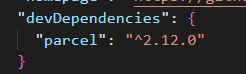
1. There’s not just React, rather there are a lot of packages that are making our app faster.
2. npm -> It is a package manager and it manages all the packages that we install in our system but does not stand for node package manager. Sometime these packages are also known as dependencies.
   * Our project is dependent on lot of packages and the packages that our project is dependent on are known as **dependencies** and npm manages that.
   * Suppose we’re using any dependency or package, so what version of packages we’re using?
     + npm will take care of it. And it will take care inside package.json.
3. Any package you need to include in your project, you can use npm. All the packages are hosted over there.
4. When you create a “createReactApp” -> it has already npm inside it. So, you don’t have to configure it.
5. When we do **npm init** => it will ask us several questions in terminal and after replying those questions it will create package.json.
6. Package.json => it is a configuration for npm.
   * We are using so many packages and npm will take care of what version of packages we are using and it will take care in package.json.
   * It keeps a track of version of all the dependencies, all the packages that is installed into our system.
7. The most important dependency in our project is **Bundler.**
   * What is Bundler?
     + When we have normal html, css, js file; our whole code needs to be bundled together, our whole code needs to be minified, our whole code needs to be cached, our whole code needs to be compressed, our whole code needs to be cleaned before it can be sent to production. **So, a bundler take care of all these things.**
     + WebPack, Parcel, Vite => All these are bundlers
     + Job of Bundler:
       - Bundler bundles your app. It packages your app properly so that it can be shipped to production

* When we create a **createReactApp** => It also uses Webpack bundler behind the scenes.

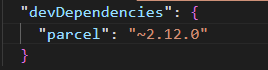
1. Parcel comes as a node package. And since we’ve npm in our project, we can install Parcel using npm.

**npm install -D parcel**

* We’re using -D => means installing parcel as a dev dependency.
* Now, this parcel is a bundler. This chunking, minification is done where?
  + It’s not done in production. We’ll do this in development phase.
* There are 2 different dependencies we can install:
  + Dev Dependency : It means it is generally required in a development phase.
  + Normal Dependency : It is used in production also.

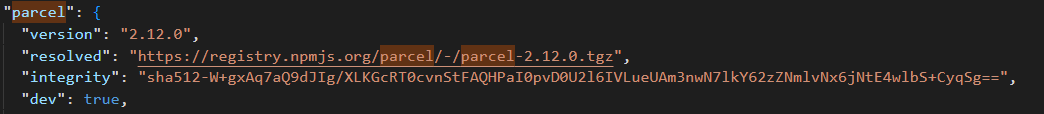
1. **PARCEL is a beast.**
2. **** ^ => this symbol is known as **caret**

Suppose tomorrow a new version of parcel came up (new minor version), say *2.12.1*. So using *caret(^),* parcel will automatically get upgraded to *2.12.1* version.

**** ~ => this symbol is known as **tilde**

And suppose tomorrow a major version of parcel came up, say *3.12.1*. So using *tilde(~),* parcel will automatically get upgraded to *3.12.1* version.

* *It’s always safe to put caret(^). Because if a major upgrade comes up, then a lot of things will break into your app.*

1. When we added parcel, 3 different things happened:
   * devDependencies added over package.json
   * A new file **package-lock.json** has been created
   * **node\_modules** has also been added
2. package-lock.json:
   * It locks the exact version of package/dependency that we installed in our project and keeps a record of it.
     + ***package.json*** *keeps a approx version of packages while* ***package-lock.json*** *keeps the exact version of the packages.*
   * 

What is this integrity? Why is this integrity?

* This integrity is basically a hash, a sha512 hash.
* Why is this hash present over here?
  + Have you heard this thing? It is working in my local but I don’t know how it break on production?
    - So basically to avoid that, package-lock.json keeps a hash to verify that whatever is there in my machine is the same version that is deployed into the production.

1. node\_modules:
   * This node\_modules contains all the code that we fetched from npm.
   * When we did **npm install parcel** => it was fetching all the code for parcel and putting it inside node\_modules.
   * *package.json is a configuration for our npm and node\_modules is like database, it contains actual data of dependencies/packages that our project need.*
   * node\_modules is a collection of dependencies.
   * Why there are so many folders inside node\_modules even though we’ve just installed parcel?
     + Our project is dependent on parcel and now parcel can itself be dependent on a lot of things. That’s why when we install parcel, parcel says install other packages also. And this inter-dependency is knows an *Transitive dependency.*
     + Parcel needs help of **babel** also.
     + Parcel needs all the packages that are inside node\_modules. None of these packages are garbage packages.

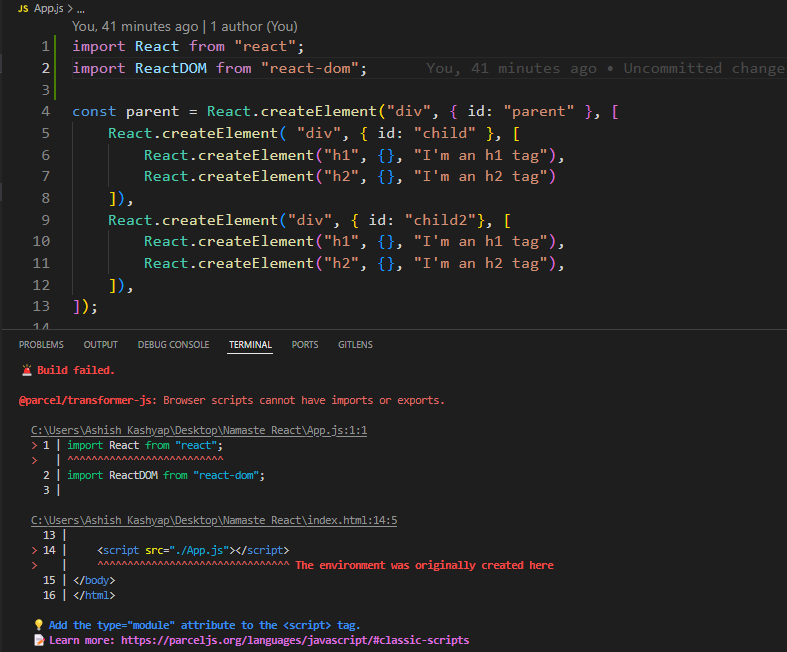
* Our project has package.json, parcel has package.json and all the libraries inside that will have their own package.json.
* The size of node\_modules is very large as compared to number of lines we’ve written in our project. Should I push this node\_modules in production / github?
* The answer is – **NO.** So, we’ll put this file in .gitignore.
* .gitignore => If you want any of your files not to go in production / git, just put those files in .gitignore.
* Should we push package.json & package-lock.json on github?
* The answer is – absolutely **YES.**
* package.json & package-lock.json maintains a note of all the dependencies that our project need.
* If you’ve package.json & package-lock.json, you can re-create all your node\_modules.
  + Just do **npm install =>** it’ll re-create node\_modules.

1. Now, it’s time to ignite our app:
   * We’ll do **npx parcel index.html** => *npx parcel source-file*
     + It has created 2 more folders:
       - .parcel-cache
       - Dist
     + What happens actually?
       - Parcel goes to source index.html and builds a development build for our app and host that development build to our localhost:1234.

* **npm** – this command is used to install a package (npm install *package-name*);
* **npx** – this command is used for executing the package (npx *package-name source-file-name*).

1. CDN links are not the preferred way to bring react and ReactDOM into our project. Why?

* Because when we try to fetch from CDN, it will make a network call to unpkg.com and it will get react from unpkg.com. Suppose if you already have react in your node\_modules, how easy would it be to use react in our code.
  + First thing is, we don’t want to make another network call to react, we’ll already have it in our node\_modules.
  + Second thing is, what if newer version of react came in? => In this case you have to always keep changing URL of CDN.

1. So, we’ll install react through npm. => **npm install react**
2. Now, let’s install react-dom => **npm install react-dom**
3. import React from “react”
   * “react” : it refers to react inside node\_modules
   * And above statement means React is coming from “react”
4. 

Right now we’re getting above error – Browser scripts cannot have imports or exports.

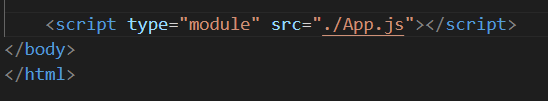
This is because we’ve used <script src="./App.js"></script>

in index.html which is treating App.js as a normal js file but it is not since we’ve injected react and react-dom into it.

To resolve this, we need to tell the browser that this file is not a normal js file rather it is a module.

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

So, just make these changes in index.html.



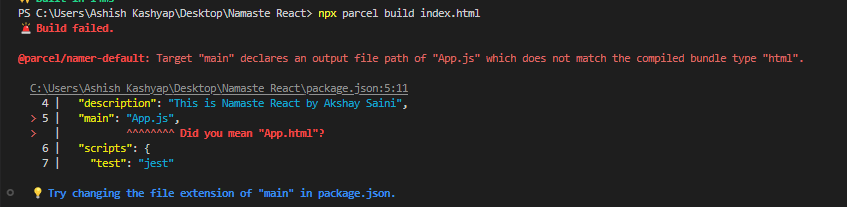
1. When we make any changes to our file (App.js) and save it, then it gets automatically updated to browser even without saving it. And this is done by **Parcel**.

And this is called **HMR (Hot Module Replacement).**

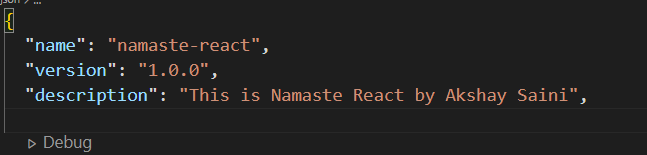
1. The reason for HMR is it uses a File Watching Algorithm that is written in C++.
2. It also gives you faster build whenever you keep saving things means it develops next build faster because it is caching all the things you do.
3. **PARCEL:**

* Dev Build
* Creates local server
* It does **HMR** => Hot Module Replacement.
  + This is the reason for when we make any changes and save it then it gets automatically updated to browser even without refreshing our web page.
  + This is because it has **File Watching Algorithm** that is keeping a watch on all the files. And this algorithm is written in C++.
* It gives you faster build because of **Caching**.
* It also does **Image Optimization**, which is the most important thing in your web browser – to load image in web page.
* It’ll also do minification (when we do production build)
* Bundling
* Compressing
* Consistent Hashing
* Code Splitting (It’ll split your files)
* Differential Bundling
  + It means when we open our webpage, it can be open on any browser – be it any older version of any browser or any mobile browser.
* Diagnostic
  + Better Error Suggestion – that we view in Console tab after Inspecting it or even the error that we see in terminal
* HTTPs
  + It also gives you a way to host it on HTTPs server also.
* Tree Shaking
  + Suppose you have 20 function in your file and you’re using just 5 of them. So, it’ll remove unused code from file. This is called **Tree Shaking.**
* It’ll create different dev and prod build.

1. **To create prod build: npx parcel build index.html**
2. When we do prod build, we’ll encounter an error.



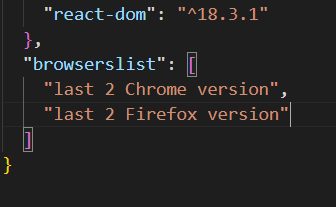
To resolve this, we need to remove => "main": "App.js" from package.json because it is conflicting the starting point as "main": "App.js" is **indicating starting point of our file.**



Then again do **npx parcel build index.html**

1. When we do npx parcel index.html => it creates a dev build and put that inside dist folder.
   * So the code that we were viewing in pour webpage was coming from dist, not from App.js and when we were making the changes to the page and saving it, it was using dist & parcel-cache folder to update it and doing HMR.
   * **We can create dist folder again and again after creating dev build or prod build.**
   * **We can also create .parcel-cache folder again and again after creating dev build or prod build.**
2. Now, let’s make it compatible for older browsers. So, this can be done using **browserslist.**

For this we need to make changes in package.json:

You can check these things in : <https://browserslist.dev/?q=bGFzdCAyIHZlcnNpb25z>