2nd / npm is a package manager where we install in system

DELTA Date 1 1 Pg No.

- package.json is a configuration for npm in form of JSON.
- sometimes these packages are known as dependencies.
- our project is dependent on lot of packages, npm will take care of what is the version of that package.
- It will take care in package.~~s~~ json

→ Bundler is a most important package in our project.

→ when we have normal HTML, CSS, JS file our whole codes needs to be bundle together, our whole code need to be minified, compressed, cached, cleaned before it can be sent to production, a bundler help us to do all that things.

→ Webpack, ~~parcel~~ parcel, Vite → these are bundlers

→ Bundler basically bundles the app, packages in app properly so that it can be shipped to production

Scanned with CamScanner

→ create-react-app → it behind the scenes uses webpack bundler.

→ parcel basically comes as a node package

→ npm install ~~parcel~~ -D parcel.

→ There are 2 types of dependencies which App can have
- Dev dependencies
- normal dependencies

→ Dev Dependency required in a development phase, when we are developing our app, then we need Dev Dependencies

→ Normal dependency are use in production also.

→ So the ~~use~~ use of parcel bundler (minification, chunking) is use in development phase, not in production, so we will install parcel as a dev dependency
- npm install -D parcel

→ ~~package.lock~~ package.json is a configuration for npm, it basically keeps a track of all the dependencies, packages codes are using.

Scanned with CamScanner

package.json -

"devDependencies": {
"parcel": "^2.8.3"

→ ^ → Caret
→ ~ → tilde

→ Suppose if tommorow a new minor version of parcel comes in → 2.8.4 is released; than what will happen if we put this caret (^) into our app, parcel will automatically be upgraded to 2.8.4

→ And if we put tilda (~), so it will install the major version automatically

ex → 3.0.0

→ Always it is safe to put a caret, because it's OK to upgrade the minor version but don't upgrade the major version, Sometimes when we have a major upgrade a lot of things breaks into our App.

→ When we install parcel, devDependencies were added over package.json but we also have got another file.

— package-lock.json.

Scanned with CamScanner

- This file keeps a track of exact version that is being installed.
suppose if today "2.8.5" released so our package.json

package-lock.json will stay like this, it locks the version and keeps a record of it, it ~~works~~ means the record of exact version of that package which is installed.

CX - "2.8.3"

---

package-lock.json →

"parcel": {

"version":

"resolved":

"integrity":

↓

This integrity typically a hash, this hash is present because package-lock.json keeps a hash to verify that whatever is there in my machine right now is the same version which is being deployed onto the production.

Scanned with CamScanner

DELTA
Date / /
Pg No.

package-lock.json

- It keeps record of all the version - exact version of all the dependencies.

→ One more thing created that is -

- node_modules

→ This node_modules contains all the code that we fetched from npm.

→ package.json is the configuration for our npm, node modules is like a kind of database - it contains the actual data of that dependencies & packages that project needs

→ But we just install parcel so there should only be a single package named parcel why there are so many package & dependencies there, while we didn't install them

→ It came over here because our project need parcel but parcel as a project has its own dependencies and those dependencies can have their own dependencies. This is known as transitive Dependency

→ Every dependency in node_modules have their own package.json and they have their own 'normal' & 'dev' dependencies

Scanned with CamScanner

DELTA
Date / /
Pg No.

- Therefore npm install ~~parcel~~ other dependencies along with parcel.

→ "Dependencies have their own ~~dependencies~~ dependencies".

→ Should I put all these node modules/ by whole code to github?

- NO, I don't want put all these, we will put these in gitignore file

→ .gitignore

↳ /node_modules

→ Should we put package.json onto git?

- Yes, for package, package-lock

- Both maintain a note of what all dependencies a project needs.

- If I have my package & package-lock, I can regenerate my node modules & even if we delete node modules. We have to just write → npm install

Scanned with CamScanner

npm doesn't stands for node package manager
it manages the package.

DELTA
Date / /
Pg No.

- whatever we can regenerate, don't put it on git, git should have essential things.

→ npm init — package.json were created
↓
It maintains the versions of our dependencies, not the exact version, it can have caret or tilda, we can remove caret/tilda

exactVersion → package-lock.json
~~if we~~
~~all the transitive~~

→ package-lock.json → It contains exact version of all the transitive & all the dependencies that my project will ever need.

→ We are building our app using parcel

- npx parcel index.html
↳ source file of our project

→ It created a local server for us and our App is running there.

= when we want to execute a package we use npx

Scanned with CamScanner

18:12:00

DELTA
Date / /
Pg No.

- npx parcel index.html

→ Here parcel package is executed

→ parcel basically goes to the source index.html and build a development build for our app and it host that development build to localhost

→ Another than CDN another way to get react into our app via npm

→ React is a JS package (some piece of JS code) it is hosted over npm also we will get react from npm now.

→ CDN link is not a prefferred way. Because we don't want to make a network call to get react, we will have it our node modules

- second reason is CDN link will not update with react version, so we have to keep changing the with version, so it's better to have it in package.json.

- Now we will ~~react~~ install react as our package.

Scanned with CamScanner

→ npm install react

We are not writing "-D" here, because we need react as normal dependency not as a dev dependency.

→ npm install react-dom

App.js → import React from "react"

This is in node modules

→ import ReactDOM from "react-dom";

→ This will trough an error "Browser script can't have import or exports".

→ In our HTML → <script src="./App.js"></script>

We injected this

It treats this script as browser script that this is normal JS file and normal JS can't have import.

→ Now we have to tell the browser that this file is not a normal browser file it's a module.

Scanned with CamScanner

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

Parcel – (Advantages)

→ Dev Build (created)
→ creates local server
→ Regreshing page automatically
↓
HMR = Hot Module Replacement

→ Parcel uses a File watching Algorithm (It is written in C++)
↓
It keeps an eye on all the files as soon as we are developing something we change anything into our code parcel is keeping a track of every save we are doing in our project. It will build again

→ Every time after saving the building process reduces, now it is taking so less time because parcel is caching things for us

"Caching – Faster Builds"

Scanned with CamScanner

→ Parcel is giving faster development experience because it is also doing caching.

- When we install parcel, we build our project, parcel quickly sniffed into our project, also took some space

- .parcel-cache
  - → binary files

→ Parcel also does image optimization
- most expensive thing in web browser to load images in your page.

→ we are doing development build but if we create a production build it will ~~minify~~ minify our file, minification of our file

→ Parcel Bundle the files.

→ It compress the file also so that the size is small

→ Parcel make our app fast but there are so many factors like Bundles to fast our app.

Scanned with CamScanner

→ for all these functionalities parcel is using some of the libraries listed in node modules, parcel is manager of all these libraries.

→ more things parcel is doing
- consistent Hashing
- code splitting
- Differential bundling
  ↳ so that our app run on older browser as well (support of older browser)
  ↳ diff bundles for different types of browsers, it will add some extra things to do all such things
- Diagnostic
- Error Handling (better error suggestions)
- HTTP → HTTPS hosting
- tree shaking alg.
  ↳ will remove unused code
- Different dev and production bundles
  ↳ prod build take little more time than building it for dev, some optimization are more in prod build

Scanned with CamScanner

→ npx parcel build index.html
for production build
but it will throw an error

so because we give entry point to index.html but in package.json -

"main": "App.js",

we have to remove this line

→ Now, parcel will make a "dist" folder ~~where~~ for dev build ~~a history of build~~ files.

→ When we write npx parcel index.html it generates a development build project and hosted on local server So when it generates a development build it puts it up in dist folder

When we refresh the page, parcel files it was using parcel-cache and dist to update page using HMR.

→ If we build it for production it takes more time as compare to development

→ It builds 3 files
- index.html
- .css
- .js

Scanned with CamScanner

node_modules → Browserslist → folder

→ these 3 files will contain all code that we write & these 3 files are the production ready code of app

- parcel-cache & dist can be regenerated like node modules.

→ we have to tell our project that what all browsers should our app be supported for.

→ package.json →

```
"browserlist": [
    "last 2 Chrome Version",
    "  "  10    "     "  ",
    "  "  "  Firefox  "  "
]
```

= last 2 version → 79% of all browsers

Scanned with CamScanner