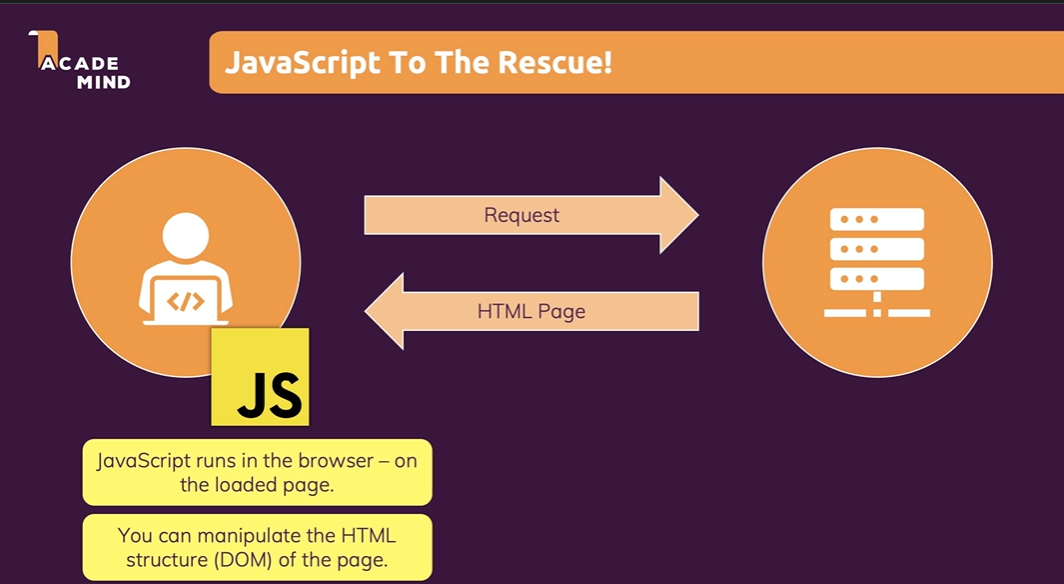
**JavaScript**

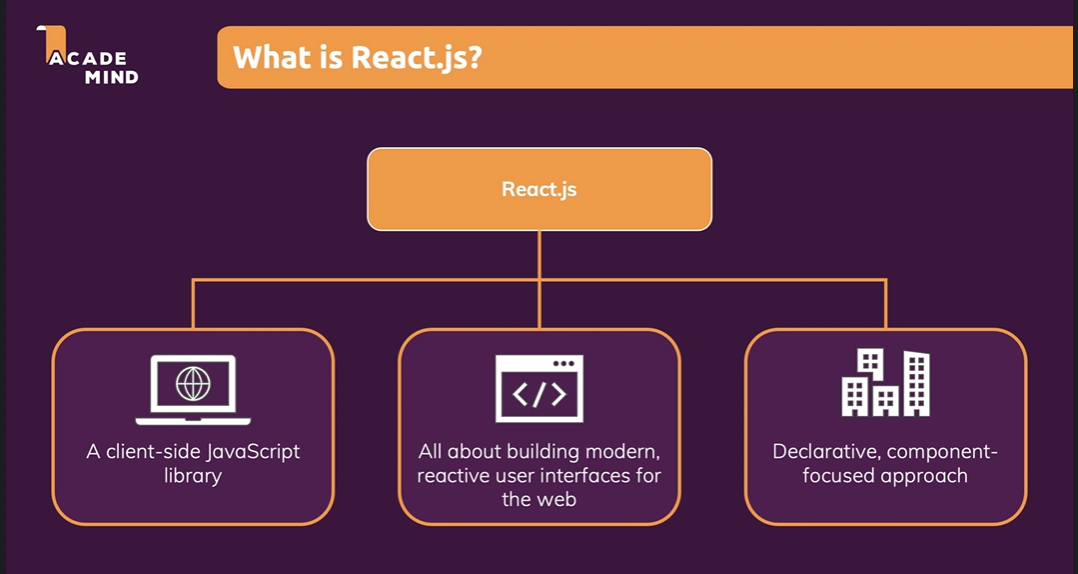
**Es6**

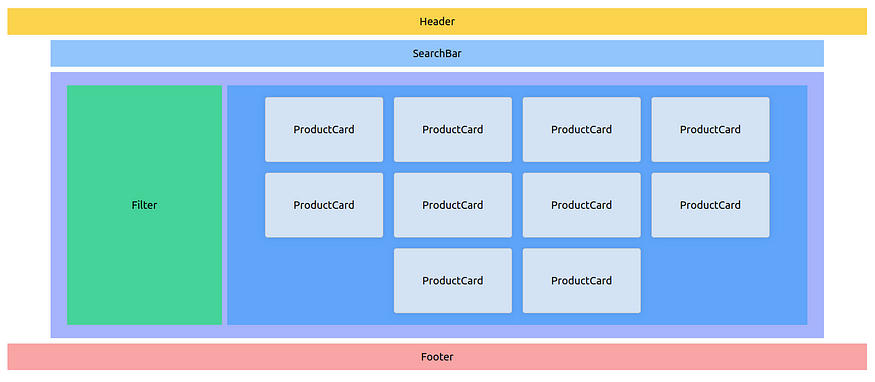
Object

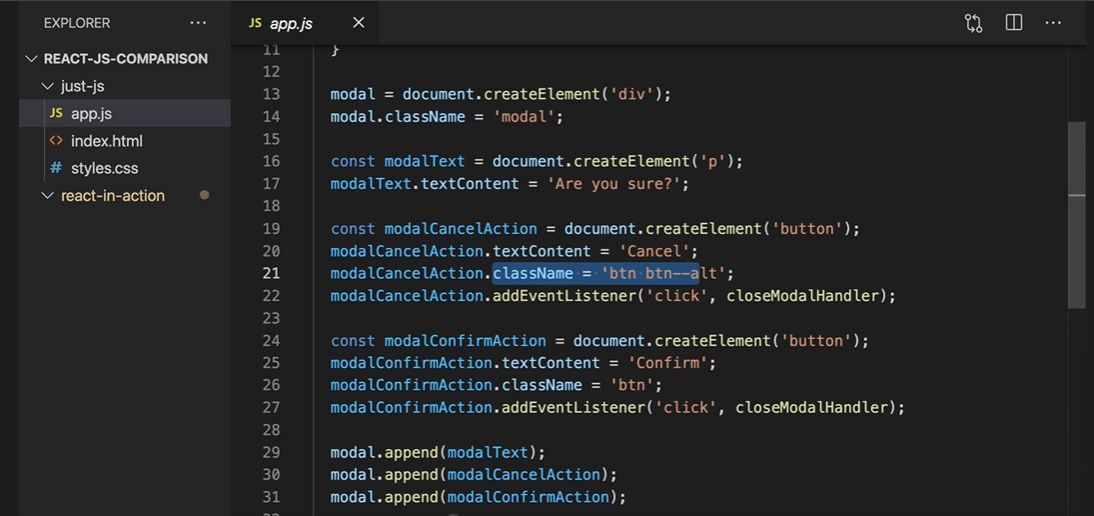
In JavaScript, an object is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics

1. Better user experience
2. Mobile friendly









A picture containing text, font, screenshot

Description automatically generated

React applications are built from isolated pieces of UI called components. A React component is a JavaScript function that you can sprinkle with markup. Components can be as small as a button, or as large as an entire page

Create React Application

* Npm star // to run

**Node.js**

Node.js is a cross-platform, open-source server environment.

Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.

**npm** is a package manager for the JavaScript programming language maintained by npm, Inc. npm is the default package manager for the JavaScript runtime environment Node.js. It consists of a command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry.

Npx

**npx** : An npm package runner — helps to execute packages without installing explicitly. npx makes it easy to install and manage dependencies hosted in npm registry. It simplifies the process and provides a better for executables

npm vs npx

NPM is a package manager used to install, delete, and update Javascript packages on your machine. NPX is a package executer, and it is used to execute javascript packages directly, without installing them q

A screenshot of a computer

Description automatically generated with low confidence

└── src/

├── assets/

├── api/

├── configs/

├── components/

│ ├── SignUpForm.tsx

│ ├── Employees.tsx

│ ├── PaymentForm.tsx

│ └── Button.tsx

├── hooks/

│ ├── usePayment.ts

│ ├── useUpdateEmployee.ts

│ ├── useEmployees.ts

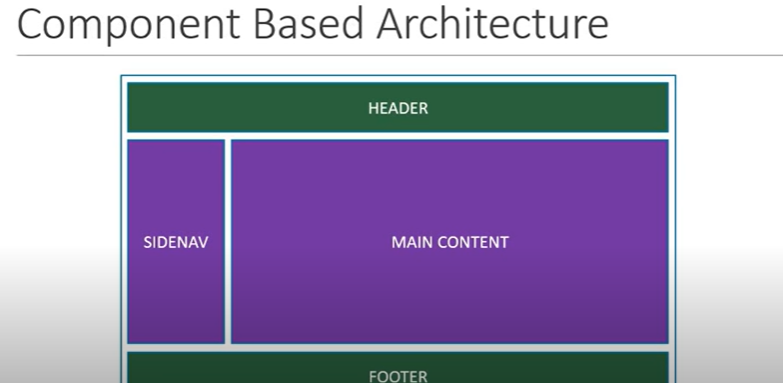
│ └── useAuth.tsx

├── lib/

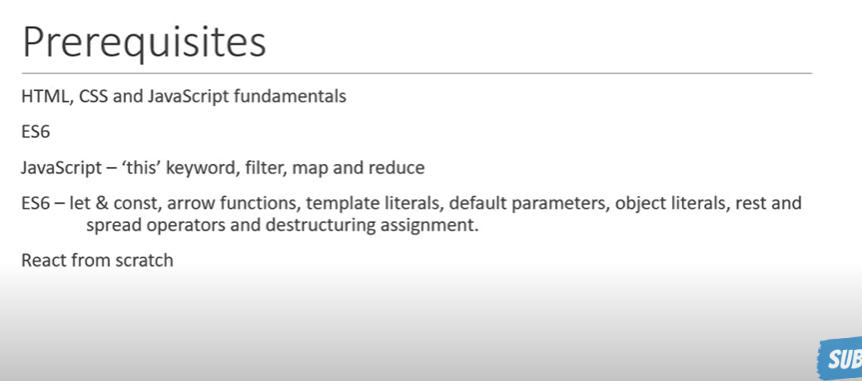
├── services/

├── states/

└── utils/



**React Having component-based architecture.**



A screenshot of a computer

Description automatically generated with medium confidence

The react-router-dom package contains bindings for using [**React Router**](https://github.com/remix-run/react-router) in web applications

Redux

Npm install redux react-redux

The state is a built-in React object that is used to contain data or information about the component. A component's state can change over time; whenever it changes, the component re-renders.

State allows us to manage changing data in an application

A picture containing text, screenshot, font, line

Description automatically generated

* Installation
* How to create and nest components
* How to add markup and styles
* How to display data
* How to render conditions and lists
* How to respond to events and update the screen
* How to share data between components
  + Props
  + Redux
* Adding Styles

There are two primary ways to export values with JavaScript: default exports and named exports. So far, our examples have only used default exports. But you can use one or both of them in the same file. **A file can have no more than one *default* export, but it can have as many *named* exports as you like.**

How you export your component dictates how you must import it. You will get an error if you try to import a default export the same way you would a named export! This chart can help you keep track:

| **Syntax** | **Export statement** | **Import statement** |
| --- | --- | --- |
| Default | export default function Button() {} | import Button from './Button.js'; |
| Named | export function Button() {} | import { Button } from './Button.js'; |

Hooks

Flux

To solve this state management issue, many companies and people have developed various solutions. Facebook, who developed ReactJS, came up with a solution called [**Flux**](https://facebook.github.io/flux/).

**What is flux?**

Flux uses a **unidirectional data flow pattern** to solve state management

A diagram of a model

Description automatically generated

To avoid this

A diagram of a diagram

Description automatically generated

npm install flux@3.1.3

Index.js file is the entry point of your React application. It is responsible for rendering the root component (App. js) and attaching it to the HTML document. It sets up the React DOM and provides the initial rendering of your app.

**JSX**

JSX is stricter than HTML. You have to close tags like <br />. Your component also can’t return multiple JSX tags. You have to wrap them into a shared parent, like a <div>...</div> or an empty <>...</> wrapper

Class => className

**components**

React apps are made out of components. A component is a piece of the UI (user interface) that has its own logic and appearance. A component can be as small as a button, or as large as an entire page.

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation

return (

<>

<h1>{user.name}</h1>

<img

className="avatar"

src={user.imageUrl}

alt={'Photo of ' + user.name}

style={{

width: user.imageSize,

height: user.imageSize

}}

/>

</>

);

**Conditional rendering**

let content;

if (isLoggedIn) {

content = <AdminPanel />;

} else {

content = <LoginForm />;

}

return (

<div>

{content}

</div>

);

Or

<div>

{isLoggedIn ? (

<AdminPanel />

) : (

<LoginForm />

)}

</div>

OR

<div>

{isLoggedIn && <AdminPanel />}

</div>

**Rendering lists**

const listItems = products.map(product =>

<li key={product.id}>

{product.title}

</li>

);

return (

<ul>{listItems}</ul>

);

**events**

<button onClick={handleClick}>

Click me </button>

function handleClick() {

alert('You clicked me!');

}

## Hooks 16.8

Functions starting with use are called Hooks. useState is a built-in Hook provided by React.

**Hooks were added to React in version 16.8.** Hooks allow function components to have access to state and other React features.

Hooks are more restrictive than other functions. You can only call Hooks at the top of your components (or other Hooks). If you want to use useState in a condition or a loop, extract a new component and put it there.

**State**

The state is a built-in React object that is used to contain data or information about the component. A component's state can change over time; whenever it changes, the component re-renders

import { useState } from 'react';

const [count, setCount] = useState(0);

**Props**

Props are arguments passed into React components.

 const callbackFn = (testParam) => {

    console.log(testParam);

    return <h1>{testParam}</h1>

  }

**<Profile prData={name} prData1={profileData} profileToHome={callbackFn}/>**

function Profile({prData, prData1, profileToHome}) {

  profileToHome('test')

  return (

    <div className="home">

      <h1>Nested Profile</h1>

      Name: {prData} <br/>

      Address: {prData1.address}

    </div>

  );

}

Hooks

1. – useState
2. - **useEffect**

React components before. We call these operations “side effects” - because they can affect other components and can’t be done during rendering

useEffect, adds the ability to perform side effects from a function component.

have access to its props and state -

useEffect(() => { //  this component sets the document title after React updates the DOM:

    document.title = `You clicked ${count} times`;

  });

useEffect(() => { //  this component sets the document title after React updates the DOM – state is dependancy

    document.title = `You clicked ${count} times`;

  }, [state]);

useEffect(() => { //  only initial rendering

    document.title = `You clicked ${count} times`;

  }, []);

**AXIOS - api call**

<https://stackblitz.com/edit/react-zag2oi?file=src%2FApp.js>

<https://stackblitz.com/edit/react-axios-api?file=index.js>

**Fetch**

https://stackblitz.com/edit/react-http-post-request-examples-fetch?file=App%2FPostRequest.jsx

***Navigation*** – useNavigate

**Form**

<https://stackblitz.com/edit/react-x78kd8?file=src%2FComponents%2FForm.js>

**Useref**

The useRef is a hook that allows to directly create a reference to the DOM element in the functional component

<https://stackblitz.com/edit/react-7nuita?file=src%2FComponents%2FForm.js>

In a controlled component react, state handles all the form data, whereas, in an uncontrolled component, the HTML form element data is managed by only the DOM.

***Context API.***

***Use To Avoid props drilling***

***Context provides a way to pass data through the component tree without having to pass props down manually at every level.***

***a feature of the React library that helps share data and manage state across components***

What is Context API? The React Context API is a way for a React app to effectively produce global variables that can be passed around

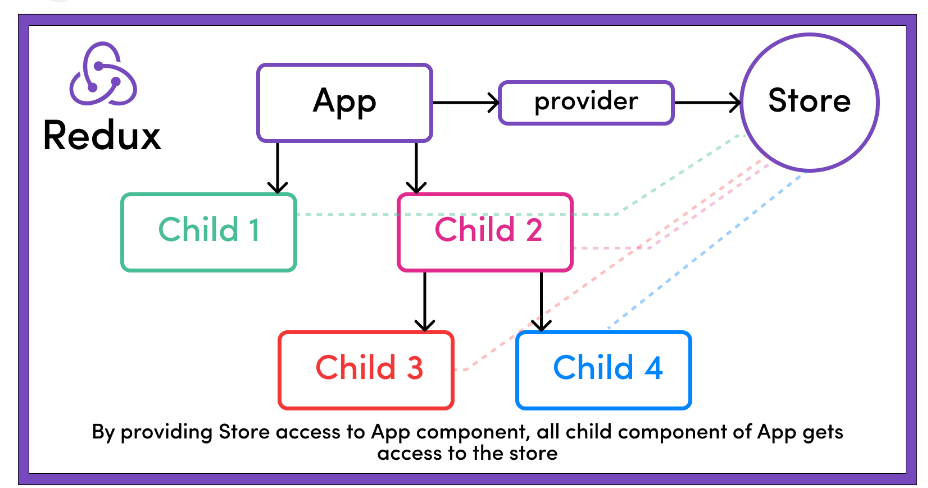
***Hook:*** useContext

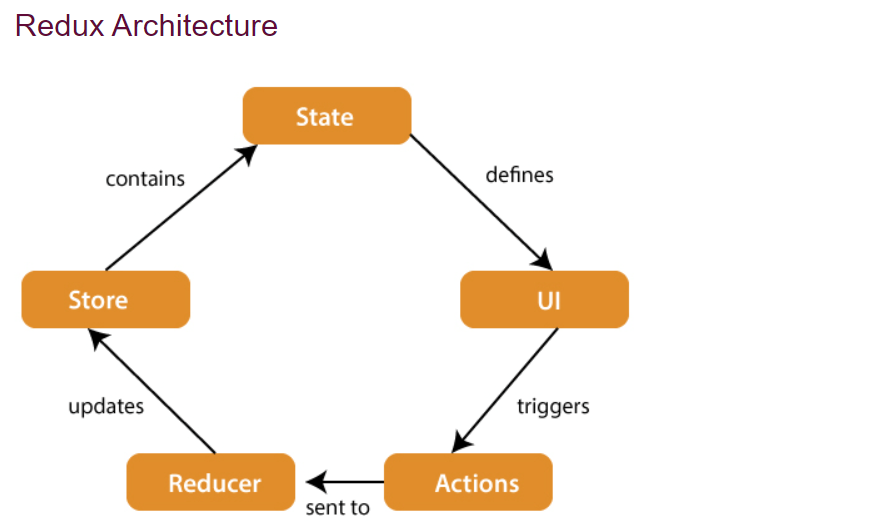
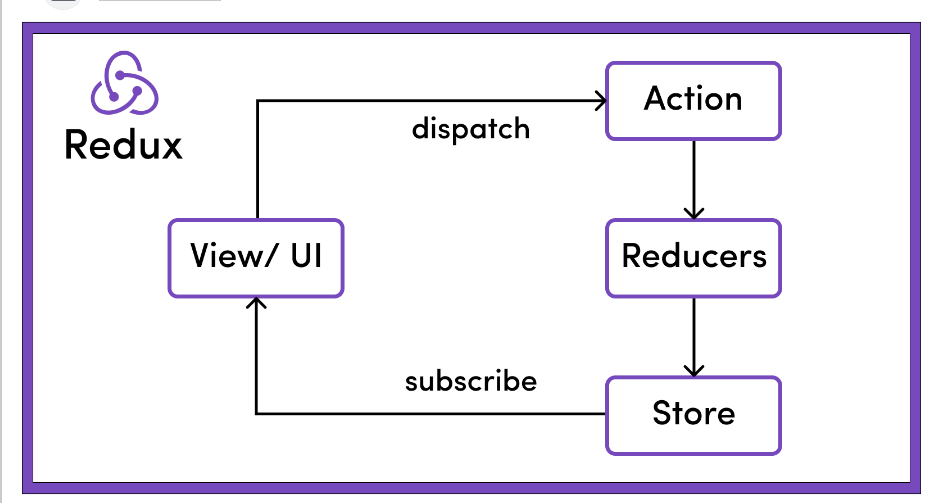
<https://stackblitz.com/edit/react-khhprn?file=src%2Findex.js,src%2FthemeContext.js,src%2FHeader.js,src%2FApp.js,src%2FButton.js>

**Redux –**

Redux is an open-source JavaScript library for managing and centralizing application state. It is most commonly used with libraries such as React or Angular for building user interfaces

Context api vs Red





{

“profileData”: {“name”:””, “avatar”:’/path’},

Login: {}

}

Redux is an open-source JavaScript library for managing and centralizing application state

 It allows React components to read data from a Redux Store, and dispatch **Actions** to the **Store** to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple.

It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component. Q

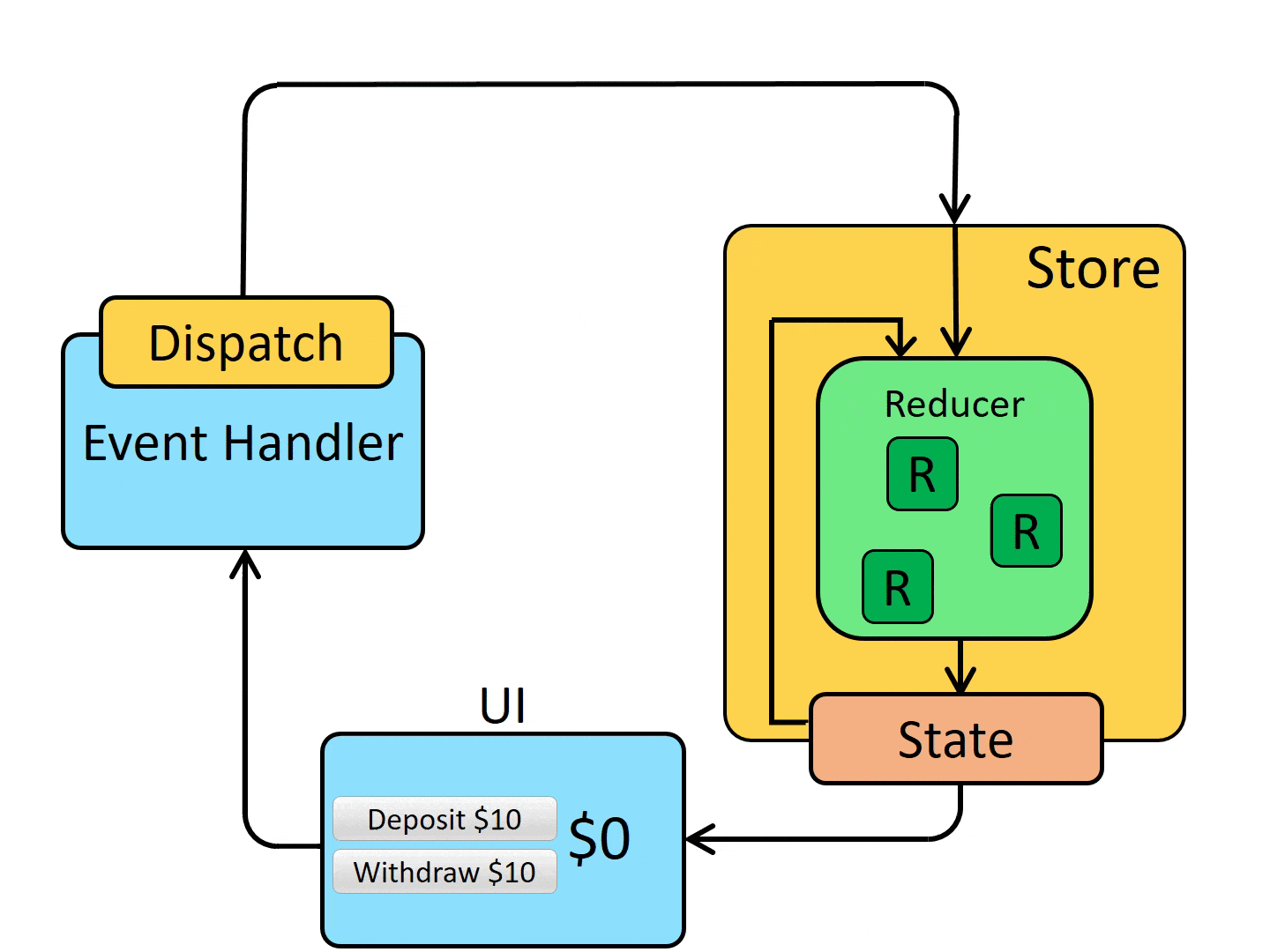
**STORE:** A Store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch(action) function. It is like a brain responsible for all moving parts in Redux.

**ACTION:** Action is sent or dispatched from the view which are payloads that can be read by Reducers. It is a pure object created to store the information of the user's event. It includes information such as type of action, time of occurrence, location of occurrence, its coordinates, and which state it aims to change.

**REDUCER:** Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.

<https://stackblitz.com/edit/easy-react-redux-example-5w6szr?file=components%2Fcounter.js,store%2Factions%2FcounterActions.js,store%2Freducers%2FcurrencyReducer.js,store%2Findex.js,Hello.js>

1. **Redux**: Redux is a predictable state container for JavaScript apps. It provides a centralized store that holds the entire state of your application, and the state can only be modified by emitting an action. Redux works with any UI layer, and it's not tied to React specifically. It's commonly used with React, but you can use it with Angular, Vue.js, or even vanilla JavaScript.
2. **react-redux**: React Redux is the official Redux binding for React. It provides React bindings for Redux, allowing React components to interact with the Redux store. It provides a set of APIs that enable you to connect your React components to the Redux store, access the state, and dispatch actions. React Redux makes it easier to integrate Redux with React applications by providing higher-order components and hooks like **connect()** and **useSelector()**.



# Performance/optimization

# useMemo =

# useCallback =

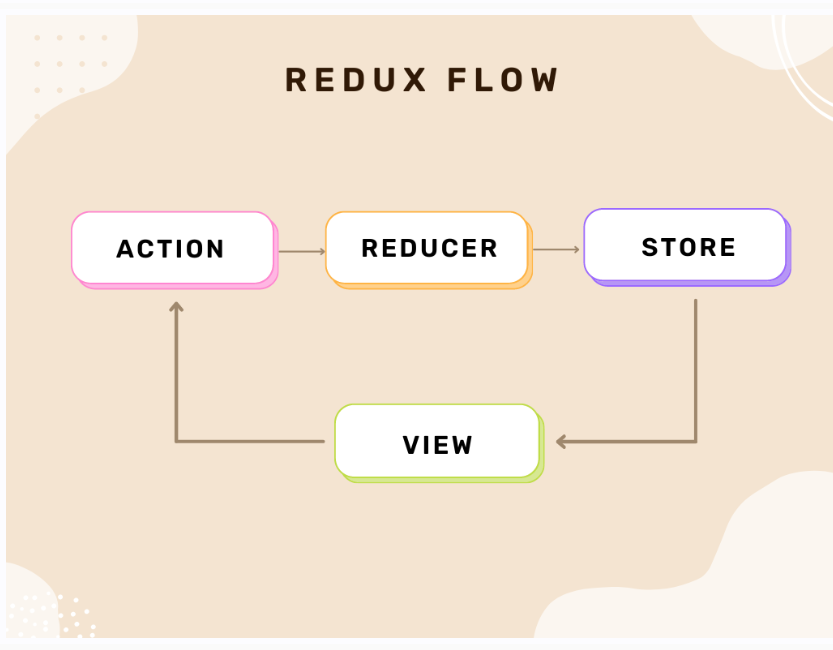
useCallback is a React Hook that lets you cache a function definition between re-renders.

<https://stackblitz.com/edit/usecallback-emhpdp?file=index.js,button.js>

<https://codesandbox.io/p/sandbox/react-usememo-d5vqy?file=%2Fsrc%2Findex.js%3A11%2C11-11%2C53>

Redux Thunk

Problem - [https://stackblitz.com/edit/react-department-employees-redux-thunk?file=in dex.js](https://stackblitz.com/edit/react-department-employees-redux-thunk?file=in%20%20%09dex.js)



Context api vs Redux

The difference between Context API and Redux lies in how they handle state changes. Redux takes a centralized approach, where state changes are managed in a central store. On the other hand, Context API deals with state changes on a components level, as they happen within each component.

https://reactchallenges.live/dashboard/overview

import React, { useEffect, useState } from 'react' const App = () => { const [click, setClick] = useState(0); const [counter, setCounter] = useState(100); useEffect(()=>{ setClick(click +1) }, [counter]) const counterFn = ()=>{ setCounter(counter+1) } return ( <div> <h1>click count {click} </h1> <h1>click count {counter} </h1> <button onClick={counterFn}>Click ME</button> </div> ) } export default App

Routing

[**https://stackblitz.com/edit/react-7n42ev?file=src%2FApp.js**](https://stackblitz.com/edit/react-7n42ev?file=src%2FApp.js)

[**https://dummy.restapiexample.com/**](https://dummy.restapiexample.com/)

**axios**

[**https://stackblitz.com/edit/react-zag2oi?file=src%2FApp.js**](https://stackblitz.com/edit/react-zag2oi?file=src%2FApp.js)

**npm install** [**react-redux@1.0**](mailto:react-redux@1.0)

**Class components**

[**https://stackblitz.com/edit/demo-react-class-component?file=Counter.js,index.js**](https://stackblitz.com/edit/demo-react-class-component?file=Counter.js,index.js)

**JS**

**//promise**

**// const sample = "ss"**

**// const sample2 = "ss"**

**// var i = 0;**

**// setTimeout(function() {**

**// console.log("Hiiiiiiii")//calculation**

**// i = 10;**

**// }, 3000);**

**// console.log("Finish",i)**

**function primiseFn() {**

**return new Promise((resolve, reject) => {**

**setTimeout(() => {**

**reject(5)**

**}, 1500)**

**})**

**}**

**primiseFn().then((resp)=>{**

**console.log(resp)**

**}).catch((err)=>{**

**console.log(err, 'err')**

**});**

**// //es6 - 2015 release**

**// // higher order function => map filter reduce**

**// // arrow function -**

**// // let, const**

**// // function sample() {**

**// // // function defition**

**// // }**

**// // const sample = (param) => {**

**// // console.log(sample)**

**// // }**

**// // sample("test");**

**// function parentFn(param1) {**

**// let i=0;**

**// while(i < 10) {**

**// if(i==9) {**

**// param1();**

**// }**

**// i = i+1;**

**// }**

**// }**

**// const passingFn = () => {**

**// console.log("Hi")**

**// }**

**// parentFn(passingFn)**

**// //map(param) {**

**// }**

**// // [].map((element)=>{})**

**//Default Parameters in ES6**

**// const sample = (age = 10, address = "sssss") => {**

**// if(age) {**

**// //age update apicall**

**// console.log(age)**

**// }**

**// if(address) {**

**// //address update apicall**

**// console.log(address)**

**// }**

**// }**

**// //age update button click**

**// sample(undefined)**

**//2-Template Literals**

**// const firtName = "manoj";**

**// const lastName = "mohanan"**

**// const genter = 'm'**

**// console.log(genter === 'm'? 'MR '+firtName+" " + lastName:'MS' )**

**// console.log(`Name:${ genter === 'm'?'MR':'MS'} ${firtName} ${lastName}`)**

**//Multi-line Strings in ES6**

**// const sample = `Hi,**

**// how are you**

**// are you fine`;**

**// console.log(sample)**

**// //1-default parameter**

**// Destructuring Assignment in ES6**

**const carObj = { name:"nexon", brand: 'tata', prize: 100, color:'red', varient:'xz' };**

**const nameArray = ["nexon", "tiago", "altroz", "audiz"]**

**const [car1, car2, asd] = nameArray;**

**// const carName = carObj.name;**

**// const carbrand = carObj.brand;**

**// const carprize = carObj.prize;**

**const {name, , prize, varient, brand} = carObj**

**console.log(`${brand}`);**

**// spread opperator // a feature that lets you break down an object or array into individual elements or properties // const firstArray = [1,2,3] // const secArray = [4,5,6] // const newarrray = [...firstArray, ...secArray] // console.log(newarrray) // const firstArray = {firstname: 'manoj'} // const secArray = {secName: 'mohan'} // {firstname: 'manoj', secName: 'mohan'} // const newarrray = {...firstArray, ...secArray} // console.log(newarrray) // const firstArray = "testsample**

**keep**

**Pin message**

**// const secArray = {secName: 'mohan'} // {firstname: 'manoj', secName: 'mohan'} // const newarrray = {...firstArray, ...secArray} // console.log(newarrray) // const firstArray = "testsample"; // console.log([...firstArray]) let sample = "Hi how are you "; // iH woh era uoy const a = sample.includes("Hi")**

**const products = [**

**{ id: 1, name: 'Laptop', category: 'Electronics', price: 1500, rating: 4.5 },**

**{ id: 2, name: 'Phone', category: 'Electronics', price: 800, rating: 4.7 },**

**{ id: 3, name: 'Shoes', category: 'Fashion', price: 120, rating: 4.2 },**

**{ id: 4, name: 'T-shirt', category: 'Fashion', price: 25, rating: 4.0 },**

**{ id: 5, name: 'Fridge', category: 'Home Appliance', price: 1000, rating: 4.6 },**

**{ id: 6, name: 'Washing Machine', category: 'Home Appliance', price: 850, rating: 4.3 },**

**];**

**//{}**

**const groupedProducts = products.reduce((acc, product) => {**

**if(!acc[product.category]) {**

**acc[product.category] = []**

**}**

**acc[product.category].push(product)**

**return acc**

**}, {});**

**console.log(Object.keys(groupedProducts));**

**console.log(groupedProducts);**