EPISODE-02 | Igniting our App

=> npm does not stands for node package manager

=> npm does not have any full form

=> npm manages packages. It behind the scene manages packages only but does not stand for node package manager.

=> npm has all libraries, all packages hosted on it

=> When we do npm init and do all that stuff we got package.json and it is the configuration file for npm.

=> Our project is depended on lot of other packages also called as dependencies and npm takes care of these packages and version and it will take care of these packages in package.json.

Date:

⇒ Since now we have configured npm for our projects we can now install packages (dependencies) to our project

⇒ The most important package in our project is a bundler.

⇒ When we have these html, css, js files our code needs to bundled, minified, cached, compressed, cleaned before it can sent to production. So the bundler helps in doing these things.

⇒ Bundler basically bundles our app, packages our apps so that it can be pushed to production. ex:- webpack, parcel, vite these all are bundlers that bundles our app.

There are two dependencies which our app can have :-

(i) dev dependencies
(ii) Normal dependencies.

⇒ dev dependencies are required during the development of our app.

⇒ Normal dependencies are used in the production also.

⇒ Now when we write npm install -D parcel to install parcel package we are telling our npm to install parcel having dev dependencies.

⇒ With the help of this command we are basically fetching (installing) parcel from the npm.

⇒ Now this package.json is having :

```
"devDependencies": {
        "parcel": "^2.15.4"
}
```

because it keep tracks of all the dependencies that we are using.

⇒ We have ^ (caret) in ^2.15.4 because if there would be minor change in the version like (2.15.5) then parcel will automaticlly update it.

(~) tilde will install major version of it like 3.0.0.

it is generally safe to use (^) over here b/c for major updates a lot of changes goes over here.

⇒ After we install parcel we got another file package-lock.json and a folder node_modules.

⇒ package-lock.json keeps the track of exact version that is being installed.

and allows updated versions

⇒ package.json has the same data with it like (^2.15.4) but package-lock.json locks the exact version and keep track of it. installed package-lock.json has also integrity in which it has sha512 hash to verify that whatever there is the version of app in local machine it must be the same to the version that is being deployed to the production.

=> node_modules contains all the code that we fetched from npm. npm install -D parcel It takes all the code of parcel and put it over here in node_module.

=> So, package.json is the configuration file for npm and node_module is like the database that has data of all dependencies and packages that our project need.

=> Now since we just install the parcel then why the other packages come in the node_modules. It is because parcel also have some dependencies and their dependencies have other dependencies and this is called the Transitive dependencies.

=> Just like our project is dependent upon the parcel. Parcel itself can be dependent on lot of other things and in turn also dependent on other things and that's why we have lot of packages (folder) inside it like parcel uses babel.

=> Just like our project has package.json every dependencies have their own package.json which maintains their own dependencies.

=> Now, since we have a lot of files folder, packages (dependencies) inside the node module is it not the good practice to put / push on to the production on github → No and that's why we use gitignore and it is used when we don't want some files to go to git on production.

=> but we can put the package.json and package-lock.json on the github because it contains all the dependencies that our project need and package-lock.json maintains the exact version it.
and if we have package.json and package-lock.json we can recreate all our node modules. by hitting npm install if our node modules are deleted. and that's why it is not required to push the node_module to github.

=> Whatever things that can be regenerated don't put it that to git.

=> There is only one package-lock.json and it maintains the exact version of all the dependencies that we have in our project including that of node.module and that's why it is lengthy.

Page
Date

Building the app :

npx parcel index.html

When we hit this command our app is now hosted on to the web server with source file as index.html.

o/p:- server running at http://localhost:1234

Hence parcel is now hosted our app on server and it creates a local host for us and give us port 1234 and creates a dist folder and contain these things (dev builds),

=> Hence to install any package use npm and to execute any package we use:
npx parcel index.html ← source
↑
name of the package

=> Parcel basically takes index.html build a development build for us and host that on local server (local host : 1234)
Note:- Remove "main": "App.js" because will act as starting point our app and will conflict index.html

=> Now Remove the cdn links of React from html because our node module is already having react in it and fetching react library using cdn involves the network call which, is a costly operation.

=> So, install React as a package now:

npm install react.
OR npm i react

=> and also install React DOM :-
npm install react-dom

=> Now we has just installed React and React-DOM and now just use it by Importing from Node-module folder using:-

import React from "react"; packages inside node-module

import ReactDOM from "react-dom";

=> Import these files in app.js now if you try to run this, it will give error Browser scripts ~~does~~ cannot ~~not~~ have ~~support~~ Imports ~~and~~ or Exports.

It happens because normal javascript files cannot support imports or exports so to tell them this is a module write inside script tag of html type = "module"

also in newer version we import ReactDOM as :-
import ReactDOM from "react-dom/client"

Page :
Date :

=> Now if any changes made to any file parcel will automatically updates that. because of caching.

Hence Parcel does :-
- Dev Build
- Local Server
- HMR = Hot Module Replacement (instant refreshing)

Parcel uses :- File Watching Algorithm written in C++ and if any changes made it saves quickly

Parcel also does - caching - which makes faster Builds for us.

So, parcel is caching the things in .parcel-cache and saves them and by next time when you do anything, it quickly builds that next time. and takes less time than initial builds.

- Image optimization
- Minification (of files for production build)
- Bundling
- Compress (like removing white spaces from code)

Hence react is fast because of these (bundlers) bundlers (parcel) are fast and they uses other libraries and hence thee are fast.

- Consistent Hashing
- Code splitting
- Differential Bundling (to support app on older browsers)
- Diagnostic (presenting error in beautiful way)
- Error handling
- Make local server uses HTTPs instead of HTTP
- Tree shaking (remove unused code).
- Different dev and prod bundles

=> When we do any changes and saves it parcel does with the help of .parcel-cache and dist folder using HMR

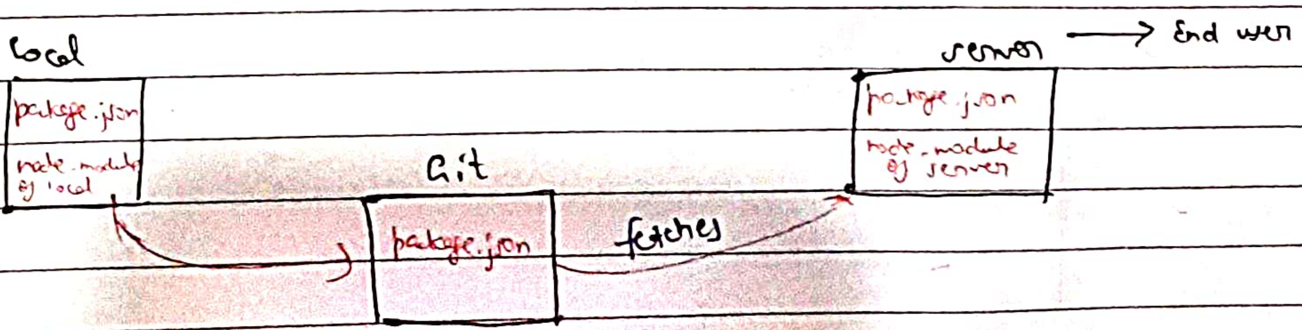
=> To build production app use:
npx parcel build index.html

and it will generate production level files in dist folder and it compress all other files into 3 main files :- index.html
index.css
index.js

These 3 files contains all our code and these are the production ready code in compressed and optimized way.

=> dist and ~~package~~ .parcel-cache are regenerated folders so no need to push to git.

Note:



In vscode All the ^dim^ light colour files and folders does not push to git
All the bright light colour files already pushed to git
All the green files needs to push to git.

Page :
Date :

=> When we upload of our files to local and push it to git the server fetches the code from the git and not directly from local server
The node_module, dist, .parcel-cache (all regenerated things) and are different for local and server

=> How to make our app support to different browsers and their different versions.

Inside package.json type :-

```
"browserslist": [
        "last 2 chrome version",
        "last 2 firefox version"
    ]
}
```

OR

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

of all browsers.

Check website browserslist.