Date: 13 June 2024 **Day: Thursday**

Overview:

Day 7 of the internship focused on exploring streams in Node.js, a powerful feature for handling data efficiently and asynchronously. Streams facilitate the processing of large datasets without consuming excessive memory, making them essential for tasks like file I/O and network communication.

Learning Objectives:

- **Readable and Writable Streams:**
    - Explored the concept of readable streams for reading data chunk by chunk from a source.
    - Implemented writable streams for writing data chunk by chunk to a destination, such as files or network sockets.
    - Learned about duplex and transform streams for bidirectional data flow and data manipulation during streaming operations.

Stream > JS Readable.js > ...

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

const stream = fs.createReadStream("../text.txt",'utf-8')

stream.on('data',(chunk)=>{
    console.log(chunk)
})
stream.on('end',()=>{
    console.log('finished')
})
```

- **Streaming Operations:**
    - Practiced streaming operations to process data efficiently without loading entire datasets into memory.
    - Utilized event listeners (data, end, error) and methods (pipe(), write(), end()) to manage stream behavior and data flow.
    - Implemented error handling strategies to manage exceptions during stream processing, ensuring robust application behavior.

Stream > JS Pipes.js > ...

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

const Readstream = fs.createReadStream("../text.txt",'utf-8')
const Writestream = fs.createWriteStream('./text_Pipe_stream.txt')

// Readstream.on('data',(chunk)=>{
//     Writestream.write(chunk)
// })
// Readstream.on('end',()=>{
//     Writestream.end()
// })

Readstream.pipe(Writestream).on('error',(err)=>console.log(err))
Writestream.on('close',()=>{
    process.stdout.write('file copied \n')
})
```

- **Piping Streams:**
    - Explored the pipe() method to connect readable streams to writable streams, facilitating seamless data transfer.
    - Implemented stream pipelines to process data from sources such as files, HTTP requests, or databases to destinations efficiently.
    - Optimized stream pipelines for performance by managing backpressure and ensuring data integrity across asynchronous operations.

```
Stream > JS Writeable.js > ...
const fs = require('fs');

const Readstream = fs.createReadStream("../text.txt",'utf-8')
const Writestream = fs.createWriteStream('./text_stream.txt')

Readstream.on('data',(chunk)=>{
    Writestream.write(chunk)
})
Readstream.on('end',()=>{
    Writestream.end()
})
Writestream.on('close',()=>{
    process.stdout.write('file copied \n')
})
```

Activities and Insights:

- **Implementation of Stream Operations:**
    - Created scripts to read from and write to files using streams, demonstrating the efficiency of streaming over traditional file operations.
    - Tested streaming operations with various data types and sizes to evaluate performance and resource utilization.
- **Error Handling and Event Management:**
    - Implemented error handling techniques using error events and try-catch blocks to manage exceptions during stream processing effectively.
    - Utilized event listeners (data, end) to orchestrate stream operations and ensure data completeness and integrity.