Node Js Cheatsheet

Events and Event Emitters: An event is an action or occurrence that can be observed and handled by the application.

* Events are typically represented as strings that indicate the type of event
* Event emitters are objects that can emit events and notify registered listeners when these events occur.

Event Loop: At the heart of Node JS's event driven architecture is the event loop.

* The event loop is a single-threaded mechanism that continuously checks for pending events and executes their associated event handlers.
* It ensures that the Node.JS application remains responsive and can handle multiple concurrent operations without blocking.

Asynchronous Programming in Node JS: Ability to execute multiple tasks concurrently without waiting for each task to finish before starting the next one. This is crucial for handling I/O bound operations like reading files, making network requests, or querying databases, where waiting for one operation to complete could significantly slow down the entire process.

Callbacks → They allow you to specify what should happen once an asynchronous operation completes. For example, `fs.readFile('file.txt', (err, data) => { })`, the callback function handles the file reading operation's result or error.

Async/await → Simplifies asynchronous code even further by allowing you to write asynchronous code in a synchronous like manner.

npm (Node Package Manager): Provides a vast ecosystem of reusable code (package/module) that developers can integrate into their projects. It simplifies dependency management, version control, and package installation, making it a cornerstone of node.js development.

- Package installation: npm allows developers to install packages from the npm registry using commands like 'npm install package-name'. This installs the specified package and its dependencies into the project's 'node_modules' directory.

Scripts: npm allows developers to define custom scripts in the 'package.json' file, such as 'npm start', 'npm test', or 'npm run build'.

Versioning and Updates: npm provides tools to manage package versions, update dependencies, and handle version conflicts. The 'npm update' and 'npm outdated' commands are useful.

CommonJS Module: It is a standard for organizing and structuring modular Javascript code. It is the module system used by NodeJS for modular development.

Module Definition

Exporting Modules → module.exports = function add(a,b) { return a+b }};

Importing Modules → const add = require('./add');

console.log(add(2,3));

Dependency Resolution → When you import a modules using 'require()', node.js resolves the modules path and loads its contents into the current module's scope:

Exporting multiple values → exports.add = function add(a,b){

return a+b;

};

exports.subtract = function substract(a,b){

return a-b;

};

Loading core Modules → NodeJS provides built in core modules that can be imported using 'require()'.

Middleware : Middleware functions that have access to the request object (req), the response object (res) and the next middleware function in the application's request-response cycle.

Application Level

```
app.use((req, res, next) => {
    console.log('Time', Date.now());
    next();
});
```

Router-level

```
const router = express.Router();
router.use((req, res, next) => {
    console.log('Request url:', req.originalUrl);
});
```

Error Handling

```
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).send("Error");
});
```

Performance Optimization:

Clustering

```
const cluster = require('cluster');
const http = require('http');

if (cluster.isMaster){
    const numCPUs = require('os').cpus().length;
    for (int i = 0; i < numCPUs; i++)
        cluster.fork();
}
else {
    http.createServer((req, res) => {
        res.writeHead(200);
        res.end('Hello');
    }).listen(8000);
}
```

Caching

```
const cache = {};
function getData (key){
    if (cache[key]){
        return Promise.resolve (cache[key]);
    }
    return fetchFromDatabase (key).then (data => {
        cache [key] = data;
        return data;
    });
}
```

Efficient I/O

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