Chapter 02 Igniting our app

Tags

> Production environment is the one where your code or app is accessible to everyone eg. flipkart.com, amazon.in etc. and local environment is the one where we do our local development of the app eg. localhost:1234, localhost:4200 etc.

To make an app production ready, we should:

- minimfy our file(remove our console logs , bundle things up)
- need a swever to run things

Even though we can load our App.js we cant cant get optimised version

“Minify→Optimization→Clean console→ Bundle

WHAT IS PARCEL?

Parcel.js is an open-source bundler. It supports many popular languages like Typescript and SASS, and can also handle file types like images and fonts. Parcel comes with a few extra tools built-in: a development server, diagnostics, minification, and even image compression. If your project requires additional configurations, you have the option to use Parcel’s many plugins.

What is a JavaScript module bundler?

A bundler is a development tool that combines many JavaScript code files into a single one that is production-ready loadable in the browser. A fantastic feature of a bundler is that it generates a dependency graph as it traverses your first code files. This implies that beginning with the entry point you specified, the module bundler keeps track of both your source files’ dependencies and third-party dependencies. This dependency graph guarantees that all source and associated code files are kept up to date and error-free.

You can only imagine how complicated the procedure was before bundlers. Keeping all of the files and their dependencies up to date and ready was a herculean task for web developers.

> The create-react-app, the bundler used is “webpack”. Most bundlers do the same job. Bundlers are packages. If we want to use a package in our code , we have to use a ‘package manager’.
> Thus we use a “package manager” known as either npm or yarn.

Parcel Features:

- HMR (Hot Module Replacement): As you make changes to your code, Parcel automatically rebuilds the changed files and updates your app in the browser. By default, Parcel fully reloads the page, but in some cases it may perform Hot Module Replacement (HMR). HMR improves the development experience by updating modules in the browser at runtime without needing a whole page refresh. This means that application state can be retained as you change small things in your code. HMR KNOWS SOMETHING TO BE UPDATED USING FWA
- FWA (File Watcher Algorithm) : To support an optimal caching and development experience Parcel utilizes a very fast watcher written in C++ that integrates with low-level file watching functionality of each operating system. Using this watcher Parcel watches every file in your project root (including all `node_modules`). Based on events and metadata from these files, Parcel determines which files need to be rebuilt.

  > There will be a folder called parcel `cache`, which will be there automatically. In our project, parcel did some space so it creates `parcel – cache`

- Bundling
- Minify
- Cleaning our Code
- Dev and Production Build
- Super fast Build Algorithm

- Image Optimization
- Caching while development
- Compression
- Compatible with older version of browser
- HTTPS on dev
- port number
- Consistent Hashing Algorithms (To do all bundling)
- Zero Config (Other bundlers have much configuration)
- Browser List (How to make our app work on old browsers(IE))

> `dist` folder keeps the files minified for use

> When we run command:
>
> `npx parce index.html`
>
> This creates a faster development version of our project and serves it on the server.
> For parcel to make production build:
>
> `npx parcel build index.html`
>
> It creates a lot of things, minify your file and "parcel will build all the production files to the `dist` folder

Parcel also takes care of your older version of browser Eg: Sometimes we need to test our app on https because something only works on https. Parcel gives us the functionality to build our own app on https. on dev machine using:
`npx parcel index.html –https`

NPM

Dosent stand for node package manager, stand for any random name, i.e No official name

Since bundler is a package, we ise NPM.

It forms `package.json`

Why we use NPM? because react cant be just created by inserting react, it need many more packages and helper. So we use NPM

Dependencies

A dependency is just a package that your project uses

Very few javascript projects are entirely self-contained. When your project needs code from other projects in order to do its thing, those other projects are “dependencies”; your project depends on them to run.

```
npm install "packagename"
✅this command adds package to dependency
```

> When you install third-party packages via , you’re adding a dependency. Your project’s package.json file includes a list of your project’s dependencies.

In package.json file, there is an object called **dependencies** and it consists of all the packages that are used in the project with its version number. So, whenever you install any library that is required in your project that library you can find it in the dependencies object.

What are devDependencies?

devDependencies are modules which are only required during local development and testing, while dependencies are modules which are also required at runtime (that is during production)

> The production environment is where users access the final code after all of the updates and testing

devDependencies are packages used for development purposes,

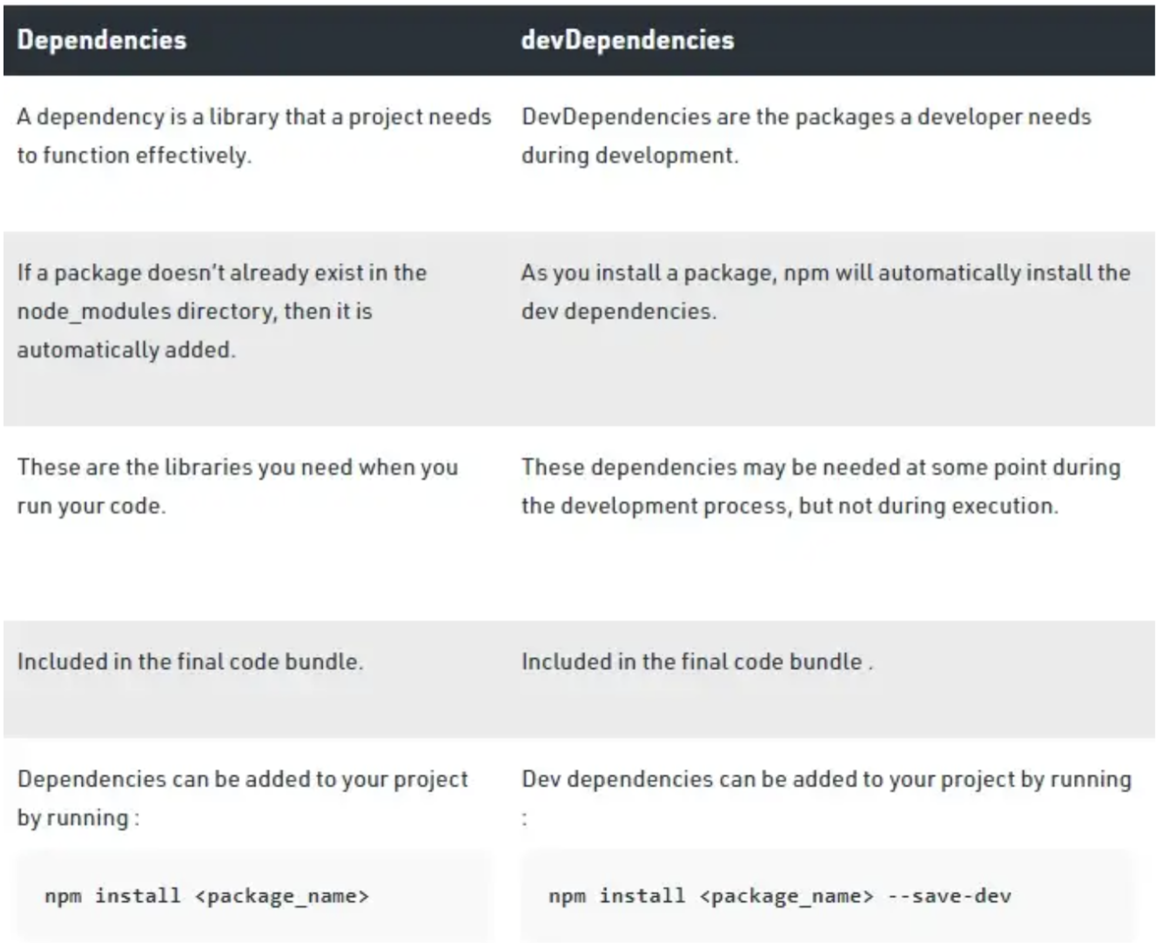
> e.g for running tests or transpiling your code. Many packages that you install during development are not required for your app to work in production — so we add those to our package.json devDependencies object.

In devDependencies, you’ll find different types of libraries such as:

- formatting libraries: eslint, prettier, …
- bundlers: webpack, gulp, parceljs, …
- babel and all its plugins
- everything related to tests: enzyme, jest, …
- a bunch of other libraries: storybook, react-styleguidist, husky, …

```
npm install "packagename" --save-dev
or
npm install -D "packagename"
✅This command adds the package to devDependency
```

In package.json file, there is an object called as **dev Dependencies** and it consists of all the packages that are used in the project in its development phase and not in the production or testing environment with its version number. So, whenever you want to install any library that is required only in your development phase then you can find it in the dev Dependencies object.



Dependencies	devDependencies
A dependency is a library that a project needs to function effectively.	DevDependencies are the packages a developer needs during development.
If a package doesn't already exist in the node_modules directory, then it is automatically added.	As you install a package, npm will automatically install the dev dependencies.
These are the libraries you need when you run your code.	These dependencies may be needed at some point during the development process, but not during execution.
Included in the final code bundle.	Included in the final code bundle .
Dependencies can be added to your project by running : `npm install <package_name>`	Dev dependencies can be added to your project by running : `npm install <package_name> --save-dev`

What's the difference between tilde(~) and caret(^) in package.json?

- `~version` **"Approximately equivalent to version"**, will update you to all future patch versions, without incrementing the minor version. `~1.2.3` will use releases from 1.2.3 to <1.3.0.
- `^version` **"Compatible with version"**, will update you to all future minor/patch versions, without incrementing the major version. `^2.3.4` will use releases from 2.3.4 to <3.0.0.

> Our project will automatically update if we use caret sign (^)

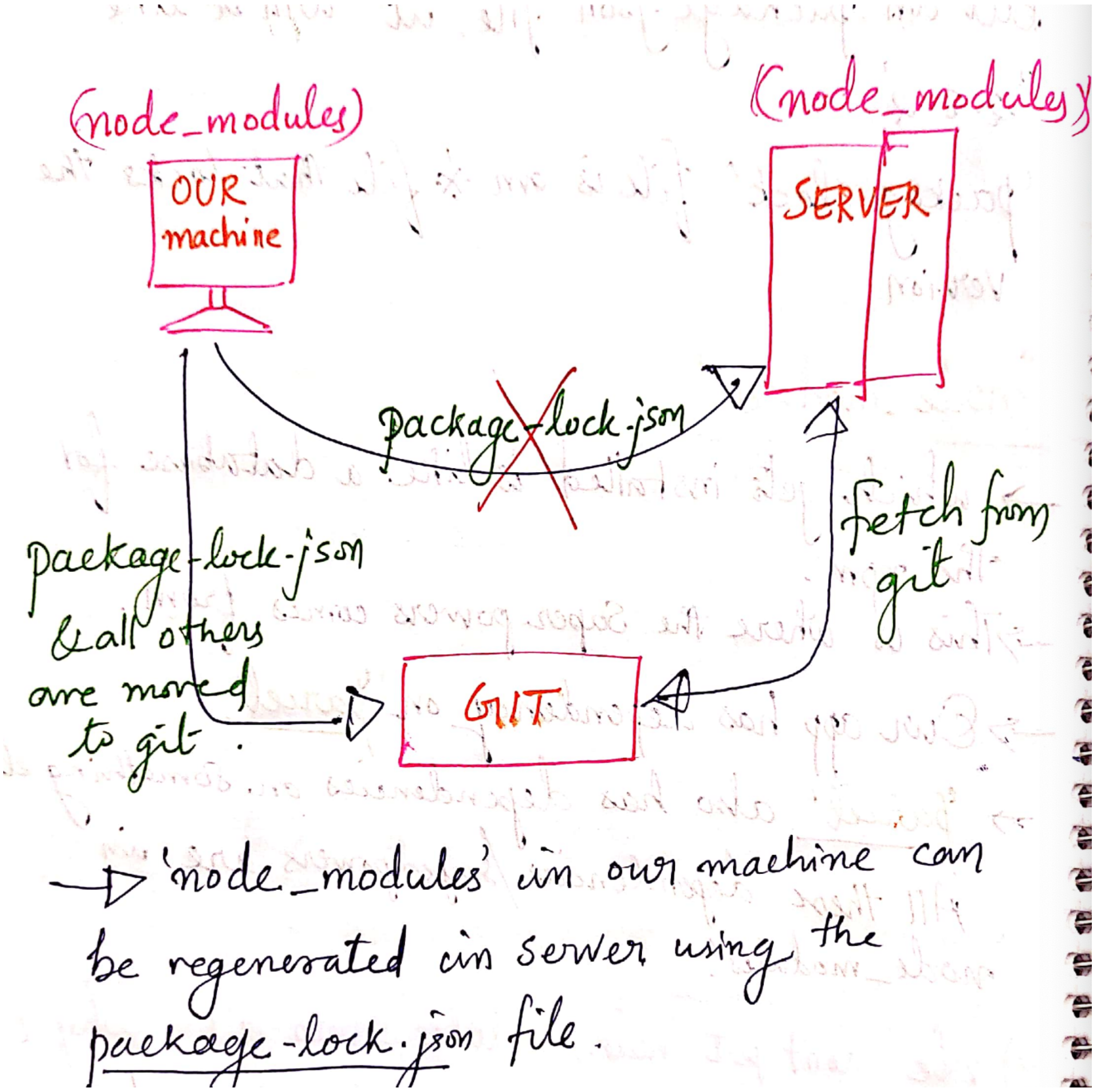
```
"deveDependencies":{
          "parcel":"^2.8.2"
}
```

💡 Package-lock.json will tell you which exact version of the library you are using currently

Node_modules

This folder gets installed, when we install react and other dependencies.

This folder acts as a database for npm. This is where the superpower comes from. Our app has dependency on parcel. Parcel also has dependencies on something else. All these dependencies/superpowers are in 'node-modules'.



→ 'node_modules' in our machine can be regenerated in server using the package-lock.json file.

→ We can use CDN links to get react into our app. This is not a good way. We need to keep react on our node-moudles

npm install react

→ To ignite our app

npx parce index.html

npx: execute using npm

index.html: is the entry point

After this a mininserver is created for us, like localhost:1234

→ As we removed cdn links, we dont have react in our app. So, we want to import it into our app.For that we use the keyword “import”

> Never touch ‘node_module’ & ‘package-lock.json’

Norman JS browser doesn't know import, so it shows error

```
import React from "react";
import ReactDOM from "react-dom/client"
```

As we got an error, we have to specify the browser that we are not using a normal `script` tag but a `module`

```
<script type="module" src="App.js"></script>
```

> We cannot import and export scripts inside a tag. Modules can import and export

💡 Anything which can be auto-generated should be put inside gitignore

How do I make our app compatible with older browser?

There is a package called browserlist and parcel automatically gives it to us. Browserlist makes our code compatible for lot of browsers:

go to:

Browserslist

A page to display compatible browsers from browserslist string.

https://browserslist.dev/

eg, In package.json file do:-

```
"browserlist":[
  "last 2 versions"
]
```

It indicates that parcel app will make sure that my app works in last 2 version of all the browsers available

Tree shaking

Tree shaking is a term commonly used within a JavaScript context to describe the removal of dead code.

It relies on the import and export statements to detect if code modules are exported and imported for use between JavaScript files.

In modern JavaScript applications, we use module bundlers (e.g., webpack or Rollup) to automatically remove dead code when bundling multiple JavaScript files into single files. This is important for preparing code that is production ready, for example with clean structures and minimal file size.