Week 4:

• Promises, Promise Chaining

• Command Line Arguments

• Working with the File System, Reading Files, Writing Files

Lab Exercises:

1) To create a Promise with Asynchronous code, using of promise's then() method.

1_Promises.js

```
var fs = require('fs');
var promise = new Promise(function (resolve, reject) {
fs.readFile('example.txt', 'utf8', function (error, data) {
if (error) {
return reject(error);
}
resolve(data);
});
});
promise.then(function (result) {
console.log(result);
}, function (error) {
console.error(error.message);
});
```

Output:

Run the script as follows:

>node 1_Promises.js

After executing the script, file contents should be displayed.

2) Demonstrating Promise Chaining

2_PromiseChaining.js

```
let promise = new Promise(function(resolve, reject) {
```

```
const user = {
name: 'ABC XYZ',
email: 'nouser@gmail.com',
password: 'abc'
};
resolve(user);
});
promise.then(function(user) {
console.log(`Got user ${user.name}`);
// Return a simple value
return user.email;
}).then(function(email) {
console.log(`User email is ${email}`);
});
```

Output:

Run the script as follows:

```
>node 2_PromiseChaining.js
```

After executing the script, it will print below:

```
Got user ABC XYZ
User email is nouser@gmail.com
```

3) Demonstrating handling rejections in Promise using catch()

3_PromisesChainingUsingCatch.js

```
/*Demonstrating handling rejections in Promise using catch() method
The code will behave as though the catch() callback were passed as the
second callback to then(), but is more convenient for chaining.
In the event that the promise is rejected, the catch() callback will
display the error message.*/
var fs = require('fs');
var promise = new Promise(function (resolve, reject) {
fs.readFile('example.txt', 'utf8', function (error, data) {
```

```
if (error) {
return reject(error);
}
resolve(data);
});
});
promise.then(function (result) {
console.log(result);
return 'THE END!';
}).catch(function (error) {
console.error(error.message);
});
```

Output:

Run the script as follows:

```
>node 3_PromisesChainingUsingCatch.js
```

After executing the script, file contents should be displayed.

4) Demonstrating promise interleaving then, catch and finally in a chain

4_PromisesChainingCatchfinally.js

```
/* Promise- Interleaving then, catch and finally in a chain*/
let myPromise = new Promise((resolve, reject) => {
setTimeout(() => {
let success = true; // true for resolved and false for rejection
if (success) {
resolve("Data fetched successfully!");
} else {
reject("Error fetching data.");
}
}, 2000);
});
```

```
myPromise.then((message) => {
console.log("Success:", message);
}).catch((error) => {
console.error("Error:", error);
}).finally(() => {
console.log("Promise settled (fulfilled or rejected).");
});
```

Output:

Run the script as follows:

```
>node 4_PromisesChainingCatchfinally.js
```

After executing the script, it will print below:

```
Success: Data fetched successfully!
Promise settled (fulfilled or rejected).
```

5) Demonstrating accessing command line arguments using process.argv-foreach()

5_CommandLineArgs.js

```
/*Accessing command line arguments using process.argv.
Using foreach method to iterate the command line arguments*/
/*Note that the first two elements of this array are the node
executable,
followed by the name of the invoked JavaScript file.
This means that the actual application arguments begin at
process.argv[2]. */
process.argv.forEach(function (value, index, args) {
console.log('process.argv[' + index + '] = ' + value);
});
```

Output:

Run the script as follows:

```
>node 5_CommandLineArgs.js
```

After executing the script, it will print below:

process.argv[0] = C:\Program Files\nodejs\node.exe

process.argv[1] = C:\Week4Project\5_commandlineargs.js

6) Demonstrating accessing command line arguments using process.argv-slice()

6_CommandLineArgs.js

```
/*Accessing command line arguments using process.argv slice method
to start slicing from a given index. The result of slice()
will store the elements in an array.
Further we perform addition on the given numbers.*/
//slice method starts taking values from the 2nd index since
// first two arguments are node and filename
const args = process.argv.slice(2);
// Check if two arguments are provided
if (args.length !== 2) {
console.log("Usage: node <currentFileName> <num1> <num2>");
process.exit(1);
}
// Convert arguments to numbers

const num1 = parseFloat(args[0]);
const num2 = parseFloat(args[1]);
// Validate input
if (isNaN(num1) || isNaN(num2)) {
console.log("Both arguments must be valid numbers.");
process.exit(1);
}
// Add and display the result
const sum = num1 + num2;
console.log(`The sum of ${num1} and ${num2} is ${sum}`);
```

Output:

Run the script as follows:

> node 6_CommandLineArgs.js 20 40

After executing the script, it will print below:

The sum of 20 and 40 is 60

7) Accessing command line arguments using process.argv to add a list of numbers given through command line arguments.

7_CommandLineArgs.js

```
/*Accessing command line arguments using process.argv to add a list of
numbers given through command line arguments*/
/*All command line arguments passed to a Node application are
available via the process.argv array and printed to the
console using the forEach() method.*/
/*Note that the first two elements of this array are the node
executable,
followed by the name of the invoked JavaScript file.
This means that the actual application
arguments begin at process.argv[2]. */
process.argv.forEach(function (value, index, args) {
// Skip the first two default arguments
if (index < 2) return;
// Convert the argument to a number
const num = parseFloat(value);
// Initialize args.sum if it doesn't exist
if (!args.sum) {
args.sum = 0;
}
// Add to the total if it's a valid number
if (!isNaN(num)) {
args.sum += num;
```

```
}
});

// Print the result using the args array
console.log("Sum of numbers:", process.argv.sum);
```

Output:

Run the script as follows:

```
> node 7_CommandLineArgs.js 10 20 30 40 50 60 70
```

After executing the script, it will print below:

```
Sum of numbers: 280
```

8) Demonstrating working with File System using __filename and __dirname

8_FileSystem.js

```
/*Demonstrating working with File System-Using __filename and __dirname
to print file paths.
Also to display and change the current working directory*/
console.log('Currently executing file is ' + __filename);
console.log('It is located in ' + __dirname);
console.log('Current working directory is in ' + process.cwd());
try {
process.chdir('/');
} catch (error) {
console.error('chdir: ' + error.message);
}
console.log('Current working directory is now ' + process.cwd());
```

Output:

Run the script as follows:

```
> node 8_FileSystem.js
```

After executing the script, it will print below:

```
It is located in C:\Week4Project
```

Current working directory is in C:\Week4Project

Current working directory is now C:\

9) Demonstrating readFile() with and without encoding

9_ReadFile_Encoding_WithoutEncoding.js

```
var fs = require('fs');
fs.readFile(__filename, function (error, data) {
if (error) {
return console.error(error.message);
}
console.log("Printing without encoding:")
console.log(data);
});
fs.readFile(__filename, "utf8", function (error, data) {
if (error) {
return console.error(error.message);
}
console.log("Printing by encoding:")

console.log(data);
});
```

Output:

Run the script as follows:

> node 9_ReadFile_Encoding_WithoutEncoding.js

After executing the script, it will print below:

```
Printing without encoding:
<Buffer 76 61 72 20 66 73 20 3d 20 72 65 71 75 69 72 65 28 27 66 73 27 29 3b 0d 0a
66 73 2e 72 65 61 64 46 69 6c 65 28 5f 5f 66 69 6c 65 6e 61 6d 65 2c 20 66 ... 372
more bytes>
Printing by encoding:
var fs = require('fs');
```

```
fs.readFile(__filename, function (error, data) {
if (error) {
return console.error(error.message);
}
console.log("Printing without encoding:")
console.log(data);
});
fs.readFile(__filename, "utf8", function (error, data) {
if (error) {
return console.error(error.message);
}
console.log("Printing by encoding:")
console.log(data);
});
```

10) Demonstrating asynchronous file reading using readFile()

10_ReadFileAsynchronously.js

```
/*Asynchronously reading a file using a callback function.*/
var fs = require('fs');
fs.readFile('example.txt', 'utf8', function (error, data) {
if (error) {
return console.error(error);
}
console.log(data);
});
console.log("End of code");
```

Output:

Run the script as follows:

```
> node 10_ReadFileAsynchronously.js
```

After executing the script, it will print below:

```
End of code
```

<<text from file>>

11) Demonstrating Synchronous file reading using readFileSync()

11_ReadFileSynchronously.js

```
/*Synchronously reading a file.*/
var fs = require('fs');
try {
var data = fs.readFileSync('example.txt', 'utf8');
console.log(data);
} catch (error) {
console.error(error);
}
console.log("End of code");
```

Output:

Run the script as follows:

> node 11_ReadFileSynchronously.js

After executing the script, it will print below:

<<text from file>>

End of code

12) Demonstrating asynchronous writing into a file using WriteFile(). Also demonstrated usage of flag wx.

12_WriteFileAsynchronously.js

```
/*Writing data to a file using Asynchronous file writeFile().
Udage of writeFile() with and without flag demonstrated*/
var fs = require('fs');
var data = 'This is my BCA file content.';
//writeFile() will create a new file, or overwrite an existing file
with the same name.
fs.writeFile(__dirname + '/bca.txt', data, function (error) {
if (error) {
```

```
return console.error(error.message);
}
});
//passing the flag wx causes an error to be thrown if the file already
exists,
// while the a flag causes data to be appended to an existing file
instead of overwriting.
fs.writeFile(__dirname + '/bca1.txt', data, {
flag: 'wx'
}, function (error) {
if (error) {
return console.error(error.message);
}
});
```

Output:

Run the script as follows:

```
> node 12_WriteFileAsynchronously.js
```

After executing the script, it will create a file bca.txt with given file contents if file is not present. If bca.txt is present, it will overwrite it.

The second writefile in bca1.txt will create the file with given file contents if file is not present. If bca1.txt is present, it will throw an error.

13) Demonstrating Synchronous writing into a file using WriteFileSync()

13_WriteFileSynchronously.js

```
/*Synchronously writing into a file.*/
var fs = require('fs');
try {
// Synchronous function: writeFileSync throws exceptions that can
be caught
fs.writeFileSync('myfile.txt', 'This is some sample content written
```

```
synchronously.', 'utf8');
console.log('File written successfully.');
} catch (error) {
console.error('Error writing to file:', error);
}
console.log("End of code");
```

Output:

Run the script as follows:

```
> node 13_WriteFileSynchronously.js
File written successfully.
End of code
```

Week 5:

- Readable Streams, Writable Streams
- The Standard Streams, Creating a Server, Routes
- Accessing Request Headers

LAB Exercise:

1) Use createReadStream() method and demonstrate reading of a large text file.

```
//readFile.js
const fs = require('fs');
let streamin = fs.createReadStream('log.txt', 'utf8');
let filedata='';
//emitted multiple times
streamin.on('data', function(data){
filedata += data;
});
//emitted at the end
streamin.on('end', function(){
console.log(filedata);
});
```

```
//emitted on error
streamin.on('error', function(){
console.log('Error reading the file');
})
```

Output:

Run the script as follows:

```
>node readFile.js
```

After executing the script, file contents should be displayed to the console.

2) Use createReadStream() and createWriteStream() methods and transfer a large text file to a new folder. Also demonstrate the transfer progress.

```
//copyFile.js
const fs = require('fs');
let streamin = fs.createReadStream('log.txt', 'utf8');
let streamout = fs.createWriteStream('bkup/log.txt', 'utf8');
streamin.pipe(streamout);
streamin.on('data', function(data){
console.log('transferred: %d', data.length);
});
streamin.on('end', function(){
console.log('finished');
});
streamin.on('error', function(){
console.log('Error copying');
})
```

Output:

```
> node copyFile.js
transferred: 65536
transferred: 65536
transferred: 37762
finished
```

3) Demonstrate the use of createReadStream() and pipe() method to stream a small video.

```
//videoStream.js
var http = require('http')
var fs = require('fs');
//create a server object
var serv = http.createServer(func);
serv.listen(8080);//the server object listens on port 8080
console.log('server is running');
var stream;
function func(req, res)
{
//console.log(req.url);

var path = 'sample.mp4';
const stat = fs.statSync(path);
const fileSize = stat.size;
const head = {'Content-Length': fileSize,
'Content-Type': 'video/mp4' }
res.writeHead(200, head);
stream = fs.createReadStream(path);
stream.pipe(res);
}
```

Output:

```
>node videoStream.js
```

Open the browser and access localhost:8080

4) Demonstrate the use of Standard streams.

```
//stdstreams.js
process.stdin.on("data", data => {
data = data.toString().toUpperCase()
process.stdout.write(data + "\n")
```

})

process.stderr.write("error! some error occurred\n")

Output:

>node stdstreams.js

the script will wait for the user input and print it in uppercase. Also it throws the above error message.

5) Create a new project and Install the express node module.

a) Create a new folder

b) Run >npm init within the folder

c) install the express module: >npm install express

6) Import the express module and create the express app object. Run the server on port no 8080.

//index.js file

```
const fs = require ('fs'); // built-in module
const express = require ('express'); // express framework
const app = express(); // create the express app obj

app.listen(8080, ()=>{ console.log('server started');});
```

7) Define routes for the following endpoints:

SN	Endpoint	Request
1	/	GET
2	/user	GET
3	/login	POST
4	/intro	GET
5	/video	GET

a) Demonstrate the use of header parameters on the '/' endpoint using a GET request.

```
// access User-Agent request header
app.get('/', function (req, res){
console.log(req.url);
console.log(req.headers['user-agent']);
res.end('From Express');
})
```

b) Demonstrate the use of query string parameters on the '/' endpoint using a GET request.

```
//access the query-string parameter
app.get('/user', function (req, res){
console.log(req.query.name);
res.end(req.query.name);
})
```

c) Demonstrate the use of route parameters on the '/user' endpoint using a GET request.

```
// access the route parameters (path params)
app.get('/user/:uid', function(req, res){
console.log(req.params.uid);
res.end('seeking user details' );
})
```

d) Demonstrate sending the body parameters on the '/login' endpoint using a POST request.

```
//access the body parameters

app.post('/login', function(req, res){
console.log(req.body.name);
res.end(req.body.name);
})
```

e) Serve the video.html file on the '/intro' endpoint using sendFile() method.

```
// serve the video.html file
app.get('/intro', function(req, res){
console.log('intro page requested');
res.sendFile(__dirname + '/video.html');
})
```

f) Stream a video in chunks on the '/video' endpoint.

```
// stream a video in chunks
app.get('/video', function(req, res){
const range = req.headers.range;
const videoPath = __dirname + '/sample.mp4';
const videoSize = fs.statSync(videoPath).size;
const chunkSize = 1 * 1e6;
const start = Number(range.replace(/\D/g, ""));
const end = Math.min(start + chunkSize, videoSize - 1);
const contentLength = end - start + 1;
//console.log(videoSize);
const headers = {
"Content-Range": `bytes ${start}-${end}/${videoSize}`,
"Accept-Ranges": "bytes",
"Content-Length": contentLength,
"Content-Type": "video/mp4"
}
//console.log(contentLength);
//console.log(headers);
res.writeHead(206, headers);
const stream = fs.createReadStream(videoPath, {'start': start,
'end': end})
stream.on('error', (err)=> {
console.log(err.message);
})
stream.pipe(res);
})
```

Output:

Run the server script as follows:

```
>node index.js
```

make multiple requests (GET and POST) on the endpoints as stated above

8) Install the express-generator module globally and initiate a new project.

>npm install -g express-generator

9) Setup the new project and run it.

>express app_name

>npm install

>npm start

10) Test a route and also supply image and icon files.

Copy the favicon.ico file to the 'public' folder.

Access the server on localhost and check if the icon is rendered.

localhost:3000

Copy an image to the 'public/images' folder and check if the image is accessible

localhost:3000/images/img3.jpg