Cookies

* Web Cookies

→ HTTP cookies are small blocks of data created by a web server while a user is browsing a website and placed on the user's computer or other device by the user's web browser.

We can see them in Inspect > storage > cookies as a key-value pair (Name & Value) column.

* Send Cookies

```
app.get("/setCookies", (req, res) => {
    res.cookie("greet", "namaste");
    res.cookie("origin", "India");
    res.send("we sent you a cookie");
});

app.get("/random" (req, res) => {

    console.log dir (req.cookies);
});
```

(key: "greet", value: "namaste")

→ Now, go to both path in browser, you can see cookie values in browser. But in console you cannot access cookies because of different routes.

* Cookie Parser npm Package

- npm install cookie-parser

→ To access & read from different routes

- const cookieParser = require("cookie-parser");

app.use(cookieParser());

→ Now, we can see output of "/random" path route in console:

{ greet: 'hello', origin: 'India' }

- eg. app.get("/greet", (req, res) => {

let { name = "anonymous" } = req.cookies;

res.send(`Hi, ${name}`);
});

→ Now, if set "name" named cookie from "setCookies" route then here we have that name from "setCookies" route.

* Signed Cookies

→ If someone changes cookie values then we can detect the changes by using signed cookies

- app.use(cookieParser("secretcode"));

```
app.get("/getSignedCookie", (req, res) => {
    res.cookie("color", "red", { signed: true });
    res.send("done!");
});
```

name: color
value: s%3Ared.IUR...

```
app.get("/verify", (req, res) => {
    res.send(req.signedCookies);
});
```

{ color: "red" }

→ If we change whole value of "color" cookie from browser itself then in output will become "{}"

→ If we change only "red" from "s%3Ared.IUR..." to "s%3Ayellow.IUR..." in browser itself then we will get output as "{ color: false }"

Sessions

→ Session client interacts with server; this single interaction is called single session.

→ protocol rules which req, res follows

→ state all information stored from req

→ stateful Protocol require server to save the status and session information e.g. ftp

→ stateless Protocol does not require server to retain the server information e.g. http

* Express Session

→ An attempt to make our session stateful.

- npm install express-session

→ This npm package, we will use to create a session middleware with given options

- const session = require("express-session");
app.use(session({ secret: "secretcode", resave: false, ~~saveUntimited~~ saveUninitialized: true}));

app.get("/test", ...);

output : Name connect.sid ← cookie Value s%.3A3L....

→ In one browser but within different tabs a single session will be stored for a website

eg. Refresh this website or open in new tab count will increase from before

-
```
app.get("/reqcount", (req, res) => {

    if (req.session.count) {
        req.session.count ++;
    } else {
        req.session.count = 1;
    }

    res.send(`You sent a request
        ${req.session.count}  times`);
});
```

⇒ This session data will be stored in temporary database (MemoryStore) but it is not recommended for production level application.

→ Use session store such as connect-mongo, connect-neo4j, connect-redis, ...

* Data Storing & Using info *

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

const session = require("express-session");

const sessionOptions = {
    secret: "mysecretcode",
    resave: false,
    saveUninitialized: true
};

app.use(session(sessionOptions));

app.get("/register", (req, res) => {

    let { name = "anonymous" } = req.query;
    console.log(req.session);
    res.send(name);
});

app.get("/hello", (req, res) => {

    res.send(`hello, ${req.session.name}`);
});
```

Console output: Session {
cookie: { path: '/', _expires: null,
originalMaxAge: null,
httpOnly: true }
}

* connect-flash npm package

→ The flash is a spaciel area of the session used for storing messages. Messages are written to the flash and cleared after being displayed to the user once.

- const flash = require("connect-flash");

```
app.use(flash());

app.get("/register", (req, res) => {
    req.flash("success", "user registered");
    res.redirect("/hello");
});

app.get("/hello", (req, res) => {
    consde.log(req.flash("success"));
    res.render("page.ejs", {msg: req.flash("success")});
});
```

key

message

access message through key

* res.locals

→ Use this property to set variables accessible in templates rendered with res.render

→ In above example, instead of passing data to ejs, we can directly access variable

- app.get("/hello", (req, res) => {

  res.locals.msg = req.flash("success");
  res.render("page.ejs");
});

In page.ejs ⇒ <%= msg %>

⇒ We can use middlewares for these flash messages. For that remove res.locals from above code of app.get("/hello", ...)

- app.use((req, res, next) => {

  res.locals.successMsg = req.flash("success");
  res.locals.errorMsg = req.flash("error");
  next();
});

Note: res.locals.successMsg is an array