Hands-on Lab - CRUD operations with Node.js

Skills Network

Estimated Time Needed: 1 hour

In this lab you will learn how to create a **Friend's list** using Express server. Your application should allow you to add a friend with the following details: `First name`, `Last name`, `Email` and `Date of birth`. You will also be providing the application the ability to retrieve details, change details and delete the details.

You will be creating an application with API endpoints to perform Create, Retrieve, Update and Delete operations on the above data using an Express server.

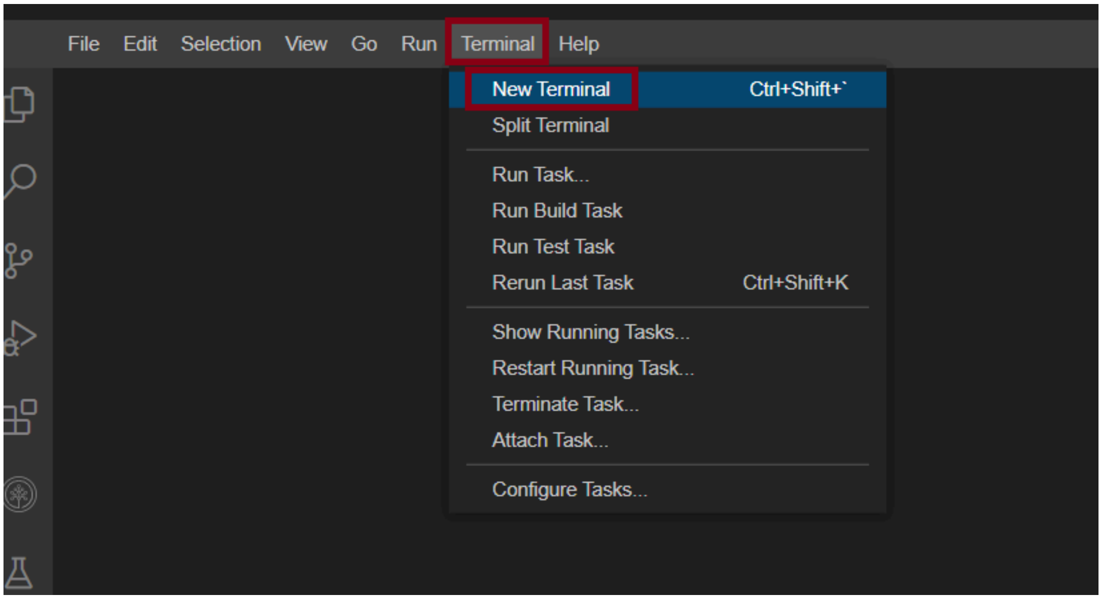
You will also learn to provide authenticated access to the endpoints. You will use cURL and Postman to test the implemented endpoints.

Objectives:

- Create API endpoints to perform Create, Retrieve, Update and Delete operations on transient data with an Express server.
- Implement authentication at the session level using JSON Web Tokens (JWT) for authorized access.

Set-up : Create application

1. Open a terminal window by using the menu in the editor: Terminal > New Terminal.

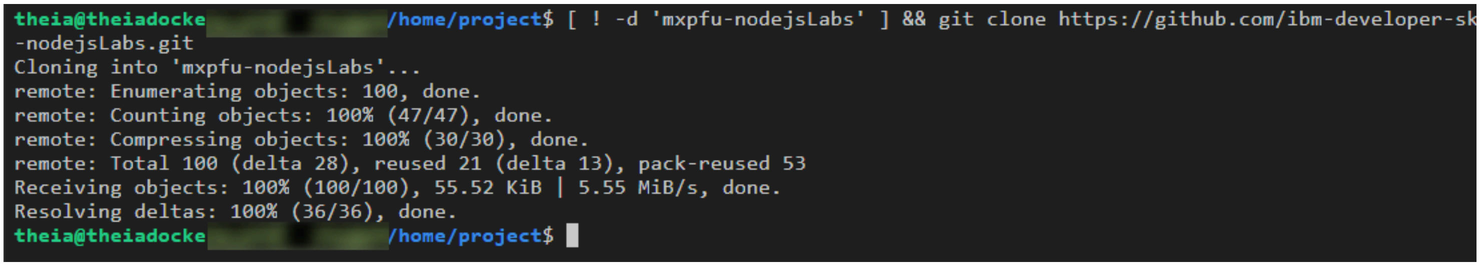


2. Change to your project folder, if you are not in the project folder already.

```
cd /home/project
```

3. Run the following command to clone the git repository that contains the starter code needed for this lab, if it doesn't already exist.

```
[ ! -d 'mxpfu-nodejsLabs' ] && git clone https://github.com/ibm-developer-skills-network/mxpfu-nodejsLabs.git
```



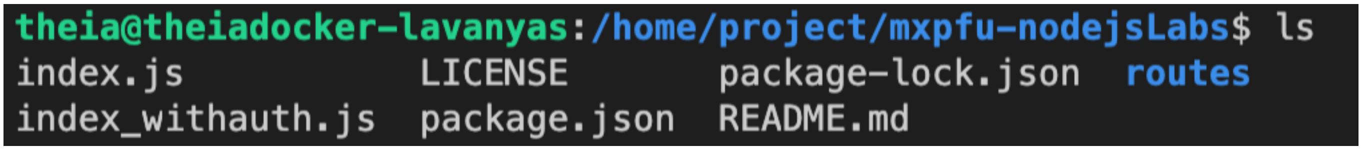
```
theia@theiadocke[redacted]/home/project$ [ ! -d 'mxpfu-nodejsLabs' ] && git clone https://github.com/ibm-developer-sk
-nodejsLabs.git
Cloning into 'mxpfu-nodejsLabs'...
remote: Enumerating objects: 100, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 100 (delta 28), reused 21 (delta 13), pack-reused 53
Receiving objects: 100% (100/100), 55.52 KiB | 5.55 MiB/s, done.
Resolving deltas: 100% (36/36), done.
theia@theiadocke[redacted]/home/project$ 
```

5. Change to the directory **mxpfu-nodejsLabs** directory to start working on the lab.

```
cd mxpfu-nodejsLabs/
```

6. List the contents of this directory to see the artifacts for this lab.

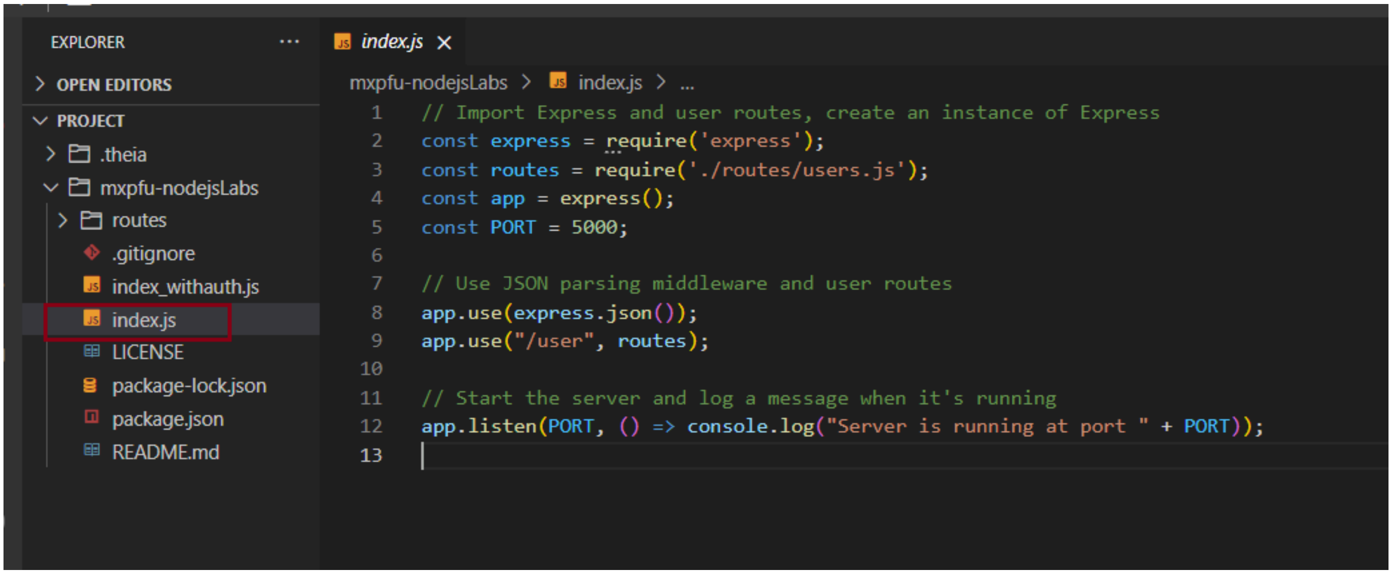
```
ls
```



```
theia@theiadocker-lavanyas:/home/project/mxpfu-nodejsLabs$ ls
index.js           LICENSE        package-lock.json  routes
index_withauth.js  package.json   README.md
```

Exercise 1: Understand the server application

1. In the Files Explorer open the **mxpfu-nodejsLabs** folder and view **index.js**.



```
// Import Express and user routes, create an instance of Express
const express = require('express');
const routes = require('./routes/users.js');
const app = express();
const PORT = 5000;

// Use JSON parsing middleware and user routes
app.use(express.json());
app.use("/user", routes);

// Start the server and log a message when it's running
app.listen(PORT, () => console.log("Server is running at port " + PORT));
```

You have an Express server that has been configured to run at port 5000. When you access the server with **/user** you can access the endpoints defined in **routes/users.js**.

Recall that GET, POST, PUT and DELETE are the commonly used HTTP methods to perform CRUD operations. Those operations retrieve and send data to the server.

- `GET` is used to request data from a specified resource.

- `POST` is used to send data to a server for creating a resource.

- `PUT` is used to send data to a server to update a resource.

- `DELETE` is used for deleting a specified resource.

  `POST` AND `PUT` are sometimes used interchangeably.

2. This lab requires some packages to be installed. The **express** and **nodemon** package for starting and running the Express server and **jsonwebtoken** and **express-session** for session based authentication.

- **express** - This is for creating a server to serve the API endpoints.
- **nodemon** - This will help to restart the server when you make any changes to the code.
- **jsonwebtoken** - This package helps in generating a JSON web token which we will use for authentication. A **JSON web token (JWT)** is a JSON object used to communicate information securely over the internet (between two parties). It can be used for information exchange and is typically used for authentication systems.
- **express-session** - This package will help us to maintain the authentication for the session.

These packages are defined in as **dependencies** in packages.json.

```
  "dependencies": {
    "express": "^4.18.1",
    "express-session": "^1.17.3",
    "jsonwebtoken": "^8.5.1",
    "nodemon": "^2.0.19"
}
```

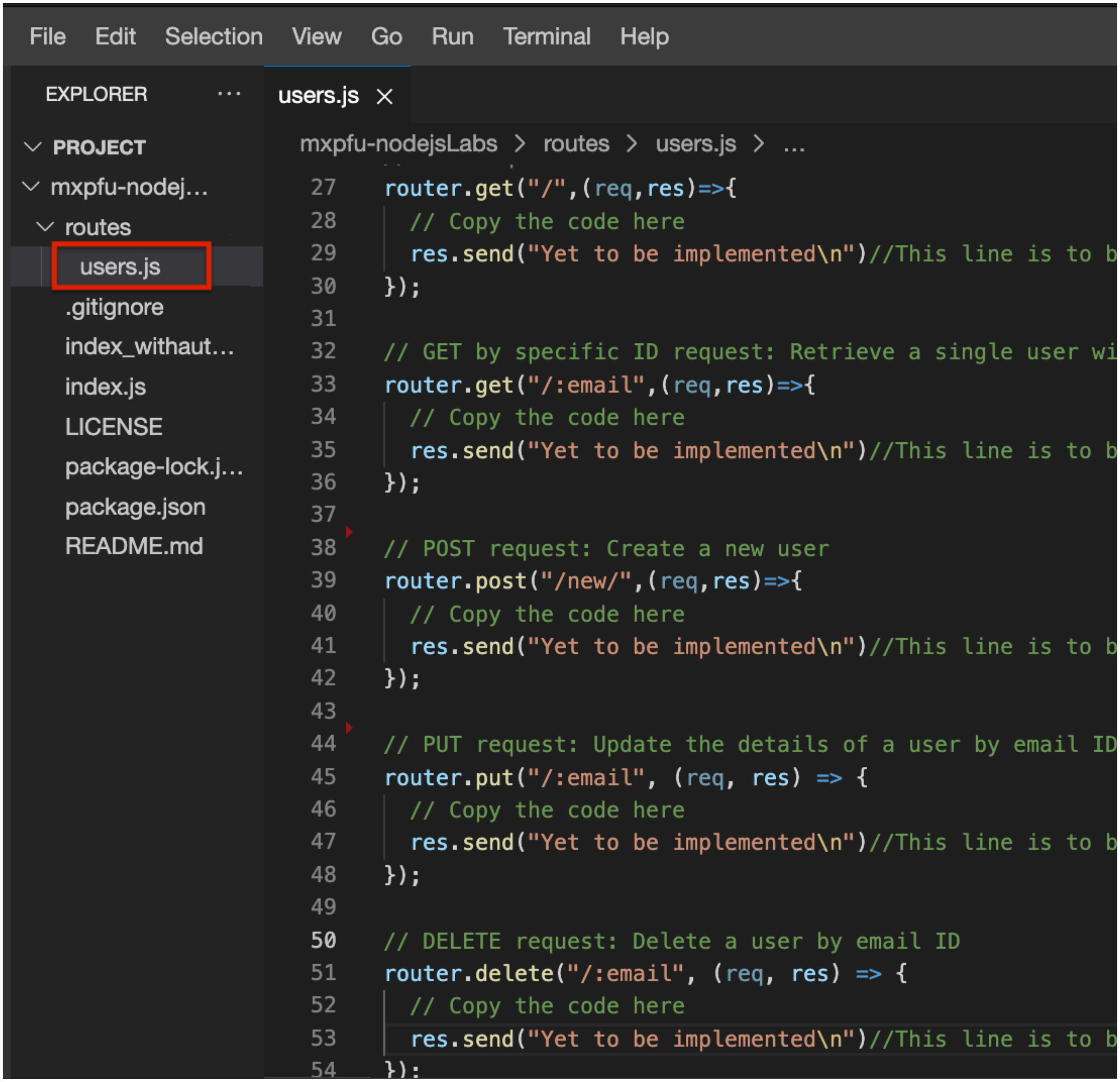
3. Observe that the express app uses the middleware **express.json()** to handle the request as a json object.

```
app.use(express.json());
```

4. Observe that the express app uses routes to handle the endpoints which start with **/user**. This means that for all the endpoints starting with **/user**, the server will go and look for an endpoint handler in **users.js**.

```
app.use("/user", routes);
```

5. All the endpoints have skeletal, but working implementation in **users.js**. Navigate to users.js under the directory routes and observe the endpoints defined in it.



```
router.get("/",(req,res)=>{
  // Copy the code here
  res.send("Yet to be implemented\n")//This line is to b
});

// GET by specific ID request: Retrieve a single user wi
router.get("/:email",(req,res)=>{
  // Copy the code here
  res.send("Yet to be implemented\n")//This line is to b
});

// POST request: Create a new user
router.post("/new/",(req,res)=>{
  // Copy the code here
  res.send("Yet to be implemented\n")//This line is to b
});

// PUT request: Update the details of a user by email ID
router.put("/:email", (req, res) => {
  // Copy the code here
  res.send("Yet to be implemented\n")//This line is to b
});

// DELETE request: Delete a user by email ID
router.delete("/:email", (req, res) => {
  // Copy the code here
  res.send("Yet to be implemented\n")//This line is to b
});
```

Exercise 2: Run the server

The starter code given is a functioning server with dummy return values. Before starting to implement the actual endpoints, run the server.

1. In the terminal, print the working directory to ensure you are in **/home/projects/mxpfu-nodejsLabs**.

   ```
   pwd
   ```

2. Install all the packages that are required for running the server. Copy, paste, and run the following command.

   ```
   npm install
   ```

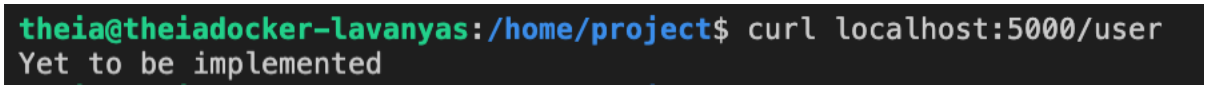
This will install all the required packages as defined in packages.json.

3. Start the express server.

```
npm start
```

4. Open a **New Terminal** from the top menu. Test an endpoint to retrieve these users. This has not yet been implemented to return the users.

```
curl localhost:5000/user
```

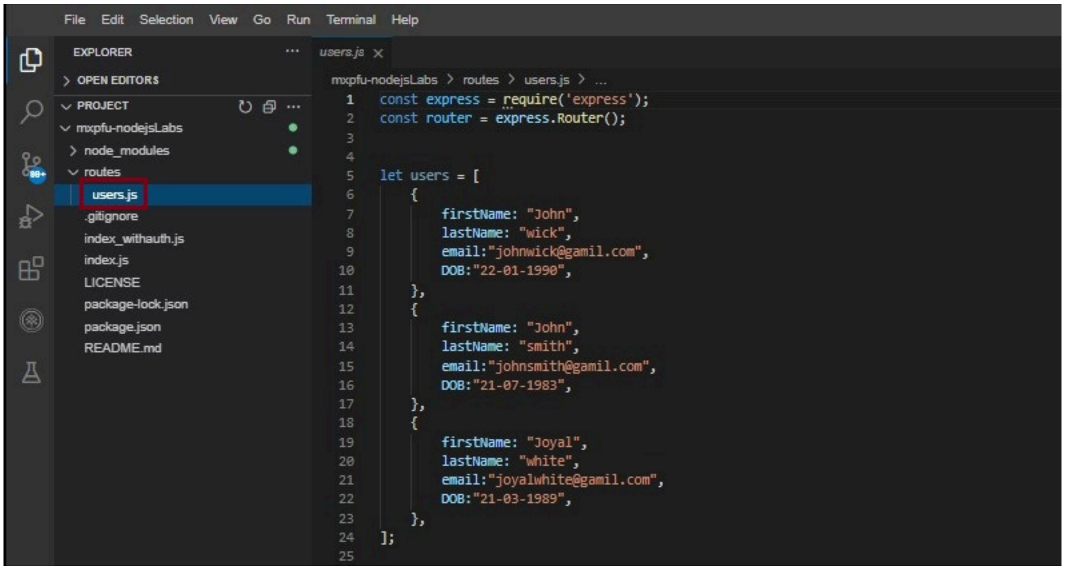


```
theia@theiadocker-lavanyas:/home/project$ curl localhost:5000/user
Yet to be implemented
```

5. If you see the output as displayed above, it means the server is running as expected.

Exercise 2: Implement your endpoints

1. Navigate to the file named **users.js** in the **routes** folder. The endpoints have been defined and space has been provided for you to implement the endpoints.

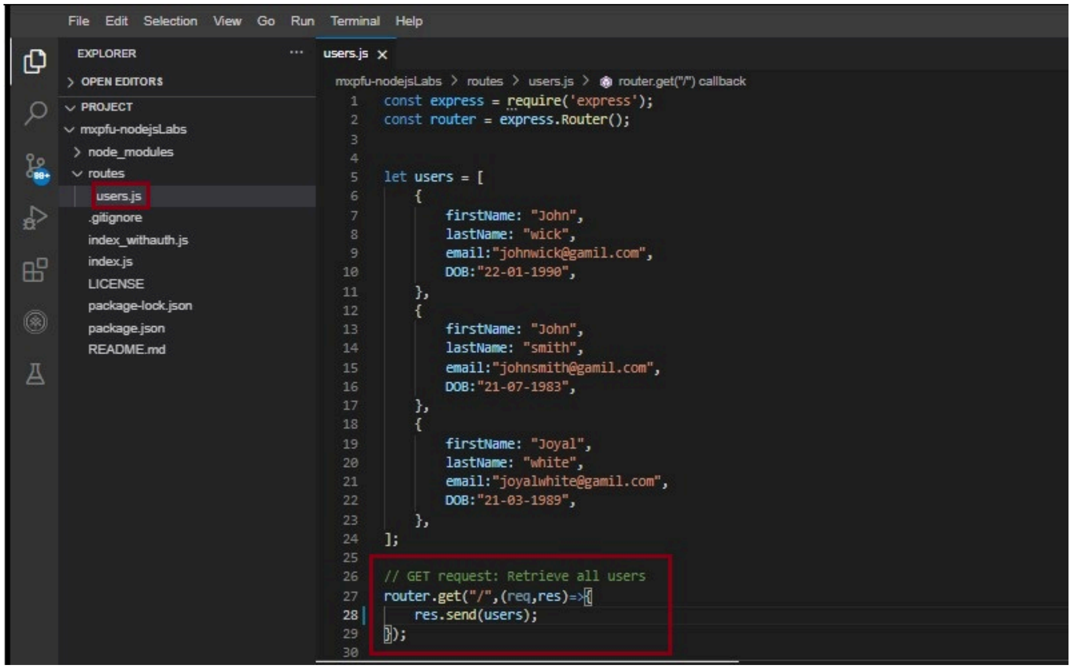


```
const express = require('express');
const router = express.Router();


let users = [
    {
        firstName: "John",
        lastName: "wick",
        email:"johnwick@gamil.com",
        DOB:"22-01-1990",
    },
    {
        firstName: "John",
        lastName: "smith",
        email:"johnsmith@gamil.com",
        DOB:"21-07-1983",
    },
    {
        firstName: "Joyal",
        lastName: "white",
        email:"joyalwhite@gamil.com",
        DOB:"21-03-1989",
    },
];

```

2. **R** in CRUD stands for retrieve. You will first add an API endpoint, using the **get** method for getting the details of all users. A few users have been added in the starter code.

- Copy the code below and paste in users.js inside the { } brackets within the **router.get("/",(req,res)=>{}** method.

```
res.send(users);
```

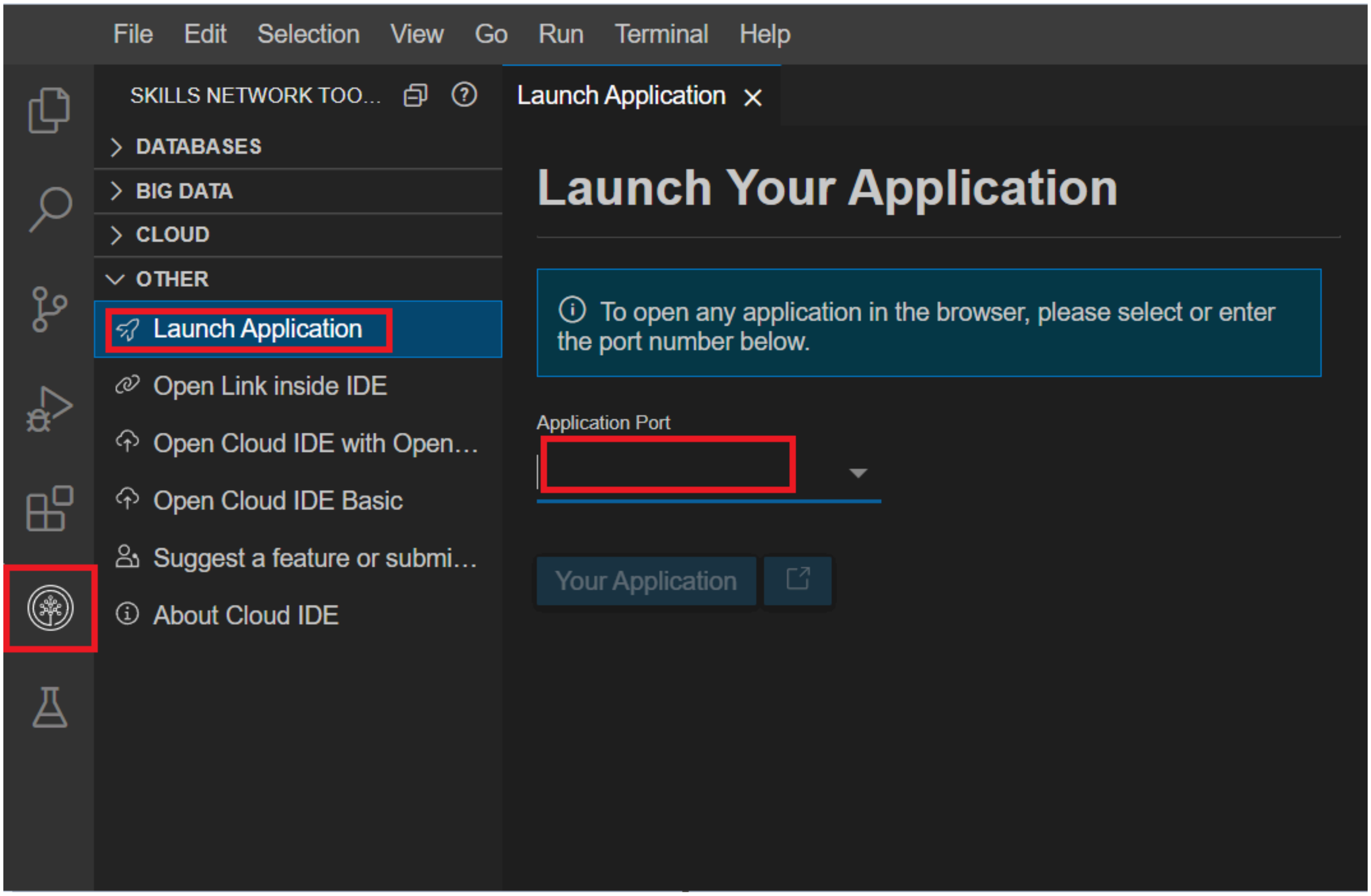


```javascript
const express = require('express');
const router = express.Router();


let users = [
    {
        firstName: "John",
        lastName: "wick",
        email:"johnwick@gamil.com",
        DOB:"22-01-1990",
    },
    {
        firstName: "John",
        lastName: "smith",
        email:"johnsmith@gamil.com",
        DOB:"21-07-1983",
    },
    {
        firstName: "Joyal",
        lastName: "white",
        email:"joyalwhite@gamil.com",
        DOB:"21-03-1989",
    },
];

// GET request: Retrieve all users
router.get("/",(req,res)=>{
    res.send(users);
});
```

3. Ensure that your server is running. As you make changes to the code, the server that you started in the previous task, should be restart. If the server is not running, start it again.

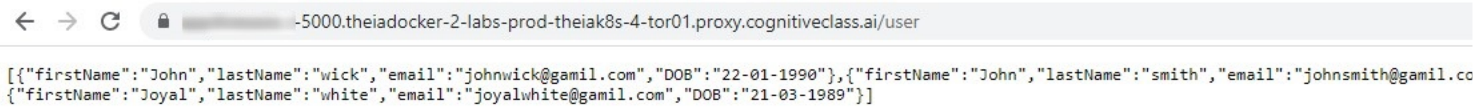
   ```
   npm start
   ```

3. Click on the **Skills Network** button on the left. It will open the "Skills Network Toolbox". Then click `OTHER` then `Launch Application`. From there you should be able to enter the port as `5000` and launch the development server.



To open any application in the browser, please select or enter the port number below.

4. When the browser page opens up, suffix **/user** to the end of the URL on the address bar. You will see the below page.



[redacted]-5000.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user

```
[{"firstName":"John","lastName":"wick","email":"johnwick@gamil.com","DOB":"22-01-1990"},{"firstName":"John","lastName":"smith","email":"johnsmith@gamil.co
{"firstName":"Joyal","lastName":"white","email":"joyalwhite@gamil.com","DOB":"21-03-1989"}]
```

5. Check the output of the GET request using the curl command just the way you did in the previous exercise.

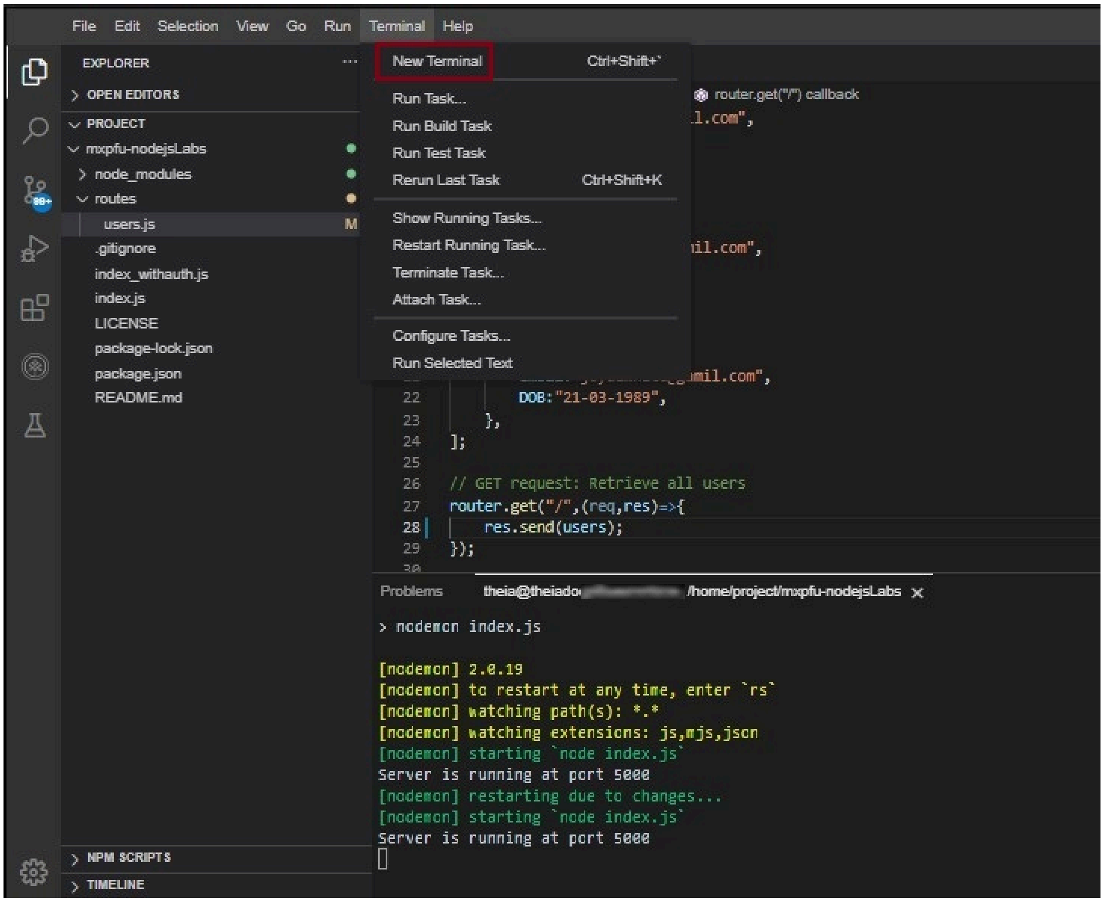
```
curl localhost:5000/user/
```

Exercise 3: Creating a GET by specific email method:

1. Implement a **get** method for getting the details of a specific user based on their email ID by using the `filter` method on the user collection. Once you write the code and save it, the server will restart.

▶ Click here to view the code

2. Click on Terminal > New Terminal

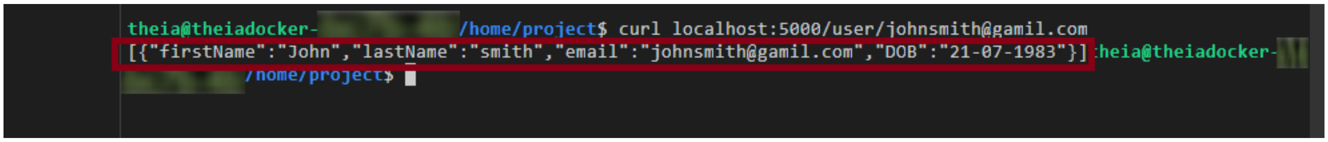


```
l.com",
il.com",
mil.com",
        DOB:"21-03-1989",
    },
];

// GET request: Retrieve all users
router.get("/",(req,res)=>{
    res.send(users);
});
```

```
> nodemon index.js

[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server is running at port 5000
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Server is running at port 5000
```

3. In the new terminal, use the below command to view the output for the user with mail id johnsmith@gamil.com

```
curl localhost:5000/user/johnsmith@gamil.com
```



```
theia@theiadocker-[redacted]/home/project$ curl localhost:5000/user/johnsmith@gamil.com
[{"firstName":"John","lastName":"smith","email":"johnsmith@gamil.com","DOB":"21-07-1983"}]theia@theiadocker-[redacted]/home/project$
```

Exercise 4: Creating the POST method:

1. The **C** in CRUD stands for **Create**. Implement the **/user** endpoint with the POST method to create a user and add the user to the list. You can create the user object as a dictionary. You can use the sample user object displayed below.

```
{
    "firstName":"Jon",
    "lastName":"Lovato",
    "email":"jonlovato@theworld.com",
    "DOB":"10/10/1995"
}
```

Use `push` to add the dictionary into the list of users. The user details can be passed as query paramters named *firstName*, *lastName*, *DOB and email*.

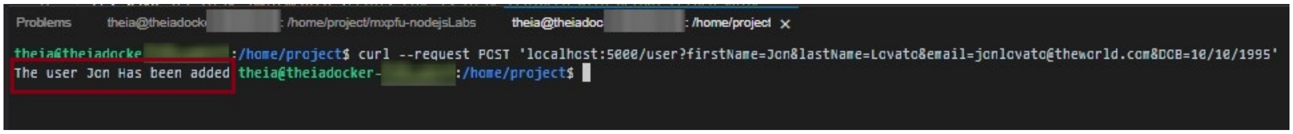
Hint: Query param can be retrieved from the request object using request.query.paramname

▶ Click here to view the code

2. Use the below command to post a new user with mail id 'jonlovato@theworld.com' on the new terminal:

```
curl --request POST 'localhost:5000/user?firstName=Jon&lastName=Lovato&email=jonlovato@theworld.com&DOB=10/10/1995'
```

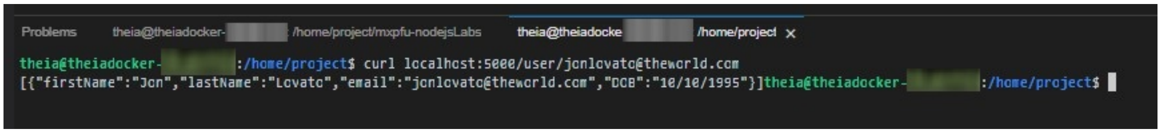
3. The ouput will be as below:


Problems
theia@theiadock[redacted] : /home/project/mxpfu-nodejsLabs
theia@theiadoc[redacted] : /home/project ×

```
theia@theiadocke[redacted]:/home/project$ curl --request POST 'localhost:5000/user?firstName=Jon&lastName=Lovato&email=jonlovato@theworld.com&DOB=10/10/1995'
The user Jon Has been added theia@theiadocker-[redacted]:/home/project$ 
```

4. To verify if the user with email 'jonlovato@theworld.com' has been added, you can send a GET request as below:

```
curl localhost:5000/user/jonlovato@theworld.com
```


Problems
theia@theiadocker-[redacted] /home/project/mxpfu-nodejsLabs
theia@theiadocke[redacted] /home/project ×

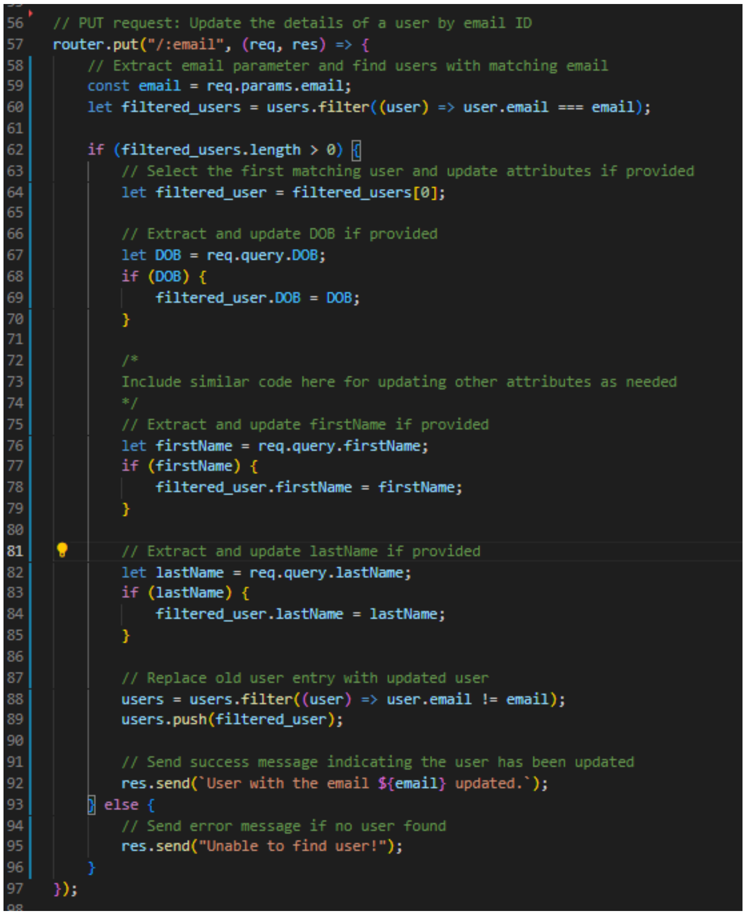
```
theia@theiadocker-[redacted]:/home/project$ curl localhost:5000/user/jonlovato@theworld.com
[{"firstName":"Jon","lastName":"Lovato","email":"jonlovato@theworld.com","DOB":"10/10/1995"}]theia@theiadocker-[redacted]:/home/project$ 
```

Exercise 5: Creating the PUT method:

1. The **U** in CRUD stands for update which can be achieved using the PUT method. To make updates in the data, you will use the PUT method. You should first look at the user with the specified email id and then modify it. The code below shows how the date of birth (DOB) of a user can be modified. Make the necessary code changes to allow changes to the other attributes of the user.

```
router.put("/:email", (req, res) => {
    // Extract email parameter and find users with matching email
    const email = req.params.email;
    let filtered_users = users.filter((user) => user.email === email);

    if (filtered_users.length > 0) {
        // Select the first matching user and update attributes if provided
        let filtered_user = filtered_users[0];

         // Extract and update DOB if provided

        let DOB = req.query.DOB;
        if (DOB) {
            filtered_user.DOB = DOB;
        }

        /*
        Include similar code here for updating other attributes as needed
        */

        // Replace old user entry with updated user
        users = users.filter((user) => user.email != email);
        users.push(filtered_user);

        // Send success message indicating the user has been updated
        res.send(`User with the email ${email} updated.`);
    } else {
        // Send error message if no user found
        res.send("Unable to find user!");
    }
});
```

2. The completed code will look like this.

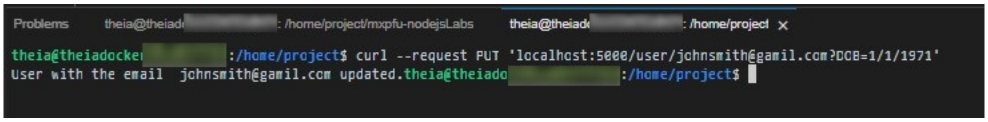


```
// PUT request: Update the details of a user by email ID
router.put("/:email", (req, res) => {
    // Extract email parameter and find users with matching email
    const email = req.params.email;
    let filtered_users = users.filter((user) => user.email === email);

    if (filtered_users.length > 0) {
        // Select the first matching user and update attributes if provided
        let filtered_user = filtered_users[0];

        // Extract and update DOB if provided
        let DOB = req.query.DOB;
        if (DOB) {
            filtered_user.DOB = DOB;
        }

        /*
        Include similar code here for updating other attributes as needed
        */
        // Extract and update firstName if provided
        let firstName = req.query.firstName;
        if (firstName) {
            filtered_user.firstName = firstName;
        }

        // Extract and update lastName if provided
        let lastName = req.query.lastName;
        if (lastName) {
            filtered_user.lastName = lastName;
        }

        // Replace old user entry with updated user
        users = users.filter((user) => user.email != email);
        users.push(filtered_user);

        // Send success message indicating the user has been updated
        res.send(`User with the email ${email} updated.`);
    } else {
        // Send error message if no user found
        res.send("Unable to find user!");
    }
});
```

3. Use the below command to update the `DOB` as `1/1/1971` for the user with mail id 'johnsmith@gamil.com' in the split terminal:

```
curl --request PUT 'localhost:5000/user/johnsmith@gamil.com?DOB=1/1/1971'
```

4. The ouput will be as below:

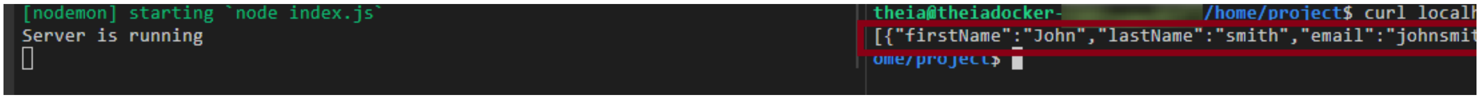


Problems
theia@theiad[redacted]: /home/project/mxpfu-nodejsLabs
theia@theiad[redacted]: /home/project ✕

```
theia@theiadocker[redacted]:/home/project$ curl --request PUT 'localhost:5000/user/johnsmith@gamil.com?DOB=1/1/1971'
User with the email  johnsmith@gamil.com updated.theia@theiado[redacted]:/home/project$ 
```

5. To verify if the `DOB` of the user with email 'johnsmith@gamil.com' has been updated, you can send a GET request as below:

```
curl localhost:5000/user/johnsmith@gamil.com
```



```
[nodemon] starting `node index.js`
Server is running
```

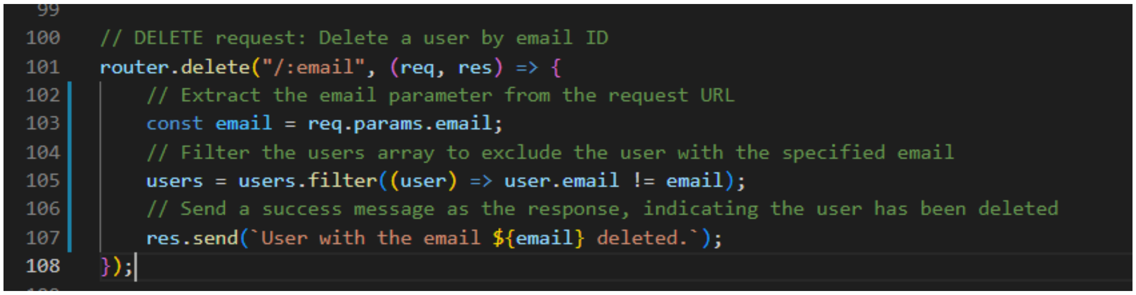
```
theia@theiadocker-[redacted]/home/project$ curl localh
[{"firstName":"John","lastName":"smith","email":"johnsmit
ome/project$ 
```

Exercise 6: Creating the DELETE method:

1. The "D" in CRUD stands for **Delete**. Implement the DELETE method for deleting a specific user's email by using the below code:

```
router.delete("/:email", (req, res) => {
    // Extract the email parameter from the request URL
    const email = req.params.email;
    // Filter the users array to exclude the user with the specified email
    users = users.filter((user) => user.email != email);
    // Send a success message as the response, indicating the user has been deleted
    res.send(`User with the email ${email} deleted.`);
});
```

2. The completed code will look like this.

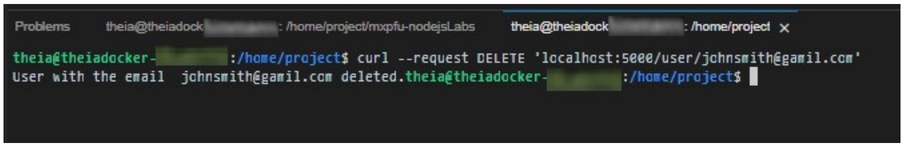


```
// DELETE request: Delete a user by email ID
router.delete("/:email", (req, res) => {
    // Extract the email parameter from the request URL
    const email = req.params.email;
    // Filter the users array to exclude the user with the specified email
    users = users.filter((user) => user.email != email);
    // Send a success message as the response, indicating the user has been deleted
    res.send(`User with the email ${email} deleted.`);
});
```

3. Use the below command to delete the user with mail id 'johnsmith@gamil.com' in the split terminal:

```
curl --request DELETE 'localhost:5000/user/johnsmith@gamil.com'
```

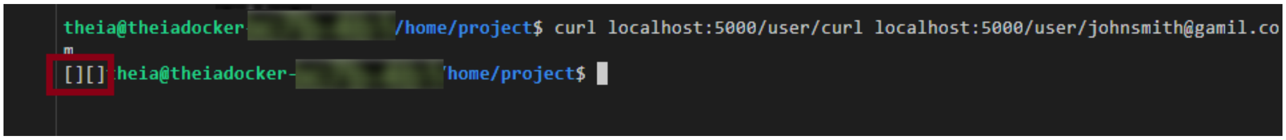
4. The ouput will be as below:



Problems
theia@theiadock[redacted]: /home/project/mxpfu-nodejsLabs
theia@theiadock[redacted]: /home/project

```
theia@theiadocker-[redacted]:/home/project$ curl --request DELETE 'localhost:5000/user/johnsmith@gamil.com'
User with the email  johnsmith@gamil.com deleted.theia@theiadocker-[redacted]:/home/project$
```

5. Send a GET request for the user with email 'johnsmith@gamil.com' and ensure that a null object is returned:



```
theia@theiadocker-[redacted]/home/project$ curl localhost:5000/user/curl localhost:5000/user/johnsmith@gamil.co
m
[][]theia@theiadocker-[redacted]/home/project$
```

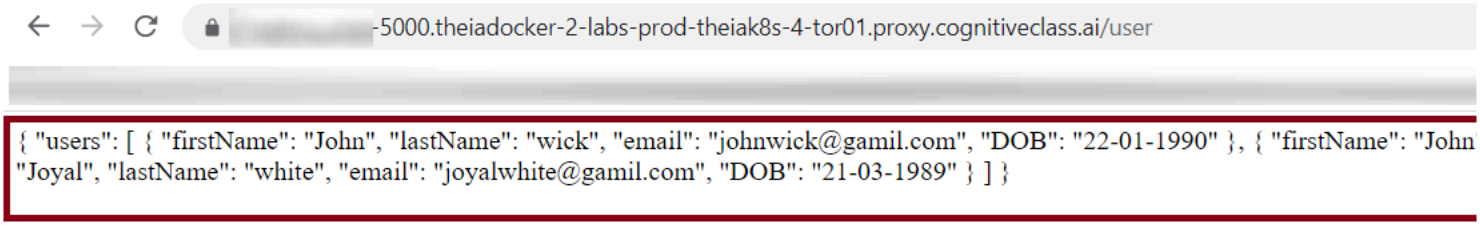
Optional Exercise: Formatting the output

1. To make the output more readable, you can use the JSON stringify method as given below. Please update the code for the GET method to:

```
// Define a route handler for GET requests to the root path "/"
router.get("/",(req,res)=>{
    // Send a JSON response containing the users array, formatted with an indentation of 4 spaces for readability
    res.send(JSON.stringify({users}, null, 4));
});
```

2. Launch the app on port 5000 & append 'user' to the end of the URL.

3. This will render the output of the GET method as a JSON string per the updated GET method shown below:



[redacted]-5000.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user

```
{ "users": [ { "firstName": "John", "lastName": "wick", "email": "johnwick@gamil.com", "DOB": "22-01-1990" }, { "firstName": "John
"Joyal", "lastName": "white", "email": "joyalwhite@gamil.com", "DOB": "21-03-1989" } ] }
```

Exercise 7: Implementing Authentication

All these endpoints are accessible by anyone. You will now see how to add authentication to the CRUD operations. This code has been implemented in **index_withauth.js**.

1. Observe the following code block in **index_withauth.js**.

```
app.use(session({secret:"fingerprint",resave: true, saveUninitialized: true}))
```

This tells your express app to use the session middleware.

- **secret** - a random unique string key used to authenticate a session.
- **resave** - takes a Boolean value. It enables the session to be stored back to the session store, even if the session was never modified during the request.
- **saveUninitialized** - this allows any uninitialized session to be sent to the store. When a session is created but not modified, it is referred to as **uninitialized**.

The default value of both **resave** and **saveUninitialized** is true, but the default is deprecated. So, set the appropriate value according to the use case.

2. Observe the implementation of the **login** endpoint. A user logs into the system providing a username. An access token that is valid for one hour is generated. You may observe this validty length specified by **60 * 60**, which signifies the time in seconds. This access token is set into the session object to ensure that only authenticated users can access the endpoints for that length of time.

```
// Login endpoint
app.post("/login", (req, res) => {
    const user = req.body.user;
    if (!user) {
        return res.status(404).json({ message: "Body Empty" });
    }
    // Generate JWT access token
    let accessToken = jwt.sign({
        data: user
    }, 'access', { expiresIn: 60 * 60 });
    // Store access token in session
    req.session.authorization = {
        accessToken
    }
    return res.status(200).send("User successfully logged in");
});
```

3. Observe the implementation of the authentication middleware. All the endpoints starting with **/user** will go through this middleware. It will retrieve the authorization details from the session and verify it. If the token is validated, the user is authenticated and the control is passed on to the next endpoint handler. If the

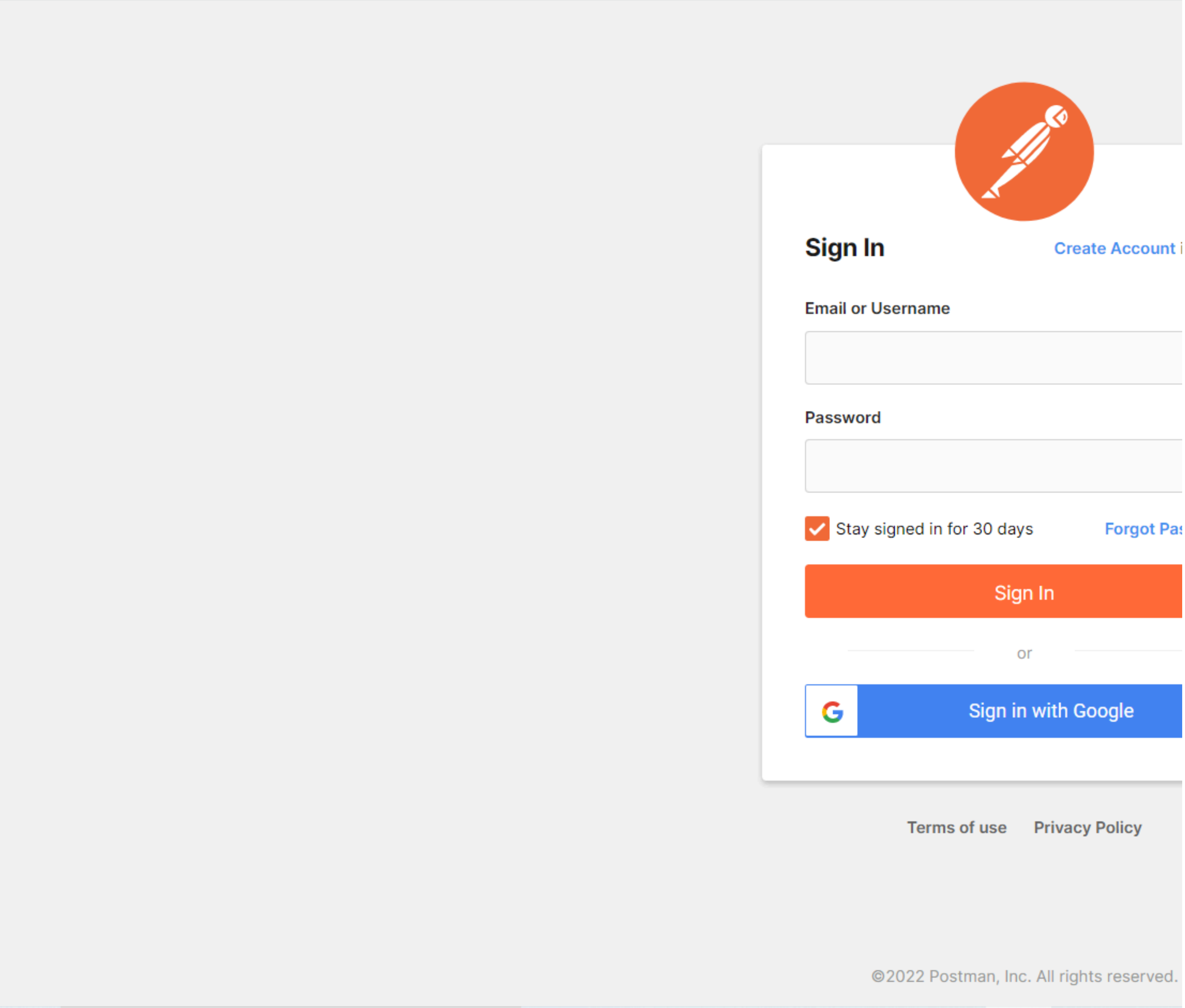
token is invalid, the user is not authenticated and an error message is returned.

```
// Middleware for user authentication
app.use("/user", (req, res, next) => {
    // Check if user is authenticated
    if (req.session.authorization) {
        let token = req.session.authorization['accessToken']; // Access Token

        // Verify JWT token for user authentication
        jwt.verify(token, "access", (err, user) => {
            if (!err) {
                req.user = user; // Set authenticated user data on the request object
                next(); // Proceed to the next middleware
            } else {
                return res.status(403).json({ message: "User not authenticated" }); // Return error if token verification fails
            }
        });

        // Return error if no access token is found in the session
    } else {
        return res.status(403).json({ message: "User not logged in" });
    }
});
```

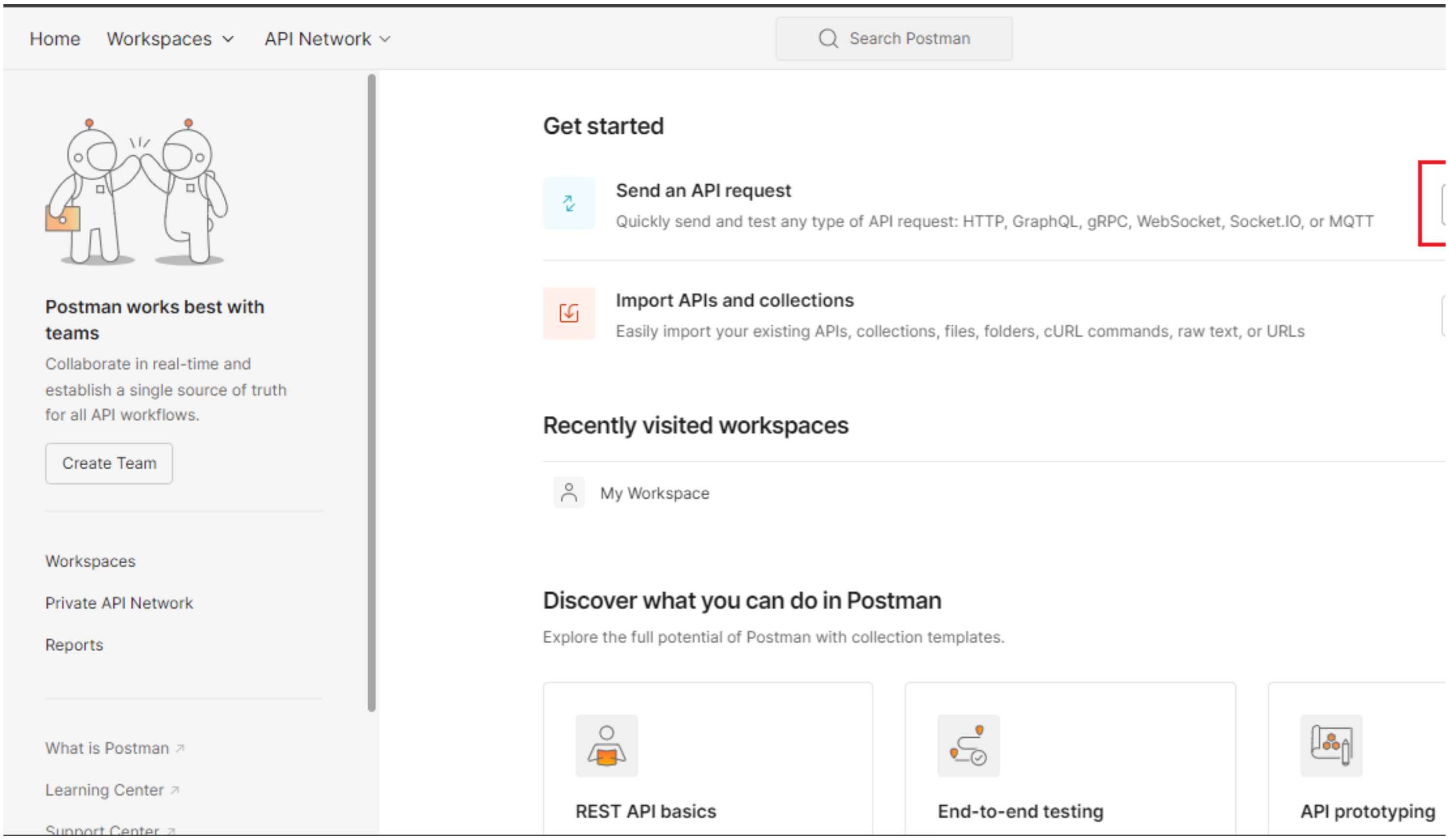
Exercise 8: Testing endpoints with POSTMAN

You have tested the API endpoints with cURL. An easier and more user-friendly way to test these endpoints with the graphical user interface tool (GUI), Postman.

1. Go to Postman. Sign-up for a new Postman account if you don't already have one. Sign-in to your account.



2. After you login to Postman, click on **New Request** as shown below:


Home
Workspaces
API Network
Search Postman
Postman works best with teams
Collaborate in real-time and establish a single source of truth for all API workflows.
Create Team
Workspaces
Private API Network
Reports
What is Postman
Learning Center
Support Center
Get started
Send an API request
Quickly send and test any type of API request: HTTP, GraphQL, gRPC, WebSocket, Socket.IO, or MQTT
Import APIs and collections
Easily import your existing APIs, collections, files, folders, cURL commands, raw text, or URLs
Recently visited workspaces
My Workspace
Discover what you can do in Postman
Explore the full potential of Postman with collection templates.
REST API basics
End-to-end testing
API prototyping

Note: If the server is running in the theia lab please stop the server by press **CTRL + C**. Now start the server by running the below command which will listen to port 5000.

```
npm run start_auth
```

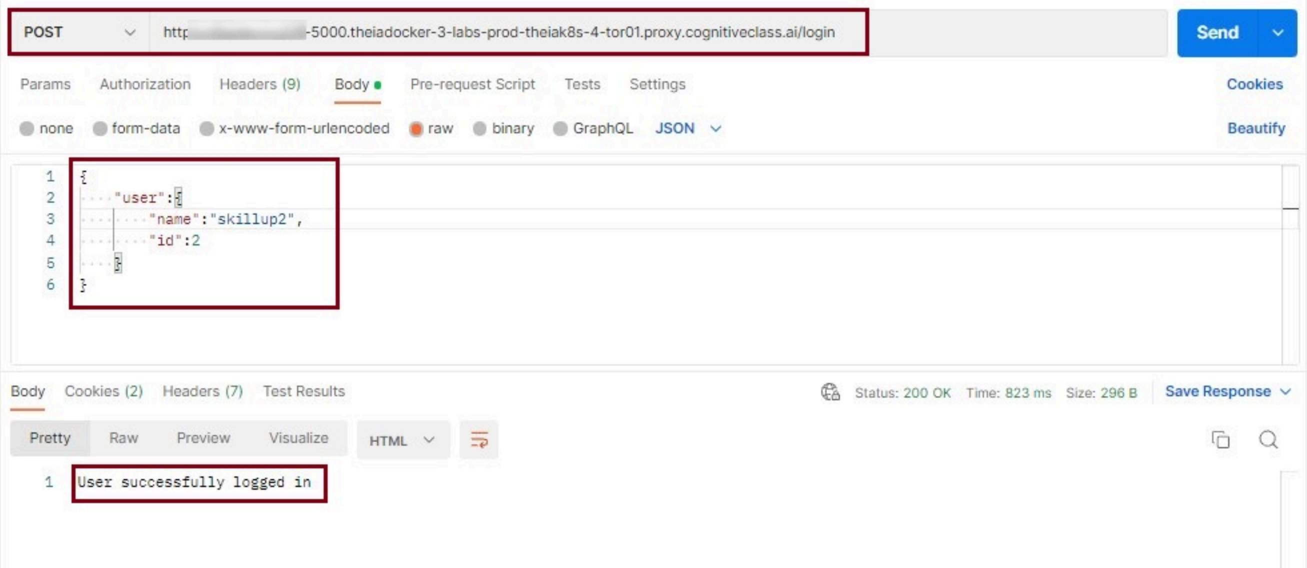
So far we were accessing all the endpoints without authentication but now we will be using authentication to access the endpoints.

3. Copy the URL from the Launch application and add the login as an endpoint to add the user details in the **POST REQUEST** which will look like below:

```
https://<sn-lab-username>-5000.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/login
```

4. User details should be in the below format:

```
{
    "user":{
        "name":"abc",
        "id":1
    }
}
```



POST http[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/login

Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "user":{
        "name":"skillup2",
        "id":2
    }
}
```

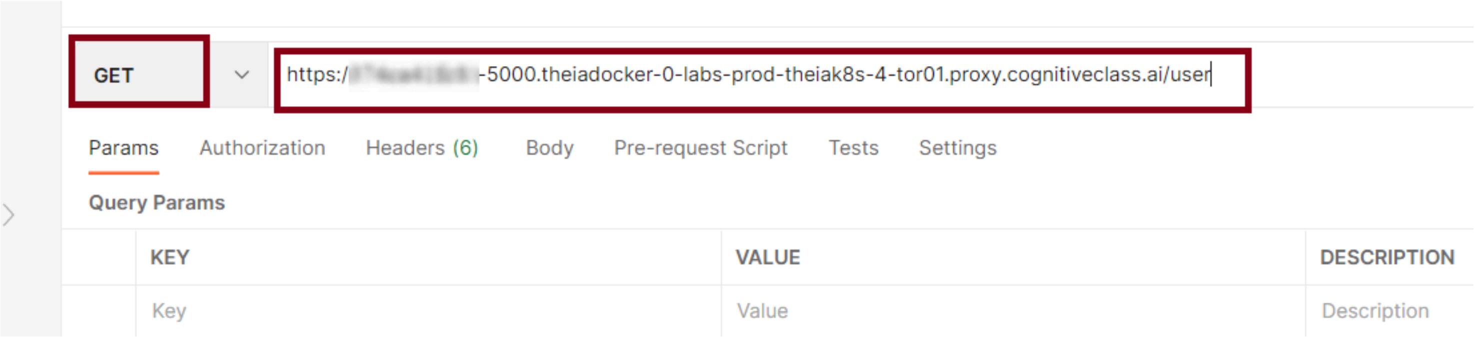
Body Cookies (2) Headers (7) Test Results
Status: 200 OK Time: 823 ms Size: 296 B Save Response
Pretty Raw Preview Visualize HTML

```
User successfully logged in
```

Now let's begin the test by sending an HTTP GET Request.

8.1 GET request

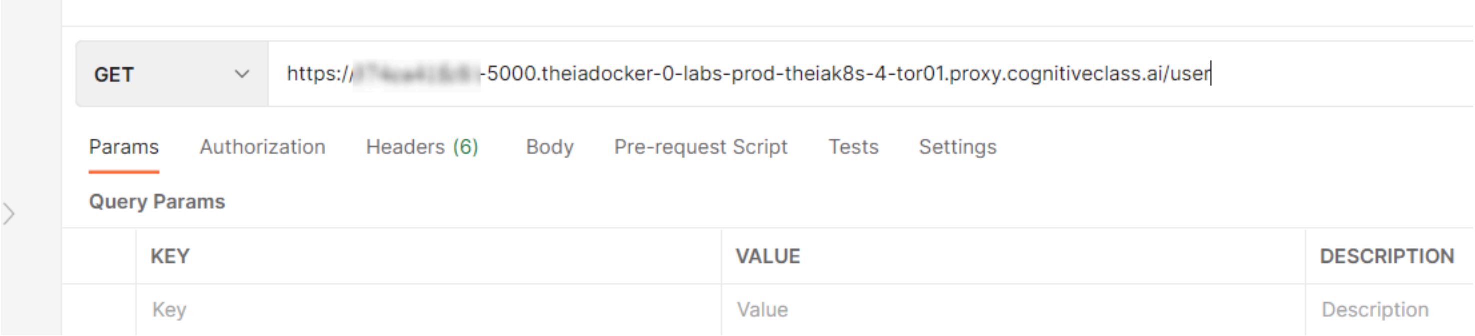
a. Enter the GET request URL: `https://XXXXXXXXXX-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user` in your input box of Postman where you see "Enter Request URL".



GET https://[redacted]-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user

Params Authorization Headers (6) Body Pre-request Script Tests Settings
Query Params
KEY VALUE DESCRIPTION
Key Value Description

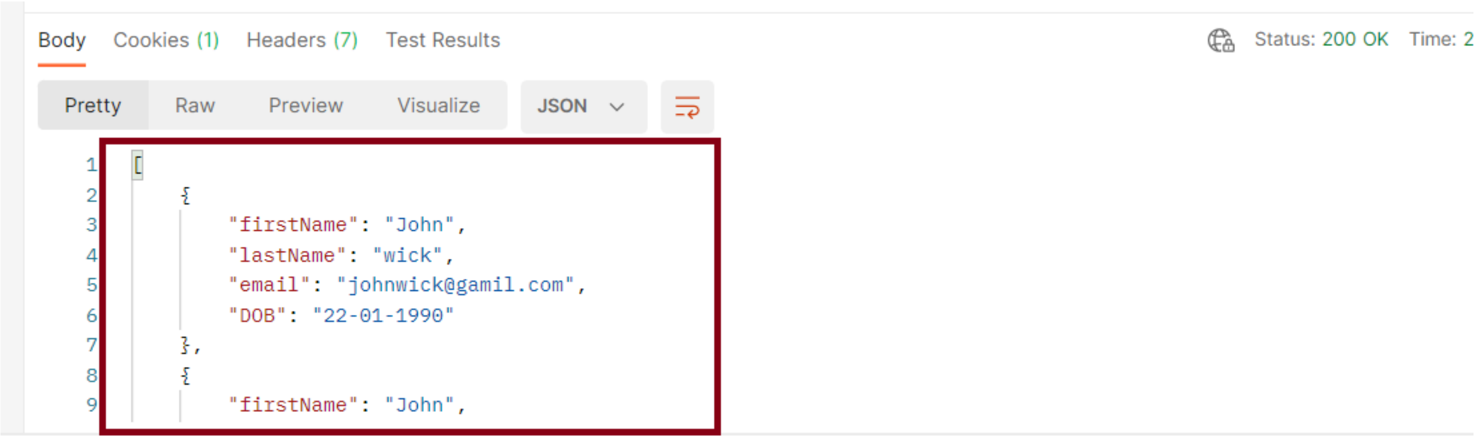
b. Click on the `Send` button after entering the URL.



GET https://[redacted]-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user

Params Authorization Headers (6) Body Pre-request Script Tests Settings
Query Params
KEY VALUE DESCRIPTION
Key Value Description

c. The output will be as below:

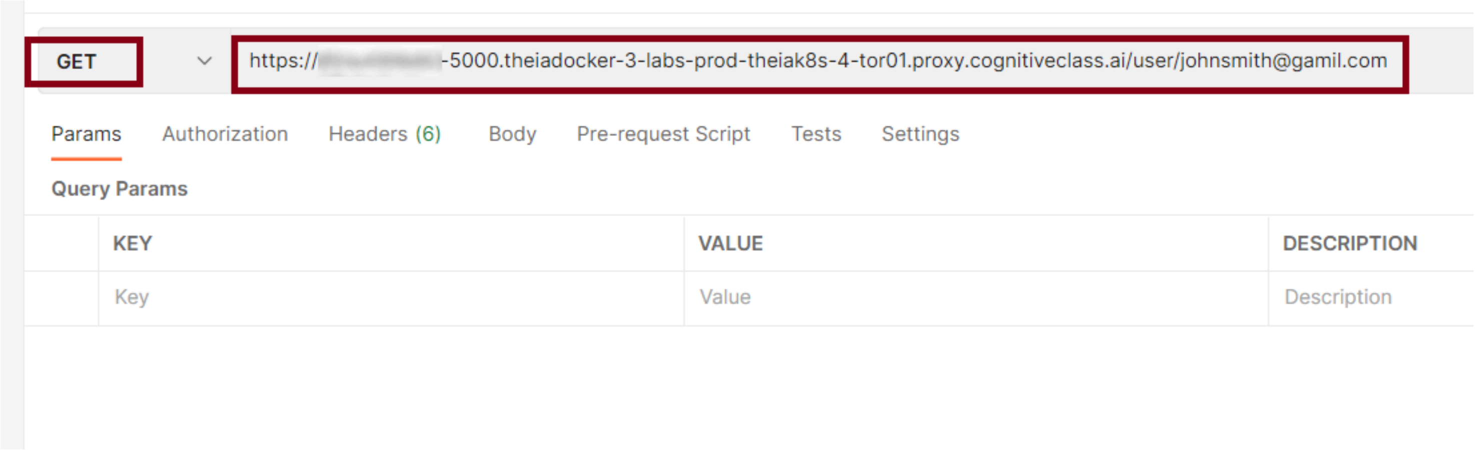


```
[
    {
        "firstName": "John",
        "lastName": "wick",
        "email": "johnwick@gamil.com",
        "DOB": "22-01-1990"
    },
    {
        "firstName": "John",
```

8.2 GET request by specific ID

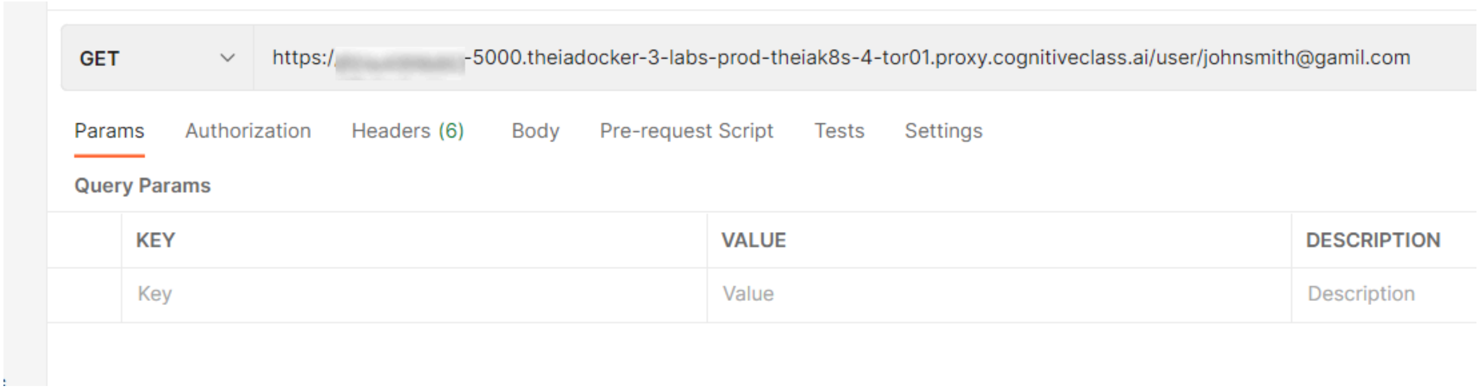
a. Enter the request URL by adding the specific email address to the above GET request URL. If the email address is johnsmith@gamil.com then enter the following URL in the input box of postman:

```
https://XXXXXXXXXX-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/johnsmith@gamil.com
```



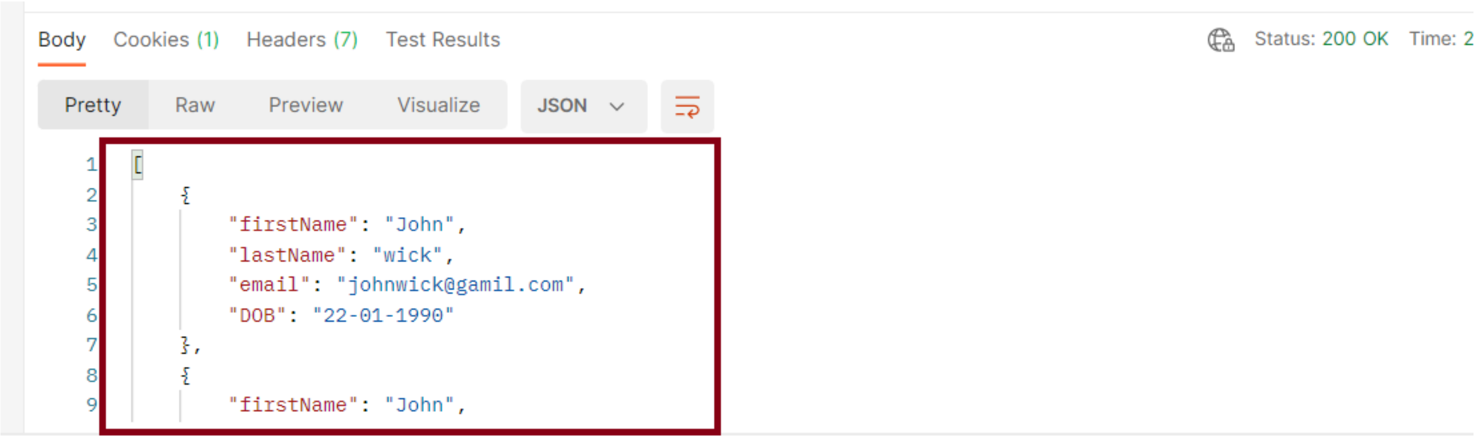
GET https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/johnsmith@gamil.com

b. Click on the **Send** button after entering the URL to view the output.



GET https:/[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/johnsmith@gamil.com

c. The output will be as below:



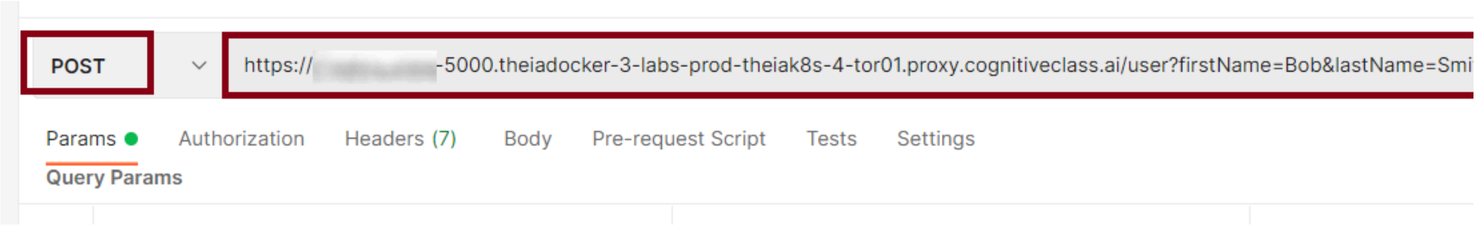
```
[
    {
        "firstName": "John",
        "lastName": "wick",
        "email": "johnwick@gamil.com",
        "DOB": "22-01-1990"
    },
    {
        "firstName": "John",
```

8.3 POST request :

a. Enter the basic post request URL:

```
https://XXXXXXXXXX-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/
```

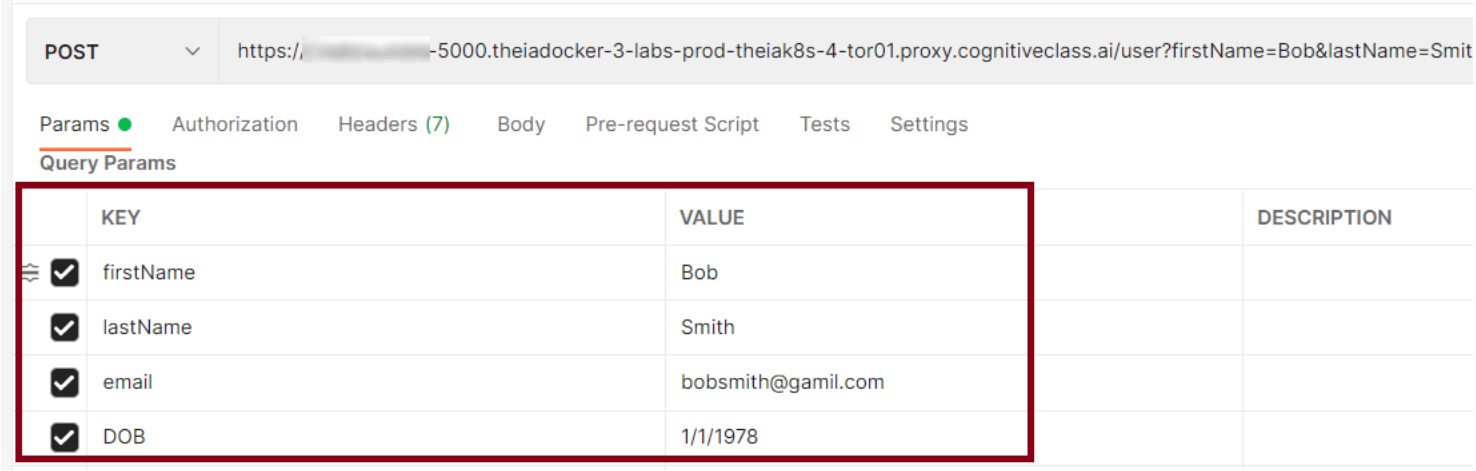
Ensure to select the POST method and select the "Params".



POST

https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user?firstName=Bob&lastName=Smi

b. Enter the firstName as 'Bob', lastName as 'Smith', email as 'bobsmith@gamil.com' and DOB as '1/1/1978' for a new user:

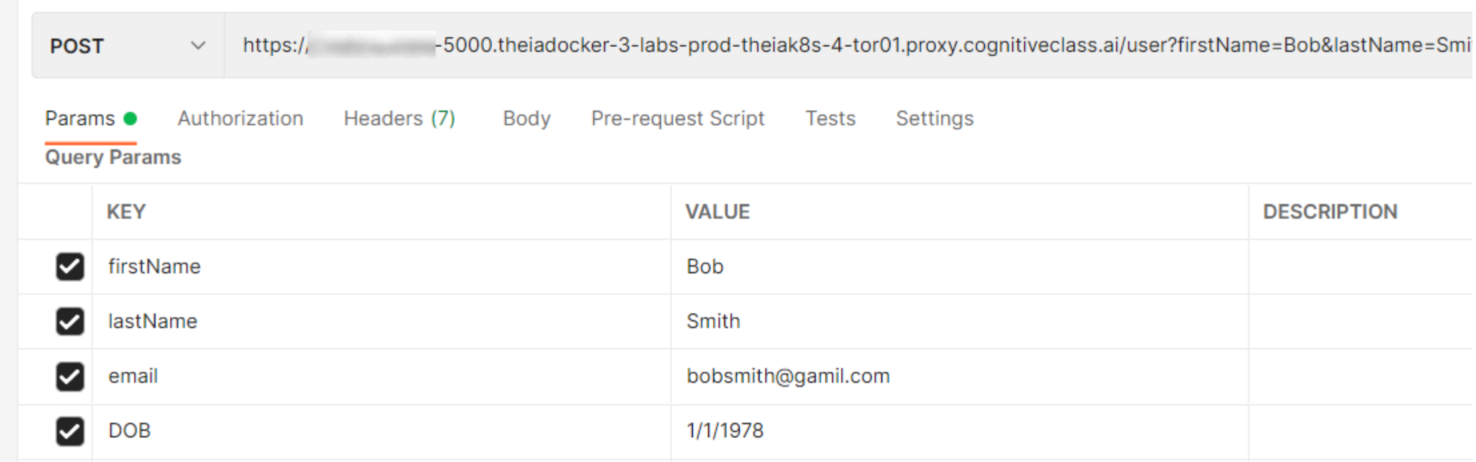


POST

https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user?firstName=Bob&lastName=Smit

	KEY	VALUE	DESCRIPTION
☑	firstName	Bob	
☑	lastName	Smith	
☑	email	bobsmith@gamil.com	
☑	DOB	1/1/1978	

c. Click on the **Send** button after entering the URL to view the output.



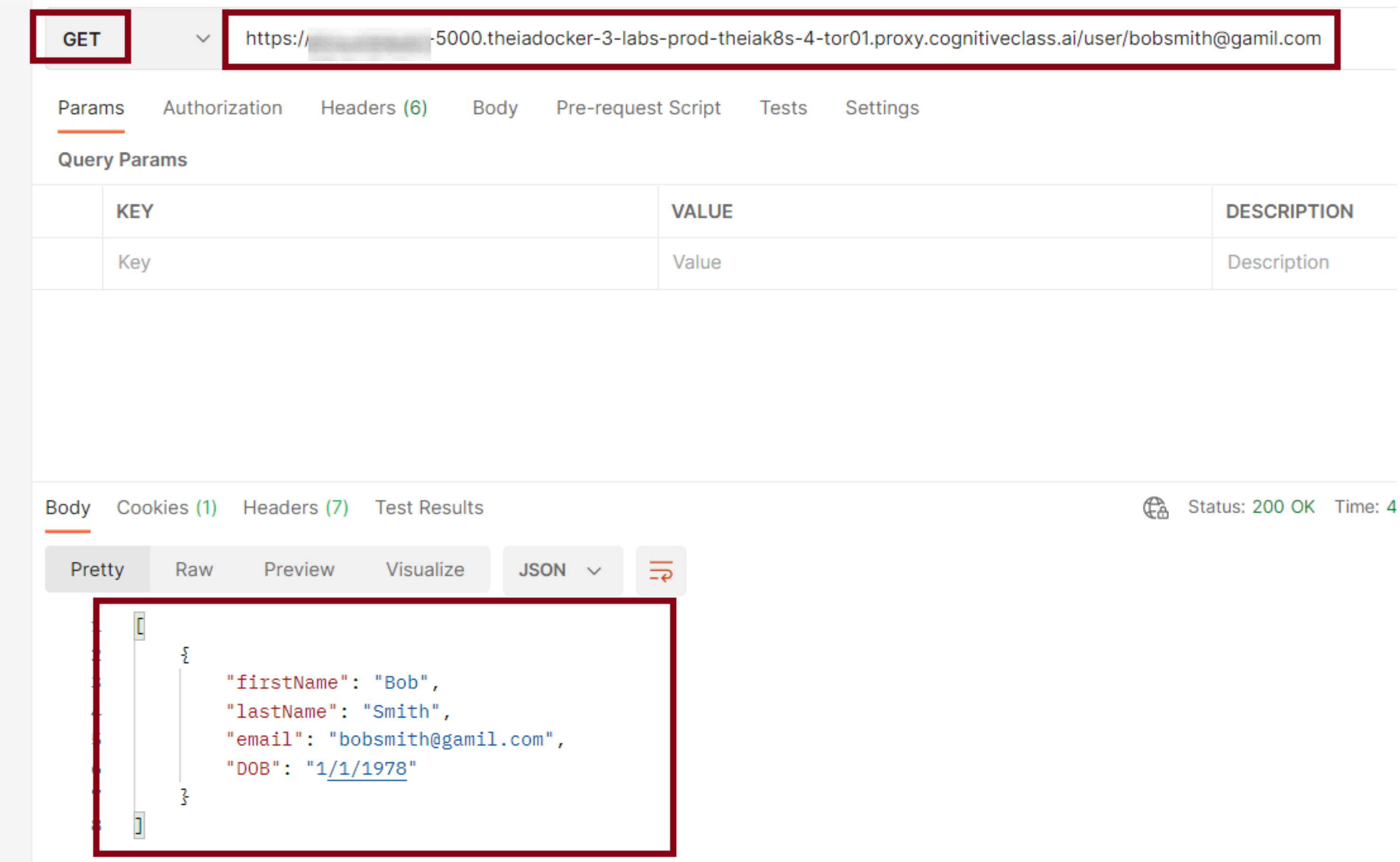
POST https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user?firstName=Bob&lastName=Smi

Query Params

	KEY	VALUE	DESCRIPTION
☑	firstName	Bob	
☑	lastName	Smith	
☑	email	bobsmith@gamil.com	
☑	DOB	1/1/1978	

Verify that the newly added values have been updated by doing the GET request.

Note: Ensure that you delete any parameters that you added for the POST request before sending the GET request.



GET https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com

Query Params

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

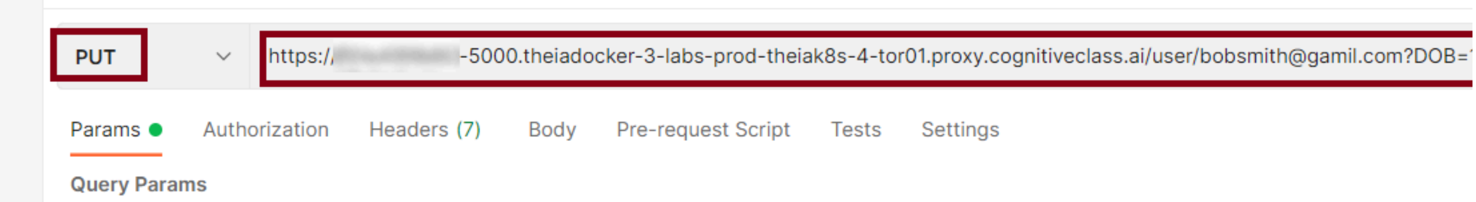
```
[
    {
        "firstName": "Bob",
        "lastName": "Smith",
        "email": "bobsmith@gamil.com",
        "DOB": "1/1/1978"
    }
]
```

8.4 PUT request

a. Enter the URL by adding the specific email address. If the email address is bobsmith@gamil.com then enter this URL in the input box of the Postman:

```
https://XXXXXXXXXX-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com
```

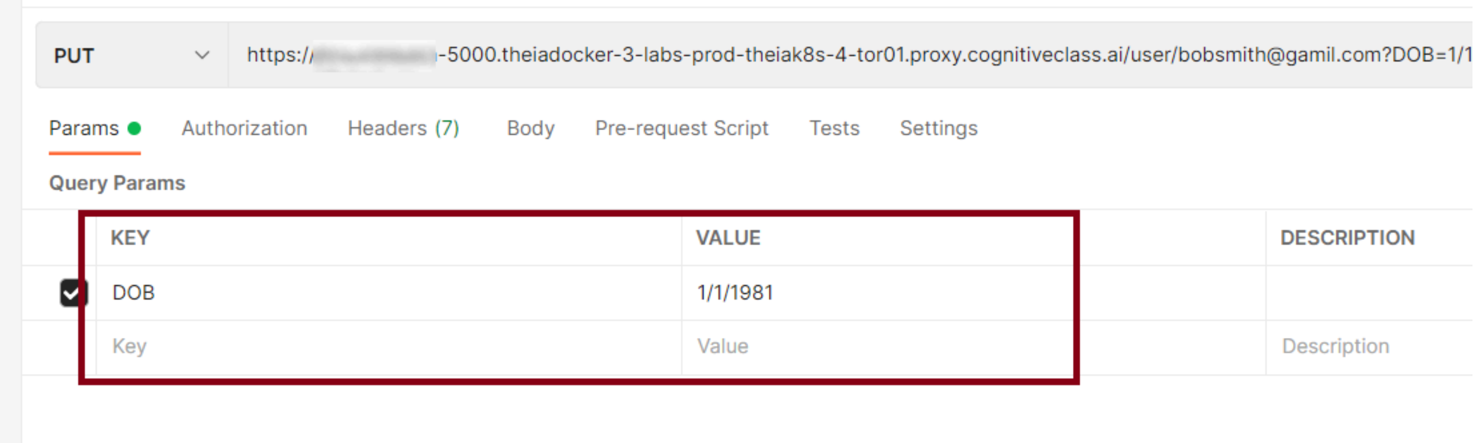
Ensure to select the PUT method and select the "Params".



PUT

https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com?DOB=1

b. Enter the key and values to be changed. For example, if you want to change the "DOB" key and replace it with the new value 1/1/1981 it will be as below.

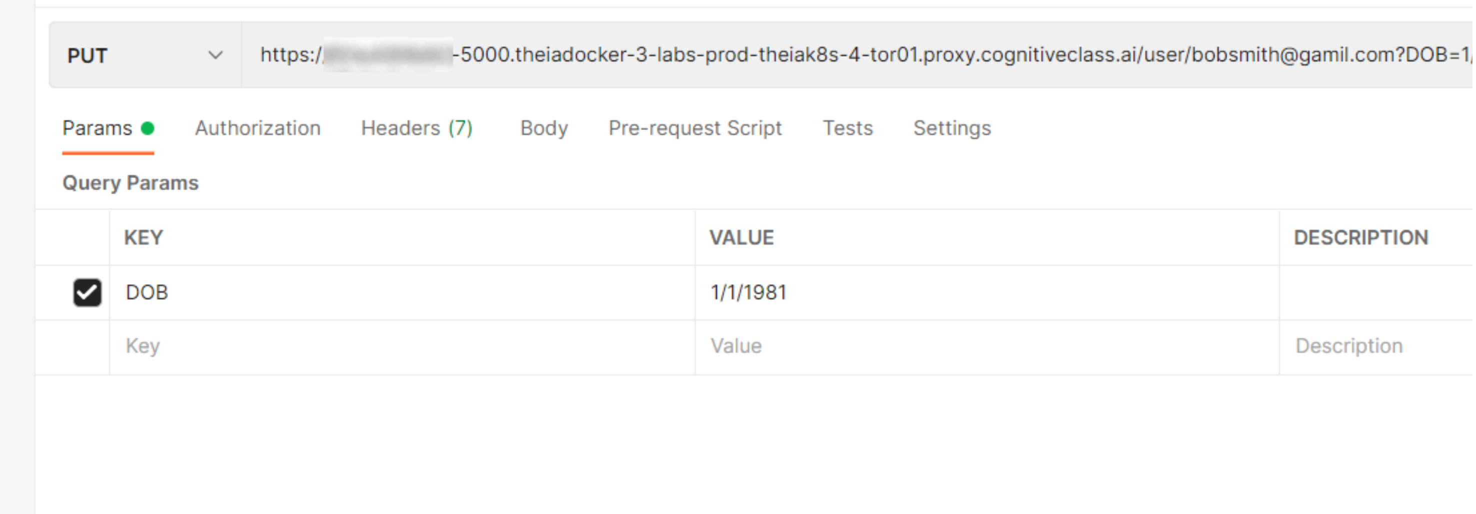


PUT

https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com?DOB=1/1

	KEY	VALUE	DESCRIPTION
☑	DOB	1/1/1981	
	Key	Value	Description

c. Click on the **Send** button after entering the URL to view the output.



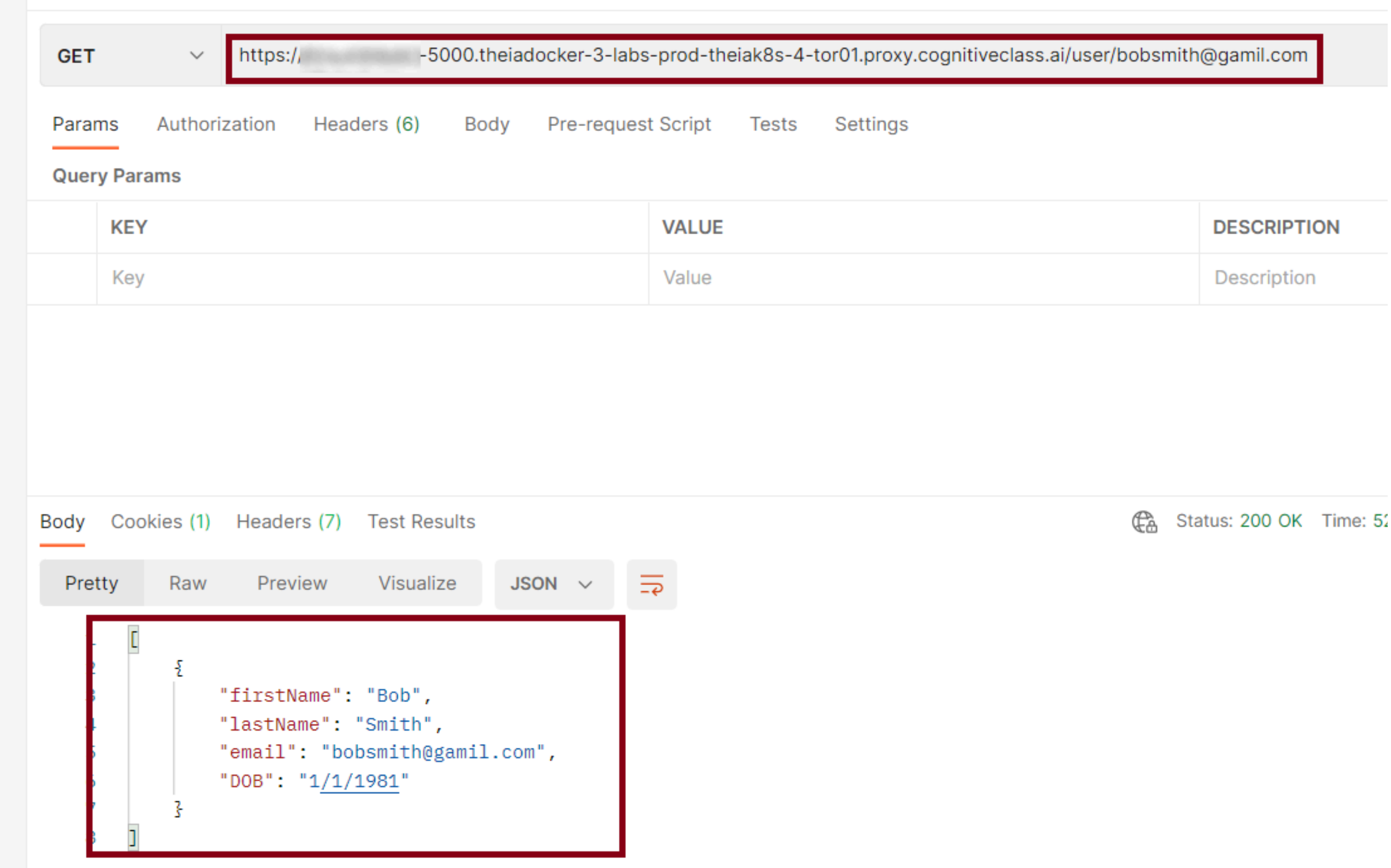
PUT

https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com?DOB=1/

	KEY	VALUE	DESCRIPTION
☑	DOB	1/1/1981	
	Key	Value	Description

Verify that the newly added values are been updated by doing a GET request.

Note: Ensure that you delete any parameters that you added for the PUT request before sending the GET request.



GET https://[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com

Query Params

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

Status: 200 OK Time: 52

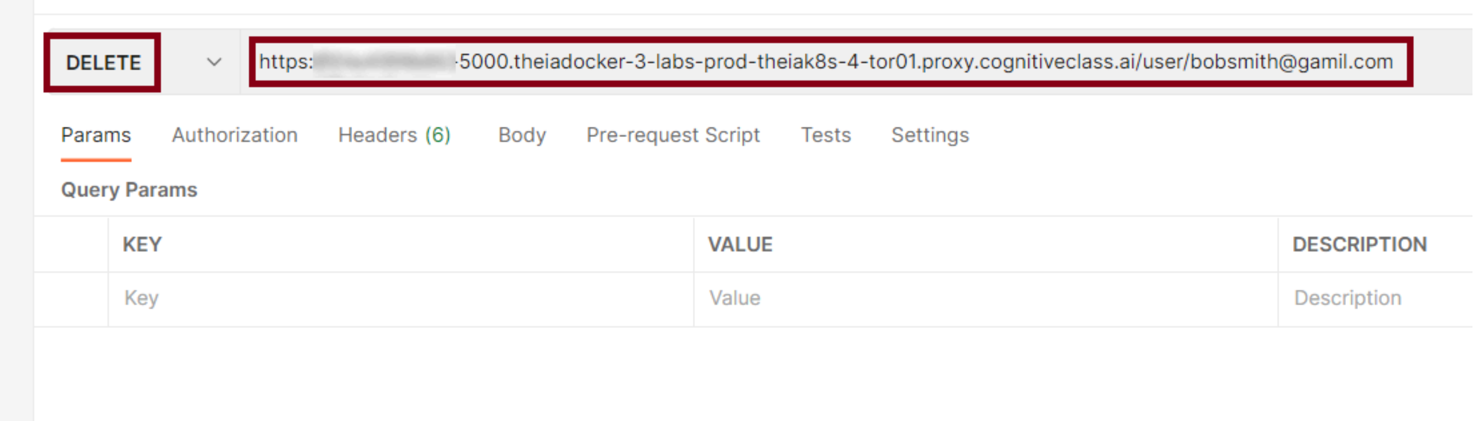
```
[
    {
        "firstName": "Bob",
        "lastName": "Smith",
        "email": "bobsmith@gamil.com",
        "DOB": "1/1/1981"
    }
]
```

8.5 DELETE Request:

a. Enter the URL by adding the specific email address. If the email address is bobsmith@gamil.com then enter this URL in the input box of the Postman:

```
https://XXXXXXXXXX-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com
```

Be sure to select the DELETE method.

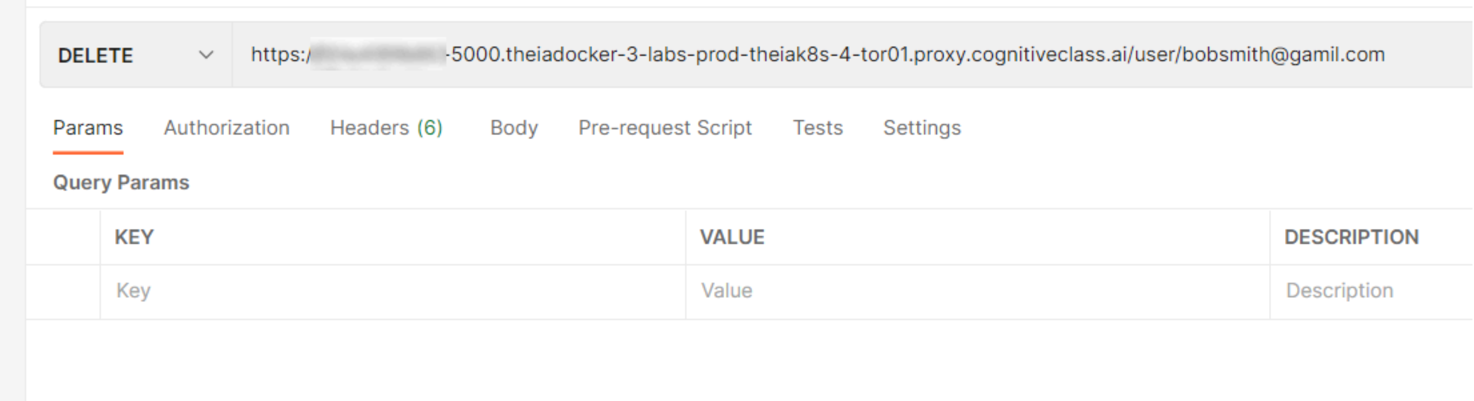


DELETE https:[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com

Query Params

	KEY	VALUE	DESCRIPTION
	Key	Value	Description

b. Click on the "Send" button after entering the URL to view the output.

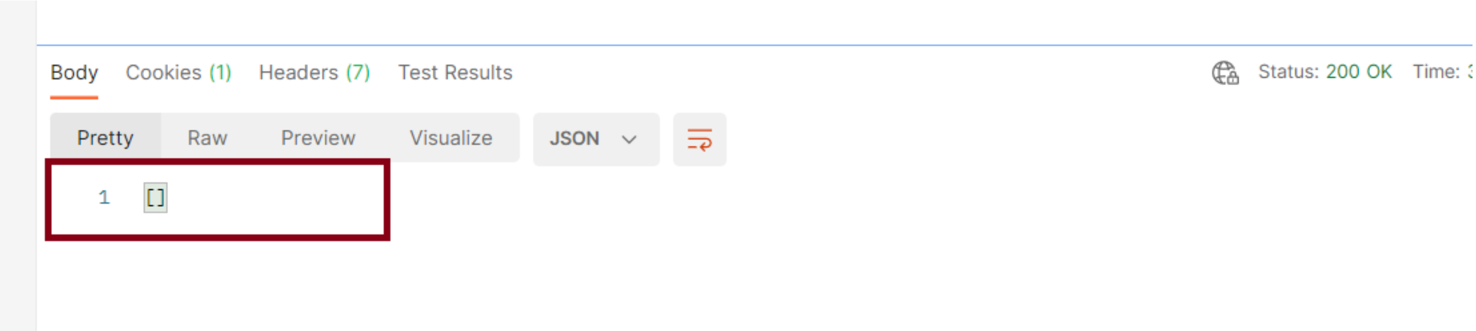


DELETE https:/[redacted]-5000.theiadocker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/bobsmith@gamil.com

KEY	VALUE	DESCRIPTION
Key	Value	Description

c. Verify that the GET user by ID bobsmith@gamil.com returns a null object by sending a GET request.

Note: Ensure that you delete any parameters (if any are there) before sending the GET request.



```
1 []
```

Practice labs

1. Create an endpoint in the same code for getting all users with a particular Last Name.

- ▶ Click here for a hint!
- ▶ Click here for the Solution!

2. Create an endpoint in the same code for sorting users by date of birth.

- ▶ Click here for a hint!
- ▶ Click here for the Solution!

Congratulations! You have completed the lab for CRUD operations with Node.js and Express.js using Postman.

Summary:

In this lab, we have performed CRUD Operations like GET, POST, PUT and DELETE on an Express App and tested the above methods using Postman.

Author(s)

Sapthashree K S

K Sundararajan