动手实验 – 使用 Node.js 进行 CRUD 操作

Skills Network

预计所需时间：1小时

在本实验中，您将学习如何使用 Express 服务器创建一个 **好友列表**。您的应用程序应该允许您添加好友，包含以下详细信息：名、姓、电子邮件 和 出生日期。您还将为应用程序提供检索详细信息、修改详细信息和删除详细信息的功能。

您将创建一个应用程序，使用 API 端点对上述数据执行创建、检索、更新和删除操作，使用 Express 服务器。

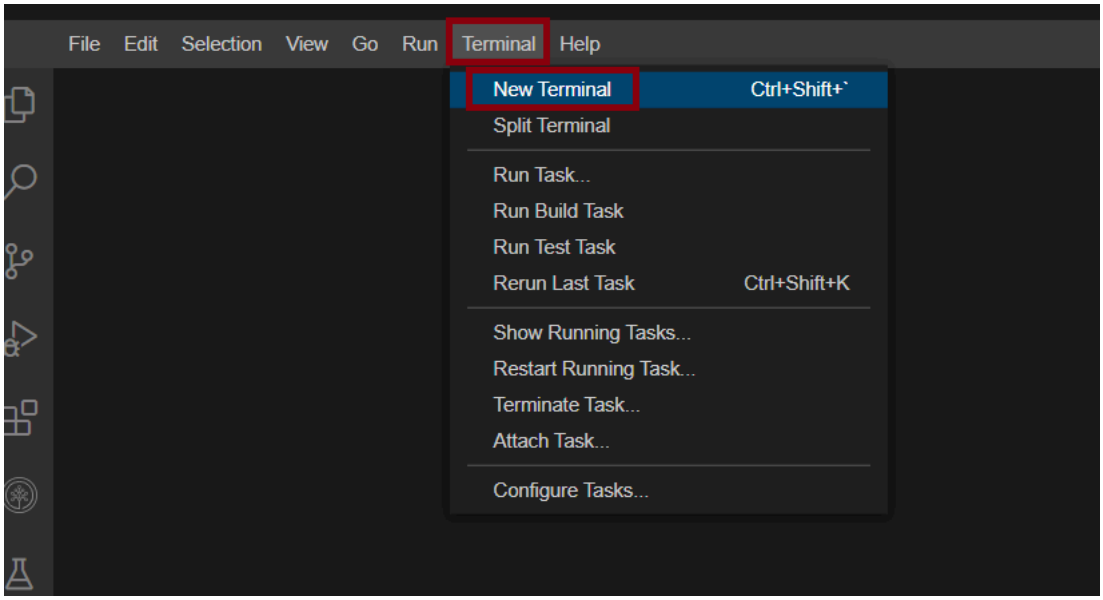
您还将学习如何为端点提供经过身份验证的访问。您将使用 cURL 和 Postman 来测试已实现的端点。

目标：

- 创建 API 端点以在 Express 服务器上对瞬态数据执行创建、检索、更新和删除操作。
- 使用 JSON Web Tokens (JWT) 在会话级别实现身份验证，以便进行授权访问。

设置：创建应用程序

1. 通过编辑器中的菜单打开终端窗口：终端 > 新建终端。



2. 如果您尚未在项目文件夹中，请切换到您的项目文件夹。

```
cd /home/project
```

3. 运行以下命令以克隆包含本实验所需启动代码的 git 仓库，如果该仓库尚不存在。

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

```
theia@theiadocke          /home/project$ [ ! -d 'mxpfu-nodejsLabs' ] && git clone https://github.com/ibm-develope
-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          /home/project$
```

5. 切换到 **mxpfu-nodejsLabs** 目录以开始进行实验。

```
cd mxpfu-nodejsLabs/
```

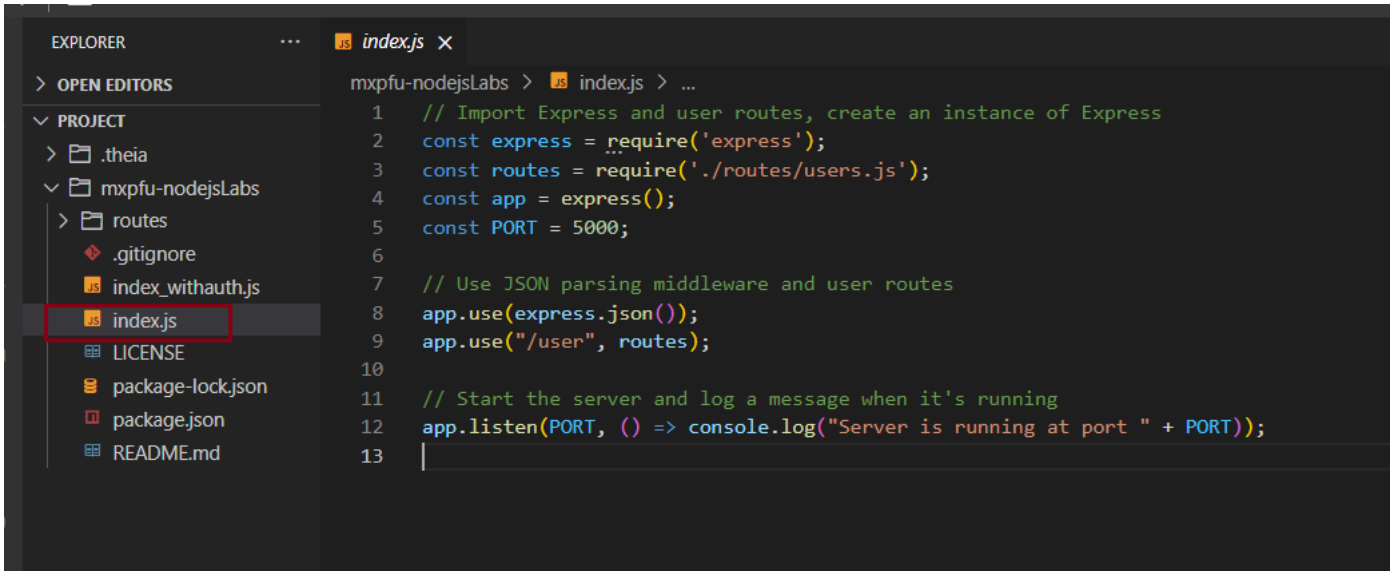
6. 列出该目录的内容以查看此实验的工件。

```
ls
```

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

练习 1：了解服务器应用程序

1. 在文件资源管理器中打开 **mxpfu-nodejsLabs** 文件夹并查看 **index.js**。



你有一个已配置为在 5000 端口运行的 Express 服务器。当你通过 **/user** 访问服务器时，可以访问在 **routes/users.js** 中定义的端点。

回想一下，GET、POST、PUT 和 DELETE 是常用的 HTTP 方法，用于执行 CRUD 操作。这些操作用于从服务器检索和发送数据。

- `GET` 用于请求指定资源的数据。
- `POST` 用于向服务器发送数据以创建资源。
- `PUT` 用于向服务器发送数据以更新资源。
- `DELETE` 用于删除指定资源。

  `POST` 和 `PUT` 有时可以互换使用。

2. 本实验需要安装一些软件包。**express** 和 **nodemon** 软件包用于启动和运行 Express 服务器，**jsonwebtoken** 和 **express-session** 用于基于会话的身份验证。

- **express** – 用于创建服务器以提供 API 端点。
- **nodemon** – 当你对代码进行任何更改时，它将帮助重启服务器。
- **jsonwebtoken** – 该软件包帮助生成 JSON Web Token，我们将用于身份验证。**JSON Web Token (JWT)** 是一个 JSON 对象，用于在互联网上安全地传递信息（在两个方之间）。它可以用于信息交换，通常用于身份验证系统。
- **express-session** – 该软件包将帮助我们维护会话的身份验证。

这些软件包在 packages.json 中定义为 **dependencies**。

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

3. 请注意，express 应用使用中间件 **express.json()** 来处理请求作为 json 对象。

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

4. 请注意，express 应用使用路由来处理以 **/user** 开头的端点。这意味着对于所有以 **/user** 开头的端点，服务器将去 **users.js** 中查找端点处理程序。

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

5. 所有的端点在 **users.js** 中都有骨架，但实现是可工作的。导航到 routes 目录下的 users.js，并观察其中定义的端点。

练习 2：运行服务器

提供的起始代码是一个功能正常的服务器，返回虚拟值。在开始实现实际端点之前，请运行服务器。

1. 在终端中，打印工作目录以确保您在 /home/projects/mxpfu-nodejsLabs。

   ```
   pwd
   ```

2. 安装运行服务器所需的所有软件包。复制、粘贴并运行以下命令。

   ```
   npm install
   ```

   这将安装在 packages.json 中定义的所有必需包。

3. 启动 express 服务器。

   ```
   npm start
   ```

4. 从顶部菜单中打开一个 **新终端**。测试一个端点以检索这些用户。此功能尚未实现以返回用户。

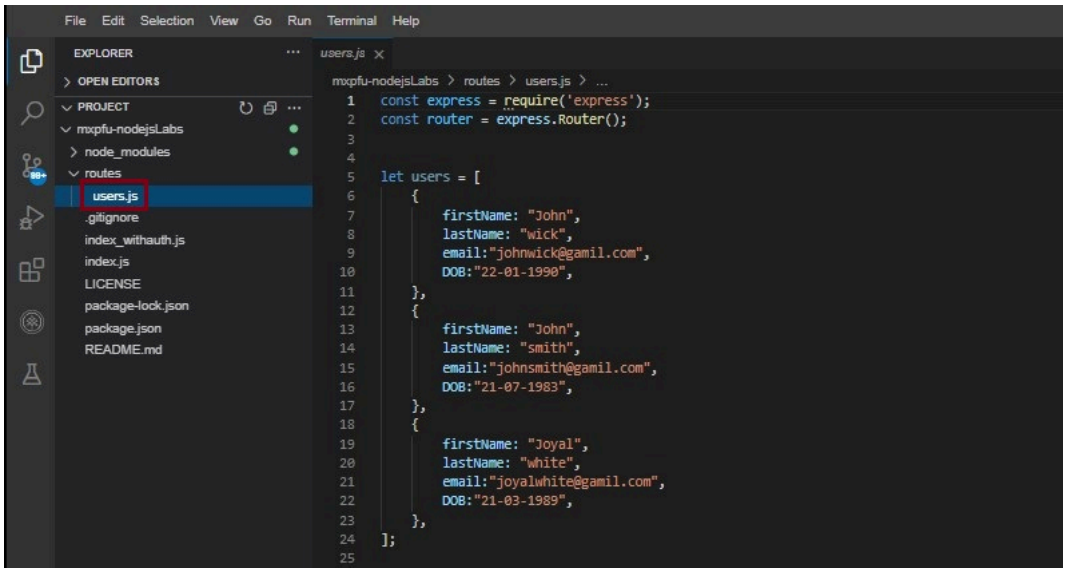
   ```
   curl localhost:5000/user
   ```

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

5. 如果您看到如上所示的输出，这意味着服务器正在正常运行。

练习 2：实现你的端点

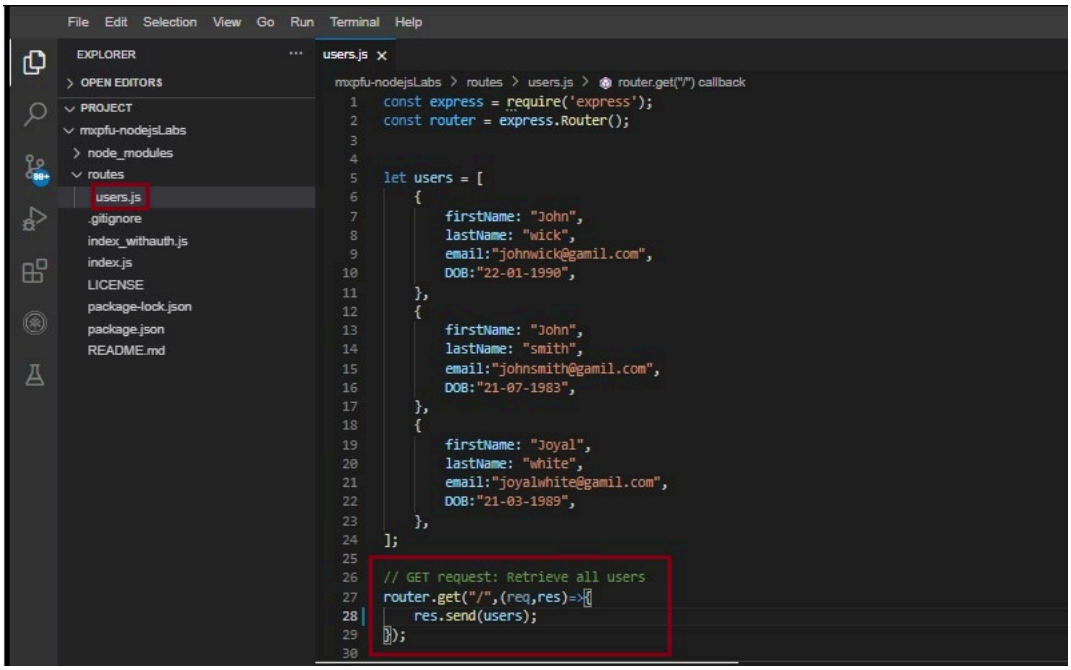
1. 导航到 **routes** 文件夹中的 **users.js** 文件。端点已经定义，并为你实现端点提供了空间。



2. CRUD 中的 **R** 代表检索。你将首先添加一个 API 端点，使用 **get** 方法获取所有用户的详细信息。起始代码中已经添加了一些用户。

- 复制下面的代码并粘贴到 **router.get("/",(req,res)=>{}** 方法中的 { } 括号内。

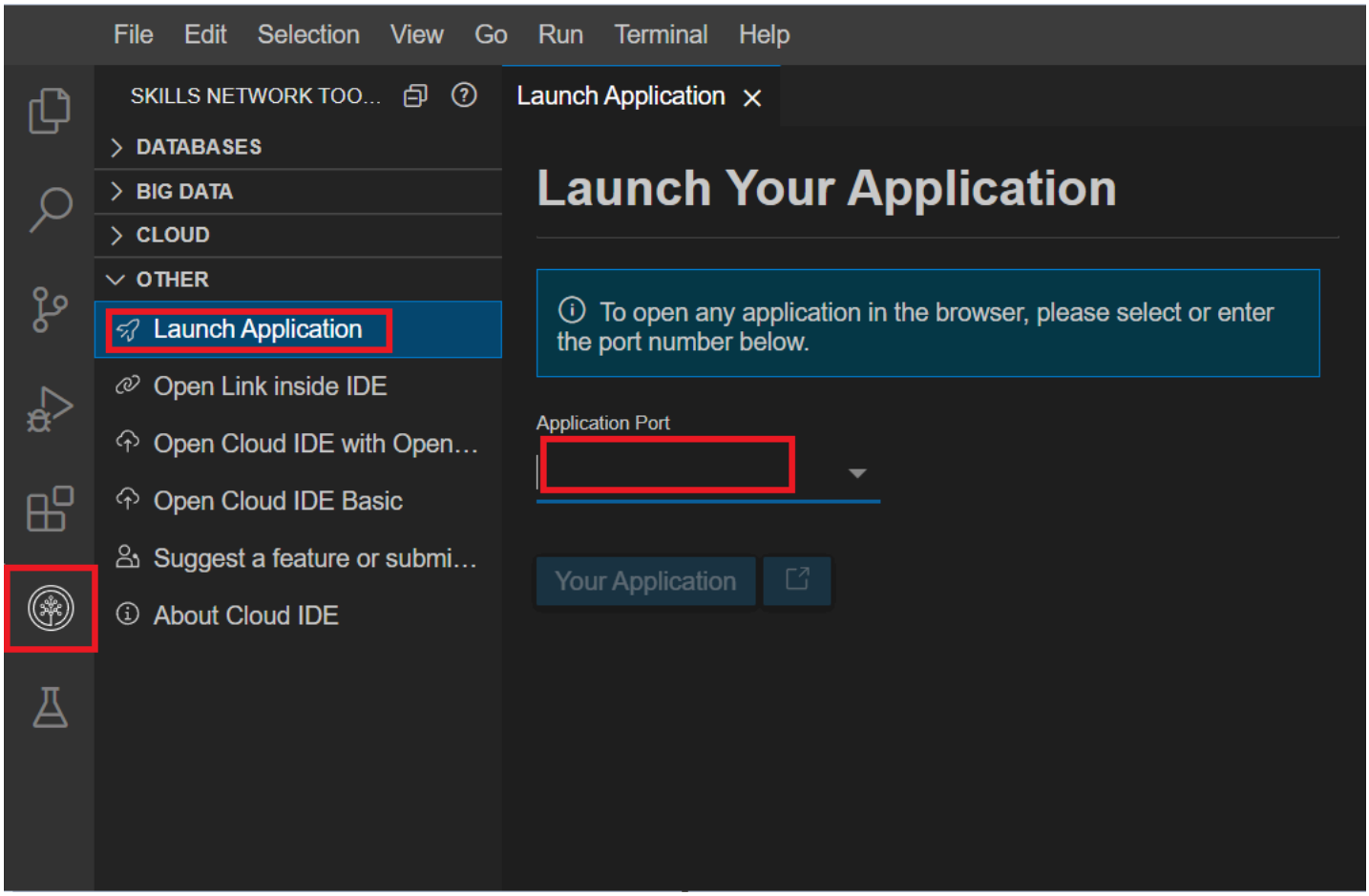
  ```
  res.send(users);
  ```



3. 确保你的服务器正在运行。当你对代码进行更改时，之前任务中启动的服务器应该重新启动。如果服务器未运行，请再次启动它。

```
npm start
```

3. 点击左侧的 **Skills Network** 按钮。这将打开“技能网络工具箱”。然后点击 `OTHER` 然后点击 `Launch Application`。从那里你应该能够将端口输入为 `5000` 并启动开发服务器。



4. 当浏览器页面打开时，在地址栏的 URL 末尾添加 **/user**。你将看到下面的页面。



5. 使用 curl 命令检查 GET 请求的输出，就像你在之前的练习中所做的那样。

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

练习 3：通过特定电子邮件方法创建 GET：

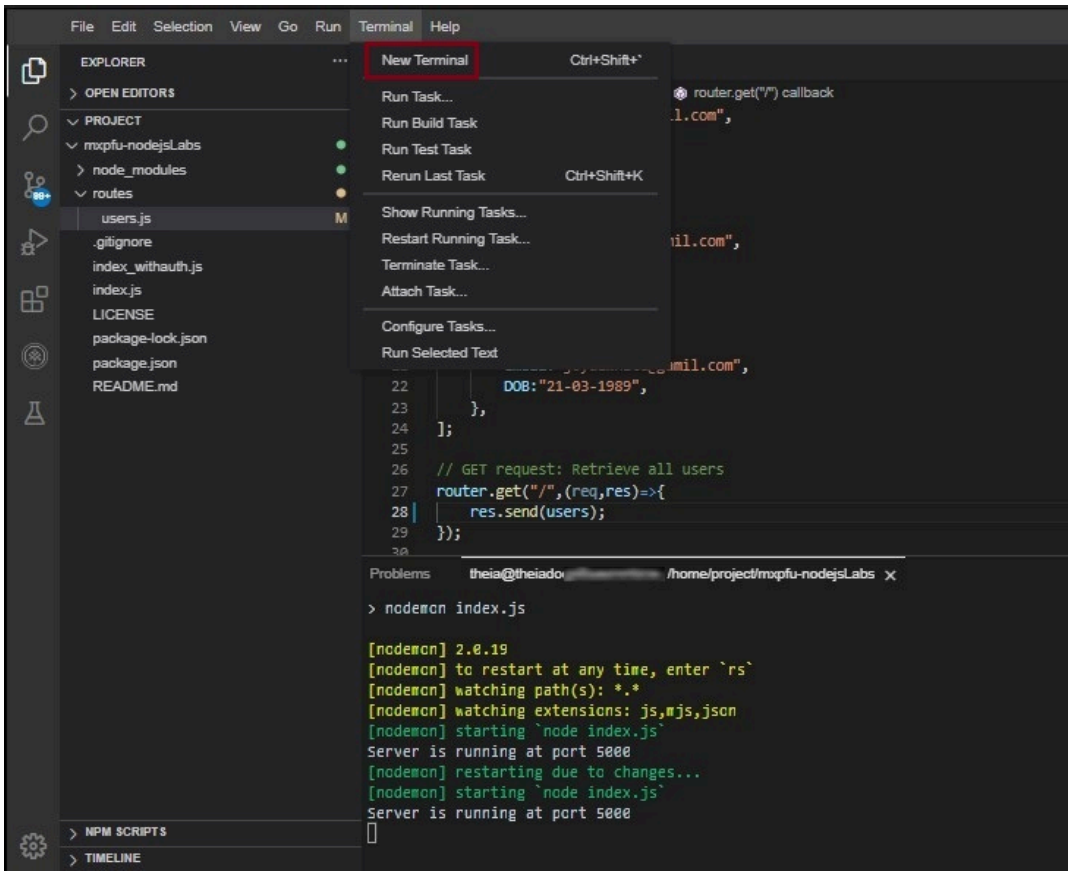
1. 实现一个 get 方法，通过在用户集合上使用 `filter` 方法，根据用户的电子邮件 ID 获取特定用户的详细信息。编写代码并保存后，服务器将重新启动。

▼ 点击这里查看代码

```
router.get("/:email",(req,res)=>{
    // Extract the email parameter from the request URL
    const email = req.params.email;
    // Filter the users array to find users whose email matches the extracted email parameter
    let filtered_users = users.filter((user) => user.email === email);
    // Send the filtered_users array as the response to the client
    res.send(filtered_users);
});
```

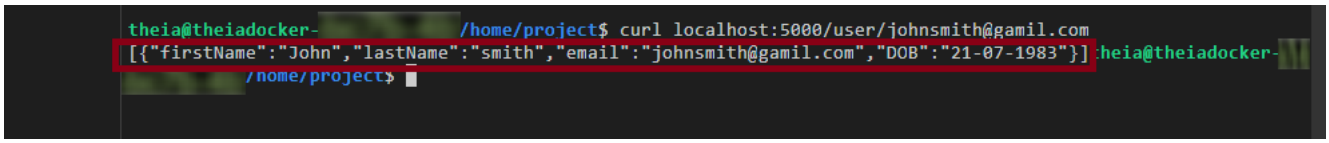
```
// GET by specific ID request: Retrieve a single user with email ID
router.get("/:email",(req,res)=>{
    // Extract the email parameter from the request URL
    const email = req.params.email;
    // Filter the users array to find users whose email matches the extracted email parameter
    let filtered_users = users.filter((user) => user.email === email);
    // Send the filtered_users array as the response to the client
    res.send(filtered_users);
});
```

2. 点击终端 > 新建终端



3. 在新终端中，使用以下命令查看邮件 ID 为 johnsmith@gamil.com 的用户的输出

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



练习 4：创建 POST 方法：

1. CRUD 中的 C 代表 **创建**。实现 **/user** 端点，使用 POST 方法创建用户并将用户添加到列表中。您可以将用户对象创建为字典。您可以使用下面显示的示例用户对象。

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

使用 `push` 将字典添加到用户列表中。用户详细信息可以作为名为 *firstName, lastName, DOB 和 email* 的查询参数传递。

提示：可以使用 request.query.paramname 从请求对象中检索查询参数

▼ 点击这里查看代码

```
router.post("/",(req,res)=>{
    // Push a new user object into the users array based on query parameters from the request
    users.push({
        "firstName": req.query.firstName,
        "lastName": req.query.lastName,
        "email": req.query.email,
        "DOB": req.query.DOB
    });
    // Send a success message as the response, indicating the user has been added
    res.send("The user " + req.query.firstName + " has been added!");
});
```

```
42
43  // POST request: Create a new user
44  router.post("/",(req,res)=>{
45      // Push a new user object into the users array based on query parameters from the request
46      users.push({
47          "firstName": req.query.firstName,
48          "lastName": req.query.lastName,
49          "email": req.query.email,
50          "DOB": req.query.DOB
51      });
52      // Send a success message as the response, indicating the user has been added
53      res.send("The user " + req.query.firstName + " has been added!");
54  });
55
```

2. 使用下面的命令在新终端上发布一个邮件地址为 ‘jonlovato@theworld.com‘ 的新用户：

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

3. 输出将如下所示：

```
Problems  theia@theiadocke...:/home/project/mxpfu-nodejsLabs  theia@theiadoc...:/home/project
theia@theiadocke...:/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-...:/home/project$
```

4. 要验证电子邮件为 ‘jonlovato@theworld.com‘ 的用户是否已添加，可以发送如下的 GET 请求：

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

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

练习 5：创建 PUT 方法：

1. CRUD 中的 U 代表更新，可以通过 PUT 方法实现。要对数据进行更新，您将使用 PUT 方法。您应该首先查看具有指定电子邮件 ID 的用户，然后对其进行修改。下面的代码展示了如何修改用户的出生日期（DOB）。请进行必要的代码更改，以允许修改用户的其他属性。

```
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. 完整的代码将如下所示。

```
// 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. 使用以下命令在分割终端中将用户的 DOB 更新为 1/1/1971，该用户的邮箱为 ‘johnsmith@gamil.com‘：

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

4. 输出将如下所示：

```
Problems  theia@theiad...: /home/project/mxpfu-nodejsLabs  theia@theiad...: /home/project
theia@theiadocke...:/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...:/home/project$
```

5. 要验证邮箱为 ‘johnsmith@gamil.com‘ 的用户的 DOB 是否已更新，可以发送如下的 GET 请求：

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

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

```
theia@theiadocker-...:/home/project$ curl l
[{"firstName":"John","lastName":"smith","email":"joh
```

练习 6：创建 DELETE 方法：

1. CRUD 中的 "D" 代表 **删除**。使用以下代码实现 DELETE 方法以删除特定用户的电子邮件：

```
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.`);
});
```

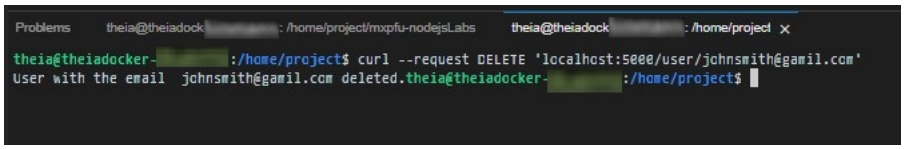
2. 完成的代码将如下所示。

```
// 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.`);
});
```

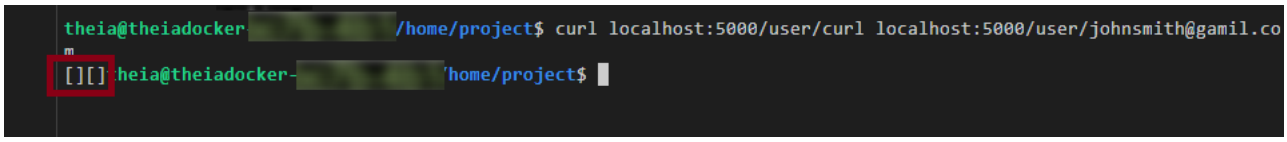
3. 在分割终端中使用以下命令删除邮箱为 ‘johnsmith@gamil.com‘ 的用户：

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

4. 输出将如下所示：



5. 发送一个针对电子邮件为 ‘johnsmith@gamil.com‘ 的用户的 GET 请求，并确保返回一个空对象：



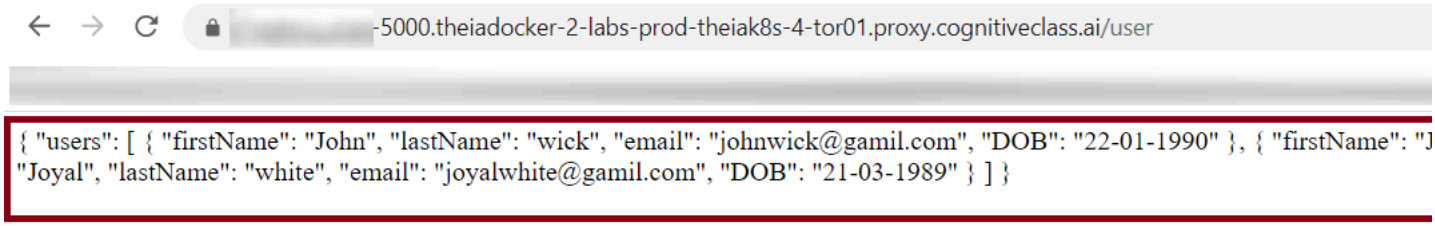
可选练习：格式化输出

1. 为了使输出更易读，您可以使用如下所示的 JSON 字符串化方法。请将 GET 方法的代码更新为：

```
// 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. 在5000端口启动应用程序，并在URL末尾添加'user'。

3. 这将根据下面显示的更新后的GET方法，将GET方法的输出呈现为JSON字符串：



练习 7：实现身份验证

所有这些端点都可以被任何人访问。您将看到如何为 CRUD 操作添加身份验证。此代码已在 **index_withauth.js** 中实现。

1. 请查看 **index_withauth.js** 中的以下代码块。

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

这告诉你的 Express 应用使用会话中间件。

- **secret** – 一个随机的唯一字符串密钥，用于验证会话。
- **resave** – 接受一个布尔值。即使会话在请求期间从未被修改，也允许将会话存储回会话存储中。
- **saveUninitialized** – 这允许任何未初始化的会话被发送到存储中。当会话被创建但未被修改时，称为 **未初始化**。

resave 和 **saveUninitialized** 的默认值都是 true，但默认值已被弃用。因此，根据用例设置适当的值。

2. 观察 **login** 端点的实现。用户提供用户名登录系统。生成一个有效期为一小时的访问令牌。你可以观察到这个有效期长度由 **60 * 60** 指定，表示时间以秒为单位。这个访问令牌被设置到会话对象中，以确保只有经过身份验证的用户才能在该时间段内访问端点。

```
// 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. 观察身份验证中间件的实现。所有以 **/user** 开头的端点将经过此中间件。它将从会话中检索授权详细信息并进行验证。如果令牌有效，则用户已通过身份验证，控制权将传递给下一个端点处理程序。如果令牌无效，则用户未通过身份验证，并返回错误消息。

```
// 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

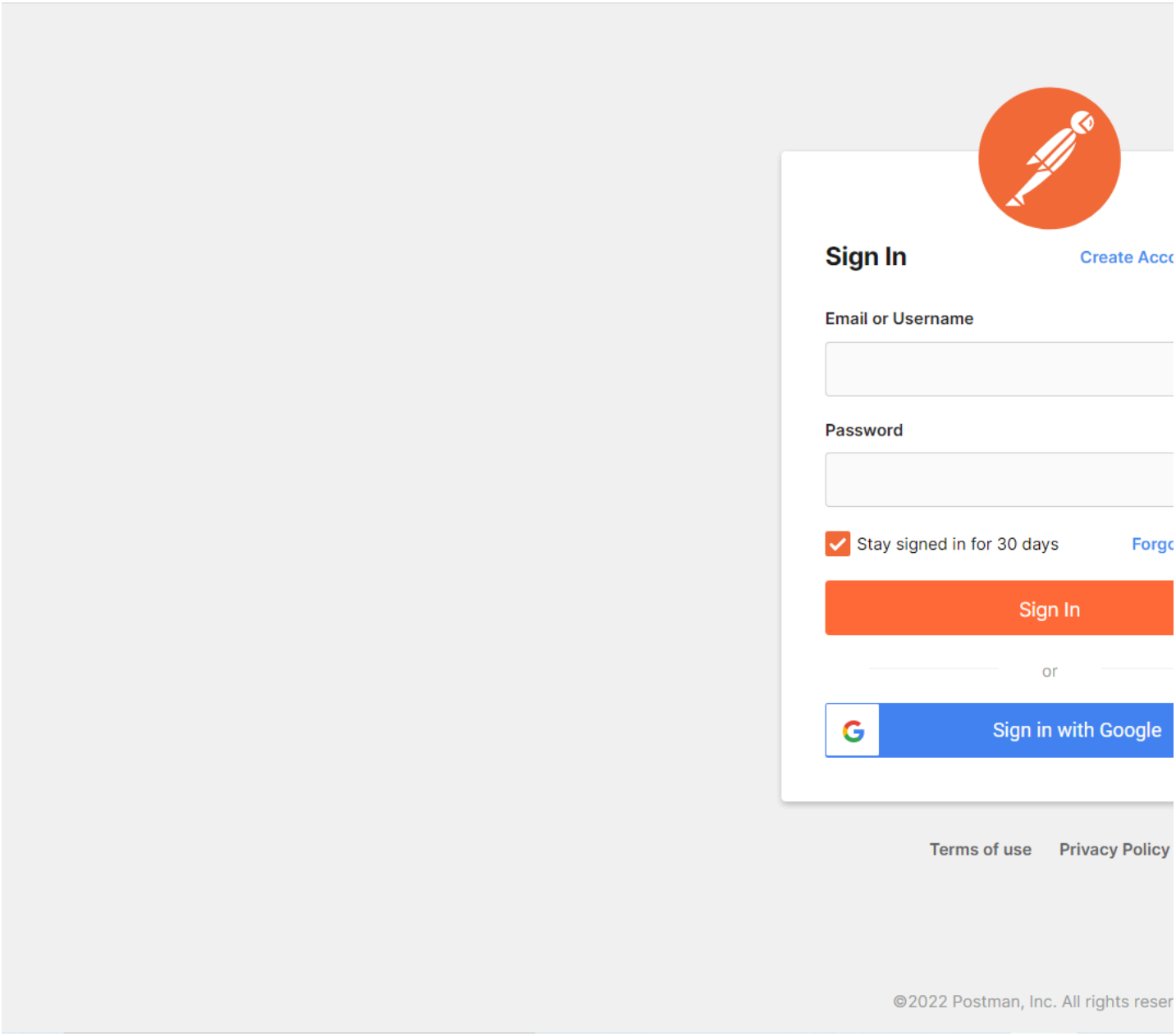
        // Verify JWT token for user authentication
        jwt.verify(token, "access", (err, user) => {
            if (!err) {
                req.user = user; // Set authenticated user data on the request object
                next(); // Proceed to the next middleware
            } else {
                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" });
    }
});
```

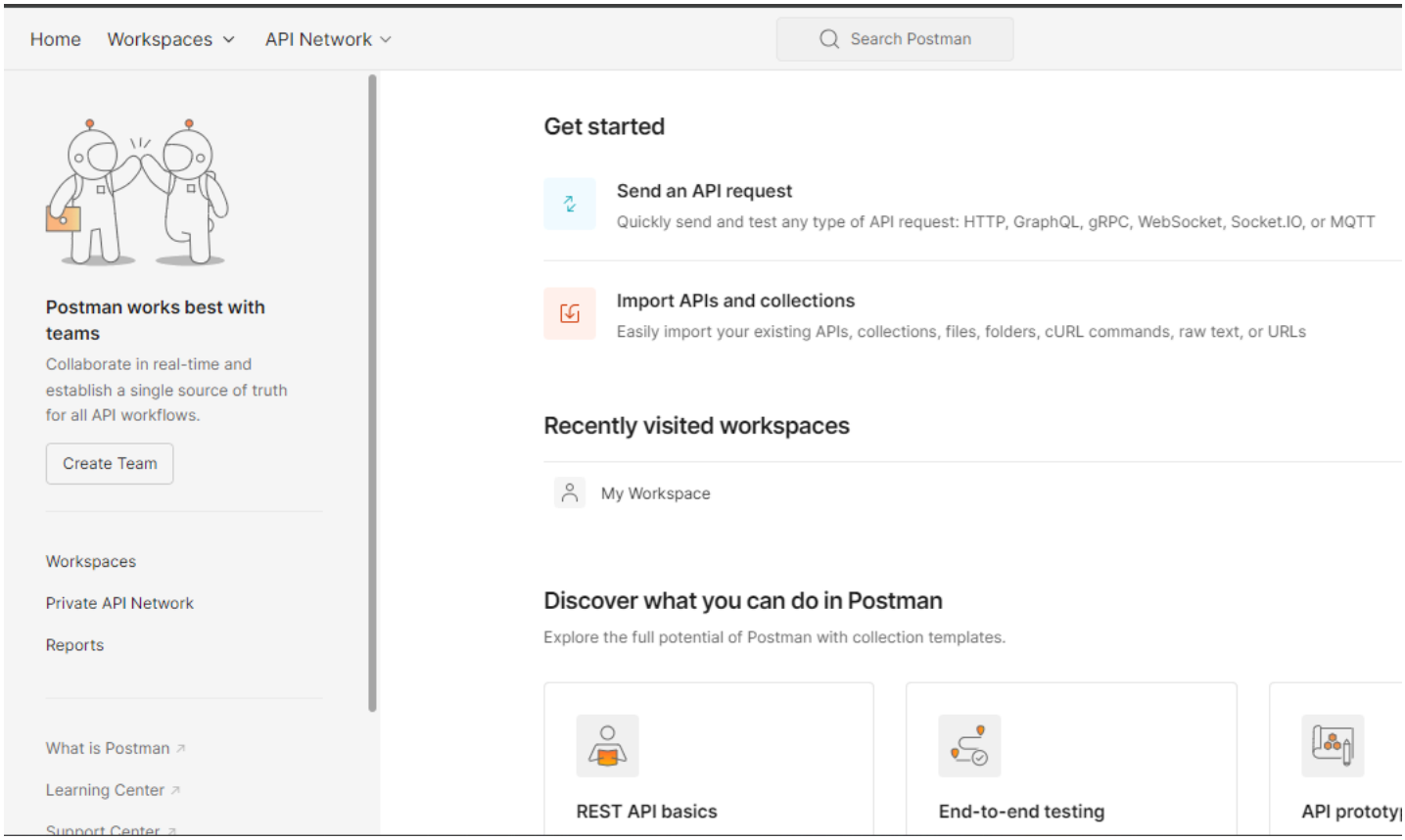
练习 8：使用 POSTMAN 测试端点

您已经使用 cURL 测试了 API 端点。使用图形用户界面工具（GUI）Postman 测试这些端点是一种更简单、更用户友好的方法。

1. 前往 Postman。如果您还没有 Postman 账户，请注册一个新账户。登录到您的账户。



2. 登录 Postman 后，点击下方的 **新建请求**：



注意：如果服务器在 theia 实验室中运行，请按 **CTRL + C** 停止服务器。现在通过运行以下命令启动服务器，该命令将监听 5000 端口。

```
npm run start_auth
```

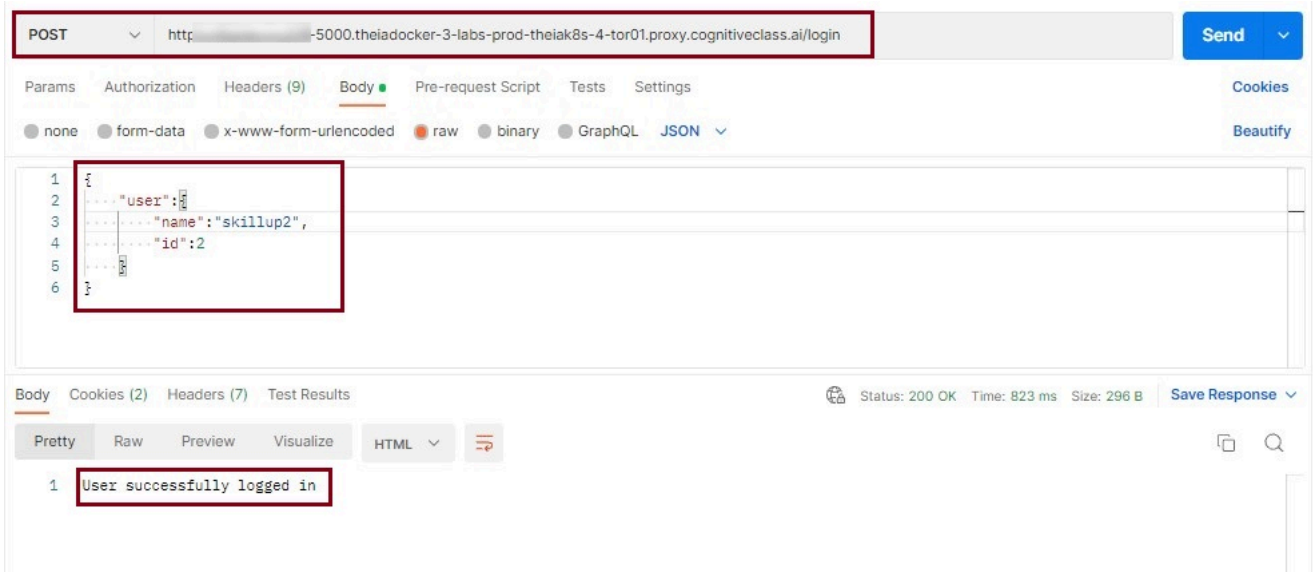
到目前为止，我们访问所有端点时都没有进行身份验证，但现在我们将使用身份验证来访问这些端点。

3. 从启动应用程序中复制 URL，并将登录作为端点添加到 **POST REQUEST** 中，格式如下：

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

4. 用户详细信息应如下格式：

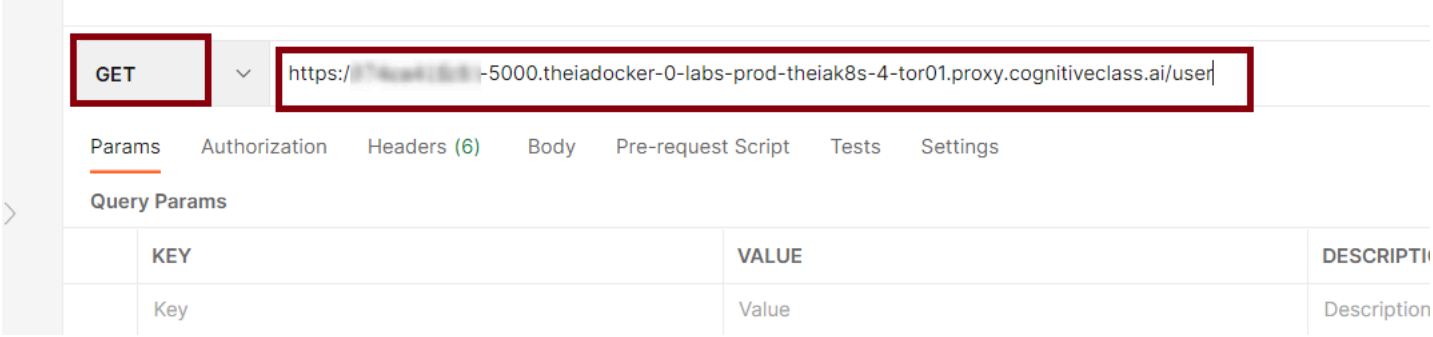
```
{
    "user":{
        "name":"abc",
        "id":1
    }
}
```



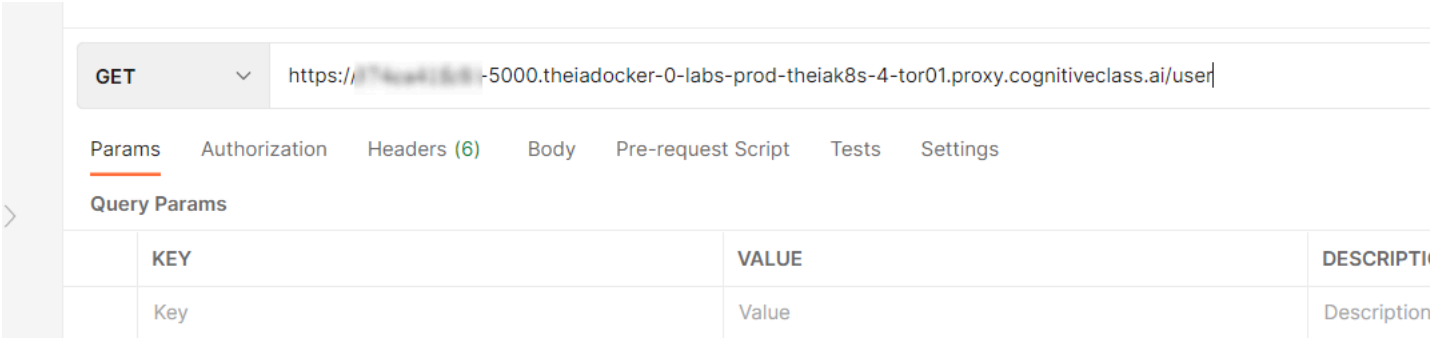
现在让我们通过发送一个HTTP GET请求来开始测试。

8.1 GET 请求

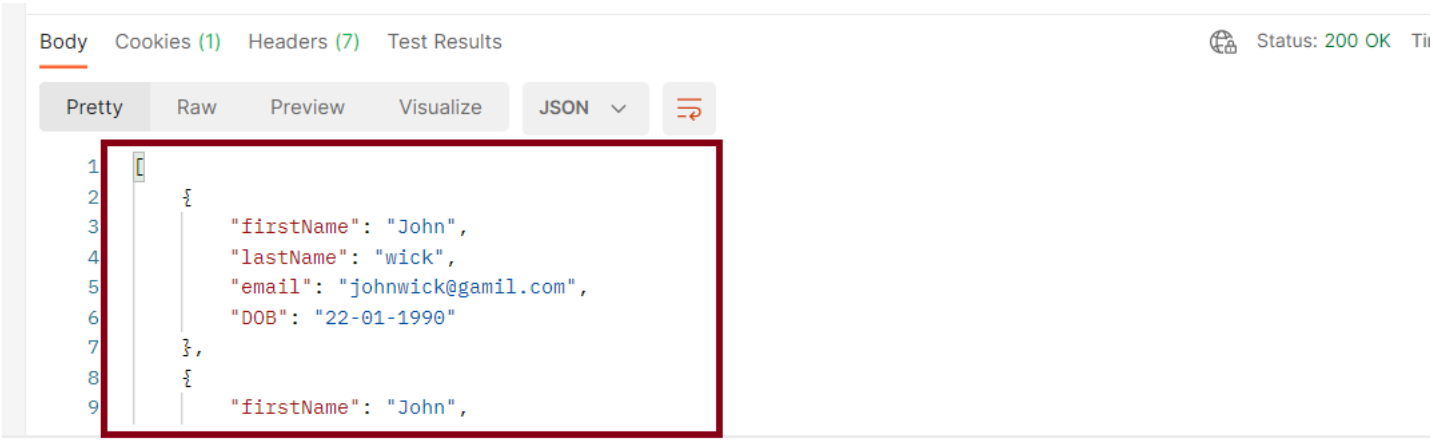
a. 在 Postman 的输入框中输入 GET 请求 URL: `https://XXXXXXXXXX-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user`，在你看到的“输入请求 URL”处。



b. 输入 URL 后，点击 发送 按钮。



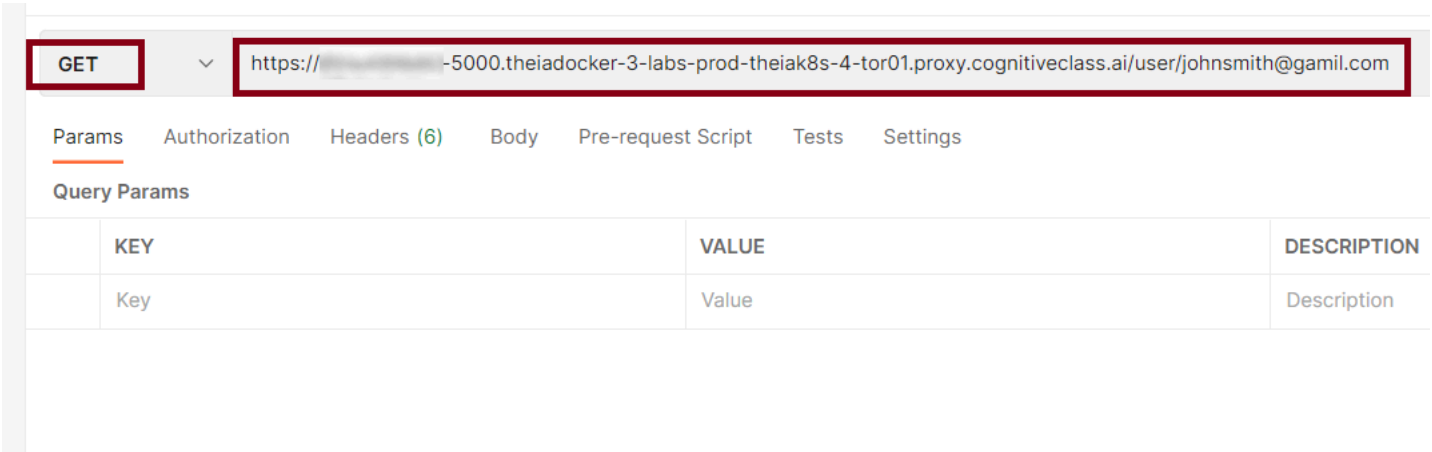
c. 输出结果如下：



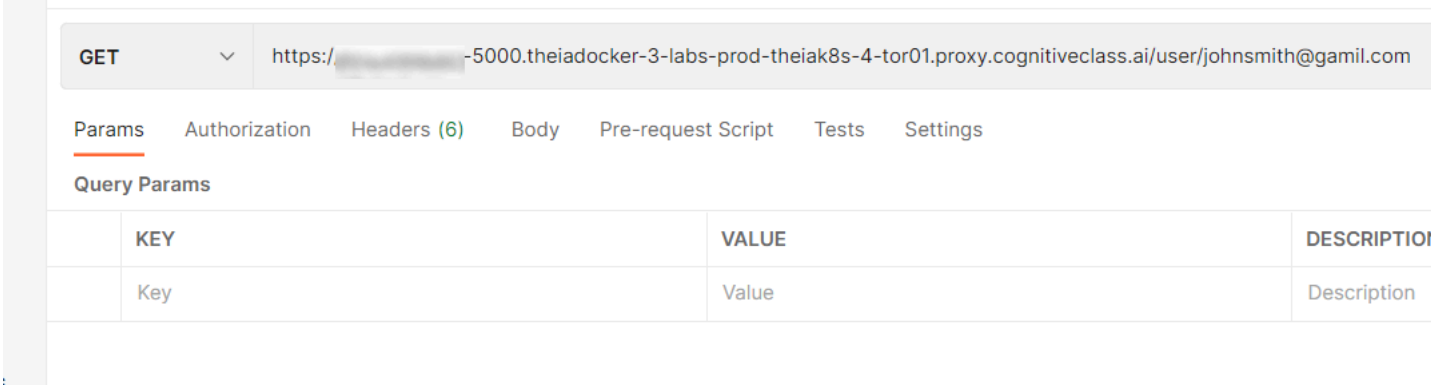
8.2 通过特定 ID 的 GET 请求

a. 在上述 GET 请求 URL 中添加特定的电子邮件地址以输入请求 URL。如果电子邮件地址是 johnsmith@gamil.com，则在 Postman 的输入框中输入以下 URL：

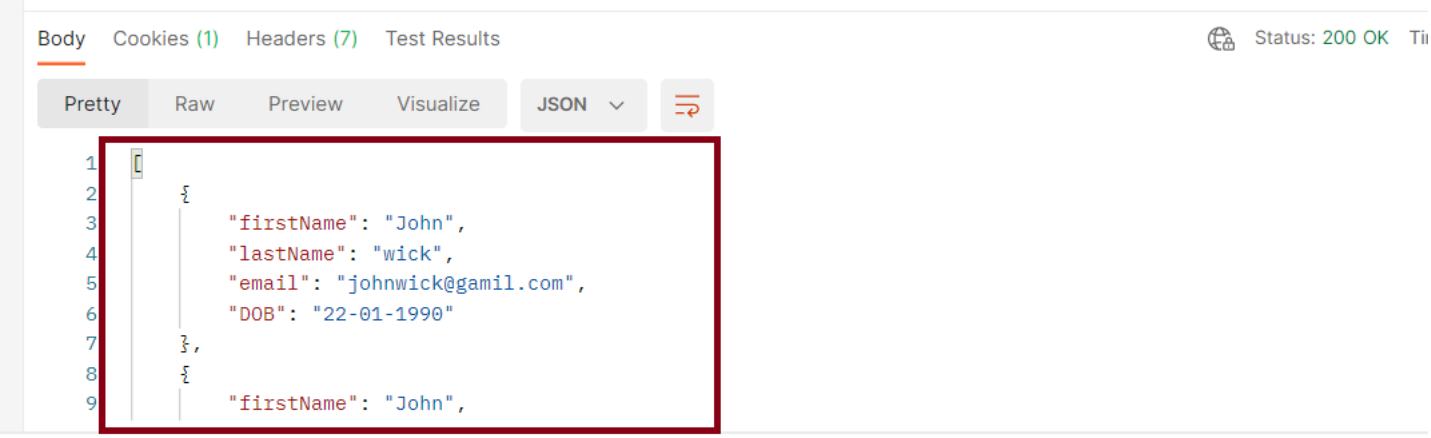
```
https://XXXXXXXXXX-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user/johnsmith@gamil.com
```



b. 输入URL后点击**发送**按钮以查看输出。



c. 输出将如下所示：

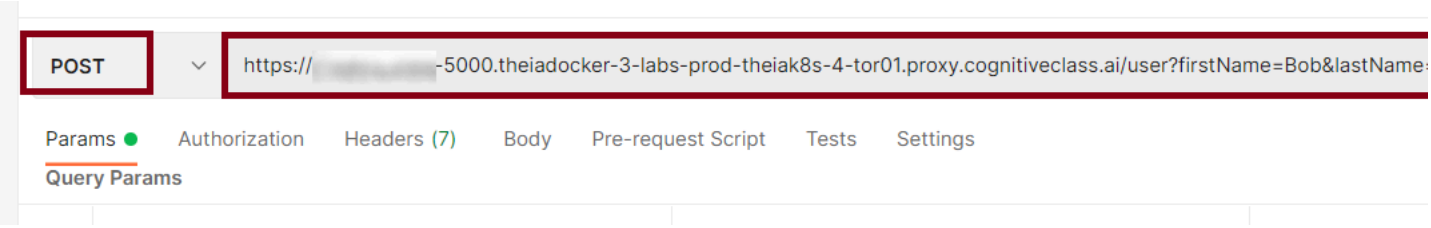


8.3 POST 请求：

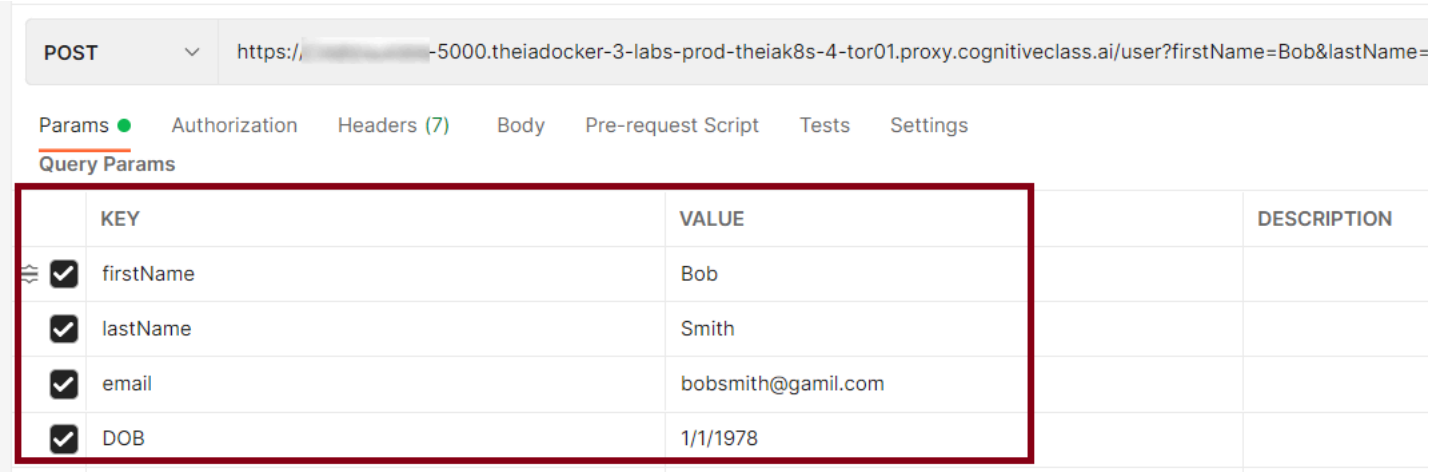
a. 输入基本的 POST 请求 URL:

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

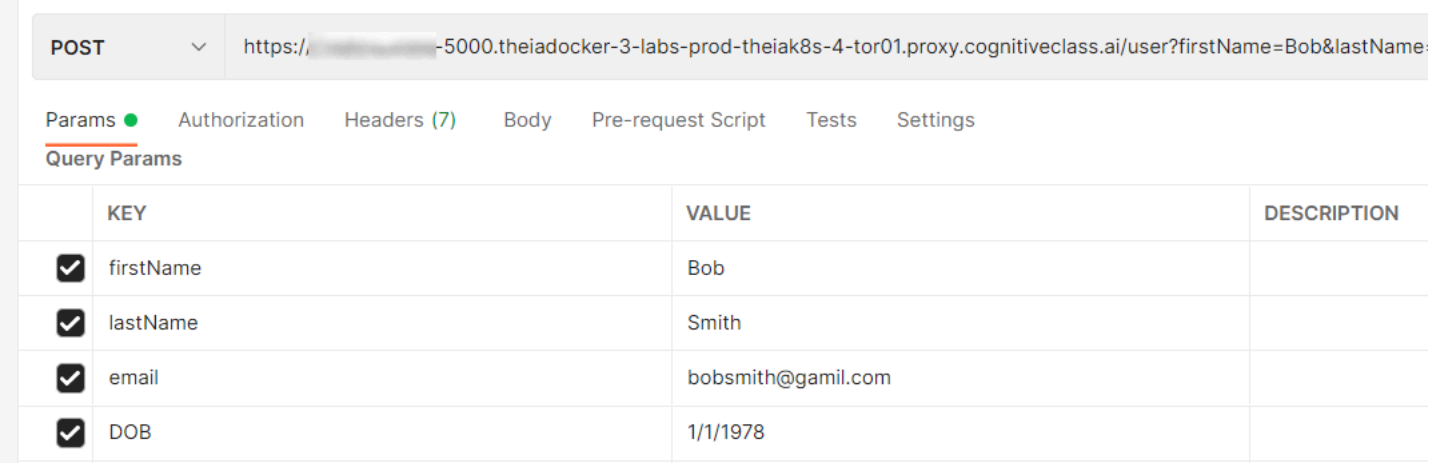
Ensure to select the POST method and select the "Params".



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

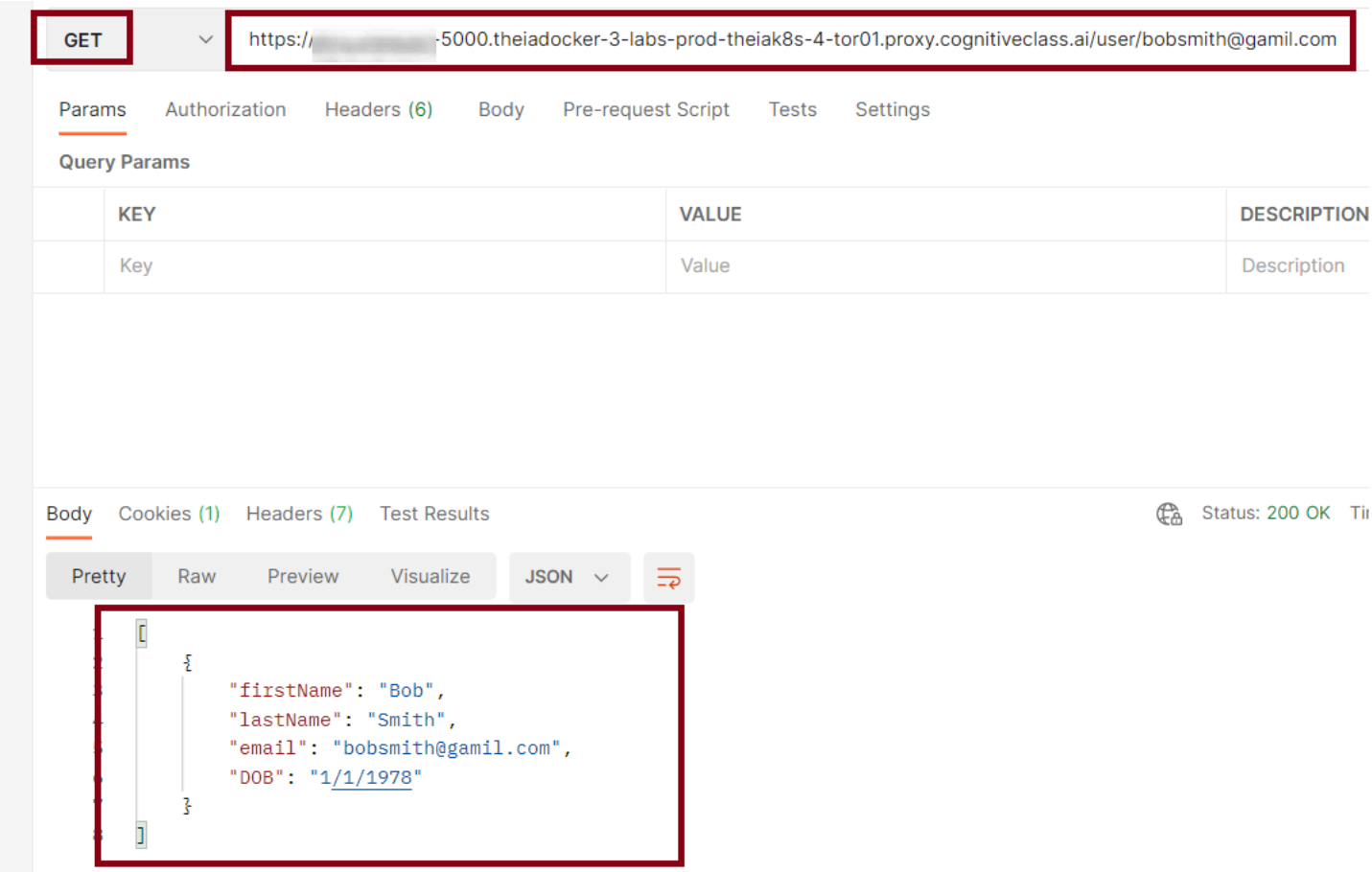


c. Click on the **Send** button after entering the URL to view the output.



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.

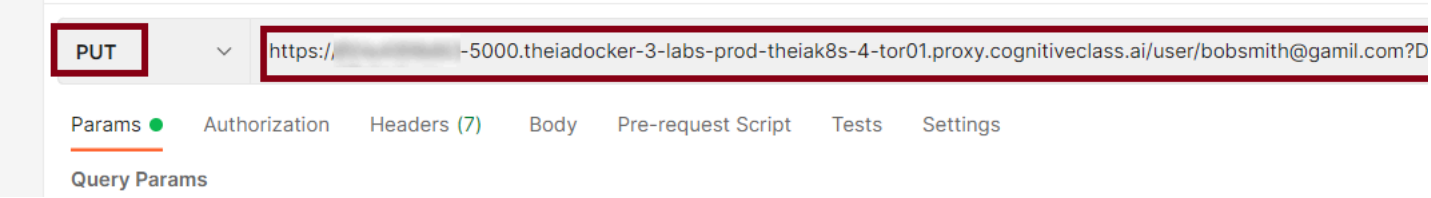


8.4 PUT request

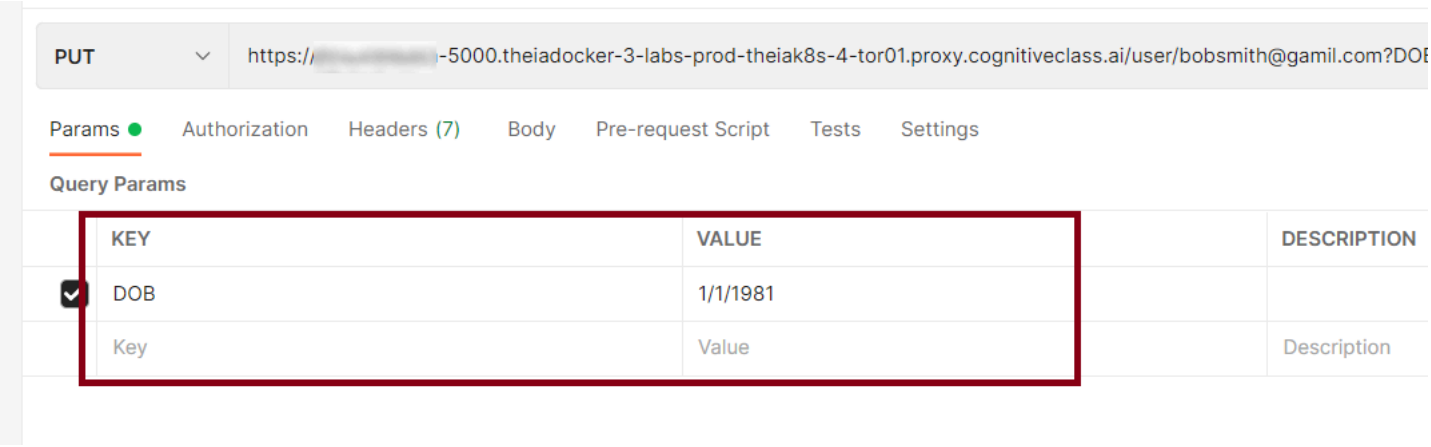
a. Enter the URL by adding the specific email address. If the email address is bobsmith@gamil.com then enter this URL in the input box of the Postman:

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

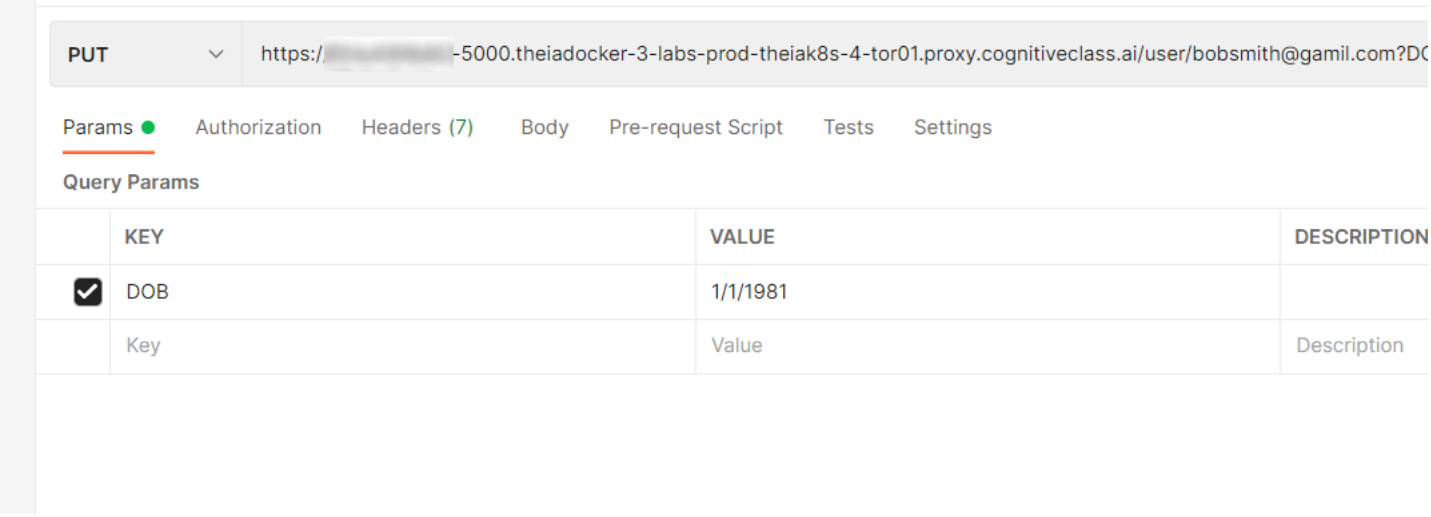
确保选择 PUT 方法并选择“参数”。



b. 输入要更改的键和值。例如，如果您想更改“DOB”键并将其替换为新值 1/1/1981，格式如下。

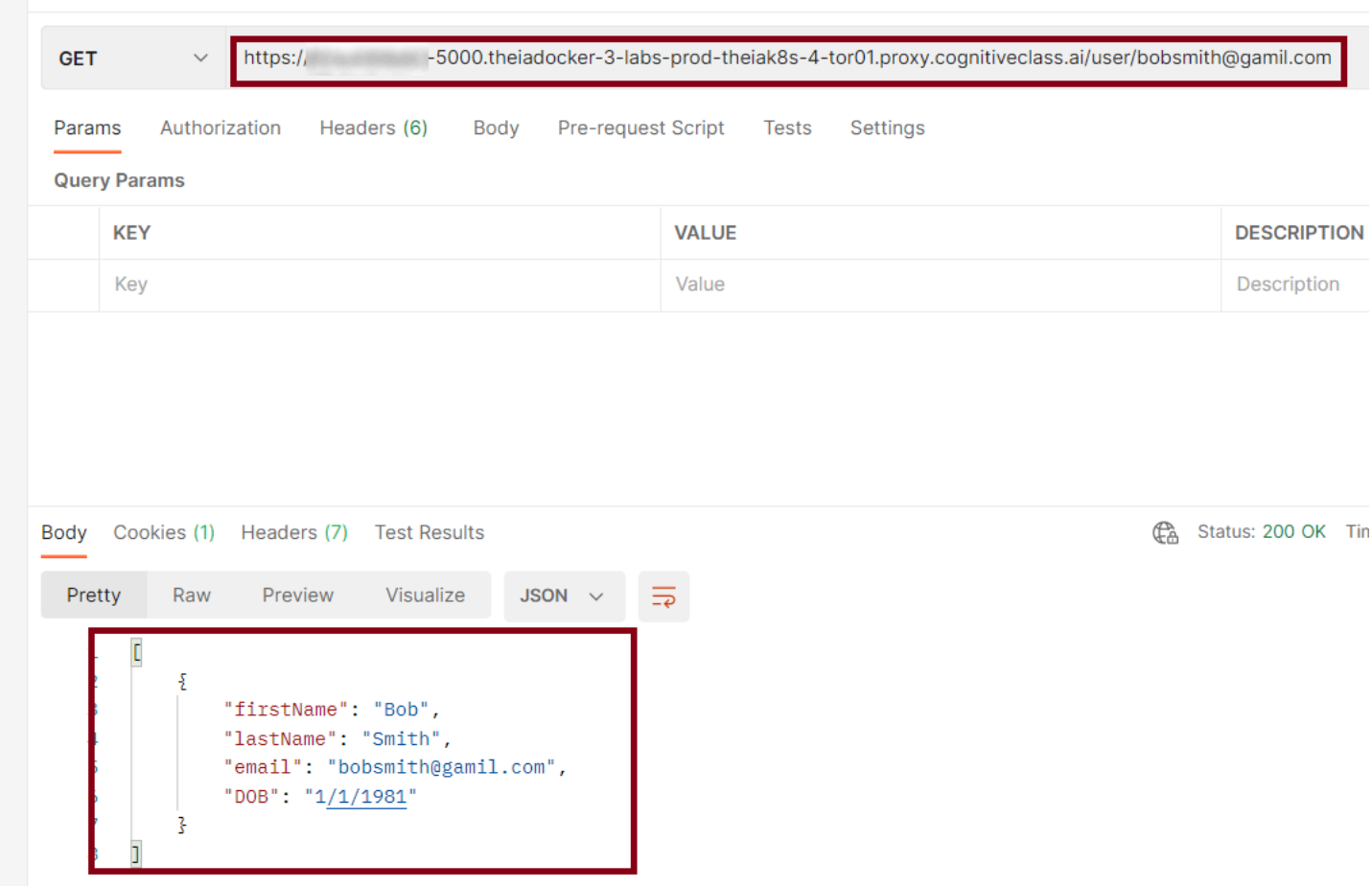


c. 输入 URL 后，点击 **发送** 按钮以查看输出。



通过执行 GET 请求验证新添加的值是否已更新。

注意：在发送 GET 请求之前，请确保删除您为 PUT 请求添加的任何参数。

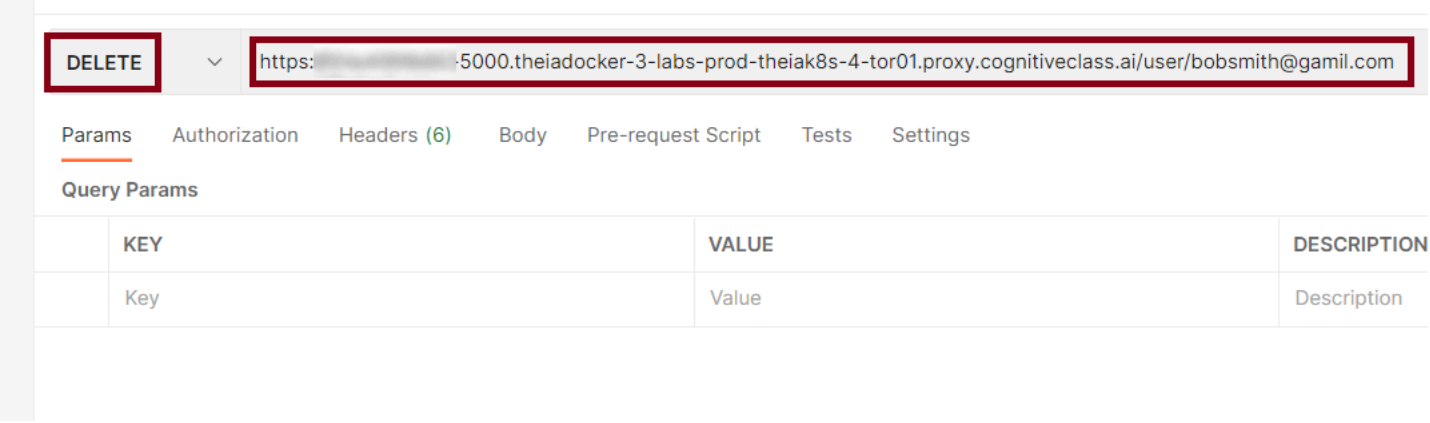


8.5 删除请求：

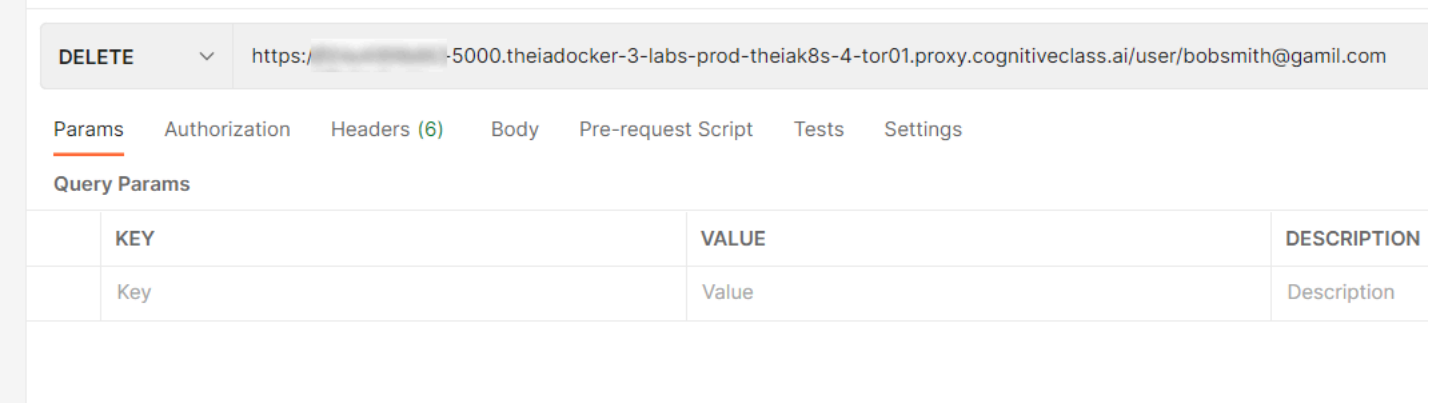
a. 通过添加特定的电子邮件地址输入 URL。如果电子邮件地址是 bobsmith@gamil.com，则在 Postman 的输入框中输入此 URL：

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

确保选择 DELETE 方法。

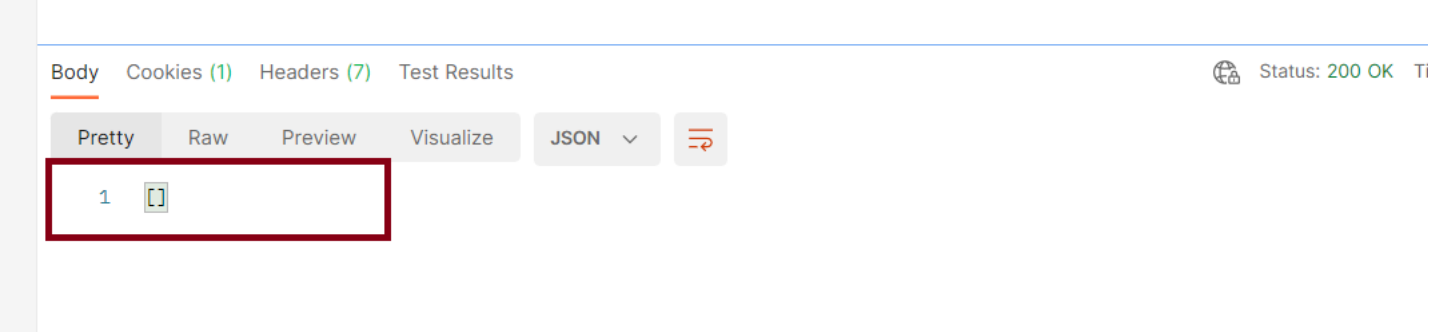


b. 输入 URL 后，点击“发送”按钮以查看输出。



c. 通过发送 GET 请求，验证 ID 为 bobsmith@gamil.com 的用户返回一个空对象。

注意：确保在发送 GET 请求之前删除任何参数（如果有的话）。



实践实验

1. 在相同的代码中创建一个端点，以获取具有特定姓氏的所有用户。

▼ 点击这里获取提示！
提示：从 `users` 数组中过滤 `lastName`。
▼ 点击这里查看解决方案！

解决方案：

```
router.get("/lastName/:lastName", (req, res) => {
    // Extract the lastName parameter from the request URL
    const lastName = req.params.lastName;
    // Filter the users array to find users whose lastName matches the extracted lastName parameter
    let filtered_lastname = users.filter((user) => user.lastName === lastName);
    // Send the filtered_lastname array as the response to the client
    res.send(filtered_lastname);
});
```

2. 在相同的代码中创建一个端点，以按出生日期对用户进行排序。

▼ 点击这里获取提示！
提示：拆分出生日期并转换为 yyyy/mm/dd 格式，然后进行排序。
▼ 点击这里查看解决方案！
解决方案：

```
// Function to convert a date string in the format "dd-mm-yyyy" to a Date object
function getDateFromString(strDate) {
    let [dd, mm, yyyy] = strDate.split('-');
    return new Date(yyyy + "/" + mm + "/" + dd);
}
// Define a route handler for GET requests to the "/sort" endpoint
router.get("/sort", (req, res) => {
    // Sort the users array by DOB in ascending order
    let sorted_users = users.sort(function(a, b) {
        let d1 = getDateFromString(a.DOB);
        let d2 = getDateFromString(b.DOB);
        return d1 - d2;
    });
    // Send the sorted_users array as the response to the client
    res.send(sorted_users);
});
```

恭喜！您已完成使用 Postman 进行 Node.js 和 Express.js 的 CRUD 操作实验。

摘要：

在本实验中，我们在一个 Express 应用程序上执行了 CRUD 操作，如 GET、POST、PUT 和 DELETE，并使用 Postman 测试了上述方法。

Author(s)

Sapthashree K S

K Sundararajan

© IBM Corporation. 保留所有权利。