Parcel has a different build for dev and prod bundles.

To create a prod build we write: npx parcel build index.html – this is however will throw out an error. We have a main property in the package.json which we don’t need when using parcel. This is the way to tell npm that it is the entry point of our app. So index.html is the entry point for parcel but the main conflicts with this and it throws out an error. So basically, you must remove this main to avoid this error.

dist folder: when we do npx parcel build index.html it will bundle, it will minify, it will do all the crazy stuff and put all those files inside the folder called dist folder. Until now we were doing development related builds so the dist folder contained everything related to development.

Three main files that we should care: index.html - it will compress the index.html which will be in one line. Index.css and index.jss. These three files will contain all the code that we will write. These three files are the production ready code of our project.

It will also compress the js file and will change the name of the variable for you.

Parcel cache and dist folder can be regenerated even if you delete it. It’s not our core code.

So even if you build a dev build it will automatically generate dist and .parcel-cache folder. So, anything that is automatically generated is not to be put into GitHub. So, to ignore those files we put those files in .gitignore. So basically, dist and .parcel-cache will both go into .gitignore.

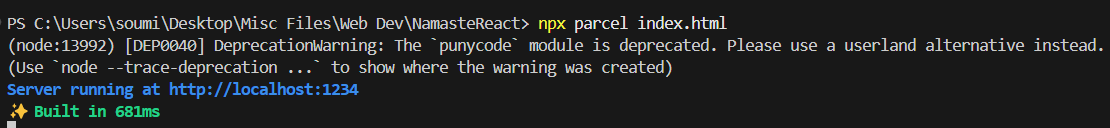
To make the app compatible with older browsers we need to use browsersList. There is already an existing browerselist in node\_modules. Browserslist is a npm package. It needs some configuration. <https://browserslist.dev/?q=bGFzdCAyIHZlcnNpb25z> this website is nice to check out browserslist. So, we must specify what all browsers our app should support. To configure it we need to make use of the package.json file. We have to add browserslist in our package.json file. Browserslist will take an array of browsers. So, if I want to support our app to the last two chrome versions we will write “last 2 Chrome version” in the array with double quotes both sides of the string. So, when we specify this this means that it will definitely will work on the last 2 chrome versions but it can also work in other browsers such as firefox, edge, brave, etc, but it definitely will work on the last 2 chrome versions.

A screen shot of a computer

Description automatically generated

**Laying the Foundation**:

To run the project, we use the command: npx parcel index.html



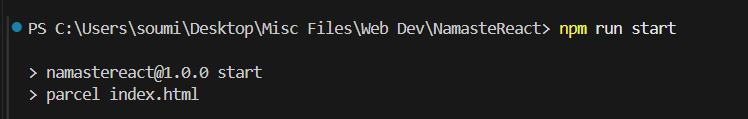
Something like this in the above screenshot.

* It will create a development build and host it on localhost:1234
* npx means executing a npm package
* This command means that I am executing a npm package in this case (parcel) and the source file is index.html
* It’s easier to build a script that automatically runs this command rather than repeatedly running this command.

**How to create a script?**

* It’s a script that needs to be created in package.json.
* We can build a variety of different scripts. For example, a script to start the project in dev mode. We can even create a script to start the project in a production ready app.
* In the scripts property of the package.json file we write:
  + “start”: “parcel index.html”
* We write the command in scripts which needs to be executed by npm.
  + The command to automatically build a script is:
  + “build”: “parcel build index.html”
* These commands will remove the need to write npx parcel index.html.
* This is the industry standard way of automating our way. If in a company we were to be lost we go to the scripts property in the package.json

**How do we run these scripts?**

* We write: npm run start
* 
* This command will execute the same thing as we were doing with npx parcel index.html
* If I were to build my project I would write this command instead: npm run build.
* A screen shot of a computer program

  Description automatically generated
* There’s a shortcut to npm run start and that is npm start without the run command.
* But npm build will not work. It only works for npm start but not for npm build.
* start is a keywork reserved by npm.

**What are React Elements?**

* React elements are like DOM elements.
* React elements are not HTML elements.
* React.createElement creates an object. React elements are objects.
* When rendering them to the DOM it becomes an HTML tag.
* React.createElement => Object => HTMLElement(render)
* To render inside the root element we write:
  + const root = ReactDOM.createRoot(document.getElementById(“root”));
  + To display onto the browser we write ReactDOM.createRoot.
  + Then we write root.render(heading); -> It basically takes the object (React.createElement) and converts it into an HTML.
  + Basically, it will replace the already existing HTML to what we are trying to render from the react.render method.
  + JSX => Babel transpiles it to React.createElement => ReactElement-JS Object => HTML Element (render)

**What is a React Component?**

* In react everything is a component. A search bar, a button, an icon, an input form, a dialog box, etc. all of them are components.
* Two types of components in react: Class Based components and Functional components
* Class based component is the old way of writing code. Functional component is the new way of writing code.
* Class based components use JS Classes and functional components use JS functions.

**What is a React Functional component**?

* It’s just a normal JavaScript function.
* Each functional component always starts with a capital letter. So, for example: function Factorial(){…} or const HeadingComponent = () => {…}
* In react a functional component is a component which returns some JSX element.

Some code explanation:

const fn = () => {

return true;

};

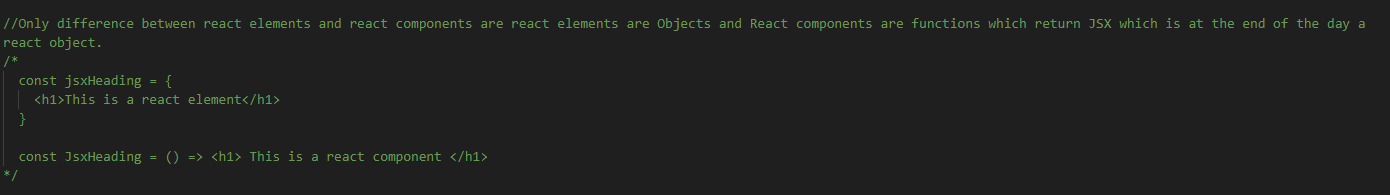
Or

const fn1 = () => true;

Are one in the same thing.

* A screen shot of a computer program

  Description automatically generated
* These things are the same as well. Just the way of writing is different.
* JSX can be nested as well.
* A black screen with white text

  Description automatically generated
* An example of nested elements. A h1 element inside the div which is the parent.
* 

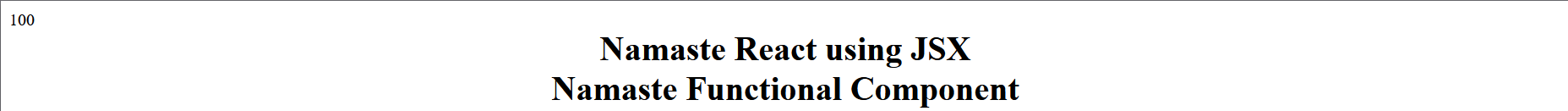
**How to render a component**?

* You can’t render a functional component using the root method of react. But you can with a react element.
* To render a component using the root method we have to wrap it like this <ReactFunctionalComponent /> inside the root method.
* 
* This is the syntax babel understands
* To render a component inside another component we use the same angle brackets that we use when we use to render the component.
* So for example: A computer screen shot of a program code

  Description automatically generated
* Above is the demonstration of how to nest one component inside another.
* Just for reference this is the title component: A screenshot of a computer program

  Description automatically generated
* Component composition is a component inside another component. Basically the nesting of components is called component composition.
* Inside JSX if you put curly braces then you can write any piece of JavaScript.
* A computer screen with text and symbols

  Description automatically generated
* So any JS can go inside that curly brace.
* A screen shot of a computer program

  Description automatically generated
* So, for example I took number variable and put it inside the curly brace in the JSX. Now the output would look something like this: 
* We can even mix and match that. We can put an h2 tag around the curly braces <h2> {number} </h2> or a button around it <button> {number} </button>.
* It can literally execute any piece of javascript code inside that curly brace.

**How to put React Elements inside Components?**

* Basically, a react element is an object and to put an object inside a component just use the curly brace syntax where any JavaScript code can go. Since an element is just an object, we are storing it inside a variable and to display that variable just write the name of it inside the curly brace and you will have it.
* Similarly, we can use react elements inside another react element.
* We can even use functional components inside react elements.
* Cross site scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XWS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.
* Luckily, JSX handles injection attacks. Any JavaScript code inside the curly braces prevents cross side scripting for us.
* Including a component in either of these ways are same: <title /> or <title> </title>.
* We can include components multiple times as we want. It’s a reusable piece of code since it’s at the end of the day is a javascript function which returns some JSX element or some piece of JSX. And functions are reusable blocks of code.
* You can even ***CALL*** the functional component using the curly brace syntax.