**CHAPTER NO:2 – ‘IGNITING OUR’ APP Theory Notes**

* What is NPM?

Ans:-

NPM is the world’s largest software registry founded in 2014. Its not mentioned in NPM website that the full form is ‘Node Package Manager’.

* NPM is package manager for Node.js. It was created in 2009 as an open source project to help JavaScript developers easily share packaged modules of code.
* The NPM registry is a public collection of packages of open -source code for Node.js [front-end web apps](http://www.ember-cli.com/), [mobile apps](http://cordova.apache.org/), [robots](https://tessel.io/), [routers](https://linerate.f5.com/), and countless other needs of the JavaScript community.
* npm is the command line client that allows developers to install and publish those packages.
* What is Parcel/Webpack? Why do we need it?

Ans:-

Parcel and Webpack are BUNDLERS which makes bundles of modules and makes provides features like Hot Module Replacement(HMR), Clean code, code splitting, dev & production build generation, minifying, image optimization, caching while development, compression, renaming variable, old browser compatibility, feasibility to run development build with HTTPS, zero config bundler etc.

* What is parcel-cache?

Ans:-

Parcel-cache is a folder created by parcel bundler while building the application as it requires its own space to function on its own features. It stores information about your project when build it. When parcel re-builds the same, it doesn’t have to reparse or re-analyze everything from scratch thus build time will get reduce significantly.

* What is npx?

Ans:-

Npx is represents npx run which we use to ignite out app. It stands for Node Package Execute and is used to execute any package in npm registry without even installing it.

* What is difference between dependencies vs dev-Dependencies?

Ans:-

Dev dependencies are modules which are only required during development whereas dependencies are required at runtime/production. If you are deploying your application, dependencies has to be installed, or else your app simply will not work. Libraries that you call from your code that enables the program to run can be considered as dependencies.

Eg- React , React - dom

Dev dependency modules need not be installed in the production server since you are not going to develop in that machine. Compilers that covert your code to JavaScript , test frameworks and document generators can be considered as dev-dependencies since they are only required during development .

Eg- ESLint , Babel , webpack,parcel,vite

* What is Tree Shaking?

Ans:-

Tree-shaking is basically removing the unwanted code/dead code while building the app. This job is done by bundler who bundles the app. This process is important for production ready application to have minified ,clean version.

* What is Hot Module Replacement?

Ans:-

HMR is one advantage feature given by Parcel. It is use in rendering only particular section of page rather than whole page. Thus, it exchanges, adds, or removes modules in the browser at runtime without a full page reload. This means that application state can be retained as you change small things in your code.This means that application state can be retained as you change small things. Parcel's HMR implementation supports both JavaScript and CSS assets.

* List down your favorite 5 superpowers of Parcel and describe any 3 of them in your own words.

Ans:-

HMR, Image Optimization, Caching while development, ability of dev build with HTTPS, Old browser compatibility

* HMR :- Hot Module Replacement (HMR) improves the development experience by automatically updating modules in the browser at runtime without needing a whole page refresh. This means that application state can be retained as you change small things. Parcel's HMR implementation supports both JavaScript and CSS assets.

As of version 1.12.0, the default implementation has changed to fully refresh the page when files change. You can opt-in to enable true HMR by adding the following in your app. This will only apply in development; HMR is automatically disabled when bundling in production mode.

if (module.hot) {

module.hot.accept()

}

As you save files, Parcel rebuilds what changed and sends an update to any running clients containing the new code. The new code then replaces the old version, and is re-evaluated along with all parents. You can hook into this process using the module.hot API, which can notify your code when a module is about to be disposed, or when a new version comes in. Projects like [react-hot-loader](https://github.com/gaearon/react-hot-loader) can help with this process, and work out of the box with Parcel.

There are two methods to know about: module.hot.accept and module.hot.dispose. You call module.hot.accept with a callback function which is executed when that module or any of its dependencies are updated. module.hot.dispose accepts a callback which is called when that module is about to be replaced.

if (module.hot) {

module.hot.dispose(function() {

// module is about to be replaced

})

module.hot.accept(function() {

// module or one of its dependencies was just updated

})}

* Image Optimization:- Parcel also includes lossless image optimization for JPEGs and PNGs by default in production mode, which reduces the size of images without affecting their quality. This does not require any query parameters or configuration to use. However, since the optimization is lossless, the size reduction possible may be less than if you use the quality query param, or use a modern format such as WebP or AVIF.
* Caching while building code:- Parcel caches everything it builds to disk. If you restart the dev server, Parcel will only rebuild files that have changed since the last time it ran. Parcel automatically tracks all of the files, configuration, plugins, and dev dependencies that are involved in your build, and granularly invalidates the cache when something changes. For example, if you change a configuration file, all of the source files that rely on that configuration will be rebuilt. By default, the cache is stored in the .parcel-cache folder inside your project. You should add this folder to your .gitignore (or equivalent) so that it is not committed in your repo. You can also override the location of the cache using the --cache-dir CLI option. Caching can also be disabled using the --no-cache flag. Note that this only disables reading from the cache – a .parcel-cache folder will still be created.
* Build with HTTPS mode:- Sometimes, you may need to use HTTPS during development. To use an automatically generated self-signed certificate, use the --https CLI flag. The first time you load the page, you may need to manually trust this certificate in your browser.

Parcel index.html –https

* Minification: Parcel performs minification for HTML, CSS, Js and SVG. It performs minification in various ways like by removing whitespace, renaming variables to shorter names(Uglification), and many other optimizations. By default, it is enabled when building the application but we can suppress it using flag --no-optimize flag. Parcel uses terser to minify JavaScript, lightningcss for CSS, htmlnano for HTML, and svgo for SVG. If needed, you can configure these tools using a .terserrc, .htmlnanorc, or svgo.config.json config file.
* zero config bundler
* What is .gitignore? What should we add and not add into it?

Ans:-

The purpose of gitignore files is to ensure that certain files not tracked by Git remain untracked. We should add all the auto generated file, folders to git ignore as it can be generated on the fly and hence no tracking is required.

We can add .parcel-cache to git but we **should not** add package.json,package-lock.json files and node\_modules folder to .gitignore.

* What is the difference between package.json and package-lock.json?

Ans:- Package.json is the versioning file which keeps track of all the package installed in the project. package-lock.json is the file which it locks the exact version of the packages getting used in the project.

In short, the package.json defines the rules for pulling in your dependencies, and the package.json.lock describes the exact decencies that were downloaded based on those rules.

* Why should I not modify package-lock.json?

Ans:- You should never modify package-lock.json file as it locks the exact compatible version getting used in the project. Modifying this file might create unecessary conflicts between different transitive dependencies in the project. Whenever we need to modify the version for any package, we should do it in package.json or by running npm install specific version command.

* What is node\_modules ? Is it a good idea to push that on git?

Ans:-Node\_modules is the package folder which get install when we do npm init. It consists of all require packages present in npm. We should never put node\_modules in git folder as it consists of large size.

node\_modules is the cache for the external modules that your project depend upon. It includes all the packages required for your project, and it generates on the fly when you build the project and hence, should not be pushed to git

* What is the dist folder?

Ans:- dist is the production-ready compiled and distributable version of your code. It gets generated when you build your app. This compiled code alone is sufficient to serve the application. It generated automatically when we build our app.

* What is browserslist?

Ans:- browserslist is the library/array which helps us to define the list of past versions of browsers that should be supported by our app. This is featured by Babel.

* Difference between ^ - caret and ~ - tilda?

**Ans:-** When we open our package.json file and search for dependency property and in there we find the packages that are listed as a nested object of the dependency property **package-name:package-version.**Now look at the package version, we find some numbers separated by three dots (e.g. **2.6.2**).

NPM versions are written using three dots separated numbers the first number from the left shows the major release and the second number from the left shows the minor release and the third number shows the patch release of the package.

**Syntax:**The syntax of the npm version looks like the following.

Major.Minor.Patch

**Tilde (~) notation:**It is used to match the most recent patch version. Tilde ~ notation freezes the major version and minor version. As we know patch updates are bug fixes that’s why we can say ~ notation allows us to automatically accept bug fixes.

**Example:**The ~1.2.0 will update all the future patch updates. We have to write just ~1.2.0 and all the next patch update dependencies. For example, 1.2.**1,**1.2.**2,**1.2.**5**……………1.2.x.

**Note:**Patch updates are very small security changes in a package that is why the **~version**is approximately equivalent to the version.

**Caret (^) notation:**It is used for automatically updating the minor updates along with patch updates.

**Example:**The^1.2.4 will update all the future **Minor**and **patch**updates, for example, ^1.2.4 will automatically change the dependency to **1.x.x** if any update occurs.

Using caret notation it is important to look at our code regularly if it is compatible with the newest version or not.

* What is different types in script?

Ans:-

Type=”module”

This value causes the code to be treated as a JavaScript module. The processing of the script contents is deferred. The charset and defer attributes have no effect. For information on using module, see our [JavaScript modules](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules) guide. Unlike classic scripts, module scripts require the use of the CORS protocol for cross-origin fetching.

Type=”importmap”

This value indicates that the body of the element contains an import map. The import map is a JSON object that developers can use to control how the browser resolves module specifiers when importing [JavaScript modules](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules).

Any other value

The embedded content is treated as a data block, and won't be processed by the browser. Developers must use a valid MIME type that is not a JavaScript MIME type to denote data blocks. All of the other attributes will be ignored, including the src attribute.

Type= ”[blocking](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script)”

This attribute explicitly indicates that certain operations should be blocked on the fetching of the script. The operations that are to be blocked must be a space-separated list of blocking attributes listed below.

render: The rendering of content on the screen is blocked.