9/02/24

Episode - 02 Igniting the app.

→ We need to minify, remove unwanted comments & optimize our code to make it production ready.

→ when we use 'create React App', we already get a Small ready to production react app. & it have npm into it already.

→ There are allot of different packages which comes with react which makes our app fast, Not just react alone.

★ NPM: It is everything but not Node package manager ✗
↓
officially it does not have any full form.

→ Basically NPM → manages the packages ★★★

NPM basically helps in managing & adding different Packages. (Packages are pre-written tested code for a Specific task) into our project. It is a collection of millions of Packages.

→ Millions of Packages exist and there is atleast one package for your task.

→ npm init: → To use npm into our project.
It is like a box for us into which we can arrange or organize all our packages.

→ It basically creates a package.json file where we get all our Packages ^ ~~Packages and~~ informations like name, version etc. organized at a Single place.

→ with npm init we can add npm into our project.

★ Package.Json is configuration for our npm
↓
the way in which the parts of Something, or a group of things are arranged.

→ In package.json we can see all the installed packages & their versions in dependencies section.

→ When we write npm init, it asks us some questions like package name, version etc. So it is also like we are creating a new package.

→ Packages & dependencies are same.

★ Bundler :- It is a node packages which basically bundles our project and make it production ready.

→ Some popular bundlers are webpack, Parcel, Rollup, ~~Vite~~ Vite etc

→ A bundler takes all different files like CSS, JS, HTML etc and combines them in a single organized unit.

→ npm install -D → dev dependency → It is generally required during development

→ Normal dependencies can be also used during production.

★ npm install -D parcel: As bundler combines different files & minifies it, and we do it during production ~~that~~ that's why we are installing it as an development dependency.

→ (^) caret: This is present in version of the dependency, listed in dependencies section.

for ex: ^1.4.2 (^) caret only allows path version to be updated, if any new version comes automatically.

major minor patch

1.4.2 to 1.4.3.

* (~) tilde : In dependencies Section if we have version like ~1.4.
Then it will automatically install the minor updated version only. from 1.4.2 to 1.5.0

* If we doesnot have any Sign nor tilde (~) nor (^) caret, if any updates of that dependencies comes it will not get installed automatically.

* It is recommended to have caret (^) and not tilde (~) because with minor updates allot of things might break into our app.
So always have caret (^) because minor update will not have changes much more than previous one.

10/02/24

Track /

→ package-lock.json :- It keeps the record log of exact version which is installed.

→ package.json : It keeps an approx version of package.

Suppose in package.json we have ^1.8.0 & new update comes as 1.8.5 then our package.json will remain same & 1.8.5 will be reflected into package-lock.json.

→ package-lock.json keeps the record of which version is installed into our project. (exact) (of any package)

→ You may have faced a situation where you say that "my project is working fine on my local storage, but when I deployed it, it is not working !!!"

→ It can may happen because on our local & on the production we have different versions of packages / dependencies.

→ To solve this, package-lock.json have a hash (code) known as integrity, where it keeps track that every dam exact version is being used in production, as it is in my local storage.

→ Node-module :- It is the database or we can say that they are the files of actual codes of all the dependencies or packages which we are using in our project.

→ Transitive dependency :- As we installed parcel, in node modules we can see a file/folder named as parcel, it have all the code of parcel. But along with it we can see allot of additional folder they are the dependencies of parcel.

Parcel can have its own dependencies, dependencies of parcel can also have their own dependencies and much more.

So all these dependencies files are available into node module and this property of dependencies are called Transitive dependency.

→ Every package which we install have ~~their~~ their own Package.Json and in their package.Json their own dependencies/packages are listed. Through which npm will know about the other dependencies and it will automatically install all of them.

→ We don't want to put all node modules into our github or into our production ? → No { when we do ~~git init~~ create react app then we automatically get .gitignore file in which node module is already added. }

→ So add it into .gitignore file, So that git will ignore that file and not upload into ~~github~~.

→ Should I put Package & Package-lock.Json into git ? Yes

Git: A version control System which keeps a track of all the changes which we have made into our project

GitHub: A web based hosting Service for git repositories

→ why you should upload package.Json & package-lock.Json on git ?? Because with help of package.Json & package-lock Json we can recreate our node modules. → Just write npm inst into your terminal.

→ All things which you can re-generate dont upload it into git

★ NPX: used to execute any package.

npx parcel index.html
- parcel → package name
- index.html → Source file of our project

→ after this command parcel make a development build and host it into our local host

→ cdn links are not preferred to bring react into our project.
why ?? → ① Because through cdn we are making another network connection with those cdn links. Instead we can have it in our node module. It will be more easier for us.

② Due to version changes in react, we need to update the cdn links. That's why use npm install react, npm install react-dom

npm i is a short form of npm install

→ But still after installing react & react-dom, our code will not be able to understand what is React & what is ReactDom (browser)

```
const heading = React.createElement("h1", {}, "hiee");
const root = ReactDOM.createRoot(docum... );
```

our browser will not understand these keywords

import React from 'react';
import ReactDOM from 'react-dom';

present in node modules.

→ Also write type = "module" into Script tag into HTML file because, import or export functionality is not available into Simple Java Script.

→ When browsers treats our JS file as ~~Script~~ module then we get features like import or export into our JS file.

☆ Go & read the documentation of Parcel ☆

→ When we write npx parcel index.html : It creates a development build and stores it into dist folder.

→ When we do npx parcel build index.html : It Simplifies / compress & Production ready our app and stores 3 main folders into dist Html, css & Js files and some map files.

→ when we write npx parcel build index.html we get an error

This is because in package.json we have entry point i.e, main set as app.js but we are giving entry point as index.html, so there is a conflict.

To remove this erase map: "app.js" from package.json

☆ Note: We can re-generate ~~regenerate~~, dist & .~~cache~~ .parcel-cache, so no need to upload it to git. So add into .gitignore

→ ★ Browserslist: We can add browserslist into our package.json & add the browser's list, Then our app will make sure that it will run 100% on those browsers. For other rest browsers, it may work or may not.

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

"last 2 versions" → It will support to all the last 2 versions of all the browsers present over internet.

An array of list of browsers.

"last 2 chrome versions" → It will support last 2 versions of chrome

→ If you want to run your app in all browsers available on internet then your app may get slower, because due to this there will allot of differential bundles for different browsers.

→ So it will basically make our app more heavy & it will become slow.

★★★★ Note: use browserslist.dev → To check how much % of browsers our app will 100% support.

Who downloads the node modules at server because we don't upload it?

There are several ways `node_modules` can be downloaded on a server, and who does it depends on your specific setup. Here are the most common scenarios:

1. **Automated Deployments:**

- **CI/CD Pipeline:** When you push code changes, a CI/CD pipeline (Continuous Integration/Continuous Delivery) automatically builds and deploys your application. This pipeline often includes a step to install dependencies on the server using commands like `npm install` or `yarn install`. These commands fetch dependencies from the public registry and store them in `node_modules` on the server.
- **Deployment Tools:** Specific deployment tools like Capistrano or Deploy.js might have built-in functionality to handle dependency installation during the deployment process.

2. **Containerized Environments:**

- **Docker Image:** If you use containerization with tools like Docker, your containers can include specific tools and configurations to install dependencies at startup. This could involve embedding a package manager and its configuration within the container image itself.

3. Manual Installation:

- **Rarely:** In less common scenarios, you might manually install dependencies on the server using the same `npm install` or `yarn install` commands you use locally. This approach is becoming less frequent due to the rise of automation.

Who Does It?

In most cases, **nobody needs to explicitly download or install dependencies manually** since automated processes handle it. The specific actor depends on your setup:

MODULES

Modules are designed to be reusable, which means that they can be imported and used in multiple places in an application. In contrast, plain old JavaScript code is not organized into reusable units, which makes it more difficult to reuse and maintain.

In modules we have the feature of import and export, which was not available into old js.