L-1 Hello world using JS

```
<script>
const heading = document.createElement('h1');
heading.innerHTML = 'Hello world from JS';
const root = document.getElementById('root')
root.appendChild(heading);
</script>
```

H.W
cdn
cross origin

React

```
const heading = React.createElement("h1", {}, "Hello World from React") // React
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(heading)
```

(what to make, attributes, what to put, children)

console.log(heading) → gives object

ReactElement(Object) → HTML (Browser understands)
↳ Rendering

```
<div id="parent">
    <div id="child">
        <h1></h1>
    </div>
</div>
```

⟹

L-2 Bundler - parcel

^2.8.3 → caret → automatically update minor version without incrementing major version

tilde

→ transitive dependencies.

- Two types of dependencies
  1) dev 2) normal

package.js vs package-lock.js

"integrity" → hash to verify only same version as local environment to production

- node_modules

If we have package.json, package-lock.json, we can recreate node modules.

- npx parcel index.html (development)
  → installing executing package

npm
→ installing package

npm install react
npm install react-dom

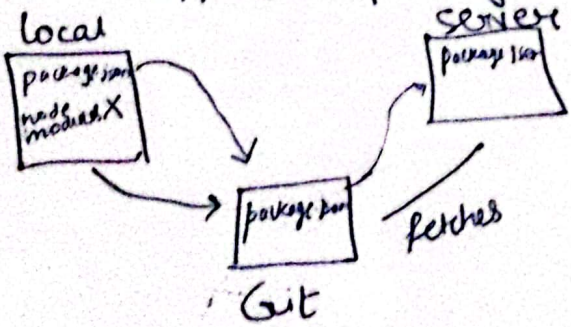
*Normal browser cannot have import <script type="module">

- parcel features

→ Devbuild, Localserver, HMR, FileWatching Algorithm - C++
Image optimisation, Minification, Bundling, Compressing
Consistent hashing, codesplitting, differential bundling - support older browsers
can also host on https.
Tree Shaking - remove unused code.

→ npx parcel build index.html (Production)
↳ /dist

Remove:
main: "App.js" in package.json.




```
"browserslist": [
    "last 2 version"
]
```

JSX

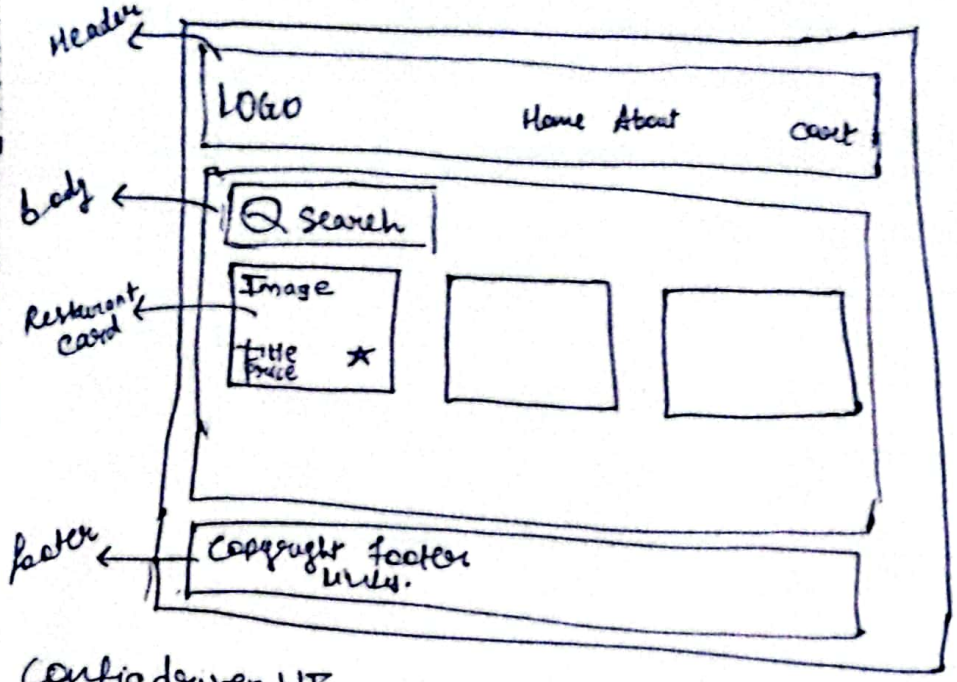
It is not HTML inside JS
HTML like Syntax
Parcel is transpiling JSX
JSX is not HML.
<h1 class="react"> JSX <h1 className="react">

React.createElement → ReactElement (object) → HTML Element (Render)

JSX → React.createElement

Babel transpiles JSX to createElement

★ Food ordering



Header
- LOGO
- Navigation

Body
- Search Bar
- Restaurant List
  - Restaurant card
    - Image
    - Name
    - Price
    - Rating

footer
- Copyright
- Links
- Contact

Config driven UI

- React is fast
  → DOM manipulation faster, Efficiently.
- Two types of Export/Import
  1) Default
     export default Comp
     import Comp from "path"
  2) Named.
     export const Comp
     import {Comp} from "path"

Every child must have its own key.
Otherwise it will re-render entire page (optimization) ⇒ key

- Hooks.

1) Normal JS utility functions.
   - useState() - Superpowerful State variables (Named import)
     useEffect()

Whenever state variable updates, React re-renders the (whole) component ★★★★★★

const [res, setRest] = useState([]) default value

const arr = useState(resList)

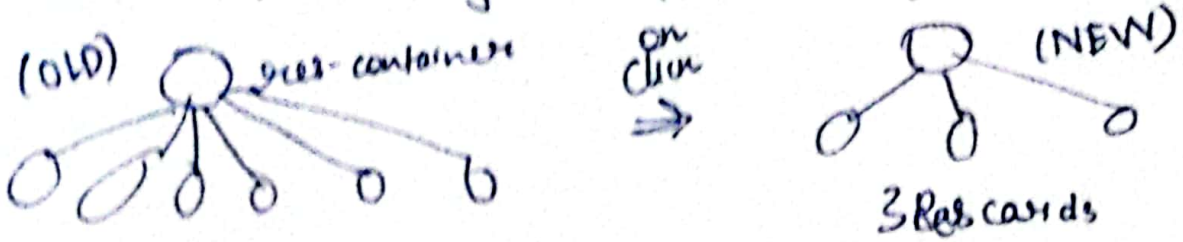
arr[0]

arr[1]

∴ setRest is able to update value of (res) while rendering.

const ←

☆ Reconciliation Algorithm (React Fiber) [React 16] aadlite/react



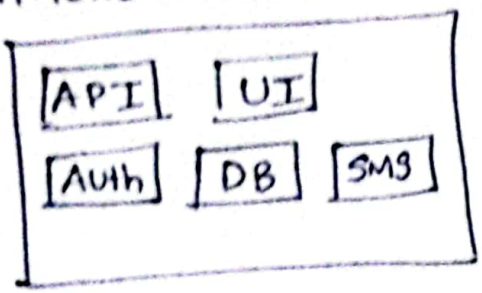
Virtual DOM is representation of actual DOM →   <img>

Object

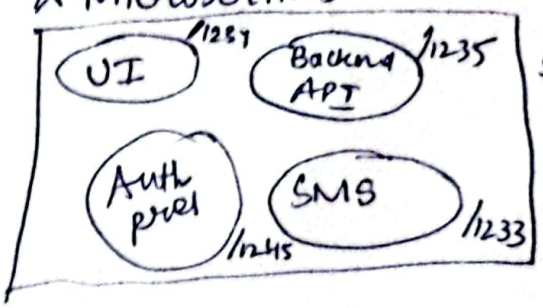
☆ Diff Algorithm invokes second function is weState()
→ Finds out differences bet" updated Virtual DOM & prev Virtual Dom and then actually update the DOM in one render cycle
difference is upd reflected on DOM

∴ React is Fast

☆ Monolith Architecture



☆ Microservices



Seperation of concern

☆ Fetch an API

① Loads → API (500ms) → Render (100) } 600ms

② Loads → Render (100ms) → API (300ms) → Update Re-render app (1000) } 500 ms

Better UX

☆

```
useEffect(()=>{
  setInterval(()=>{
    console.log()
  },1000)
  return ()=>{
    clearInterval()
  }
},[])
```

called after page is exited.

UseEffect.

① Called after every time a component render (if no dependency array)

useEffect(() ⇒ { }, [ ])

② dependency array empty, useEffect is called on initial render (just once)

③ if dependency array consist of state variable, useEffect is called every time that state variable updates.

✱ UseState

Never use this hook outside body of function component/if/for/function.

Whenever state variable updates. React re-renders whole component.

✱ CLASS BASED COMPONENTS.

- Loading a class based component means, creation of instance of this class → mounting a component

Inside constructor,
props, State variables.
(Super(props)) this.state.

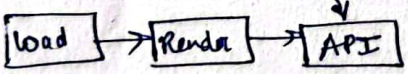
✱ Lifecycle.

Parent constructor → Parent Render → Child constructor → Child render

Component did mount called after render and called only once

If Parent's child is mounted properly, then parent component did mount called.

componentDidMount → use to make API call.



Optimisation.

React will batch render phase of two children

WHY?

componentdid mount

this.timer = setInterval(() ⇒ { }, 100

It will remain counting even after page is closed.

The page is not reloaded ∴ unmount is used.

clearInterval(this.timer)

☆ Chunking | Codesplitting | Dynamic Bundling | Lazy Loading | On demand Loading

Break down code into smaller chunks. Dynamic Import

Makemytrip.

↙ ↓ ↘

Flight trains buses

```
const Grocery = lazy(() => import("path"))
```

☆ Tailwind CSS

w → width

p → padding px → padding x-axis.

m → margin.

mb → margin bottom

☆ Higher Order Component.

→ Function Takes a component, & returns a new component.
enhances it

UI Layer	Data layer
JSX	State, props.

Controlled / Uncontrolled comp.

Lifting State up.

Props drilling (props passing through children to grandchild & soon.

- React context used to avoid props drilling

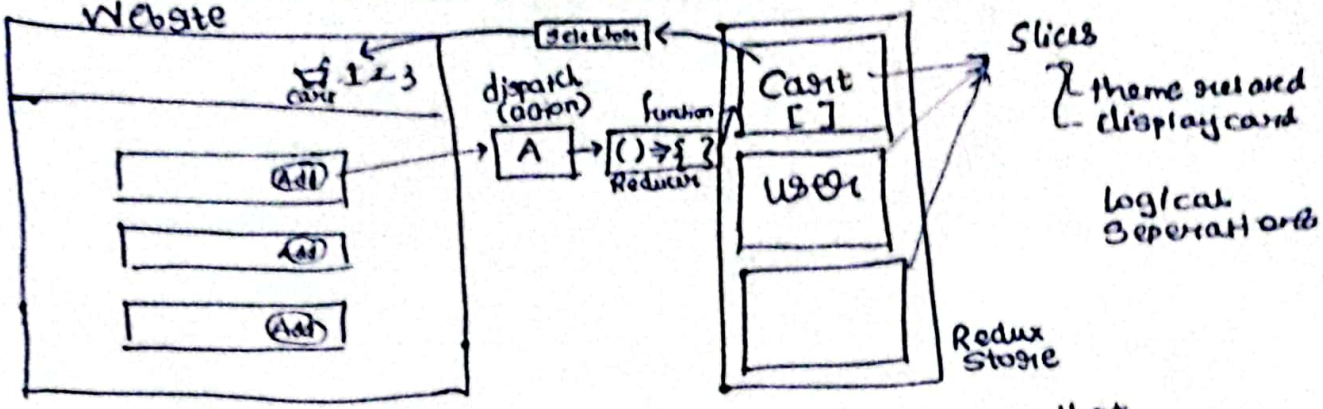
Context is kinda a global ~~thing~~ ~~object~~ Space ~~set~~ provide to whole app or any one compo

useContext takes argument as createdcontext.

UserContext.Provider

☆Redux

Used for handling State of application, easier debugging.
Redux Store is big object & kept in global central space. i.e any component can access data from this Store.



When clicked on Add button, it dispatches an action that it calls a function & this function modifies the cart Slice.
This function is known as Reducer.
Header component is subscribing to the store which means it is sync with the store through Selector and the cart Slice will get updated.
This store is provided to our parent component

Provider comes from react-redux
ConfigureStore from reduxtoolkit

Actions → Create, Read, Update, Delete

A store also have reducer for itself for modyfying itself

Selector is a hook inside React
useDispatch

★★
★ Whenever we are using selector make sure we have Subscribed
★ to that portion of store.

```
const store = useSelector((store) => store);
const cartItems = store.cart.item
```

→ Whenever anything changes inside store, we need not store the update of whole store