ASSIGNMENT – 3

FULL STACK WEB DEVELOPMENT

By,

J MOHAMED RIYAS

Cookie Management

INTRODUCTION:

A cookie is usually a tiny text file stored in your web browser. A cookie was initially used to store information about the websites that you visit. But with the advances in technology, a cookie can track your web activities and retrieve your content preferences. I am sure, at some point, you have seen a pop-up screen similar to the one shown below.

Therefore, cookies are small strings that contain key-value pairs of information sent from the webserver to the browser to get information about the user. The browser will then save them locally. This way, subsequent requests can be made to the server to immediately update user content on the website depending on the previous requests that a user made to the server. A cookie is HTTP generated; thus, called an HTTP cookie.

AIM AND OBJECTIVE:

Aim:

- ✓ Google Analytics (_ga): Used to identify and distinguish a user. They store a unique client IDENTIFIER (client ID), which is randomly generated. It is used to calculate user interactions (visits, user data, sessions, and campaigns), with the aim to optimize the services offered.
- ✓ Google Analytics (_gid): Used to identify a single user throughout the day.
- ✓ Google Analytics (_gat): Used to differentiate between the different trace objects created in a session. The cookie is updated each time the data is sent to Google Analytics.

Objective:

- ✓ Anonymously identify browsing users through a cookie (identifies browsers and devices, not people) in order to count the number of visitors and track usage trends.
- ✓ Anonymously identify the most visited content and therefore the most attractive content to Users
- ✓ Determine whether the user accessing the site is a new or a repeat visitor.

A brief history of cookies:

The first HTTP cookie was created in 1994 by ***Lou Montulli***, an employee of Netscape Communications, the company that created the ***Netscape browser***. Lou was creating an online store for a company that claimed that their servers were getting full from storing each user's shopping cart data.

Lou, therefore, had to figure out how to store the contents of the shopping cart locally. He came up with an idea to save the shopping cart info on the user's computer to save server space. He borrowed the concept of HTTP cookies from a computing token called the magic cookie which was used to identify a user when logging into a system.

Lou recreated this concept and implemented it in a web browser. In ***1994, the Netscape*** browser implemented cookies, followed by ***Internet Explorer in 1995 and that marked the birth of HTTP cookies***.

The different types of cookies include:

- ***Session cookies*** - store user's information for a short period. When the current session ends, that session cookie is deleted from the user's computer.

- ***Persistent cookies*** - a persistent cookie lacks expiration date. It is saved as long as the webserver administrator sets it.

- ***Secure cookies*** - are used by encrypted websites to offer protection from any possible threats from a hacker.

- ***Third-party cookies*** - are used by websites that show ads on their pages or track website traffic. They grant access to external parties to decide the types of ads to show depending on the user's previous preferences.

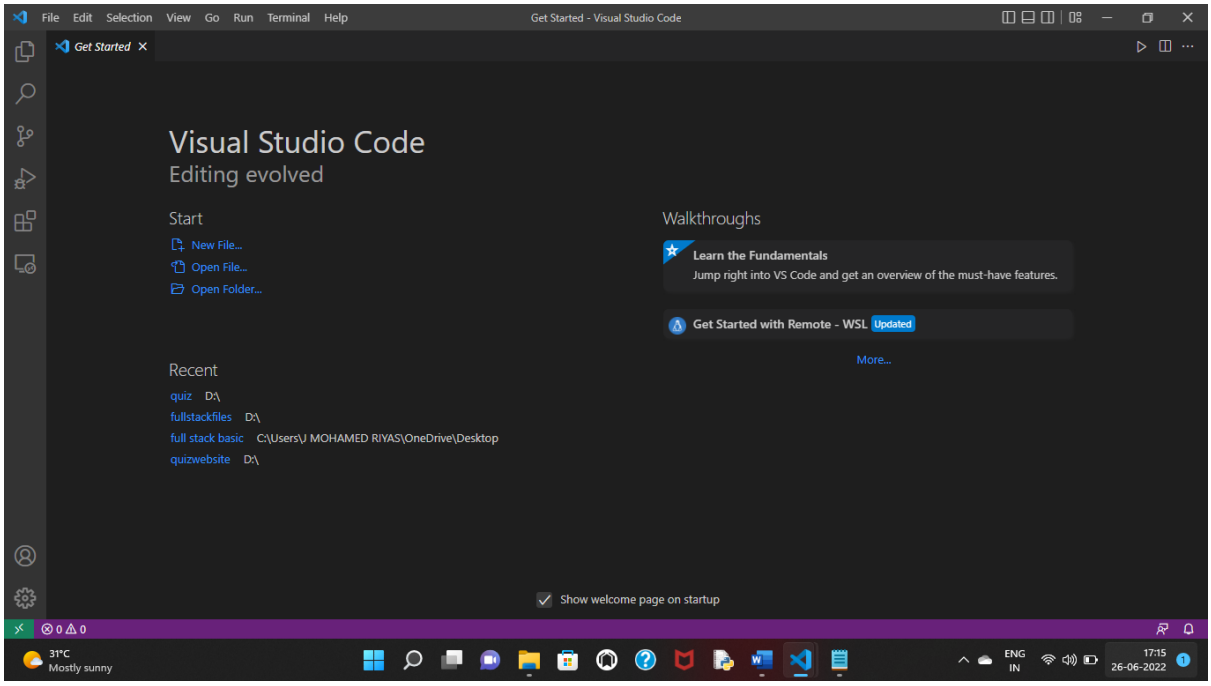
JUSTIFICATION OF PLATFORM:

Justification means why I choose this project to make because not just for making Project for College It is for improving my skills.

- **why visual studio code?**

Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging.

An integrated development environment (IDE) is a feature-rich program that supports many aspects of software development. The Visual Studio IDE is a creative launching pad that you can use to edit, debug, and build code, and then publish an app. Over and above the standard editor and debugger that most IDEs provide, Visual Studio includes compilers, code completion tools, graphical designers, and many more features to enhance the software development process.



Web Technology: (Cookie Management)

- **Node.js:** (The major points of Node.js are given below)

    1) Node.js tutorial provides basic and advanced concepts of Node.js.
    2) Our Node.js tutorial is designed for beginners and professionals both.
    3) Node.js is a cross-platform environment and library for running JavaScript applications which is used to create networking and server-side applications.

- **Express.js:** (The major points of Express.js are given below)

    1) Express provides a minimal interface to build our applications. It provides us the tools that are required to build our app. It is flexible as there are numerous modules available on **npm**, which can be directly plugged into Express.
    2) Express was developed by **TJ Holowaychuk** and is maintained by the **Node.js** foundation and numerous open source contributors.
    3) Pug (earlier known as Jade) is a terse language for writing HTML templates. It −
        - Produces HTML
        - Supports dynamic code
        - Supports reusability (DRY)

HOW TO SETTING UP COOKIES: (Cookie Management)

- Let's dive in and see how we can implement cookies using Node.js. We will create and save a cookie in the browser, update and delete a cookie.
- Go ahead and create a project directory on your computer.

Step 0:

Initialize ***Node.js*** using ***npm init -y*** to generate a ***package.json*** file to manage Node.js project dependencies.

PS D:\cookies_management> npm init -y

npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

Wrote to D:\cookies_management\package.json:

```
{
  "name": "cookies_management",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Step 1:

We will use the following NPM packages:

- ✓ ***Express*** - this is an opinionated server-side framework for Node.js that helps you create and manage HTTP server REST endpoints.

- ✓ ***cookie-parser*** - cookie-parser looks at the headers in between the client and the server transactions, reads these headers, parses out the cookies being sent, and saves them in a browser. In other words, cookie-parser will help us create and manage cookies depending on the request a user makes to the server.

Run the following command to install these NPM packages:

PS D:\cookies_management> npm install express cookie-parser

npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 59 packages, and audited 60 packages in 3s

7 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

Step 2:

We will create a simple example to demonstrate how cookies work.

Source code:

(File name: *app.js***)**

```
const express = require('express')
const cookieParser = require('cookie-parser')

const app = express()
app.use(cookieParser());

app.get('/', (req, res) => {
    res.send('welcome to a simple HTTP cookie server');
});

app.listen(2000, () => console.log('The server is running port 2000...'));
```

Now we have a simple server set. ***Run*** ***`node app.js`*** *to test if it is working.*

PS D:\cookies_management> node app.js

The server is running port 2000...

And if you access the localhost on port 2000 ***(`http://localhost:2000/`)****, you should get an HTTP response sent by the server. Now we're ready to start implementing cookies.*

SETTING COOKIES:

Let's add `routes` and `endpoints` that will help us ***create cookie****.*

Step 1: (Set a cookie)

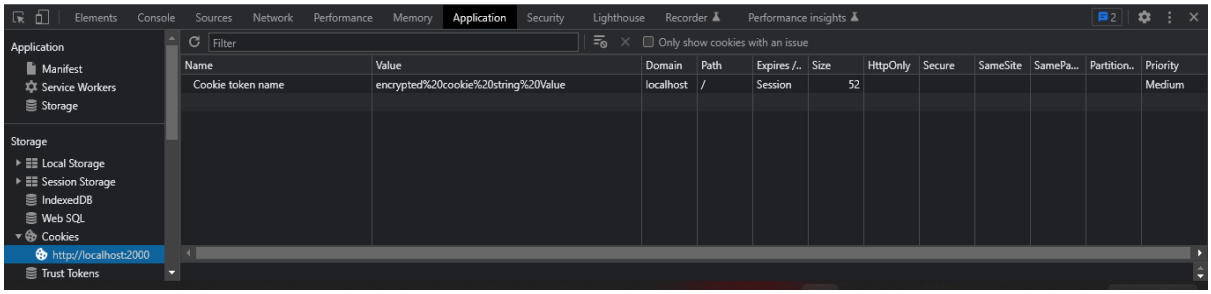
We will set a route that will save a cookie in the browser. In this case, the cookies will be coming from the server to the client browser. To do this, use the ***`res` object and pass `cookie`*** *as the method, i.e.* ***`res.cookie()`*** *as shown below.*

(File name: *app.js***)**

```
const express = require('express')
const cookieParser = require('cookie-parser')

const app = express()
app.use(cookieParser());

app.get('/', (req, res) => {
    res.send('welcome to a simple HTTP cookie server');
});

app.get('/setcookie', (req, res) => {
    res.cookie(`Cookie token name`,`encrypted cookie string Value`);
    res.send('Cookie have been saved successfully');
});

app.listen(2000, () => console.log('The server is running port 2000...'));
```

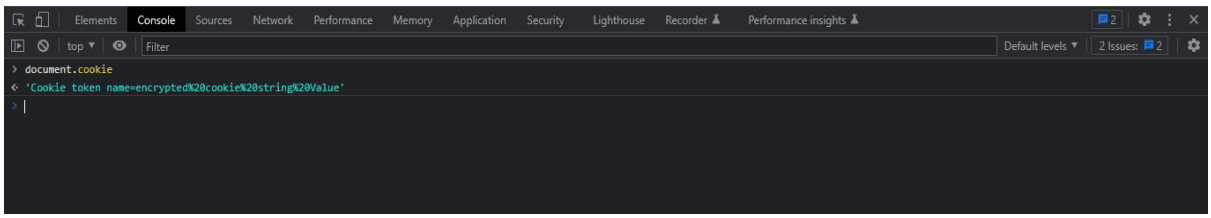
Go ahead and run `node app.js` to serve the above endpoint. Open ***`http://localhost:2000/setcookie`*** *your browser and access the route.*

*To confirm that the cookie was saved, go to your browser's **More tools** => select the **developer tools** => select the **Application tag** => **cookies** => select your **domain URL**.*



Step 2: (Secure cookies)

- ✓ One precaution that you should always take when ***setting cookies is security***. In the above example, the cookie can be deemed insecure.
- ✓ For example, you can access this cookie on a browser console using JavaScript ***(`document.cookie`)***. This means that this cookie is exposed and can be exploited through cross-site scripting.
- ✓ You can see the cookie when you open the browser ***inspector tool*** and execute the following in the console.



As a precaution, you should always try to make your cookies inaccessible on the client-side using JavaScript.

We can add several attributes to make this cookie more secure:

- ***HTTPonly*** ensures that a cookie is not accessible using the JavaScript code. This is the most crucial form of protection against cross-scripting attacks.
- A `secure` attribute ensures that the browser will reject cookies unless the connection happens over HTTPS.
- ***sameSite*** attribute improves cookie security and avoids privacy leaks.
- By default, ***sameSite*** was initially set to `none` (***sameSite = None***). This allowed third parties to track users across sites. Currently, it is set to `Lax` (***sameSite = Lax***) meaning a cookie is only set when the domain in the URL of the browser matches the domain of the cookie, thus eliminating third party's domains. ***sameSite*** can also be set to ***Strict (sameSite = Strict)***. This will restrict cross-site sharing even between different domains that the same publisher owns.
- You can also add the maximum time you want a cookie to be available on the user browser. When the set time elapses, the cookie will be automatically deleted from the browser.

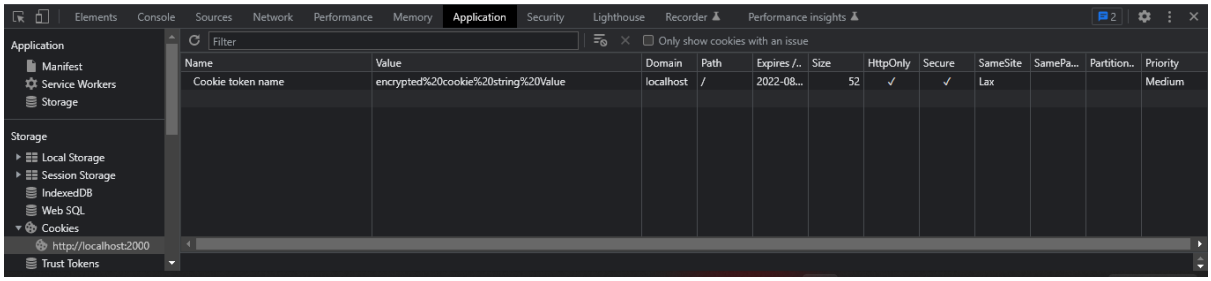
Read this guide to learn **more attributes** *and how you can use them in* **JavaScript** *and* **Node.js**.

(File name: *app.js*)

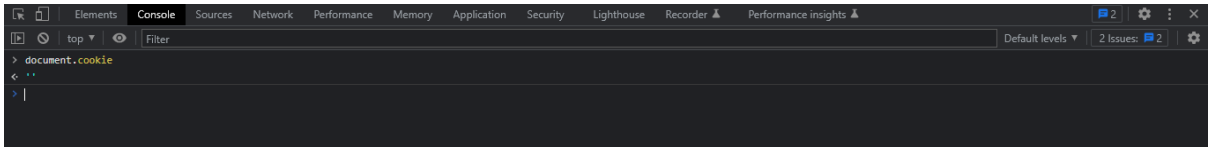
```
const express = require('express')
const cookieParser = require('cookie-parser')

const app = express()
app.use(cookieParser());

app.get('/', (req, res) => {
    res.send('welcome to a simple HTTP cookie server');
});

app.get('/setcookie', (req, res) => {
    res.cookie(`Cookie token name`,`encrypted cookie string Value`);
    res.cookie(`Cookie token name`,`encrypted cookie string Value`,{
```

```
        maxAge: 5000,
        expires: new Date('01 12 2021'),
        secure: true,
        httpOnly: true,
        sameSite: 'lax'
    });
    res.send('Cookie have been saved successfully');
});

app.listen(2000, () => console.log('The server is running port 2000...'));
```

If you run the server again (`node app.js`) and navigate to ***http://localhost:2000/setcookie*** on the browser, you can see that the values of the cookie have been updated with security values.



Furthermore, you cannot access the cookie using JavaScript, i.e., ***`document.cookie`***.



CONCLUSION:

Note that cookies are not intended for transmitting sensitive data. As a developer, you must ensure that the response you send to a client does not contain sensitive information such as passwords. A cookie is saved on a browser and can, therefore, be manipulated if it falls in the wrong hands. There are cookie regulations that make sure cookies are not used in the wrong way. These guidelines also restrict the type of data that a cookie can get from a user to avoid compromising user privacy. According to GDPR, internet identifiers such as cookies are considered personal data, and organizations can only exploit them with the client's permission. This the reason why cookie opt-in prompts are so common when you visit a website. I hope this guide helps you to understand what cookies are and how to use them in a Node.js web application in the correct way. Happy coding.

REFERENCE:

1. https://www.javatpoint.com/
2. https://www.tutorialspoint.com/index.htm
3. https://www.section.io/engineering-education/what-are-cookies-nodejs/
4. https://www.managementsolutions.com/en/cookies-policy
5. https://youtu.be/RNyNttTFQoc