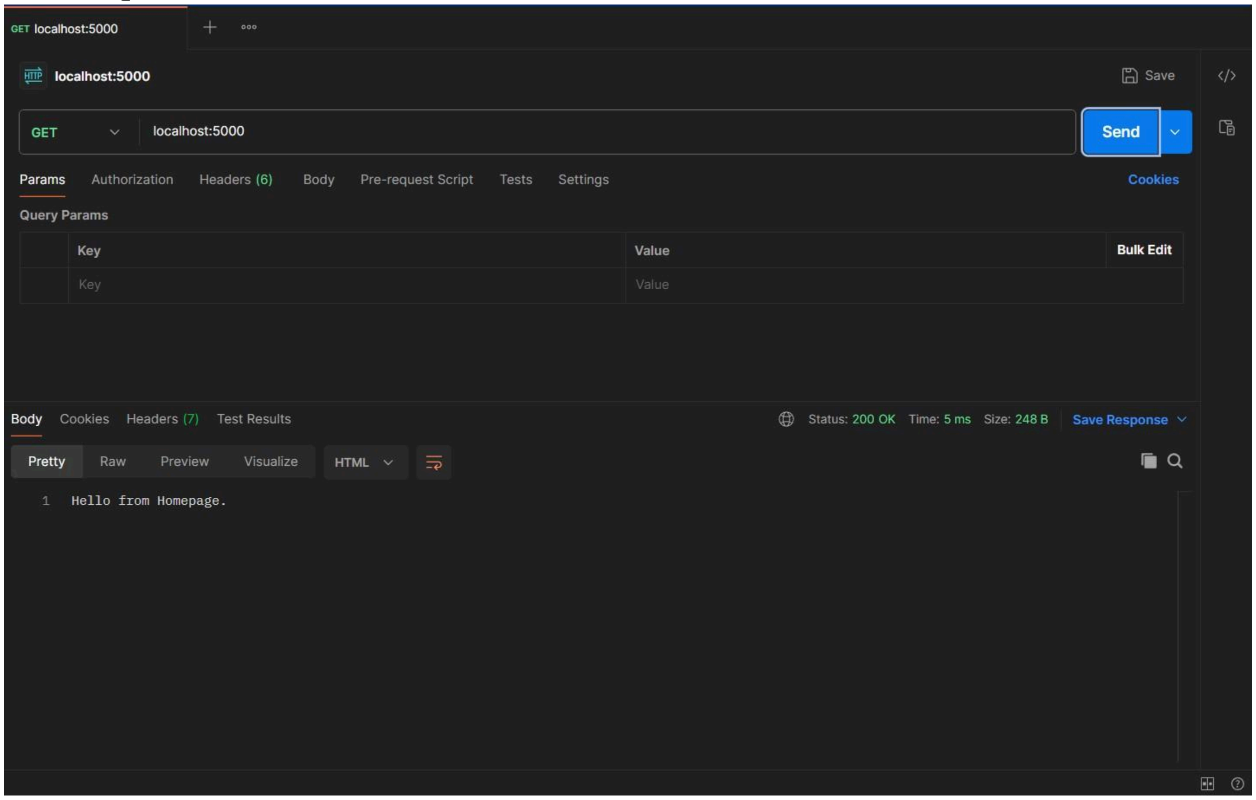
```
import bodyParser from 'body-parser';
import express from 'express';
import usersRoutes from './routes/users.js';

const app = express();
const PORT = 5000;

app.use(bodyParser.json());
app.use('/users', usersRoutes);

app.get('/', (req,res)=>{
    console.log('[Test]');
    res.send("Hello from Homepage.");
});
app.listen(PORT, () => console.log(`Server running on port: http://localhost ${PORT}`));

```



```json
{
  "name": "node_express_api",
  "version": "1.0.0",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.21.0",
    "uuid": "^10.0.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.7"
  }
}
```

Output: Screenshots of the output to be attached.

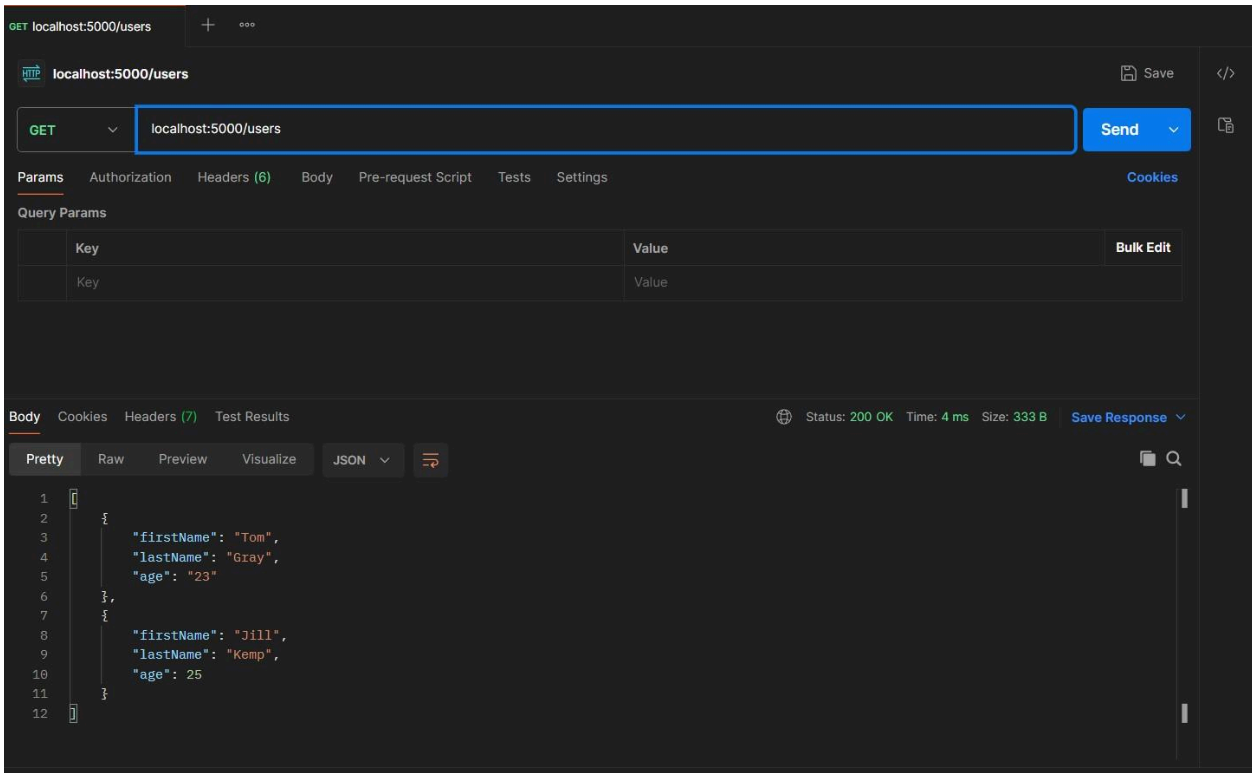
GET Request



GET localhost:5000

Status: 200 OK Time: 5 ms Size: 248 B

```
Hello from Homepage.
```

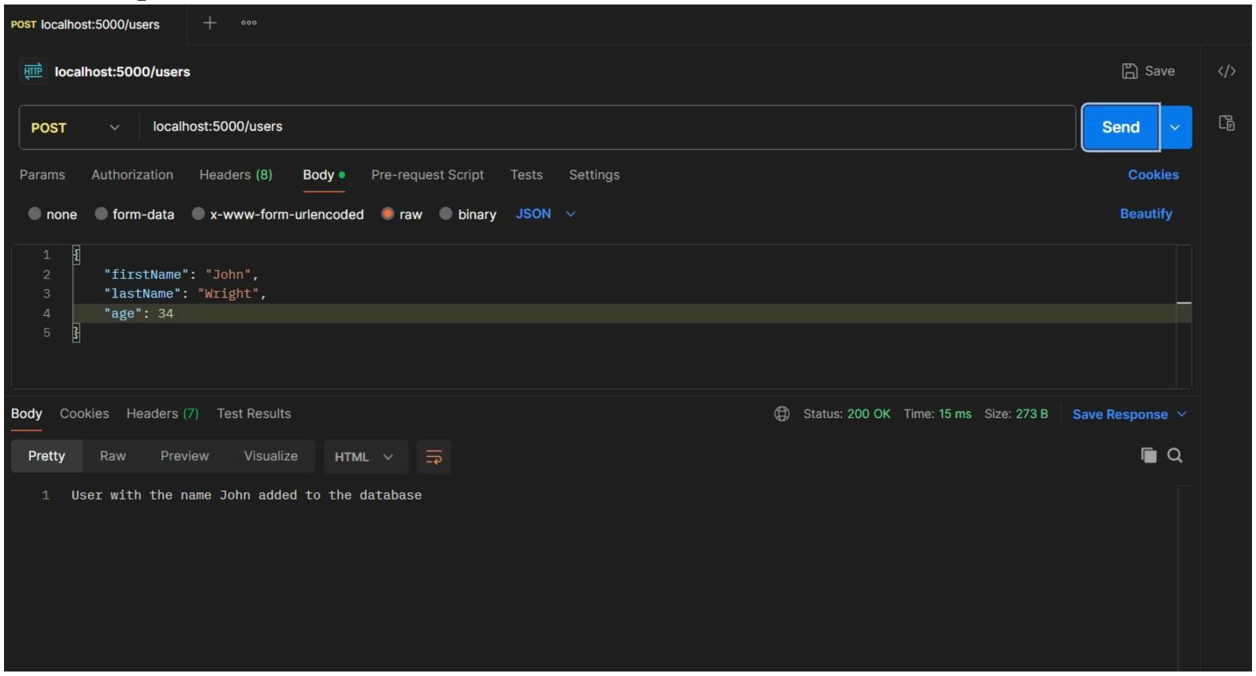


GET localhost:5000/users

Status: 200 OK Time: 4 ms Size: 333 B

```
[
    {
        "firstName": "Tom",
        "lastName": "Gray",
        "age": "23"
    },
    {
        "firstName": "Jill",
        "lastName": "Kemp",
        "age": 25
    }
]
```

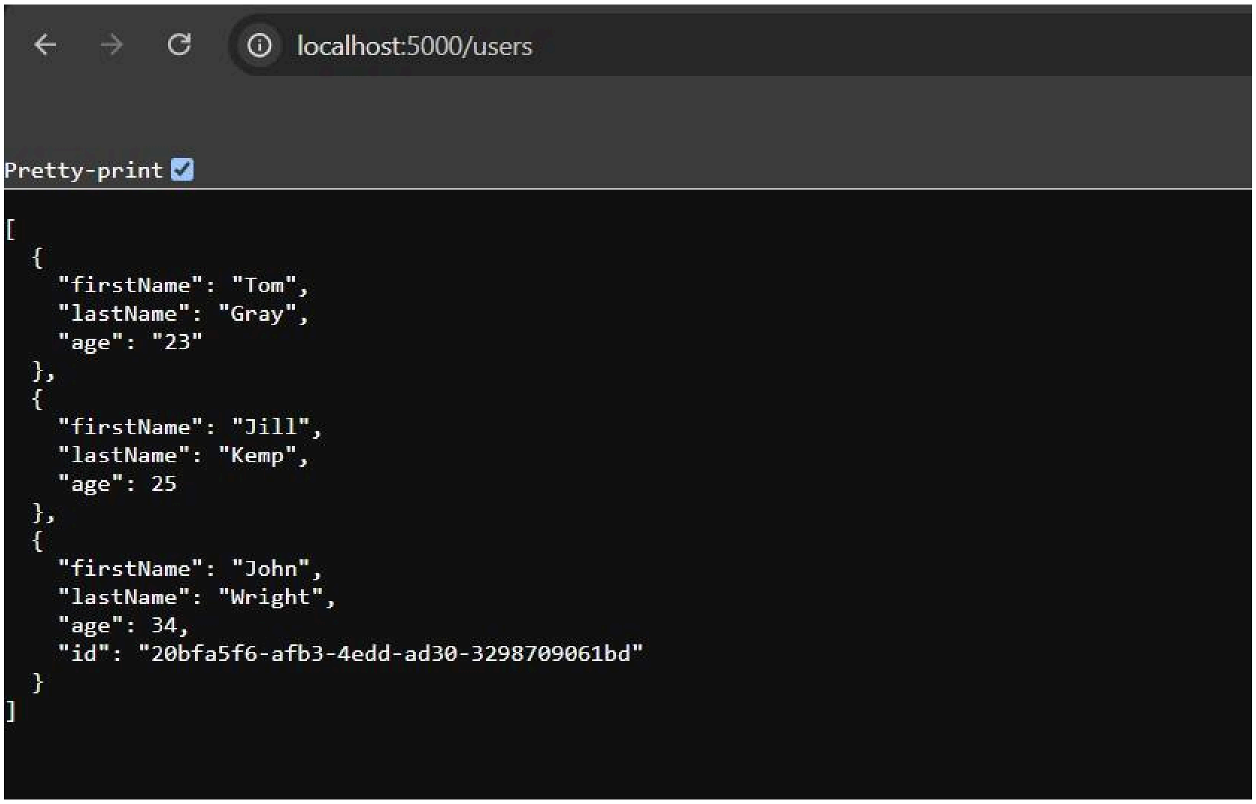
POST Request:


POST localhost:5000/users

```
{
    "firstName": "John",
    "lastName": "Wright",
    "age": 34
}
```

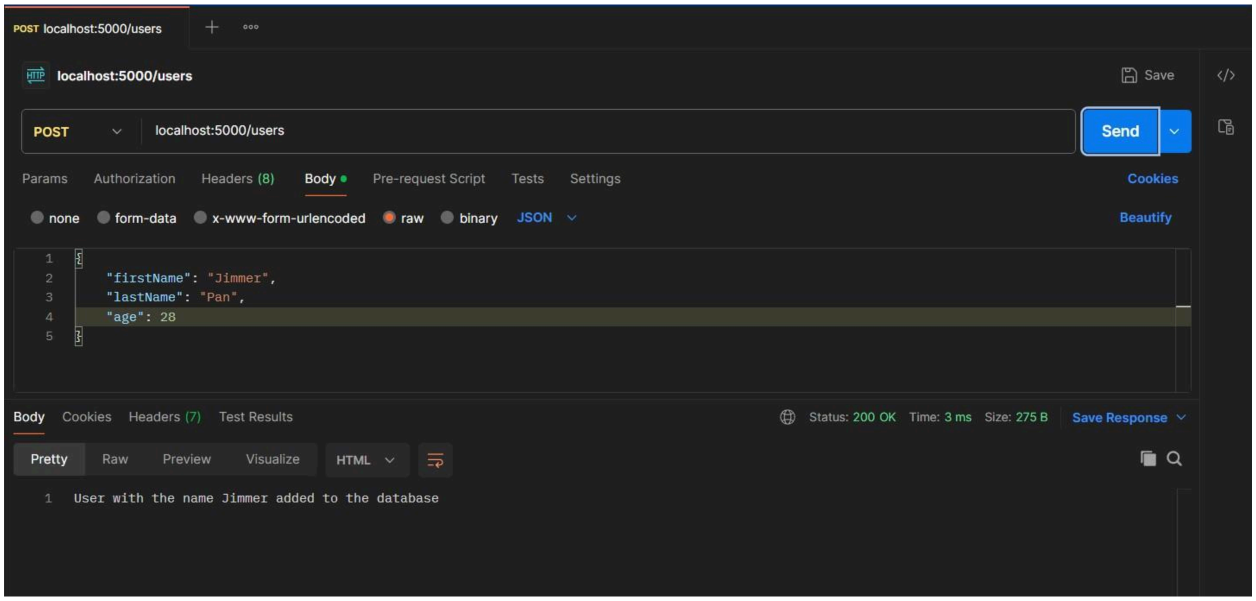
Status: 200 OK Time: 15 ms Size: 273 B

```
User with the name John added to the database
```


localhost:5000/users

Pretty-print ✓

```
[
  {
    "firstName": "Tom",
    "lastName": "Gray",
    "age": "23"
  },
  {
    "firstName": "Jill",
    "lastName": "Kemp",
    "age": 25
  },
  {
    "firstName": "John",
    "lastName": "Wright",
    "age": 34,
    "id": "20bfa5f6-afb3-4edd-ad30-3298709061bd"
  }
]
```



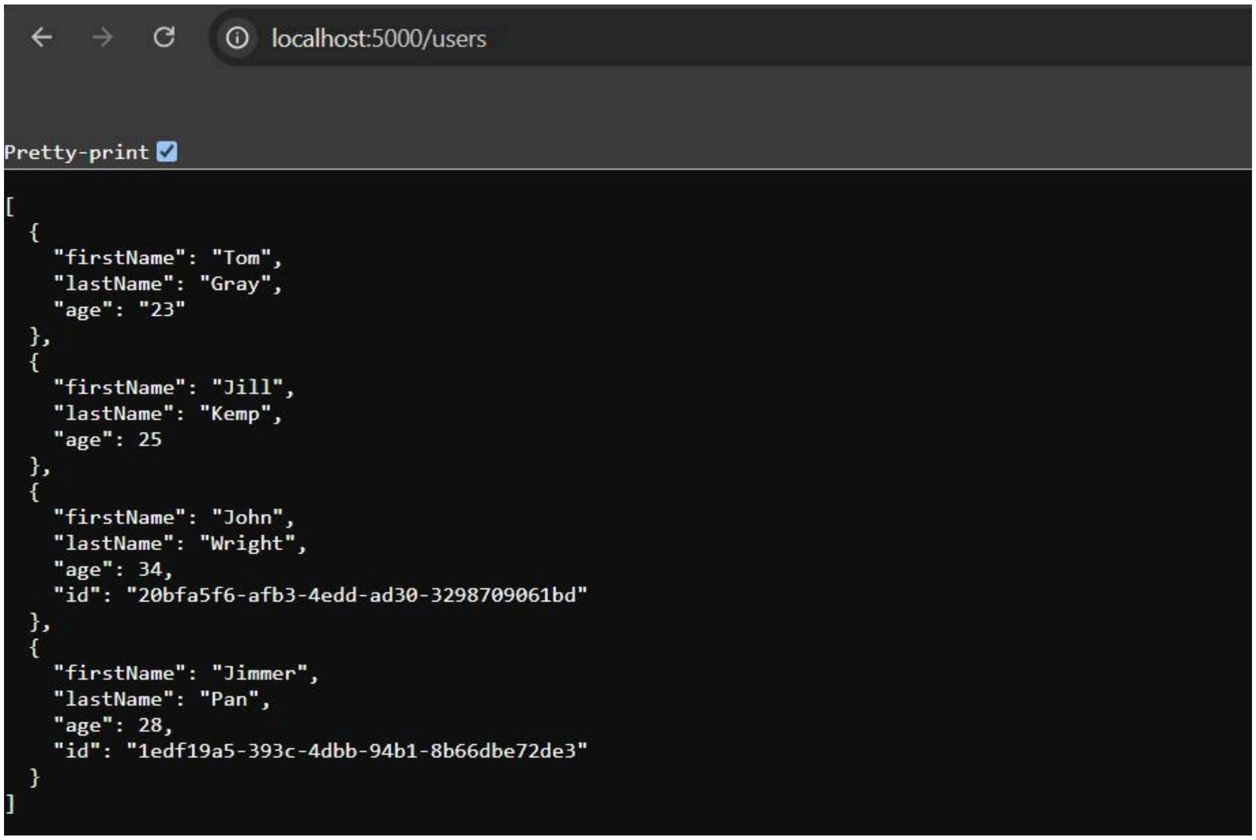
POST localhost:5000/users

Body (raw, JSON):

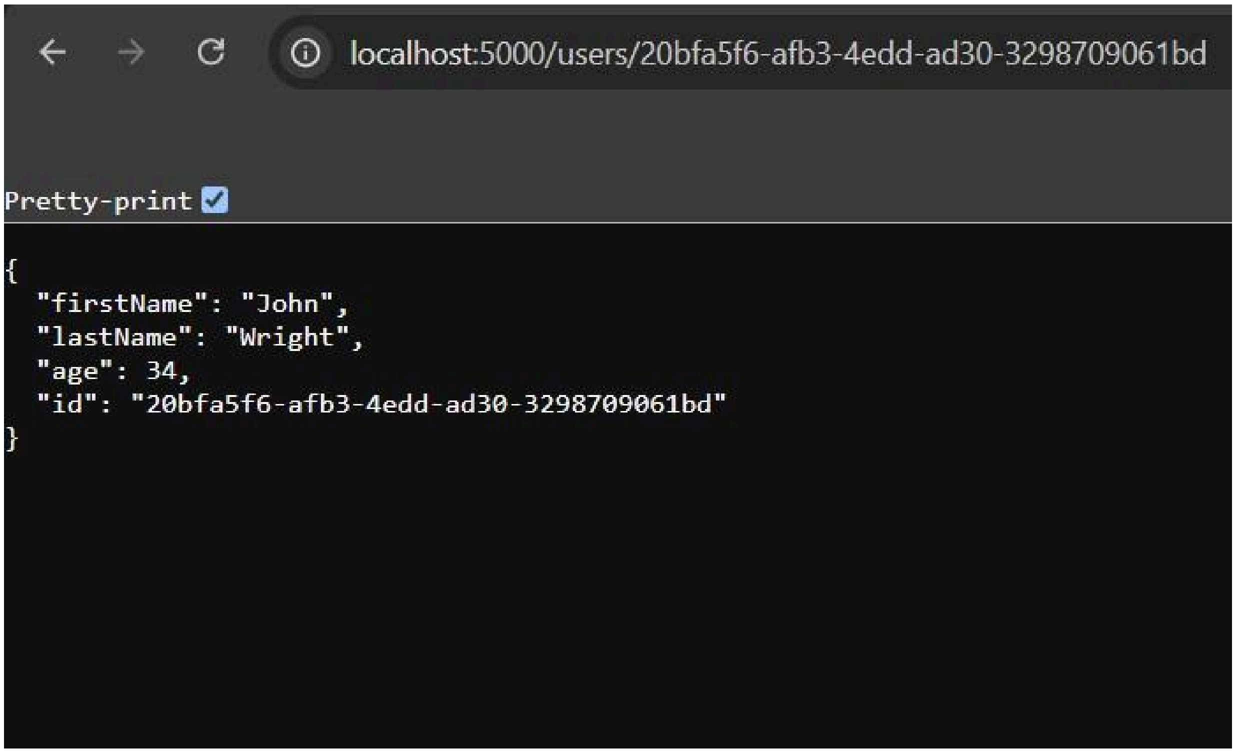
```
{
    "firstName": "Jimmer",
    "lastName": "Pan",
    "age": 28
}
```

Status: 200 OK Time: 3 ms Size: 275 B

```
User with the name Jimmer added to the database
```



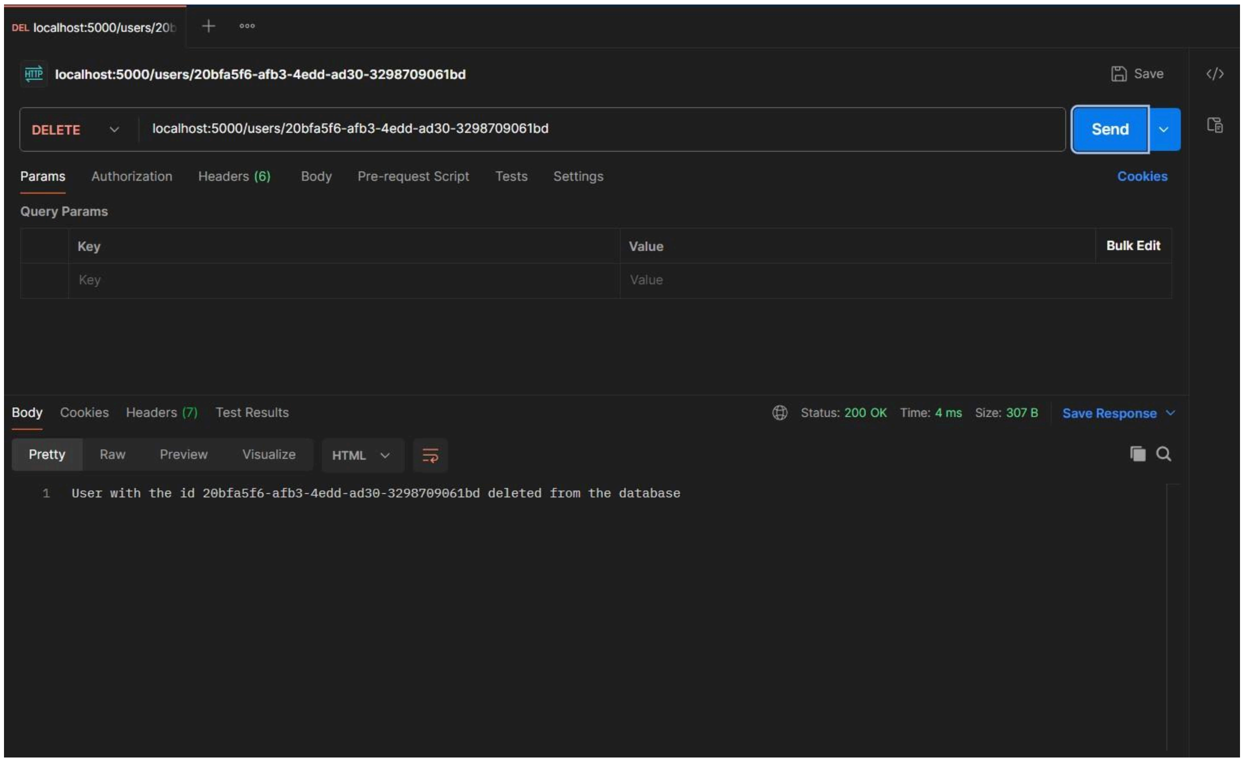
```
[
  {
    "firstName": "Tom",
    "lastName": "Gray",
    "age": "23"
  },
  {
    "firstName": "Jill",
    "lastName": "Kemp",
    "age": 25
  },
  {
    "firstName": "John",
    "lastName": "Wright",
    "age": 34,
    "id": "20bfa5f6-afb3-4edd-ad30-3298709061bd"
  },
  {
    "firstName": "Jimmer",
    "lastName": "Pan",
    "age": 28,
    "id": "1edf19a5-393c-4dbb-94b1-8b66dbe72de3"
  }
]
```


localhost:5000/users/20bfa5f6-afb3-4edd-ad30-3298709061bd

Pretty-print ✓

```
{
  "firstName": "John",
  "lastName": "Wright",
  "age": 34,
  "id": "20bfa5f6-afb3-4edd-ad30-3298709061bd"
}
```

DELETE Request:

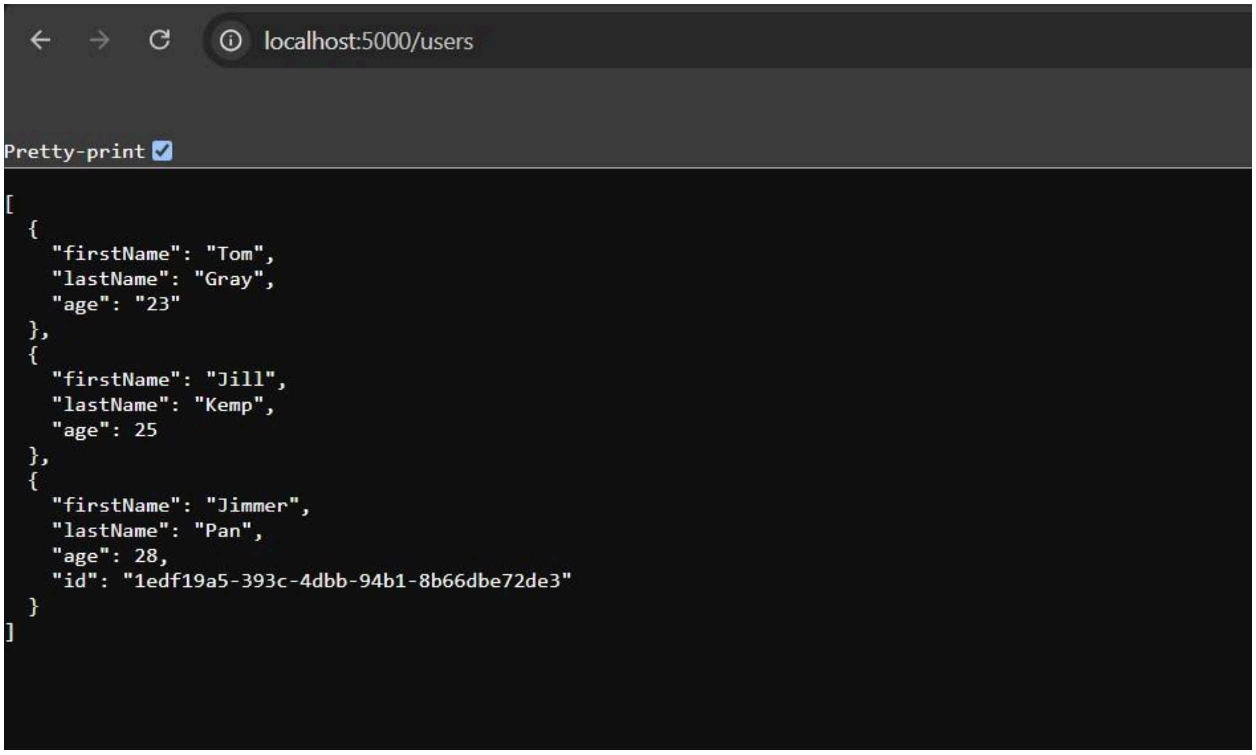


localhost:5000/users/20bfa5f6-afb3-4edd-ad30-3298709061bd

DELETE localhost:5000/users/20bfa5f6-afb3-4edd-ad30-3298709061bd

Status: 200 OK Time: 4 ms Size: 307 B

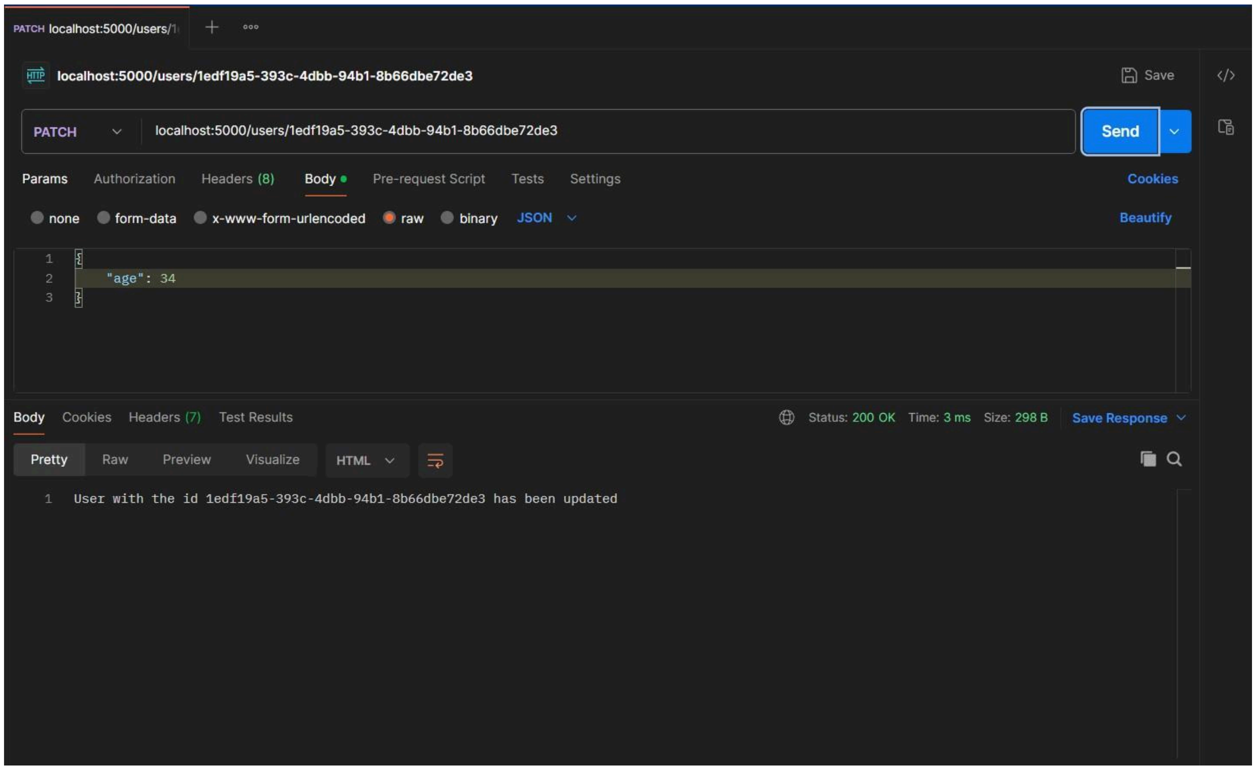
```
User with the id 20bfa5f6-afb3-4edd-ad30-3298709061bd deleted from the database
```


localhost:5000/users

Pretty-print ✓

```
[
  {
    "firstName": "Tom",
    "lastName": "Gray",
    "age": "23"
  },
  {
    "firstName": "Jill",
    "lastName": "Kemp",
    "age": 25
  },
  {
    "firstName": "Jimmer",
    "lastName": "Pan",
    "age": 28,
    "id": "1edf19a5-393c-4dbb-94b1-8b66dbe72de3"
  }
]
```

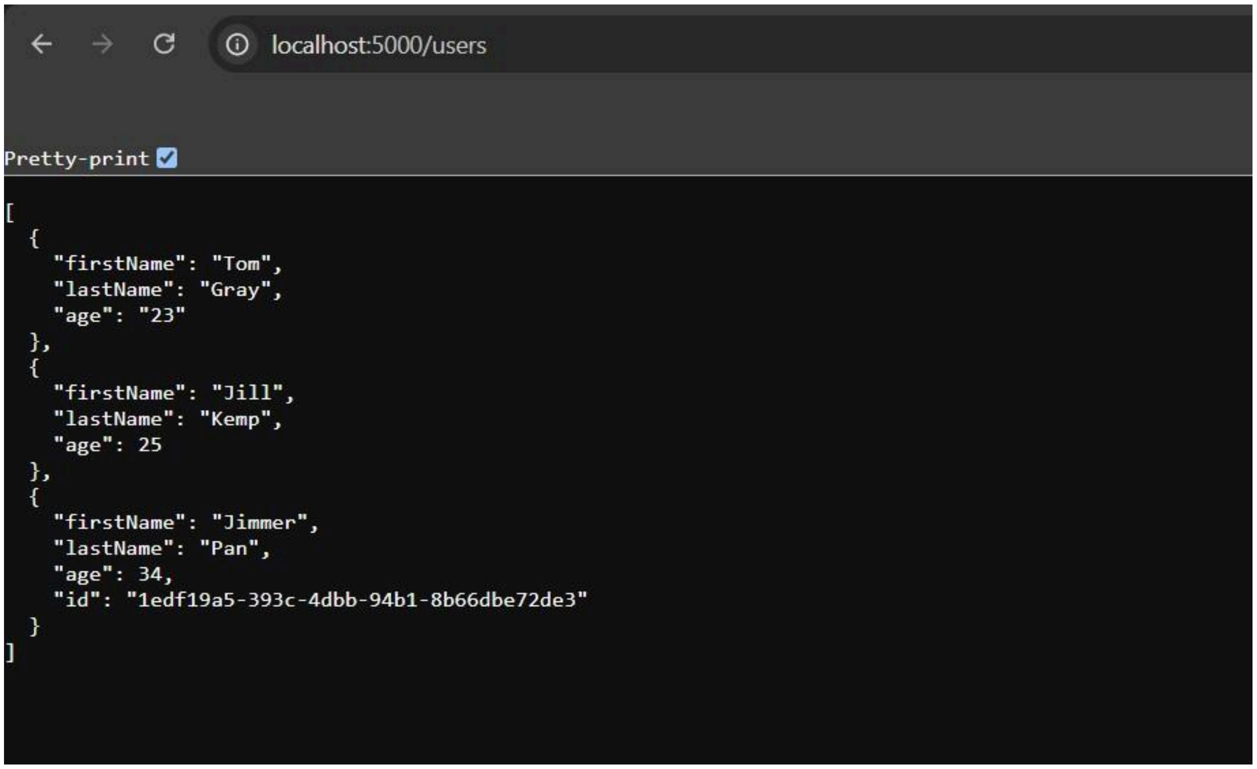
PATCH Request:


PATCH localhost:5000/users/1edf19a5-393c-4dbb-94b1-8b66dbe72de3

```
{
    "age": 34
}
```

Status: 200 OK Time: 3 ms Size: 298 B

```
User with the id 1edf19a5-393c-4dbb-94b1-8b66dbe72de3 has been updated
```



localhost:5000/users

Pretty-print ✓

```
[
  {
    "firstName": "Tom",
    "lastName": "Gray",
    "age": "23"
  },
  {
    "firstName": "Jill",
    "lastName": "Kemp",
    "age": 25
  },
  {
    "firstName": "Jimmer",
    "lastName": "Pan",
    "age": 34,
    "id": "1edf19a5-393c-4dbb-94b1-8b66dbe72de3"
  }
]
```

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/