Praktisches Labor - CRUD-Operationen mit Node.js

Geschätzte benötigte Zeit: 1 Stunde

In diesem Labor lernen Sie, wie Sie eine **Freundesliste** mit einem Express-Server erstellen. Ihre Anwendung sollte es Ihnen ermöglichen, einen Freund mit den folgenden Angaben hinzuzufügen: `Vorname`, `Nachname`, `E-Mail` und `Geburtsdatum`. Sie werden der Anwendung auch die Möglichkeit geben, Details abzurufen, zu ändern und zu löschen.

Sie werden eine Anwendung mit API-Endpunkten erstellen, um die oben genannten Daten mit einem Express-Server zu erstellen, abzurufen, zu aktualisieren und zu löschen.

Außerdem lernen Sie, wie Sie authentifizierten Zugriff auf die Endpunkte bereitstellen. Sie werden cURL und Postman verwenden, um die implementierten Endpunkte zu testen.

Ziele:

- Erstellen Sie API-Endpunkte, um Create-, Retrieve-, Update- und Delete-Operationen auf flüchtigen Daten mit einem Express-Server durchzuführen.
- Implementieren Sie die Authentifizierung auf Sitzungsebene mithilfe von JSON Web Tokens (JWT) für autorisierten Zugriff.

Einrichten : Anwendung erstellen

1. Öffnen Sie ein Terminalfenster über das Menü im Editor: Terminal > Neues Terminal.

2. Wechseln Sie in Ihren Projektordner, falls Sie sich noch nicht im Projektordner befinden.

   ```
   cd /home/project
   ```

3. Führen Sie den folgenden Befehl aus, um das Git-Repository zu klonen, das den Starter-Code für dieses Labor enthält, falls es noch nicht existiert.

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

5. Wechseln Sie in das Verzeichnis **mxpfu-nodejsLabs**, um mit der Laborarbeit zu beginnen.

   ```
   cd mxpfu-nodejsLabs/
   ```

6. Listen Sie den Inhalt dieses Verzeichnisses auf, um die Artefakte für dieses Labor zu sehen.

   ```
   ls
   ```

Übung 1: Verstehen der Serveranwendung

1. Öffnen Sie im Datei-Explorer den Ordner **mxpfu-nodejsLabs** und sehen Sie sich **index.js** an.

Sie haben einen Express-Server, der so konfiguriert ist, dass er auf Port 5000 läuft. Wenn Sie auf den Server mit **/user** zugreifen, können Sie auf die in **routes/users.js** definierten Endpunkte zugreifen.

Erinnern Sie sich daran, dass GET, POST, PUT und DELETE die gängigen HTTP-Methoden sind, um CRUD-Operationen durchzuführen. Diese Operationen rufen Daten ab und senden sie an den Server.

- `GET` wird verwendet, um Daten von einer bestimmten Ressource anzufordern.
- `POST` wird verwendet, um Daten an einen Server zu senden, um eine Ressource zu erstellen.
- `PUT` wird verwendet, um Daten an einen Server zu senden, um eine Ressource zu aktualisieren.
- `DELETE` wird verwendet, um eine bestimmte Ressource zu löschen.

  `POST` UND `PUT` werden manchmal austauschbar verwendet.

2. Dieses Labor erfordert die Installation einiger Pakete. Das **express**- und **nodemon**-Paket zum Starten und Ausführen des Express-Servers sowie **jsonwebtoken** und **express-session** für die sitzungsbasierte Authentifizierung.

- **express** - Dies dient zum Erstellen eines Servers, um die API-Endpunkte bereitzustellen.
- **nodemon** - Dies hilft, den Server neu zu starten, wenn Sie Änderungen am Code vornehmen.
- **jsonwebtoken** - Dieses Paket hilft bei der Generierung eines JSON-Webtokens, das wir für die Authentifizierung verwenden werden. Ein **JSON-Webtoken (JWT)** ist ein JSON-Objekt, das verwendet wird, um Informationen sicher über das Internet (zwischen zwei Parteien) zu kommunizieren. Es kann für den Informationsaustausch verwendet werden und wird typischerweise für Authentifizierungssysteme verwendet.
- **express-session** - Dieses Paket hilft uns, die Authentifizierung für die Sitzung aufrechtzuerhalten.

Diese Pakete sind als **Abhängigkeiten** in packages.json definiert.

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

3. Beachten Sie, dass die Express-App die Middleware **express.json()** verwendet, um die Anfrage als JSON-Objekt zu verarbeiten.

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

4. Beachten Sie, dass die Express-App Routen verwendet, um die Endpunkte zu verwalten, die mit **/user** beginnen. Das bedeutet, dass der Server für alle Endpunkte, die mit **/user** beginnen, nach einem Endpunkt-Handler in **users.js** suchen wird.

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

5. Alle Endpunkte haben eine grundlegende, aber funktionierende Implementierung in **users.js**. Navigiere zu users.js im Verzeichnis routes und beobachte die darin definierten Endpunkte.

Übung 2: Den Server ausführen

Der bereitgestellte Startercode ist ein funktionierender Server mit Platzhalter-Rückgabewerten. Bevor Sie mit der Implementierung der eigentlichen Endpunkte beginnen, führen Sie den Server aus.

1. Drucken Sie im Terminal das Arbeitsverzeichnis aus, um sicherzustellen, dass Sie sich in **/home/projects/mxpfu-nodejsLabs** befinden.

   ```
   pwd
   ```

2. Installieren Sie alle Pakete, die für den Betrieb des Servers erforderlich sind. Kopieren Sie den folgenden Befehl, fügen Sie ihn ein und führen Sie ihn aus.

   ```
   npm install
   ```

   Dies installiert alle erforderlichen Pakete, die in packages.json definiert sind.

3. Starte den Express-Server.

   ```
   npm start
   ```

4. Öffnen Sie ein **Neues Terminal** aus dem oberen Menü. Testen Sie einen Endpunkt, um diese Benutzer abzurufen. Dies wurde noch nicht implementiert, um die Benutzer zurückzugeben.

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

5. Wenn Sie die Ausgabe wie oben angezeigt sehen, bedeutet das, dass der Server wie erwartet läuft.

Übung 2: Implementiere deine Endpunkte

1. Navigiere zur Datei **users.js** im **routes**-Ordner. Die Endpunkte wurden definiert und es wurde Platz für dich geschaffen, um die Endpunkte zu implementieren.

2. **R** in CRUD steht für abrufen. Du wirst zuerst einen API-Endpunkt hinzufügen, der die **get**-Methode verwendet, um die Details aller Benutzer abzurufen. Einige Benutzer wurden im Starter-Code hinzugefügt.

- Kopiere den folgenden Code und füge ihn in users.js innerhalb der { }-Klammern der **router.get("/",(req,res)=>{}**-Methode ein.

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

3. Stellen Sie sicher, dass Ihr Server läuft. Wenn Sie Änderungen am Code vornehmen, sollte der Server, den Sie in der vorherigen Aufgabe gestartet haben, neu gestartet werden. Wenn der Server nicht läuft, starten Sie ihn erneut.

```
npm start
```

3. Klicken Sie auf die Schaltfläche **Skills Network** auf der linken Seite. Es öffnet sich das „Skills Network Toolbox". Klicken Sie dann auf `OTHER` und dann auf `Launch Application`. Von dort aus sollten Sie in der Lage sein, den Port als `5000` einzugeben und den Entwicklungsserver zu starten.

4. Wenn die Browserseite geöffnet wird, fügen Sie **/user** am Ende der URL in der Adressleiste hinzu. Sie werden die folgende Seite sehen.

5. Überprüfen Sie die Ausgabe der GET-Anfrage mit dem curl-Befehl, genau so, wie Sie es in der vorherigen Übung getan haben.

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

Übung 3: Erstellen einer GET-Methode nach spezifischer E-Mail:

1. Implementieren Sie eine **get**-Methode, um die Details eines bestimmten Benutzers basierend auf seiner E-Mail-ID abzurufen, indem Sie die `filter`-Methode auf der Benutzersammlung verwenden. Sobald Sie den Code geschrieben und gespeichert haben, wird der Server neu gestartet.

▶ Klicken Sie hier, um den Code anzuzeigen

2. Klicken Sie auf Terminal > Neues Terminal

3. Verwenden Sie im neuen Terminal den folgenden Befehl, um die Ausgabe für den Benutzer mit der E-Mail-Adresse **johnsmith@gamil.com** anzuzeigen.

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

Übung 4: Erstellen der POST-Methode:

1. Das **C** in CRUD steht für **Create**. Implementiere den **/user** Endpunkt mit der POST-Methode, um einen Benutzer zu erstellen und den Benutzer zur Liste hinzuzufügen. Du kannst das Benutzerobjekt als Dictionary erstellen. Du kannst das unten angezeigte Beispiel-Benutzerobjekt verwenden.

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

Verwende `push`, um das Wörterbuch in die Liste der Benutzer hinzuzufügen. Die Benutzerdetails können als Abfrageparameter mit den Namen *firstName, lastName, DOB und email* übergeben werden.

Hinweis: Abfrageparameter können mit request.query.paramname aus dem Anfrageobjekt abgerufen werden.

▶ Klicken Sie hier, um den Code anzuzeigen

2. Verwenden Sie den folgenden Befehl, um einen neuen Benutzer mit der E-Mail-Adresse 'jonlovato@theworld.com' im neuen Terminal zu erstellen:

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

3. Die Ausgabe wird wie folgt sein:

4. Um zu überprüfen, ob der Benutzer mit der E-Mail ‘jonlovato@theworld.com‘ hinzugefügt wurde, können Sie eine GET-Anfrage wie folgt senden:

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

Übung 5: Erstellen der PUT-Methode:

1. Das **U** in CRUD steht für Aktualisierung, die mit der PUT-Methode erreicht werden kann. Um Daten zu aktualisieren, verwenden Sie die PUT-Methode. Zuerst sollten Sie den Benutzer mit der angegebenen E-Mail-Adresse ansehen und dann ihn ändern. Der folgende Code

zeigt, wie das Geburtsdatum (DOB) eines Benutzers geändert werden kann. Nehmen Sie die erforderlichen Codeänderungen vor, um Änderungen an den anderen Attributen des Benutzers zu ermöglichen.

```
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. Der vollständige Code sieht folgendermaßen aus.

3. Verwenden Sie den folgenden Befehl, um das `DOB` auf `1/1/1971` für den Benutzer mit der E-Mail-Adresse 'johnsmith@gamil.com' im geteilten Terminal zu aktualisieren:

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

4. Die Ausgabe wird wie folgt aussehen:

5. Um zu überprüfen, ob das `DOB` des Benutzers mit der E-Mail 'johnsmith@gamil.com' aktualisiert wurde, können Sie eine GET-Anfrage wie folgt senden:

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

Übung 6: Erstellen der DELETE-Methode:

1. Das "D" in CRUD steht für **Löschen**. Implementieren Sie die DELETE-Methode zum Löschen der E-Mail eines bestimmten Benutzers, indem Sie den folgenden Code verwenden:

```
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. Der vollständige Code sieht so aus.

3. Verwenden Sie den folgenden Befehl, um den Benutzer mit der E-Mail-Adresse 'johnsmith@gamil.com' im geteilten Terminal zu löschen:

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

4. Die Ausgabe wird wie folgt aussehen:

5. Senden Sie eine GET-Anfrage für den Benutzer mit der E-Mail ‘johnsmith@gamil.com‘ und stellen Sie sicher, dass ein null-Objekt zurückgegeben wird:

Optionale Übung: Ausgabe formatieren

1. Um die Ausgabe lesbarer zu machen, können Sie die JSON-Stringify-Methode wie unten angegeben verwenden. Bitte aktualisieren Sie den Code für die GET-Methode auf:

```
// 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. Starten Sie die App auf Port 5000 und fügen Sie 'user' am Ende der URL hinzu.

3. Dies wird die Ausgabe der GET-Methode als JSON-String gemäß der unten gezeigten aktualisierten GET-Methode rendern:

Übung 7: Implementierung der Authentifizierung

Alle diese Endpunkte sind für jeden zugänglich. Sie werden nun sehen, wie man Authentifizierung zu den CRUD-Operationen hinzufügt. Dieser Code wurde in **index_withauth.js** implementiert.

1. Beachten Sie den folgenden Codeblock in **index_withauth.js**.

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

Das weist Ihre Express-Anwendung an, die Session-Middleware zu verwenden.

- **secret** - ein zufälliger, eindeutiger Schlüssel, der zur Authentifizierung einer Sitzung verwendet wird.
- **resave** - nimmt einen booleschen Wert an. Es ermöglicht, dass die Sitzung zurück in den Sitzungsspeicher gespeichert wird, selbst wenn die Sitzung während der Anfrage nie modifiziert wurde.
- **saveUninitialized** - dies erlaubt, dass jede nicht initialisierte Sitzung an den Speicher gesendet wird. Wenn eine Sitzung erstellt, aber nicht modifiziert wird, wird sie als **nicht initialisiert** bezeichnet.

Der Standardwert sowohl für **resave** als auch für **saveUninitialized** ist true, aber der Standardwert ist veraltet. Setzen Sie daher den entsprechenden Wert je nach Anwendungsfall.

2. Beobachten Sie die Implementierung des **login**-Endpunkts. Ein Benutzer meldet sich im System mit einem Benutzernamen an. Ein Zugriffstoken, das eine Stunde gültig ist, wird generiert. Sie können diese Gültigkeitsdauer, die durch **60 * 60** angegeben ist, beobachten, was die Zeit in Sekunden bedeutet. Dieses Zugriffstoken wird im Sitzungsobjekt gespeichert, um sicherzustellen, dass nur authentifizierte Benutzer für diesen Zeitraum auf die Endpunkte zugreifen können.

```
// 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. Beobachten Sie die Implementierung der Authentifizierungs-Middleware. Alle Endpunkte, die mit **/user** beginnen, werden durch diese Middleware geleitet. Sie ruft die Berechtigungsdetails aus der Sitzung ab und überprüft sie. Wenn das Token validiert wird, ist der Benutzer authentifiziert und die Kontrolle wird an den nächsten Endpunkt-Handler übergeben. Wenn das Token ungültig ist, ist der Benutzer nicht authentifiziert und eine Fehlermeldung wird zurückgegeben.

```
// 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" });
    }
});
```

Übung 8: Testen von Endpunkten mit POSTMAN

Sie haben die API-Endpunkte mit cURL getestet. Eine einfachere und benutzerfreundlichere Möglichkeit, diese Endpunkte mit dem grafischen Benutzeroberflächen-Tool (GUI) Postman zu testen.

1. Gehen Sie zu Postman. Registrieren Sie sich für ein neues Postman-Konto, falls Sie noch keines haben. Melden Sie sich bei Ihrem Konto an.

2. Nachdem Sie sich bei Postman angemeldet haben, klicken Sie auf **Neue Anfrage** wie unten gezeigt:

   Hinweis: Wenn der Server im Theia-Labor läuft, stoppen Sie den Server bitte mit **CTRL + C**. Starten Sie jetzt den Server, indem Sie den folgenden Befehl ausführen, der auf Port 5000 hören wird.

   ```
   npm run start_auth
   ```

   Bis jetzt haben wir auf alle Endpunkte ohne Authentifizierung zugegriffen, aber jetzt werden wir eine Authentifizierung verwenden, um auf die Endpunkte zuzugreifen.

3. Kopiere die URL aus der Anwendung und füge den Login als Endpunkt hinzu, um die Benutzerdaten in der **POST-AnFRAGE** hinzuzufügen, die wie folgt aussehen wird:

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

4. Die Benutzerdaten sollten im folgenden Format vorliegen:

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

Jetzt lassen Sie uns den Test beginnen, indem wir eine HTTP GET-Anfrage senden.

8.1 GET-Anfrage

a. Geben Sie die GET-Anfrage-URL: `https://XXXXXXXXXX-5000.theiadocker-0-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/user` in das Eingabefeld von Postman ein, wo Sie „Enter Request URL“ sehen.

b. Klicken Sie auf die Schaltfläche `Send`, nachdem Sie die URL eingegeben haben.

c. Die Ausgabe wird wie folgt aussehen:

8.2 GET-Anfrage nach spezifischer ID

a. Geben Sie die Anforderungs-URL ein, indem Sie die spezifische E-Mail-Adresse zur obigen GET-Anforderungs-URL hinzufügen. Wenn die E-Mail-Adresse johnsmith@gamil.com ist, geben Sie die folgende URL in das Eingabefeld von Postman ein:

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

b. Klicken Sie auf die **Send**-Schaltfläche, nachdem Sie die URL eingegeben haben, um die Ausgabe anzuzeigen.

c. Die Ausgabe wird wie folgt aussehen:

8.3 POST-Anfrage :

a. Geben Sie die grundlegende POST-Anfrage-URL ein:

```
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
```

Stellen Sie sicher, dass Sie die PUT-Methode auswählen und die „Params“ auswählen.

b. Geben Sie den Schlüssel und die Werte ein, die geändert werden sollen. Wenn Sie beispielsweise den Schlüssel „DOB“ ändern und ihn durch den neuen Wert 1/1/1981 ersetzen möchten, sieht es wie folgt aus.

c. Klicken Sie auf die **Send**-Schaltfläche, nachdem Sie die URL eingegeben haben, um die Ausgabe anzuzeigen.

Überprüfen Sie, ob die neu hinzugefügten Werte aktualisiert wurden, indem Sie eine GET-Anfrage durchführen.

> Hinweis: Stellen Sie sicher, dass Sie alle Parameter, die Sie für die PUT-Anfrage hinzugefügt haben, vor dem Senden der GET-Anfrage löschen.

8.5 DELETE-Anfrage:

a. Geben Sie die URL ein, indem Sie die spezifische E-Mail-Adresse hinzufügen. Wenn die E-Mail-Adresse bobsmith@gamil.com ist, geben Sie diese URL in das Eingabefeld von Postman ein:

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

Stellen Sie sicher, dass Sie die DELETE-Methode auswählen.

b. Klicken Sie auf die Schaltfläche „Send“, nachdem Sie die URL eingegeben haben, um die Ausgabe anzuzeigen.

c. Überprüfen Sie, ob die GET-Anfrage für den Benutzer mit der ID bobsmith@gamil.com ein Null-Objekt zurückgibt, indem Sie eine GET-Anfrage senden.

> Hinweis: Stellen Sie sicher, dass Sie alle Parameter (sofern vorhanden) vor dem Senden der GET-Anfrage löschen.

Übungslabore

1. Erstellen Sie einen Endpunkt im selben Code, um alle Benutzer mit einem bestimmten Nachnamen abzurufen.

▶ Klicken Sie hier für einen Hinweis!
▶ Klicken Sie hier für die Lösung!

2. Erstellen Sie einen Endpunkt im selben Code, um Benutzer nach Geburtsdatum zu sortieren.

▶ Klicken Sie hier für einen Hinweis!
▶ Klicken Sie hier für die Lösung!

Herzlichen Glückwunsch! Sie haben das Labor für CRUD-Operationen mit Node.js und Express.js unter Verwendung von Postman abgeschlossen.

Zusammenfassung:

In diesem Labor haben wir CRUD-Operationen wie GET, POST, PUT und DELETE in einer Express-App durchgeführt und die oben genannten Methoden mit Postman getestet.

Author(s)

Sapthashree K S

K Sundararajan