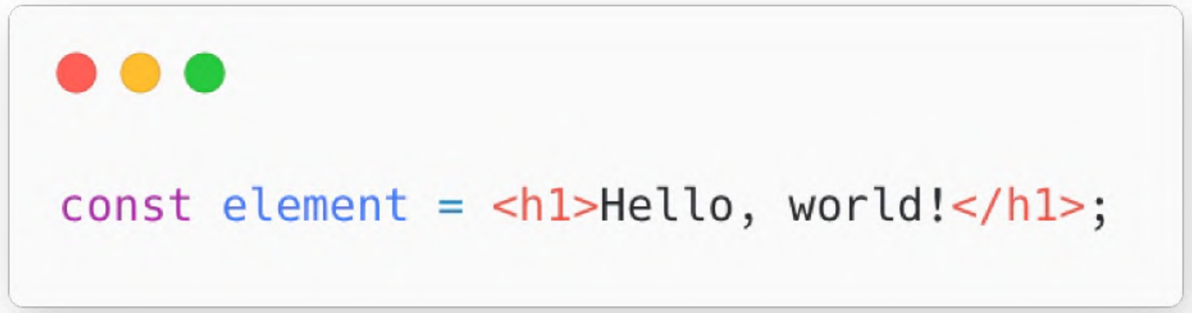
avion

Lesson : JSX and Components

FRONT-END COURSE (V1.2.0)

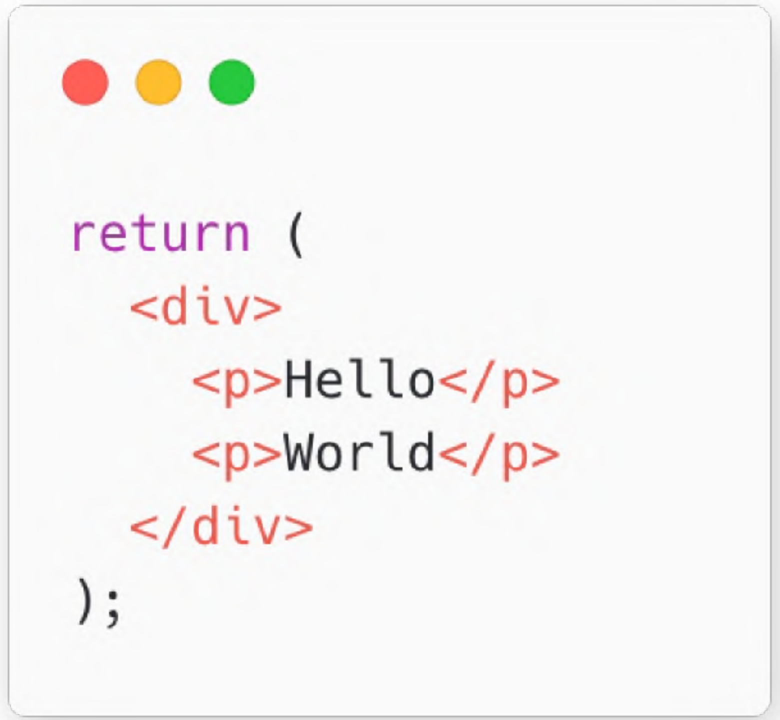
Introducing JSX

- In React projects, we don't create separate HTML files, because JSX allows us to write HTML and JavaScript combined together in the same file.
- JSX may remind you of a template language, but it comes with the full power of JavaScript.
- JSX produces React “elements”.



```
const element = <h1>Hello, world!</h1>;
```

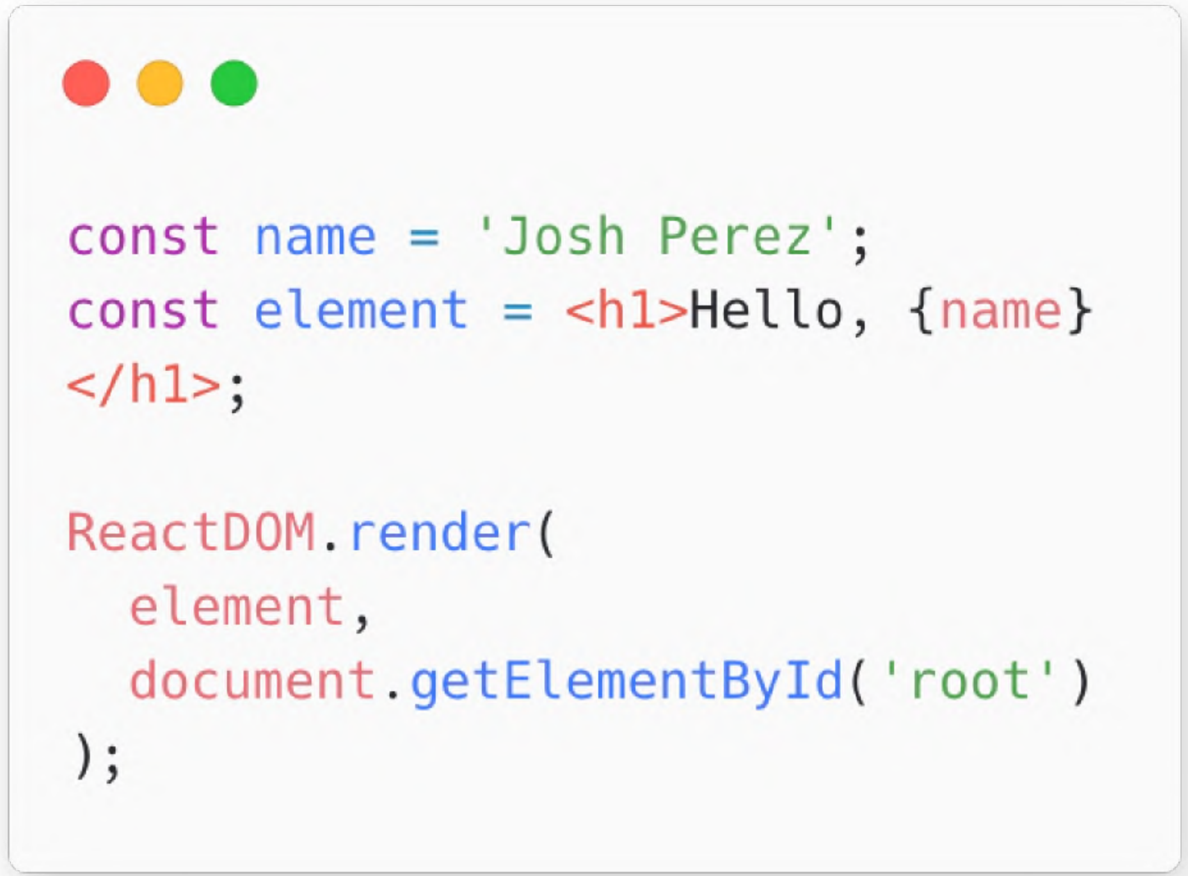
- HTML and component tags must always be closed < />
- Some attributes like “class” become “className” (because class refers to JavaScript classes), “tabindex” becomes “tabIndex” and should be written camelCase
- We can’t return more than one HTML element at once, so make sure to wrap them inside a parent tag:



```
return (
  <div>
    <p>Hello</p>
    <p>World</p>
  </div>
);
```

Embedding Expressions in JSX

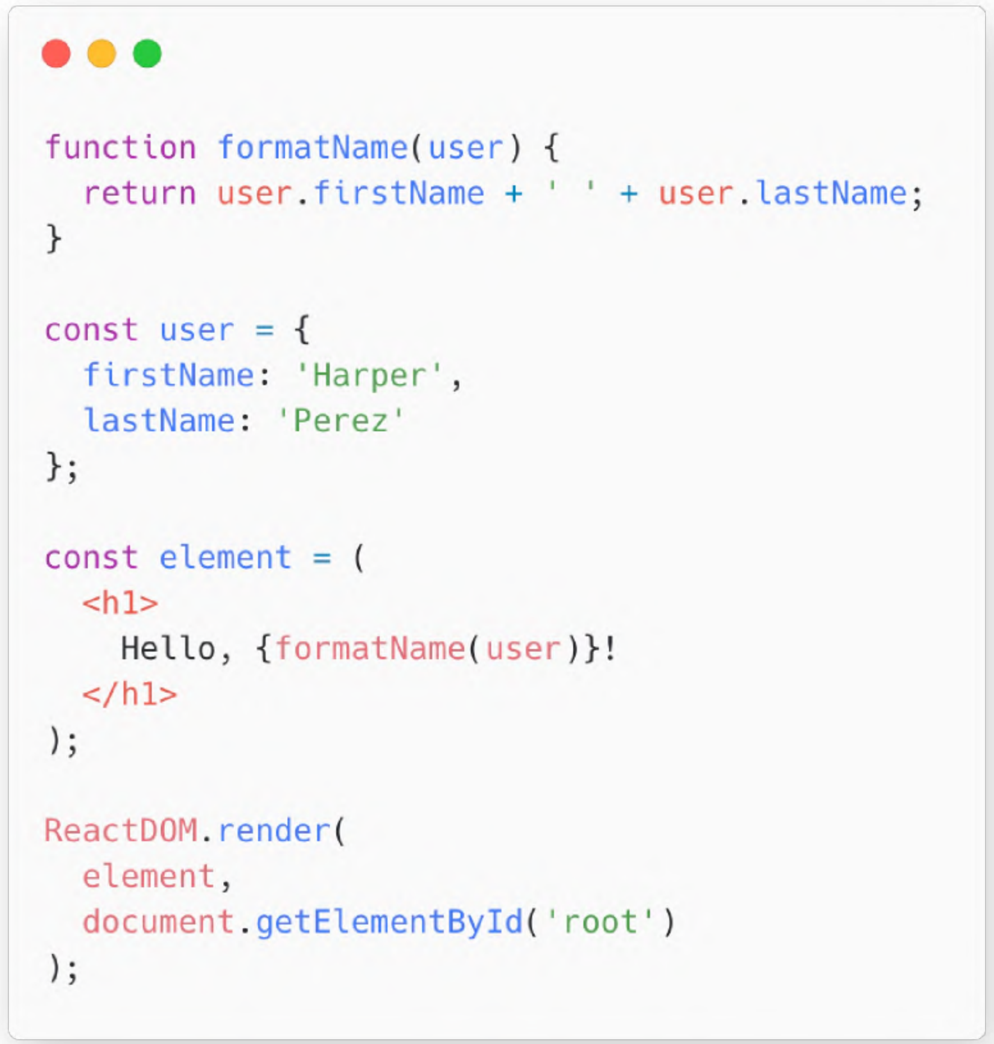
- We declare a variable called name and then use it inside JSX by wrapping it in curly braces



```
const name = 'Josh Perez';
const element = <h1>Hello, {name}
</h1>;

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

Embedding Expressions in JSX

- We embed the result of calling a JavaScript function, formatName(user), into an <h1> element.

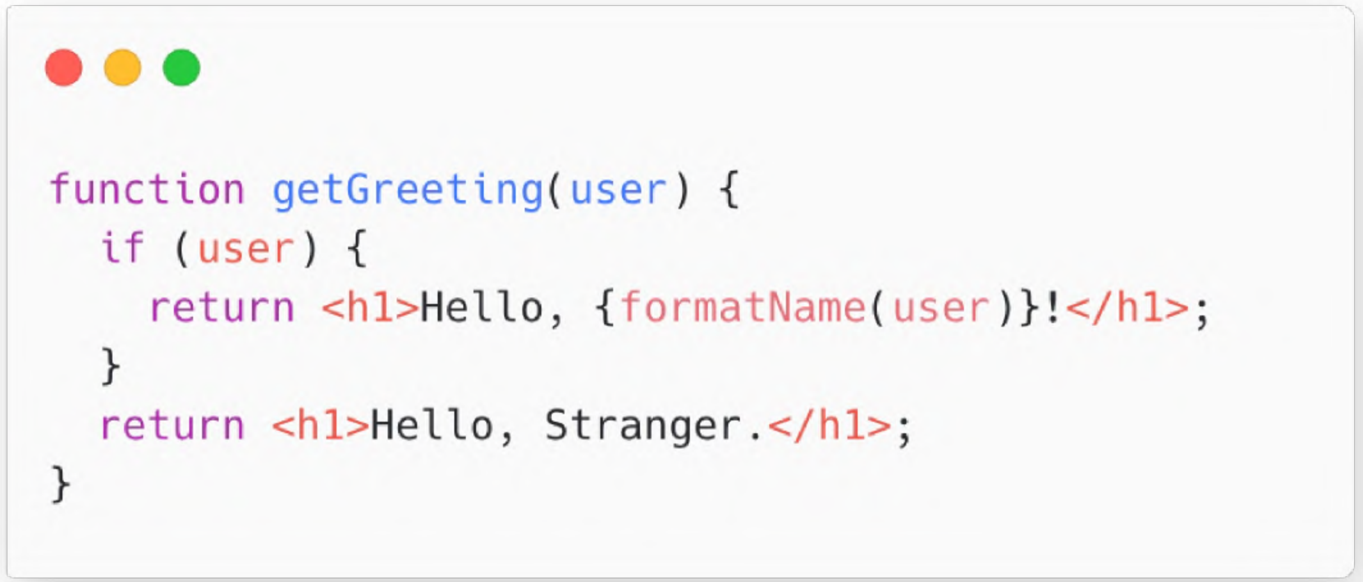


```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

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

- We split JSX over multiple lines for readability. While it isn't required, when doing this, we also recommend wrapping it in parentheses to avoid the pitfalls of automatic semicolon insertion.

JSX is an Expression Too

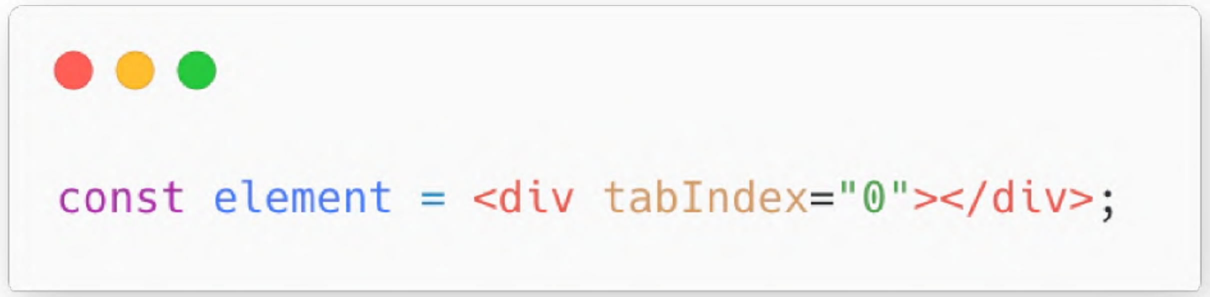
- After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.
- This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions.



```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

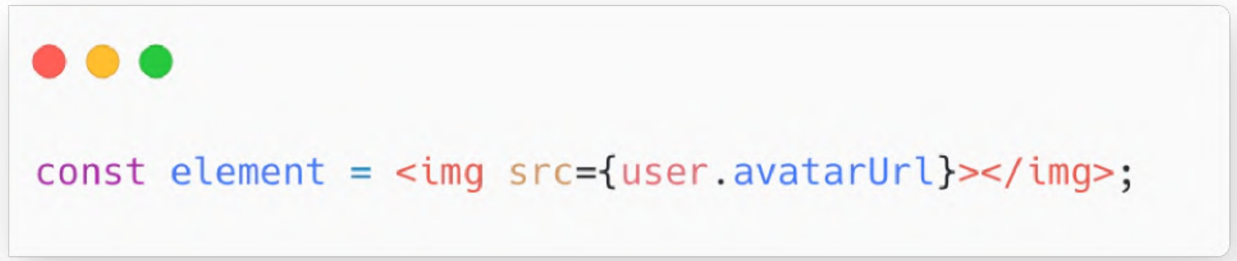
Specifying Attributes with JSX

- You may use quotes to specify string literals as attributes



```
const element = <div tabIndex="0"></div>;
```

- You may also use curly braces to embed a JavaScript expression in an attribute:

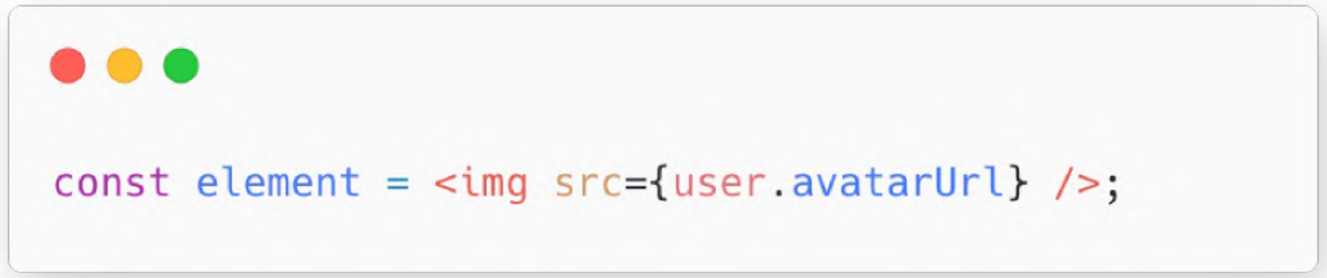


```
const element = <img src={user.avatarUrl}></img>;
```

- Don’t put quotes around curly braces when embedding a JavaScript expression in an attribute. You should either use quotes (for string values) or curly braces (for expressions), but not both in the same attribute.

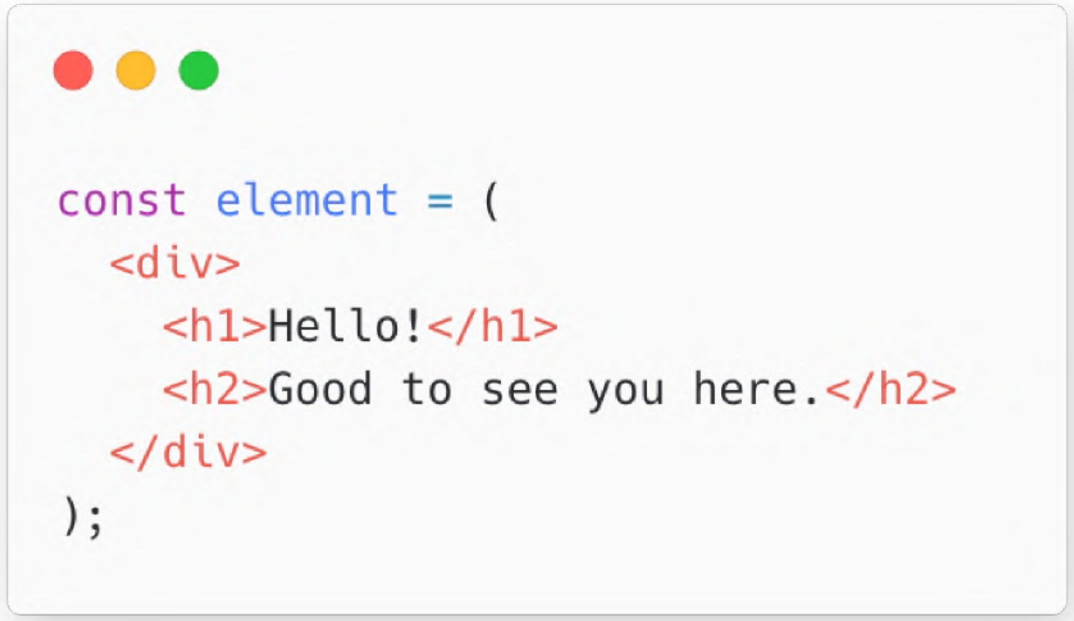
Specifying Children with JSX

- If a tag is empty, you may close it immediately with />, like XML



```
const element = <img src={user.avatarUrl} />;
```

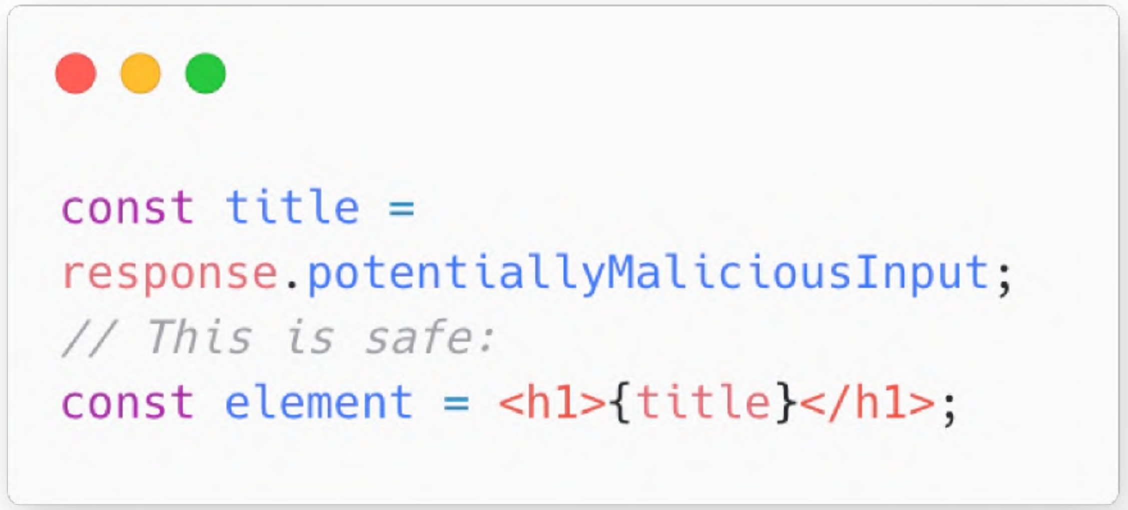
- JSX tags may contain children:



```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

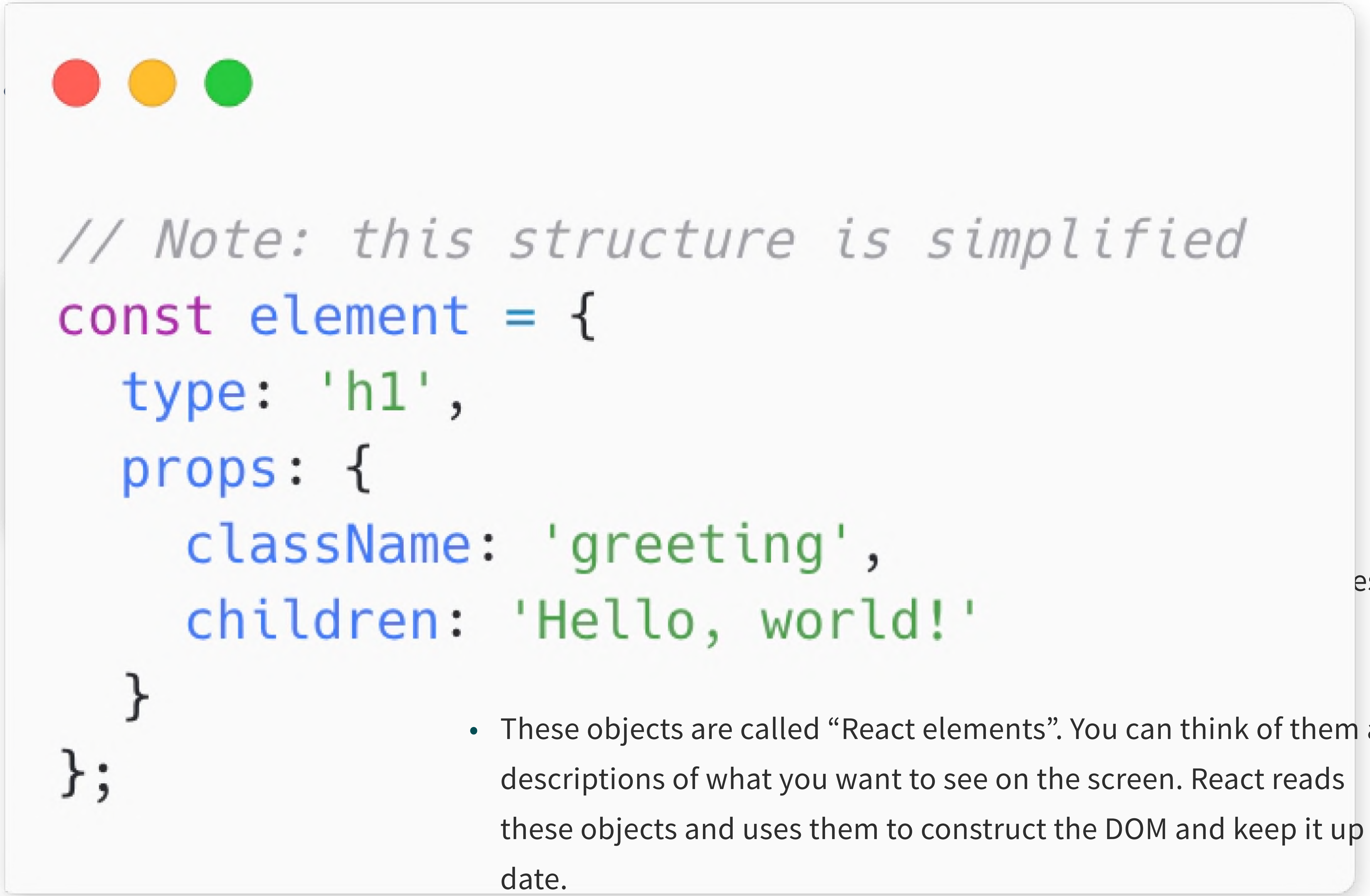
JSX Prevents Injection Attacks

- It is safe to embed user input in JSX



```
const title =
response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

- By default, React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that’s not explicitly written in your application. Everything is converted to a string before being rendered. This helps prevent XSS (cross-site-scripting) attacks.



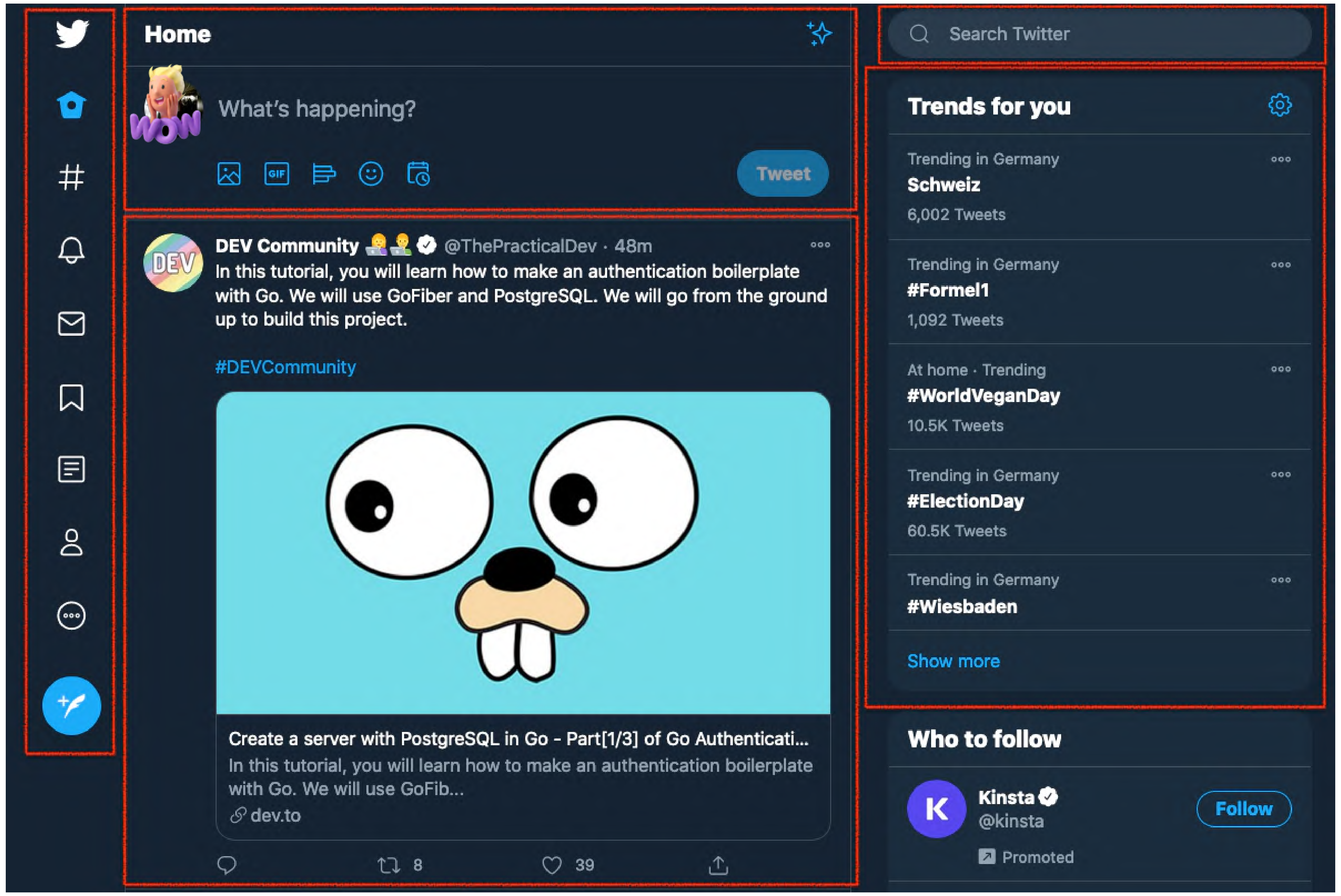
```
// Note: this structure is simplified
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

es an object like this:

- These objects are called “React elements”. You can think of them as descriptions of what you want to see on the screen. React reads these objects and uses them to construct the DOM and keep it up to date.

Components

- A component is an independent, reusable code block which divides the UI into smaller pieces. For example, if we were building the UI of Twitter with React:



DEV Community @ThePracticalDev · 48m
In this tutorial, you will learn how to make an authentication boilerplate with Go. We will use GoFiber and PostgreSQL. We will go from the ground up to build this project.

#DEVCommunity

Create a server with PostgreSQL in Go - Part[1/3] of Go Authenticati...
In this tutorial, you will learn how to make an authentication boilerplate with Go. We will use GoFib...
dev.to

8 39

Trending in Germany
Schweiz
6,002 Tweets

Trending in Germany
#Formel1
1,092 Tweets

At home · Trending
#WorldVeganDay
10.5K Tweets

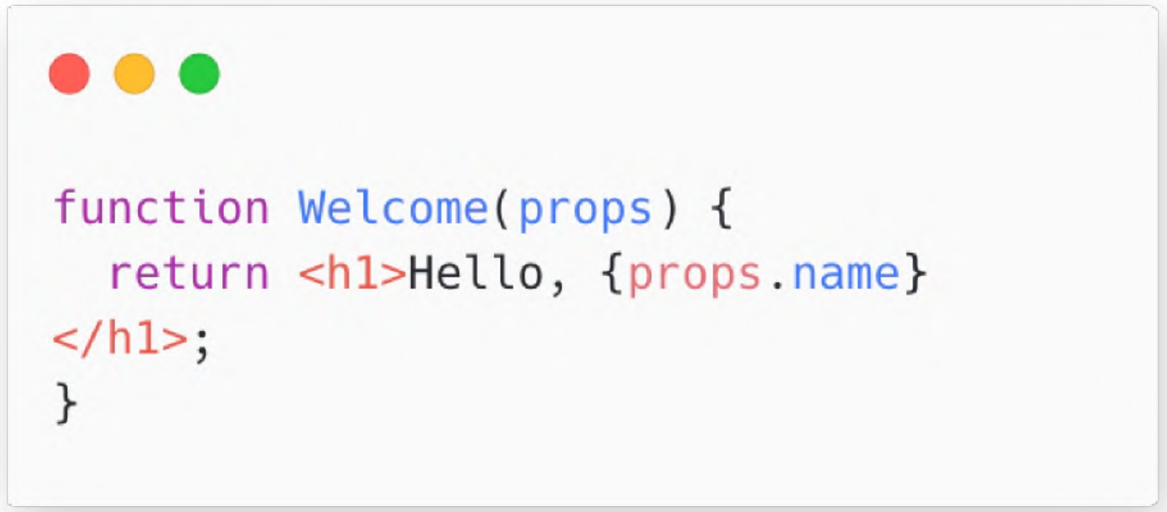
Trending in Germany
#ElectionDay
60.5K Tweets

Trending in Germany
#Wiesbaden

Kinsta
@kinsta
Promoted

Functional Components

- A functional component is basically a JavaScript/ES6 function that returns a React element (JSX).
- always starts with a capital letter (naming convention)
- takes props as a parameter if necessary

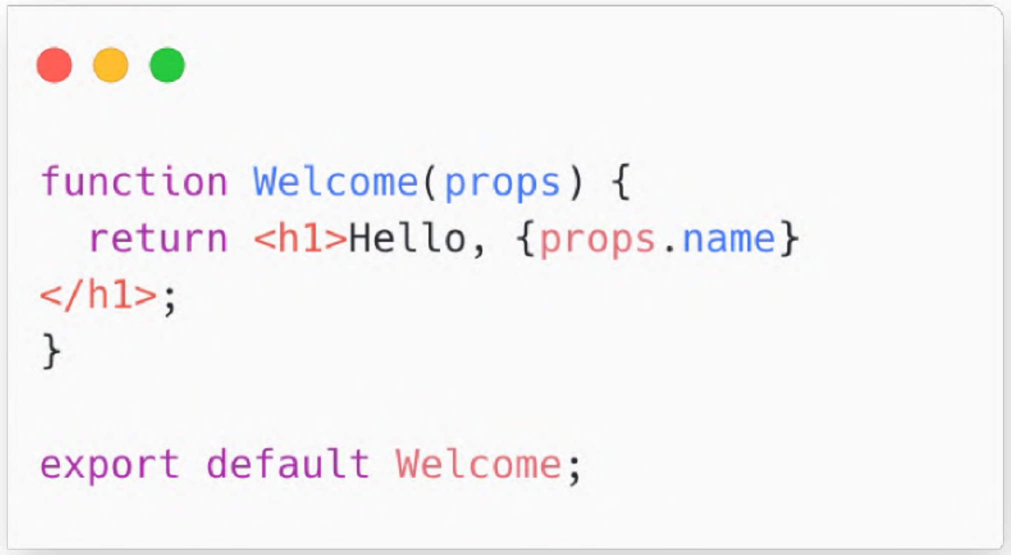


```
function Welcome(props) {
  return <h1>Hello, {props.name}
</h1>;
}
```

This function is a valid React component because it accepts a single "props" (which stands for properties) object argument with data and returns a React element. — reactjs.org

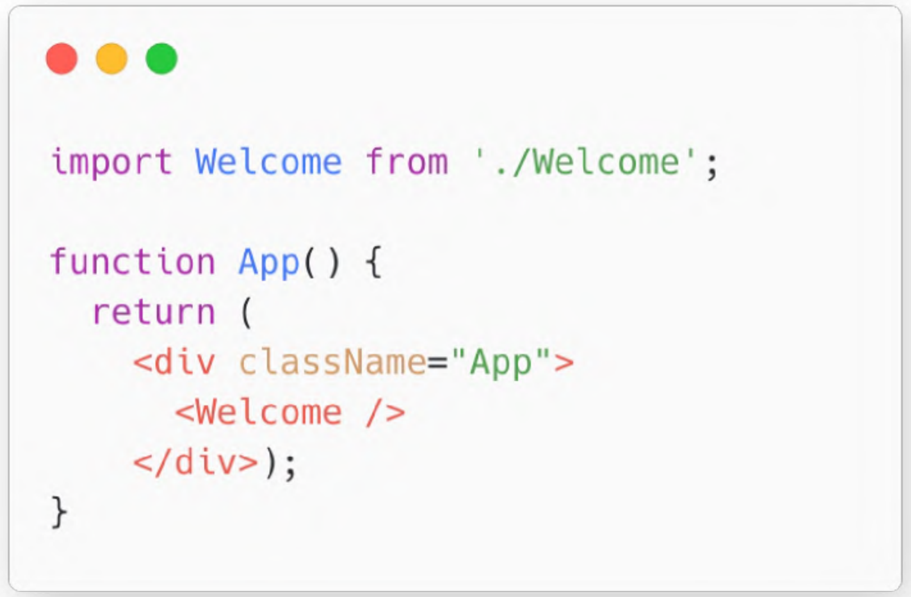
Functional Components

- To be able to use a component later, you need to first export it so you can import it somewhere else.



```
function Welcome(props) {
  return <h1>Hello, {props.name}
</h1>;
}

export default Welcome;
```

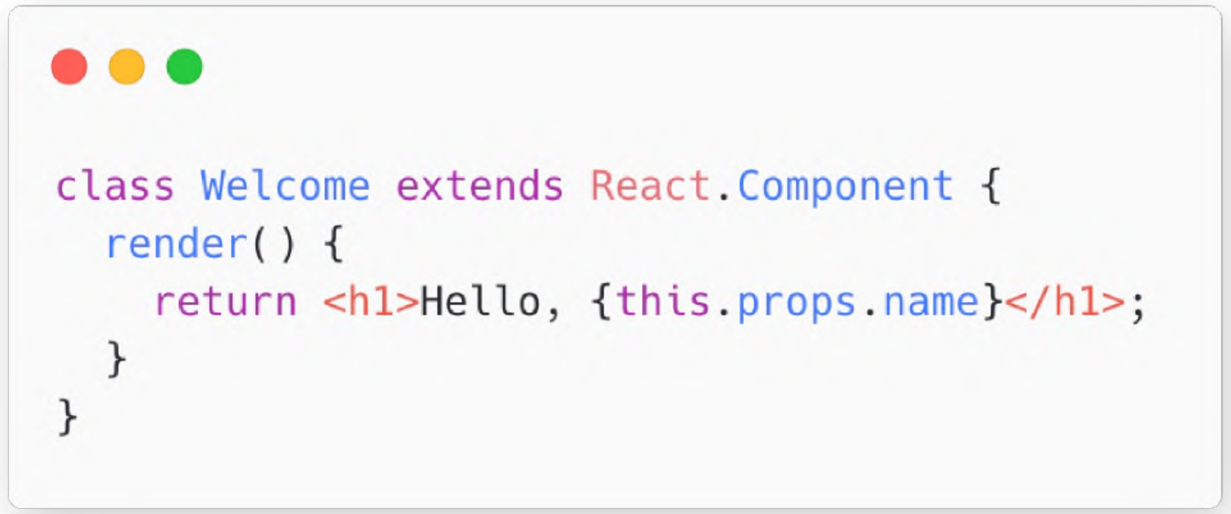
After importing it, you can call the component like in this example:



```
import Welcome from './Welcome';

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

Class Components

- Class components are ES6 classes that return JSX.



```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

- Different from functional components, class components must have an additional render( ) method for returning JSX.

Why Use Class Components?

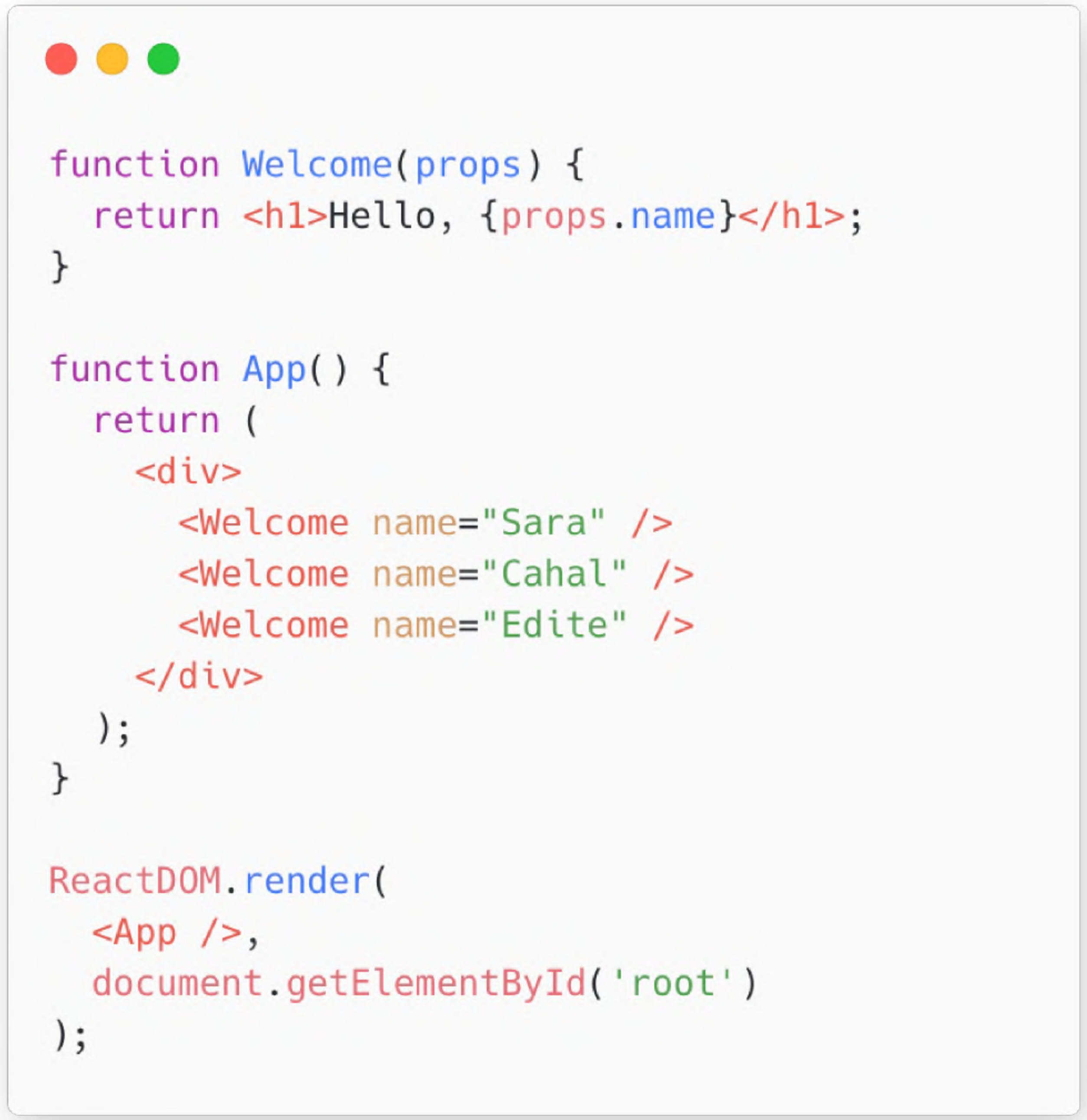
- We used to use class components because of "state". In the older versions of React (version < 16.8), it was not possible to use state inside functional components.
- Therefore, we needed functional components for rendering UI only, whereas we'd use class components for data management and some additional operations (like life-cycle methods).
- This has changed with the introduction of React Hooks, and now we can also use states in functional components as well.

A CLASS COMPONENT:

- is an ES6 class, will be a component once it ‘extends’ a React component.
- takes Props (in the constructor) if needed
- must have a render() method for returning JSX

Composing Components

- Components can refer to other components in their output.
- This lets us use the same component abstraction for any level of detail.
- A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.

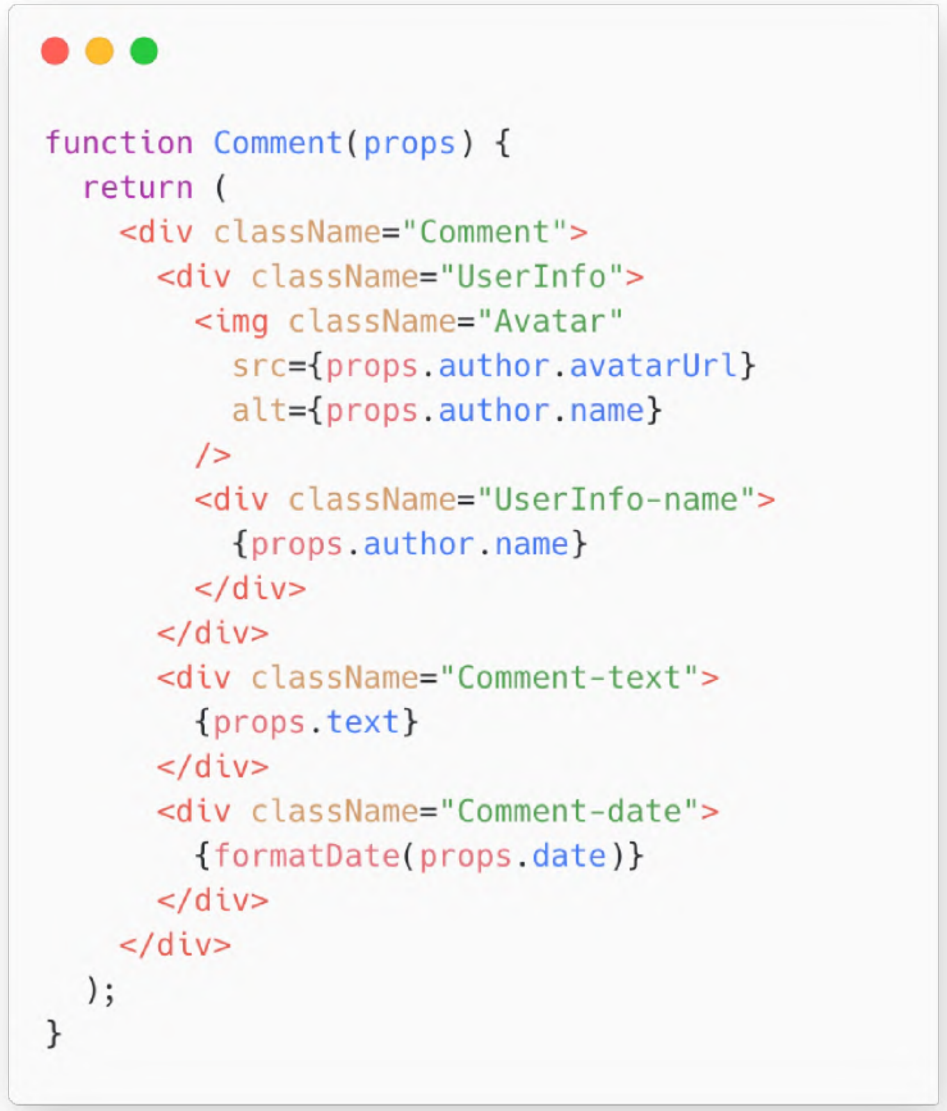


```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

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

Extracting Components

- Don't be afraid to split components into smaller components.

For example, consider this `Comment` component:

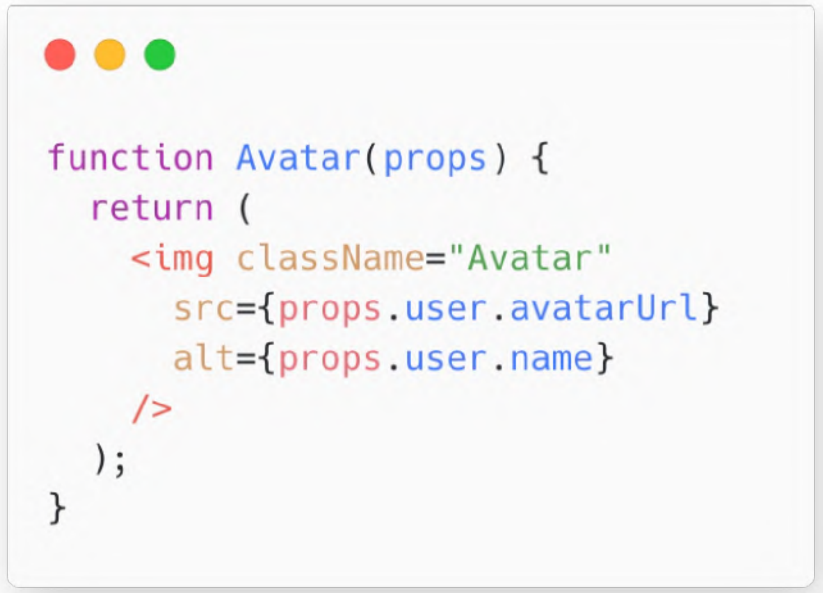


```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

It accepts `author` (an object), `text` (a string), and `date` (a date) as props, and describes a comment on a social media website.

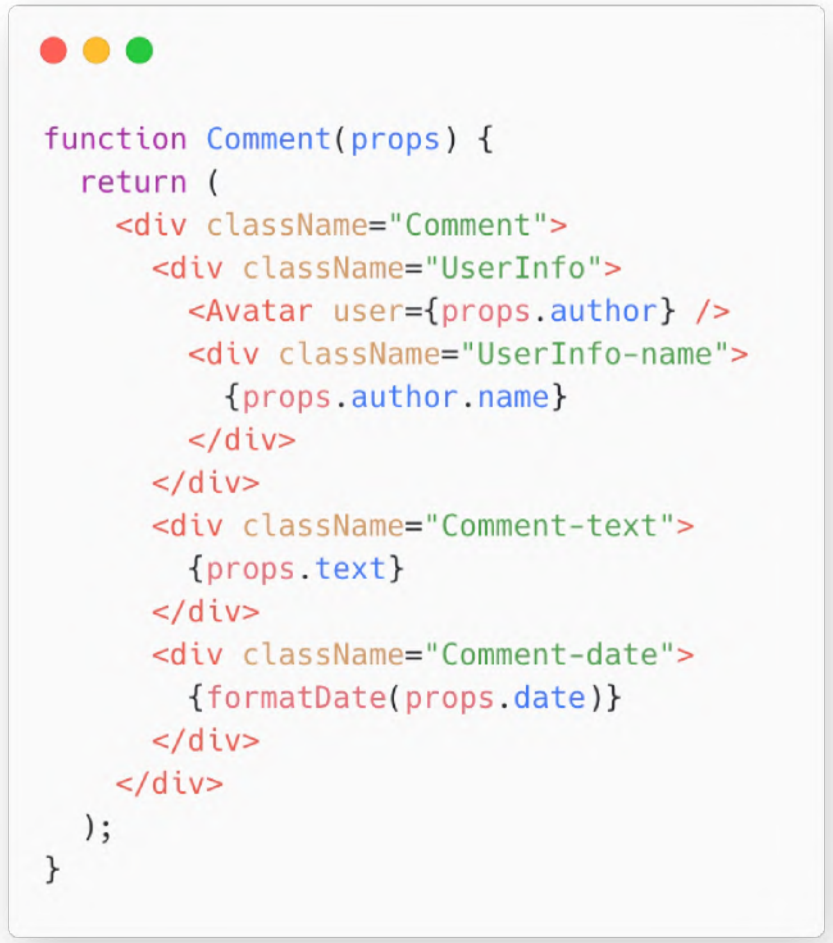
Extracting Components

First, we will extract `Avatar`



```
function Avatar(props) {
  return (
    <img className="Avatar"
      src={props.user.avatarUrl}
      alt={props.user.name}
    />
  );
}
```

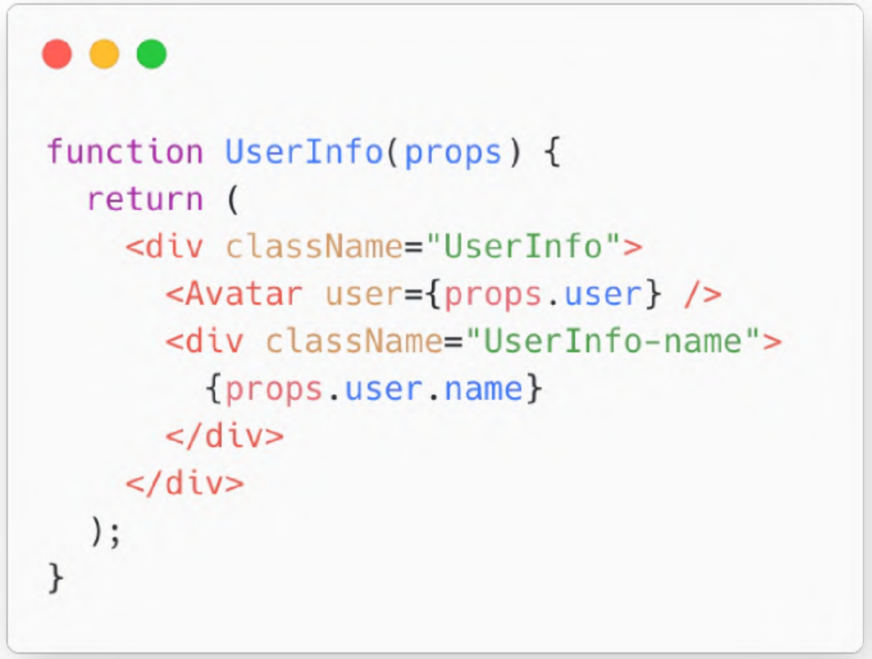
We can now simplify `Comment` a tiny bit:



```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={props.author} />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

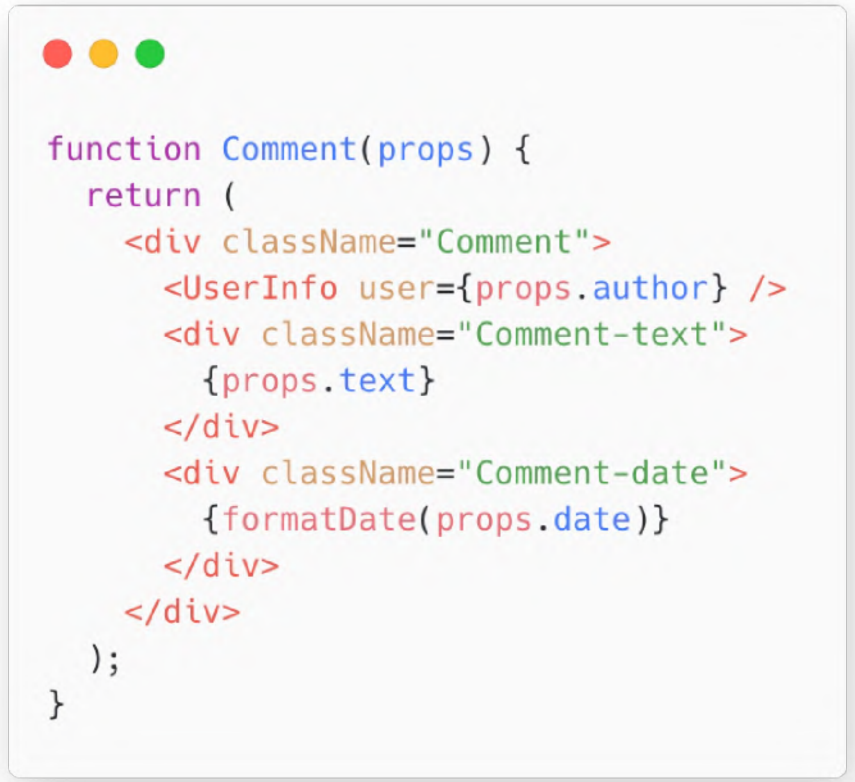
Extracting Components

We will extract a `UserInfo` component that renders an Avatar next to the user's name



```
function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">
        {props.user.name}
      </div>
    </div>
  );
}
```

This lets us simplify `Comment` even further:



```
function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```