1_2_introduction

11 September 2023 08:14 PM

What is Library and framework?
React is a library made by facebook, similar to jQuery
Carousel is a JS library
Framework is complete in itself, it has everything needed to create an app

A framework is a set of pre-written code that provides a structure for developing software applications. A library, on the other hand, is a collection of pre-written code that can be used to perform specific tasks

Both the framework vs library is precoded support programs to develop complex software applications. However, **libraries target a specific functionality, while a framework tries to provide everything required to develop a complete application**.

It takes minimum effort for a library to put in inside our code

What is emmet?
Emmet is a set of plug-ins for text editors that allows for high-speed coding and editing in HTML, XML, XSLT, and other structured code formats via content assist.

How to create a H1 using JS and put it inside a div with id root?
Browser has a JS engine which interprets the code written below and react accodingly

```
  <title>Namaste React</title>
</head>
<body>
  <div id="root"></div>
</body>
<script>

  const heading = document.createElement("h1");

  heading.innerHTML = "Namaste Everyone from JavaScript!";

  const root = document.getElementById("root");

  root.appendChild(heading);
```

What is React CDN?
Content delivery/distribution network
A content delivery network (CDN) is **a network of interconnected servers that speeds up webpage loading for data-heavy applications**. CDN can stand for content delivery network or content distribution network.

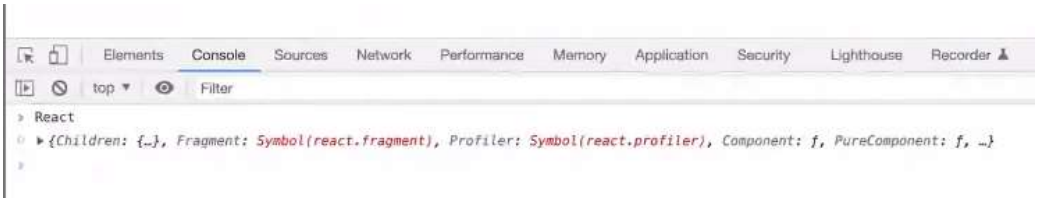
What is cross origin in Script tag?

```
<script
  crossorigin
  src="https://unpkg.com/react@18/umd/react.development.js"
></script>
<script
  crossorigin
  src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
></script>
```

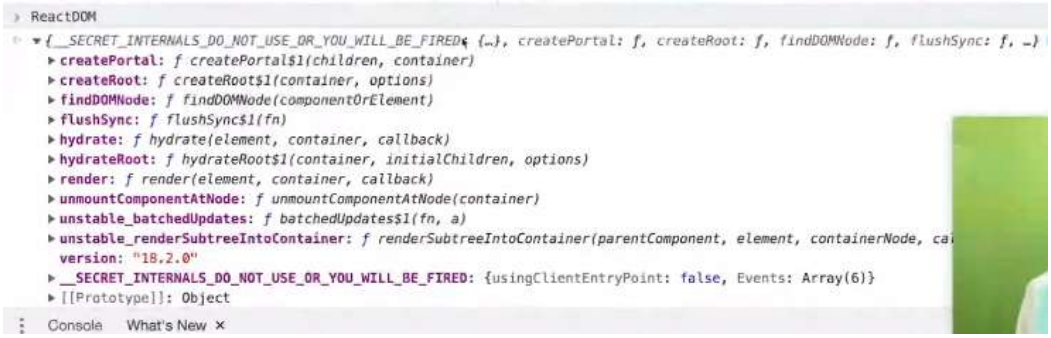
The crossorigin attribute **sets the mode of the request to an HTTP CORS Request**. Web pages often make requests to load resources on other servers. Here is where CORS comes in. A cross-origin request is a request for a resource (e.g. style sheets, iframes, images, fonts, or scripts) from another domain

Shortest Program of React?
The below is shortest program of react, we have just injected react CDN into our document

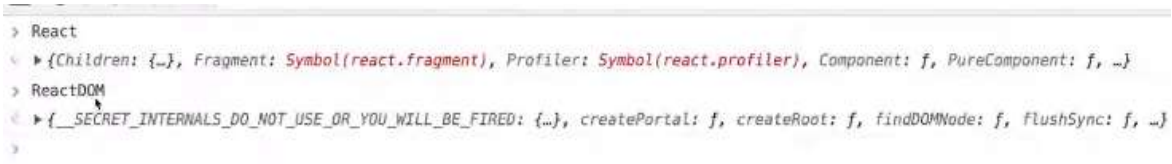
Now if we write React in console we get this



We can also use React.createContext etc etc in console now.
React is a global object and it can be used anywhere using React.something anywhere because we have added react CDN now
We also access to React DOM now



Use of those 2 CDN links were



Why there are 2 files for React and ReactDOM?
React is not limited to browsers only there is React native also for mobiles
ReactDOM means web version of React which gives us access to DOM

Let us use React Now, do same H1 thing using React

We create h1 now like this:

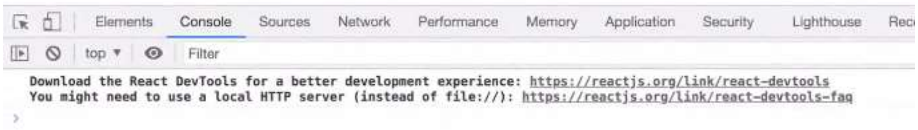
```
<script>

  const heading = React.createElement("h1",{}, "Namaste Everyone!");
```

Now we want to render this heading inside div of id = "root"
We use const root = ReactDOM.createRoot(document.getElementByid('root')) to tell react that this is my root
Now we use root.render(heading)

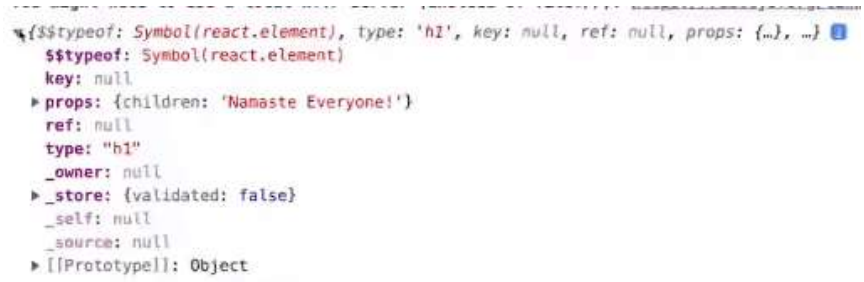
```
<script>
  const heading = React.createElement("h1", {}, "Namaste Everyone!");

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

  root.render(heading);
</script>
```

Output is

Namaste Everyone!



If we do console.log(heading)
We see an object of type h1
render() injects element in to DOM



Can we have multiple roots?
Generally in our react app, we have only 1 root and 1 render method

What if we have something on top of div with Id = root?
Let say we make div of id header
And div of Id = root
And div of Id = footer
What will be output?

Everything will run as it is just that React will be rendered inside root only. There can be header or footer also. So we can use react anywhere in our project like search bar, footer etc etc

Header

Namaste Everyone!

Footer

What is { } in React.createElement?

```
const heading = React.createElement("h1", {}, "Namaste Everyone!");
```

Let say we have one more heading with id = title inside div with id = root

```
<body>
  <div id="root">

      <h1 id="title">Hello world</h1>
  </div>
</body>
<script
  crossorigin
  src="https://unpkg.com/react@18/umd/react.development.js"
></script>
<script
  crossorigin
  src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
></script>
<script>
  const heading = React.createElement("h1", {
      id
  }, "Namaste Everyone!");
```

So to make changes in id = title we can pass these parameters inside { }

```
const heading = React.createElement(
  "h1",
  {
    id: "title",
  },
  "Namaste Everyone!"
);
```

Our DOM now looks like

```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body>
  ▼<div id="root">
      <h1 id="title">Namaste Everyone!</h1> == $0
    </div>
    <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  ▶<script>…</script>
  </body>
</html>
```

Now if we fill root with many h1 headings
Now if we render heading "Namaste Everyone" inside root what will happen?

React will overwrite everyhitng and only Namaste Everyone will be there inside root

```
<body>
  <div id="root">
    <h1 id="title">Hello world</h1>
    <h1 id="title">Hello world</h1>
    <h1 id="title">Hello world</h1>
    <h1 id="title">Hello world</h1>
    <h1 id="title">Hello world</h1>
    <h1 id="title">Hello world</h1>
    <h1 id="title">Hello world</h1>
  </div>
</body>
```

Output

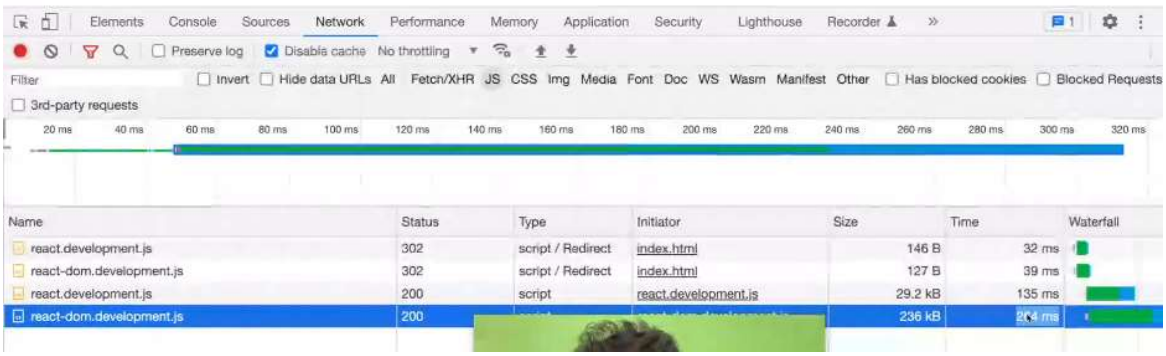
Namaste Everyone!

We generally write "Not Rendered Correctly" inside root div
To make sure that if there is some error we see that message and can solve error that why root not rendered correctly

```
<body>
  <div id="root">Not Rendered</div>
</body>
```

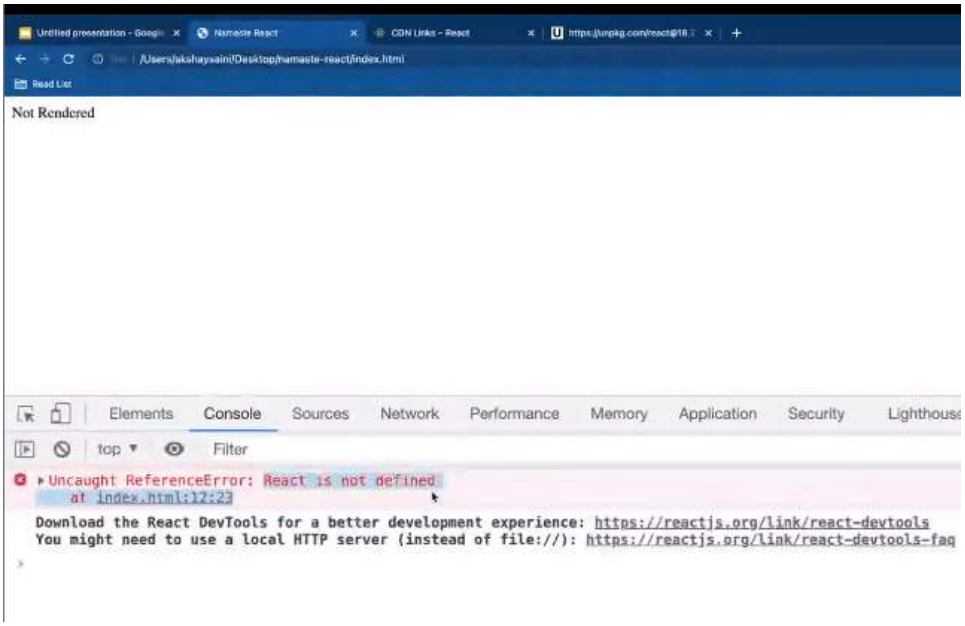
It takes some time for react to render in our browser so we always see "Not Rendered Correctly" for sometime on doing refresh.
It takes sometime for scripts to load

Namaste Everyone!



What if we put script tag inside body below div id = root?
It will not work.

```
<body>
  <div id="root">Not Rendered</div>
  <script>
    const heading = React.createElement(
      "h1",
      {
        id: "title",
      },
      "Namaste Everyone!"
    );

    console.log(heading);

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

    //passing a react element inside the root
    root.render(heading);
  </script>
  <script
    crossorigin
    src="https://unpkg.com/react@18/umd/react.development.js"
  ></script>
  <script
    crossorigin
    src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
  ></script>

</body>
```



What is difference between async/defer?

Async allows your script to run as soon as it's loaded, without blocking other elements on the page. Defer means your script will only execute after the page has finished loading. In most cases, async is the better option — but there are exceptions

Async in script tag is a way to load scripts asynchronously. That means, if a script is async, it will be loaded independently of other scripts on the page, and will not block the page from loading.
If you have a page with several external scripts, loading them all asynchronously can speed up the page load time, because the browser can download and execute them in parallel.

To use async, simply add the async attribute to your script tag:

```
<script async src="script.js"></script>
```

By using the defer attribute in HTML, the browser will load the script only after parsing (loading) the page. This can be helpful if you have a script that is dependent on other scripts, or if you want to improve the loading time of your page by loading scripts after the initial page load.

To use defer, simply add the defer attribute to your script tag:

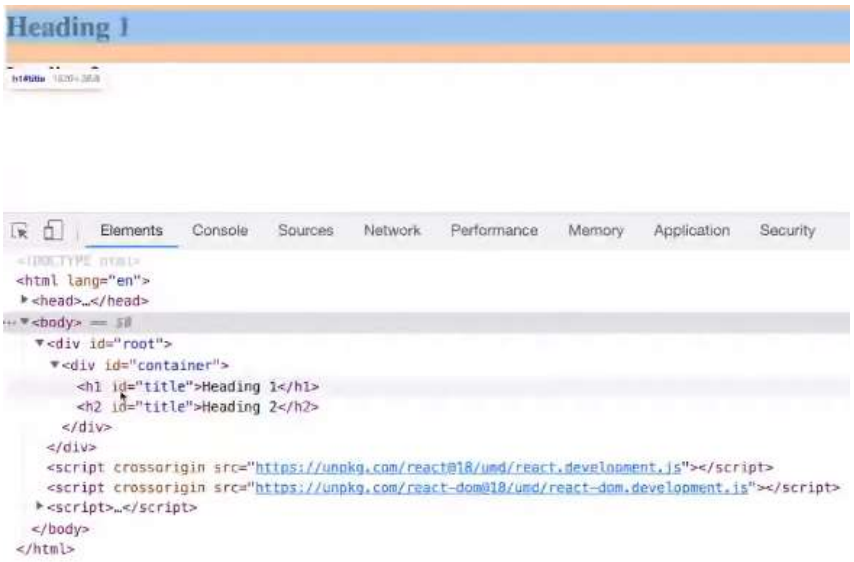
```
<script defer src="script.js"></script>
```

If we want to build the below structure in our HTML using React. How to do it?

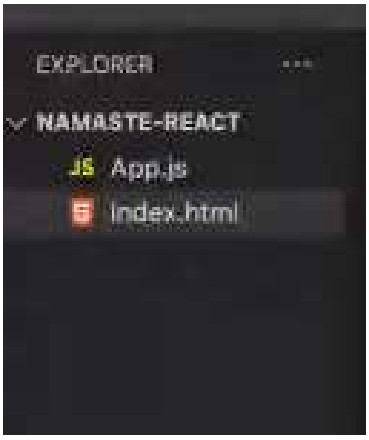
We do it like:
We will pass heading 1 and heading 2 as an Array
We put heading 1 and heading 2 inside container
Now we render the container inside root

```
<script>
  const heading = React.createElement(
    "h1",
    {
      id: "title",
    },
    "Heading 1"
  );

  const heading2 = React.createElement(
    "h2",
    {
      id: "title",
    },
    "Heading 2"
  );

  const container = React.createElement(
    "div",
    {
      id: "container",
    },
    [heading, heading2]
  );

  console.log(heading);

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

  //passing a react element inside the root

  //async defer
  root.render(container);
```

Our DOM looks like:



If we want to build a big index.html
The above method is not development-friendly
React came with a idea to build HTML using JS
We need not to go to HTML anymore, we can do anything from React using APIs like createElement() etc etc.
To make a complex file, its better to split our components and put all JS in app.js



Why do we import CSS inside head in HTML?
seeks to reduce the number of times the browser must re-flow the document by ensuring that the CSS styles are all parsed in the head, before any body elements are introduced.
This is based on the best practice for optimizing browser rendering.

What is rel = stylesheet in link tag while linking CSS in HTML?
The REL attribute is used **to define the relationship between the linked file and the HTML document**. REL=StyleSheet specifies a persistent or preferred style while REL="Alternate StyleSheet" defines an alternate style. A persistent style is one that is always applied when style sheets are enabled.

CDN of react.development.js is for development
CDN of react.production.js has same code but it is much more optimised and for production

Session 2

Revising JS

What is function keyword in JS?
It is present inside JS,

Data Structure used for memory in JS is Heap

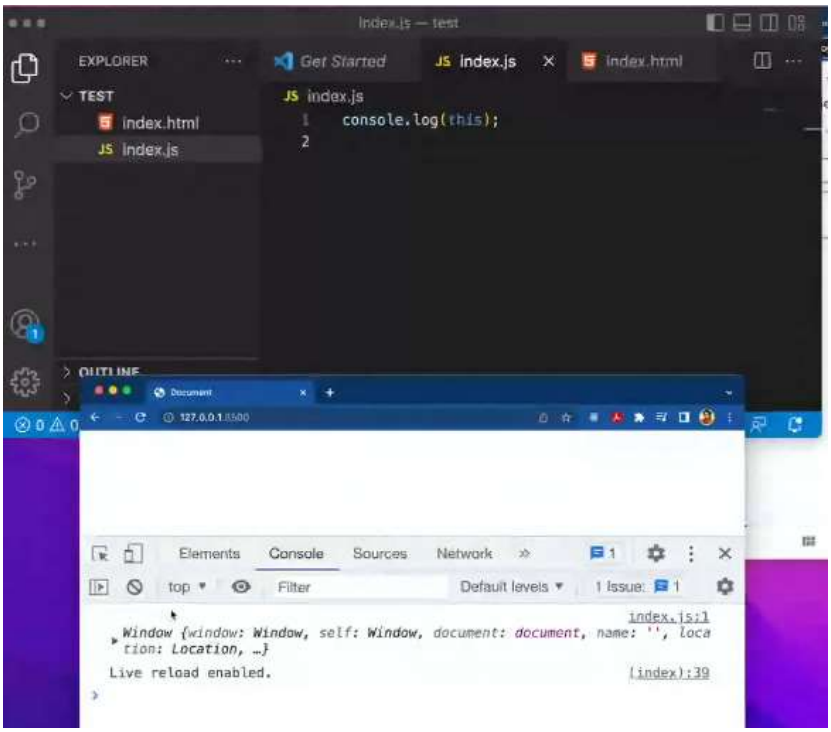
Function without name is anonymous function which can be assigned to a variable as well

```
function x() {
    const a = 10;
}
var xyz = 30;

x(); //functional excution context is created

var x = function () {
    console.log("I'm an anonymous function");
}
```

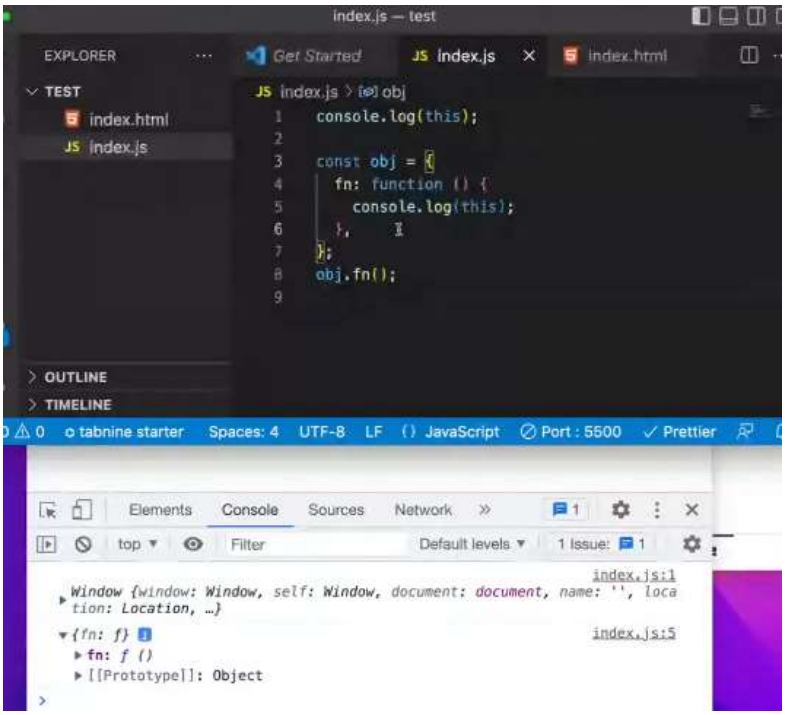
Expression is something that executes. Like console.log is a expression but a = 30 is not a expression.

Arrow function was introduced in ES6 with let, const, promises, spread operator etc etc
=> is known as fat-arrow
Only difference between normal function and arrow function is 'this' keyword
Arrow function, 'this' refers to window

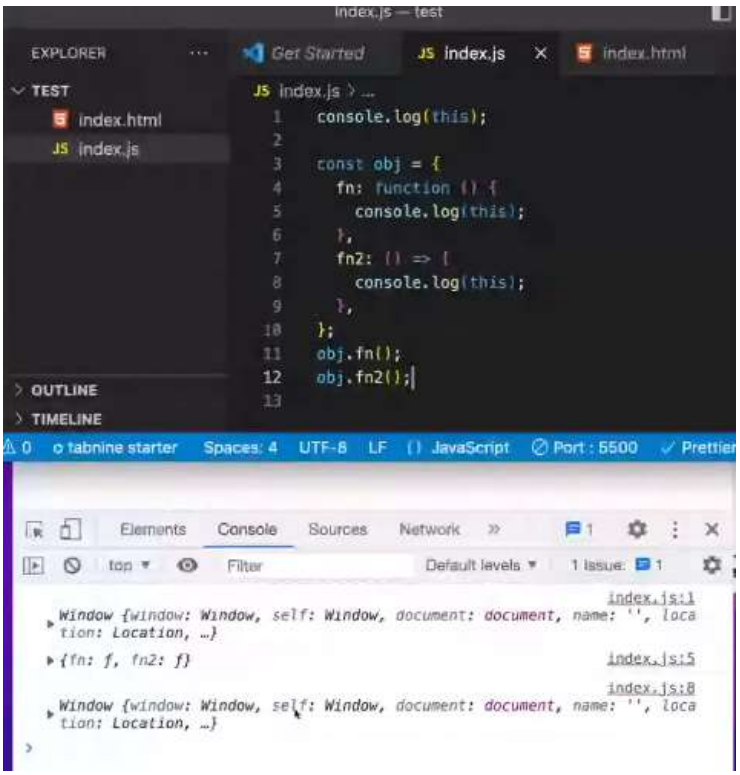
Output of console.log(this) is window object



In normal function 'this' refer to parent object

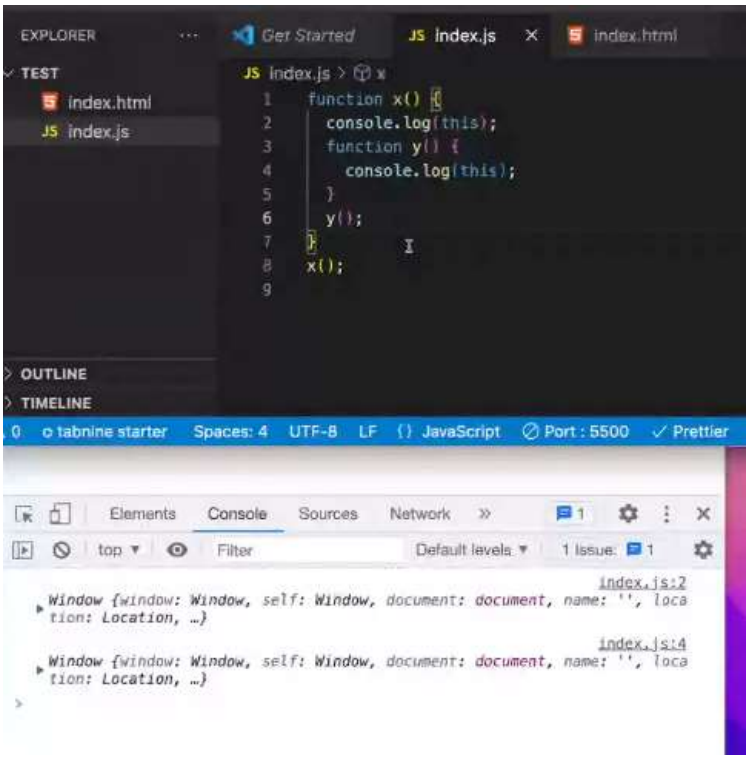


But for arrow function it refers to window object

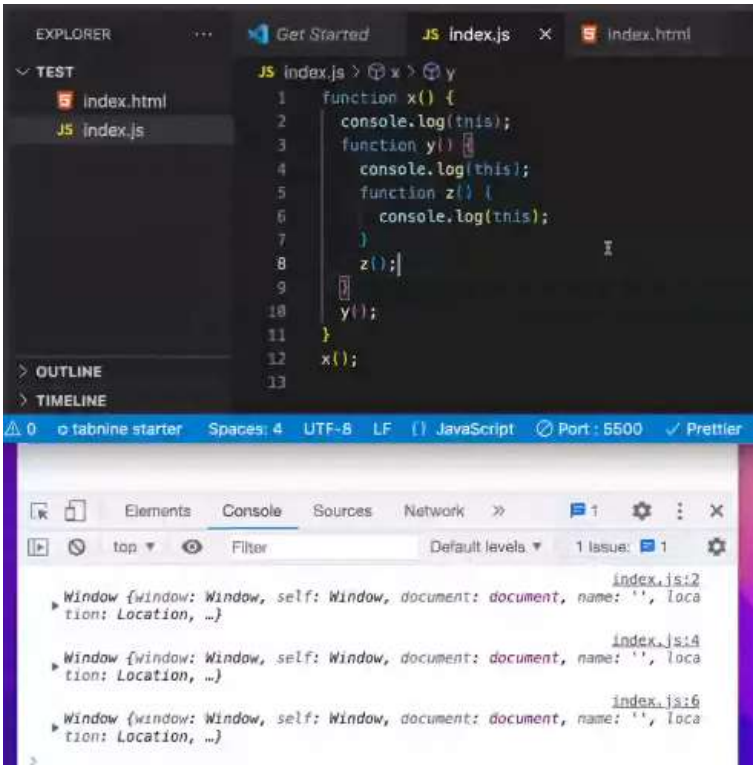


Window is the global object given to us by the browser

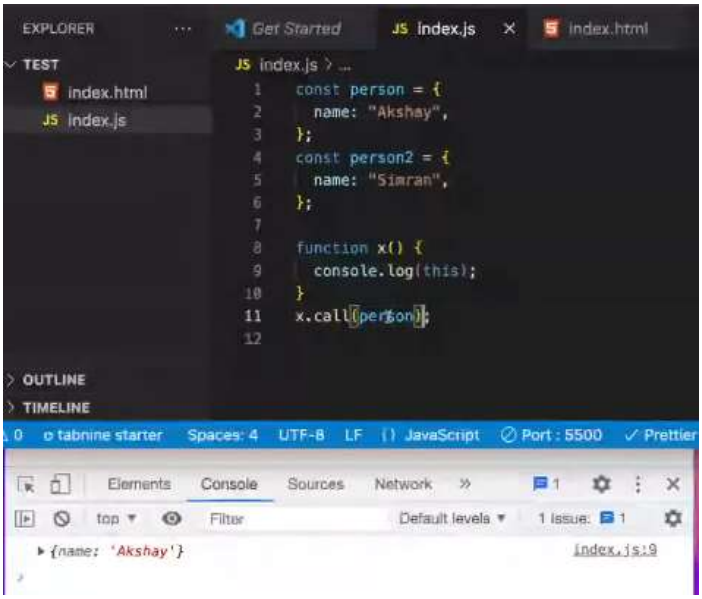
What if we have normal function and another normal function inside it.

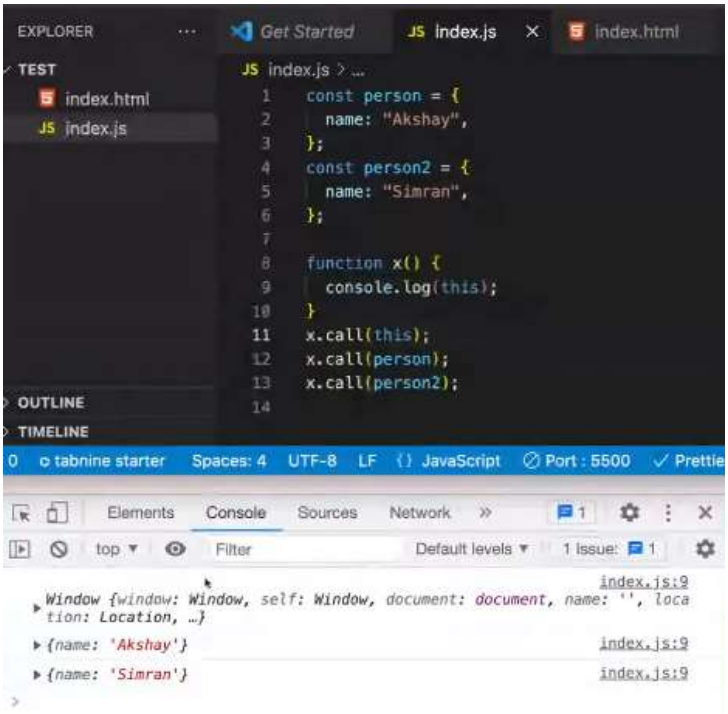


What if we have one more function inside it Z( )

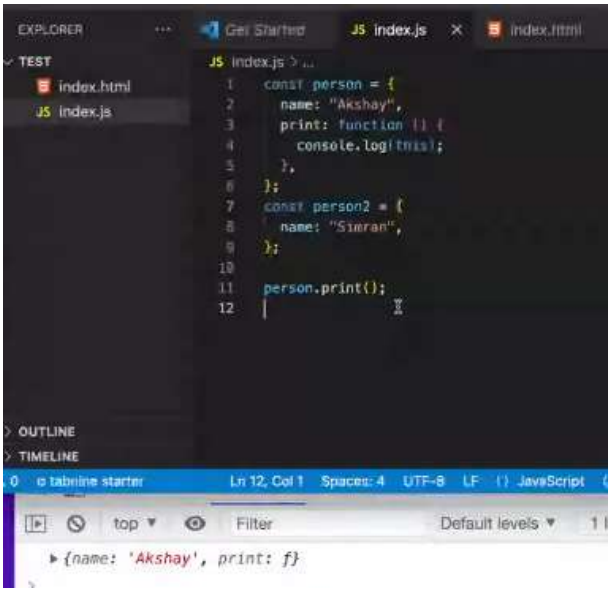


If we make a function and an object.
We call that object using the function. Let's see what happens

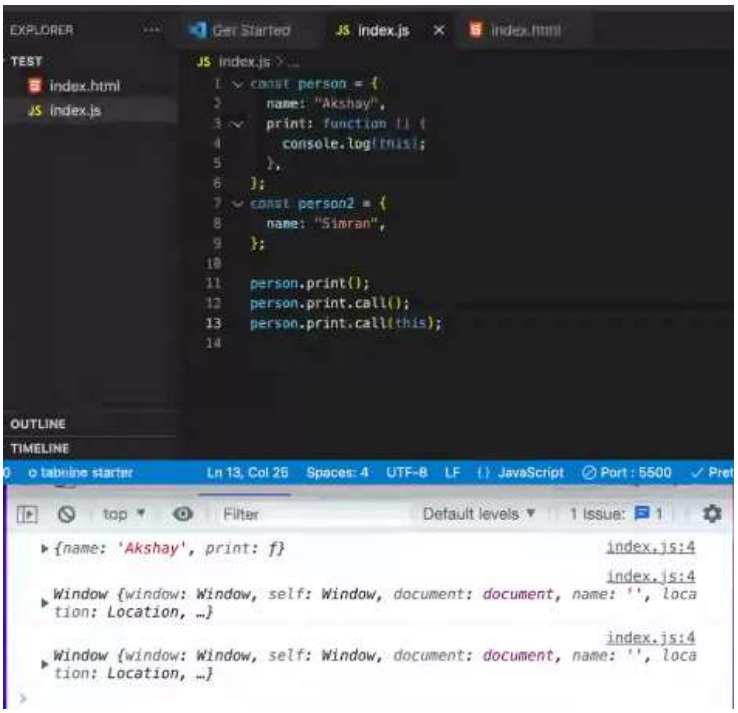




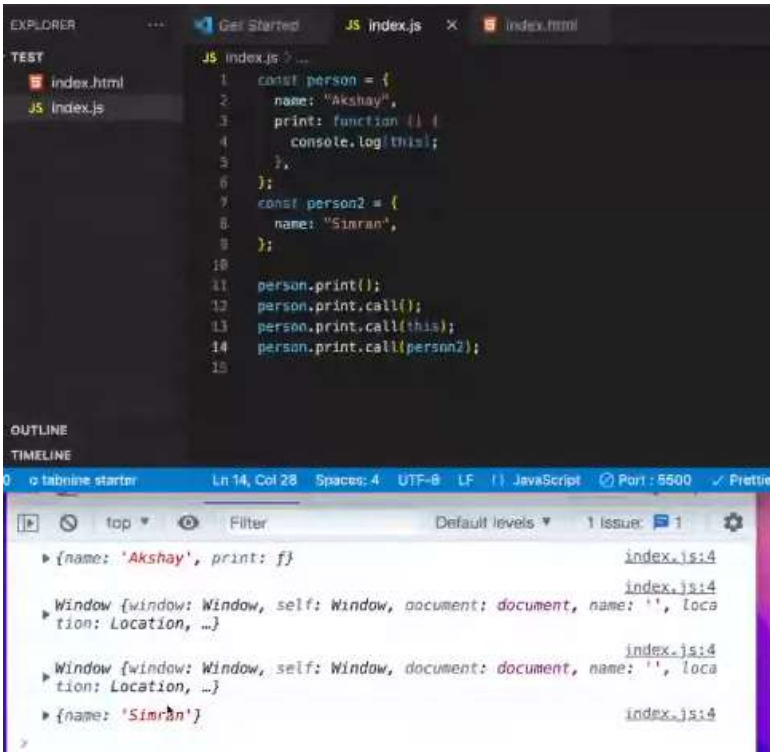
What if we put that function inside object



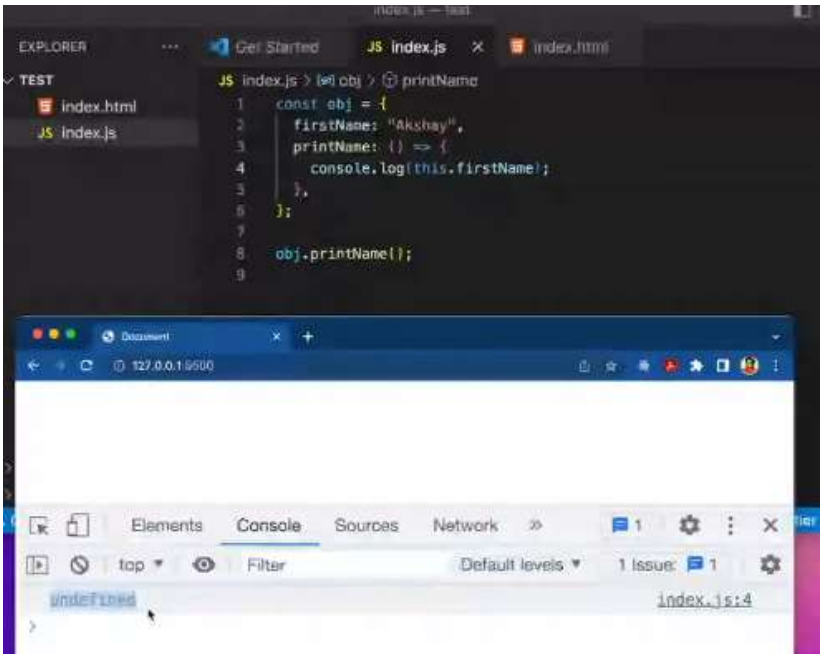
What if we use .call()
call takes window object and then this refers to window object



What if we use .call(object)



This refers to window in arrow function that is why we are getting undefined



Interview Tips

Luck is very important and we cannot control it.
Many Companies don't train their interviewers well
A person can be a good software engineer but bad interviewer

What we can control is:

1. Our preparation
   Technical preparation
   Communication skills: Lot of people fail due to communication skills ,We spend so much time for tech skills but comm skills are equally important.
   Learn to speak while you write, speak your thoughts.
   Practice to speak even when you are coding alone.
   Mock Interviews

2. Talk while you are coding so that interviewer can also see what you are doing.
   If you cannot explain, interviewer thinks you also don't know or you have crammed things.

3. In a company you don't work alone so comm skills should be good. Does not matter how good Software engineer you are, you should be able to communicate your thoughts and ideas to others.

4. Spoken english is very important.

5. Preparation on the interview day (not technical preparation)
   You should not be in panic state before/during interview
   Keep your pen and paper handy
   Keep you water, next to you.
   Keep your laptop charged, keep your charger handy.
   Keep power point near to you.
   These small things mess your interview.
   Keep your camera always open during interview.
   Keep your phone on silent.
   Have a power backup, mini UPS (it does not cost much).

6. Confidence comes from preparation (not just your technical prep) (it includes non tech prep also which are mentioned above).

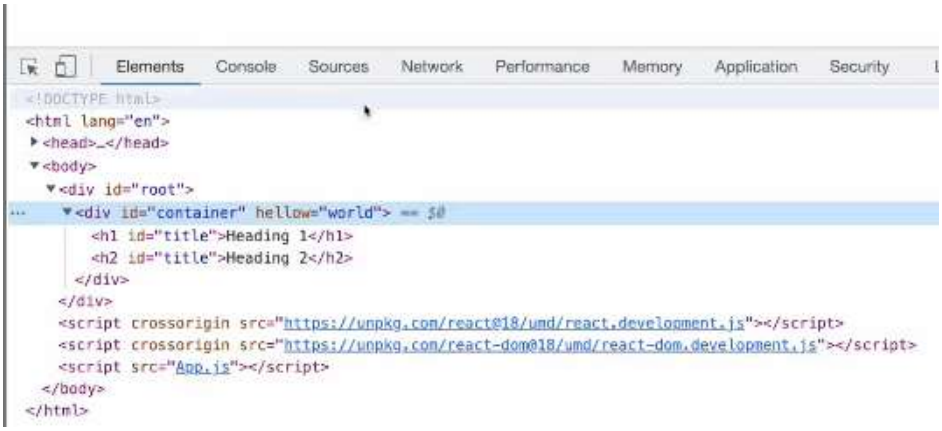
Session 3

How to make our app deployable?
How to launch our app
How to build our own create-react-app

Let say with ID we also pass something which is not an attribute, will it also get passed in container?

```
const heading2 = React.createElement(
  "h2",
  {
    id: "title",
  },
  "Heading 2"
);
const container = React.createElement(
  "div",
  {
    id: "container",
    hellow: "world",
  },
  [heading, heading2]
);
```

Yes, we can put in anything over { } while using createElement



They are not called as attributes
They are called as props just like properties or attributes

Is this a production ready app?
What it needs to be production ready?
Bundle Everything, remove console, server, optimisation, Caching, minify many things

We need to use something known as Bundlers
Same as web-pack, it is a type of bundler
Vite is also a bundler, parcel is also a bundler
We will be using Parcel, in original create-react-app, web-pack is used as Bundler and it also uses Babel.

Module Bundling, on a high level, is a process of integrating together a group of modules in a single file so that multiple modules can be sent to the browser in a single bundle.
When we write our code in a modular pattern:

- We keep our javascript in separate files and folders based on functionality.
- Add the script tag for each file that we are using in correct order of dependency.

For each script tag , browser will send the request to the server which will have bad effect on the performance of our application. In order to overcome this we usually create a single bundled file which will integrate all the other files and that bundled file is sent to browser.
Module bundling can also include a minification step i.e. all the unnecessary characters like space, comma, comments etc. are removed from the file and its minified version is created

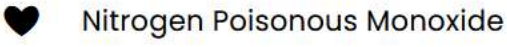
and whenever a request comes from the browser that minified version is sent back. Less data means less browser processing time.

Webpack vs Rollup vs Parcel :
All these differ in following things, although there is very little difference.
They almost do the same job
Configuration, Entry points, Transformations, Tree Shaking, Dev Server , Hot Module Replacement, Code Splitting

All above optimisations are done by bundlers only.

What is Parcel?
Parcel is a package, means a module of JS files
Some piece of code so we need a package manager also
That is why we do "npm init"
What can we do other than "npm"?
We can also use "yarn"

What is npm?
It does not stands for node package manager
If we go to official doc of npm we see



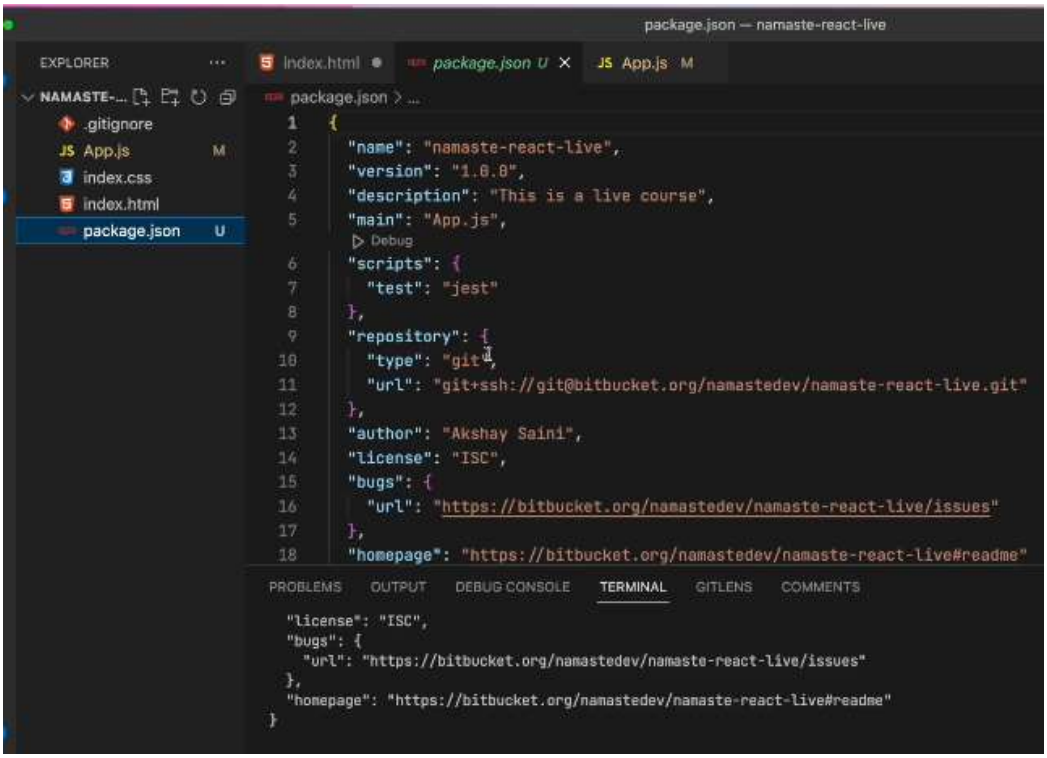
But now-where is written node package manager.
There is a repo of npm where they take any name for npm.
So there is no official name for npm

npm is the world's largest **Software Registry**.
The registry contains over 800,000 **code packages**.

How to install npm?
We write "npm init"
{
"name" : "foo",
"version" : "1.2.3",
"description" : "A package for fooing things",
"main" : "foo.js",
"keywords" : ["foo", "fool", "foolish"],
"test command": jest,
"author" : "John Doe",

"licence" : "ISC"
}

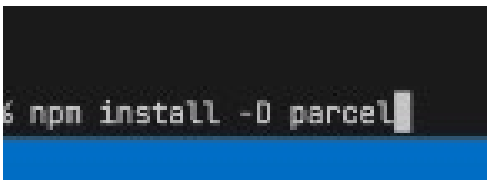
Now we get package.json in our file and folder structure



Why do we use npm?
Because we use so many packages so we need someone to manage these many packages.
Our app does not run only on react, we need certain packages to build it. These packages come inside npm.

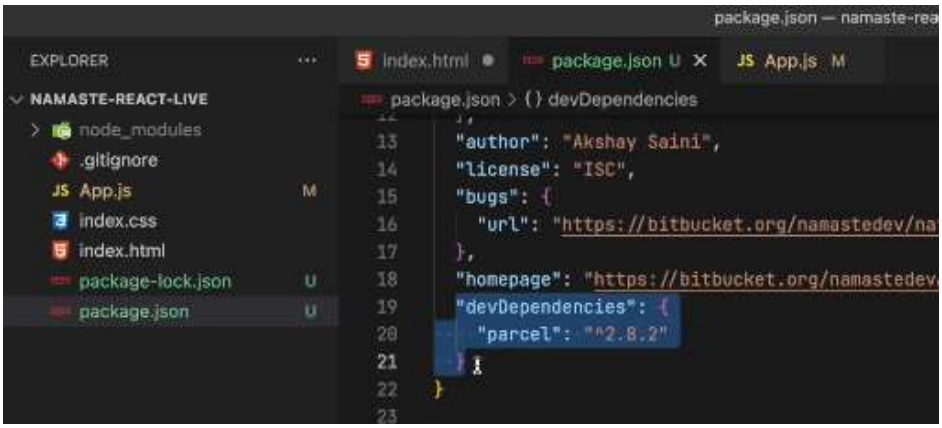
What is inside package.json?
All information we filled.

How to ignite our app?
We use parcel (Zero configuration build tool for everything)
npm install to install a package + package name
We do not want parcel in production we want it for development so we use -D
npm install -D parcel where -D means devDependency



Some people also do --save-dev also instead of -D, it is same

Dependency means all packages that my project needs, my project is dependent on it.
Parcel is one of the dependency which our react project need as bundler.
Now we get package.lock.json and in package.json we have devDependencies {
Parcel : "version" }



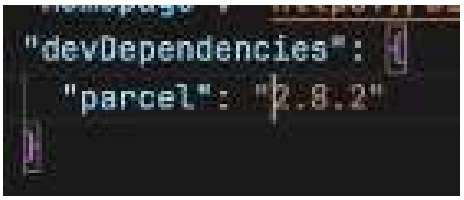
Package-lock.json is an important file.
^ is known as caret

What's the difference between tilde(~) and caret(^) in package.json?

- ~version **"Approximately equivalent to version"**, will update you to all future patch versions, without incrementing the minor version. ~1.2.3 will use releases from 1.2.3 to <1.3.0. Our package will automatically update to new version for major changes.

- ^version **"Compatible with version"**, will update you to all future minor/patch versions, without incrementing the major version. ^1.2.3 will use releases from 1.2.3 to <2.0.0. Our package will automatically update to new version for minor changes.

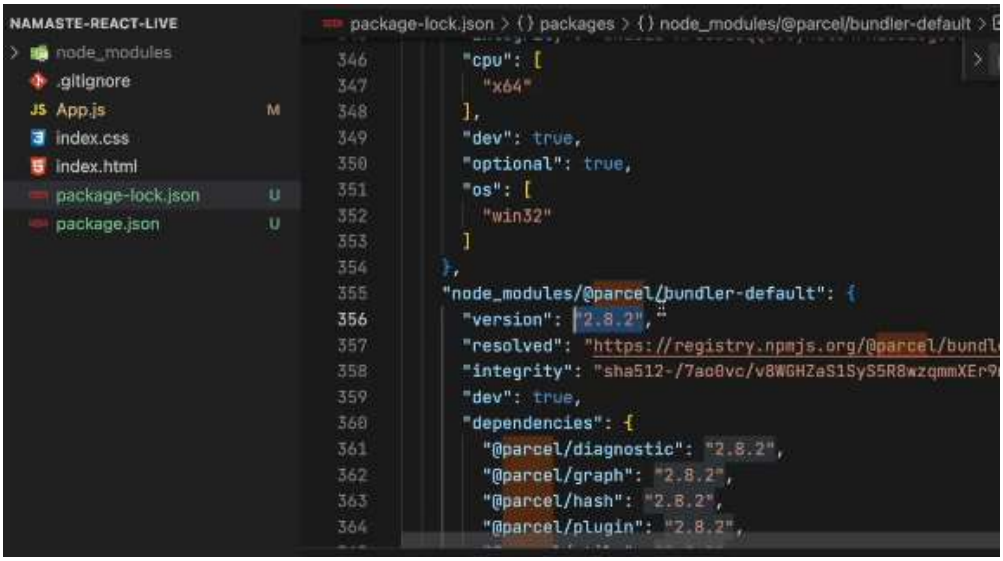
  If we do not use anything means I just want this version only, do not update



What is package-lock.json?

package.json	**package.lock.json**
It records important metadata about the project.	It allows future devs to install the same dependencies in the project. It records the exact version of every installed dependency, including its sub-dependencies and their versions.

We see our package-lock.json and we get exact version of our Parcel.



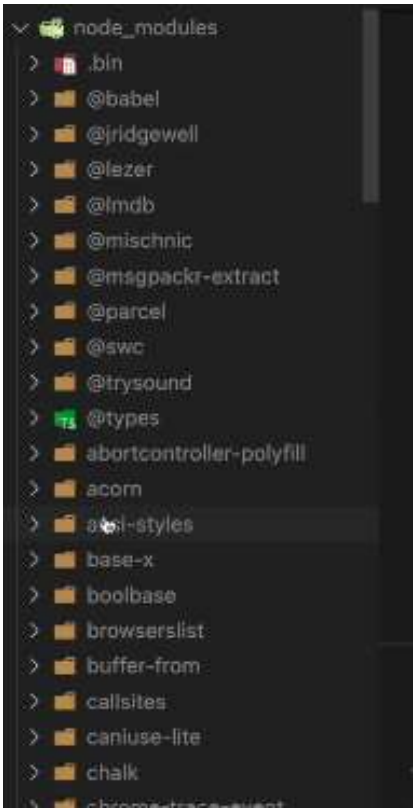
It is working on my local but breaking in production?
Let say we use ^ in package.json
So package updates itself automcatically and it breaks in production so package-lock.json has exact version, it locks the version.
It takes snapshot of exact version we have in our project.

Important things about package-lock

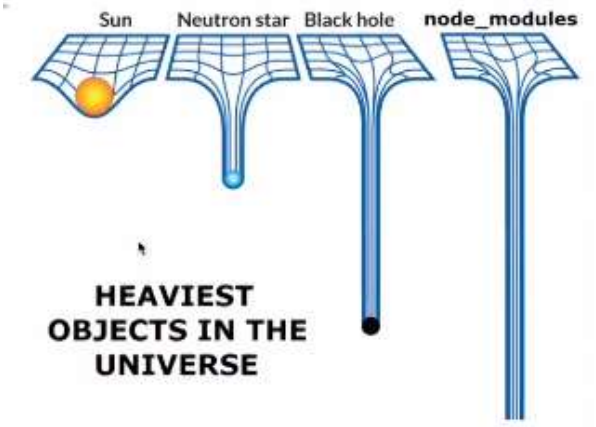
Never put package-lock.json in git ignore
Always put it in your git with your project.
We need to push our package-lock to server and we cannot do it directly so we push it in git and server fetch it from git so we need to push package-lock in git.

It maintains a hash of version of package also, keeps track that hash is same in local and production. It maintains the integrity.

When we installed parcel, node modules also got created.



Whatever we install, it gets installed in node modules.
So node modules is like database of npm.
Parcel is there in node modules.
Parcel uses lot of things to optimise our app.
Parcel also has many dependencies to do this optimisation, they are also inside node modules so our node modules becomes huge.
We have something like browserlist in node modules which help our app work nicely in older version of browsers, sameway there are many packages to optimise our app.



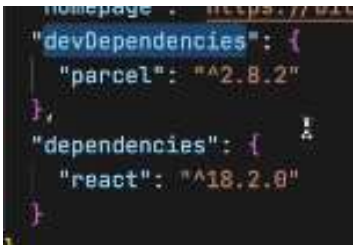
Should we add our node_modules to our git??
It is foolish to put node modules in git repo, it is the heaviest thing in your project. It 1GB large.
Our package-lock.json has sufficient material to make another node modules
We can generate another node modules using package-lock.json so no need to push node_modules.
Just push package-lock.json

We will generate our node modules in server using package-lock.json and it will make sure our app does not break due to node modules not being pushed inside git.

We are using CDN to get react in our project
This is not the good way.
Currently we are using react 18, what if it gets updated to react 19?
CDN is on different server
What if we create our own server and fetch react from it?
That is why we do not use CDN and create our own react app

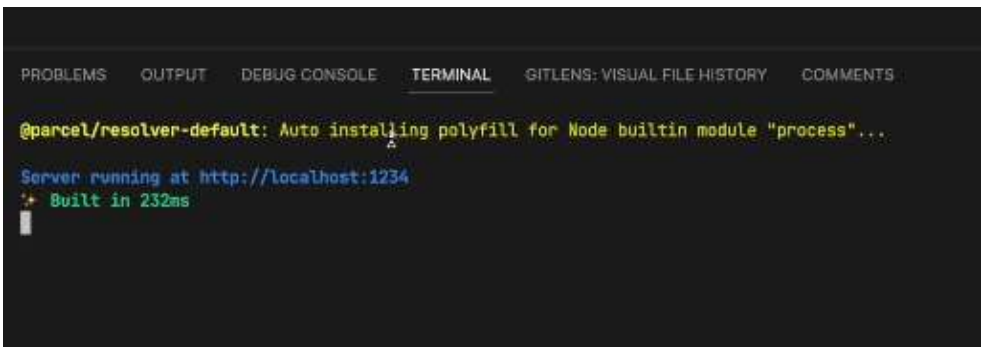
How to install react in our project?

Now we do not use -D as we need react globally and we get react in our dependencies in package.json



This is the right way to install react in our project.
Now we install ReactDOM
"npm i " is same as "npm install"

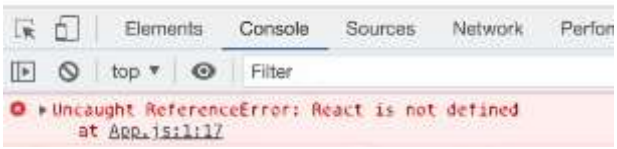
npx means execute using npm
Now we use "npx parcel index.html"
Where index.html is our entry-point
It starts a mini-server for us.



Parcel gave a mini-server to us.

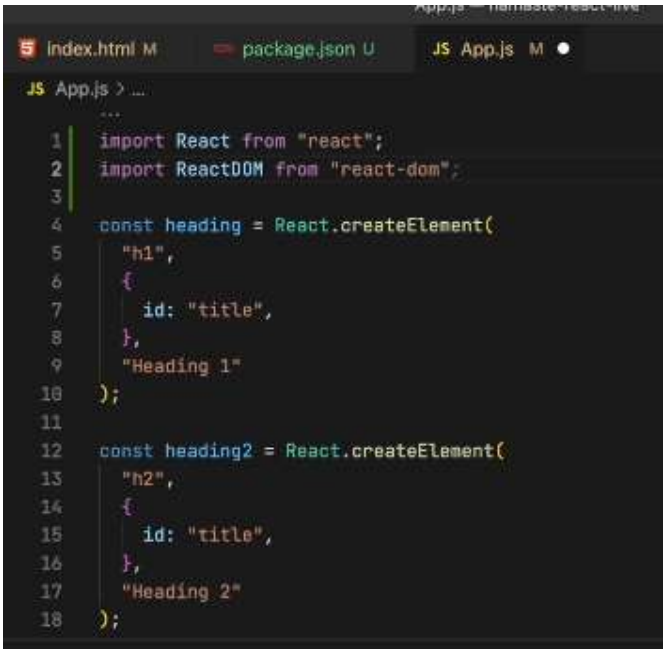
Now our app runs on localHost:1234

Now our console gives this error

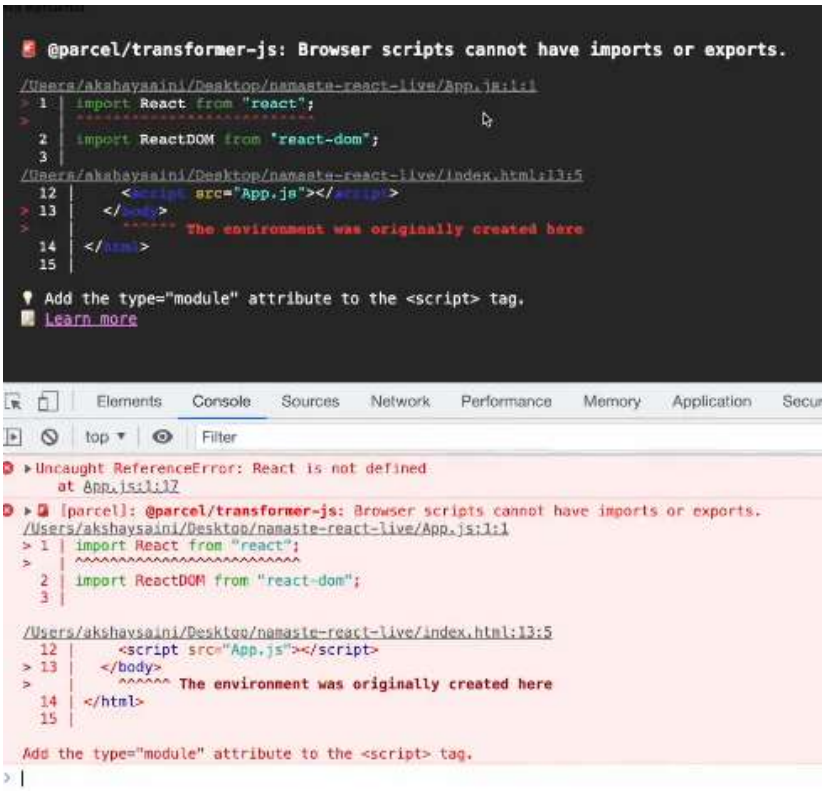


Because we have use React.createElement() etc in our app.js but we have not imported React in our js file

How to import React??
We use a keyword "import"
We did not get this error before because we were using a CDN earlier.
We want to use react and reactDOM from our node modules now so we use "import"



Now our code gives below error



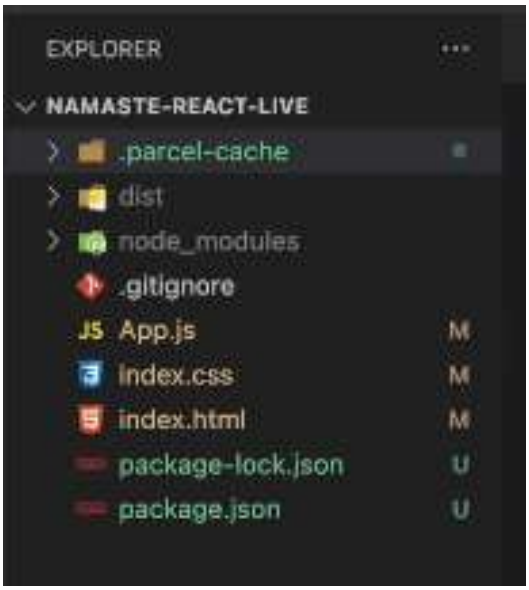
Never touch your node modules
Never update your package.json or node modules.
The error is coming because we have use <script src= "app.js"> in index.html
And we use import in app.js
Browser does not understands import
So we need to tell browser that this is not a normal JS file, it's a module so what we do is
Specify it's a type = "module"

Earlier it was like
import ReactDOM from "react-dom" but now they have updated it to

There is something known as **Hot Module Replacement (HMR)**
Means keep on updating the page on saving any changes
Parcel does this HMR, and saves all the changes in our HTML,CSS or JS.

How does Parcel does HMR?
There is something known as **file watcher algorithm** and parcel uses this algo which is written in **C++ internally**.

What is parcel_cache and dist in file and folder structure?

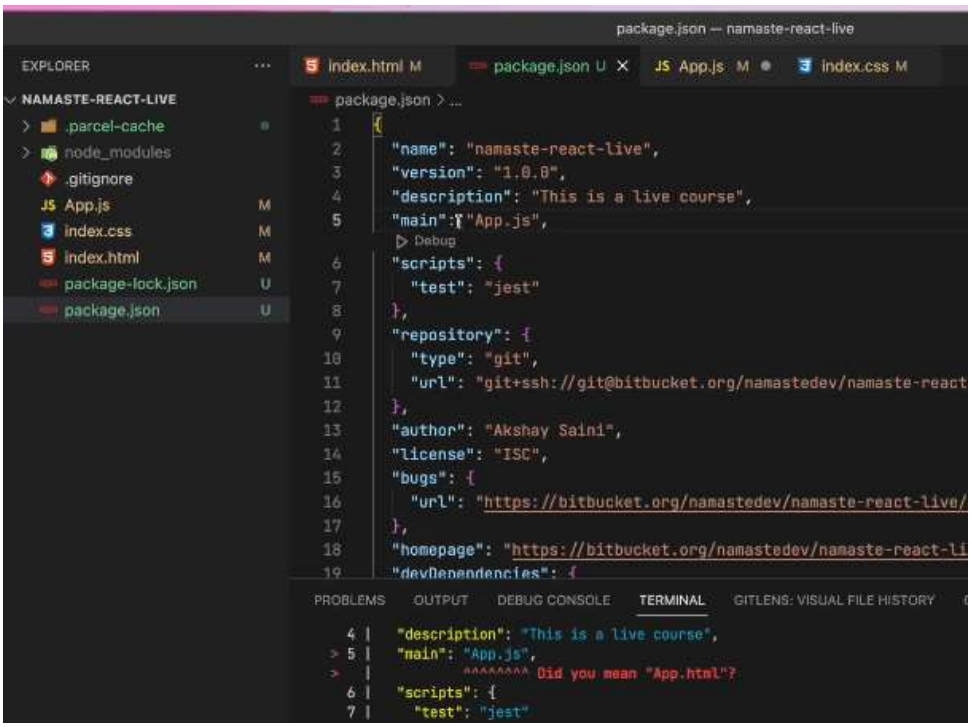


Parcel need some space to do this HMR and other things
So it creates parcel_cache and it contains all files which does all optimisation things for us in parcel
dist folder keeps the file minified for us
npx parcel index.html creates a development build and host it on our server.

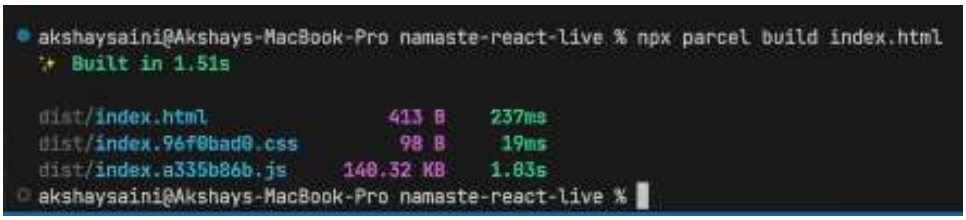
How to tell parcel to make a production build?
We use "build" command and it minifies all our files, get it ready for production and push them inside dist folder.

```
npx parcel build index.html
```

We use "main: App.js" in our package.json which tells entry point of our app as App.js
Which is not needed if we are using parcel so we delete this command

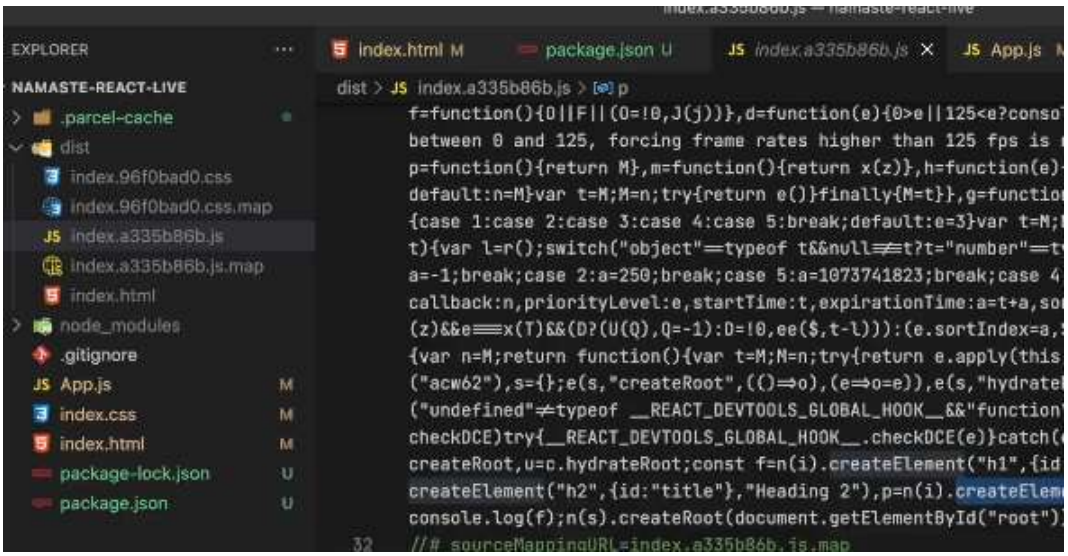


We give entry point while installing parcel only
npx parcel build index.html
And it creates 3 files which are below:



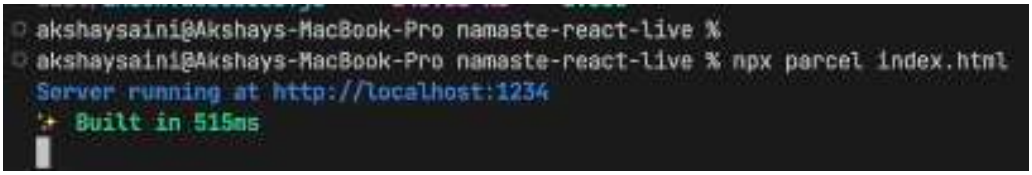
These 3 files are bundle of our app which will go to production build and will go to dist folder.

These 3 files has our code for HTML, CSS, JS



Parcel minified everything for us.
It bundled everything for us.
Removed all console logs and cleaned the code

What takes lot of time to load in browser?
No, HTML
No, CSS
No, JS
node modules is in server so it is not the answer.
Images, media takes most time to load in browser
Parcel manages dev and production build and is superfast
Parcel does image optimisation also, it minifies the images also if they are in our project.
Parcel also does **Caching while development for u**s, What is caching?
We see our build took 515ms initially for dev build.

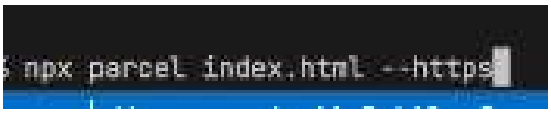


Now if we refresh our page we see time is 5ms, 4ms, 9ms
The time is reducing means parcel is using something known as caching.

Parcel also compresses our files, renames our variables. This process also called compression.

Parcel checks your project's compatiblity with older version of browsers.

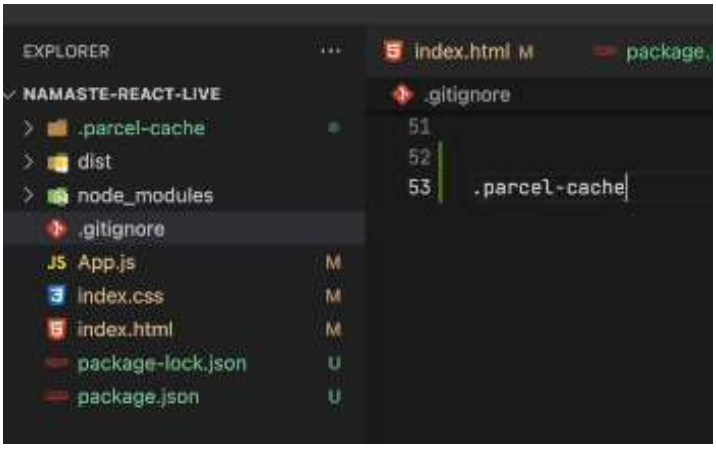
Sometimes we need our build to test in https also. Parcel gives us that functionality to enable https in our localserver



It took lot of build time this time as we are doing this change for first time, if we do it again
It takes lesser time due to caching

If we run 2 project at a time, Port number of server will also change automatically, it is also managed by Parcel.

Should we push parcel-cache in git?
We should not push our parcel-cache in git. And we should put it in our gitignore



Because anything which can be auto-generated on server will be put inside git-ignore. We do not push it to git.
So we do not push parcel-cache and dist in git as we can generate them later

Parcel also uses something known as consistent hashing algorithms to cache things up and bundling up.

Parcel is a zero-config bundler.

If React is Narender Modi then Parcel is Amit shah

If BJP wins, its not only because of Namo and amit shah
There are some other ministers which are more or less important.
React (modi) need Parcel (amit shah) and parcel need many dependencies (other ministers) .

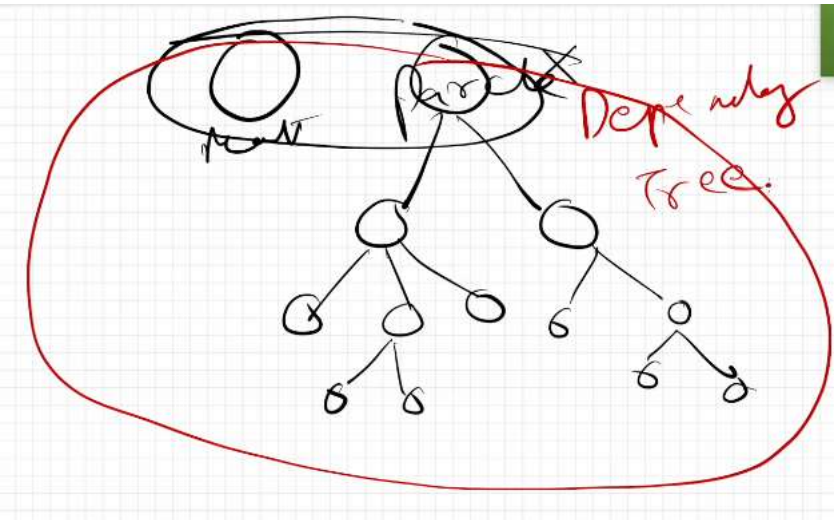
What are benefits of using Parcel?

```
*
* HMR - Hot Module Reloading
* File Watcher algorithm - C++
* BUNDLING
* MINIFY
* Cleaning our Code
* Dev abd Production Build
* Super Fast build algorithm
* Image Optimization
* Caching while development
* Compression
* Compatble with older version of browser
* HTTPS on dev
* port Number
* Consistent Hashing Algorithm
* Zero Config
*
*/
```

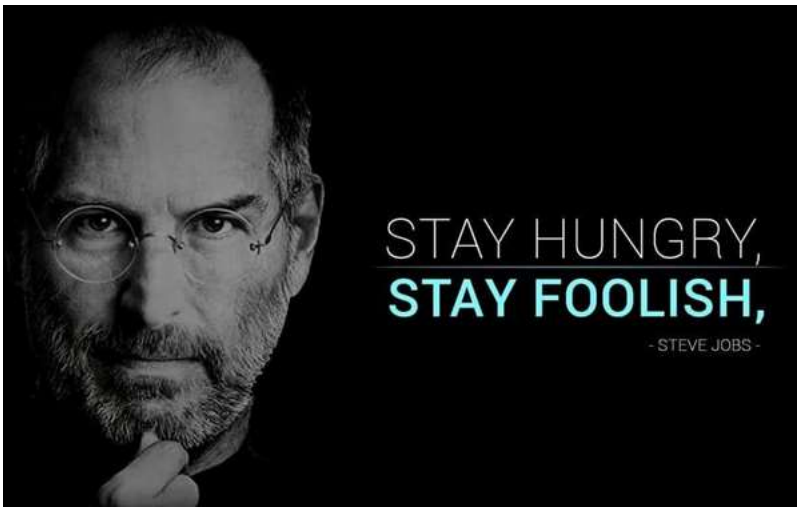
Parcel uses many dependencies in itself and all dependencies can have there dependencies so this is inifinite loop and this process is called Transitive Dependencies (dependencies ki dependencies)

```
},
"node_modules/@parcel/bundler-default": {
  "version": "2.8.2",
  "resolved": "https://registry.npmjs.org/@parcel/bundle
  "integrity": "sha512-/7aoBvc/v8W6HZaS1SyS5R8wzqmmXEr9
  "dev": true,
  "dependencies": {
    "@parcel/diagnostic": "2.8.2",
    "@parcel/graph": "2.8.2",
    "@parcel/hash": "2.8.2",
    "@parcel/plugin": "2.8.2",
    "@parcel/utils": "2.8.2",
    "nullthrows": "^1.1.1"
  },
  "engines": {
    "node": ">= 12.0.0",
    "parcel": "^2.8.2"
  },
  "funding": {
    "type": "opencollective",
    "url": "https://opencollective.com/parcel"
```

This structure of Transitive Dependencies is called **Dependency Tree**



Read the docs (Be Curious)

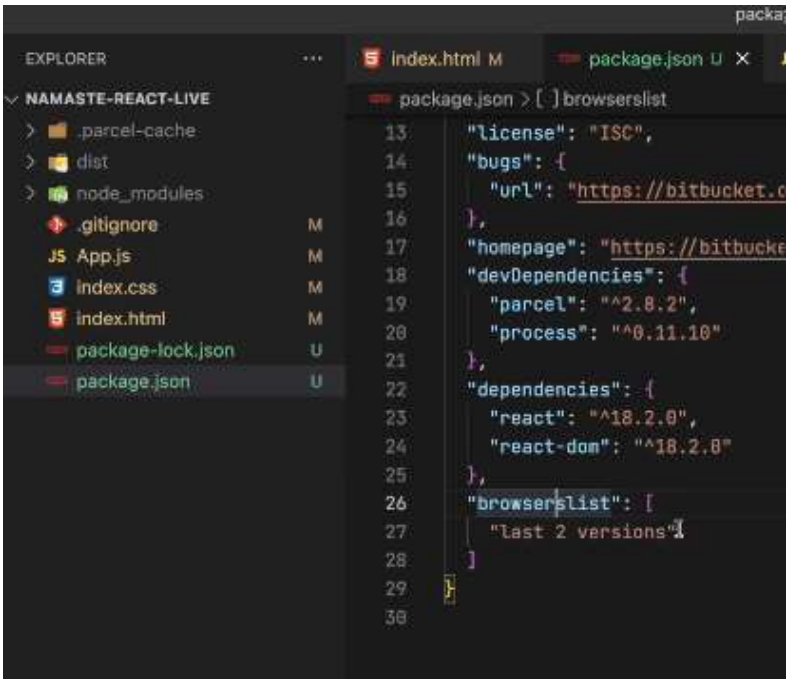


This is our create react app without using create-react-app

How to make your app, compatible with older version of browsers?
We use **browserList** which is already used by parcel.

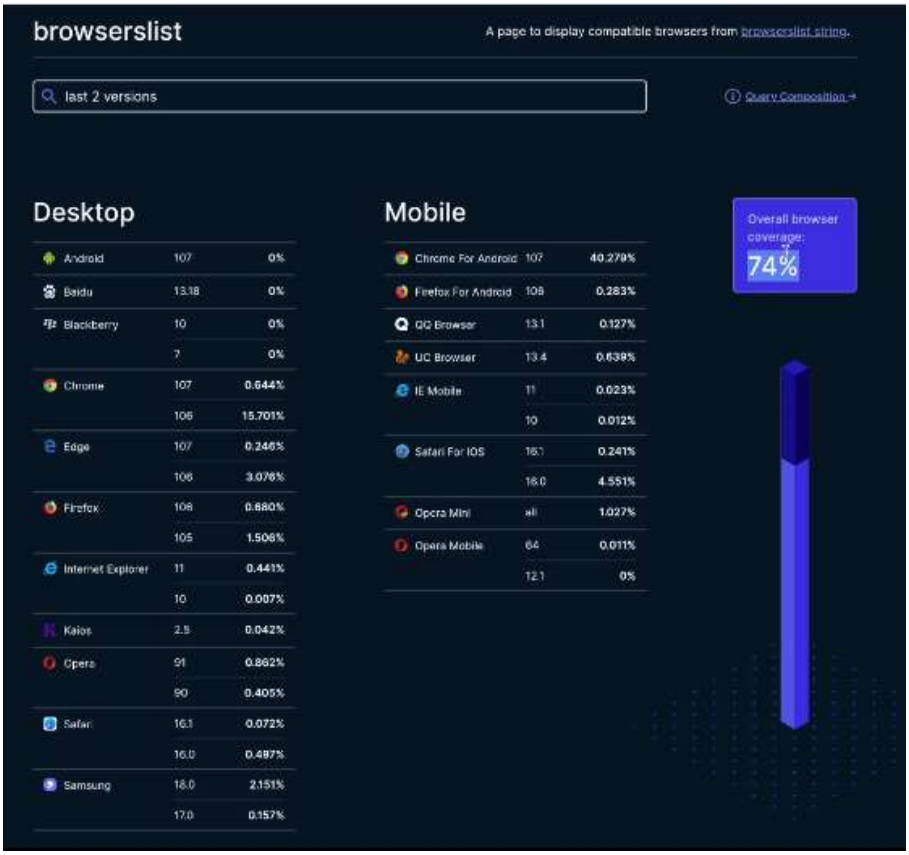
```
"browserslist": [
  "defaults and supports es6-module",
  "maintained node versions"
]
```

We put this code in our package.json
browserList is feeded with a Array which has some configurations.
If we write "last 2 versions" our app will run in last 2 versions of all browsers available



If we want support for only last 2 chrome versions, we write

We can use browserList.dev website to see



Full List

You can specify the browser and Node.js versions by queries (case insensitive):

- `defaults` : Browserslist's default browsers (`> 0.5%, last 2 versions, Firefox ESR, not dead`).
- By usage statistics:
  - `> 5%` : browsers versions selected by global usage statistics. `>=` , `<` and `<=` work too.
  - `> 5% in US` : uses USA usage statistics. It accepts two-letter country code.
  - `> 5% in alt-AS` : uses Asia region usage statistics. List of all region codes can be found at `caniuse-lite/data/regions` .
  - `> 5% in my stats` : uses custom usage data.
  - `> 5% in browserslist-config-mycompany stats` : uses custom usage data from `browserslist-config-mycompany/browserslist-stats.json` .
  - `cover 99.5%` : most popular browsers that provide coverage.
  - `cover 99.5% in US` : same as above, with two-letter country code.
  - `cover 99.5% in my stats` : uses custom usage data.
- Last versions:
  - `last 2 versions` : the last 2 versions for *each* browser.
  - `last 2 Chrome versions` : the last 2 versions of Chrome browser.
  - `last 2 major versions` or `last 2 iOS major versions` : all minor/patch releases of last 2 major versions.
- `dead` : browsers without official support or updates for 24 months. Right now it is `IE 11` , `IE_Mob 11` , `BlackBerry 10` , `BlackBerry 7` , `Samsung 4` , `OperaMobile 12.1` and all versions of `Baidu` .
- Node.js versions:
  - `node 10` and `node 10.4` : selects latest Node.js `10.x.x` or `10.4.x` release.
  - `last 2 node versions` : select 2 latest Node.js releases.
  - `last 2 node major versions` : select 2 latest major-version Node.js releases.

Different Type of script tags in HTML

The **HTML <script> type Attribute** is used to specify the MIME **(MIME (Multipurpose Internet Mail Extensions) is an extension of the original Simple Mail Transport Protocol (SMTP) email protocol. It lets users exchange different kinds of data files, including audio, video, images and application programs, over email.)** type of script and identify the content of the Tag. It has a Default value which is "**text/javascript**".

Syntax:
<script type="media_type">
Attribute Values: It contains a single value i.e media_type which specifies the MIME type of script.
Common “media_type” values are:

- text/javascript (this is default)
- text/ecmascript
- application/ecmascript
- application/javascript