**COMPILING - TRANSPILING**

Contents

[1. **WEBPACK** 2](#_Toc162425384)

[ **Entry:** 2](#_Toc162425385)

[ **Output:** 3](#_Toc162425386)

[ **Loaders:** 3](#_Toc162425387)

[ **Plugins:** 4](#_Toc162425388)

[ **Mode:** 4](#_Toc162425389)

[ **Hash:** 4](#_Toc162425390)

[ **Square brackets / Template strings** 4](#_Toc162425391)

[ **Resolve:** 5](#_Toc162425392)

[ **DevServer** 6](#_Toc162425393)

[ **Code Splitting** 6](#_Toc162425394)

[**2.** **npm: dependencies vs devDependencies** 7](#_Toc162425395)

[**3.** **Tsconfig.json** 7](#_Toc162425396)

[**4.** **Tree Shaking** 8](#_Toc162425397)

[**5.** **Process ko dc dùng trong webpack 5 nữa** 9](#_Toc162425398)

[**6.** **Webpack - Hot module reload(HMR)** 10](#_Toc162425399)

[**7.** **React-Hot-Loader** 11](#_Toc162425400)

[**8.** **Tsc ts-node typescript compling** 13](#_Toc162425401)

[**9.** **Yarn vs npm** 13](#_Toc162425402)

[**10.** **Quá trình tối ưu Module Task** 13](#_Toc162425403)

[**11.** **ViteJS** 15](#_Toc162425404)

[**12.** **CDN vs NPM** 15](#_Toc162425405)

1. **WEBPACK**

Webpack là một **module bundler.** Nó giúp gom tất cả các tệp JavaScript, CSS, hình ảnh, font chữ,… thành một hoặc nhiều file tối ưu để tăng hiệu suất và quản lý dự án dễ dàng hơn.

* **Entry:**
* the entry point is the module that webpack uses to start building its internal dependency graph.
* Thông thường sẽ trỏ đến src folder.
* By default: ./src/index.js
* Thêm vendor file: có thể thêm **nhiều entry** vào dạng object(preferred)
* [name] output.filename sẽ lấy từ key của entry



* Hoặc có thể thêm dạng array
* Vì những file này ko phụ thuộc lẫn nhau, nên có thể dùng array
* Báo có webpack bít là sẽ append **file\_2.js** vào cuối vào **bundle.js**

module.exports = {

entry: ['./src/file\_1.js', './src/file\_2.js'],

output: {

filename: 'bundle.js',

},

};

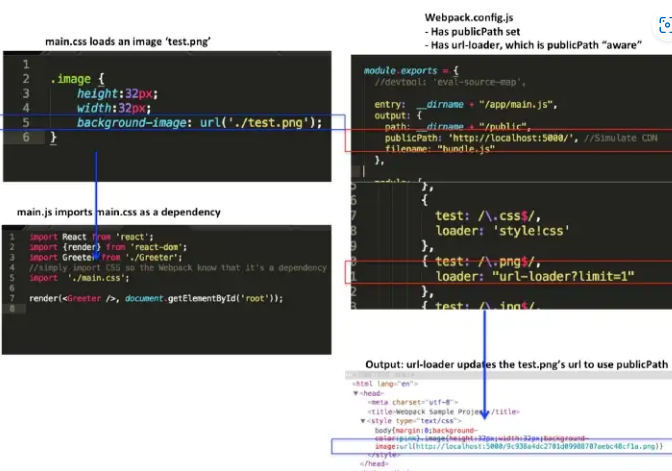
* Có thể kết hợp object và array cho entry



* **Output:**
* the output property instructs webpack where to emit the bundle(s) and what name to use for the file(s).
* Đầu ra của webpack. Nơi để nó emit cái file bundle ra.
* By default: ./dist/main.js for the main bundle
* and ./dist for other generated files — such as images



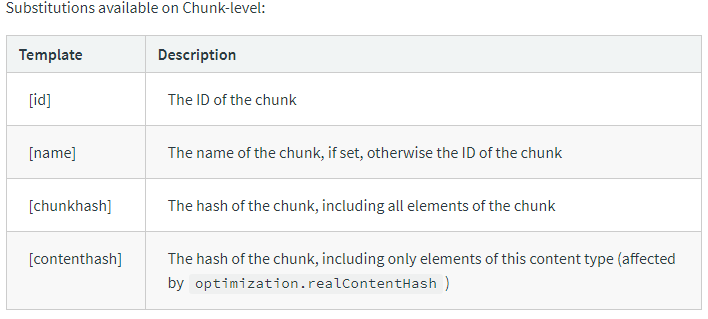
* 2 props chính là filename, path
* **Filename**: tên file cần xuất ra
* **path**: đường dẫn tuyệt đối và tên thư mục webpack sẽ bundle ra.
* **publicPath**: để plugin cập nhất đường dẫn url trong file html,css
  + vd: define output.publicPath để url-loader bít và thêm vào phần phía trước

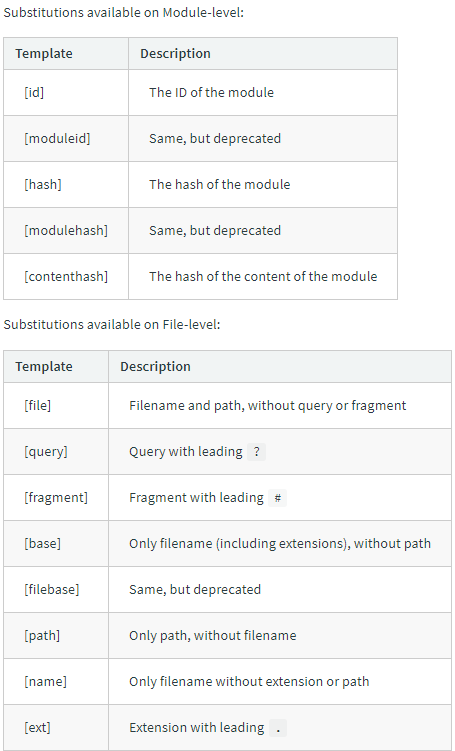


* **Loaders:**
* Là cái sẽ đc đưa vào **trước** khi compile 1 file (Pre-process).
* **Loaders** work at the individual file level **during** or **before** the bundle is generated
* Thường là sẽ cần loaders cho những file non-js. Và chuyển nó thành dạng module vd: Typescript language to JavaScript, inline images, import CSS files directly from our JavaScript
* Theo webpack: mún process thì tất cả các file là dạng module và phải là ngôn ngữ js
* Cách viết rules loaders trong webpack: vì tất cả là module nên có thể thấy là module: {} và bên trong sẽ là rules

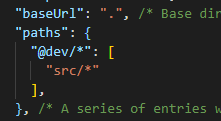


* **use**: 1 array và là nơi khai báo các loaders cần thiết.
* **Note**: nó sẽ chạy kiểu pipeline **right to left / bottom to top**)
* **Use**: các loaders có thể có **options object**: use: [ {loaders:””, options: {...} }]
* **Loader** chỉ dùng khi đi qua từng file
* **Plugins:**
* **plugins** are used for any other task that **loaders** can’t do (mạnh hơn loaders).
* **deeply integrate** into webpack because they can register hooks and access (and modify) the compiler
* **Plugins** có thể tích hợp sâu vào bất cứ quá trình nào khi bundle
* Vd: asset management, bundle minimization, optimization
* Thường chạy ở *cuối quá trình* **bundle** hoặc **chunk**
* Các Plugins thông dụng:
  + HtmlWebpackPlugin (tạo file html khi bundle dựa trên [contentHash])
* **Mode:**
* Thường có 3 môi trường: **development, staging, production**.
* Trong webpack thì thông thường chỉ có 2: “**development”, “production”**
* This allows webpack to use built-in optimizations corresponding to each environment. Khi có define mode thì webpack sẽ dùng những cái có sẵn theo từng môi trường để tối ưu bundle.
* **Hash:**
* Dùng để cache lại các file đã build/bundle.
* Nếu có thay đổi trong file thì sẽ cache bursting (mất cache) và sẽ build lại file mới.
* **Square brackets / Template strings**
* Those are placeholders that Webpack will replace with their actual value
* Thường dành cho phần **output**
* **[hash]:** The hash of the module identifier.
* **[contenthash]:** hashes generated from the generated content
* **[chunkhash]**: The hash of the chunk content.
* **[name]**: The module name.
* **[id]**: The module identifier.
* **[query]**: The module query, i.e., the string following ? in the filename.

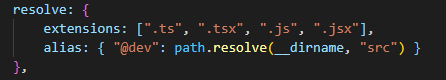




* **Resolve:**
* Tsconfig.json

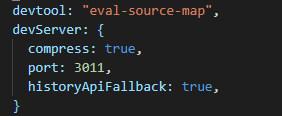


Tương ứng với alias webpack

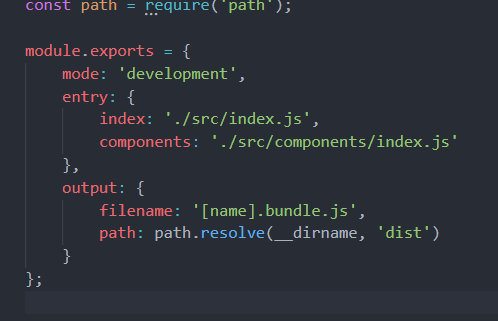


**Note**: cấu hình ở tsconfig sẽ hơi khác với resolve.alias vì webpack sẽ tìm đến path của folder (path.resolve) rồi tìm đến /src => ./src. Vậy có thể hiểu là: @dev = ./src. Với những case wildcard (\*) thì trong webpack ko dùng dấu \* nữa

* **DevServer**



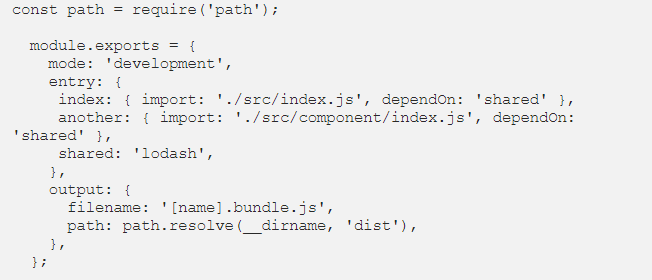
* **historyApiFallback** để khi text lên url thì sẽ theo cái router nếu ko sẽ bị case: cannot GET/URL
* **Code Splitting**

Có thể sử dụng nhiều đầu vào entry points để split code

SplitChunksPlugin:



Nó sẽ là cái built-in plugin, dùng optimization để chunk. Cái này là dùng với lodash là phù hợp

Dùng entry và thuộc tính dependOn

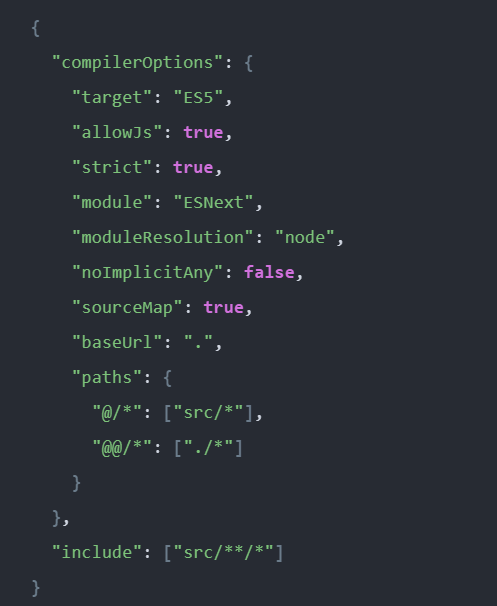
1. **npm: dependencies vs devDependencies**

[npm: dependencies vs devDependencies with bundled dependencies - Stack Overflow](https://stackoverflow.com/questions/36218100/npm-dependencies-vs-devdependencies-with-bundled-dependencies)

[Clarification on using dependencies or devDependencies · Issue #520 · webpack/webpack (github.com)](https://github.com/webpack/webpack/issues/520)

* nếu build app client thì ưu tiên đưa hết vào devDependencies. Vì app chỉ nhận file html,js( đã dc bundled). Nên ko quan tam đến các package. nó non-transitive
* Còn nếu mún tạo package thì đưa vào dependencies vì cần phải run các package này. Và nó có tính transitive: A depends B, B depends C => B,C đều dc install
* A browser app built by webpack has no runtime node dependencies. Ko cần runtime dependencies nên ko cần đưa vào dependencies trong package.json

1. **Tsconfig.json**

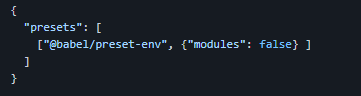


1 số lưu ý với file tsconfig.json:

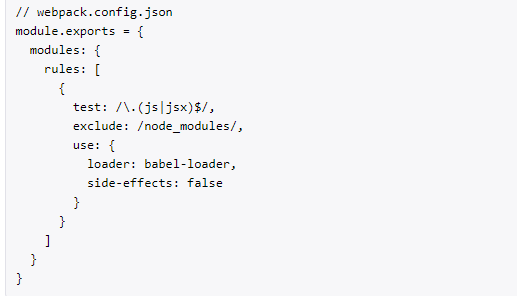
* **module**: sau khi biên dịch ra mã javascript thì mã này được viết dưới dạng module **ESNext**. Riêng option này có rất nhiều tùy chọn, khuyên anh em không nên chọn **CommonJs**, vì nó sẽ làm mất đi tính năng **Tree-Shaking** của webpack (Tính năng **Tree-Shaking** chỉ hoạt động với code dùng cú pháp [**ES Module**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules) (**import export** ), còn lại thì sẽ không hoạt động nhé.).
* **sourceMap**: cho phép hiện souremap TS (sourcemap với TS thì bạn phải mở trong **tsconfig.json** và trong **webpack** mới đầy đủ)
* **baseUrl**: đường dẫn cơ sở, thường là “.”. Nếu bạn dùng option **paths**  thì phải quy định **baseUrl**
* **paths**: tạo alias để thuận tiện việc import. Ví dụ thay vì bạn dùng ../../../ thì bây giờ bạn có thể rút ngắn lại thành @/. Việc cấu hình alias ở **tsconfig.json** chỉ giúp editor code hiểu, nó không có tác dụng với **webpack**. Vì thế bạn phải cấu hình với **alias với webpack** nữa.

1. **Tree Shaking**

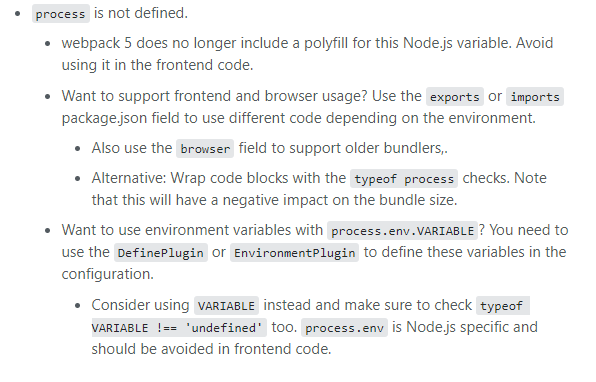
* **Tree Shaking** được thực thi ở webpack – lúc bundling
* Để webpack có thể Tree Shaking thì **phải để module ở dạng ESModule**.
* Babel, tsconfig **ko dc transpile về commonJS** (vì commonjs chỉ có require() thôi)
* Để **prevent Babel** transpile to **commonJS**



* **Loại bỏ side-effect – vì webpack ko bít là cái var đó có dc sử dụng hay ko => sẽ để lại.**



1. **Process ko dc dùng trong webpack 5 nữa**

- ý nói là biến process này chỉ được dùng cho node.js – phía Back-End chứ ko được dùng ở client – phía Front-End

- mún dùng thì phải dùng plugin **DefinePlugin** hoặc **EnvironmentPlugin**



Dùng 1 trong 2 cách trên là được

Note: nhớ phải install thêm **package** process nữa.

**webpack.common.js**

plugins: [

        new webpack.ProvidePlugin({

            process: 'process/browser',

        })

    ],

**webpack.dev.js**

plugins: [

    new webpack.DefinePlugin({

      'process.env.ENVIRONMENT': JSON.stringify('dev')

    }),

  ],

**webpack.prod.js**

plugins: [

    new webpack.DefinePlugin({

      'process.env.ENVIRONMENT': JSON.stringify('prod')

    }),

  ],

Và tương tự nếu như có thêm các môi trường khác

1. **Webpack - Hot module reload(HMR)**

* Dùng để reload lại trang khi có thay đổi về code(khi nhấn **Ctrl + S**). Mà ko cần phải full load lại cả trang (chú ý ở favicon).
* ở **webpack 5** khi start **webpack-dev-server** thì **mặc định là bật hot reload** rồi.
* Cần phải cấu hình ở **file** **entry point**  của project(project hiện tại index.tsx)

Index.tsx

// Allow Hot Module Replacement

if (module.hot) {

    module.hot.accept()

}

Index.tsx (Cấu hình cụ thể hơn)

function renderApp(*Component*: React.ReactElement) {

    ReactDOM.render(*Component*, document.getElementById('root'));

}

renderApp(<App />);

// Allow Hot Module Replacement

if (module.hot) {

    module.hot.accept('./App', () => {

         // timeStart

        renderApp(<App />);

// timeEnd

    })

}

Note: cách này có thể thêm cái **timeStamp để biết dc bundled** tốn bao lâu và cách này sẽ ko tính file **entry point** này

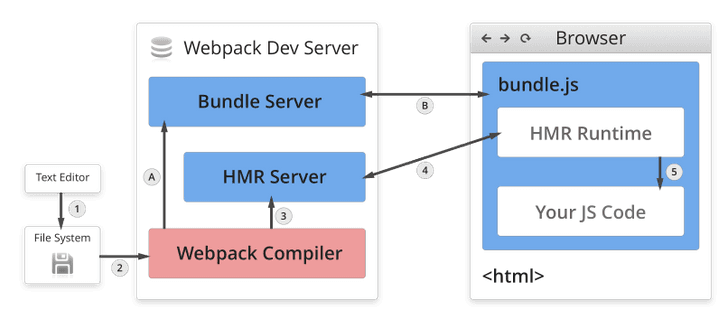
Note: **ts** sẽ ko biết dc module có property hot nên phải cài thêm package **@types/webpack-env.**

* Và trong tsconfig.json cấu hình

"types": ["webpack-env"],

* Sơ đồ flow của HMR

[Understanding Webpack HMR (javascriptstuff.com)](https://www.javascriptstuff.com/understanding-hmr/)



**Launch Flow**: lúc start server

1 > 2 > 3 > A > B

1. **Webpack Compiler** compiles your JS code.
2. **Bundle Server** serves your bundle.js to the browser.

**Update Flow :** Lúc có thay đổi trong file (Ctrl + S)

1. You make a **change** to one of **your text files in text editor**.
2. **File System** picks up the change and notifies **Webpack**.
3. **Webpack Compiler** rebuilds one or more modules and notifies the **HMR Server** that an update has been made.
4. **HMR Server** uses websockets to notify the **HMR Runtime** that it needs an update. **HMR Runtime** requests those updates over HTTP .
5. **HMR Runtime** replaces the modules in the update - OR - if it determines those modules can't be updated
6. **React-Hot-Loader**

* Dùng để reload lại page khi **Ctrl + S** mà ko bị mất state (HMR thì ko giữ dc state)
* Package cần **@hot-loader/react-dom react-hot-loader**

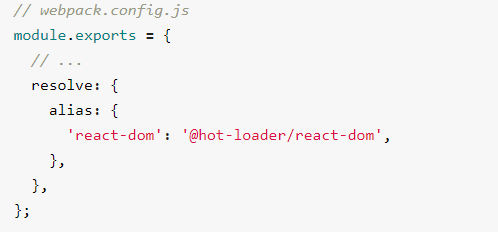


Thêm config như trên để hoạt động

Đối với cách cũ thì sẽ dùng **AppContainer** wrap lại <App />



Note: khi cài **@hot-loader/react-dom** thì phải thêm **config alias** ở **webpack**



1. **Tsc ts-node typescript compling**

Khi dùng paths, thì ts-node/tsc sẽ ko tự resolve paths đó, nên phải dùng mấy cái package tsconfig-paths (dev), tsc-alias(prod)

 "scripts": {

    "start": "nodemon -e ts,js --exec ts-node -r tsconfig-paths/register ./src/index.ts",

    "build": "tsc -p tsconfig.json && tsc-alias -p tsconfig.json"

  },

1. **Yarn vs npm**

**Yarn**

- these tasks are executed in **parallel** (download song song)

- caching mechanisms on disk so next installation of this package you don’t even need to have an internet connection.

\* Nên sẽ tốn dung lượng trong ổ đĩa

**Npm**

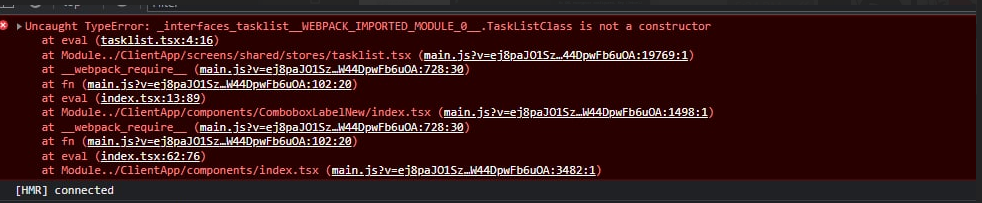
- these tasks are executed **per package and sequentially** (download tuần tự)

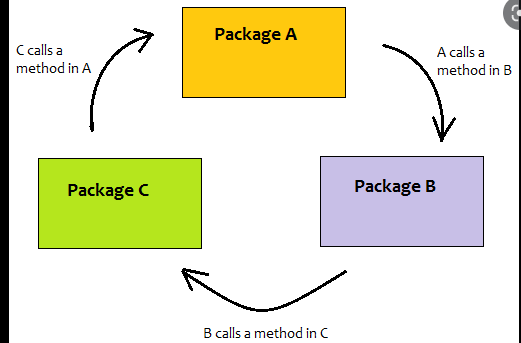
- no caching

1. **Quá trình tối ưu Module Task**

Chuyển sang dùng babel-loader, tsconfig set noEmit:true

* Bug: circular dependency



Do: 

circular-dependency-plugin

Fix: chuyển các component page sang react.lazy()

Lần đầu, chuyển bằng tay thì vẫn bị lỗi.

Lần 2, dùng lib react-lazily chuyển => works

* Bug: css

Lý do: dùng css của các component khác ,component đó là lazy nên chưa được load vào.

Fix: dùng sass-loader check đường dẫn của các file có đuôi .css và import vào style.ts

Minor bugs: có thể bị đè css ở đâu đó vì thứ tự đi theo file style.ts

* Bug: Mobx

Do: dùng cách viết cũ của Mobx nên ko khởi tạo dc class, nên ko observable dc

Fix: babel-loaders 

* Bug: export enum

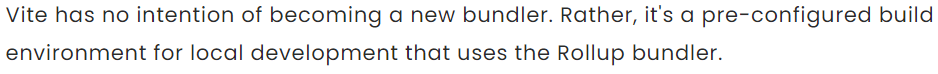
Lý do: đặt enum trong component lazy nên khi component khác import thì ko nhận dc.

Fix: hiện tại là đưa hết enum trong component ra ngoài file component lazy (vẫn đang làm)

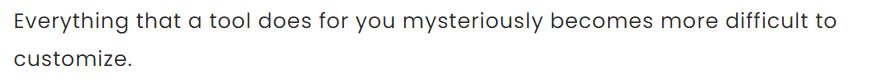
Minor bugs: trùng tên enum và phải thay đổi đường dẫn

* Note: có 85 warnings là do ko tìm thấy Interfaces, với interfaces thì ko cần move ngoài nhưng enum là phải đưa ra ngoài vì enum có giá trị

1. **ViteJS**



* Vite ko phải là 1 bundler. Mà chỉ là môi trường dc cấu hình cho việc dev ở local. Còn việc bundler là của Rollup



* Việc 1 cái tool làm hết mọi thứ cho bạn ở bên dưới sẽ trở nên khó khăn hơn để tùy chỉnh nó.
* Bản chất của ViteJS ở dev dùng es-build, ở pro dùng Rollup
* Nên có thể dùng plugin/loader của Rollup cho ViteJS

1. **CDN vs NPM**

|  |  |  |
| --- | --- | --- |
|  | CDN | NPM |
| Ưu điểm | * Lưu cached * Load nhanh * Thích hợp với các lib/package nhỏ * Làm demo, pet-project | * Tải về máy * Quản lý trong package.json * ổn định vì đã lưu trong máy * tự quản lý package |
| Nhược điểm | * Phụ thuộc vào server * Có thể chết server * File/data có thể bị remove hoặc bị thay đổi * Block user/ip hoặc nhà mạng block CDN * Có thể bị dính bản quyền, bảo mật | * Phải install vào trong node\_modules |