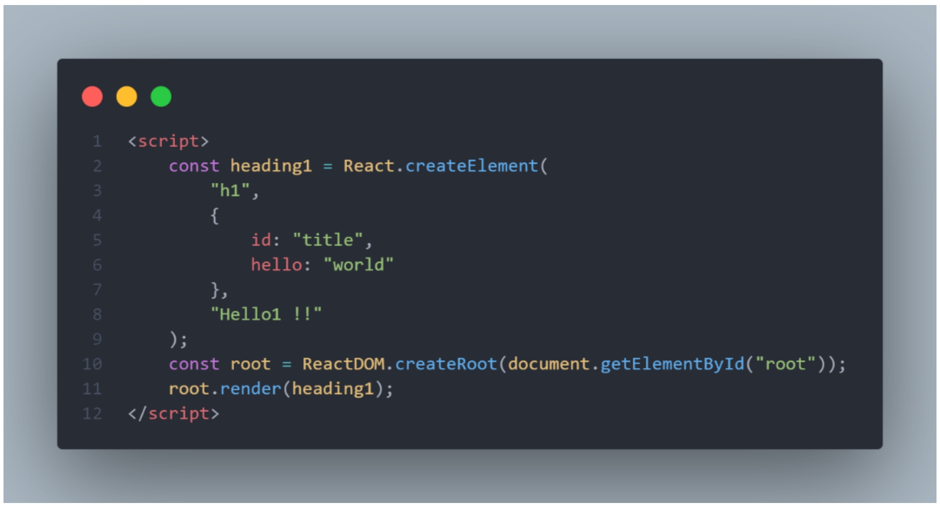
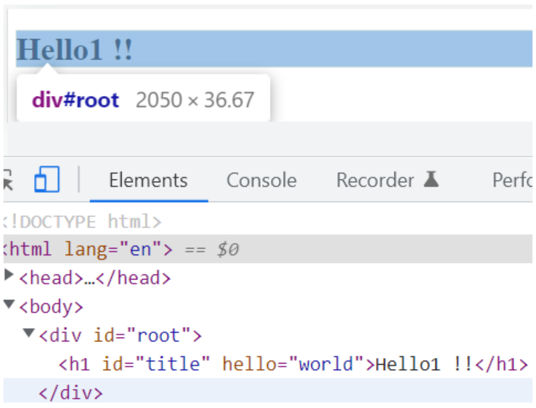
Important Points :-

- If you copy paste the links from the CDN links in the browser and press enter, then you will get a big piece of code written by the facebook developers and it is that code which gives us the functionalities of react and reactDOM library respectively.
- In the React.createElement(), in the props sections, if we write something like
  "message": "hello world"
  That will also get accepted as an HTML attribute without any error like :



```
<script>
    const heading1 = React.createElement(
        "h1",
        {
            id: "title",
            hello: "world"
        },
        "Hello1 !!"
    );
    const root = ReactDOM.createRoot(document.getElementById("root"));
    root.render(heading1);
</script>
```



Hello1 !!

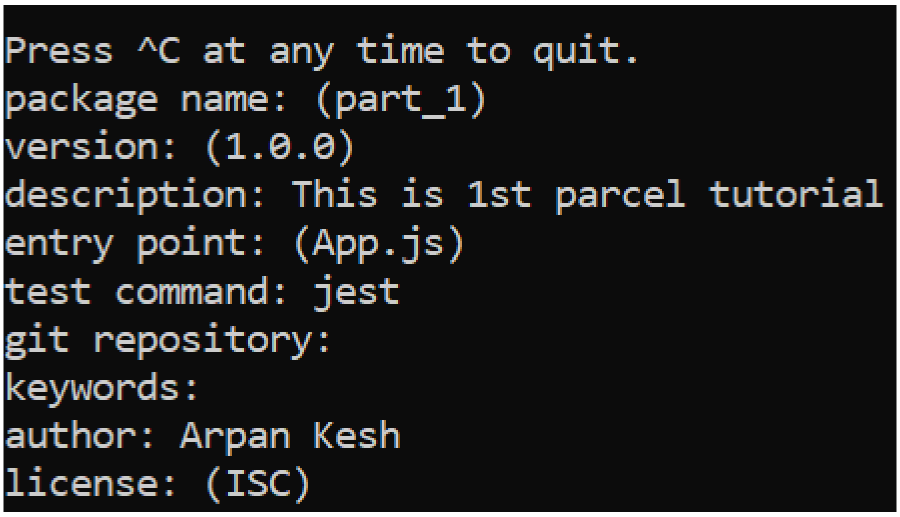
```
<!DOCTYPE html>
<html lang="en"> == $0
▶<head>…</head>
▼<body>
  ▼<div id="root">
      <h1 id="title" hello="world">Hello1 !!</h1>
    </div>
```

- "npm init -y" ignores certain steps that are required in "npm init" by taking default values.

(learn about bundlers and parcel from the notes in copy)

Setting up Parcel :-

- Go to the root directory of your project and run **"npm init"** and fill the required details like :-



```
Press ^C at any time to quit.
package name: (part_1)
version: (1.0.0)
description: This is 1st parcel tutorial
entry point: (App.js)
test command: jest
git repository:
keywords:
author: Arpan Kesh
license: (ISC)
```

No need to know about the "jest" in the "test command".

- This will create a package.json in your folder. We need npm because our React app needs functionalities from other packages other than create-react-app and npm helps to manage these packages. These packages offer functionalities like :-
  - Minimising our code for production
  - Bundling our files
  - Optimising our code , etc.
- Now, to install parcel, we can run either of the 2 below commands to install parcel as a devDependency :-
  `npm install --save-dev parcel` OR `npm install -D parcel`
  After this step, we will have a node-modules folder and a package-lock.json file.

Now, in package.json, in the devdependencies, you will see a parcel version denoted like “^2.8.2”. This ^(caret) means that your packages will get updated to the future minor/patch versions, without incrementing the major versions i.e. “^2.8.2” will use releases from 2.8.2 to < 3.0.0.

Similarly, there can also be “~2.8.2”, where `(tilde) denotes that your packages will get updated to the future patch versions, without incrementing the minor versions i.e. “^2.8.2” will use releases from 2.8.2 to < 2.9.0.

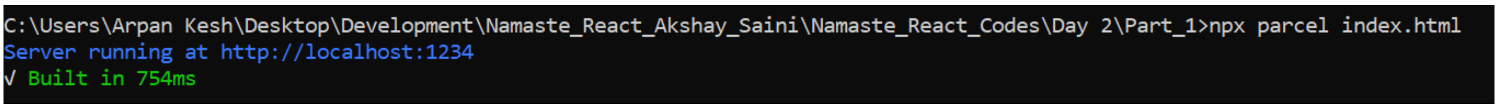
In package-lock.json, for any package, there is a key named version, which stores(or locks) the exact version that the project will be using and it is also useful for installing all the packages for a project, when you clone it from the github. That’s why, we never keep package-lock.json in the .gitignore.
There is another key named integrity, which also stores the exact version of that package, but in Hashed form.

In node-modules folder, there is also a folder named “browserlist”, which helps us to make our app compatible with older versions of any or all browsers (We will see how later on). Node-modules is the heaviest folder in your project and it can also be regenerated exactly with the help of the above 2 json files. That’s why we don’t put node-modules folder in the git repo.

Now, the way we were inserting React earlier into our projects is faulty because we were injecting only a specific major version of React. (like in the project of Day-1, we used React@18). But if React is updated to 19th version, we will not be able to use those features. That’s why we should import React form the packages we have installed in the node-modules folder instead.

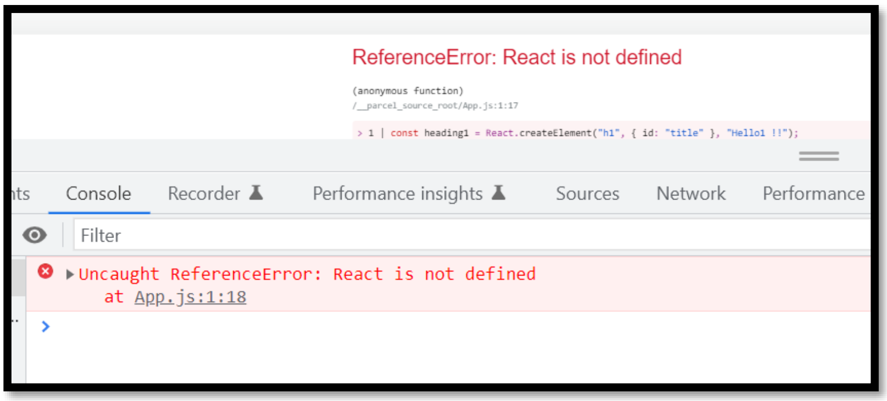
Also, this technique is faster because earlier we were importing the React library from someone else’s CDN using those CDN links, but now we can import the react directly from our local machine/server (i.e. from the node modules folder).

- Now to install react and react-dom npm packages do :- **npm i react react-dom**
- Now, delete the script tags for react and run the app using the command :- **npx parcel index.html**
  - npx is the keyword to execute a command using npm
  - index.html is given as the entry point for our app
- This will give us a server for us. This server is both a Parcel development server which has a feature called Hot Reloading :- If we change anything in our code while our server is running, those changes will be automatically reflected in out browser without -> stopping the server, then making code changes and restarting the server.



```
C:\Users\Arpan Kesh\Desktop\Development\Namaste_React_Akshay_Saini\Namaste_React_Codes\Day 2\Part_1>npx parcel index.html
Server running at http://localhost:1234
√ Built in 754ms
```

- So, we paste the http://localhost:1234/ in our browser and run it. This will also give a reference error in the browser

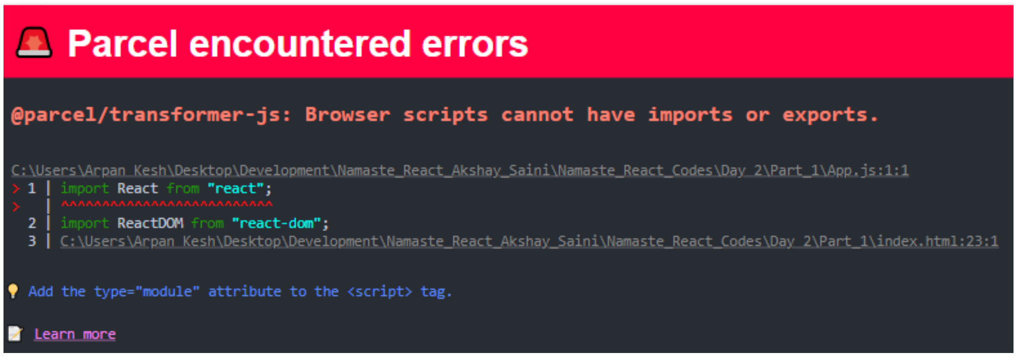


```
ReferenceError: React is not defined

(anonymous function)
/__parcel_source_root/App.js:1:17

> 1 | const heading1 = React.createElement("h1", { id: "title" }, "Hello1 !!");
```

```
Uncaught ReferenceError: React is not defined
    at App.js:1:18
```

This happens because we have removed the script tags and now we can not get the React global variables.

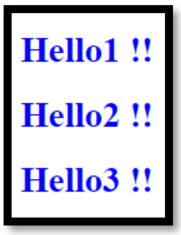
- To get the React and React-DOM global variable, we import it using the import keyword and also add “ type=’module’ “ in the script tag in HTML file where we are calling App.js because import is a EJS module feature. If we don’t use ‘ type=”module” ‘, we will get the below error :-



```
Parcel encountered errors

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

C:\Users\Arpan Kesh\Desktop\Development\Namaste_React_Akshay_Saini\Namaste_React_Codes\Day 2\Part_1\App.js:1:1
> 1 | import React from "react";
>   | ^^^^^^^^^^^^^^^^^^^^^^^^^^
  2 | import ReactDOM from "react-dom";
  3 | C:\Users\Arpan Kesh\Desktop\Development\Namaste_React_Akshay_Saini\Namaste_React_Codes\Day 2\Part_1\index.html:23:1

Add the type="module" attribute to the <script> tag.

Learn more
```

This happens because when browser encounters the script tag with App.js and goes there, it can't understand import. Because App.js is not a normal js file, rather it's a module, so we have to denote that in the script tag with (type= "module").

- Now, if we save our project, the server will automatically show the updated version

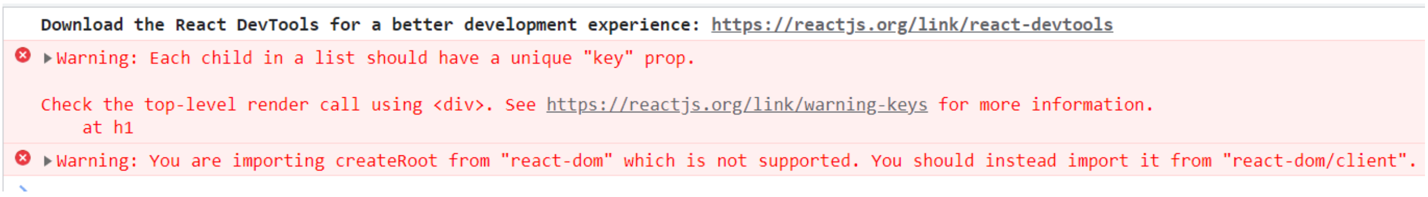


Hello1 !!

Hello2 !!

Hello3 !!

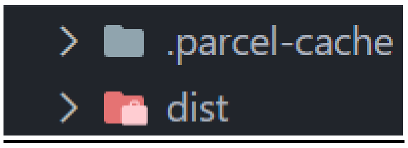
- However, we will get 2 warnings in our console (not in our command line) :- the first one will be handled later. To handle the second one, just change the line where you import react-dom to "react-dom/client" and it will be gone.



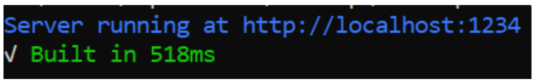
```
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
⊗ ▸Warning: Each child in a list should have a unique "key" prop.

  Check the top-level render call using <div>. See https://reactjs.org/link/warning-keys for more information.
      at h1
⊗ ▸Warning: You are importing createRoot from "react-dom" which is not supported. You should instead import it from "react-dom/client".
```

<u>Important Observations :-</u>

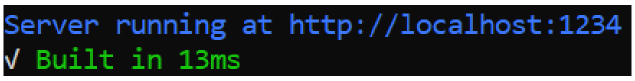
- When we run the "npx parcel index.html" command, 2 new folders are added in the root directory :-_**.parcel-cache** and **dist** (we will talk about them later).



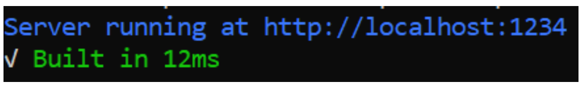
- Every-time you save make no changes or very minute changes and press Ctrl+S to save the project, the project gets reloaded to the server. And every-time you will see that in the cmd line, the time to start the server decreases :-



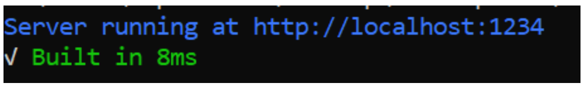
```
Server running at http://localhost:1234
√ Built in 518ms
```



```
Server running at http://localhost:1234
√ Built in 13ms
```



```
Server running at http://localhost:1234
√ Built in 12ms
```

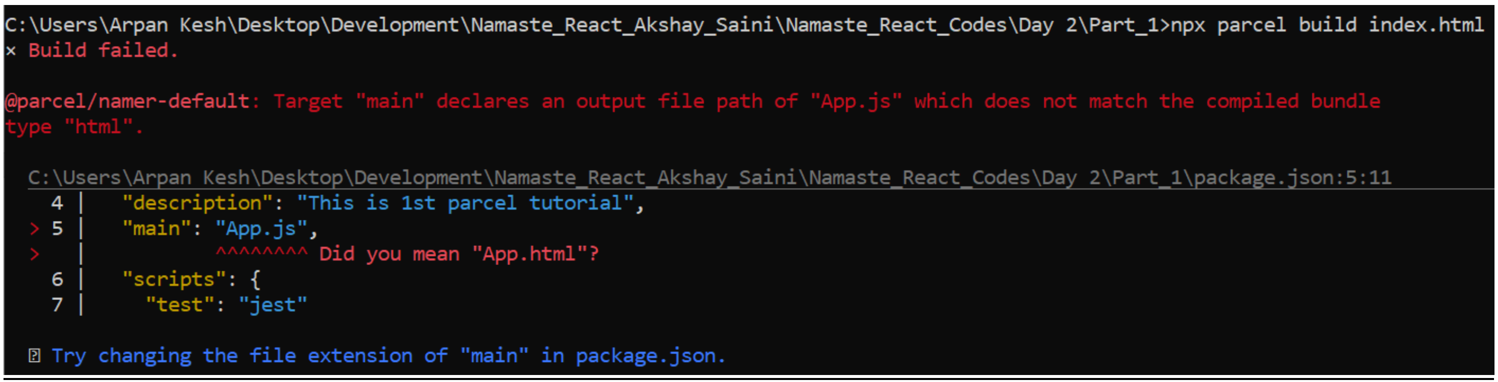


```
Server running at http://localhost:1234
√ Built in 8ms
```

This happens because Parcel uses the concept of Hot Reloading i.e. As you make changes to your code, Parcel automatically rebuilds the changed files and updates your app in the browser. By default, Parcel fully reloads the page, but in some cases it may perform **<u>Hot Module Replacement (HMR</u>**). HMR improves the development experience by 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 in your code.

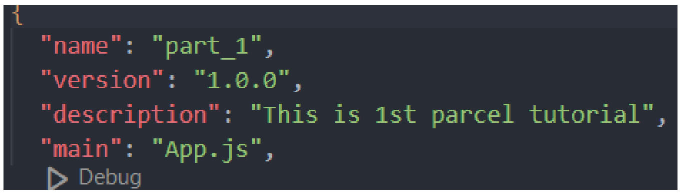
This also happens because Parcel uses the **.parcel-cache** to cache the files, so that it does not have to work from scratch.

- But how does HMR get to know that files have changed? For this, Parcel has a **File Watcher** that uses **File Watching Algorithm (written in C++)** to watch every file in our project root (including node_modules).
- **Dist Folder** :- It creates a faster minimised, development version of our project and serves it to the server. When I tell the parcel to make a production build using "**npx parcel build index.html**", it will modify our files (minifying them, optimising the codes etc.) and push that build in the dist folder. If you run the above command, it might give an error :-



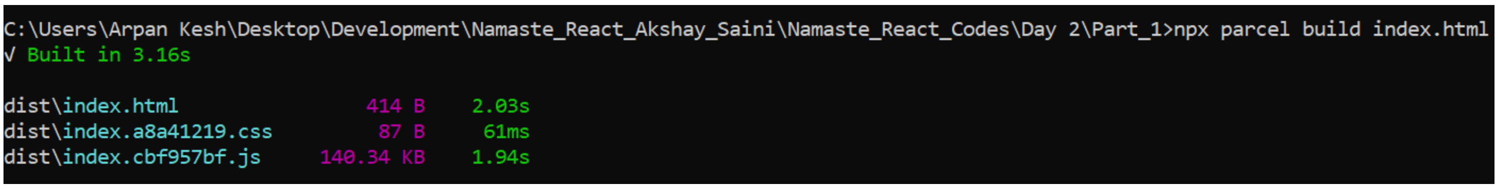
```
C:\Users\Arpan Kesh\Desktop\Development\Namaste_React_Akshay_Saini\Namaste_React_Codes\Day 2\Part_1>npx parcel build index.html
× Build failed.

@parcel/namer-default: Target "main" declares an output file path of "App.js" which does not match the compiled bundle
type "html".

  C:\Users\Arpan Kesh\Desktop\Development\Namaste_React_Akshay_Saini\Namaste_React_Codes\Day 2\Part_1\package.json:5:11
    4 |   "description": "This is 1st parcel tutorial",
  > 5 |   "main": "App.js",
  >   |           ^^^^^^^^ Did you mean "App.html"?
    6 |   "scripts": {
    7 |     "test": "jest"

  Try changing the file extension of "main" in package.json.
```

This error occurs because when setting up the npm, we gave our entry point as App.js, but here we are telling that we want the production build, with parcel, to have index.html as the entry point. To remove the error, just remove the ("main" : "App.js") from the package.json after stopping the server



```
{
  "name": "part_1",
  "version": "1.0.0",
  "description": "This is 1st parcel tutorial",
  "main": "App.js",
  ▷ Debug
```

Then again run the same command, and you will see that it runs successfully and new files have been created in the dist folder. Those files are the production version of our original files.



```
C:\Users\Arpan Kesh\Desktop\Development\Namaste_React_Akshay_Saini\Namaste_React_Codes\Day 2\Part_1>npx parcel build index.html
√ Built in 3.16s

dist\index.html                414 B    2.03s
dist\index.a8a41219.css         87 B     61ms
dist\index.cbf957bf.js     140.34 KB    1.94s
```

Now, you will see that after you run the command "npx parcel index.html", the dist folder will have some files already present before even running the "npx parcel build index.html" command. Those files are dev build files and the files after the build command are the production build files which have actual minification by Parcel. Among those files, there are also files with .map . Those are source maps created by Parcel to tell the browser how to locate the original source code from your bundled code.

Also, if you delete the dist folder, and run either of the 2 above commands, the dist folder will be automatically generated.

- We already know that **.parcel-cache** is used to store the applications necessary to store all the functions that the Parcel apply on out project (HMR, minimising files etc.). If we delete that file and run the "npx parcel index.html" or "npx parcel build index.html", then the cache file will automatically be reloaded in our project just like the dist file.

- We have to put the .parcel-cache and dist folder in our .gitignore because anything that can be autogenerated should not be pushed to git.

- In the package.json, you will see another devDependency named "process". Even though we don't manually install it, it gets automatically injected in our project at some point due to Parcel.

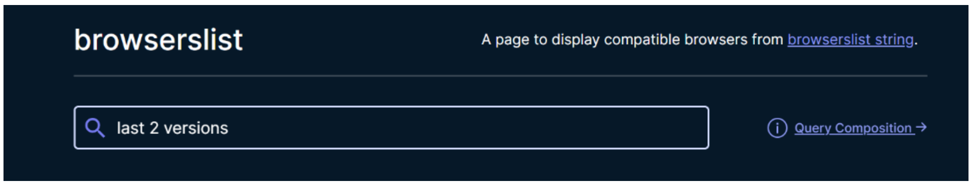
Features of Parcel / What does Parcel do to our project

- HMR – Hot Module replacement
- Implements a File Watcher using File Watcher Algo (written in C++)
- Bundling all the necessary files
- Minifying the code . For ex : All the code in the libraries i.e. react and react-dom, that we imported in App.js will be placed in a single js file, that you can find in the dist folder (there can be multiple js files for a single original js file too, don't worry).
- Cleaning our code. For ex :- It removes all the console.logs from our project in the production build
- Image optimization. If there's an image in our project, parcel will optimize it so that the image does not take too much time to load in the user's website. This is very helpful, since images are the most heavy things in our website.
- Caching while development
- Compression. It compresses the files by renaming the variable names to smaller names and other techniques.
- Compatible with older versions of browsers
- It can automatically handle the port number in which our server runs.
- Uses Consistent Hashing Algorithm.
- It's a Xero Config Module Bundler.

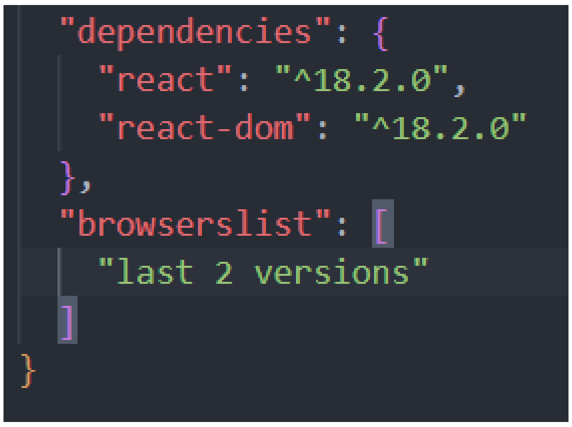
#NOTE :-
React is fast. But to create a React app, React needs many more packages to make it faster like Parcel etc. In turn these packages also need other packages/dependencies. This forms a Dependency Tree and this phenomenon is called Transitive Dependency.

Browserlist

Now how do I make my app compatible with older browsers? For that we have a package named browserslist., which is automatically given to us by Parcel. We can use this package with the help of a key called "browserslist" in the package.json. This key will take an array as it's value.
The values in the array will denote the versions of the specific browsers that our app can run on. We can see that values using a website named browserslist.dev (link :- https://browserslist.dev/?q=bGFzdCAyIHZlcnNpb25z).


browserslist
A page to display compatible browsers from browserslist string.
last 2 versions
Query Composition →

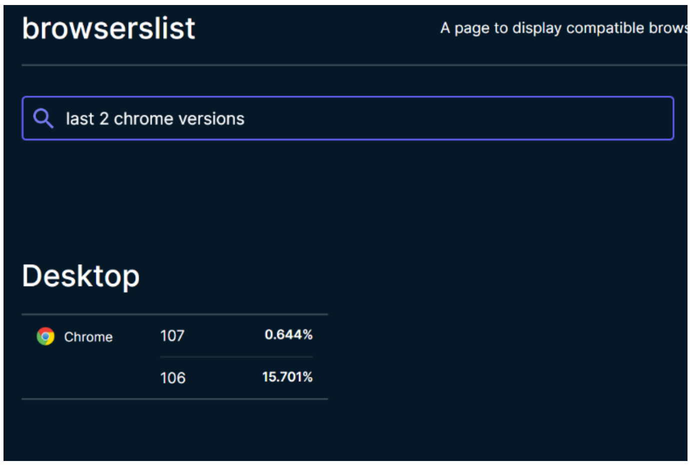
If we put this value in the package.json like this :-



```
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  },
  "browserslist": [
    "last 2 versions"
  ]
}
```

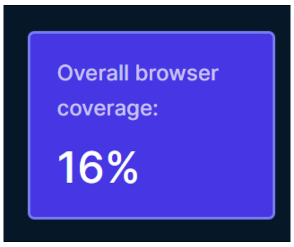
Then Parcel will make sure that my app is working in last 2 versions of every browser. Does not mean that our app will only run on the previous 2 browser versions, rather it will definitely run on 2 previous versions of all browsers and it might run on other versions too. This Similarly we can also do :-

⇨ Last 2 chrome versions (i.e. it will definitely run on 2 previous chrome versions) :-

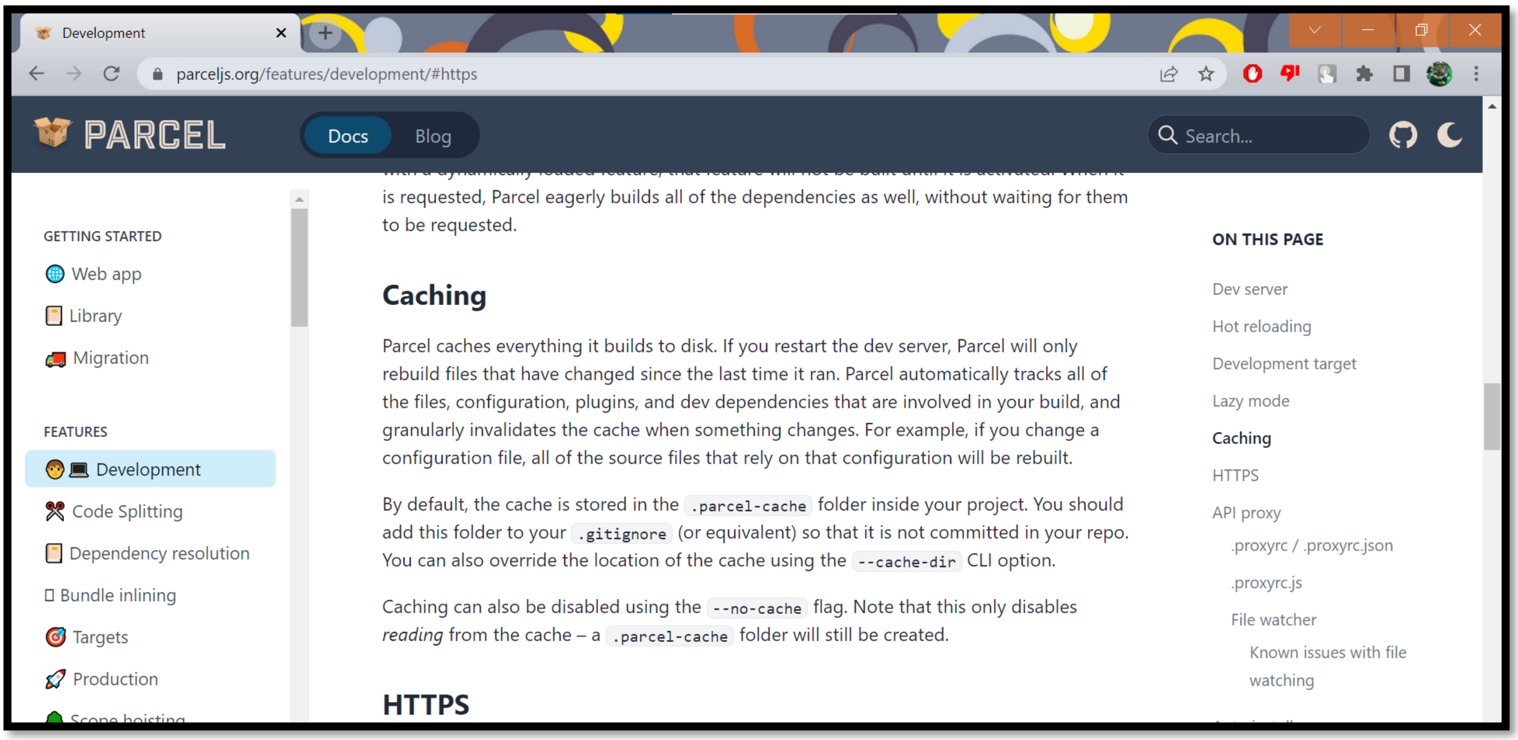


Chrome	107	0.644%
	106	15.701%

But then, you can see in the right side of that webpage that it's showing a mush lesser percentage. That percentage indicates the number of users using that version of chrome.



You can read the features of the Parcel from it's docs. Go to the official docs and click on "Development" in the "Features" section in the right side. There you will get all the functionalities of Parcel. You can explore more in other sections too.



is requested, Parcel eagerly builds all of the dependencies as well, without waiting for them to be requested.

Caching

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.

HTTPS

<u>Tree Shaking (actually taught in Day-3 class) :-</u>

Another feature provided by Parcel. It means removing unwanted code. Suppose our code imports a library which gives us access to a lot of helper functions and even though I use only a few of those functions, all the remaining helper functions are still included in our project. So these unwanted functionalities are ignored by Parcel while making the production build of our app (It just shakes the tree).