SOEN 287

# Server side programming with Node.js (3) Cookies and Sessions and Files

Dr. Yuhong Yan
CSE, Concordia University
Winter, 2024

## Stateless vs. stateful services

- Recall that the HTTP protocol is stateless
- What does stateless mean?
- Why do we need stateful service?
  - Shopping cart
  - Targeted advertising
- Cookie and Session

## Cookie

- A *cookie* is a name/value pair that is passed between a browser and a server in the HTTP header
- Cookies can be initiated by the client or by the server.
- Cookies are used to store information that can be accessed across different pages or sessions.
- Cookies are stored on the client's device.
- Cookies are implicitly deleted when their lifetimes are over
- Cookies must be created before any other HTML is created by the script

## What we can do with cookies

- Know who visit which part of the site
- Know when is the visit
- Record session id (session uses cookie too!)
- What the server can do
  - generate customized interface
  - target advisement according to past interests

## Traffic

```
GET /somedir/page.php HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

**(extra carriage return, line feed)**

**Client → Server**

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html
Set-Cookie: name=value;expires= …;path=/
data data data data data ...
```

**Server → Client**

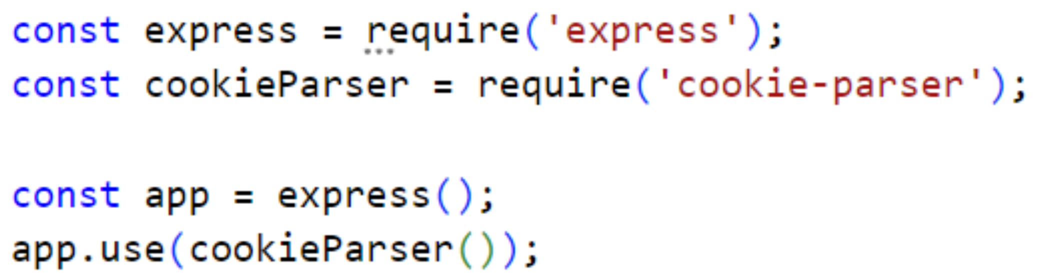
## Traffic (3)

```
GET /somedir/page2.html HTTP/1.1
Host: www.someschool.edu
Cookie: name=value
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr

(extra carriage return, line feed)
```

**Client → Server**

## CookieParser middleware

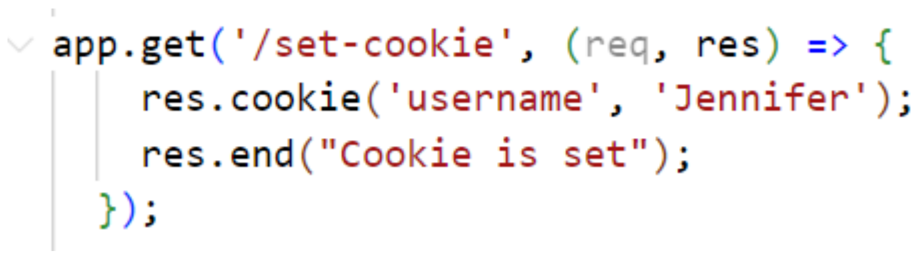
- cookieParser is a middleware in the express framework that simplifies cookie handling in Node.js applications.
- To use cookieParser:
- Install the package using npm: npm install cookie-parser
- Require and initialize it in your Express app:



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

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

## Baking a Cookie!

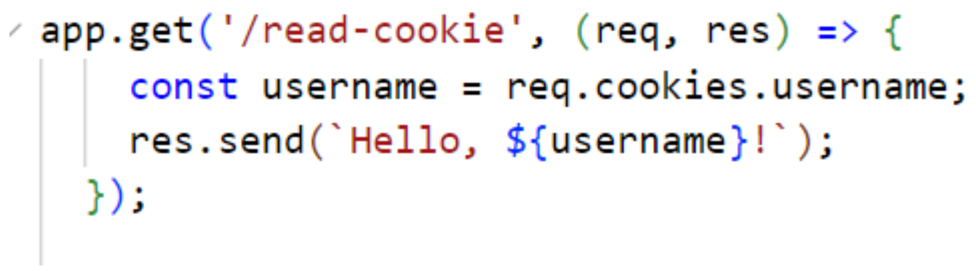
- Writing Cookies:
- Set cookies using the **res.cookie** method:
- Here, maxAge sets the cookie's expiration time in milliseconds.
- httpOnly restricts cookie access to server-side, enhancing security.



```
app.get('/set-cookie', (req, res) => {
    res.cookie('username', 'Jennifer');
    res.end("Cookie is set");
});
```

## Serving a Cookie!

- **Reading Cookies:**
- Access cookie values using **req.cookies** object, e.g., req.cookies.cookieName.



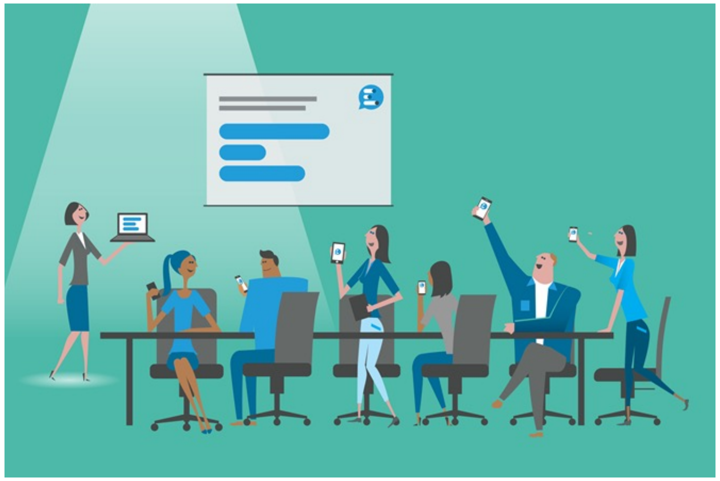
```
app.get('/read-cookie', (req, res) => {
    const username = req.cookies.username;
    res.send(`Hello, ${username}!`);
  });
```

## Sessions

- Understanding Sessions and Their Role

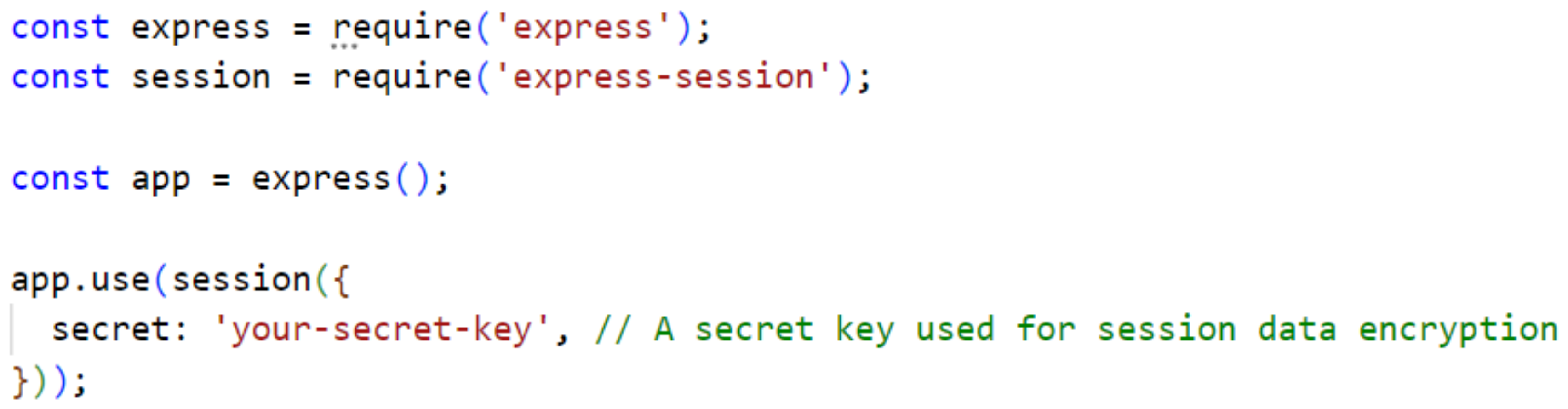
## Understanding Sessions and Their Role

- Sessions are a mechanism for maintaining user state across multiple requests.
- Unlike cookies, which are stored on the client-side, sessions are stored on the server.
- Advantages of Sessions:
- More secure: Sensitive information is stored server-side.
- Larger data storage: Not limited by the browser's cookie size.
- Sessions are especially useful for user authentication, managing shopping carts, and personalized experiences.



## express-session middleware

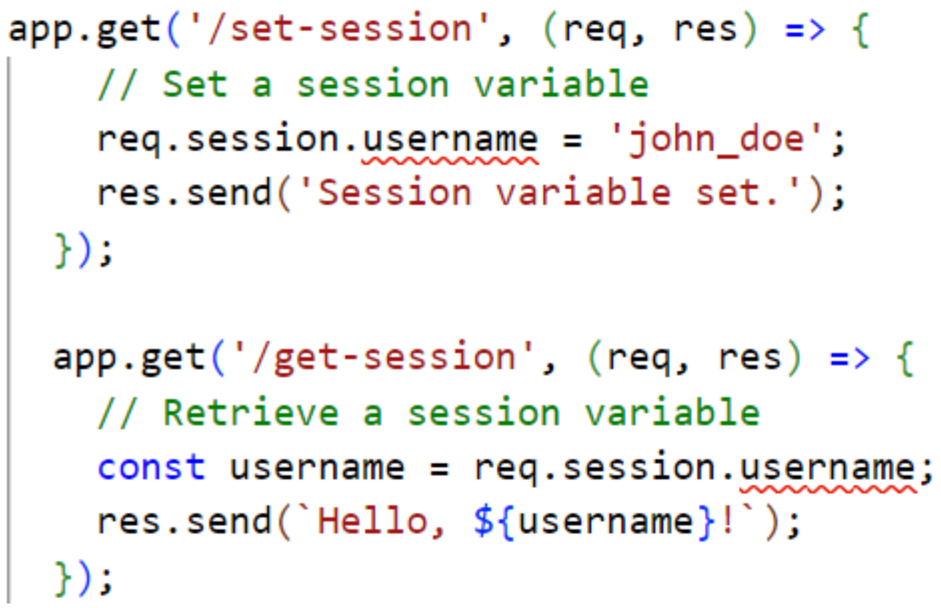
- express-session is a popular middleware for the Express.js framework that enables session management in your Node.js applications.
- **To use express-session:**
- Install the package using npm: npm install express-session and require it



```
const express = require('express');
const session = require('express-session');

const app = express();

app.use(session({
  secret: 'your-secret-key', // A secret key used for session data encryption
}));
```

## Working with express-session

- Once the express-session middleware is set up, you can start using sessions within your route handlers.



```
app.get('/set-session', (req, res) => {
    // Set a session variable
    req.session.username = 'john_doe';
    res.send('Session variable set.');
  });

  app.get('/get-session', (req, res) => {
    // Retrieve a session variable
    const username = req.session.username;
    res.send(`Hello, ${username}!`);
  });
```

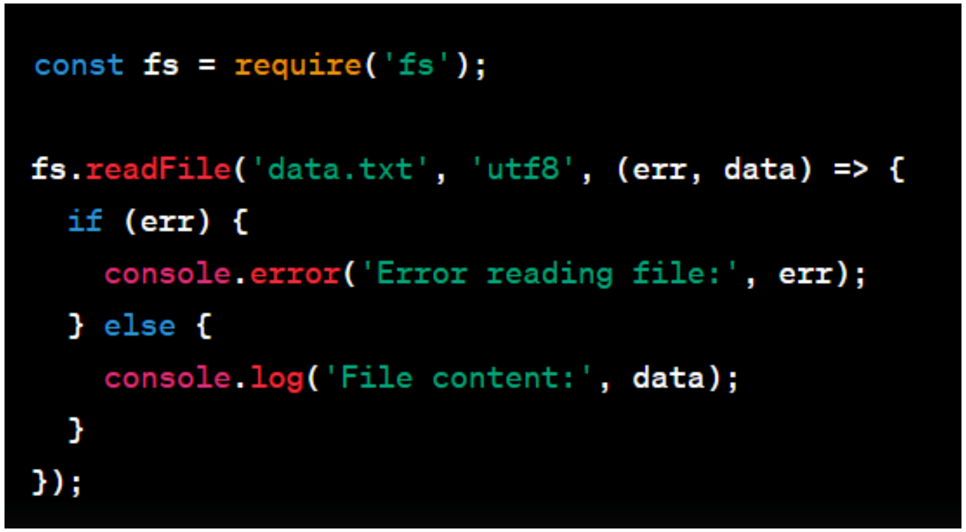
## Reading and Writing a Text File

- Hands-on exercise to read and write data to a text file using Node.js.

## const http = require('http');

### Reading a Text File

- Node.js provides built-in modules for file system operations, including reading files.
- Use the fs module's **readFile()** method to read the contents of a text file.

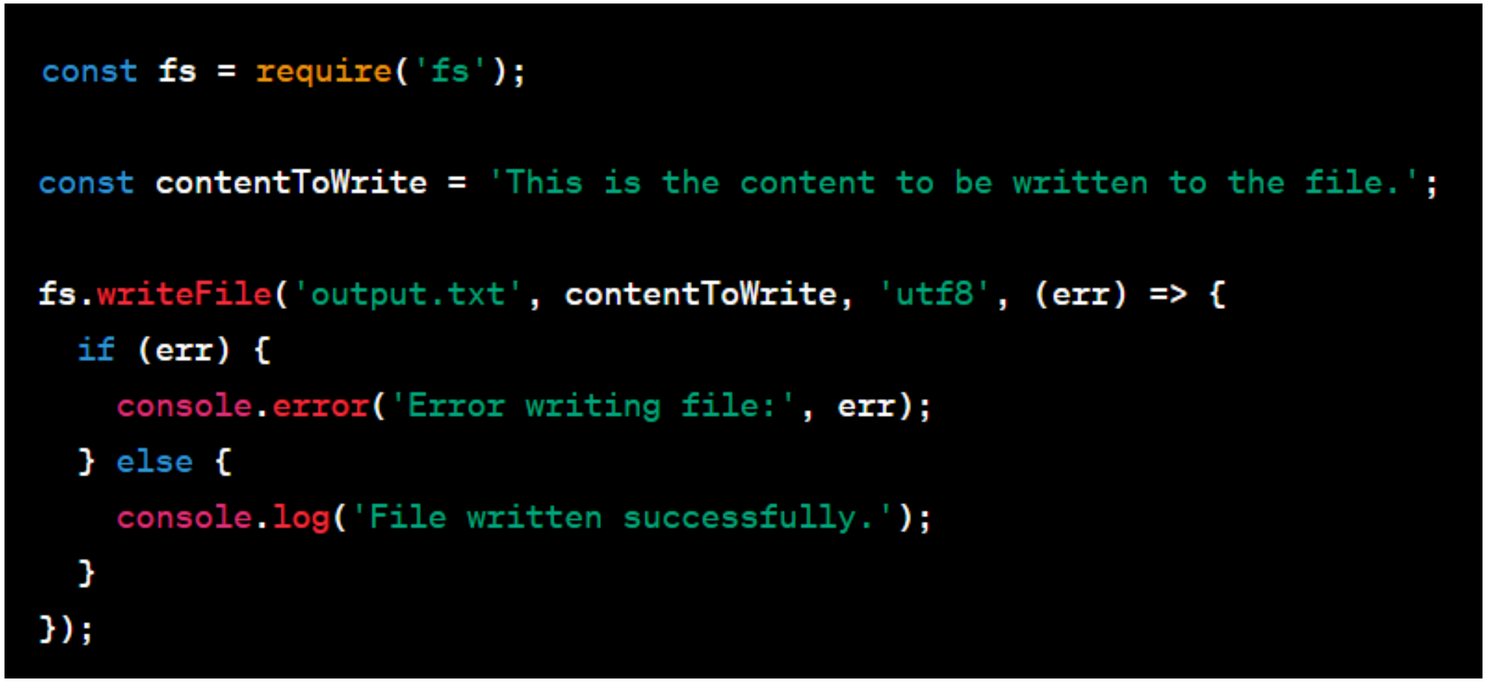


```
const fs = require('fs');

fs.readFile('data.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
  } else {
    console.log('File content:', data);
  }
});
```

const http = require('http');

## Writing to a Text File

- To write data to a text file, use the fs module's writeFile() method.
- Be careful, as the writeFile() method will overwrite the file if it already exists.

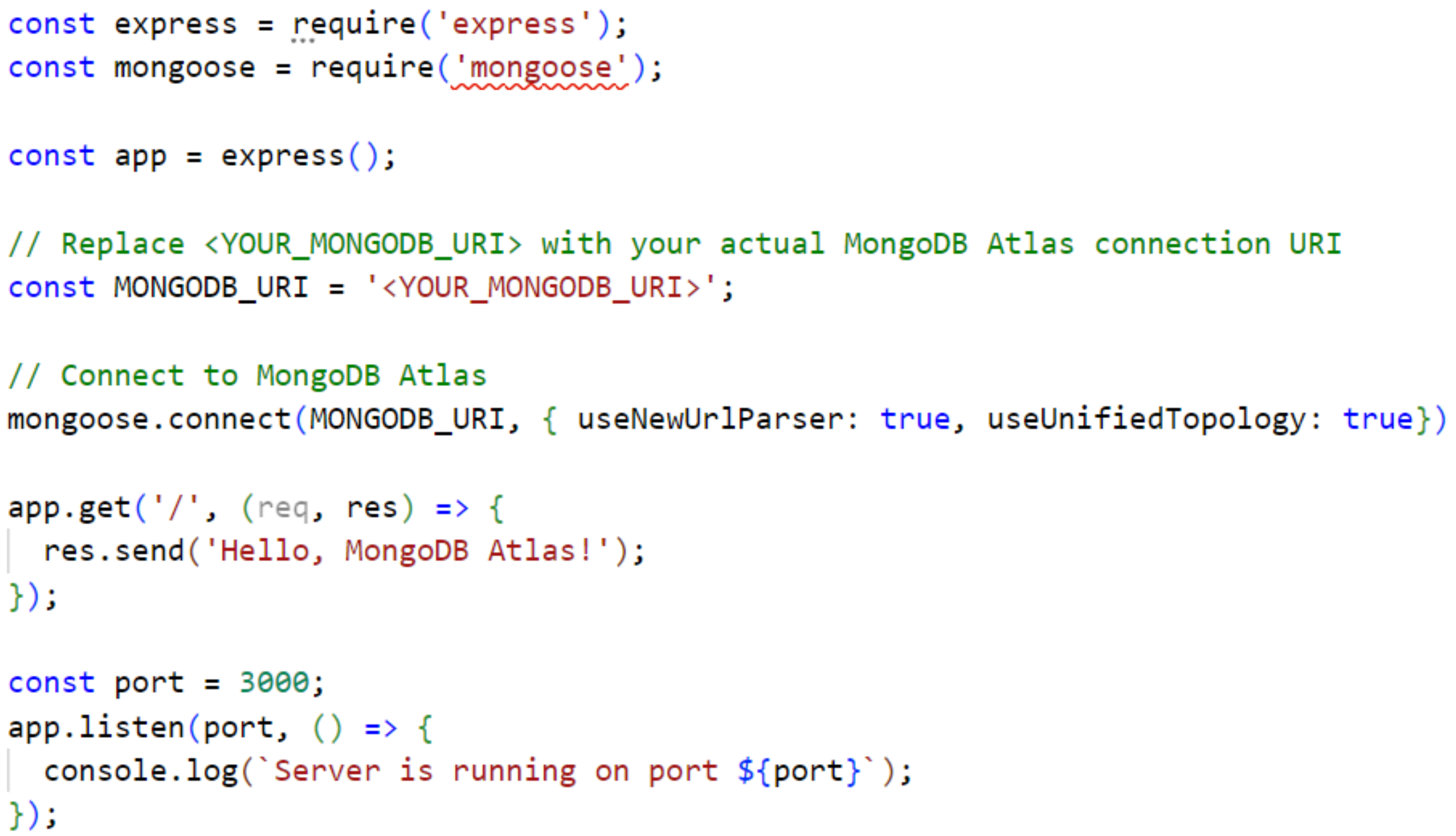


```
const fs = require('fs');

const contentToWrite = 'This is the content to be written to the file.';

fs.writeFile('output.txt', contentToWrite, 'utf8', (err) => {
  if (err) {
    console.error('Error writing file:', err);
  } else {
    console.log('File written successfully.');
  }
});
```

## Bounce !

node + mongoDB

## Example of connection to a database



```javascript
const express = require('express');
const mongoose = require('mongoose');

const app = express();

// Replace <YOUR_MONGODB_URI> with your actual MongoDB Atlas connection URI
const MONGODB_URI = '<YOUR_MONGODB_URI>';

// Connect to MongoDB Atlas
mongoose.connect(MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology: true})

app.get('/', (req, res) => {
  res.send('Hello, MongoDB Atlas!');
});

const port = 3000;
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

## The End