MODULE 04

NODE.JS AND MONGODB

ASSIGNMENT 01

Advanced Node.js Concepts Cheat Sheet

Introduction

Node.js is a powerful runtime environment that allows developers to run JavaScript on the server side. Its non-blocking, event-driven architecture makes it particularly well-suited for building scalable network applications. This cheat sheet summarizes key advanced concepts in Node.js, providing a comprehensive reference for developers looking to deepen their understanding and enhance their applications.

1. Asynchronous Programming

Overview

Asynchronous programming is fundamental to Node.js, enabling it to handle multiple operations simultaneously without blocking the execution thread. This is crucial for building responsive applications.

Key Components

- **Event Loop:** The event loop is the mechanism that allows Node.js to perform non-blocking I/O operations. It continuously checks the call stack and the callback queue, executing callbacks when the call stack is empty.
- **Callbacks:** Functions that are passed as arguments to other functions. They are executed after the completion of an asynchronous operation, allowing for handling results or errors.

javascript
```
fs.readFile ("file.text , (err , data) => {
    if (err) throw err;
    console.Log (data) ;
});
```

- Promises: An improvement over callbacks, promises represent a value that may be avaible now, or in the future, or never. They can be in one of three states: pending, fulfilled, or rejected.

javascript
```
const readFilePromise = fs.promises.readFile('file.txt');
readFilePromise
    .then (data => console.log(data))
    .catch (err => console.error(erri));
```

- Async/Await: A syntactic sugar built on promises that allows writing asynchronous code in a synchronous style, improving readability.

javascript
```
async function readFile() {
  try {
     const data = await fs.promises.readFile('file.text');
      console.log(data);
   } catch (err) {
            console.error(err);
}  }
```

2. Streams

Definition

Streams are objects that allow reading data from a source or writing data to a destination in a continuous manner. They are particularly useful for processing large amounts of data efficiently.

Types of Streams

- Readable Streams: Allow data to be read from a source (eg: files, HTTP requests).

  For example:

  javascript
  ```
  const readableStream = fs.createReadStream('file.txt');
  readableStream.on('data', chunk => {
        console.log('Received ${chunk.length} bytes of data.');
  });
  ```

Name - Aarjaw Jain
B.Tech. [2nd Year] Student

- Writable Streams: Allow data to be written to a destination (eg., files, HTTP responses).

javascript
```
const WritableStream = fs.createWriteStream('output.txt');
WritableStream.write('Hello, World!');
WritableStream.end();
```

- Duplex Streams: Can both read and write data (eg. TCP sockets).
- Transform Streams: Modify data as it is written and read (eg: compression).

3. Event Loop

- Functionality: Core mechanism enabling non-blocking I/O operations.
- Execution Phases:
  - Timers Phase. Executes callbacks from 'setTimeout' and 'setInterval'.
  - I/O Callbacks Phase: Processes I/O events.
  - Poll Phase: Retrieves new I/O events; executes their callbacks.
  - Check Phase: Executes callbacks from 'setImmediate'.

4. Child Processes

- Purpose: Allows parallel execution of tasks by creating child processes.
- Methods:
  - 'child_process.exec()': Runs command in shell; buffers output.
  - 'child_process.spawn()': Launches a new process with a command.
  - 'child_process.fork()': Creates a new Node.js process for IPC communication.

5. Cluster Module

- Definition: Facilitates Load balancing across multiple CPU cores by creating worker processes.
- Benefits:
  - Enhances performance through parallel processing.
  - Provides fault experience tolerance; if one worker fails, others continue functioning.

6. Debugging Tools; and Profiling Tools

- Debugging Tools:
  - Built-in debugger accessed via 'node inspect'.
  - Chrome DevTools can be used with the '--inspect' flag for debugging.

- Profiling Tools:
  - Use packages like 'v8-profiler' to analyze performance and memory usage.

7. Security Best Practices

- Validate user input to prevent injection attacks.
- Utilize HTTPS to secure data transmission.
- Regularly update dependencies to fix vulnerabilities.

8. Scalability and Performance Optimization

- Techniques:
  - Implement caching (eg: Redis) to reduce database load.
  - Optimize database queries and use indexing in MongoDB for efficiency.
  - Employ clustering and worker threads for horizontal scaling.

♦ Conclusion

This cheat sheet provides a concise overview of advanced Node.js concepts, essential for building efficient and scalable applications. Understanding these concepts will enhance your ability to develop robust server-side solutions.