NodeJs & React Intermediate

IMY 220 ● Class Discussion

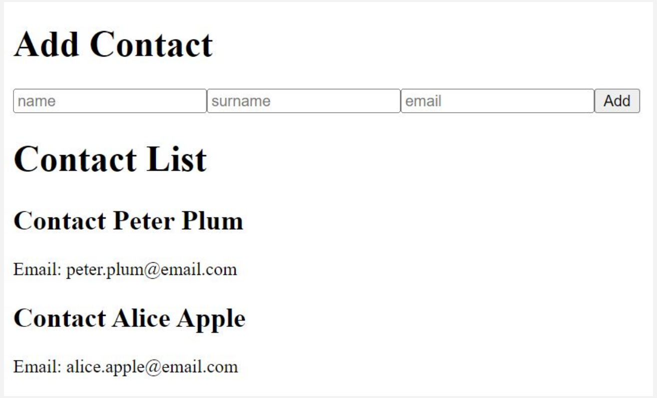
Quiz!

https://quiz.com/77046fbc-0b36-4882-84e1-48326ed70858

⭐ Component State - Exercise

Note: Using babel CDN (do NOT do this in production - very slow!)

- *Just to get used to it*



Add Contact

Contact List

Contact Peter Plum

Email: peter.plum@email.com

Contact Alice Apple

Email: alice.apple@email.com

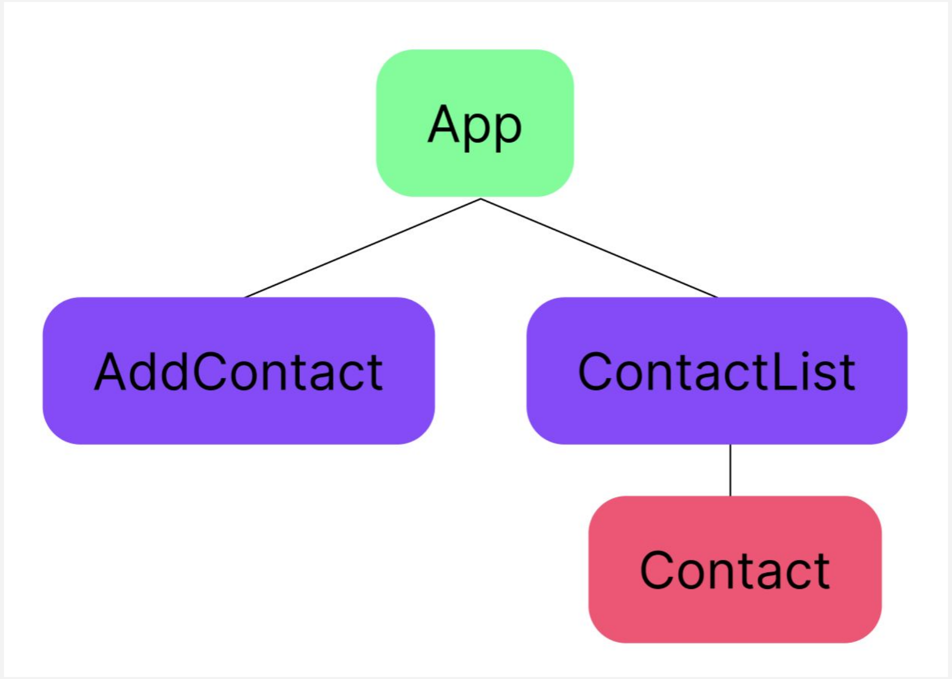
Use the same files off of ClickUP. In the file called **App.js** implement (10-15mins):

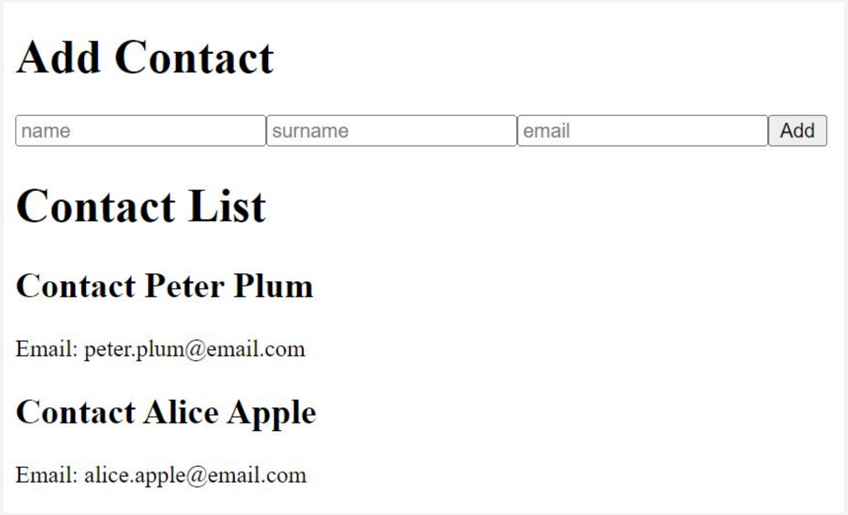
- A component called **`AddContact`** that contains a **form** to add contacts to the list. It should define a function called **addContact** that it uses to add contacts to the list by **getting the form values (hint: ref)**, and then calling **onContactAdded** that is passed in as a prop.
- A component called **App** that acts as the parent component.
  - It should have two children: **AddContact** and **ContactList**.
  - It should take in a list of contacts as its prop and use this to initialize its state. It should pass this state to its child **ContactList**.
  - It should also define a **onContactAdded** function which updates its state (adds a contact to the array). It should pass this function to its other child **AddContact**.

*Render the **App** component to index.html, passing it the provided list of contacts.*

⭐ Components - Exercise

Final structure should look like this:




Add Contact

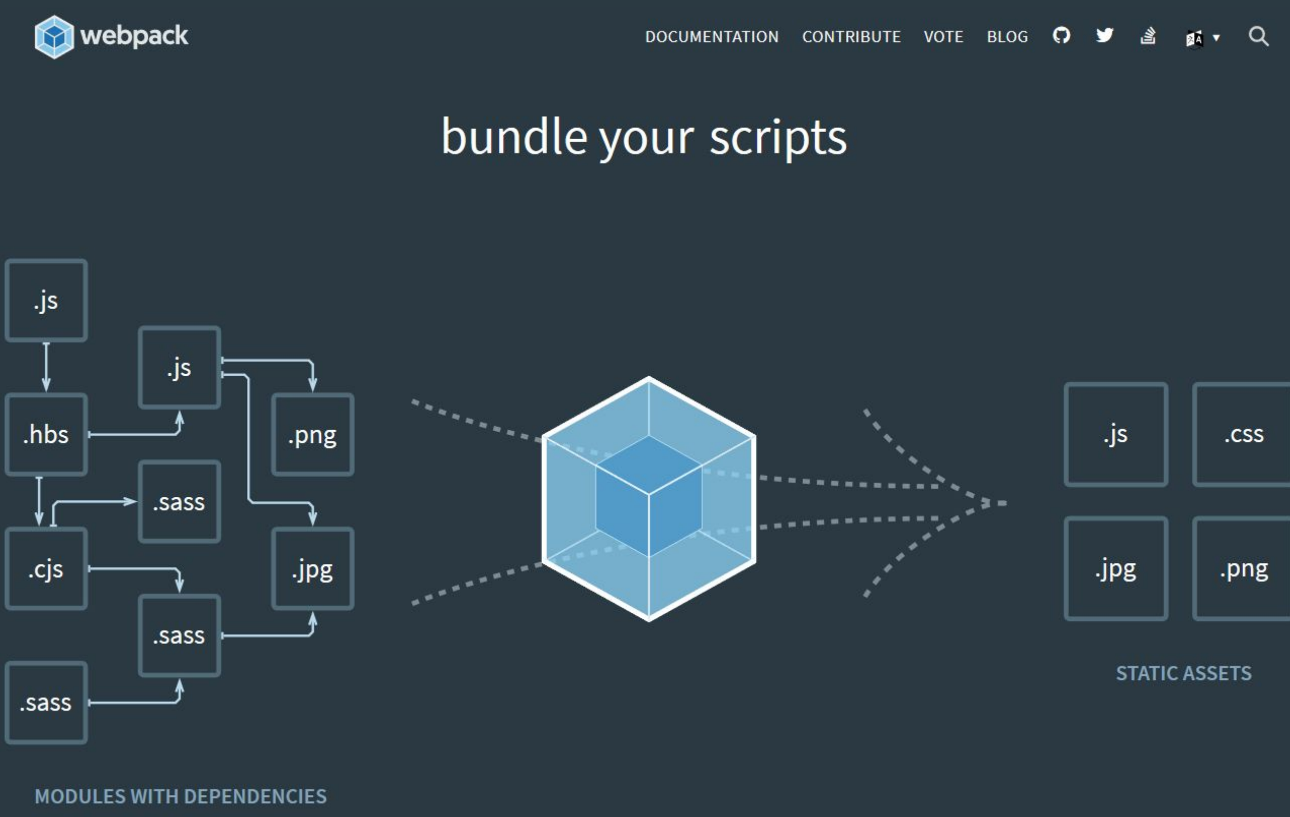
Contact List

Contact Peter Plum

Email: peter.plum@email.com

Contact Alice Apple

Email: alice.apple@email.com


webpack
DOCUMENTATION
CONTRIBUTE
VOTE
BLOG
bundle your scripts
.js
.js
.hbs
.png
.sass
.cjs
.jpg
.sass
.sass
MODULES WITH DEPENDENCIES
.js
.css
.jpg
.png
STATIC ASSETS

Babel

“Babel is a tool that helps you write code in the latest version of JavaScript. When your supported environments don't support certain features natively, Babel will help you compile those features down to a supported version.”

https://github.com/babel/babel

- Babel also contains presets that allow you to use other syntaxes as well, such as JSX
- **Babel can thus be used to compile JSX code into valid JS code.**
- Babel also requires a configuration file called ***.babelrc*** to work. This tells Babel to transpile all ES6 code by default using its environment preset (Babel has many other presets, including the react-preset)

Webpack

Webpack takes one/more files to be bundled and optionally uses some plugins, loaders, transpilers, etc. to deliver an output file that is usable by our main webpage files.

Similar to Babel's *.babelrc* file, Webpack requires a configuration file to work properly, called *webpack.config.js*

```
class Person extends React.Component {
    render(){
        return (
            <li>{`${this.props.person.name[0]}. ${this.props.person.surname}`}</li>
        );
    }
}

class AddPersonForm extends React.Component{
    constructor(props){
        super(props);
        this.submit = this.submit.bind(this);
        this.nameInput = React.createRef();
        this.surnameInput = React.createRef();
    }

    submit(e){
        e.preventDefault();
        let name = this.nameInput.current.value;
        let surname = this.surnameInput.current.value;
        this.props.onNewPerson(name, surname);
    }

    render(){
        return (
            <form onSubmit={this.submit}>
                <input type="text" ref={this.nameInput} /> <br/>
                <input type="text" ref={this.surnameInput} /> <br/>
                <input type="submit" value="Add" />
            </form>
        );
    }
}
```

```
class PersonList extends React.Component {
    constructor(props){
        super(props);
        this.state = {people: this.props.people || []};
        this.addPerson = this.addPerson.bind(this);
    }

    addPerson(name, surname){
        this.setState({people: [...this.state.people, {name, surname}]});
    }

    render(){
        return (
            <div className="container">
                <h1>
                    {this.state.people.length} in the list:
                </h1>
                <ul>
                    {this.state.people.map( (person, i) => <Person key={i} person={person} /> )}
                </ul>
                <div>
                    <AddPersonForm onNewPerson={this.addPerson} />
                </div>
            </div>
        );
    }
}

var peopleList1 = {
    name: "Troy", surname: "Barnes",
    name: "Abed", surname: "Nadir"
}

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<PersonList />);
```

Node + React

A better way to do this would be to create a separate file for each component and to import the different components as each file needs it.

This way, each component is logically separated into its own file and *index.js* simply renders the main component.

Node + React

1. Move each class declaration into its new file. (e.g. *Person.js*)
2. Also export the class declaration for the component. We could do this using default exports, but since this is somewhat of a bad practice, we'll export the class definition instead.

Node + React

```
// Contents of AddPersonForm.js inside the new "components" directory

import React from "react";

import {Person} from "./Person";
import {AddPersonForm} from "./AddPersonForm";

export class PersonList extends React.Component {
   // rest of class definition goes here (see L20 - React 2)
}
```

Node + React

And finally, since PersonList imports the other two components, we only need to import PersonList when we want to render it inside *index.js*

```
import React from "react";
import ReactDOM from "react-dom/client";

import {PersonList} from "./components/PersonList";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<PersonList />);
```

Prop Types

Since JS is a **loosely typed** language, you can supply a variable with a value of any type without getting an error

This can lead to some confusion, such as if a variable is given a value which is syntactically acceptable, but leads to **logic errors**

(For example, unknowingly adding a numerical string to an integer)

To prevent this type of confusion, React provides functionality for Property Validation in the form of **Prop Types**.

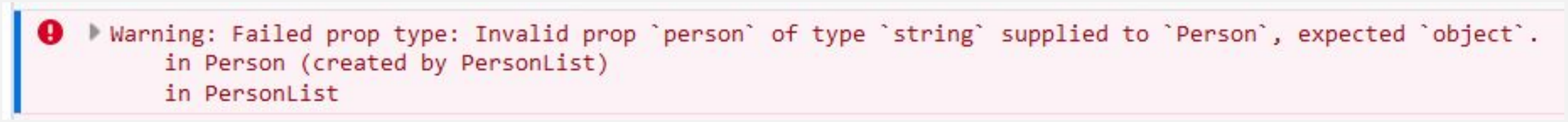
```
import React from "react";
import PropTypes from "prop-types";

export class Person extends React.Component {
    render(){
        return (
            <li>{`${this.props.person.name[0]}.
                  ${this.props.person.surname}`}</li>
        );
    }
}

Person.propTypes = {
    person: PropTypes.object
}
```

Prop Types

If we now try to create a Person component with a person prop that is anything except a JS object, we'll get a descriptive error

For example, if we try to add one as follows inside *PersonList.js*

```
{this.state.people.map ( (person, i) => <Person key={i} person="Name" />
)}
```



```
Warning: Failed prop type: Invalid prop `person` of type `string` supplied to `Person`, expected `object`.
    in Person (created by PersonList)
    in PersonList
```

Prop Types

It is always a good idea to do this, as it can help debug otherwise tricky errors

The Prop Types library has many built-in type checks, such as:

- PropTypes.array
- PropTypes.bool
- PropTypes.func
- PropTypes.number
- PropTypes.object
- PropTypes.string
- PropTypes.symbol

Prop Types

It is also a good idea to validate the existence of a required prop using PropTypes.isRequired

You can also chain Prop Type validation and add validation for multiple props simultaneously, for example:

```
Person.propTypes = {
     person: PropTypes.object.isRequired,
     example2: PropTypes.func.isRequired
}
```

You can find many more examples of Prop Type validation here:

https://react.dev/reference/react/Component#static-proptypes

⭐ PropTypes - Exercise

Note: Using babel CDN (do NOT do this in production - very slow!)

- *Just to get used to it*

Use the same files off of ClickUP. In the **index.html** file, implement (5 mins):

- Add Proptypes to the **Pet** component so that the props that are passed in can be checked.
- You can assume the given props are correct.
- Hint: Find out how to validate objects with fields here: https://www.npmjs.com/package/prop-types
- All of these props, besides the adoption price, are **required**.

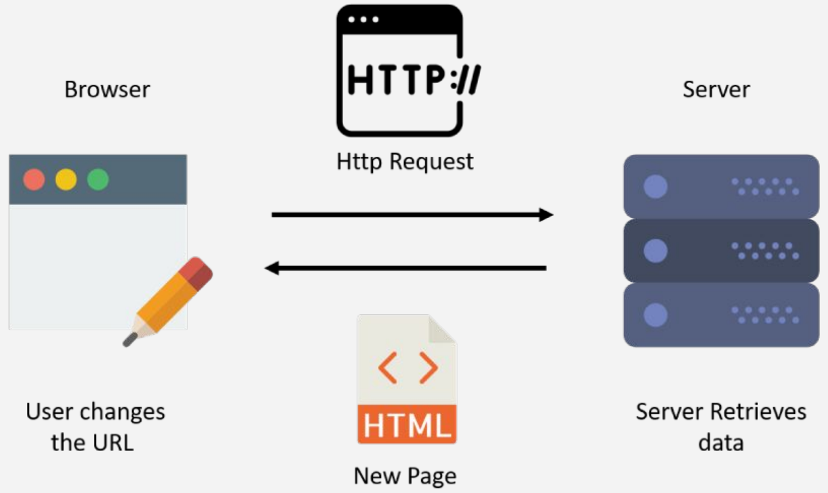
After you have written this, mess with the props passed in to see the results of passing in the wrong props.

React Router

How Routes and Requests Work

Recap from COS216

1. Client navigates to URL
   a. “https://blahblah.com/**home**”
   b. “.../**home**” is the route
2. Client makes a request to server
   a. “Fetch me the **home**.html page”
3. Server finds the file and sends it back to Client
   a. “Here is **home**.html”
4. Otherwise 404 not found :(

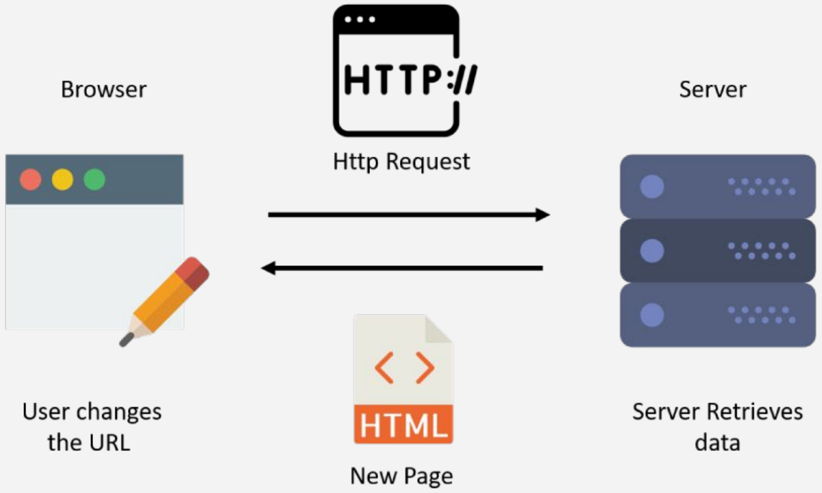


How Routes and Requests Work

This happens for **every single file your website needs** (.html, .css, .js)

> <link **rel=”`style.css`”** type=”text/stylesheet”/>
>
> <script **src=”`index.js`”** type=”text/javascript”></script>
>
> These are **both requests to the server for `style.css` and `index.js`** files

Every time your user navigates to a new page by clicking a **link**, **a new set of files** (.html, .css, .js) must be requested, fetched and returned.

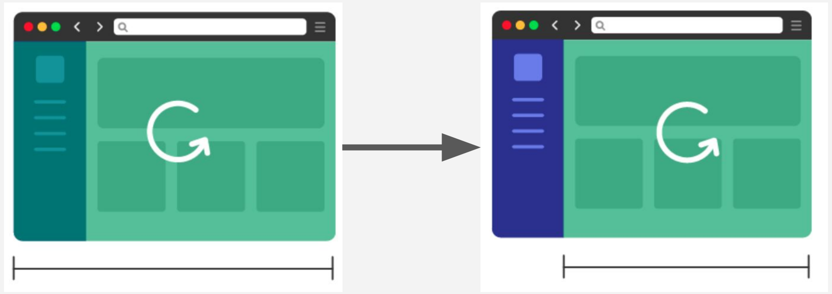


So… What is the Solution?

React Router (react-router-dom)

- A **React Package** that handles the client-side routing for you.

There are other React frameworks that also solve this issue (e.g., *NextJs*). For our purposes we are going to stick to React Router because it’s easier to implement.



So… What is the Solution?

From their docs:

*“Client side routing allows your app to update the URL from a link click **without making another request for another document from the server**.*

[...]

*This enables **faster user experiences** because the browser doesn't need to request an entirely new document or re-evaluate CSS and JavaScript assets for the next page.”*

https://reactrouter.com/en/main/start/overview

How React Router Works

1. **Intercepts** server requests for ‘pages’ e.g., “../home”, “../about”, “../contact”, etc.
2. **Renders** the corresponding component (tree) for each route that’s needed e.g., <Home/>, <About/>, <Contact/>, etc.
    a. I say ‘tree’ because these components can have child components.
3. **Updates** the DOM with the relevant components, without needing to re-render all of them (i.e., reload the page)

*Important: the page does **not** reload. The DOM is simply **updated**.*

React Router

Basic React Router Demo (Old Method)

`<BrowserRouter></BrowserRouter>`

- Demo
    - Setting up the ‘pages’ (components)
    - Installing React Router
    - Setting up the `BrowserRouter` component
    - Setting up the routes (`Routes` and `Route` components)
    - Setting up links (`Link` component)
    - Routing in action

Alternate React Router Syntax (New v6+ Method)

`const router = createBrowserRouter([...]);`

- You may see this syntax being used along with a `RouterProvider` component
- It functions practically the same as the method we just used, but is the newer way of doing things (v6.4+) and is recommended by React Router in their documentation because more properties / APIs you can use with it besides `path` and `element`
- Instead of `<Route path=”” element=””/>`
- You’ll have `{ path: “” , element: “”, …etc. }`

Dynamic Routes in React Router

Called ‘Dynamic Segments’

Specified using a colon (:)

- **e.g., ‘/products/`:id`’**
- Everything else is the same as we have done thus far

When a user navigates using a dynamic route, the route parameter can be accessed in the component that was loaded through React Router **useParams() function**

Dynamic Routes in React Router

However,

- **useParams()** in v6+ does **not support class components**, only functional ones.
- The method of getting around this has also been deprecated and scrapped in favour of functional components :/
- **Luckily, I have two workarounds for this (using v6)**
    - One that **only** works with the **new method**
    - One that works with both the **old** and **new methods**
    - Downgrading to v5 will let you use the established workaround method

Dynamic Routes in React Router

- Demo
  - Setting up a dynamic route e.g., "`/:id`"
  - Accessing the route parameter using `useParams()` using two workarounds

https://stackoverflow.com/questions/58548767/react-router-dom-useparams-inside-class-component

Cookies in React

IMY 220 ● Lecture 5

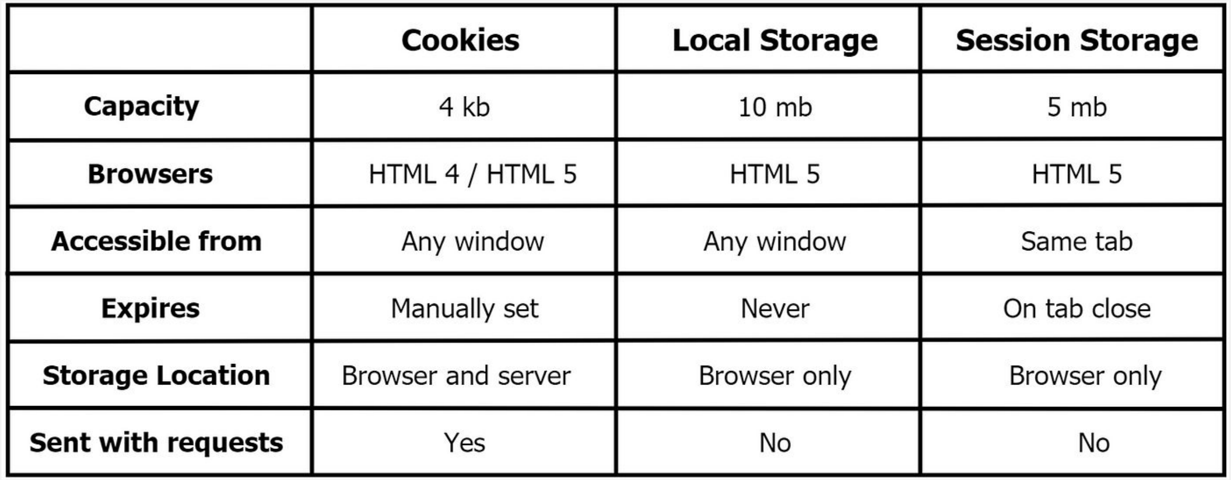
Cookies, Session & Local Storage Basics

COS 216 Recap:

1. Cookies
2. Session Storage
3. Local Storage

Differences?

Why use each?



	Cookies	Local Storage	Session Storage
Capacity	4 kb	10 mb	5 mb
Browsers	HTML 4 / HTML 5	HTML 5	HTML 5
Accessible from	Any window	Any window	Same tab
Expires	Manually set	Never	On tab close
Storage Location	Browser and server	Browser only	Browser only
Sent with requests	Yes	No	No

https://medium.com/@dimplekumari0228/describe-the-difference-between-a-cookie-sessionstorage-and-localstorage-e731a627acb1

Cookie/Session/Local Storage Setup in React

Same as in a regular HTML & JS project.

Where do you put files?

How to get data from components?

How to set data in components?

Demo - Best practices