React (Day-2)

Note: Use "git branch -M main" before pushing local repo to remote.
Else it'll create a new master branch on remote.

→ We are going to learn how to build production ready react apps.

→ Using create-react-app, it already gives us the production build. But we will never know how it does that.

→ It takes different package of javascript (not only just react) to make our react app production ready.

→ Production ready ⇒ minified, efficient, optimized, etc, cached, cleaned, compressed

Funfact: NPM doesn't stand for node package manager. node package manager is functionality of NPM, but NPM doesn't stand (full form) for that. Go check official website XD.

→ Set up your project, using 'npm init'
for test command, write "jest"

→ package.json is created, which is a configuration for npm.

→ So to build our app into production ready, we need bundler.

There are various bundlers

$$\begin{tikzcd}
 & \text{create-react-app} \arrow[r, "\text{uses}"] & \text{webpack} \\
\text{There are various bundlers} \arrow[ru] \arrow[r] \arrow[rd] & \text{parcel} & \text{babel} \\
 & \text{vite} &
\end{tikzcd}$$

→ We will be using parcel.

There are two types of packages/ dependency

normal dependency
↓
Required in production cycle too
(npm install pkgName)

dev dependency
↓
only required in the development cycle of app
(npm install -D pkgName)

→ Now go ahead and install parcel as a dev dependency.
"npm install -D parcel"

→ parcel is added in devDependencies inside package.json.

```
"devDependencies": {
    "parcel": "^2.8.3"
}
```

[What does ^ and ~ before version signify?]

caret (^) → uses all the future minor/patch versions without incrementing the major version.

tilde (~) → will update to all future patch versions without incrementing the minor version.

```
2.3.4 ──→ patch
↓   ↓
major minor
```

$^2.8.3 = 2.8.3$ to $< 3.0.0$ (↳ less than)

$\sim2.8.3 = 2.8.3$ to $< 2.9.0$

→ package-lock.json stores the exact version of the dependencies that are used in the project. Ex: "2.8.4"
But package.json uses ~~versioning~~ versioning ready for future updates. Ex: "^2.8.4" or "~2.8.4" (approx version)

→ There is also key "integrity" for dependencies in package-lock.json which has a hash value. It ensures that the code working on local machine also works on production.

node_modules: It contains all the dependencies that
(heaviest part of our project) we install in our project and also the dependencies of these dependencies.
(transitive dependency)

↳ Hence we put node-modules in .gitignore (so that git ignore it while pushing it to remote)

* package & package-lock can be used to regenerate the node_modules.

→ Run 'npx parcel index.html' to build development and our project now runs on the local server.

npx → used to execute packages
npm → used to install / remove packages.

→ Now let's remove the CDN links of react & react-dom and install them using npm.

'npm install react react-dom'

→ Now for react to work in our project, import react in the App.js.

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

→ Now it says Browser scripts can't have imports/exports. since we included App.js script in index.html, it is treated as a normal browser script.

Hence we will specify the type of script as "module"

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

→ Now we'll get a warning, that we should import ReactDOM from 'react-dom/client' and not from 'react-dom'

so change it => import ReactDOM from 'react-dom/client';

* Previously ReactDOM was imported from 'react-dom' but currently, it is available in 'react-dom/client'

→ All of this is okay, but why use parcel? Because it :-

* Dev build
* Local server, Tree shaking
* HMR = Hot Module Replacement (like live server)
* File Watching Algorithm + written in C++ (very fast)
* Caching - Faster Builds .parcel-cache folder
* Image optimization, code splitting, Differential bundling
  - diff bundle for diff apps
  - supports older browsers
* Minification, Bundling, Compressing, consistent Hashing
* Diagnostic, Better error handling, HTTPS hosting

→ It's not just React that makes our App fast, but behind the scenes it's the bundler like parcel too that makes our application super fast.

→ Tree shaking - removes unused code.
[Read more about parcel on parceljs.org]

→ Different dev and prod bundles.

→ To build 'prod build' of our project,

'npx parcel build index.html'
↳ builds 'prod build' in dist folder

and for dev build,

'npx parcel index.html'

→ node-modules, .parcel-cache, dist folders can be regenerated and hence can be put inside .gitignore file.

$$\begin{tikzcd}
 & & \text{Server can run commands and generates node-modules, etc} \\
 & & \boxed{\text{Server}} \arrow[u] \arrow[r] & \text{Served to end users} \\
\boxed{\text{local}} \arrow[r, bend right, "\text{push}"'] & \boxed{\text{GitHub}} \arrow[ur, bend right, "\text{fetches}"'] & &
\end{tikzcd}$$

browserslist

we can configure browserslists (which browsers and which versions of browsers should support our application using the browserslist and parcel.

In package.json,

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

} this by browserslists documentation, confirms that it covers 78% of browsers.

Depending on what we mention in browserslist, parcels creates different builds.

so it's not really a great idea to think of covering 100% browsers.

That will make our application very heavy by creating a lot of different builds to support different browsers and diff versions.

Govt websites generally try to achieve nearly 100% browsers coverage,

But we as developers, can concentrate on our niche audience by given diff query parameters in the browserslist.

Ex: ~~"last 10 chrome~~ "last 10 chrome version",

"cover 99.5% in US",

"last 2 major versions", etc.