Node.js Starter Notes

What is Node.js?

- **Definition**: Node.js is a JavaScript runtime built on Chrome's V8 engine that allows you to execute JavaScript code on the server side.
- **Non-blocking I/O**: It uses an event-driven, non-blocking I/O model, making it lightweight and efficient.
- **Single-threaded**: Despite being single-threaded, Node.js can handle many connections simultaneously.

Installation

1. **Download and Install**:
    - Visit the official Node.js website and download the latest version suitable for your OS.
    - Follow the installation instructions for your platform (Windows, macOS, Linux).
2. **Verify Installation**:

```
node -v      # Check Node.js version
npm -v       # Check npm version (Node Package Manager)
```

Basic Concepts

NPM (Node Package Manager)

- **Definition**: A package manager for Node.js, allowing you to install libraries and manage dependencies.
- **Common Commands**:
    - `npm init`: Create a `package.json` file.
    - `npm install <package>`: Install a package.

- `npm install -g <package>`: Install a package globally.

Creating Your First Node.js Application

1. **Set Up Project**:

```
mkdir my-node-app
cd my-node-app
npm init -y  # Initializes a new Node.js project with default settings
```

2. **Install Express** (Web framework):

```
npm install express
```

3. **Create a Basic Server**: Create a file named `app.js`:

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;

app.get('/', (req, res) => {
    res.send('Hello, World!');
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

4. **Run the Application**:

```
node app.js
```

5. **Access the Server**: Open your browser and go to `http://localhost:3000`. You should see "Hello, World!".

Common Commands

- `node <filename.js>`: Run a Node.js script.
- `npm start`: Start the application defined in the `package.json` under the `scripts` section.

Error Handling

- Use `try-catch` blocks to handle synchronous errors.
- Use `.catch()` method or `async/await` for handling promises.

Middleware

- Middleware functions are functions that have access to the request object, response object, and the next middleware function in the application's request-response cycle.

Conclusion

- Node.js is powerful for building scalable network applications.
- Explore more with databases (e.g., MongoDB), authentication, and routing.

Feel free to expand on any section as needed, and let me know if you have specific areas you'd like to dive deeper into!