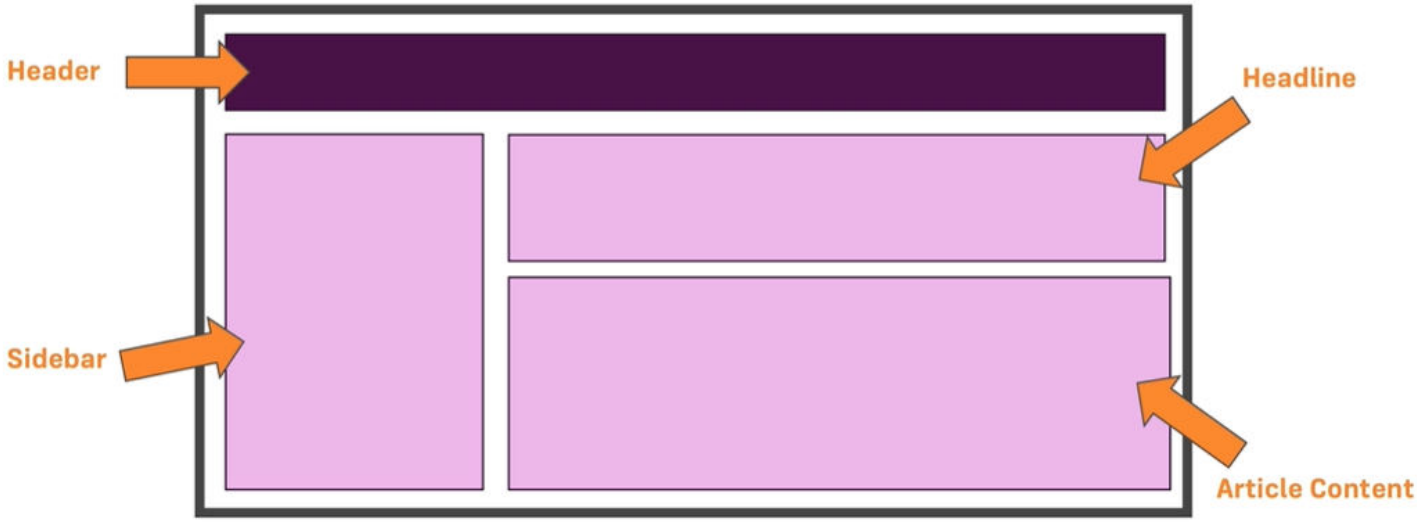
# REACT.JS

JavaScript library for building user interfaces

# What is React?

- From the official React page : A JavaScript library for building user interfaces
- Its not a framework; React does only one thing – create awesome UI!
- React is used to build single page applications.
- React.js is a JavaScript library. It was developed by engineers at Facebook.
- React is a declarative, efficient, and flexible JavaScript library for building user interfaces.
- It lets you compose complex UIs from small and isolated pieces of code called “components”.



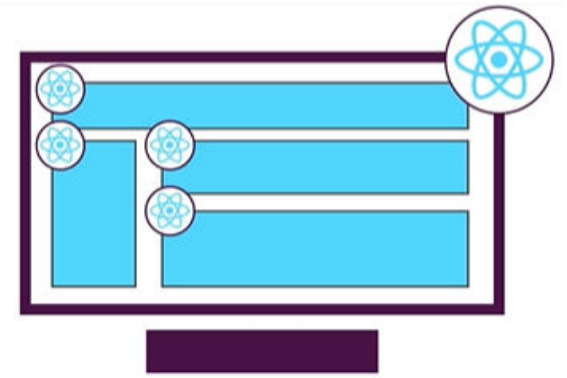
# Client-side Javascript frameworks

- Ember : was initially released in December 2011. It is  an older framework that has less users than more modern alternatives  such as React and Vue
- Angular is an open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations.
- Vue : first released in 2014; is the youngest of the big four, but has enjoyed a recent uptick in popularity.
- React : released by Facebook in 2013. By this point, FB had already been using React to solve many of its problems internally.
  - React itself is not technically a framework; it's a library for rendering UI components.
  - React is used in combination with other libraries to make applications — React and React Native enable developers to make mobile applications; React and ReactDOM enable them to make web applications, etc.

# Components

- A **Component** is one of the core building blocks of React.
- Its just a **custom HTML element**!

- Every application you will develop in React will be made up of pieces called components.
  - Components make the task of building UIs much easier. You can see a UI broken down into multiple individual pieces called components and work on them independently and merge them all in a parent component which will be your final UI.



# Why react

- Created and maintained by facebook
- Has a huge community on Github
- Component based architecture
- React is *fast* Apps made in React can handle complex updates and still feel quick and responsive.
- React is *modular* Instead of writing large, dense files of code, you can write many smaller, reusable files. React's modularity can be a beautiful solution to JavaScript's maintainability problems.
- React is *scalable* Large programs that display a lot of changing data are where React performs best.
- React is *popular*.
- UI state becomes difficult to manage with vanilla Javascript

# Requirements

- Ensure that NodeJS and typescript are installed
  - Install TypeScript as follows:
  - npm install –g  typescript

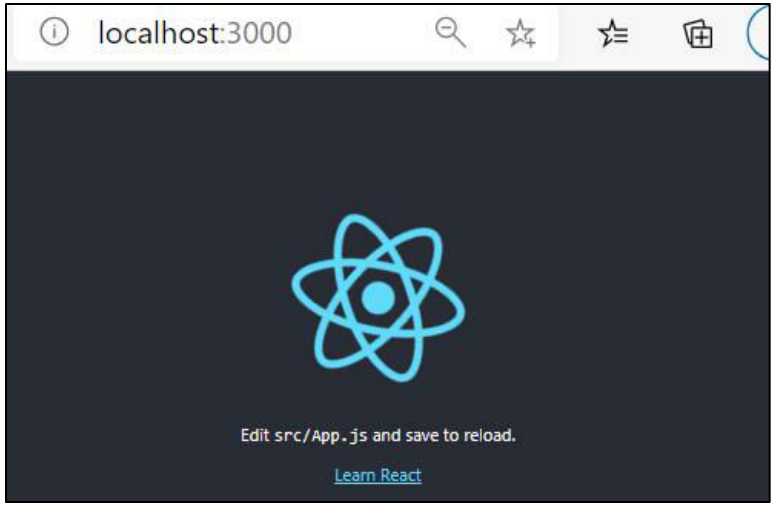
```
C:\Users\Shrilata>node --version
v14.16.0

C:\Users\Shrilata>npm --version
6.14.11

C:\Users\Shrilata>tsc --version
Version 4.4.3

C:\Users\Shrilata>npx --version
6.14.11
```

# Using create-react app

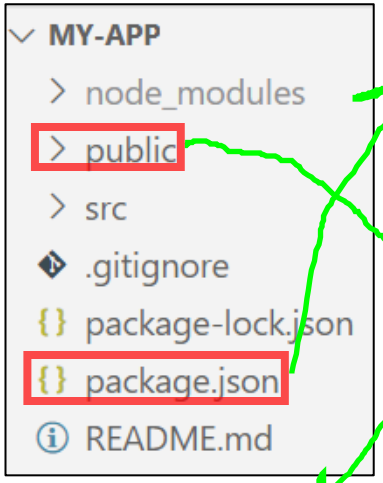
- npx create-react-app my-app
- cd my-app
- npm start

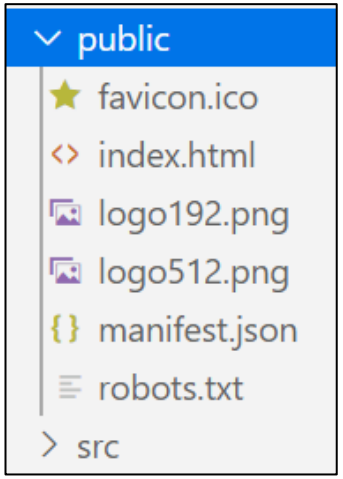
Create React App is a comfortable environment for **learning React**, and is the best way to start building **a new single-page application** in React.

It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production. You'll need to have Node >= 14.0.0 and npm >= 5.6 on your machine. To create a project, run:

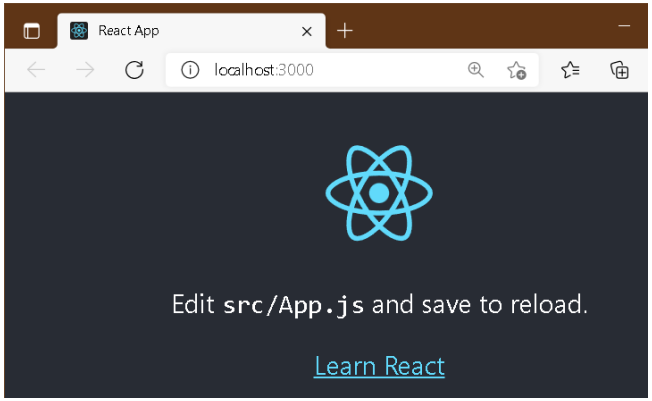


# Understanding the folder structure




```
"dependencies": {
  "@testing-library/jest-dom": "^5.11.6",
  "@testing-library/react": "^11.2.2",
  "@testing-library/user-event": "^12.2.2",
  "react": "^17.0.1",
  "react-dom": "^17.0.1",
  "react-scripts": "4.0.0",
  "web-vitals": "^0.2.4"
},
▷ Debug
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```




```
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.
```

# Index.html

```
<> index.html > ...
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--…
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--…
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
    <!--…
  </body>
</html>
```

- The root node is the HTML element where you want to display the result.
- It is like a container for content managed by React.
- It does NOT have to be a  element and it does NOT have to have the id='root

# Understanding the folder structure

```
MY-APP
  node_modules
  public
  src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
export default App;
```

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

export default App;
```

index.js

```
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'
);
```

App.css > .App

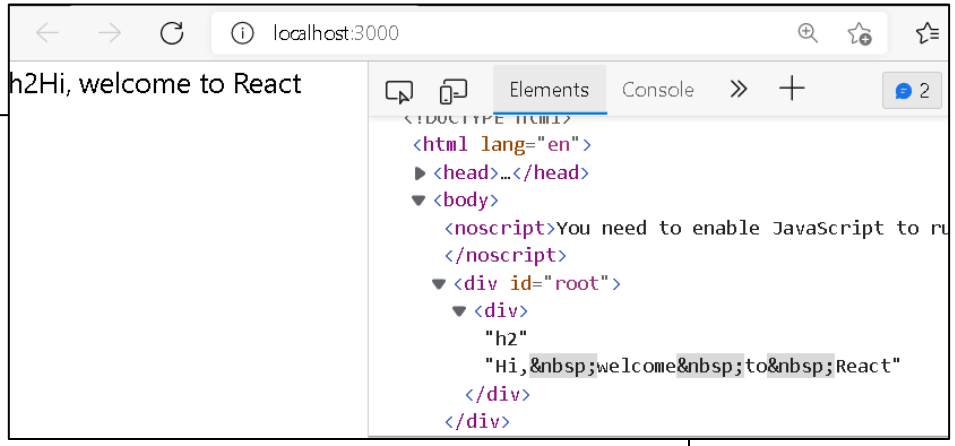
```
.App {
  text-align: center;
}
```

localhost:3000

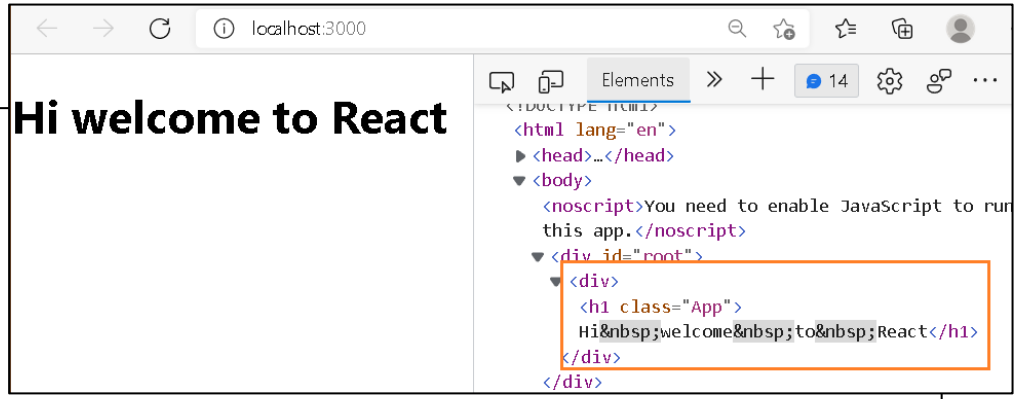
**Welcome to React!**

# Understanding JSX

```
import React from 'react';

function App() {
  /*return (
    <div >
      <h2>Welcome to React!</h2>
    </div>
  );*/
  return React.createElement('div',null,'h2','Hi, welcome to React');
}
export default App;
```




```
function App() {
  /*return (
    <div >
      <h2>Welcome to React!</h2>
    </div>
  );*/
  return React.createElement('div',null,
          React.createElement('h1',{className:'App'},'Hi welcome to React'));
}
export default App;
```



# Understanding JSX

- Eg : const mytag = <h1>Hello React!</h1>;

```
const myelement = <h1>Understanding JSX!</h1>;

ReactDOM.render(myelement, document.getElementById('root'));
```

```
const myelement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);

ReactDOM.render(myelement, document.getElementById('root'));
```

```
function App() {
  return (JSX attribute) React.HTMLAttributes<HTMLDivElement>.className?: string
    <div className="App">
      <h2>Welcome to React!</h2>
    </div>
  );
}
export default App;
```

```
▼<div class="App">
    <h1> Hi, welcome to React</h1>
  </div>
```

```
const myelement = (
  <div>
    <h1>I am a Header.</h1>
    <h1>I am a Header too.</h1>
  </div>
);

ReactDOM.render(myelement, document.getElementById('root'));
```

If we want to return more elements, we need to wrap it with one container element. Notice how we are using div as a wrapper for the two h1 elements.

# Creating a functional component

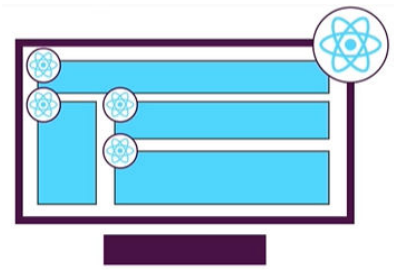
- Components are the essential building blocks of any app created with React
  - A single app most often consists of many components.
- A component is in essence, a piece of the UI - splitting the user interface into reusable and independent parts, each of which can be processed separately.
  - Components are independent and reusable bits of code.
  - It's an encapsulated piece of logic.
  - They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render function.

Ways to write functional components

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

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

ES6



# Creating a functional component

- Adding our component to main component
  - To use this component in your application, use similar syntax as normal HTML

```
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;
```

localhost:3000

**Welcome to React!**

**Expense Item**

```
<head>…</head>
<body> == $0
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root">
    <div class="App">
      <h2>Welcome to React!</h2>
      <h2>Expense Item</h2>
    </div>
  </div>
```

- Creating components makes them reusable and configurable.
- Reusing is simple. Eg, simply copy paste <ExpenseItem/> multiple times in App.js.

```
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 > JS 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

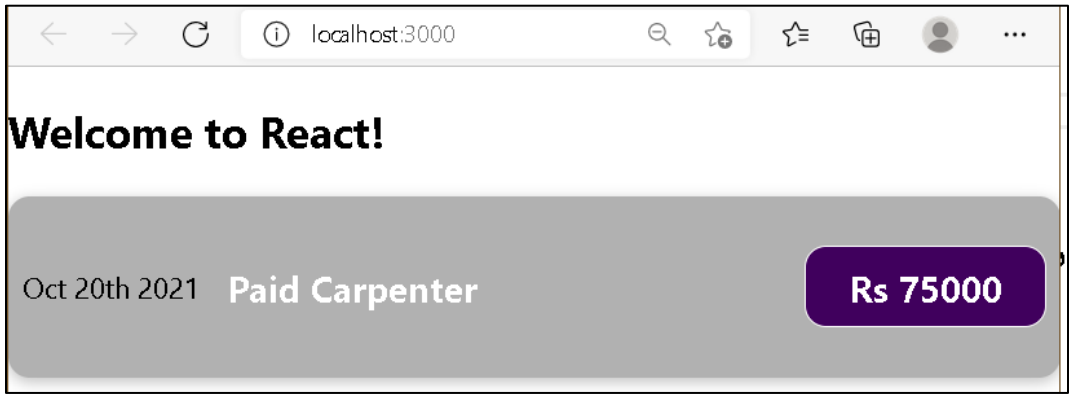
Elements Console Sources Ne

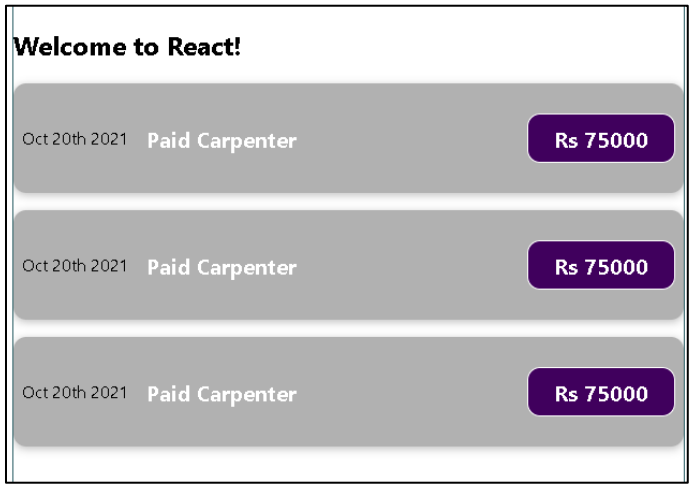
```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
...▼<body> == $0
  <noscript>You need to enable JavaSc
  ▼<div id="root">
    ▼<div class="App">
      <h1> Hi, welcome to React</h1>
      <p>Hi Person</p>
    </div>
  </div>
```

# 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 {…
}
```



# Components & JSX

- When creating components, you have the choice between two different ways:
- Functional components (also referred to as "presentational", "dumb" or "stateless" components)

```
const cmp = () => {
    return <div>some JSX</div>
}
```

*using ES6 arrow functions as shown here is recommended but optional*

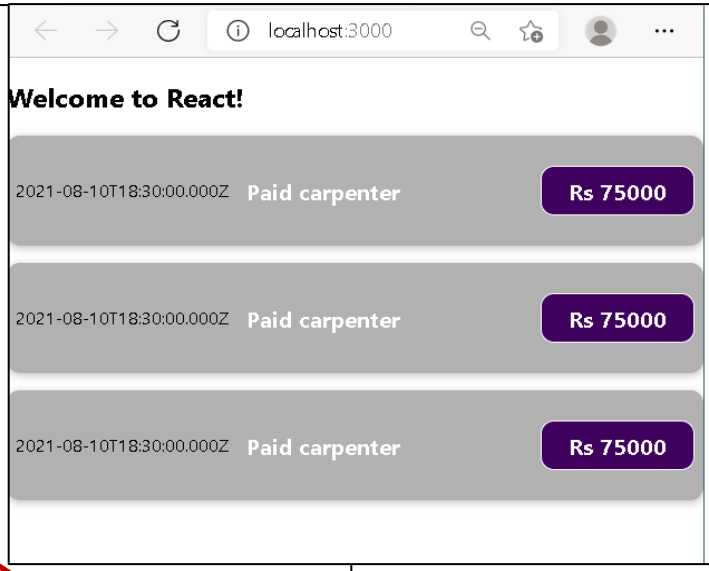
- class-based components (also referred to as "containers", "smart" or "stateful" components)

```
class Cmp extends Component {
    render () {
        return <div>some JSX</div>
    }
}
```

# 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 class="expense-item__description">
                <h2>{expTitle}</h2>
                <p class="expense-item__price">Rs {expAmount}</p>
            </div>
        </div>
    )
}
export default ExpenseItem;
```




Comments in JSX

# Outputting dynamic content – another example

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

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

Wrap dynamic content in JSX in {...}

math.random gives random value bet 0 and 1(exclu)

```
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