PROGRAM 3a

3a. Write a program to check request header for cookies.

DESCRIPTION

The question "Write a program to check request header for cookies" is to create a program that examines incoming HTTP requests and determines whether they contain cookies in their headers.

In the context of web development, cookies are small pieces of data that websites store on a user's computer. When a user visits a website, their browser sends any cookies associated with that website along with the HTTP request. These cookies are typically stored in the request headers.

The task is to write a program, likely using a web framework such as Express.js for Node.js, that intercepts incoming HTTP requests and inspects their headers. Specifically, the program should check if the "Cookie" header is present in the request. If it is, the program should log the cookies to the console or perform any other desired actions based on the presence of cookies.

Overall, the goal is to create a simple server-side application that can detect the presence of cookies in incoming HTTP requests.

1. **Initialize a New Node.js Project:** Open your terminal or command prompt, navigate to the directory you created, and run the following command to initialize a new Node.js project:

```
npm init –y
```

2. **Install Express.js:** Next, install the Express.js framework. In your terminal, run the following command:

```
npm install express
```

3. **Create Your JavaScript File: Create a new JavaScript file (e.g., app.js) in your project directory and insert the provided code into it.**

```
const express = require('express');
const app = express();

// Middleware to log request headers
app.use((req, res, next) => {
   // Check if 'Cookie' header exists in the request
   if (req.headers.cookie) {
      console.log('Cookies found in the request:');
      console.log(req.headers.cookie);
   } else {
      console.log('No cookies found in the request.');
   }
   next();
});

// Route handler
app.get('/', (req, res) => {
   res.send('Hello World!');
});

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
   console.log(`Server is listening on port ${PORT}`);
});
```

This code sets up a basic Express.js server,below is the explanation of the code:

- **Imports**: It imports the express module.
- **App Initialization**: It initializes an Express application by calling express() and assigns it to the variable app.
- **Middleware**: It defines a middleware function using app.use(). This middleware logs the request headers. If the request contains a 'Cookie' header, it prints out the cookie content. Otherwise, it logs that no cookies were found. next() is called to pass control to the next middleware in the stack.
- **Route Handler**: It defines a route handler for the root path ('/'). When a GET request is made to the root path, it sends back the response 'Hello World!'.
- **Server Initialization:** It starts the server listening on either the port specified in the environment variable PORT or port 3000. When the server starts, it logs a message indicating which port it's listening on.

In summary, this code creates a server using Express.js, logs request headers, and responds with 'Hello World!' when a GET request is made to the root path.

Run Your Program: In your terminal, navigate to the directory containing your JavaScript file (e.g., app.js). Then, run the following command to execute your program:

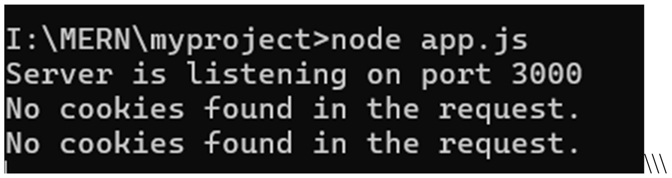
```
node app.js
```

open(**http://localhost:3000)**

- This command starts the Express.js server, and you should see a message in the terminal indicating that the server is listening on a port (by default, port 3000).

- **Test Your Program**: Open a web browser or use a tool like cURL or Postman to make HTTP requests to your server. You can send requests to **http://localhost:3000** (or the port specified in your terminal if different) to trigger the middleware that checks for cookies. The program will log any cookies found in the request headers to the terminal.

OUTPUT:



```
I:\MERN\myproject>node app.js
Server is listening on port 3000
No cookies found in the request.
No cookies found in the request.
```

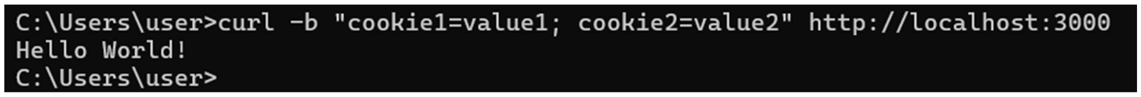
To Add cookies

- **Using cURL:**

If you're using cURL, you can add cookies to the request header using the -b or --cookie option followed by a string containing the cookie(s) you want to include. (Use new terminal window)

curl -b "cookie1=value1; cookie2=value2" http://localhost:3000

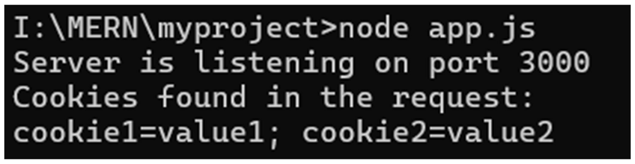
OUTPUT:



```
C:\Users\user>curl -b "cookie1=value1; cookie2=value2" http://localhost:3000
Hello World!
C:\Users\user>
```

Press Enter: After typing the cURL command with the appropriate options, press Enter on your keyboard to execute the command.

OUTPUT:



```
I:\MERN\myproject>node app.js
Server is listening on port 3000
Cookies found in the request:
cookie1=value1; cookie2=value2
```

3 b. write node.js program to print the a car object properties, delete the second property and get length of the object.

```
// Define an array of car objects
let cars = [
    { make: 'Toyota', model: 'Corolla', year: 2020, color: 'Blue' },
    { make: 'Honda', model: 'Civic', year: 2019, color: 'Red' },
    { make: 'Ford', model: 'Mustang', year: 2021, color: 'Black' }
];

// Function to print the properties of an object
function printProperties(obj) {
    for (let key in obj) {
        console.log(`${key}: ${obj[key]}`);
    }
}

// Function to get the length of an object
function getObjectLength(obj) {
    return Object.keys(obj).length;
}

// Iterate over each car in the cars array
cars.forEach((car, index) => {
    console.log(`Car ${index + 1} properties:`);
    printProperties(car);
    console.log();
    // Delete the second property (in this case, 'model')
    let propertyKeys = Object.keys(car);
    delete car[propertyKeys[1]];



    // Print properties after deletion
    console.log(`Car ${index + 1} properties after deleting the second property:`);
    printProperties(car);
    console.log();

    // Get and print the length of the object
    let length = getObjectLength(car);
    console.log(`Length of car ${index + 1} object: ${length}`);
    console.log('--------------------------');
});
```

Explanation:

Create an array of car objects: The cars array contains multiple car objects, each with properties make, model, year, and color.

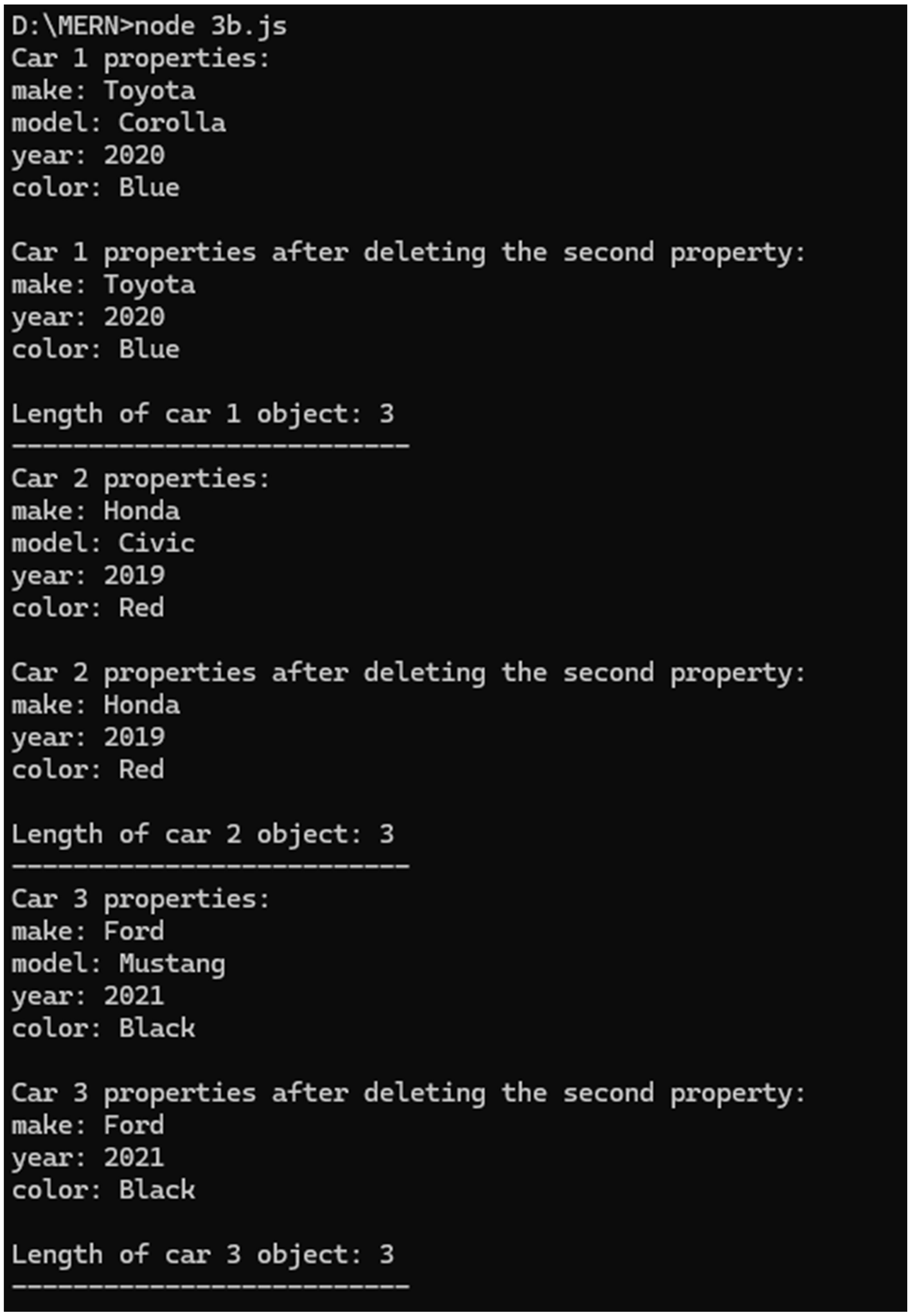
Print the properties: The printProperties function iterates over an object's keys and prints each key-value pair.

Delete the second property of each car: The code iterates over the cars array, and for each car, it gets the property keys using Object.keys(car) and deletes the second property using delete car[propertyKeys[1]].

Print the properties again: After deleting the second property, the properties of each car are printed again to confirm the deletion.

Get the length of each car object: The length (number of properties) is calculated using Object.keys(obj).length and printed for each car.

OUTPUT:



```
D:\MERN>node 3b.js
Car 1 properties:
make: Toyota
model: Corolla
year: 2020
color: Blue

Car 1 properties after deleting the second property:
make: Toyota
year: 2020
color: Blue

Length of car 1 object: 3
--------------------------
Car 2 properties:
make: Honda
model: Civic
year: 2019
color: Red

Car 2 properties after deleting the second property:
make: Honda
year: 2019
color: Red

Length of car 2 object: 3
--------------------------
Car 3 properties:
make: Ford
model: Mustang
year: 2021
color: Black

Car 3 properties after deleting the second property:
make: Ford
year: 2021
color: Black

Length of car 3 object: 3
--------------------------
```