

1. Write the Login.html which takes 2 inputs as user id and password. At the express side, validate it against your name for user id and password as admin. If this criterion matches then return as valid user else returns as an invalid user.

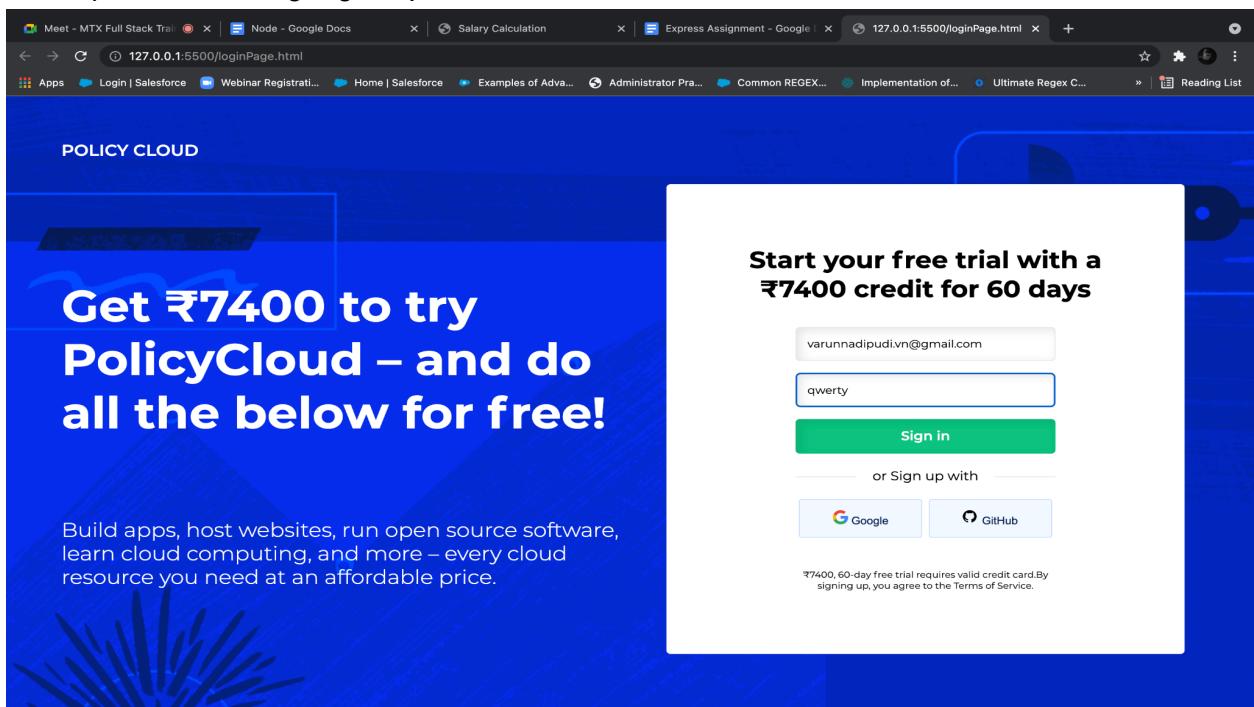
The Login.js file contains the method where the request '<http://localhost:8000/login>' is answered.

The screenshot shows a VS Code interface with the following details:

- Explorer View:** Shows files in the project: `login.js`, `loginPage.html`, `# style.css`, `node_modules`, `EXPRESS` folder containing `login.js`, `loginBackground.png`, `loginPage.html`, `package-lock.json`, `package.json`, `salaryCalc.js`, `salaryCalculation.html`, `# salaryCalculationStyle.css`, and `# style.css`.
- Code Editor:** The `login.js` file is open, displaying code for a Node.js Express application that handles a POST request to `/login`. It logs "Hello from express!", checks email and password, and returns "Invalid User" or "Valid user!" based on the input.
- Terminal:** The terminal window shows the output of running `node login.js` using `nodemon`. It starts with version `2.0.15`, watches paths, and then starts the application with `node login.js`. It then logs that the server is running at port `8000` multiple times.

Browser login details

(Since we are using the post method the browser doesn't support it, so using Postman to show the output after clicking Sign in)



Postman output

The screenshot shows the Postman interface with a collection named "Express Test1". A POST request is made to `http://localhost:8000/login`. The request body is set to JSON and contains:

```

1  {
2     "emailId": "varunnadipudi.vn@gmail.com",
3     "password": "qwerty"
4 }

```

The response status is 200 OK, and the body contains the message: "Valid user!".

2. Write an express script to take the URL as '`/getAllEmployeeData`' and when the browser gives the URL '<http://localhost:8000/getAllEmployeeData>' an array of Employee objects should be returned from express and it should be displayed on the browser.
Employee object should have the fields: Id, name, dept, and designation.

This is the method in login.js file which will answer the '`/getAllEmployeeData`' request.

```

30 var emp = [
31   {
32     "id": 102,
33     "name": "Varun",
34     "departement": "IT",
35     "designation": "Fullstack Developer"
36   },
37   {
38     "id": 103,
39     "name": "Vishal",
40     "departement": "IT",
41     "designation": "UI/UX"
42   },
43   {
44     "id": 104,
45     "name": "Pooh",
46     "departement": "IT",
47     "designation": "SDE"
48   },
49   {
50     "id": 105,
51     "name": "Mok",
52     "departement": "HR",
53     "designation": "Senior HR"
54   }
55 ];
56
57 app.get("/getAllEmployeeData", (req, res) => {
58   res.send(emp);
59 });

```

The terminal shows the command `node login` being run, and the server is running at port 8000.

Postman output

The screenshot shows the Postman interface with a collection named "Express Test1". A GET request is made to `http://localhost:8000/getAllEmployeeData`. The response status is 200 OK, time is 12 ms, and size is 520 B. The response body is a JSON array of four employees:

```
1 | {
2 |   "id": 102,
3 |   "name": "Varun",
4 |   "department": "IT",
5 |   "designation": "Fullstack Developer"
6 | },
7 | {
8 |   "id": 103,
9 |   "name": "Vishal",
10 |  "department": "IT",
11 |  "designation": "UI/UX"
12 | },
13 | {
14 |   "id": 104,
15 |   "name": "Pooh",
16 |   "department": "IT",
17 |   "designation": "SDE"
18 | },
19 | {
20 |   "id": 105,
21 |   "name": "Mok",
22 |   "department": "HR",
23 |   "designation": "Senior HR"
24 | }
```

3. Create a post method in express which takes these parameters Basic, HRA, DA, IT, and PF and this function calculates the total salary as $\text{total_salary} = \text{Basic} + \text{HRA} + \text{DA} - (\text{IT} + \text{PF})$ and displays the total_salary as the result of this function.

The salaryCalc.js method answers the '<http://localhost:8000/totalSalary>' request

```
1 const express = require('express');
2 const app = express();
3 const PORT = 8000;
4
5 app.use(express.json());
6
7 app.post('/totalSalary', (req,res) =>{
8
9   var basic = req.body.basic;
10  var hra = req.body.hra;
11  var da = req.body.da;
12  var it = req.body.it;
13  var pf = req.body.pf;
14
15  var totalSalary = basic + hra + da - (it + pf);
16  console.log(`Total salary is : ${totalSalary}`);
17  res.send(`Total salary is : ${totalSalary}`);
18 });
19
20 app.listen(PORT, ()=>{
21   console.log(`Server is running at port : ${PORT}`);
22 });

varunnadipudi@TXs-MacBook-Pro:~/Express % nodemon salaryCalc
[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 salaryCalc.js`
Server is running at port : 8000
[nodemon] restarting due to changes...
[nodemon] starting `node salaryCalc.js`
Server is running at port : 8000
Total salary is : 320
```

Browser input details

(Since we are using the post method the browser doesn't support it, so using Postman to show the output after clicking calculate)

The screenshot shows a web browser window with multiple tabs open. The active tab displays a "Salary Calculator" form. The form consists of five input fields and a "Calculate" button. The fields are labeled: "Basic" (value: 200), "HRA" (value: 100), "Direct Allowances" (value: 50), "Income Tax" (value: 20), and "Provident Fund" (value: 10). The "Calculate" button is located at the bottom of the form.

Postman Output

The screenshot shows the Postman application interface. On the left, there is a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A collection named "Express Test1" is selected. In the main workspace, there is a list of requests under the "http://localhost:8000/totalSalary" endpoint. One request is highlighted as a POST method. The "Body" tab is selected, showing a JSON payload:

```
1  {
2   "basic" : 200,
3   "hra" : 100,
4   "da" : 50,
5   "it" : 20,
6   "pf" : 10
7 }
```

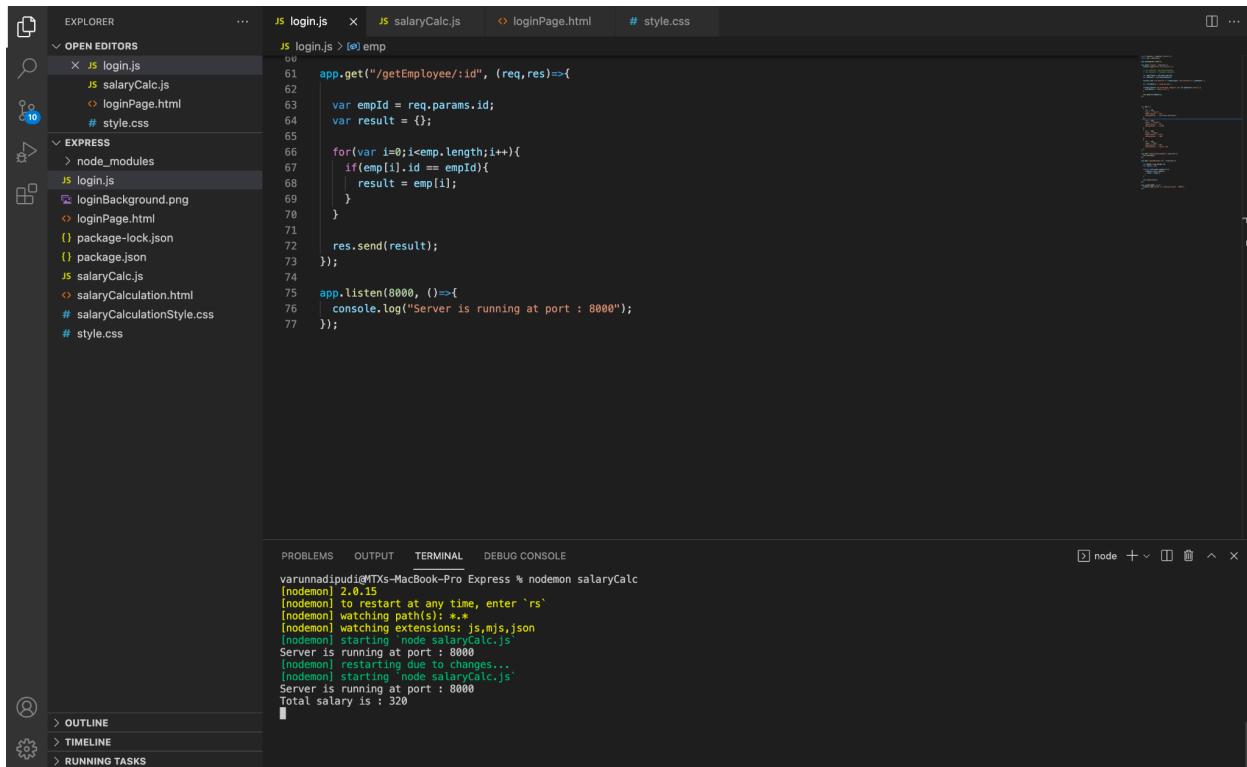
Below the body, the "Test Results" section shows the response: "Status: 200 OK Time: 36 ms Size: 249 B". The response body contains the message "Total salary is : 320".

4. Create an array of objects of employees with the fields id, name, dept, and designation and create below-given functions in the Express and perform the actions as specified.

a. get - getAllEmployees() -- Returns all the employees of the array of objects.

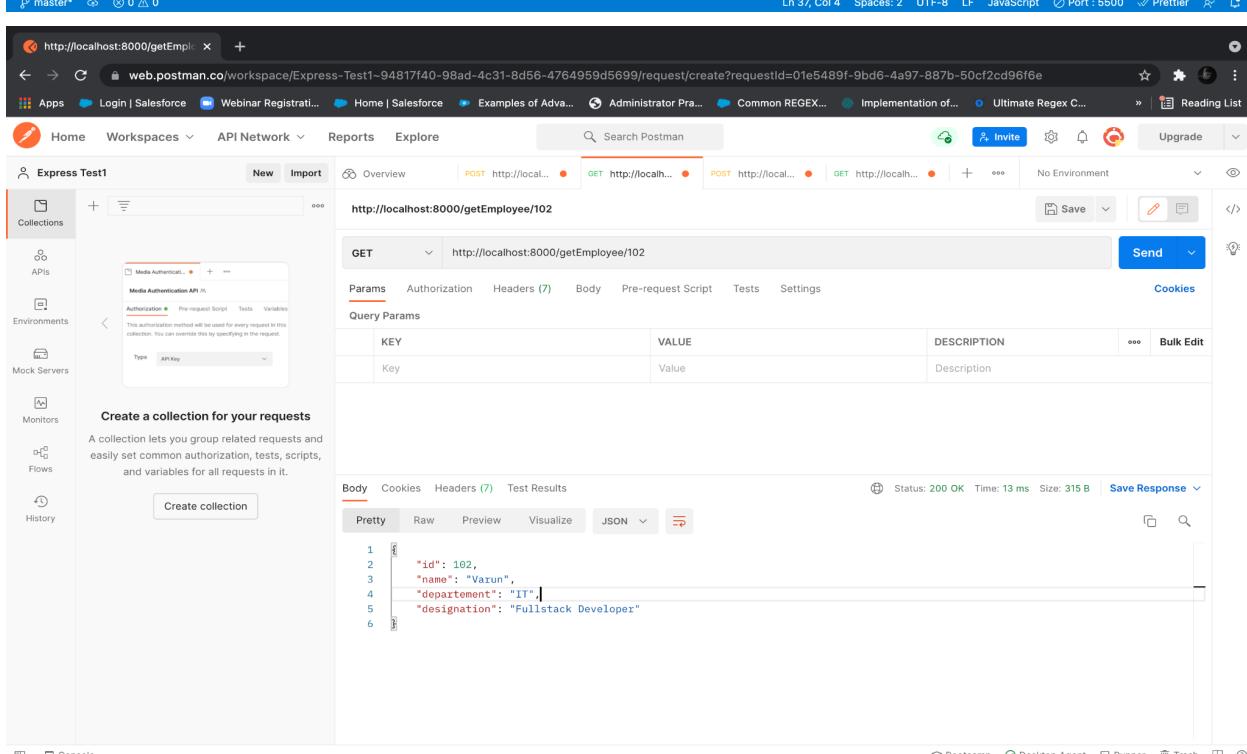
*****Already done in question 2.*****

b.get - getEmployeeById() -- Takes the empid as the path param and displays employee record based on given Id.



```

    app.get('/getEmployee/:id', (req,res)=>{
      var empId = req.params.id;
      var result = {};
      for(var i=0;i<emp.length;i++){
        if(emp[i].id == empId){
          result = emp[i];
        }
      }
      res.send(result);
    });
    app.listen(8000, ()=>{
      console.log("Server is running at port : 8000");
    });
  });
}
  
```

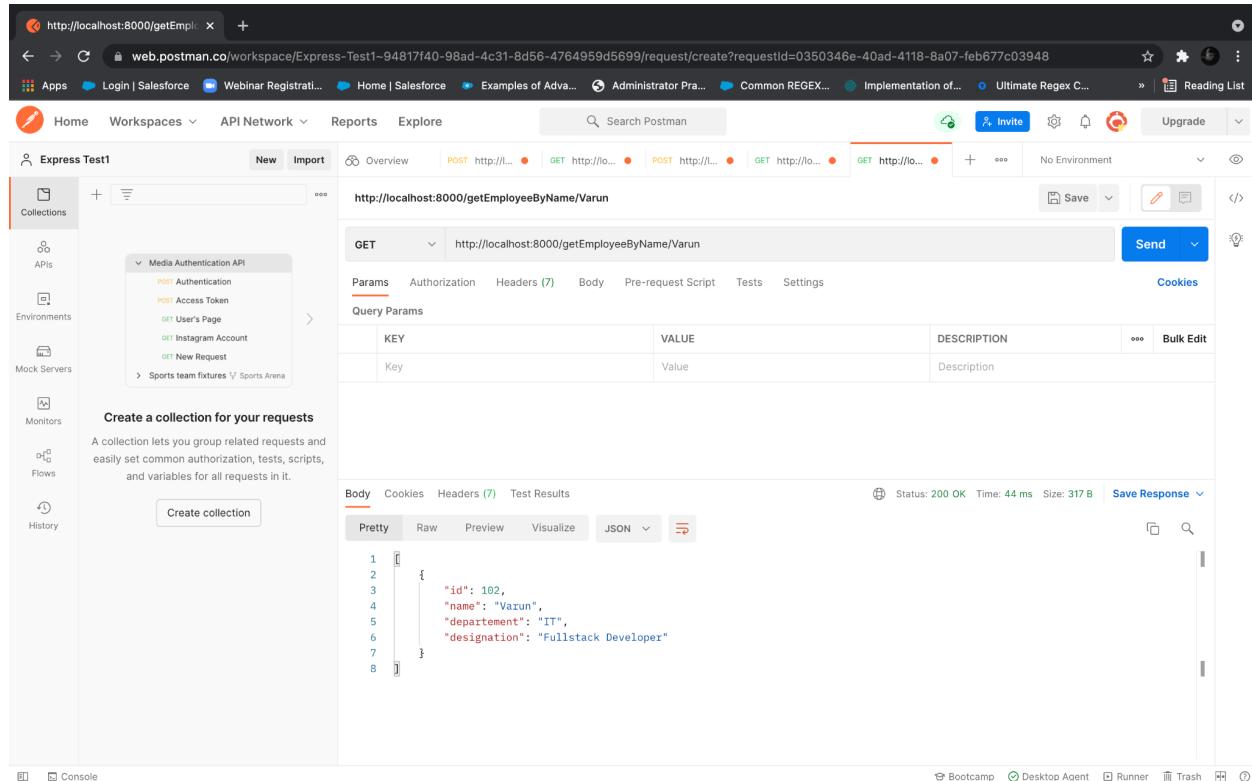
KEY	VALUE	DESCRIPTION
Key	Value	Description

```

1
2   "id": 102,
3   "name": "Varun",
4   "department": "IT",
5   "designation": "Fullstack Developer"
6
  
```

c. get - `getEmployeeByName()` -- Returns all the matching records with the given name.

```
app.get("/getEmployeeByName/:name", (req,res)=>{  
    var employee = emp.filter( e => e.name == req.params.name);  
  
    res.send(employee);  
});
```



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. A 'Create a collection for your requests' section is visible. In the main area, a collection named 'Express Test1' is selected. Under it, a 'Media Authentication API' folder contains several requests: 'Authentication', 'AccessToken', 'User's Page', 'Instagram Account', 'New Request', and 'Sports team fixtures'. A specific request titled 'http://localhost:8000/getEmployeeByName/Varun' is selected, showing a 'GET' method. The 'Body' tab of the request details panel displays the following JSON:

```
1 [ {  
2     "id": 102,  
3     "name": "Varun",  
4     "department": "IT",  
5     "designation": "Fullstack Developer"  
6 } ]
```

The 'Body' tab also shows the response status as 200 OK, time as 44 ms, and size as 317 B. The response body is displayed in a JSON viewer with tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'.

d. post - `insertEmployeeData()` -- Takes an object from input and inserts into the array of objects and returns the string "Record inserted successfully" or "Unable to insert the record". After this operation, verify in postman that the given record is inserted by giving <http://localhost:8001/getAllEmployees>

```
//inserting Employee  
app.post("/insertEmployee", (req,res)=>[  
  
    var employee = {  
        "id" : req.body.id,  
        "name" : req.body.name,  
        "department" : req.body.department,  
        "designation" : req.body.designation  
    };  
    emp.push(employee);  
  
    res.send("Employee inserted successfully!");  
]);
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A 'Create a collection for your requests' section is visible. The main area displays a request for 'http://localhost:8000/insertEmployee'. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   ...
3   "id": 106,
4   ...
5   "name": "kiran",
6   ...
7   "department": "IT",
8   ...
9   "designation": "Senior manual Tester"
10

```

Below the body, the response status is shown as 200 OK with a time of 8 ms and a size of 259 B. The response body contains the message: 'Employee inserted successfully!'

Verifying whether the data is added into emp object array using '<http://localhost:8000/getAllEmployeeData>' request

The screenshot shows the Postman application interface. The left sidebar is identical to the previous one. The main area displays a request for 'http://localhost:8000/getAllEmployeeData'. The 'Body' tab is selected, showing a JSON response containing multiple employee objects:

```

8 [
9   {
10     ...
11     "id": 103,
12     ...
13     "name": "Vishal",
14     ...
15     "department": "IT",
16     ...
17     "designation": "UI/UX"
18   },
19   {
20     ...
21     "id": 104,
22     ...
23     "name": "Pooh",
24     ...
25     "department": "IT",
26     ...
27     "designation": "SDE"
28   },
29   {
30     ...
31     "id": 105,
32     ...
33     "name": "Mok",
34     ...
35     "department": "HR",
36     ...
37     "designation": "Senior HR"
38   },
39   {
40     ...
41     "id": 106,
42     ...
43     "name": "kiran",
44     ...
45     "department": "IT",
46     ...
47     "designation": "Senior manual Tester"
48   }
49 ]

```

Below the body, the response status is shown as 200 OK with a time of 5 ms and a size of 597 B. The response body is identical to the JSON above.

e. put - updateEmployeeData() -- Takes an object and updates that record in an array of objects. After this operation, verify in postman that the given record is updated by giving '<http://localhost:8000/getAllEmployeeData>' request

```
//updating the Employee data with data from put request to updateEmployee
app.put('/updateEmployee', (req,res)=>{
  var employeeId = req.body.id;
  for(var i=0;i<emp.length;i++){
    if(emp[i].id == employeeId){
      emp[i].id = req.body.id;
      emp[i].name = req.body.name;
      emp[i].department = req.body.department;
      emp[i].designation = req.body.designation;
    }
  }

  res.send("Employee data updated successfully!");

});
```

The screenshot shows the Postman application interface. On the left, there's a sidebar with collections, APIs, environments, mock servers, monitors, flows, and history. A 'Create collection for your requests' section is visible. The main area shows a 'PUT' request to 'http://localhost:8000/updateEmployee'. The 'Body' tab is selected, showing the following JSON payload:

```
1   {
2     "id" : 102,
3     "name" : "Varun",
4     "department" : "IT",
5     "designation" : "SDE-1"
6 }
```

Below the request, the response pane shows a status of 200 OK, a time of 22 ms, and a size of 263 B. The response body is displayed as:

```
Pretty Raw Preview Visualize HTML
1   Employee data updated successfully!
```

Verifying whether the data is added into emp object array using '<http://localhost:8000/getAllEmployeeData>' request

The screenshot shows the Postman interface with a collection named 'Express Test1'. A GET request is made to `http://localhost:8000/getAllEmployeeData`. The response status is 200 OK, and the response body is a JSON array of five employees:

```

1  [
2    {
3      "id": 102,
4      "name": "Varun",
5      "department": "IT",
6      "designation": "SDE-1"
7  },
8  {
9    "id": 103,
10   "name": "Vishal",
11   "department": "IT",
12   "designation": "UI/UX"
13 },
14 {
15   "id": 104,
16   "name": "Pooh",
17   "department": "IT",
18   "designation": "SDE"
19 },
20 {
21   "id": 105,
22   "name": "Mok",
23   "department": "HR",
24   "designation": "Senior HR"
25 }

```

f. delete - `deleteRecord()` -- Takes the id as the path parameter and removes the record on given id and returns the "Record deleted successfully" or "Unable to delete record". After this operation, verify in postman that the given record is deleted by giving '<http://localhost:8000/getAllEmployeeData>' request

```

//deleting the employee with the given id
app.delete('/deleteEmployee/:id', (req,res)=>{
  var employeeId = req.params.id;
  var start = 0;
  for(var i=0;i<emp.length;i++){
    if(emp[i].id == employeeId){
      start = i;
      break;
    }
  }

  emp.splice(start,1);

  res.send(`Employee with id : ${employeeId} has been deleted successfully!`);
});

```

The screenshot shows the Postman application interface. On the left, there's a sidebar with various sections like 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. A 'Create a collection for your requests' section is also present. The main area displays a collection named 'Express Test1' which contains a 'Media Authentication API' with several requests. A specific request is selected: a 'DELETE' request to 'http://localhost:8000/deleteEmployee/106'. The 'Body' tab of the request details shows an empty JSON object. The 'Test Results' tab shows a successful response with status 200 OK, time 5 ms, and size 281 B. The response body is displayed as:

```
1 Employee with id : 106 has been deleted successfully!
```

Verifying whether the data is added into emp object array using '<http://localhost:8000/getAllEmployeeData>' request

The screenshot shows the Postman application interface, similar to the previous one but with a different request. The main area displays a collection named 'Express Test1' with a 'Media Authentication API' containing a 'GET' request to 'http://localhost:8000/getAllEmployeeData'. The 'Body' tab of the request details shows an empty JSON object. The 'Test Results' tab shows a successful response with status 200 OK, time 5 ms, and size 516 B. The response body is displayed as:

```
2 [
3   {
4     "id": 102,
5     "name": "Varun",
6     "department": "IT",
7     "designation": "Fullstack Developer"
8   },
9   {
10    "id": 103,
11    "name": "Vishal",
12    "department": "IT",
13    "designation": "UI/UX"
14  },
15  {
16    "id": 104,
17    "name": "Pooh",
18    "department": "IT",
19    "designation": "SOE"
20  },
21  {
22    "id": 105,
23    "name": "Mok",
24    "department": "HR",
25    "designation": "Senior HR"
26 }
```