Advanced Node.js Cheat Sheet

Asynchronous Programming

1. Promises and Async/Await

Promises:

```
Const myPromise = new Promise((resolve, reject)
  => { // Asynchronous operation
  if (success) {
     resolve(result); }
  else { reject(error); } });

myPromise.then(result => {
    // handle success }).catch
  (error => { // handle error
  });
```

Async/wait:

```
async function myfunction() {
try {
Const result = await someAsyncfunction
  (); } catch (error) {
  } }
```

2. Callbacks

```
functions asyncOperation(callback)
{ setTimeout() => { {
  Callback(null, 'result');
   }, 1000);
  }
asyncOperation((error, result) => {
  if (error) {} else {
     } });
  }
```

Performance Optimization

1. Clustering

Utilize multiple CPU ~~Cores~~ with cluster module.

```
Const cluster = require('cluster');
Const http = require('http');
Const numCPUs = require('os').cpus().length;
if (cluster.isMaster) {
   for (let i = 0; i < numCPUs; i++) {
   cluster.fork();
   }
cluster.on('exit', (worker, code,
    signal) => {
Console.log(`Worker ${worker.
process.pid} died`); });
} else {
http.createServer((req, res) => {
 res.writeHead(200);
 res.end('hello world\n');
 }).listen(8000);
```

2. Stream Processing

Use Stream for efficient data processing:

```
Const fs = require('fs');
Const readStream = fs.createReadStream('largefile.txt');
Const writeStream = fs.createWriteStream('output.txt');
readStream.pipe(writeStream);
```

Security Best Practices

① Secure Dependencies use tools like npm audit and snyk to check for vulnerabilities:

```
npm audit
# or
snyk test
```

② Environment Variables

Avoid hardcoding sensitive information:

```
require('dotenv').config();
const db = process.env.DB_PASSWORD;
```

③ Helmet

use helmet to secure HTTP headers:

```
const hel = require('helmet');
const exp = require('exp');
const app = exp();
Testing app.use(helmet());
```

① Unit Testing with Mocha & Chai:

```
const chai = require('chai');
const expect = chai.expect;
describe('Array', function() {
it('Should start empty', function() {
  const arr = [];
expect(arr).to.be.an('array').
that.is.empty;
});
});
```

2. Integration testing with Supertest:

```
const req = require('supertest');
const app = require('./app');
describe('GET /', function() {
  it('responds with json',
    function(done) {
  request(app).get('/')
  .set('Accept', 'application/json')
  .expect('Content-Type', /json/)
  .expect(200, done);
  });
});
```

Debugging

① Debugging with Node Inspector

```
node --inspect -brk=9229
.ola.js
# Open chrome://inspect in
    chrome Browser
```

② Using Console

Effective use of console.log, console.error, & console.trace

```
console.log('Debug message');
console.error('Error sms');
console.trace('Trace sms');
```

File System Operations

① Reading Files Asynchronously

```
const fs = require('fs');
fs.readFile('example.txt', 'utf8',
  (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

② Writing Files Asynchronously

```
const fs = require('fs');
const data = 'New data';
fs.writeFile('example.txt', data
  'utf8' (err) => {
  if (err) throw err;
  console.log('File Has');
});
```

Networking

① Creating an HTTP Server

```
Const Http = require('http');
Const server = http.create
        Server((req, res) => {
res.statusCode = 200;
res.setHeader('Content-Type', 'text/plain');
res.end('Hello world');
});
server.listen(3000, '127.0.0.1', () =>
{
Console.log("Server is running
at http://127.0.0.1:3000/");
});
```

② WebSockets with ws

```
Const WebSocket = require
        ('ws');
const wss = new WebSocket
        .Server({port: 8080});
Console.log('Running...');
});
ws.send('something');
});
```

Miscellaneous

① Using buffers

```
Const buf = Buffer.from
    ('Hello world', 'utf8');
Console.log(buf.toString('hex'));
```

② Child Processing

Spawning a new process

```
Const {spawn} = require
        ('cp');
Const ls = spawn('ls', ['-lh', '/u
ls.stdout('data', (data) => {
Console.log(stout ${data});
});
ls.on('close', (code) => {
console.log(child
process exited
with code ${code});
});
```