Q)Why is react a library and not a framework?

Ans) For frameworks we need to generate the entire project into that framework but for react which is a library it doesn't need to be

generated for the entire project as react can exist in only a small portion of our app(by injecting react cdn) eg. we can have react only

in header, we can have react only in footer.

-->React.createElement() returns a javascript object and when this object is passed inside root.render() it gets converted to html and

gets render inside root div.

-->If some html is already present inside root div used for react then when root.render() executes it will replace the content

inside root with content passed inside root.render().

-->Cross-origin

🡪**Bundler**: A **bundler package** is a tool that takes various assets such as JavaScript, CSS, HTML, and images, and combines (or "bundles") them into optimized files (minified, compressed, chunking), usually for web deployment, to improve loading times and performance. Examples include Webpack, Parcel, and Rollup, **Vite**.

**Bundler** is typically added as a **dev** dependency because it is only needed during the development process to package and optimize your code. Once the bundler completes its job (e.g., compiling, minifying, and bundling files), the resulting bundled files are what get deployed to production.

In production, the actual bundler tool itself is not required—only the output files (like bundle.js) are used. Therefore, bundlers do not need to be included in the production environment, which is why they are categorized as dev dependencies.

npm install

Normal/prod dependency

Dev Dependency

Normal dependency

npm install package-name

Only needed during development and not during production.

npm install -D parcel

🡪caret(^) vs tilde(~) inside version in package.json: **Caret (^)**: Automatically updates the dependency to the latest minor version, allowing updates that do not change the first non-zero version number. For example, ^2.8.4 will use release from 2.8.4 to < 3.0.0 .

**Tilde (~)**: Automatically updates the dependency to the latest patch version within the same minor version. For example, ~2.8.4 will use release from 2.8.4 to < 2.9.0 .

🡪**package-lock.json**: it records the specific versions of all installed packages, ensuring that the same versions are used every time the project is installed, preventing unexpected changes due to updates and speed up installation, and maintain security across environments.

🡪**package.json** is a configuration file for npm which keeps track of what versions of packages are installed.

🡪**TODO**: notes from “integrity” in package-lock.json regarding “sha…” in episode-02 at 43min

🡪**Transitive dependencies**: these are the dependencies of your direct dependencies; in other words, they are packages required by the libraries you are using, forming an indirect dependency chain. Eg npm install -D parcel installed node\_modules which in addition to parcel contains dependencies that parcel needs and dependencies of dependencies that parcel’s dependencies need. **Every dependency/libraries will have it’s own package.json.**

🡪**node\_modules**: collection of dependencies.

Q) Why don’t we push node\_modules to GitHub?

Ans) Because this folder is huge and we don’t need to push because we can easily generate node\_modules from package.json & package-lock.json.

**NPX**: used to execute a package

e.g. npx parcel index.html 🡪executes parcel package and parcel goes into the source(index.html) and builds a development build of our app and host it on localhost:1234

**Parcel**

* Dev build
* Local server
* HMR: Hot Module Replacement
* File watching algorithm: written in C++
* Caching: faster builds
* Image optimization
* Minification
* Bundling
* Compress file
* Removes white spaces from code
* Bundlers make our apps fast/optimized.
* Consistent hashing
* Code splitting
* **Differential bundling**: when used **<script type=”module”>** it creates different bundles to support older browsers by generating nomodule fallback for old browsers.
* Diagnostic
* Error handling/suggestions
* Provides **https** for production, can also generate https certificates during development
* **Tree shaking**: remove unused code/functions
* **Different dev and prod build**

**🡪Generating production build**: npx parcel build index.html

This will compress and minify all the files into only 3 files index.html, .js, .css

**🡪We don’t push file/folders to GitHub which we can generate easily eg. Node\_modules, dist, .parcel-cache etc.**

**🡪Browserslist:** configuration to support which versions of browsers does our app support. By passing the browser configuration it will make sure that our app will definitely work on these browser versions.

Inside package.json 🡪 browserslist: [“last 2 versions”]

Last 2 versions: last two versions of all browsers

Last 2 Chrome versions: last two versions of chrome

**🡪React is different and JSX is different(JSX is not part of React), react can be written without JSX.**

**🡪JSX is not HTML in JavaScript, it is HTML / XML like syntax.** JSX creates ReactElement (It gets transpiled by Babel (or another JSX compiler) into a React.createElement)

**🡪JS engine & browser does not understand JSX they only understand ECMA\_Script ( pure JavaScript ). JSX is transpiled before even reaching to JS engine by Parcel using package “Babel” which transpiles/ converts JSX into plain JavaScript that the JS engine / browser can understand.**

const element = <h1>Hello, world!</h1>;

JSX gets transpiled to JavaScript

const element = React.createElement('h1', {}, 'Hello, world!');

🡪In order to give attributes into JSX we have to use camelCase naming conventions

🡪For **multiple-line JSX** we need to enclose it inside **parenthesis**() so that babel can understand that this can be starting/ending of JSX.

🡪Babel is a JS compiler/transpiler. In addition to JSX transpilation, babel also transpiles Modern JavaScript into JavaScript that older browsers can understand.

**🡪 @parcel/transformer-js: Browser scripts cannot have imports or exports.**

<script src="./App.js" ></script> This script tag is a browser tag and inside App.js we are importing React & ReactDOM but import and export are not allowed inside browser scripts so we need to tell this script tag that this script is not a normal script and use type=” module” so that it can use import/export.

**Functional Component**: A function that returns JSX.

🡪curly brackets “{ }” are used to use JavaScript inside JSX. JSX sanitizes every thing we passed inside curly brackets to prevent Cross-site Scripting attack.

🡪We can use Functional components like <Component /> or <Component></Component> or { Component() }

**🡪JSX makes our react code readable.**

Q) Why passing passwordGenerator as a dependency in useEffect(from 03passwordGenerator react project)?

Ans) By including passwordGenerator as a dependency, you're ensuring that the useEffect always uses the latest version of the function, reflecting any changes in its dependencies. This is especially important if passwordGenerator relies on props or state values that may change over time.

WATCH SOME MORE VIDEOS FOR MORE CLARIFICATION.