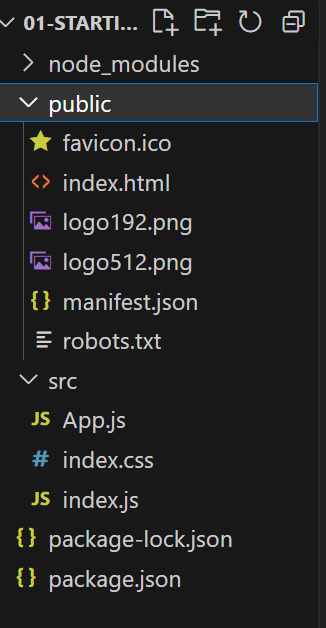
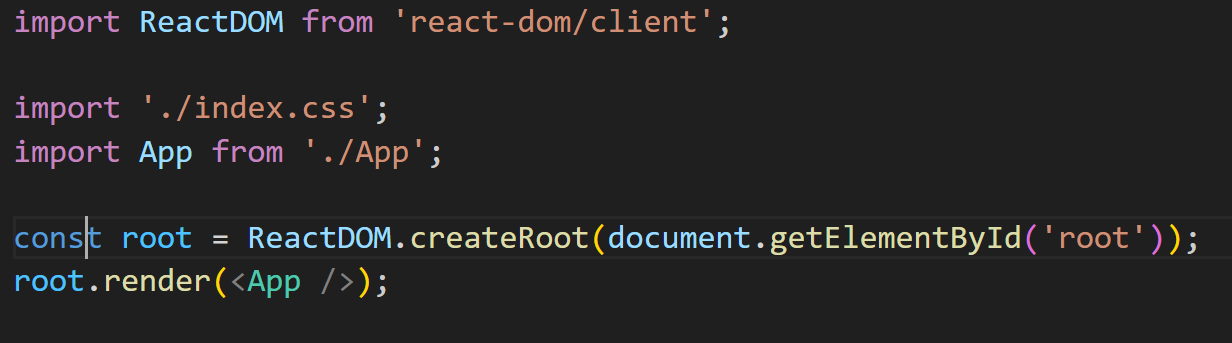
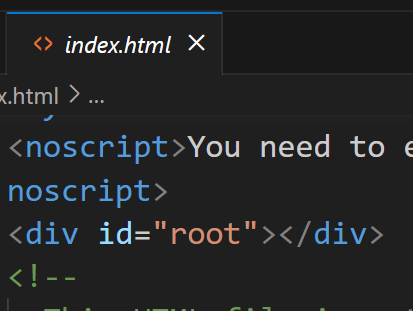
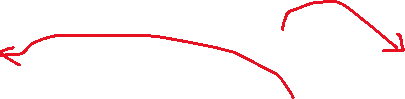
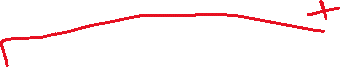
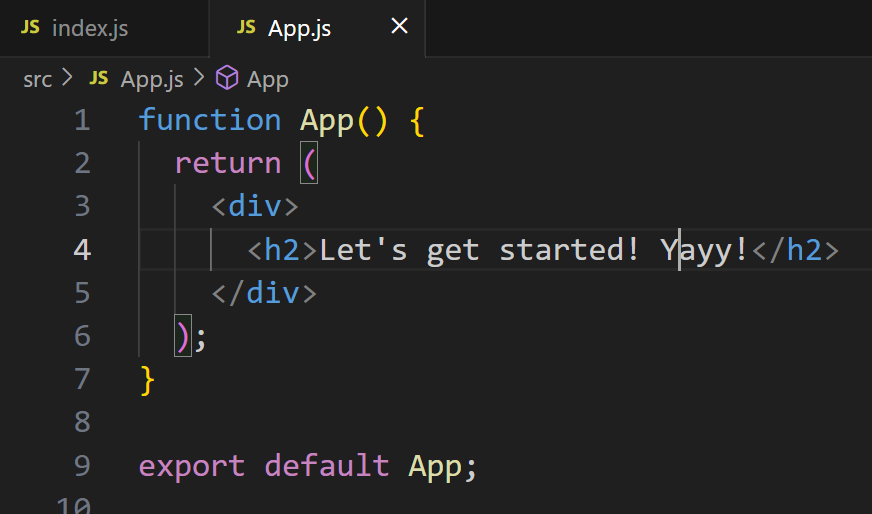
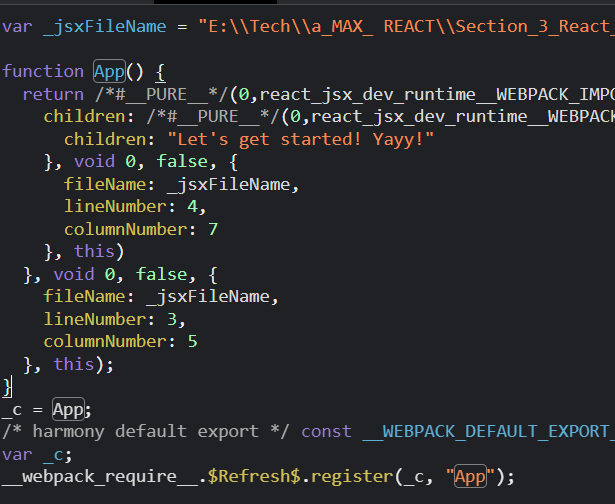
Analysing CRA folders for React Flow

1. NPM start and build processes change the react code to simple JS code which is compatible with most browsers.  
   eg 🡪We can not usually import css into JS, but we do so in react.   
   import './index.css';
2. root.render(<App/>) 🡨 this is also not normal JS, happens only in react.
3. INDEX.JS IS THE FIRST FILE GOING TO BE EXECUTED in a cra project. Or main.js in a vite project.
4. Whenever we import something from reactDOM, we use that feature in our file.
5. root.render(<App/>)   
   the root element tells us where the single HTML page application that will be rendered by the react app.
6. Index.html is the entry point for react driven user interface is executed. Basically it is containing that root element div where we inject all of our user interface built in react.   
   index.js :
7. 

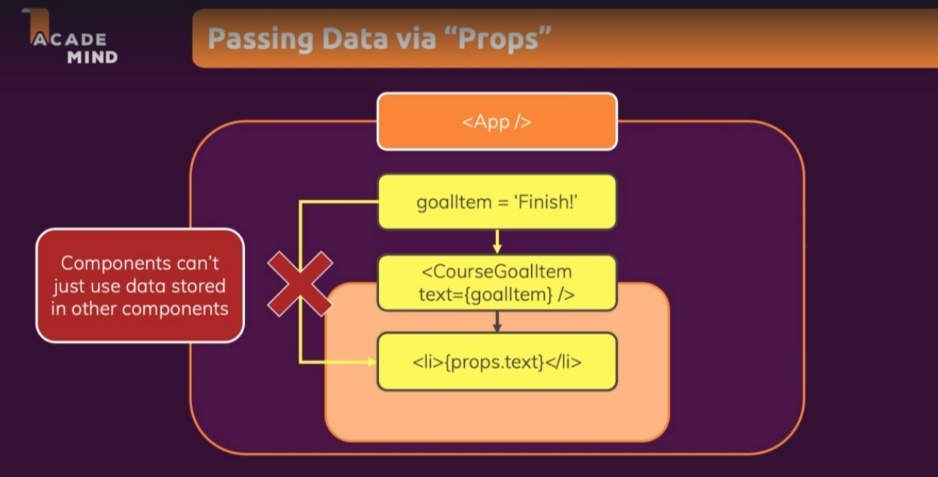


1. We can call the render method on this createRoot element which is the entry point in html file.
2. Omit .js in js component files.
3. App 🡪 is a component which is a function containing HTML code that will be rendered in place of the div in index.html file. This allows us to remove all that document.createElement(p) normal JS code to insert HTML at places of page. This becomes very cumbersome as an imperative approach.  
   But we use declarative approach to build a component which is the end output not the line by line customisation
4. HTML code inside of a JS file is react. JSX is a special syntax introduced by react team.
5. This is what the app.js code looks like when it is rendered in the browser:
6. 



1. One file per component, to avoid confusion. We build a component tree and structurize our project.
2. Other components would be used in the root component ie App component here. And only once, the root component is rendered and nothing else. Everything else will be rendered in the root component file.
3. Component naming convention – start with caps, and sub-word is capitalised.
4. Custom made components are new html tags, which MUST START with capital letters.
5. We can only return one HTML element in a jsx component.
6. We do not write *class* in jsx html, we write *className* for the attributes.

**36 Props and parameters – to make components reusable with different data.**

1. Components can not use data stored in other components.  
   
2. *Just as HTML elements can have attributes, our components can have props or properties which are similar to attributes of an HTML. These* ***props*** *can share the data stored in them as the property of component.*
3. To make the data configurable, we use **props**, and avoids hard coding.
4. The properties or props mentioned in the root component, are sent to the actual component HTML, through a prop object usually called as parameter object. **The parameter object is a JS object with key value pairs. The key spelling should match each time.**

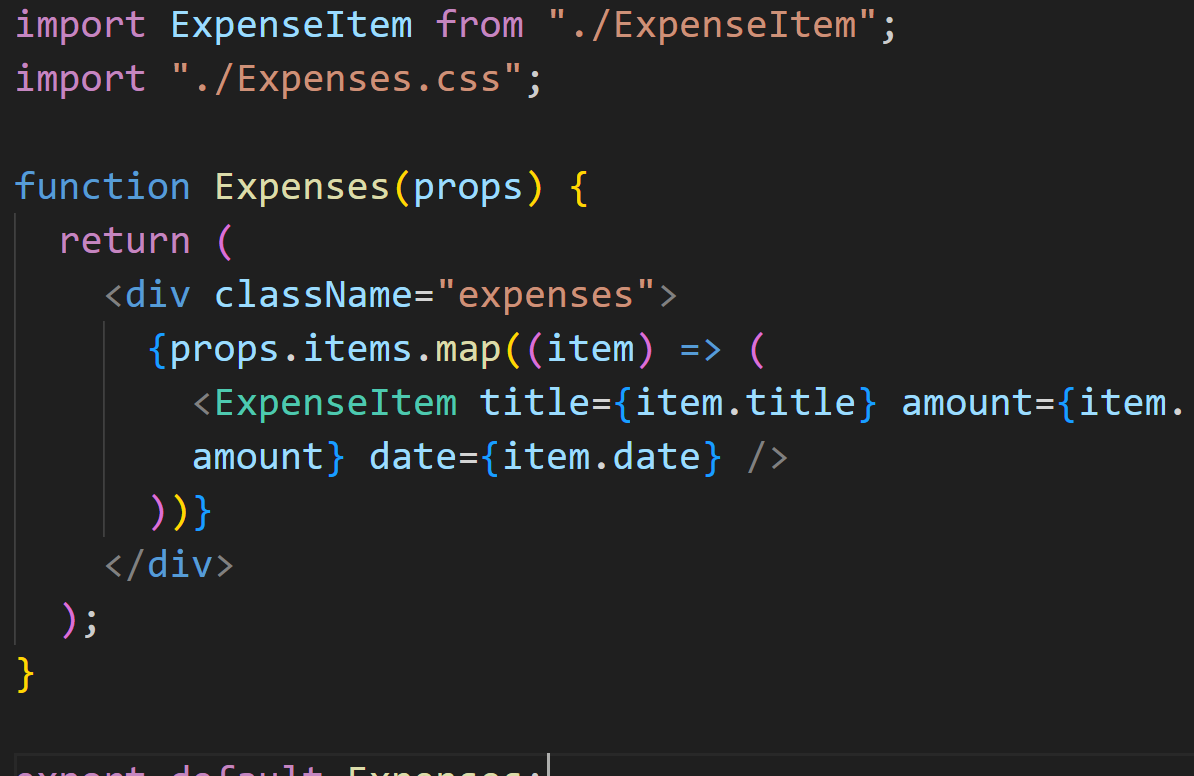
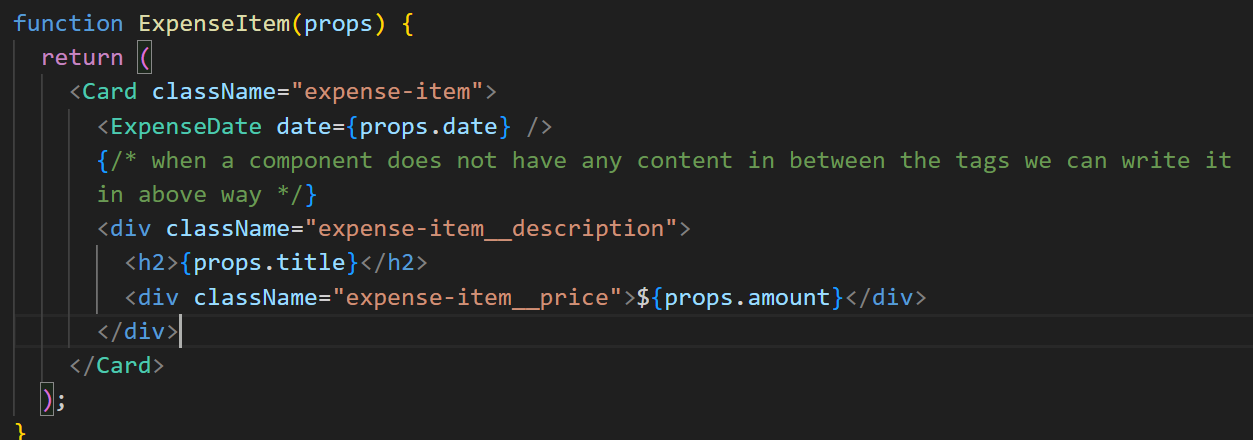
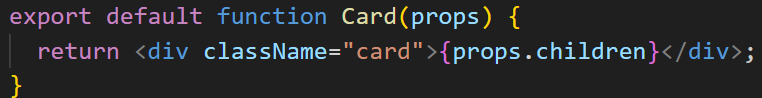
**L-37 adding normal JS logic in react**

1. We add it in the function of component, separately, to keep our JSX code lean.

**L-38 Splitting the Component**

1. To have more separation of concerns.
2. <ExpenseDate />
3. {/\* when a component does not have any content in between the tags we can write it in above way \*/}
4. We can have nested components which pass data from one component to a direct child component, and we can not skip such components, because the data needs to follow the path.

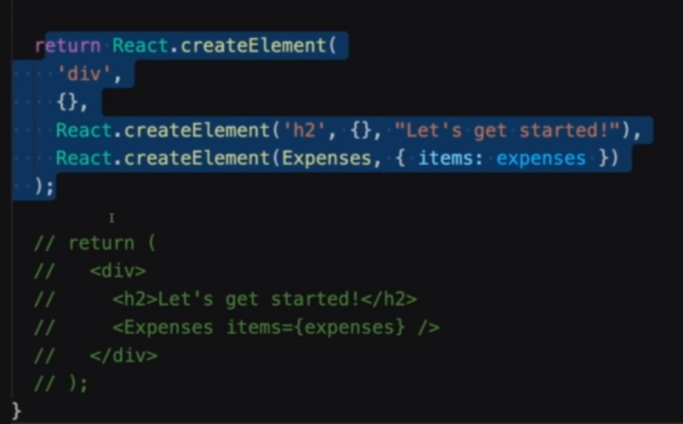
**L-39 Child components**

1. MAP FUNCTION:  
    map takes a callback function as a first argument and the callback function called for each element of array, callback function give you element on first parameter in our example is obj  
     
   props is an object containing properties passed from a parent to a child component. So props.items is the property named items which is an array.   
   props.item.map() maps the items arrary to an array of li-s.
2. 
3. **Note:**We can not have content in between of the custom built components.   
   Eg: 
4. Whatever is written between the tag of custom components is not displayed.
5. To do so, we need to **add {[props.children](https://youtu.be/Sq0FoUPxj_c)}** in the custom component file where they are initialised.  
   
6. Children is a reserved word. The value of children property is the content inside the custom component which will be displayed inside.
7. The custom component definition will also require us to mention the classes that we add to the component name in other files. It only supports only what we tell it to support.

**L-40 Summary**

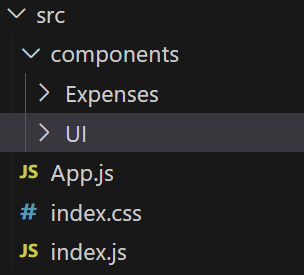
1. Component Drive UI
2. React Core Syntax and JSX and custom components.
3. Working with data across components through the props.children concept to send jsx html code.
4. In chrome Developer tools, we will only see native html tags and not react custom components.

**L-41 Deeper JSX**

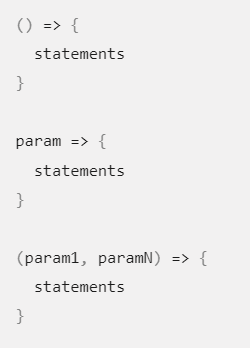
1. We used to import react from react everytime, but now in the new jsx we do not need to do so.
2. Earlier it would be – return React.createElement(‘div’, {//attributes}, , ,// other args which are the child elements and sibling elements being created through a nested use of React.creaetElement() , ,);  
   



1. This is why we need to wrap it in a single div, under the hood react uses createElement function to make the element, and we can not do so with 2 elements. One element requires one function call and we can not make 2 elements with one createElement function. We can use an array of elements, which is essentially done in a way by wrapping things in a div.

**L-42 Organising components**

1. We can divide the components like – UI component, feature component, etc.



**L-43 Arrow function syntax**