React

Introducing JSX

Content

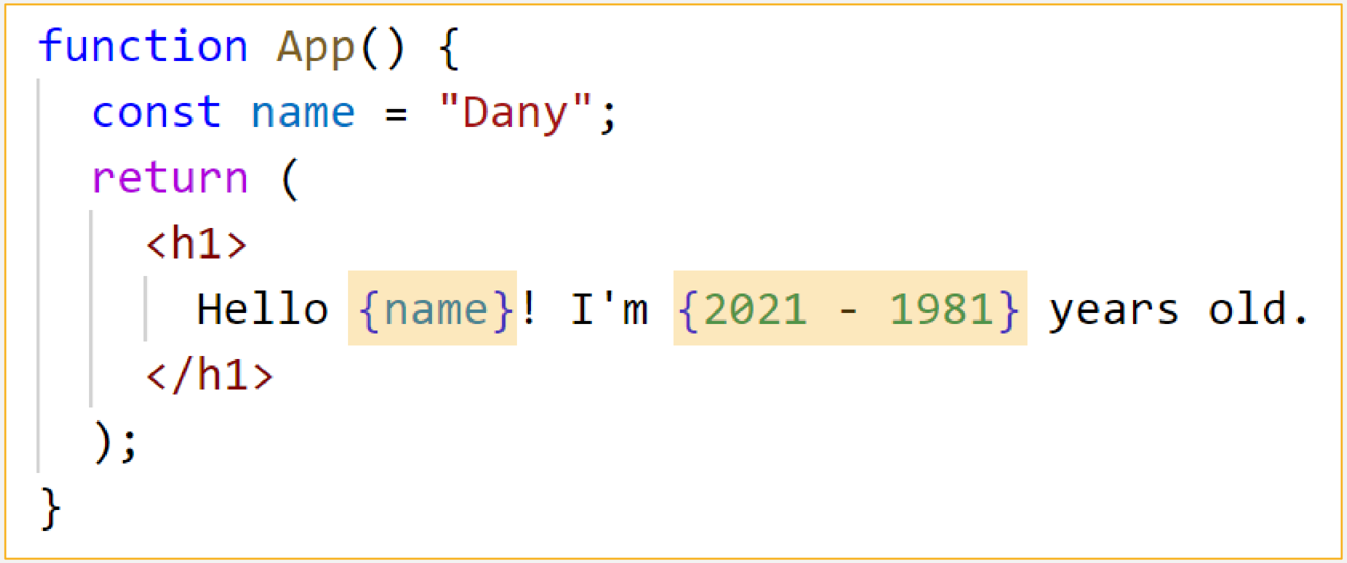
1. Why JSX?

- React embraces the fact that rendering logic is inherently coupled with other UI logic:
  - how events are handled.
  - how the state changes over time.
  - how the data is prepared for display.
- Instead of artificially separating *technologies* by putting markup and logic in separate files, React separates concerns with loosely coupled units called “components” that contain both.
- React doesn’t require using JSX, but most people find it helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages.

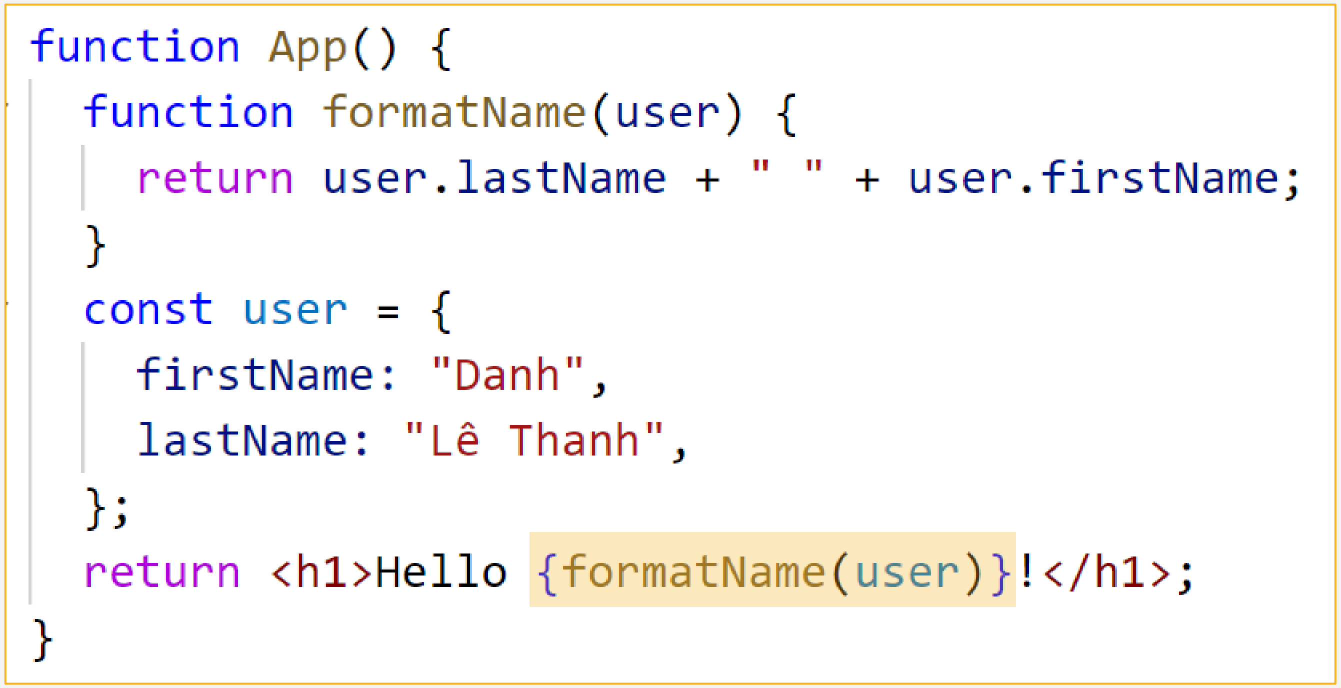
2. What JSX?

- Special dialect of JS (its no HTML)
- Browsers don’t understand JSX code! We write JSX then run tools to convert it into normal JS
- Very similar in form and function to HTML with a couple differences
- JSX vs HTML
  - Adding custom styling to an element uses different syntax
  - Adding a class to an element uses different syntax
  - JSX can reference JS variables

3. Embedding Expressions in JSX



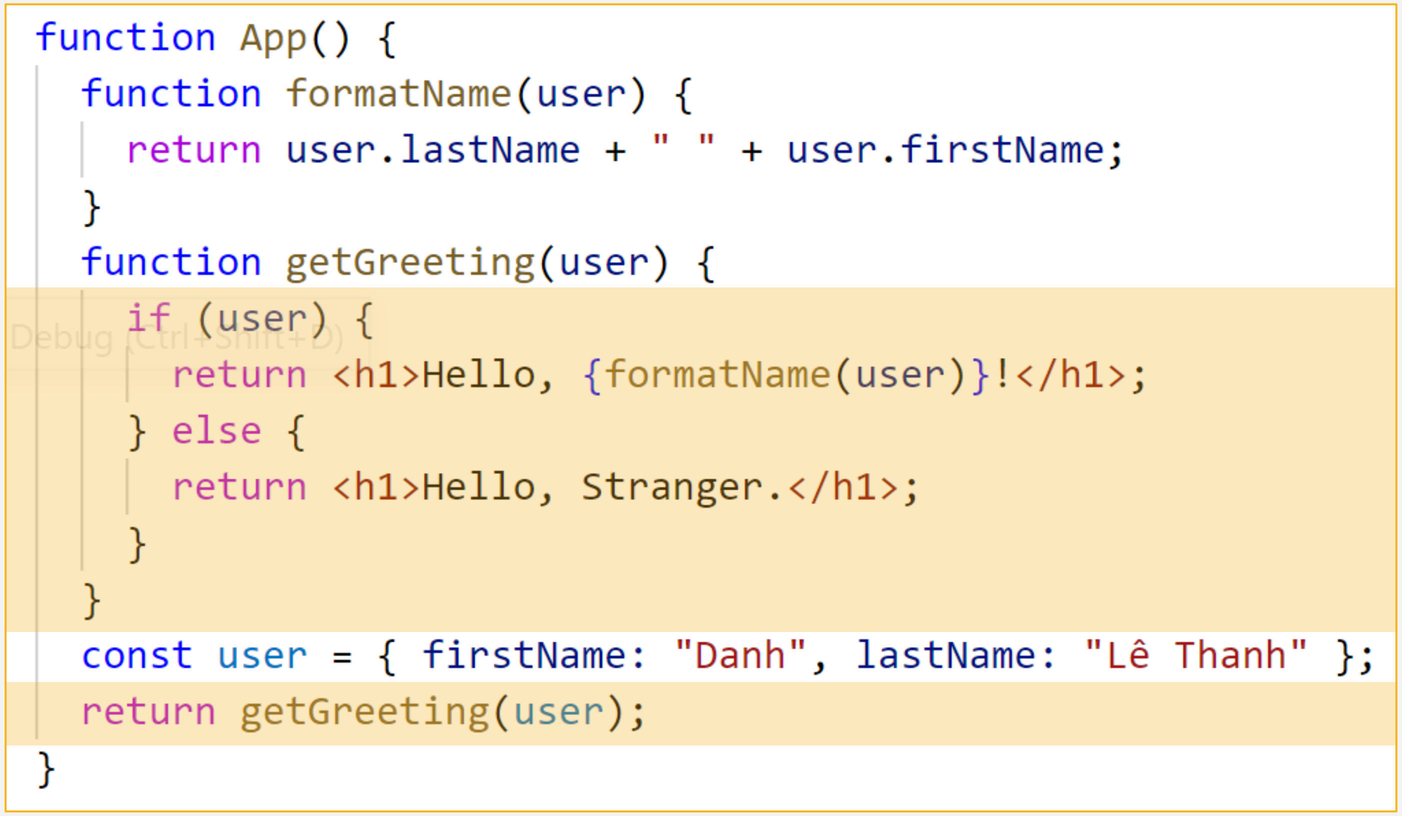
```
function App() {
  const name = "Dany";
  return (
    <h1>
      Hello {name}! I'm {2021 - 1981} years old.
    </h1>
  );
}
```



```
function App() {
  function formatName(user) {
    return user.lastName + " " + user.firstName;
  }
  const user = {
    firstName: "Danh",
    lastName: "Lê Thanh",
  };
  return <h1>Hello {formatName(user)}!</h1>;
}
```

4. JSX is an Expression

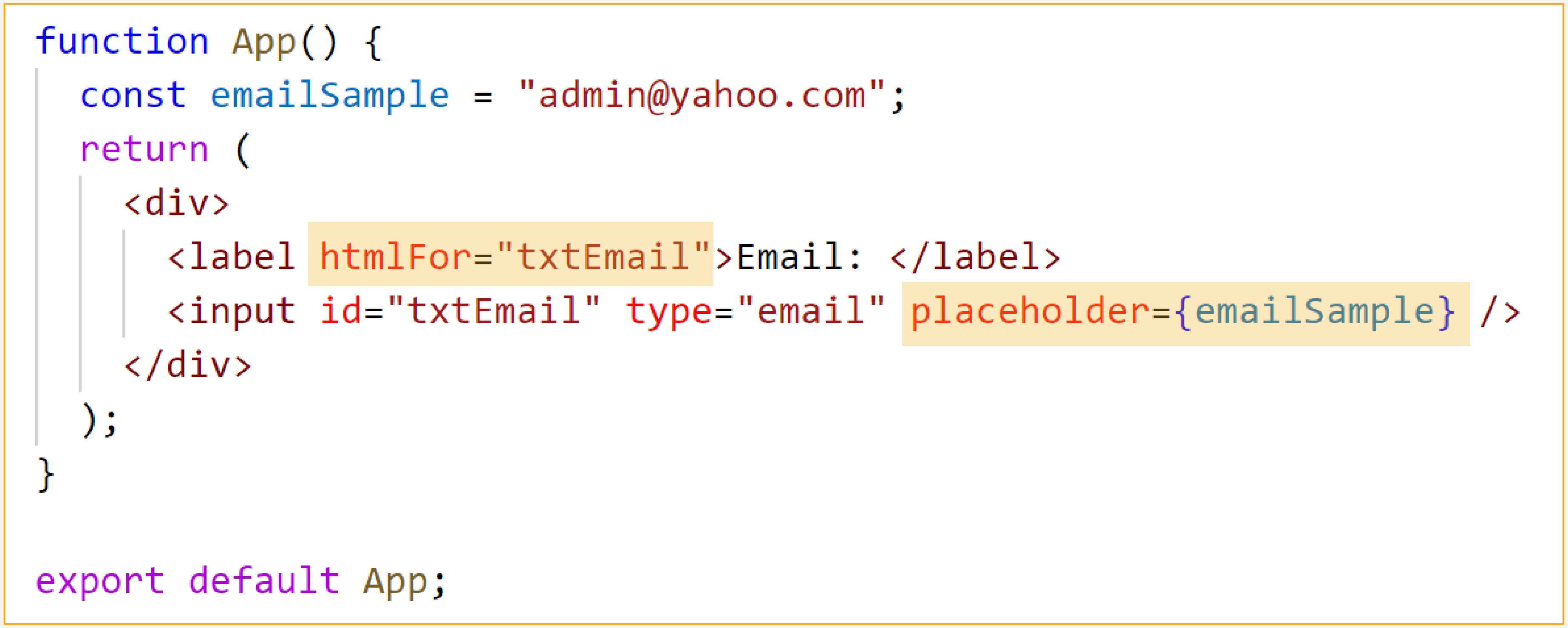
After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions.



```
function App() {
  function formatName(user) {
    return user.lastName + " " + user.firstName;
  }
  function getGreeting(user) {
    if (user) {
      return <h1>Hello, {formatName(user)}!</h1>;
    } else {
      return <h1>Hello, Stranger.</h1>;
    }
  }
  const user = { firstName: "Danh", lastName: "Lê Thanh" };
  return getGreeting(user);
}
```

5. Specifying Attributes with JSX

You may use quotes to specify string literals as attributes or curly braces to embed a JavaScript expression in an attribute.



```
function App() {
  const emailSample = "admin@yahoo.com";
  return (
    <div>
      <label htmlFor="txtEmail">Email: </label>
      <input id="txtEmail" type="email" placeholder={emailSample} />
    </div>
  );
}

export default App;
```

Note: Don’t put quotes around curly braces when embedding a JavaScript expression in an attribute.

6. JSX Represents Objects

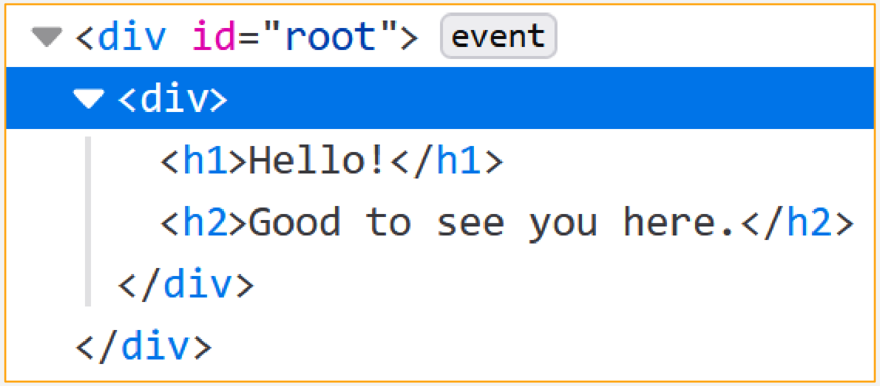
Babel compiles JSX down to React.createElement() calls

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

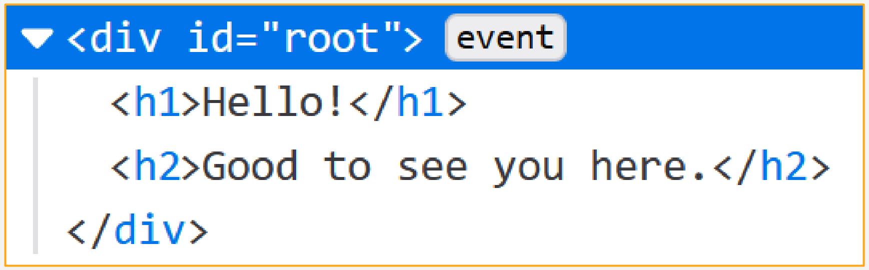
7. Specifying Children with JSX

```jsx
function App() {
  return (
    <div>
      <h1>Hello!</h1>
      <h2>Good to see you here.</h2>
    </div>
  );
}
```



```html
▼ <div id="root"> event
  ▼ <div>
      <h1>Hello!</h1>
      <h2>Good to see you here.</h2>
    </div>
  </div>
```

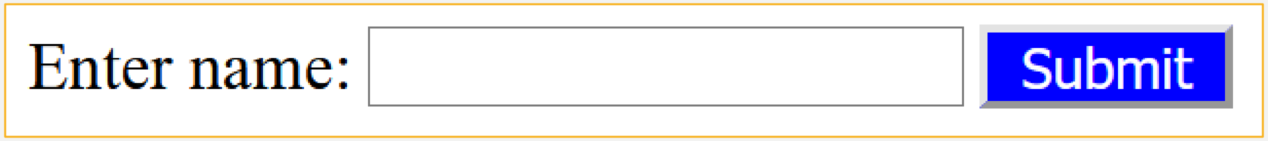
```jsx
import { Fragment } from "react";

function App() {
  return (
    <Fragment>
      <h1>Hello!</h1>
      <h2>Good to see you here.</h2>
    </Fragment>
  );
}
```



```html
▼ <div id="root"> event
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
```

```jsx
function App() {
  return (
    <>
      <h1>Hello!</h1>
      <h2>Good to see you here.</h2>
    </>
  );
}
```

8. HTML to JSX



- HTML

```html
<body>
  <label class="label" for="name">Enter name:</label>
  <input type="text" id="name">
  <button style="background-color: 🟦blue; color: ⬜white">Submit</button>
</body>
```

- JSX

```jsx
return (
  <div>
    <label className="label" htmlFor="name">Enter name:</label>
    <input id="name" type="text" />
    <button style={{backgroundColor: 'blue', color: 'white'}}>Submit</button>
  </div>
);
```

Differences In Attributes

- All DOM properties and attributes (including event handlers) should be lower camelCased. Ex: tabIndex, onClick, readOnly, …
- The exception is aria-* and data-* attributes, which should be lowercased. Ex: aria-label, data-id, …
- checked, defaultChecked: for controlled and uncontrolled component.
- value, defaultValue: for controlled and uncontrolled component.
- className ⇔ class in HTML
- htmlFor ⇔ for in HTML

- onChange:

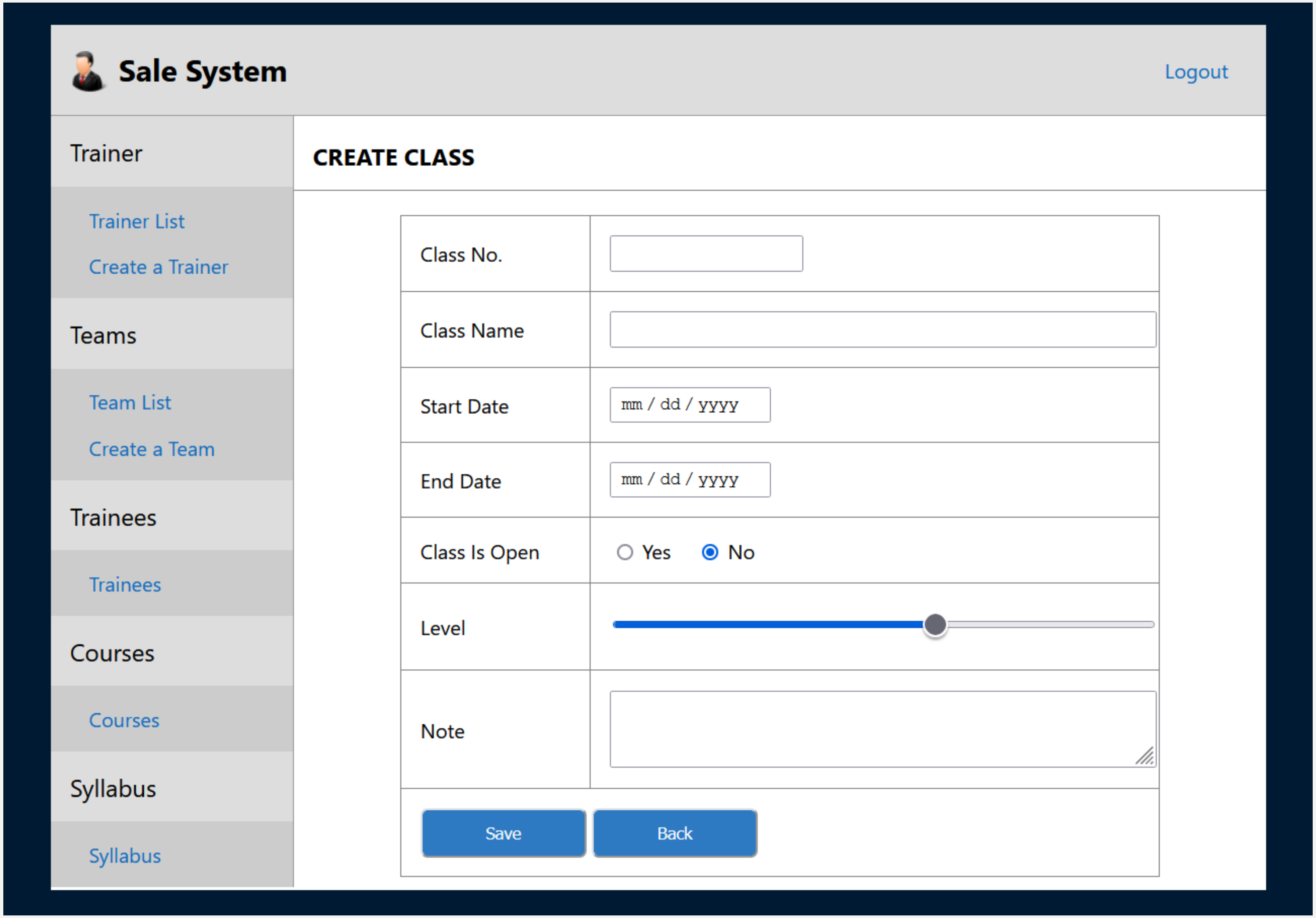
  Realtime change (in HTML, fire when data changed)
- selected:

  If you want to mark an <option> as selected, reference the value attribute of that option in the value of its <select> instead. Check out “The select Tag” for detailed instructions.
- style

  The style attribute accepts a JavaScript object with lower camelCased properties rather than a CSS string. Ex: backgroundColor, fontWeight, …
- Tag validation

  Validate opening & closing tag. Ex: <input type=“text” />

Exercise: HTML to JSX



Sale System

Logout

Trainer

- Trainer List
- Create a Trainer

Teams

- Team List
- Create a Team

Trainees

- Trainees

Courses

- Courses

Syllabus

- Syllabus

CREATE CLASS

Class No.	
Class Name	
Start Date	mm / dd / yyyy
End Date	mm / dd / yyyy
Class Is Open	Yes [✓ No]
Level	
Note	

Save Back

THE END