
React - a javascript library!

Episode1 - inception

- browsers do not know what's react.

  add react to the browser.

  cdn link - the place where react library is hosted, we have to add that link to our application

  ↓

  now react is added to the project

What is cdn

Content delivery network

* geographically distributed servers
* cache content close to end users.
* quicle transfer of assets — for loading html, js files, css, images etc.

The content is cached near website edge — improving performance.

what is cross origin + what is inside cdn link

inside cdn link.

* as react is a js library, the link contains the code for react.
* react code written by facebook developers
* First file - react.development.js
  - core of react
  - code for react
* 2nd file - react.dom
  - what we need to manipulate DOM
  - used in browsers.

react is used not only in browsers but also on different platform

difference between development and production links.....

hello world in react.

① Create elements in React → creates an object

* use React.createElement("element", {object}, "what you want to send inside the element");

gives attribute to the element. example id,

```
React.createElement("h1", {}, "hello react");
```

② Create Root to attach the object to.

* use ReactDOM.createRoot(document.getElementById("root"));

③ Attach object to root → takes the object and converts to UI tag and puts in the html.

* root.render(heading);

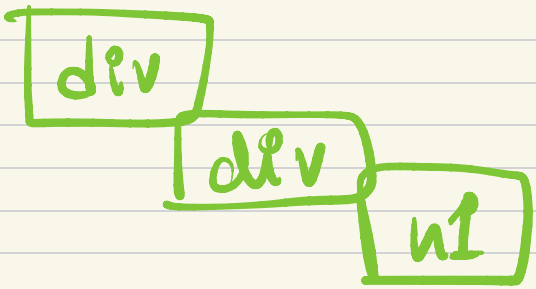
react philosophy - manipulate the DOM only using javascript

this optimizes the costly operation which is adding, deleting elements in DOM.

- If there is something in HTML root already, when render works, it will replace the html tags and renders from java.

nested elements.

at the end of the day, react.element is an object.




```
JS App.js > ...
1  const parent = React.createElement("div", {id:"parent"},
2      React.createElement("div", {id:"child"},
3          [React.createElement("h1", {}, "I am H1"), child 1
4              React.createElement("h2", {}, "I am H2")]));child 2
5          const root = ReactDOM.createRoot(document.getElementById("root"));
6          root.render(parent);| ← render parent.
```

when making siblings, pass an array of children

[child1, child2]

The react code looks messy and complicated, we usually use jsx to write react to make our code clean and easy to read.

order of files ....

1. import your react ← loads react
2. then connect your js file ← loads js file

React - a library

- react can work in any part of the application.
- whatever you make your root, that portion is where react will work.
- Its not a full feeged framework.

- dont panic, react is easy to understand.
- have curiosity and explore around.

→ CURIOSITY ←

Episode 2: igniting our app

NPM: No full form.

it is everything
BUT node package manager

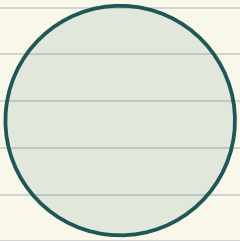
it basically manages packages

its a standard repository for all packages

→ all libraries and utilities.

React has npm configured.

* project has to be configured with npm by:
  - npm init
  - makes package.json



most important package -

bundler

to make the code. production ready
- compress
- packages
- minifies.

we will be using parcel as a bunder

builds the app to be bundled to product

node package.

parcel : use to ignite the App.

current code = skeleton
parcel will give muscles.

dependencies

dev dependency

only used in development environment

npm install -D parcel

dependency

can be used in the production environment

dev dependencies

^ caret – parcel will automatically update minor updates installed, NOT major.

~ tilde – the major version is installed automatically.

when new versions of packages comes in

package lock.json → keeps track of all different versions

---

node module – very heavy.

downloads all packages from node

Transitive dependencies – we wanted parcel but parcel itself has more dependencies, and we install all the packages needed

all libraries have their own package.json which track the dependencies etc.

Should we push all these huge dependencies on github? No, use .gitignore

Should we push package.json and package-lock.json Yes, we need the info about dependencies.

if you have package.json and package lock, you can regenerate the modules. you need to put these 2 files to github.

Me

whatever you can regenerate do not put on git

npx → executes the package.

we are executing the parcel now.

parcel uses HMR ↗ hot module replacement.

- reading all the files and keep track.
- file watching algorithm is used.
- what we change, it will build it once again.

The build time gets reduced when you run — faster builds — caching.] parcel - cache

- image optimization
- minification
- bundling
- compressing.] parcel manages everything.

- consistent hashing - read about
- code splitting
- differential bundling
  - ↓
  - support older browsers.
- https support provided
- tree shaking
- different dev and production bundles.

Tree shaking algorithm

- removes unused code that you are not using

What makes your web app fast

- write about how pascal is helping

development build

dist folder is made

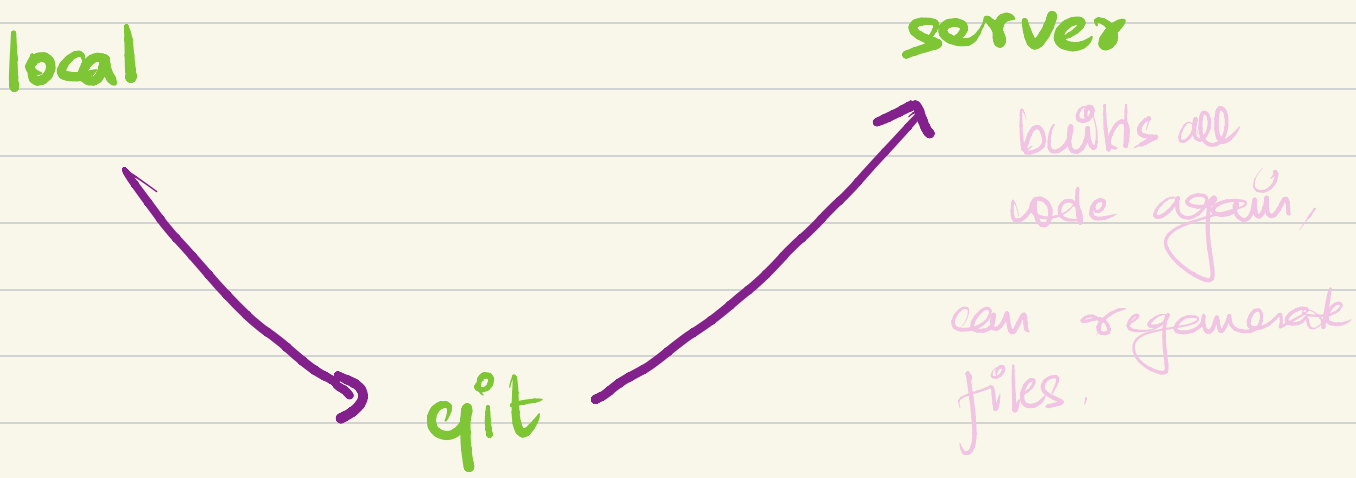
everything is hosted in dist.

parcel uses cache and dist to generate and host your application.

the three files that are compressed and minified and its production ready.

→ all the other files are just mapping file.

cache and dist should not be added to github. because they can be regenerate.



browserslist

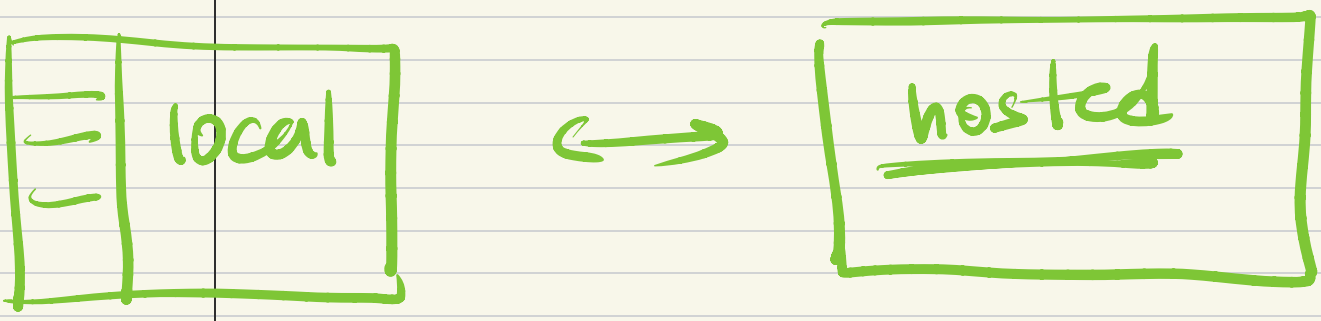
↓

tells you how specifically you can configure your app.

hot module replacement

(HMR)

keeps updating and refreshing the changes you have made



Episode 3 :

Laying the foundation.

npx parcel index.html ← command used to run.

short way.

add inside package.json → scripts

↑ key word reserved by npm

"start" : "parcel index.html"

"build" : "parcel build index.html".

run command npm run start

or

npm start

↳ does not work for Lata

recap

react element → an object (its not an html element)

'createElement" → used to create an element

when this element is rendered to dom, then it becomes html element

↓

react dom . create root renders the object

React element = Object

anything under the root will be replaced by the by whatever is being rendered.

Clunky syntax

solution ↓

JSX → syntax for react

- react and jsx are both different
- jsx is just something that makes writing in react easy.

normal apps → html : structure
css
javascript : logic

all frameworks try to merge everything together. JSX merges html + js

* jsx syntax looks like HTML
  it is NOT HTML inside javascript
  , just looks like it. (HTML like or XML like)

jsx

HTML

React Elements

Create an element in jsx

```
const jsxheading = <h1> hello! </h1>;
```

jsxheading → react element

jsx is only syntax

no difference indeed when using jsx or core react.

→ write your code for humans, so that developers can understand what you have written

React components

everything is a component in react

① class based components — old, nobody uses.

② Functional component — new, everyone uses this.

> React component =
> normal js function
> which returns
> some jsx element

→ A function that returns a piece of jsx element is called functional component.

jsx element = React element

always start with capital letter

render

```
root.render(<HeadingComp />);
```

to render something inside a div

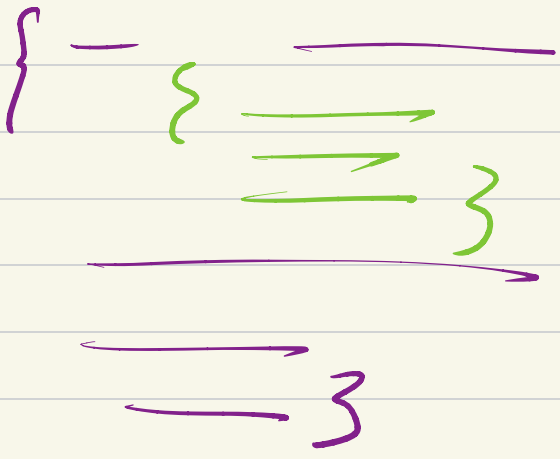
```
< component />;
```

component compositions

↕

nested compositions

- component composition, putting one component into one another.
- use arrow functions



put anything inside JSX

{ write anything }

inject any javascript code into jsx.

```
<div>
  <h2> {any component} </h2>
  ==========
</div>
```

{ }

↓

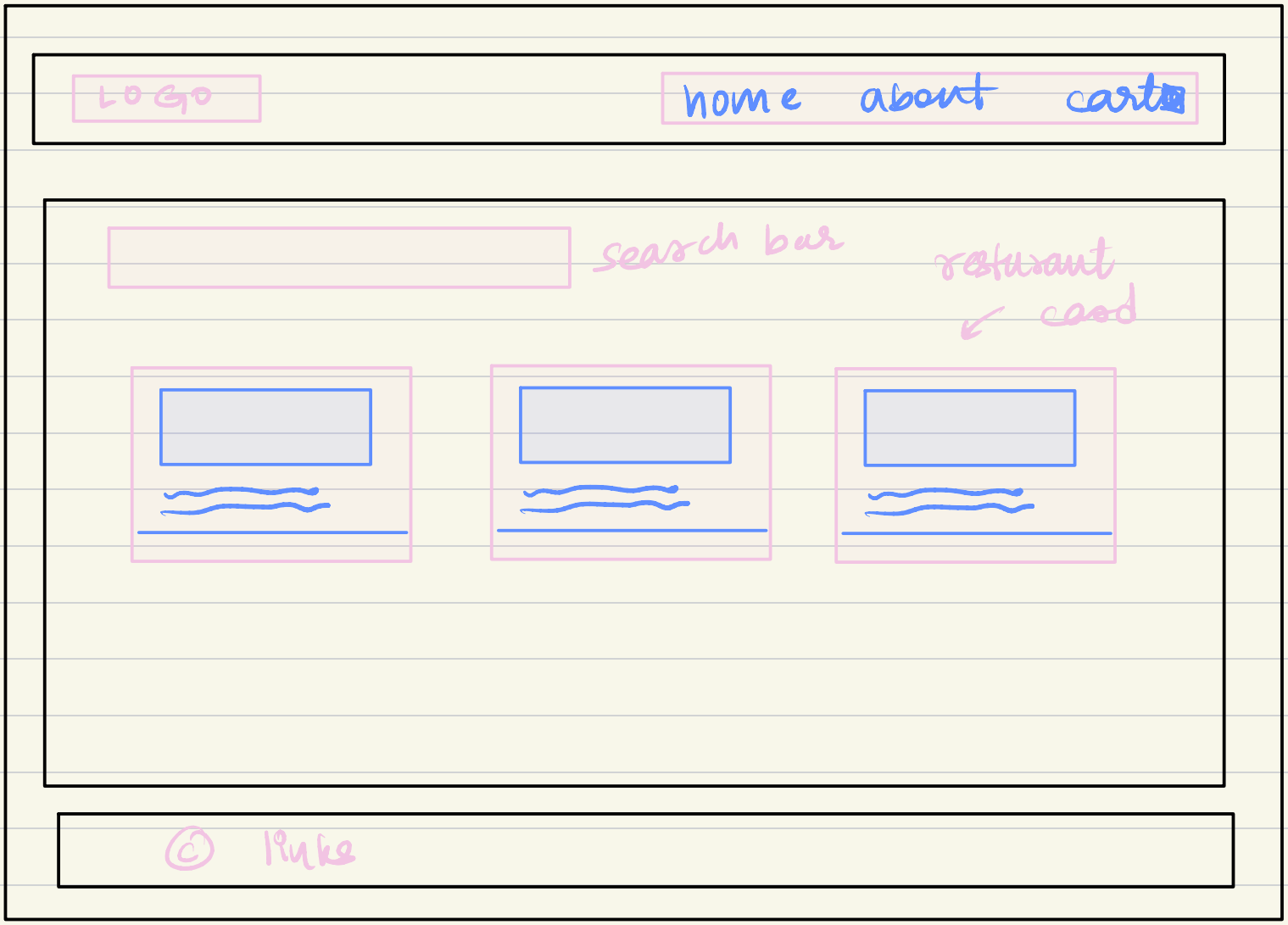
this is sanitized before executing

Episode 4

[food ordering app]

plan your app, before coding it
- think about everything

- wireframe -



- Low level planning -

what components ?

1. header
   - logo
   - navigation items (home, about, cart)
2. body
   - search bus
   - Restrwat vontainer (with all cards)
     - restusant card
3. footer
   - copy right
   - links
   - address
   - contact information

- code -

1. build the 1st component [layout component]

functional component.

return

{ //header
//Body
//footer }

all these are components.

2. build header component.

heade

logo

nav items

Tips

1. always create a component for something that needs to be repeated many times.

2. when adding javascript to the esx use {}

3. CSS is written as a javascript object

```
const styleCard = {

    background color : "#black";

}
```

dynamic data.

earlier in the restaurant card we hard coded info about restaurant, now pass in the info dynamically.

previously

```
rest card = ( ) => {

    return (
        <div>
            <img>
            <h2> Meghana food </h2>
            <rating>
            <time>
        </div>
    )
```

now use Props to make data dynamic

Props are arguments to components

properties

similar to passing arguments to function

javascript object

make this dynamic

```
Restaurant card = ( props ) => {
```

destructuring → resName, cuisine

```
(
  <h3> {props.resName} </h3>
                ↑ resName
  <h2> {props.cuisine} </h2>
                ↑ cuisine
)
}
```

this will work as per the args passed

Meghana foods

KFC

```
Body () => {

return (

  <RestaurantCard
    resName = "Meghna food";
    cuisine = "north indian ... ~"
  />

  <RestaurantCard
    resName = "KFC"
    cuisine = "burger ... ~"
  />

  );
};
```

these are args to restaurant card

these arguments will be wrapped into a js object

destructuring → just write as arguments in the restaurant card component.

easier to read.

recieving props.

→ props are wrapped and passed as js object.

→ then you have to destructure the js object to access different values.

real world data comes in as

JSON.

config driven UI

* front end system design

- UI is driven by data (config) configuration based
- data is the config
- data is different for different cases
- UI gets changed according to what comes through backend.

SWIGGY's data.

go into the website and go to inspect → network
get the real data. in the form of json objects.

not about data as array objects =

array →

" cuisine = [~ , ~ , ~]

res.data.data.cuisine.join(", ")

array join - in javascript

CDN → cloudinary.com

content delivery system.

that swiggy uses for images

map function to loop over all data for restaurants and render a restaurant card

```
reslist.map((restaurant) => (
    <RestaurantCard resData={restaurant} />
))
```

var

props

for the variable restaurants, it will loop through all resto and render restaurant card for each var. with resData = restaurant

↓

information for each restaurant

unique key property.

each of list items should have unique key.

key = {restaurant.data.id}

```
( <RestaurantCard resData={restaurant}
    />
)
```

React says, (not recommended)

dont use index as keys.

[if there is no unique id in data, then you may use index]

```
resList.map( (restaurant, index) => (
  <RestaurantCard key={index} resData=
    {restaurant} />
```

* destructuring
* optional chaining
* map instead of for loop

Study these.

why need IDs?

its for performance optimization.

* DOM rerenders all the components if there are no id's and a new component comes in.
* If there are id's, react knows the old components and places the component according to its ID.