#Chapter 2 – Igniting our App

Once we develop an application we need make our app production ready and to do that we need to optimise our code. We need to remove console statements from our code. we need to minify compress, bundle and cache our code. If our application has lots of images, we need to do image optimisation as well. Basically, we have to do a lot of processing to ignite our app for production build.

If we write npx create-react-app in the terminal, it will create an ignited react app having all these super powers in it. But what it takes to build create-react-app.

To understand this let's create our own create-react-app.

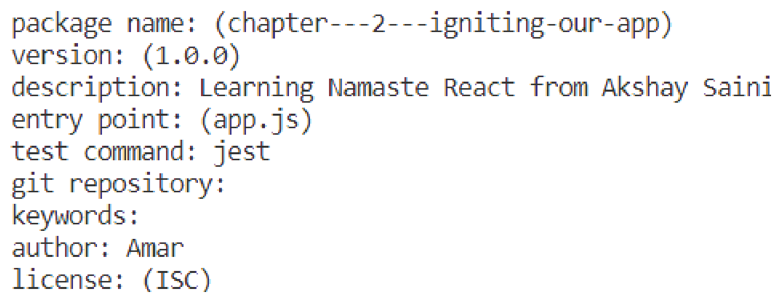
Creating our own create-react-app

React is not the only one which makes our application fast. Along with React there are other helper packages and libraries which make our app scalable, production ready and fast. So, let's install those packages into our app and to do that we need NPM.

A common misconception of NPM is It stands for Node package Manager. As per the official NPM documentation, NPM does not have a full form, it could be anything. for example - No Problem Man. But yes, behind the scene it is acting like a node package manager. NPM is a Standard repository for all the packages, meaning all the packages, libraries and utilities are hosted in NPM Repository.

Let's make our application use NPM. We use a command npm-init in the terminal to initialise NPM. Once NPM initialisation is complete we can use npm to install packages.

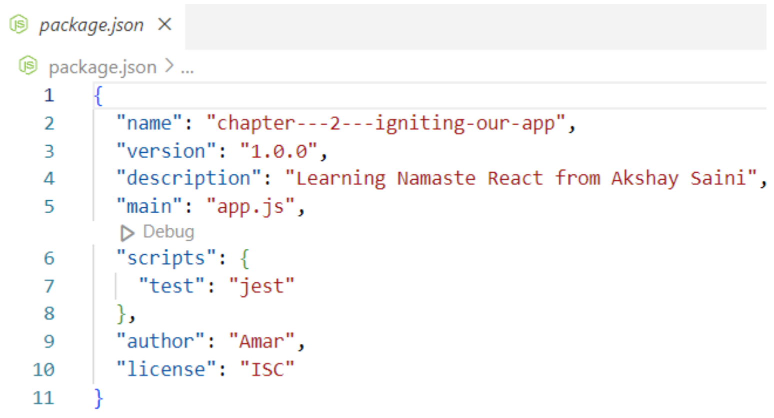
When we use npm-init command in the terminal, it asks certain questions like to enter package name, version, description, entry point, test command, GIT repository, Keywords, author, license etc.



```
package name: (chapter---2---igniting-our-app)
version: (1.0.0)
description: Learning Namaste React from Akshay Saini
entry point: (app.js)
test command: jest
git repository:
keywords:
author: Amar
license: (ISC)
```

Once we provide all the details, it generates a package. json file which contains the application related information. If we don't want to provide such details we can skip this step by using a command npm init -y

At this moment package.json file looks like



```json
{
  "name": "chapter---2---igniting-our-app",
  "version": "1.0.0",
  "description": "Learning Namaste React from Akshay Saini",
  "main": "app.js",
  "scripts": {
    "test": "jest"
  },
  "author": "Amar",
  "license": "ISC"
}
```

These are the details we provided during npm init.

What is Package.Json File?

Package.json file is a configuration for NPM. Whatever packages our project needs, we install those packages using npm install <packageName>.Once package installation is complete, their versions and configuration related information is stored as dependencies inside package. json file.

Earlier we discussed about the requirements to develop a production ready application. All such requirements are fulfilled by bundler.

What is bundler?

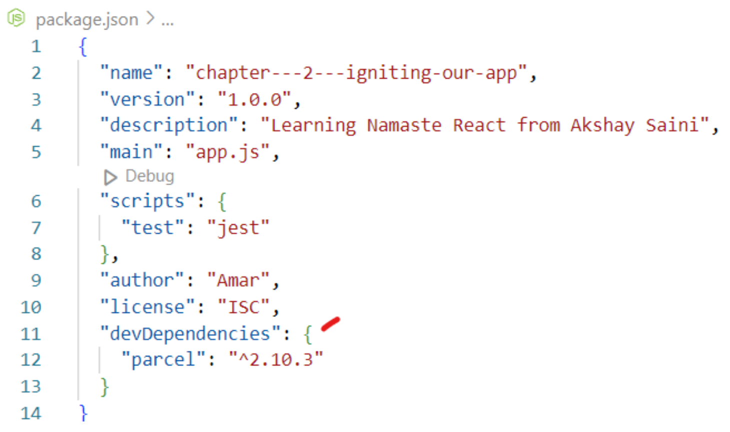
A bundler is a tool that bundles our app, packages our app so that it can be shipped to production. Before shipping our app to production bundler does a lot of optimisation such as code splitting, chunking, image processing, code compression, Tree shaking, caching and a lot of other things.

When we use create-react-app command in the terminal, it uses webpack and babel behind the scene to make the application super-fast and efficient.

Example of bundlers – Parcel, webpack, vite

Let’s use parcel bundler to ignite our app.

Parcel package installation cmd is npm install -D parcel where -D stands for dev dependency. There are two types of dependencies our app can have. One dependency is dev dependency and another one is dependency itself. dev dependency is required in development phase where as dependency is required in production phase.



package.json > ...

```
{
  "name": "chapter---2---igniting-our-app",
  "version": "1.0.0",
  "description": "Learning Namaste React from Akshay Saini",
  "main": "app.js",
  ▷ Debug
  "scripts": {
    "test": "jest"
  },
  "author": "Amar",
  "license": "ISC",
  "devDependencies": {
    "parcel": "^2.10.3"
  }
}
```

If we take a look into package.json file, it will have a devDependencies where parcel version is ^2.10.3.

^ is caret which detects minor update to package version. If there is a minor update ^ will update the minor version inside package.lock file.

If tilde ~ was there in place of ^ this will update the version to the major version inside package.lock.json.

HW - Explore more on ~ and ^ online.

Note: npm install parcel –save -dev is same as npm install parcel -D

What is package.lock.json?

Package.lock.json locks the exact version of packages being used in the project. Not only the exact version but also it takes care of package integrity meaning it keeps track of the package version both in local environment and in server environment through SHA integrity hash code format. This file is sufficient to generate node modules.

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

In package. json we have information about generic version of installed packages whereas in package.lock.json we have information about the specific or exact version of installed packages.

Relation between package.json and package.lock.json?

package.json file is generated after npm-init and updated after npm installation of any packages. Once package.json file is updated there is another file gets created which is package.lock.json

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

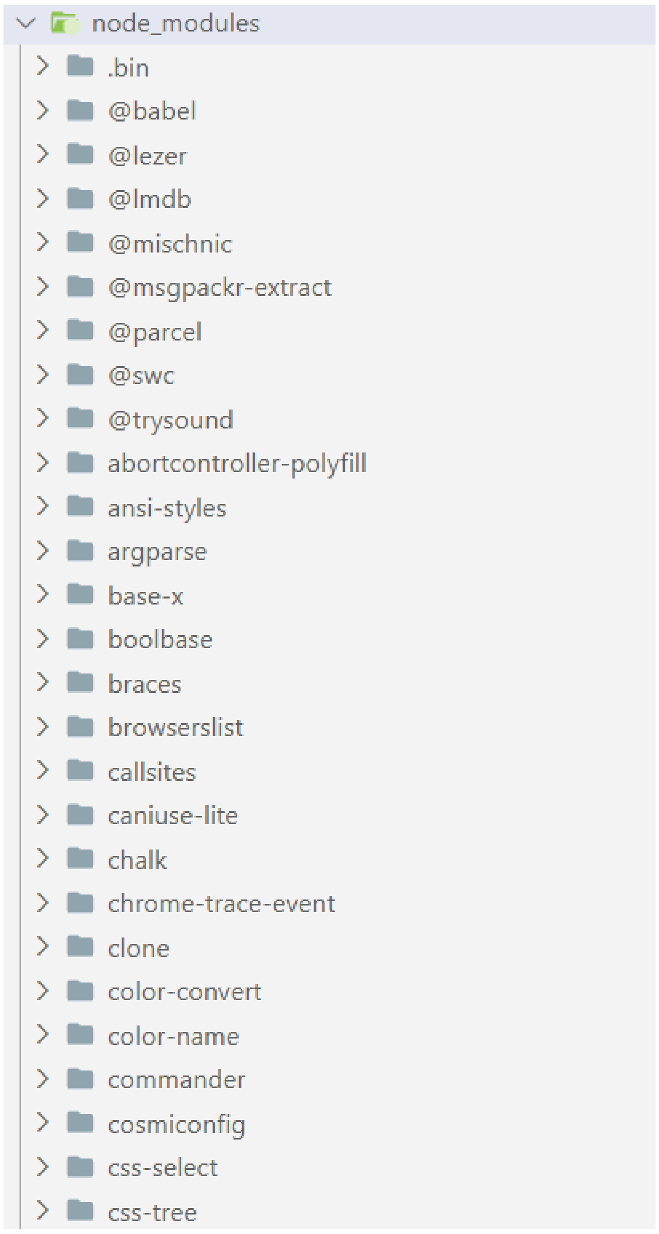
package-lock.json file contains information about the packages dependencies and their versions used in the project. Deleting it would cause dependencies issues in the production environment. So, don't modify it, it's being handled automatically by NPM.

What are node modules? What is Transitive depenancy?

When we run npm install any package, it goes to the web, download the package and push it into the node_module folder. This node module is kind of a database for all packages, our project needs.

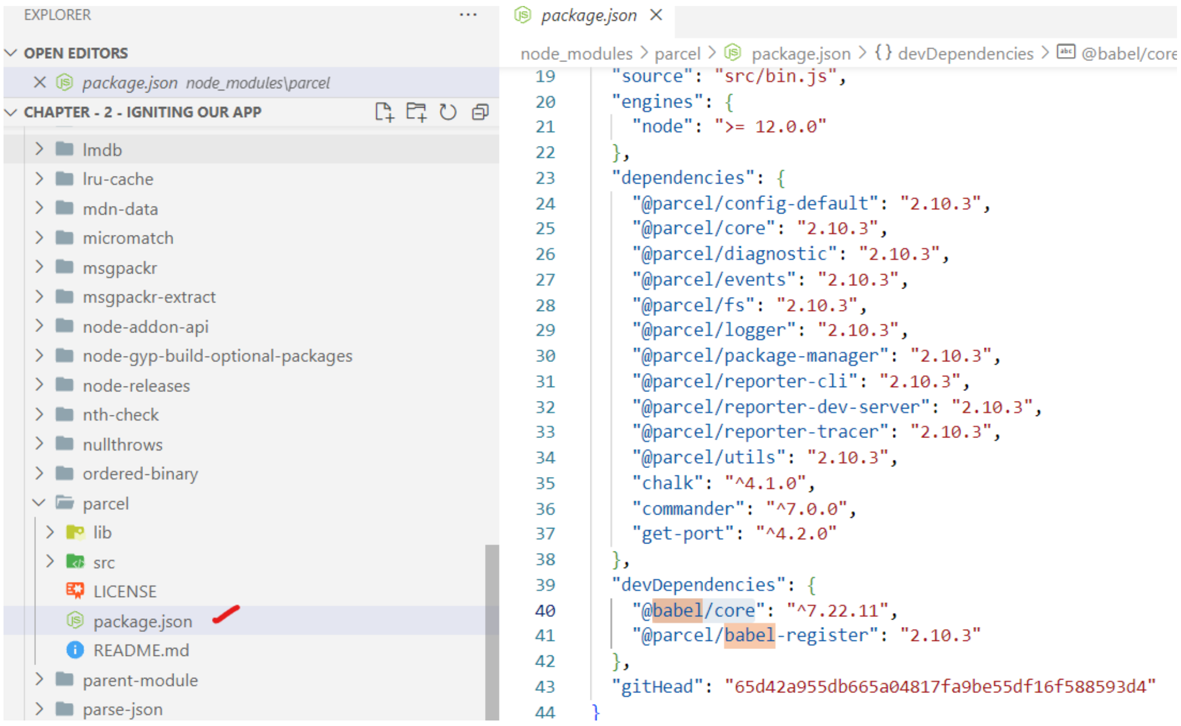
Node modules are very heavy so we should always put this in git ignore.

If we expand node modules folder, we see other packages got installed along with parcel. Why did this happen? This happened because parcel package is dependant on some other packages too. These packages again depend on some other packages and so on. This inter depenancy is called Transitive depenancy.



```
node_modules
  .bin
  @babel
  @lezer
  @lmdb
  @mischnic
  @msgpackr-extract
  @parcel
  @swc
  @trysound
  abortcontroller-polyfill
  ansi-styles
  argparse
  base-x
  boolbase
  braces
  browserslist
  callsites
  caniuse-lite
  chalk
  chrome-trace-event
  clone
  color-convert
  color-name
  commander
  cosmiconfig
  css-select
  css-tree
```

Every package folder inside node modules have their own package.json which stores the dev dependencies and normal dependencies of the packages to which the current package depends upon. Package's package.json basically has information about its transitive depenancy.



```
  "source": "src/bin.js",
  "engines": {
    "node": ">= 12.0.0"
  },
  "dependencies": {
    "@parcel/config-default": "2.10.3",
    "@parcel/core": "2.10.3",
    "@parcel/diagnostic": "2.10.3",
    "@parcel/events": "2.10.3",
    "@parcel/fs": "2.10.3",
    "@parcel/logger": "2.10.3",
    "@parcel/package-manager": "2.10.3",
    "@parcel/reporter-cli": "2.10.3",
    "@parcel/reporter-dev-server": "2.10.3",
    "@parcel/reporter-tracer": "2.10.3",
    "@parcel/utils": "2.10.3",
    "chalk": "^4.1.0",
    "commander": "^7.0.0",
    "get-port": "^4.2.0"
  },
  "devDependencies": {
    "@babel/core": "^7.22.11",
    "@parcel/babel-register": "2.10.3"
  },
  "gitHead": "65d42a955db665a04817fa9be55df16f588593d4"
}
```

What is `. gitignore`? What should we `add and not add` into it?

The . gitignore file is a text file that tells GIT which files or folders to ignore in a project during commit to the repository.

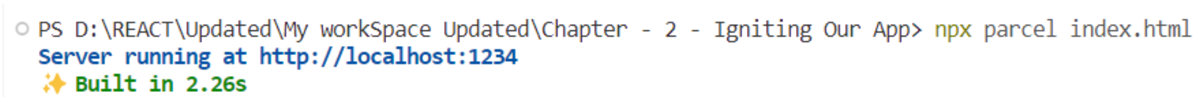
All autogenerated files/folders should go inside gitignore. For security, the security key files and API keys should get added into .gitignore file where

* is used as a wildcard match
/ is used to ignore pathnames relative to the. gitignore file
# is used to add comments to a .gitignore file

```
# Ignore node_modules folder
/node_modules

# Ignore all text files
*.txt

# Ignore files related to API keys
.env

# Ignore SASS config files
.sass-cache
```

Note: package.json and package.lock.json should not be added into .gitignore file. Because it is needed by GIT to generate node modules out of it.

Parcel has been installed and we need parcel to ignite / boot up our app using npx parcel index.html



```
PS D:\REACT\Updated\My workSpace Updated\Chapter - 2 - Igniting Our App> npx parcel index.html
Server running at http://localhost:1234
✨ Built in 2.26s
```

When we execute this command, what happens behind the scene is parcel goes to the entry point or source file index.html and makes a development build for our app and hosts that build in http://localhost:1234/. Parcel created a server for us where the app is hosted at http://localhost:1234/. That's why we are able to access our application over http://localhost:1234/

Difference between npm and npx

If we need to install a package, we need to use npm install<packageName>
If we need to execute a package, we use npx <packageName><entryPoint>
In npx parcel index.html, npx stands for execute, index.html is the entry point.

Why CDN links are not preferable to bring react and ReactDOM into our project?

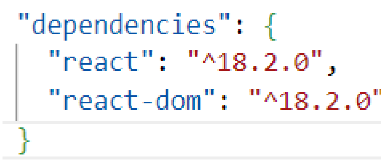
* Because taking react source code from CDNs is costly operation. when browser encounters script tags where CDN links are imported, it will make a network call and get the react source code from unpkg.com.

https://unpkg.com/react@18/umd/react.development.js
https://unpkg.com/react-dom@18/umd/react-dom.development.js

If we already have react and react-dom in node modules, it will be very easy to use react in our code. we don't have to make any network calls.

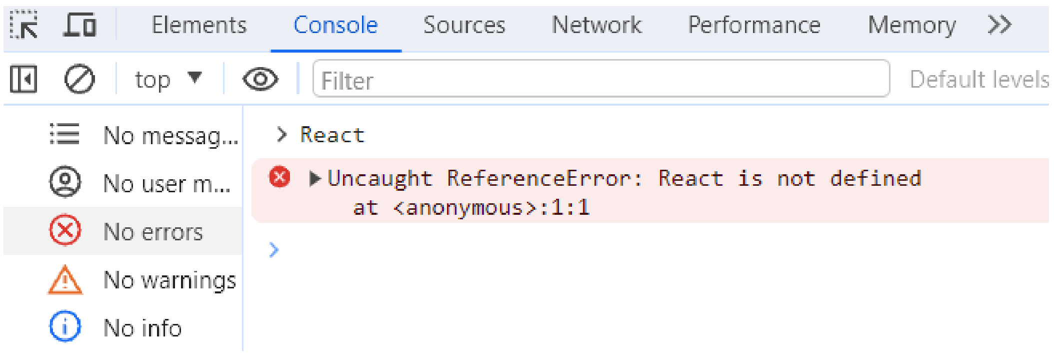
* Another reason for not using CDN is react version. what if react version is updated and we need the updated version in our code. In that case we have to modify our CDN links which is not ideal.

Let's install react and react-dom into our project using, npm install react and npm install react-dom. These are normal dependencies, not dev dependencies.



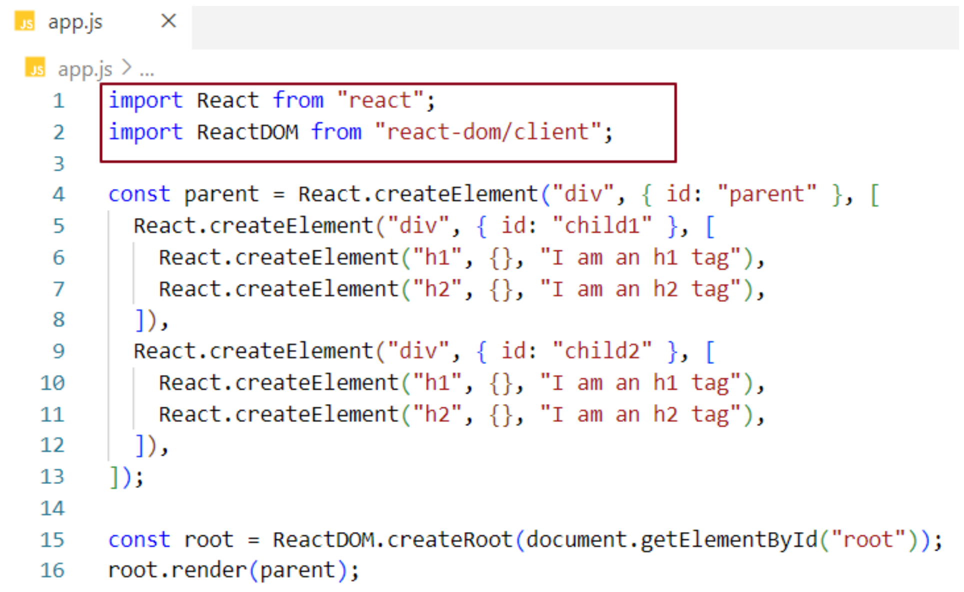
```
"dependencies": {
  "react": "^18.2.0",
  "react-dom": "^18.2.0"
}
```

Now if we run our application, we get an error stating React is not defined.



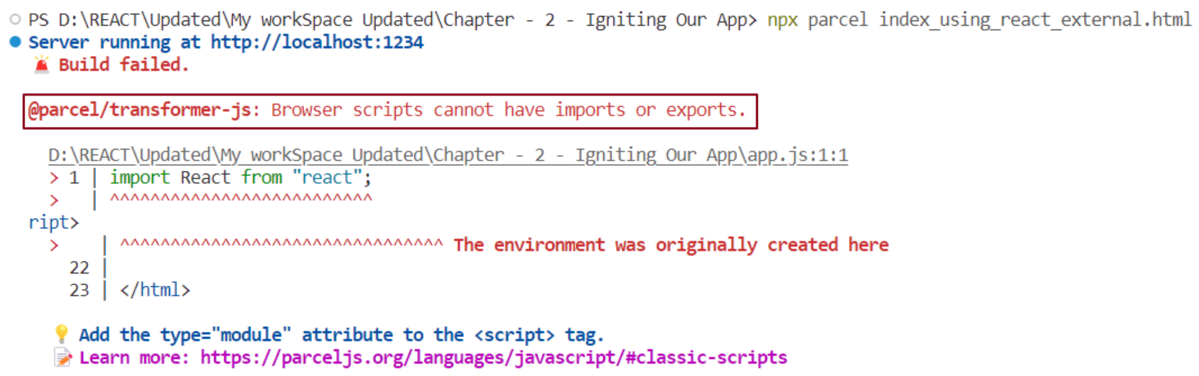
```
> React
✖ ▶Uncaught ReferenceError: React is not defined
    at <anonymous>:1:1
```

The error is very genuine because our code does not understand where this React is coming from. we have installed this package but we have not imported it yet in our project. To do that we use import key word to import React class from the package react. Similarly, we need to import ReactDOM from react-dom/client to use full features of react DOM.



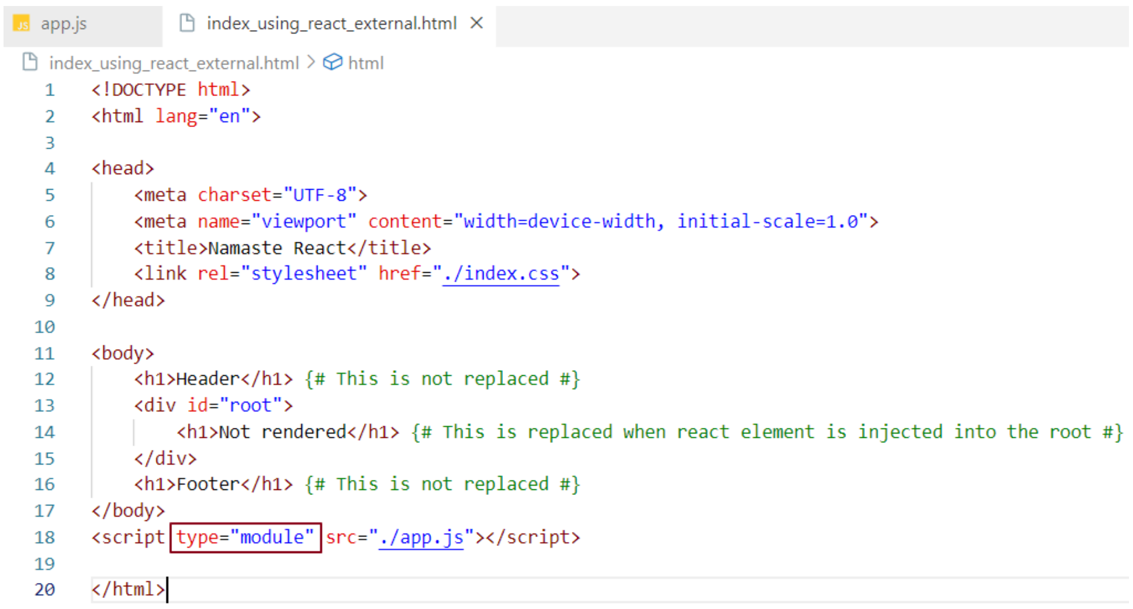
```javascript
import React from "react";
import ReactDOM from "react-dom/client";

const parent = React.createElement("div", { id: "parent" }, [
  React.createElement("div", { id: "child1" }, [
    React.createElement("h1", {}, "I am an h1 tag"),
    React.createElement("h2", {}, "I am an h2 tag"),
  ]),
  React.createElement("div", { id: "child2" }, [
    React.createElement("h1", {}, "I am an h1 tag"),
    React.createElement("h2", {}, "I am an h2 tag"),
  ]),
]);

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(parent);
```

We are not done yet. We still will face one more issue while running the app using parcel.



```
○ PS D:\REACT\Updated\My workSpace Updated\Chapter - 2 - Igniting Our App> npx parcel index_using_react_external.html
● Server running at http://localhost:1234
  🚨 Build failed.

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

    D:\REACT\Updated\My workSpace Updated\Chapter - 2 - Igniting Our App\app.js:1:1
    > 1 | import React from "react";
    >   | ^^^^^^^^^^^^^^^^^^^^^^^^^^
  ript>
    >   | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ The environment was originally created here
     22 |
     23 | </html>

    💡 Add the type="module" attribute to the <script> tag.
    📝 Learn more: https://parceljs.org/languages/javascript/#classic-scripts
```

This error is because our browser cannot understand imports and exports. Our browser only understands JavaScript. While reading app.js browser finds these imports and exports statements and throw this error. To tell browser that app.js is not a normal JavaScript file, it's a module, we need to use type="module" inside the script tag and this will fix the error.



app.js
index_using_react_external.html
index_using_react_external.html > html

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Namaste React</title>
    <link rel="stylesheet" href="./index.css">
</head>

<body>
    <h1>Header</h1> {# This is not replaced #}
    <div id="root">
        <h1>Not rendered</h1> {# This is replaced when react element is injected into the root #}
    </div>
    <h1>Footer</h1> {# This is not replaced #}
</body>
<script type="module" src="./app.js"></script>

</html>
```

Now React and ReactDOM are available to browser.

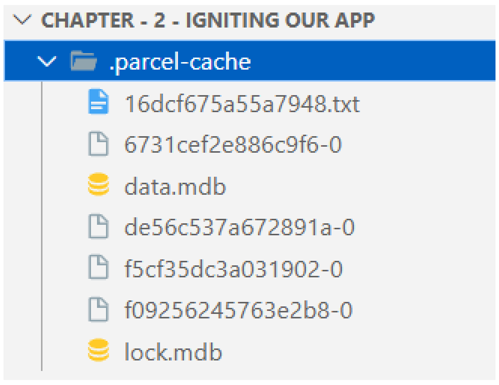
When we make any changes in our code, as soon as we save our files, the page gets automatically refreshed and the changed content will be displayed in UI. This is taken care by Parcel. Behind the scene parcel is doing HMR.

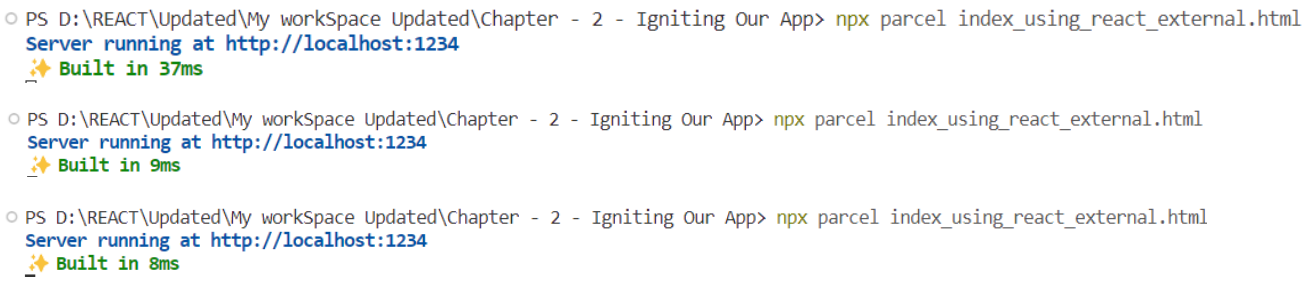
What is HMR (Hot Module Replacement)?

Parcel is reading all the files, it keeps a track of all of them using an algorithm called File watcher which internally uses c++. Once parcel detects any change in code it quickly updates that change in UI using HMR.

What is Caching? Where parcel stores caches?

Parcel caches code all the time. When we run the application, a build is created which takes some time in ms. If we make any code changes and save the application, another build will be triggered which might take even less time than the previous build. This reduction of time is due to parcel cache. Parcel immediately loads the code from the cache every time there is a subsequent build. On the very first build parcel creates a folder. parcel-cache where it stores the caches in binary code format. Parcel gives faster build, faster developer experience because of caching


```
CHAPTER - 2 - IGNITING OUR APP
  .parcel-cache
    16dcf675a55a7948.txt
    6731cef2e886c9f6-0
    data.mdb
    de56c537a672891a-0
    f5cf35dc3a031902-0
    f09256245763e2b8-0
    lock.mdb
```


```
PS D:\REACT\Updated\My workSpace Updated\Chapter - 2 - Igniting Our App> npx parcel index_using_react_external.html
Server running at http://localhost:1234
✨ Built in 37ms

PS D:\REACT\Updated\My workSpace Updated\Chapter - 2 - Igniting Our App> npx parcel index_using_react_external.html
Server running at http://localhost:1234
✨ Built in 9ms

PS D:\REACT\Updated\My workSpace Updated\Chapter - 2 - Igniting Our App> npx parcel index_using_react_external.html
Server running at http://localhost:1234
✨ Built in 8ms
```

Images are very heavy to be loaded in to the DOM. Parcel does image optimisation as well. Parcel does code compression, minification, bundles code.

React is not the only library which makes the application fast, our application needs other helper packages such as bundlers and babel etc to make itself superfast.

Parcel uses consistence hashing algorithm, code splitting, differential bundling (to support older browser), diagnostic to show parcel errors and tree shaking to make application fast.

Parcel provides https to test the application in https. parcel manages port number.

Tree shaking

Tree shaking is a process of removing the unwanted code that we do not use while developing the application. In computing, tree shaking is a dead code elimination technique that is applied when optimizing code.

When we import a library and we use some specific functionalities of that library in our code. While parcel bundles things up, it just bundles the used functions and removes the unused function from the library using Tree shaking.

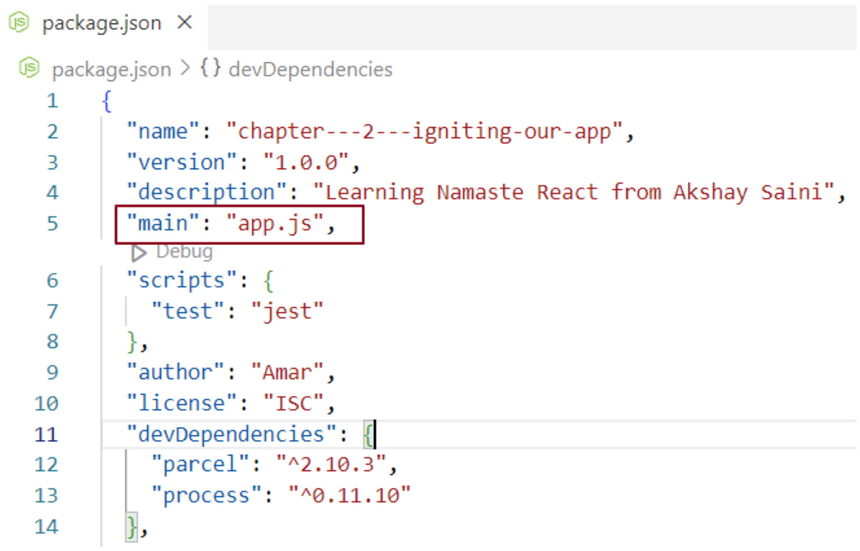
Parcel features

* HMR (Hot Module Replacement)
* File watcher algorithm - made with C++
* Minification
* Cleaning our code
* DEV and production Build
* Super-fast building algorithm
* Image optimization
* Caching while development
* Compresses files
* Compatible with older version of browser
* provides functionalities using which we can enable HTTPS in dev, because https allows certain features that http can't. (npx parcel <entry_point> --https)
* parcel handles Port Number
* parcel uses Consistent hashing algorithm to do all the bundling.
* Parcel requires Zero Configuration. We don't have to install any third-party apps or packages or anything to use parcel.
* Automatic code splitting
* Parcel created a server for us (More like a live server)

How to create a production build?

Prod build - npx parcel build index.html (more optimised than dev build)

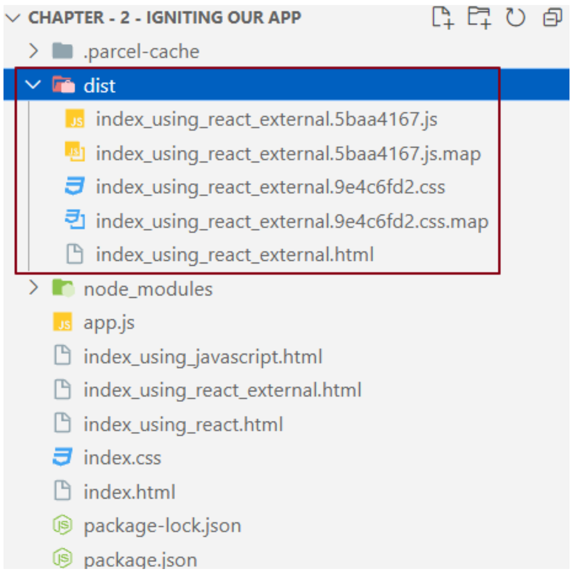
Since we have provided entry point as index.html, it does not match with the one in package.json which is app.js. So, remove below highlighted config from package file before build.



```json
{
  "name": "chapter---2---igniting-our-app",
  "version": "1.0.0",
  "description": "Learning Namaste React from Akshay Saini",
  "main": "app.js",
  "scripts": {
    "test": "jest"
  },
  "author": "Amar",
  "license": "ISC",
  "devDependencies": {
    "parcel": "^2.10.3",
    "process": "^0.11.10"
  },
```

What is the dist folder?

When bundler builds the app, the build goes into a folder called dist. The `/dist` folder contains the minimized and optimised version of the source code. The code present in the `/dist` folder is actually the code which is used on production web applications. Along with the minified code, the /dist folder also comprises of all the compiled modules that may or may not be used with other systems.



```
CHAPTER - 2 - IGNITING OUR APP
  > .parcel-cache
  v dist
      index_using_react_external.5baa4167.js
      index_using_react_external.5baa4167.js.map
      index_using_react_external.9e4c6fd2.css
      index_using_react_external.9e4c6fd2.css.map
      index_using_react_external.html
  > node_modules
    app.js
    index_using_javascript.html
    index_using_react_external.html
    index_using_react.html
    index.css
    index.html
    package-lock.json
    package.json
```

Note: Always put parcel cache and dist in git ignore as they are auto generated code.

What is browserslist?

Browserslist is a tool that specifies which browsers should be supported/compatible in your frontend app by specifying "queries" in a config file. It's used by frameworks/libraries such as React, Angular and Vue, but it's not limited to them.

```
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
}
```

This does not mean that our app will only support in last 1 version of Firefox, chrome and safari. It means our app will 100% work on last version of Firefox chrome and safari.

We have created our own create-react-app.