swipe

Node JS

best practices

swipe

1. Always create a new project with npm init and save the exact package version

```
$ mkdir my-new-project

$ cd my-new-project

$ npm init
```

Use npm install — save afterward to install a package and save it as a dependency in the package.json file.

For example, if you use the package express it will be written as `$ npm install express – save`

It saves the package with its installed version to the package.json file like this. `“express”: “⁴.17.1”,`

Here the leading carriage “^” represents that a package with a version within “>=4.17.1<5.0.0” is accepted by the application.

Consequently, when someone in your team runs npm install, it will install the latest version of that package. If the latest version is not compatible with the initial version of the package it can lead to differences in the behavior or errors.

Therefore, for the development team, it is important to be on the same version. This can be achieved by using the .npmrc file. It has useful properties that ensure npm install always updates the package.json and enforces the version to be the same.

This can be simply done by running the following command in the terminal

```
npm install express — save — save-exact

Or

npm config set save-exact=true
```

Once you run the above command the dependency is saved and will be locked down to the version you installed.

swipe

2. Add Script

NPM has a standard way to start node applications. Add the script property and object to your package.json with a start key. As shown below,

```
“scripts”: {

“start”: “node myapp.js”

}
```

When someone runs npm start, NPM runs node myapp.js with all the dependencies from node_modules/.bin on your $PATH. Therefore there is no need to have global installs of NPM modules.

3. Use Environment Variables

Use environment variables in Node.js to look up the values from the process.env in your app code. To figure out which environment variables you're running on, check the NODE_ENV environment variable.

```
console.log("Running in :" + process.env.NODE_ENV);
```

4. Use a style guide

If you are developing a project in a team, there will inevitably be differences in the coding style of each developer. Most developers prefer the particular style guide to make code readable.
Unfortunately, if you need to work on code with a different coding style, you will end up reforming the position of braces, spaces, and tabs manually for hours. So to remain productive, the best option for a team is to pick a particular style guide and follow the same for the entire project.

Here are a few style guides used by popular tech companies.

- *Airbnb's style guide on GitHub*
- *Google's style guide on GitHub*
- *Idiomatic style guide on GitHub*
- *JavaScript Standard style guide on GitHub*

Also, some tools can be used to ensure the rules of the selected style guide are enforced.

- *Prettier*
- *ESLint*
- *Automate Format and Lint on Save*
- *Husky*

5. Asynchronous Structures

The synchronous function makes the flow of application logic easy to understand. However, it blocks any other code from running until it gets complete.

You can trace the synchronous function in the code using the -trace-sync-io flag, it will display a warning when encountered with the synchronous API.

6. Error Handling

Neglecting error handling in the code makes debugging more difficult than it should be. Having a single bug in the application can cost you billions. Therefore good exception management is important for an application. The best way to deal with errors is by using the .catch() handler, which will propagate all errors to be dealt with, cleanly.

7. Avoid Garbage collection using V8 Engine flags.

Node (V8) manages the heap memory through garbage collection. It frees up the memory used by objects that are no longer referenced from the Stack, to make space for the new object creation.

The problem can be overcome by regulating the app's Garbage collector, you can raise V8 engine flags and chrome debugger to expose the garbage collector for debugging memory issues.

```
node – expose-gc – inspect myapp.js
```

swipe

8. Keep your application stateless

If you stored the data such as sessions, user data, cache in the application itself it will be scoped to that particular process. Therefore you need to store it on the external data stores.

Keeping your application stateless enables the application to survive system failures without damaging its services and performance.

To accomplish this, you can use serverless platforms like AWS Lambda that impose stateless behavior by default.

9. Use logging tools

Console.log is a great tool but has its limitations in a production application therefore you can not use it for every logging purpose. It does not provide enough configuration options, for example, there is no filter option to filter loggings.

Some good examples of logging frameworks are Bunyan, Winston, and Pino. It makes Node.js logging simpler and more efficient.

10. Test your application

It is crucial to test your application before launching it in the market. No matter what stage of your application development, it is never too late to introduce testing.

Experts recommend writing a test for every bug that gets reported. Therefore you must know:

- How to recreate the bug (make sure your test fails first!)
- How to fix the bug (make sure you pass the test once the bug is fixed)
- Make sure that the bug will never occur again.

Following are the most popular testing Libraries for Node.js applications

- Mocha

- Jest

- Jasmine