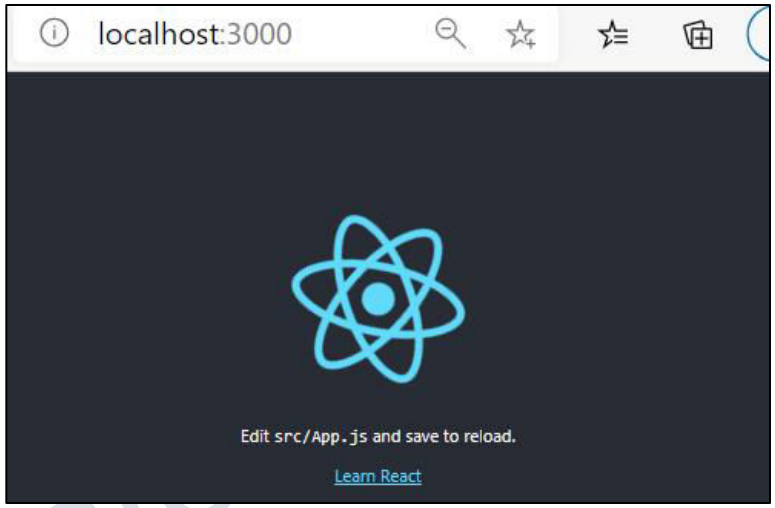
REACT.JS

JavaScript library for building user interfaces

Requirements

- npx create-react-app my-app
- cd my-app
- npm start

```
E:\FreelanceTrg\ReactJS\Demo\my-app>npm start

> my-app@0.1.0 start E:\FreelanceTrg\ReactJS\Demo\my-app
> react-scripts start

i ｢wds｣: Project is running at http://192.168.1.18/
i ｢wds｣: webpack output is served from
i ｢wds｣: Content not from webpack is served from E:\Freelar
i ｢wds｣: 404s will fallback to /
Starting the development server...
Compiled successfully!

You can now view my-app in the browser.
```



Creating a functional component

- Components are the essential building blocks of any app created with React
  - A single app most often consists of many components.
  - A component is in essence, a piece of the UI - splitting the user interface into reusable and independent parts, each of which can be processed separately.
  - Component is independent and reusable bit of code; It's an encapsulated piece of logic.

```
App.js > ...
import React from 'react';
import ExpenseItem from './components/ExpenseItem';

function App() {
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <ExpenseItem />
    </div>
  );
}
export default App;
```

```
const ExpenseItem = () => {
   return <h2>Expense Item</h2>
}
export default ExpenseItem;
```


```
function App() {
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <ExpenseItem />
      <ExpenseItem />
      <ExpenseItem />
    </div>
  );
}
```

Welcome to React!

Expense Item

Expense Item

Expense Item

Another example

OPEN EDITORS
MY-APP
- node_modules
- public
- src
  - Person
    - Person.js U

src > Person > Person.js > default

```
1  import React from 'react';
2
3  const person = () => {
4      return <p>Hi Person</p>
5  }
6  export default person;
```

```
import React, {Component} from 'react'
;
import ./App.css';
import Person from ./Person/Person';
function App() {
  return (
    <div className="App">
      <h1> Hi, welcome to React</h1>
        <Person />
    </div>
  );
}
export default App;
```

```
return (
  <div className="App">
    <h1> Hi, welcome to React</h1>
    <Person />
    <Person />
    <Person />
  </div>
);
```

Hi, welcome to React

Hi Person

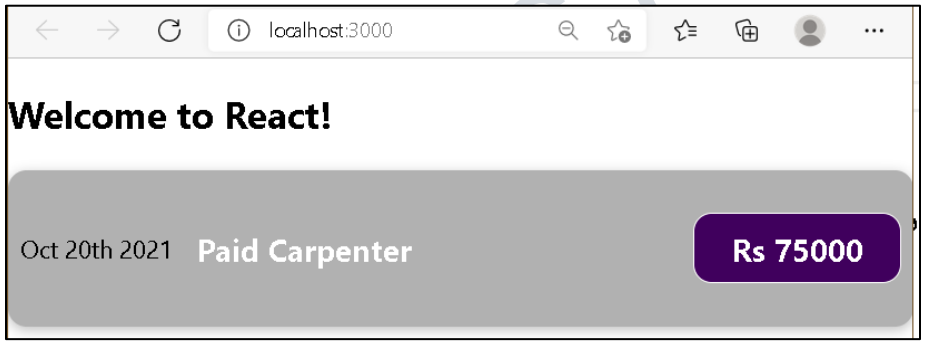
Elements Console Sources Ne

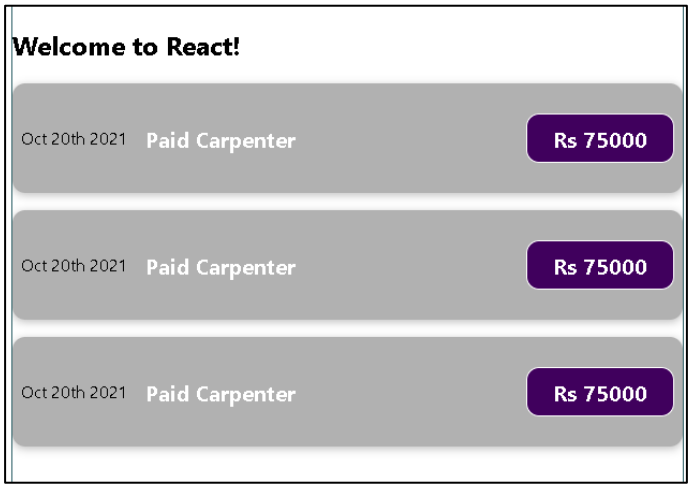
```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
…▼<body> == $0
  <noscript>You need to enable JavaSc
  ▼<div id="root">
    ▼<div class="App">
      <h1> Hi, welcome to React</h1>
      <p>Hi Person</p>
    </div>
  </div>
```

Making our functional component more complex

```
import "./ExpenseItem.css"

const ExpenseItem = () => {
   return (
       <div className="expense-item">
           <div>Oct 20th 2021</div>
           <div className="expense-item__description">
               <h2>Paid Carpenter</h2>
               <p className="expense-item__price">Rs 75000</p>
           </div>
       </div>
   )
}
export default ExpenseItem;
```

```
.expense-item {
    display: flex;
    justify-content: space-between;
    align-items: center;
    box-shadow: 0 2px 8px rgba(0,
    padding: 0.5rem;
    margin: 1rem 0;
    border-radius: 12px;
    background-color: #4b4b4b6e;
}

.expense-item__description {…
}

.expense-item h2 {…
}

.expense-item__price {…
}
```

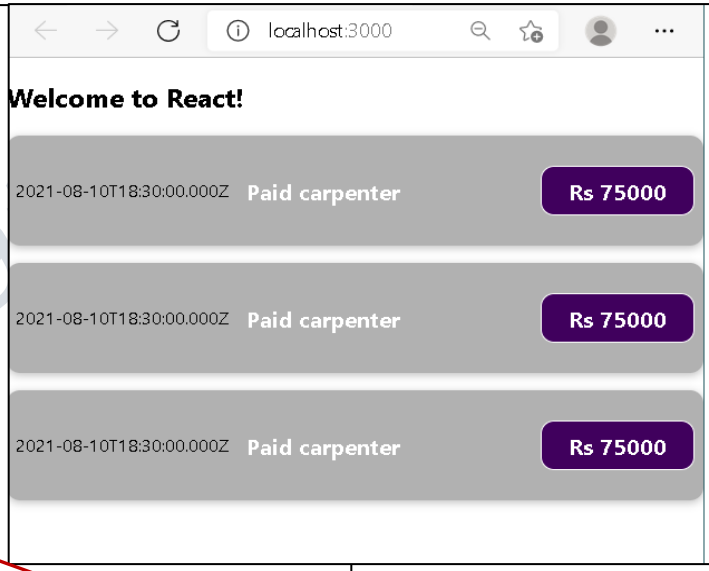




Outputting dynamic content

- If we have some dynamic content in our jsx part which we want to run as javaScript code and not interpret as text, we have to wrap it in single curly braces.

```
import "./ExpenseItem.css"

const ExpenseItem = () => {

    const expDate = new Date(2021, 7, 11);
    const expTitle = "Paid carpenter";
    const expAmount = 75000

   return (
       <div className="expense-item">

           {/* single and multiline comments in JSX */}

           <div>{expDate.toISOString()}</div>
           <div className="expense-item__description">
               <h2>{expTitle}</h2>
               <p className="expense-item__price">Rs {expAmount}</p>
           </div>
       </div>
   )
}
export default ExpenseItem;
```



Comments in JSX

Outputting dynamic content – another example

```
//Person.js
const person = () => {
  return <p>Hi Person i am {Math.floor(Math.random() * 30)} years old</p>
}
export default person;
```

```
function App() {
  return (
      <div className="App">
        <h1> Hi, welcome to React</h1>
        <Person />
        <Person />
        <Person />
      </div>
    );
  }
}
```

Hi, welcome to React

Hi Person i am 27 years old

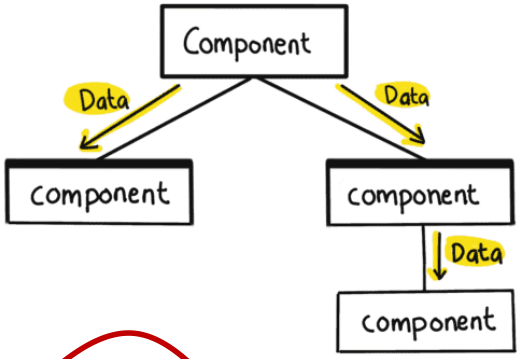
Hi Person i am 29 years old

Hi Person i am 12 years old

Passing data via 'Props'

- "**Props**" stands for properties.
  - It is a special keyword in React used for passing data from one component to another.
  - Props are arguments passed into React components.
  - props are read-only. So, the data coming from a parent component can't be changed by the child component.
  - Props are passed to components via HTML attributes.
  - Props can be used to pass any kind of data such as: String, Array, Integer, Boolean, Objects or, Functions

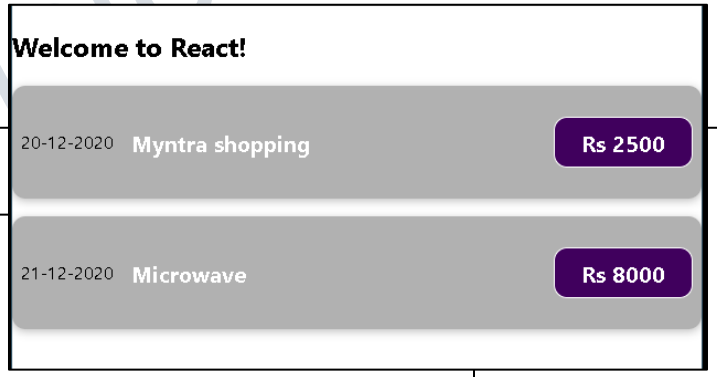
```
import HelloComponent from "./HelloComponent";
const MessageComponent = () => {
    return(
        <div>
            <HelloComponent name="Shrilata" />
        </div>
    );
}
export default MessageComponent;
```




```
const HelloComponent = (props) => {
    return (<h3>Hello, welcome {props.name}</h3>)
}
export default HelloComponent;
```

Passing data via 'Props'

```
function App() {
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <ExpenseItem expDate="20-12-2020" expTitle="Myntra shopping" expAmount="2500"/>
      <ExpenseItem expDate="21-12-2020" expTitle="Microwave" expAmount="8000"/>
    </div>
  );
}
export default App;
```




```
const ExpenseItem = (props) => {
   return (
       <div className="expense-item">
           <div>{props.expDate}</div>
           <div className="expense-item__description">
               <h2>{props.expTitle}</h2>
               <p className="expense-item__price">Rs {props.expAmount}</p>
           </div>
       </div>
   )
}
export default ExpenseItem;
```

Working with props

```
function App() {
    return (
      <div className="App">
        <h1> Hi, welcome to React</h1>
        <Person name="Shri" age="20"/>
        <Person name="Soha" age="23”>Hobbies : Coding</Person>
        <Person name="sandeep" age="45"/>
      </div>
    );
```

Hi, welcome to React

Hi i am Shri and i am 20 years old

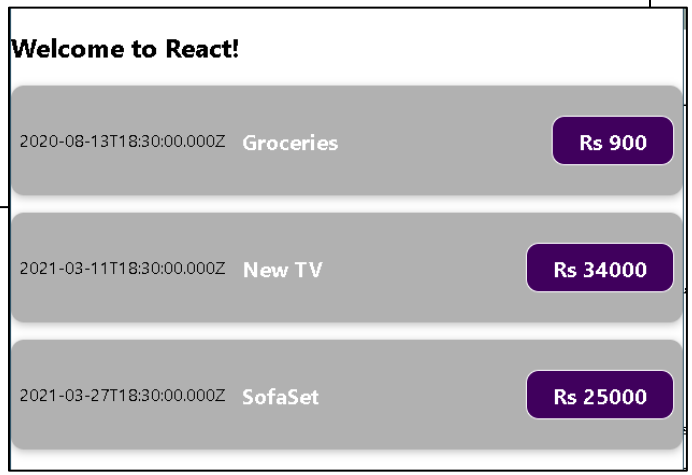
Hi i am Soha and i am 23 years old

Hi i am sandeep and i am 45 years old

```
//Person.js
import React from 'react';

const person = (props) => {
   return <p>Hi i am {props.name} and i am {props.age} years old</p>
}
export default person;
```

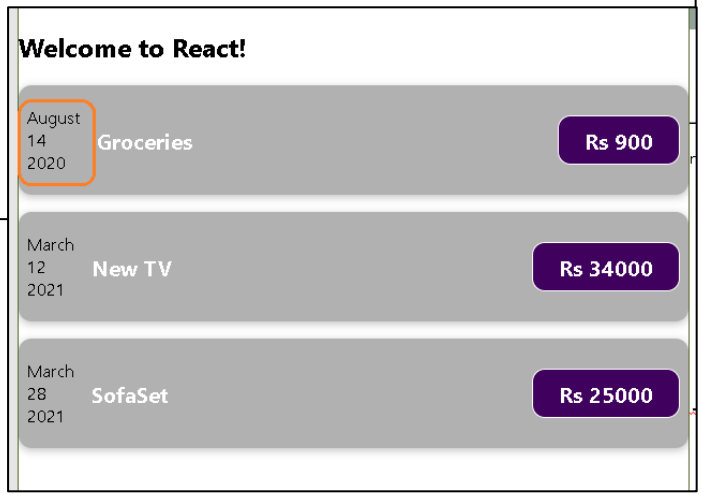
Working with props

```
function App() {
  const expenses = [
    {title: 'Groceries’, amount: 900, date: new Date(2020, 7, 14)},
    { title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },
    { title: 'SofaSet', amount: 25000, date: new Date(2021, 2, 28),
    }
  ];
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <ExpenseItem expDate={expenses[0].date} expTitle={expenses[0].title}
                   expAmount={expenses[0].amount}/>
      <ExpenseItem expDate={expenses[1].date} expTitle={expenses[1].title}
                   expAmount={expenses[1].amount}/>
      <ExpenseItem expDate={expenses[2].date} expTitle={expenses[2].title}
                   expAmount={expenses[2].amount}/>
    </div>
  );
}
export default App;
```



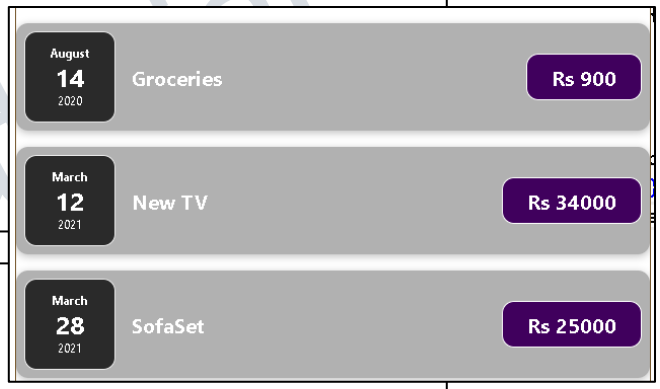
“Javascript” in components

```
const ExpenseItem = (props) => {
    const month = props.expDate.toLocaleString('en-US', {month: 'long'});
    const day = props.expDate.toLocaleString('en-US', {day: '2-digit'});
    const year = props.expDate.getFullYear();

    return (
       <div className="expense-item">
           <div>
               <div>{month}</div>
               <div>{day}</div>
               <div>{year}</div>
           </div>
           <div className="expense-item__description">
               <h2>{props.expTitle}</h2>
               <p className="expense-item__price">Rs {props.expAmount}</p>
           </div>
       </div>
   )
}
export default ExpenseItem;
```

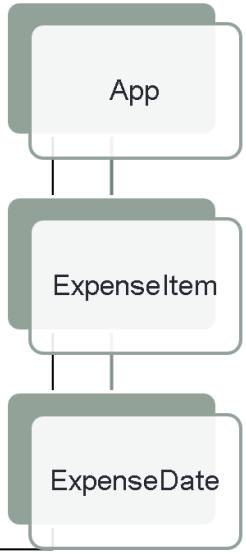


Splitting components further

```
import ExpenseDate from "./ExpenseDate";
const ExpenseItem = (props) => {
   return (
       <div className="expense-item">
           <ExpenseDate date={props.expDate}/>
           <div className="expense-item__description">
               <h2>{props.expTitle}</h2>
               <p className="expense-item__price">Rs {props.expAmount}</p>
           </div>
       </div>
   )
}
export default ExpenseItem;
```



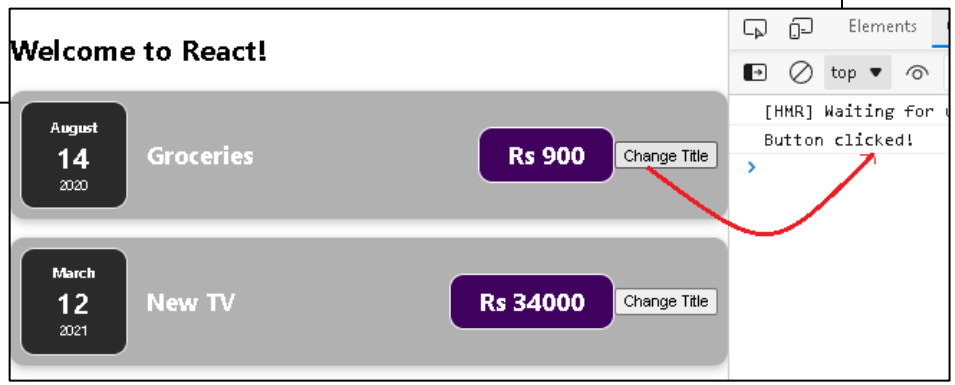

```
import "./ExpenseDate.css"

const ExpenseDate = (props) => {
    const month = props.date.toLocaleString('en-US', {month: 'long'});
    const day = props.date.toLocaleString('en-US', {day: '2-digit'});
    const year = props.date.getFullYear();
    return (
        <div className="expense-date">
            <div className="expense-date__month">{month}</div>
            <div className="expense-date__day">{day}</div>
            <div className="expense-date__year">{year}</div>
        </div>
    );
}
export default ExpenseDate;
```



Listening to events and working with event handlers

```
const ExpenseItem = (props) => {

    let btnHandler = () => {
        console.log("Button clicked!")
    }

    return (
        <div className="expense-item">
            <ExpenseDate date={props.expDate}/>
            <div className="expense-item__description">
                <h2>{props.expTitle}</h2>
                <p className="expense-item__price">Rs {props.expAmount}</p>
            </div>
            <button onClick={btnHandler}>Change Title</button>
        </div>
    )
}
export default ExpenseItem;
```

No parenthesis ()



React State and React Hooks

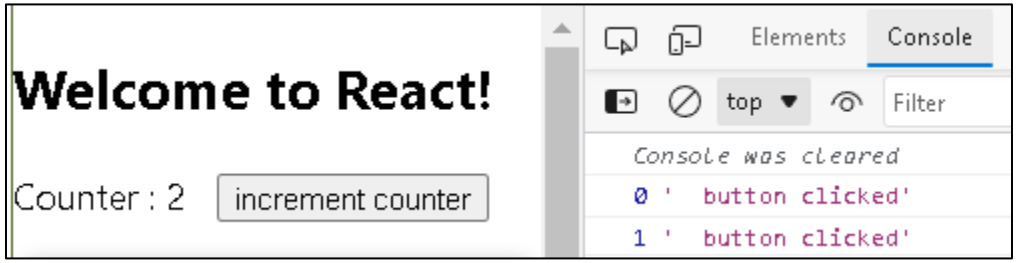
- The state is a built-in React object that is used to contain data or information about the component.
  - A component's state can change over time; whenever it changes, the component re-renders.
  - The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.
- A component with state is known as stateful component.
- State allows us to create components that are dynamic and interactive.
  - State is private, it must not be manipulated from the outside.

- Hooks allow us to "hook" into React features such as state and lifecycle methods
  - React Hooks are special functions provided by React to handle a specific functionality inside a React functional component.
  - React provides *useState()* function to manage state in a functional component.

- You must import Hooks from react
  - `import React, { useState } from "react";`
  - Here – useState() is a Hook to keep track of the application state.

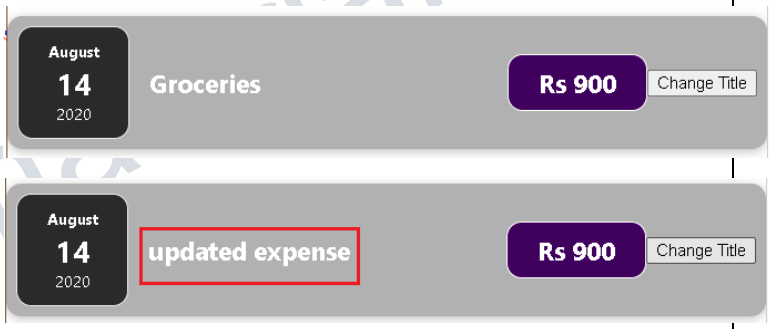
Working with “state” in functional component

```
import React, {useState} from 'react';

const UseStateComponent = () => {
    const [counter, setCounter] = useState(0);  //hooks go here

    const btnHandler = () => {
        setCounter(counter+1);
        console.log(counter, "  button clicked")
     }
    return(
           <div>
              Counter : {counter}   
              <button onClick={btnHandler}>increment counter</button>
           </div>
        );
}
export default UseStateComponent;
```

Working with “state” in functional component

```
import React, {useState} from 'react'

const ExpenseItem = (props) => {

    const [title, setTitle] = useState(props.expTitle);

    let btnHandler = () => {
        setTitle("updated expense")
       console.log("Button clicked!")
   }

   return (
       <div className="expense-item">
           <ExpenseDate date={props.expDate}/>
           <div className="expense-item__description">
               <h2>{title}</h2>
               <p className="expense-item__price">Rs {props.expAmount}</p>
           </div>
           <button onClick={btnHandler}>Change Title</button>
       </div>
   )
}
export default ExpenseItem;
```



Simple example : props + state

```
import React, {useState} from 'react'
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
   const [uname, setUname] = useState('Shrilata')
   const [email, setEmail] = useState('shrilata@gmail.com')

   return(
       <ChildComponent uname={uname} email={email} />
   );
}
export default ParentComponent;
```

```
function App() {
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <ParentComponent />
...
```

```
const ChildComponent = (props) => {
    return(
        <div>
            <div>Name : {props.uname}</div>
            <div>Email : {props.email}</div>
        </div>
    );
}
export default ChildComponent;
```

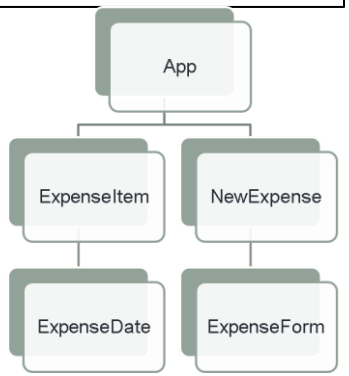
Welcome to React!

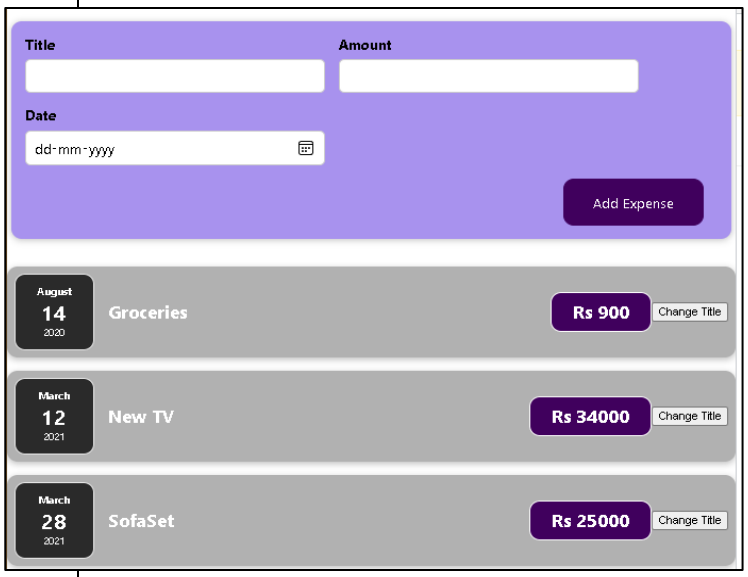
Name : Shrilata
Email : shrilata@gmail.com

Adding form inputs

```
import "./ExpenseForm.css"
const ExpenseForm = () => {
    return(
      <form>
          <div className="new-expense__controls">
             <div className="new-expense__control">
                 <label>Title</label>
                 <input />
             </div>
             <div className="new-expense__control">
                 <label>Amount</label>
                 <input type="number"/>
             </div>
             <div className="new-expense__control">
                 <label>Date</label>
                 <input type="date" min="2019-01-01"
                        max="2022-12-31" />
             </div>
          </div>
          <div className="new-expense__actions">
              <button type="submit">Add Expense</button
          </div>
      </form>
    );
}
export default ExpenseForm;
```

```
const NewExpense = () => {
    return(
        <div className="new-expense">
            <ExpenseForm />
        </div>
    );
}
export default  NewExpense;
```



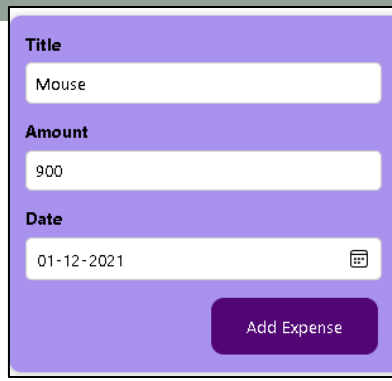


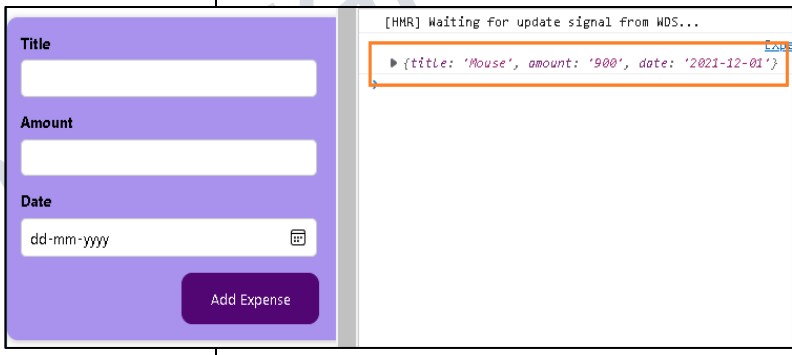
Storing input into state – working with multiple states

```
import React, {useState} from 'react';
const ExpenseForm = () => {

    const [inputTitle, setInputTitle] = useState('')
    const [inputAmount, setInputAmount] = useState('')
    const [inputDate, setInputDate] = useState('')

    const titleChangeHandler = (event) => {
        setInputTitle(event.target.value)
    }
    const amountChangeHandler = (event) => {
        setInputAmount(event.target.value)
    }
    const dateChangeHandler = (event) => {
        setInputDate(event.target.value)
    }
    return(
      <form>
          <div className="new-expense__controls">
            <div className="new-expense__control">
                <label>Title</label> <input onChange={titleChangeHandler}/>
            </div>
            <div className="new-expense__control">
                <label>Amount</label>
                <input type="number" onChange={amountChangeHandler}/>
            </div> ...
```

Form submission – extracting data, 2-way binding

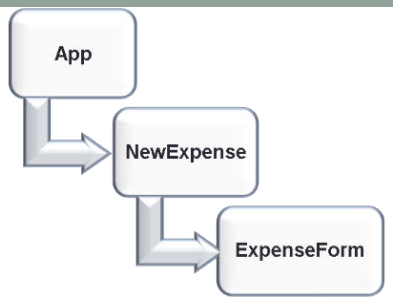
```
const ExpenseForm = () => {

    const [inputTitle, setInputTitle] = useState('')
    const [inputAmount, setInputAmount] = useState('')
    const [inputDate, setInputDate] = useState('')

    const titleChangeHandler = (event) => {earlier-code}
    const amountChangeHandler = (event) => {earlier-code}
    const dateChangeHandler = (event) => {...}

    const submitHandler = (event) => {
        event.preventDefault();
        const expenseData = {
            title:inputTitle,
            amount:inputAmount,
            date:inputDate
        }
        console.log(expenseData)
        setInputAmount('')
        setInputDate('')
        setInputTitle('')
    }
```



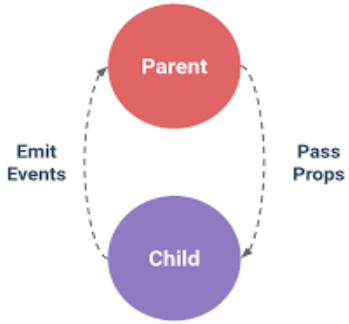



```
return(
        <form onSubmit={submitHandler}>
            <div className="new-expense__controls">
                <div className="new-expense__control">
                    <label>Title</label>
                    <input value={inputTitle}
                           onChange={titleChangeHandler}/>
                </div>
                ...
        </form>
    );
}
```

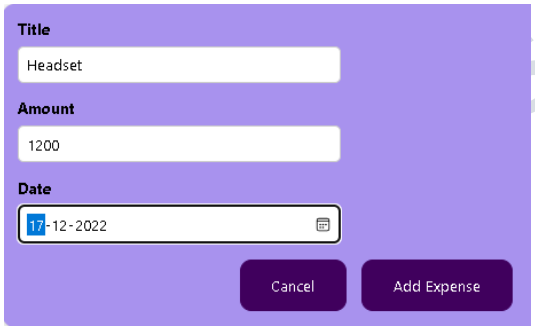
Passing data from child to parent component




```
const NewExpense = () => {
    const saveExpenseDataHandler = (inputExpenseData) => {
        const expenseData = {
            ...inputExpenseData,
            id:Math.random().toString()
        }
        console.log("In NewExpense ",expenseData)
    }
    return(
        <div className="new-expense">
            <ExpenseForm onSaveExpenseData={saveExpenseDataHandler}/>
        </div>
    );
}
export default  NewExpense;
```



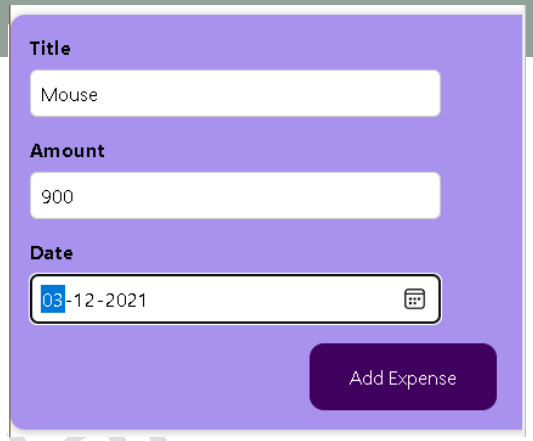

```
const ExpenseForm = (props) => {
    ...
    const submitHandler = (event) => {
        event.preventDefault();
        const expenseData = {
            title:inputTitle,
            amount:inputAmount,
            date: new Date(inputDate)
        }
        //console.log(expenseData)
        props.onSaveExpenseData(expenseData);
        ...
    }
```




```
In NewExpense                                          NewExpense.js:21
  ▶ {title: 'Headset', amount: '1200', date: Sat Dec 17 2022 05:30:00 GMT+0530
    (India Standard Time), id: '0.4603800892834633'}
```

Lifting state up

```
function App() {
  const expenses = [...];

  const addExpenseHandler = expense => {
    console.log("In App component ", expense)
  }
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <NewExpense onAddExpense={addExpenseHandler} />
      ...
  );
}
export default App;
```




```
In App component                                   App.js:20
▶ {title: 'Mouse', amount: '900', date: '2021-12-03',
  id: '0.25759054649698765'}
```

```
const NewExpense = (props) => {

    const saveExpenseDataHandler = (inputExpenseData) => {
        const expenseData = {
            ...inputExpenseData,
            id:Math.random().toString()
        }
        //console.log("In NewExpense ",expenseData)
        props.onAddExpense(expenseData)
    }
    return(...);
}
export default  NewExpense;
```

- We have seen props are passed down multiple component levels.
- That's how state is shared vertically in your application.
- Often there will be a need to share state between different components.
- The common approach to share state between two components is to move the state to common parent of the two components. This approach is called as lifting state up in React.js

Working with lists

```
const SimpleListComponent = () => {
   const nums = [1,2,3,4,5]
   const updatedNums = nums.map((num)=>{
    return <li>{num*num}</li>;
   });

   const items =[{name:'Item-1'}, {name:'Item-2'}]
   const updatedItems = items.map(item => (
      <p>{item.name}</p>
    ))

   const students = [{name:"Anita",rollno:101},
                     {name:"Sunita",rollno:102},
                     {name:"Kavita",rollno:103}]
   const updatedStudents = students.map(student => (
      <tr>
         <td>{student.name}</td><td>{student.rollno}</td>
      </tr>
   ))
return(
    <div>
        <h2> Numbers Arr: </h2> {updatedNums}
        <h2> Items : </h2>   {updatedItems}
        <h2> Students Data</h2>
        <table border="1">
          <tr><th>Name</th><th>Roll no</th></tr>
          {updatedStudents}</table>
    </div>
);}
export default SimpleListComponent
```

Numbers arr :

- 1
- 4
- 9
- 16
- 25

Items :

Item-1

Item-2

Student Data

Name	Roll no
Anita	101
Sunita	102
Kavita	103

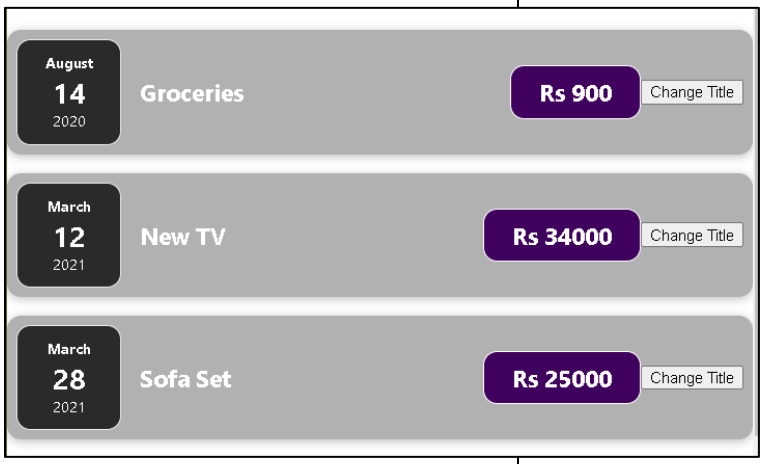
Working with the expenses list hardcoded

```
  const expenses = [
    { title: 'Groceries’, amount: 900, date: new Date(2020, 7, 14)},
    { title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },
    { title: 'SofaSet', amount: 25000, date: new Date(2021, 2, 28)}
  ];
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <ExpenseItem
           expDate={expenses[0].date}
           expTitle={expenses[0].title}
           expAmount={expenses[0].amount}
      />
      <ExpenseItem
           expDate={expenses[1].date}
           expTitle={expenses[1].title}
           expAmount={expenses[1].amount}
      />
      <ExpenseItem
           expDate={expenses[2].date}
           expTitle={expenses[2].title}
           expAmount={expenses[2].amount}
      />
    </div>
  );
}
```

Looping thru the expenses list via map()

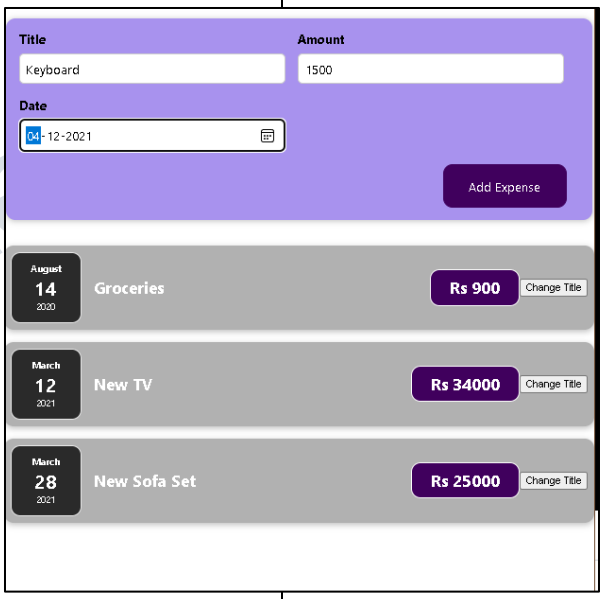
```
function App() {
  const expenses = [
    { title: 'Groceries', amount: 900, date: new Date(2020, 7, 14)},
    { title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },
    { title: 'Sofa Set', amount: 25000, date: new Date(2021, 2, 28)}
  ];
  return (
    <div className="App">
      <h2>Welcome to React!</h2>

       {expenses.map(expense => {
          return <ExpenseItem
                   expDate={expense.date}
                   expTitle={expense.title}
                   expAmount={expense.amount}
                 />
          })}
       {/* <ExpenseItem
          expDate={expenses[0].date}
          expTitle={expenses[0].title}
          expAmount={expenses[0].amount}
      />  ... */}
    </div>
  );
}
export default App;
```

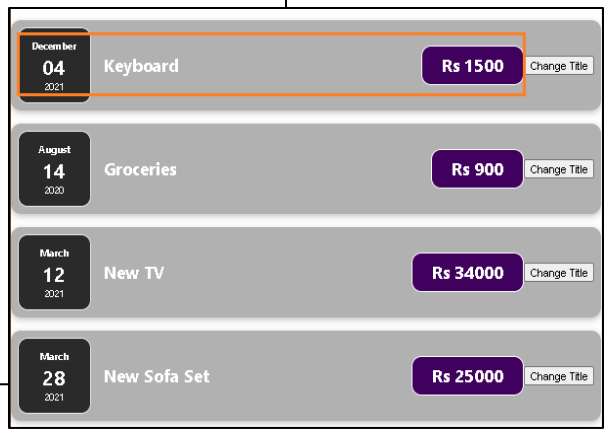
```
{expenses.map(expense =>
         (<ExpenseItem
             expDate={expense.date}
             expTitle={expense.title}
             expAmount={expense.amount}
          />))
}
```



Using stateful lists

```
const DUMMY_EXP = [
  { title: 'Groceries', amount: 900, date: new Date(2020,
  { title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },
  { title: 'New Sofa Set', amount: 25000, date: new Date(2021, 2, 28)}
];
function App() {
  const [expenses, setExpenses] = useState(DUMMY_EXP)

  const addExpenseHandler = expense => {
     //console.log("In App component ", expense)
    setExpenses(prevArr => {return [expense, ...prevArr]})
  }
  return (
    <div className="App">
      <h2>Welcome to React!</h2>
      <NewExpense onAddExpense={addExpenseHandler} />

       {expenses.map(expense => (<ExpenseItem
                 expDate={expense.date}
                 expTitle={expense.title}
                 expAmount={expense.amount}
               />))
       }
    </div>
  );
}
export default App;
```





Lists and keys

```
Warning: Each child in a list should have a unique "key" prop.        index.js:1

Check the render method of `App`. See https://reactjs.org/link/warning-keys for more
information.
    at ExpenseItem (http://localhost:3000/static/js/main.chunk.js:685:19)
    at App (http://localhost:3000/static/js/main.chunk.js:184:89)
```

```
const DUMMY_EXP = [
  { id:101, title: 'Groceries', amount: 900, date: new Date(2020, 7, 14)},
  { id:102,title: 'New TV', amount: 34000, date: new Date(2021, 2, 12) },
  { id:103,title: 'New Sofa Set', amount: 25000, date: new Date(2021, 2, 28)}
];

function App() {

  const [expenses, setExpenses] = useState(DUMMY_EXP)
  ...
  return (
    <div className="App">
      ...
       {expenses.map(expense => (
                <ExpenseItem
                  key={expense.id}
                  expDate={expense.date}
                  expTitle={expense.title}
                  expAmount={expense.amount}
                />))
       }
...
```

Rendering content conditionally

- Conditional rendering means to render a specific HTML element or React component depending on a prop or state value.
  - Eg, based on some logic it can either return a list of items or a text that says "Sorry, the list is empty".

```
/*const users = [
    { id: '1', firstName: 'Shrilata', lastName: 'T' },
    { id: '2', firstName: 'Anita', lastName: 'Patil' },
  ];*/
  const users = []
  var users;

  function ListUsers() {
    return (
      <div>
        <List list={users} />
      </div>
    );
  }
 function List({ list }) {
    if (!list) {
      return null;
    }
    return (
      <ul>
        {list.map(item => (
          <Item key={item.id} item={item} />
        ))}
      </ul>
    );
  }
```

```
function Item({ item }) {
    return (
      <li>
        {item.firstName} {item.lastName}
      </li>
    );
  }
  export default ListUsers;
```

Hello Conditional Rendering

- Shrilata T
- Anita Patil

Rendering content conditionally : example-2

```
import ListBooks from "./ListBooks";
const BookInfo = () => {
    const books = [
        {id:101, name:"Core Java", author:"author1", price:400},
        {id:102, name:"Core Servlets", author:"author2", price:450},
        {id:103, name:"Core JSP", author:"author2", price:300},
        {id:104, name:"Spring Basics", author:"author3", price:500},
    ]
    return(
       <div>
         <ListBooks books={books} />
       </div>
    );
}
export default BookInfo;
```

Using conditions with logical && operator

```
import { useState } from "react";
  function ListBooks(props) {
    const [show, setShow] = useState(false)
    const [books, setBooks] = useState(props.books)

    const btnHandler = () => {
       setShow(!show);
    }
    return (
      <div>
        <h1>Book List</h1>
        <table border="1">
        {show && books.map(book =>
           (<tr>
               <td>{book.id}</td><td>{book.name}</td>
               <td>{book.author}</td><td>{book.price}</td>
           </tr>)
        ) }
        </table>
        <button onClick={btnHandler}>Toggle book list</button>
      </div>
    );
  }
  export default ListBooks;
```

Book List

Toggle book list

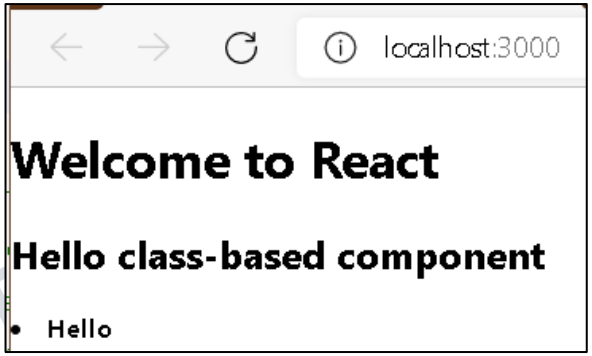
Book List

101	Core Java	author1	400
102	Core Servlets	author2	450
103	Core JSP	author2	300
104	Spring Basics	author3	500

Toggle book list

Class-based components : Examples

```
import React, {Component} from 'react';
class HelloComponent extends Component{
  render(){
     return (<h2>Hello class-based component</h2>)
  }
}
export default HelloComponent;
```




```
import './User.css';
import React, {Component} from 'react';

class User extends Component{
  render(){
      return <li className='user'>Hello User</li>
  }
};
export default User;
```

```
function App() {
  return (
    <div className="App">
      <h1> Welcome to React</h1>
      <HelloComponent />
      <User />
    </div>
  );
}
```

Class-based components : passing into props

```
import React, {Component} from 'react';
import User from './User'

const DUMMY_USERS = [
    { id: 'u1', name: 'Shrilata' },
    { id: 'u2', name: 'Soha' },
    { id: 'u3', name: 'Sia' },
  ];

class Users extends Component{
    render(){
        return(
        <div>
            <User name={DUMMY_USERS[0].name} />
            <User name={DUMMY_USERS[1].name} />
            <User name={DUMMY_USERS[2].name} />
        </div>
        );
    }
}
export default Users;
```

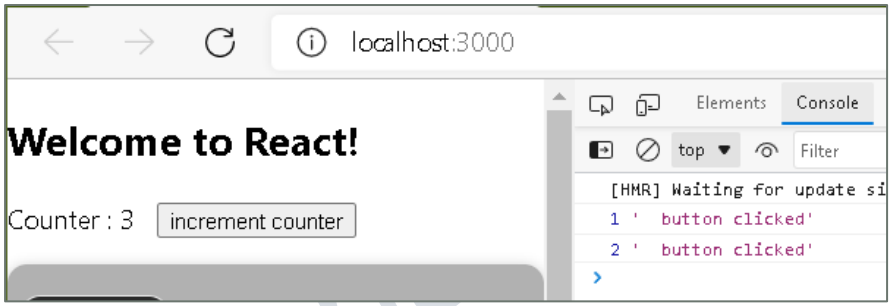
```
function App() {
  return (
    <div className="App">
      <h1> Welcome to React</h1>
      <Users />
    </div>
  );
}
```

- Hello Shrilata
- Hello Soha
- Hello Sia

```
import './User.css';
import React, {Component} from 'react';

class User extends Component{
  render(){
      return (<li className='user'>Hello {this.props.name}</li>);
  }
};
export default User;
```

React State : Class-based component

```
class App extends Component {
 render() {
   return (
     <div>
       <StatefulComponent />
     </div>
   )}
}
```




```
import React, {Component} from "react";
class StatefulComponent extends Component{
    state = {counter:1 }

    btnHandler = () => {
       this.setState({counter:this.state.counter+1})
       console.log(this.state.counter, "  button clicked")
    }
    render(){
        return(
           <div>
              Counter : {this.state.counter}   
              <button onClick={this.btnHandler}>increment counter</button>
           </div>
        );
    }
}
export default StatefulComponent;
```

```
constructor(){
  super();
  this.state = {counter:1 }
}
```

React multiple State : Example

```
import React, {Component} from 'react';
import './App.css';
import StatefulComponent from "./StatefulComponent/StatefulComponent";

class App extends Component {
 render() {
   return (
     <div>
       <StatefulComponent />
     </div>
   );
 }
}
export default App;
```

```
import React,{Component} from 'react';

class statefulComponent extends Component{
    state = {
        name: "Shrilata",
        email: "shrilata@gmail.com",
        address:"Pune"
     }
    render(){
        return(
          <div>
            <h3>Name : {this.state.name}</h3>
            <h3>Email : {this.state.email}</h3>
            <h3>Address : {this.state.address}</h3>
          </div>
        );
    }
}
export default statefulComponent;
```

Unlike useState, its so easy to group state together in class component

Name : Shrilata

Email : shrilata@gmail.com

Adress : Pune

State and props

```
class App extends Component {
  state = {
    persons:[
      {name:"Shri",age:20},
      {name:"Soha",age:23},
      {name:"Sandeep",age:30},
    ]
  }
  render() {
    return (
      <div className="App">
        <h1> Hi, welcome to React</h1>
        <Person name={this.state.persons[0].name} age={this.state.persons[0].age}/>
        <Person name={this.state.persons[1].name} age={this.state.persons[1].age} />
        <Person name={this.state.persons[2].name} age={this.state.persons[2].age}/>
      </div>
    );
  }
}
```

Hi, welcome to React

Switch name

Hi i am Shri and i am 20 years old

Hi i am Soha and i am 23 years old

Hobbies : Coding

Hi i am Sandeep and i am 30 years old

```
const person = (props) => {
return (
    <div>
       <p>Hi i am {props.name}
          and i am {props.age} years old</p>
    </div>
)
}
export default person;
```

Error Boundaries

- By default, if your application throws an error during rendering, React will remove its UI from the screen. A JavaScript error in a part of the UI shouldn't break the whole app.
- Error boundaries were introduced in React v16
  - you need to define a class component with either or both of the following lifecycle methods: getDerivedStateFromError() or componentDidCatch().
  - componentDidCatch(): This method is used for logging error information.

```
import React, {Component} from 'react';
class ErrorBoundary extends Component{
    state = {
        hasError:false,
        errorMessage:''
    }
   componentDidCatch = (error, info) => {
        this.setState({hasError:true, errorMessage:error});
   }
   render(){
     if(this.state.hasError)
        return <h1>{this.state.errorMessage}</h1>
     else
        return this.props.children
   }
}
export default ErrorBoundary;
```

Use it like this:

```
<ErrorBoundary>
   <Artists />
</ErrorBoundary>
```

```
import React from 'react'
function Artists({artistName}) {
  if (artistName === 'peruzzi') {
   throw new Error ('not performing tonight!')
   }
  return (
   <div>
    {artistName}
   </div>
  )
 }
export default Artists
```

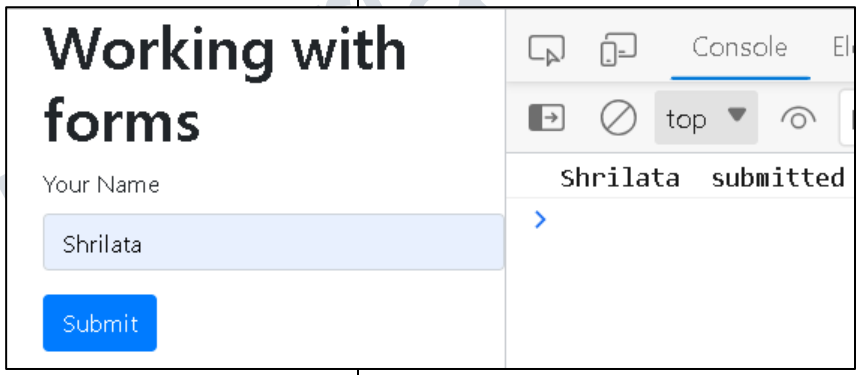
Controlled Components and Uncontrolled components

- In React forms you can either allow the browser to handle most of the form elements or you can use React to fully control the element by setting and updating the input value directly.
  - The first approach is called an uncontrolled component because React is not setting the value.
  - The second approach is called a controlled component because React is actively updating the input.

- In HTML, form data is usually handled by the DOM.
- In React, form data is usually handled by the components.
  - When the data is handled by the components, all the data is stored in the component state.

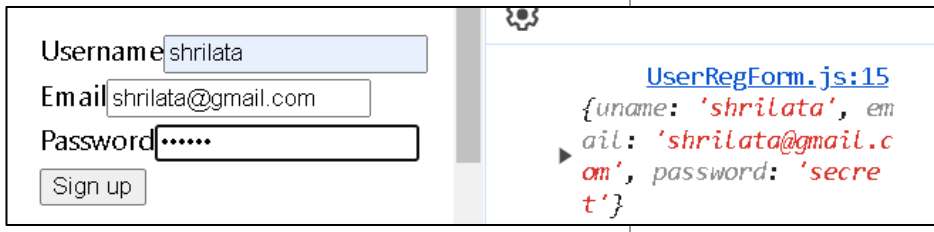
Controlled components : React forms: SimpleInput.js

```
const SimpleInput = (props) => {

    const [inputName, setInputName] = useState('')

    const inputNameHandler = (event) => {
        setInputName(event.target.value)
    }

    const formSubmitHandler = event => {
        event.preventDefault();
        console.log(inputName, " submitted")
    }
    return (
      <form onSubmit={formSubmitHandler}>
        <div className='form-group'>
            <label htmlFor='name'>Your Name</label>
            <input type='text' name='uname'
                   className='form-control'
                   onChange={inputNameHandler}/>
        </div>
        <div className="form-actions">
            <button class="btn btn-primary">Submit</button>
        </div>
      </form>
    );
};
```



Controlled components: another eg

```
import { useState } from "react"
const UserRegForm = () => {

    const initValues = {uname:"", email:"", password:""}
    const [formValues, setFormValues] = useState(initValues)

    let handleChange = (e) => {
        const {name, value} = e.target
        setFormValues({...formValues,[name]:value})
    }

    let submitHandler = (event) => {
        event.preventDefault()
        console.log(formValues) //console.log(formValues.uname, formValues.email)
        setFormValues({uname:"", email:"", password:""})
    }

    return (
        <form onSubmit={submitHandler}>
            <label htmlFor="uname">Username</label>
            <input name="uname" value={formValues.uname} onChange={handleChange}/>
            <label htmlFor="email">Email</label>
            <input type="email" name="email" value={formValues.email}
                   onChange={handleChange}/>
            <label htmlFor="password">Password</label>
            <input type="password" name="password" value={formValues.password}
                   onChange={handleChange}/>
            <button type="submit">Sign up</button>
     </form>
    );
}
export default UserRegForm
```



Uncontrolled Inputs

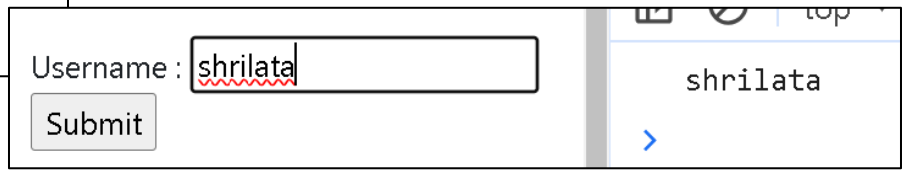
- “uncontrolled” form inputs: React doesn’t track the input’s state.
  - Uncontrolled components are inputs that do not have a value property. In opposite to controlled components, it is the application's responsibility to keep the component state and the input value in sync.
  - In order to do this, React allows us to create a “**ref**” (reference) to associate with an element, giving access to the underlying DOM node

  - “**ref** is used to receive the form value from DOM.
  - To enable this, React allows us to create a “ref” (reference) to associate with an element, giving access to the underlying DOM node.
  - Refs provide a way to access DOM nodes or React elements created in the render method.
  - In class component, Refs are created using React.createRef() and attached to React elements via the ref attribute.
    - Refs are commonly assigned to an instance property when a component is constructed so they can be referenced throughout the component.
  - In function components, refs are created using useRef() hook

Uncontrolled Inputs : function component

```
import { useRef } from "react";

let SimpleUncontrolledForm = () => {

    let unameRef = useRef("")

    let submitHandler = (e)=>{
        e.preventDefault();
        console.log(unameRef.current.value)
    }

    return (
        <form onSubmit={submitHandler}>
            Username : <input ref={unameRef} />
            <input type="submit"/>
        </form>
    )
}
export default SimpleUncontrolledForm
```

- You initialize a new ref by calling useRef() hook, assigning it to an instance property so it's available for the lifetime of the component.
- In order to associate the ref with an input, it's passed to the element as the special ref attribute.
- Once this is done, the input's underlying DOM node can be accessed via unameRef.current.value



Another example : Login form

```
class LoginForm extends Component {
   constructor(props) {
      super(props);
      this.nameEl = React.createRef();
      this.passwordEl = React.createRef();
      this.rememberMeEl = React.createRef();
    }
   handleSubmit = (e) => {
        e.preventDefault();
        const data = {
            username: this.nameEl.current.value,
            password: this.passwordEl.current.value,
            rememberMe: this.rememberMeEl.current.checked,
          }
        console.log(data)
      }
    render(){
      return (
       <form onSubmit={this.handleSubmit}>
         <fieldset><legend>Login Form</legend>
         <input type="text" placeholder="username" ref={this.nameEl} /><br></br>
         <input type="password" placeholder="password" ref={this.passwordEl} /><br></br>
         <label><input type="checkbox" ref={this.rememberMeEl} />Remember me
         </label><br></br>
         <button type="submit" className="myButton">Login</button>
         </fieldset>
       </form>
     );
}}
```

Login Form

username

password

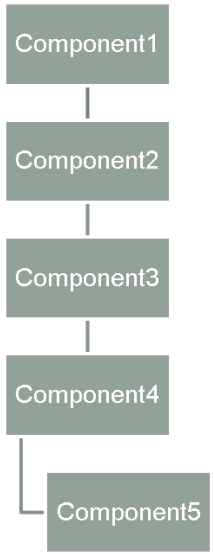
Remember me

Login

```
{username: 'aaa', password: 'bbb', rememberMe: true}
```

Context API

- The main purpose of Contex API is to avoid props drilling.
- Context API consists of two main components: the context provider and the context consumer.
  - The provider is responsible for creating and managing the context, which holds the data to be shared between components.
  - The consumer is used to access the context and its data from within a component.
  - This avoids the need to pass the information down through props, making your code more efficient and easier to manage.
  - Store the state in a Context value in the common ancestor component (called the Provider Component), and access it from as many components as needed (called Consumer Components), which can be nested at any depth under this ancestor.
  - React.js takes care of all the magic behind the scenes to make this work.



- Step 1 : Create context
  - Create Context in - UserContext.js

```
import React from 'react';
export default React.createContext();
```

  - There are two ways to assign a state to our context.
    - By assigning default values when our context object is created.
      `export default React.createContext('');`
    - Use of Provider component in parent component by using value to set state in our context object.

```react
import { useState } from "react";
import UserContext from "./UserContext";

function CtxComponent1() {
  const [user, setUser] = useState("Soha T");

  return (
    <UserContext.Provider value={user}>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 user={user} />
    </UserContext.Provider>
  );
}
export default CtxComponent1;
```

- Step-2: wrap the components that need access to the shared data with a Provider component.
  - The Provider component accepts a "value" prop that holds the shared data, and any component that is a child of the Provider component can access that shared data.
  - It's important to note that the Provider component should be wrapped around the top-level component in an application to ensure that all child components have access to the shared data.

```react
function CtxComponent5() {
  const user = useContext(UserContext);

  return (
    <>
      <h1>Component 5</h1>
      <h2>{`Hello ${user} again!`}</h2>
    </>
  );
}
```

- Step-3 :In order to use the Context in a child component, we need to access it using the useContext Hook.
  - First, include the useContext in the import statement
  - Then you can access the user Context in all components:

Hello Soha T!

Component 5

Hello again!

REDUX : Tasklist app : step-1 : set up app

- Create a new Redux app: npx create-react-app redux-tasklist-app
- cd redux-tasklist-app
- npm install redux react-redux //legacy way, don't use
- npm install @reduxjs/toolkit react-redux //Modern Redux
  - Redux Toolkit (also known as "RTK" for short) is the official recommended approach for writing Redux logic.
  - The @reduxjs/toolkit package wraps around the core redux package, and contains API methods and common dependencies that are essential for building a Redux app

Tasklist app : step-2 : Create store

- Create a folder store in src and a file called store.js

```
import { configureStore } from "@reduxjs/toolkit";

export const store = configureStore({
    reducer:{} //we havent defined any reducer as yet
})
//we are passing a collection of reducers in configureStore()
```

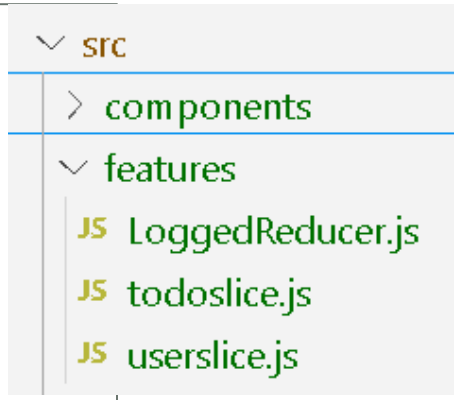
```
import { configureStore } from "@reduxjs/toolkit";
import userReducer from "../features/userslice";
import todoReducer from "../features/todoslice"

export const store = configureStore({
    reducer:{
        user:userReducer,
        todo:todoReducer,
        counter:counterReducer
    }
})
```

Once the store is created, we can make it available to our React components by putting a React-Redux <Provider> around our application in src/index.js.

```
//src/index.js
import { Provider } from 'react-redux';
import { store } from './store/store';

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <Provider store={store}>
    <App />
    </Provider>
 );
```

Tasklist app : step-3 : Create reducer & hence action

```
import { createSlice, nanoid } from "@reduxjs/toolkit"

const initialState = {
    todos :[]
}
export const todoSlice = createSlice({
    name:'todo',  //will be using this name to refer to the slice
    initialState,
    reducers : {
        addTodo : (state,action) =>{
            const todo = {
                id:nanoid(),
                text:action.payload
            }
            state.todos.push(todo)
        },
        removeTodo: (state,action) =>{
          state.todos = state.todos.filter( todo => todo.id !== action.payload)
        }
    }
})
export const {addTodo, removeTodo} = todoSlice.actions
export default todoSlice.reducer
```



A "slice" is a collection of Redux reducer logic and actions for a single feature in your app, typically defined together in a single file

Tasklist app : step-3 : counter reducer example

```
import { createSlice } from '@reduxjs/toolkit'

const initialState = {value: 0 }

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state, action) => {
      state.value += 1
    },
    decrement: (state, action) => {
      state.value -= 1
    }
  },
})
export const { increment, decrement } = counterSlice.actions
export default counterSlice.reducer
```

Tasklist app : step-4 : creating components

```
import React, { useState } from 'react'
import { useDispatch } from 'react-redux'
import { addTodo } from '../features/todoslice'

const AddTodo = () => {

    const [input, setInput] = useState("")

    const dispatch = useDispatch()

    let submitHandler = (e) => {
        e.preventDefault()
        dispatch(addTodo(input))
        setInput("")
    }
    return (
        <form onSubmit={submitHandler}>
            <div className="form-group">
                <input placeholder="Enter a Todo..."
                    value={input} className="form-control"
                    onChange={e => setInput(e.target.value)} />
            </div>
            <button type="submit" className='btn btn-danger'>
                Add Todo</button>
        </form>
    )
}
export default AddTodo
```

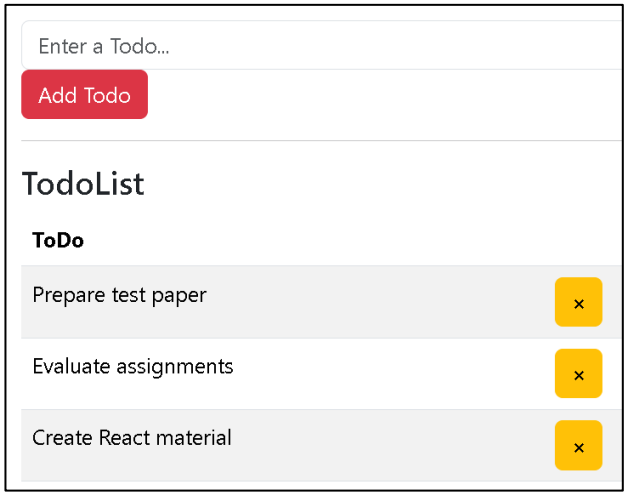
- src
  - components
    - AddTodo.js
    - Login.js
    - Profile.js
    - TodoList.js

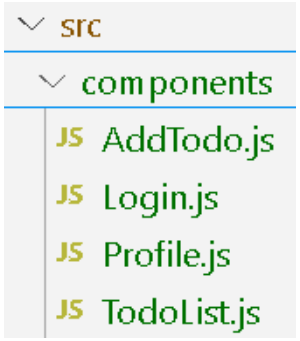
Enter a Todo...

Add Todo

Tasklist app : step-4 : creating components

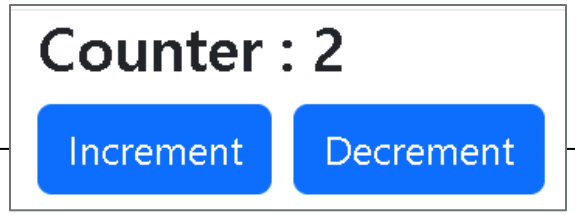
```
import React from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { removeTodo } from '../features/todoslice'

const TodoList = () => {
    const list = useSelector(state => state.todo) //to fetch all todos from store
    const dispatch = useDispatch() //to dispatch removeTodo action

    return (
        <div>
            <hr />
            <h3>TodoList</h3>
            <table className="table table-striped">
                <thead className="thead-info">
                    <tr><th>ToDo</th><th> </th></tr>
                </thead>
                <tbody>
                    {list.todos.map(todo=>(
                        <tr key={todo.id}>
                            <td>{todo.text}</td>
                            <td><button onClick={()=>dispatch(removeTodo(todo.id))}
                                        className='btn btn-warning'>&times;</button>
                            </td>
                        </tr>
                    ))}
                </tbody>
                </table>
        </div>
    )
}
export default TodoList
```





Tasklist app : step-4 : creating components

```
import { useDispatch, useSelector } from "react-redux"
import { decrement, increment } from "../features/counterslice"

let CounterComponent = ()=>{

    const count = useSelector(state => state.counter.value)
    const dispatch = useDispatch()

    return (
        <div>
            <h3>Counter : {count}  </h3>
            <button onClick={()=>dispatch(increment())}
                    className="btn btn-primary">Increment</button>
            <button onClick={()=>dispatch(decrement())}
                    className="btn btn-primary">Decrement</button>
        </div>
    )
}
export default CounterComponent
```



App component

```
import logo from './logo.svg';
import AddTodo from './components/AddTodo';
import TodoList from './components/TodoList';
import CounterComponent from './components/CounterComponent';

function App() {
  return (
    <div className="container">
      <div className="row">
        <div className="col col-sm-6">
          <AddTodo />
        </div>
        <div className="col col-sm-6">
          <CounterComponent />
        </div>
      </div>
      <div className="row"></div>
      <div className="row">
        <div className="col col-sm-12">
          <TodoList />
        </div>
      </div>
    </div>
  );
}
export default App;
```