Section 4: React State & Working with Events

Brian E. Nguyen

October 24, 2021

Contents

1 Module Introduction

In this module, we will take a closer look at the following

- user interaction
  - this includes events like click, inputs, etc.
- state management
  - so far we can only build static applications where the state never changes, and that's not what we want

2 Listening to Events & Working With Event Handlers

- we only have one state in our application, which is the inital state
- let's start with clicks on a button which you want something to happen
- in the `ExpenseItem` component, let's add a button tag
  - this will be a temporary button with no styles so that we can practice with React states

```
// Expense Item
import ExpenseDate from "./ExpenseDate";
import Card from "../UI/Card";
import "./ExpenseItem.css";

const ExpenseItem = (props) => {
  return (
    <Card className="expense-item">
      <ExpenseDate date={props.date} />
      <div className="expense-item__description">
        <h2>{props.title}</h2>
        <div className="expense-item__price">${props.amount}</div>
      </div>
      <button>Change Title</button>
    </Card>
  );
};

export default ExpenseItem;
```

- the goal of this button is to change the title when the button is clicked
- React has a simple way of detecting button clicks
- on all built-in HTML elements, we have full access to native DOM events which we can listen to
- we will add a special prop in JSX called `on`. This can be followed by `onClick`, etc.
  - from there, we can add JavaScript logic to it

```
import ExpenseDate from "./ExpenseDate";
import Card from "../UI/Card";
import "./ExpenseItem.css";

const ExpenseItem = (props) => {
  return (
    <Card className="expense-item">
      <ExpenseDate date={props.date} />
      <div className="expense-item__description">
        <h2>{props.title}</h2>
        <div className="expense-item__price">${props.amount}</div>
      </div>
      <button onClick={() => {console.log('Clicked!')}}>Change Title</button>
    </Card>
  );
};

export default ExpenseItem;
```

- we typically want to define a function before the `return` statement

```
import ExpenseDate from "./ExpenseDate";
import Card from "../UI/Card";
import "./ExpenseItem.css";

const ExpenseItem = (props) => {
  const clickHandler = () => {
    console.log("Clicked!");
  };
  return (
    <Card className="expense-item">
      <ExpenseDate date={props.date} />
      <div className="expense-item__description">
        <h2>{props.title}</h2>
        <div className="expense-item__price">${props.amount}</div>
      </div>
      <button onClick={clickHandler}>Change Title</button>
    </Card>
  );
};
```

```
export default ExpenseItem;
```

- when we call the function, we don't add parentheses to it. Why? Because JavaScript will execute the function when the entire JSX line is parsed
- it's a convention that these functions end with `Handler`. Not everyone does this, but take note of it

3 How Component Functions Are Executed

- reacting to events is an important first step. How can we now change what shows up on the screen?
- we can add a new variable called `title` and pass it into the JSX code.
- now that we created a variable, we can use the `clickHandler` function to let us change the title

```
// ExpenseItem.js
import ExpenseDate from "./ExpenseDate";
import Card from "../UI/Card";
import "./ExpenseItem.css";

const ExpenseItem = (props) => {
  let title = props.title;

  const clickHandler = () => {
    title = "Updated!";
  };
  return (
    <Card className="expense-item">
      <ExpenseDate date={props.date} />
      <div className="expense-item__description">
        <h2>{title}</h2>
        <div className="expense-item__price">${props.amount}</div>
      </div>
      <button onClick={clickHandler}>Change Title</button>
    </Card>
  );
```

```
};

export default ExpenseItem;
```

- but if we were to actually click on the button, the title doesn't change. Why is that? The click handler still executed though and the value of `title` is "Updated!"
- the reason is simply because React doesn't work like this. We'll dive deeper into the reason later in the course
- what you need to know right now is that your component is a function; the only special thing about this function is that it returns JSX code
- we never explicity call our component functions. We just use them like HTML elements
- **the main point:** by using components like HTML elements, we make React aware of these component functions. When React evaluates the JSX, it will call the component functions
  - then the component functions will call any component functions inside of them. This process will repeat itself until there is no more JSX
- now, we need a way to tell React to reevaluate these component functions when we execute a handler

4 Working with 'State'

- *state* is not a React-specific concept
- we need to trigger a re-evaluation so that we can change our title
  - note that variables, like `title` are not triggered in the re-evaluation
  - React doesn't care about that. The component function doesn't get re-evaluated just because a variable changed
- to tell React that the component function should run again, we need to import something from the React library called `useState`
  - this is a function provided by the React library which allows us to define values as state, where changes to these values should reflect in the component function

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

- *inside* of our component function, we just call `useState` to use it
- *tip:* hook functions start with *use* and can only be called inside component functions, but not inside nested functions
- with `useState`, it wants a default-state value. We can assign an initial variable of `props.title` and pass it as an argument inside `useState`

```
// ExpenseItem.js
import React, { useState } from "react";

import ExpenseDate from "./ExpenseDate";
import Card from "../UI/Card";
import "./ExpenseItem.css";

const ExpenseItem = (props) => {
  useState(props.title);

  let title = props.title;

  const clickHandler = () => {
    title = "Updated!";
  };
  return
...
}
```

- `useState` not only lets us use the special variable in other places, but it also returns a function that we can call to assign a new value to that variable
  - `useState` returns an array where it has two values inside of it:
    1. the variable itself
    2. the updating function
  - you can use array-destructuring to retrieve these values

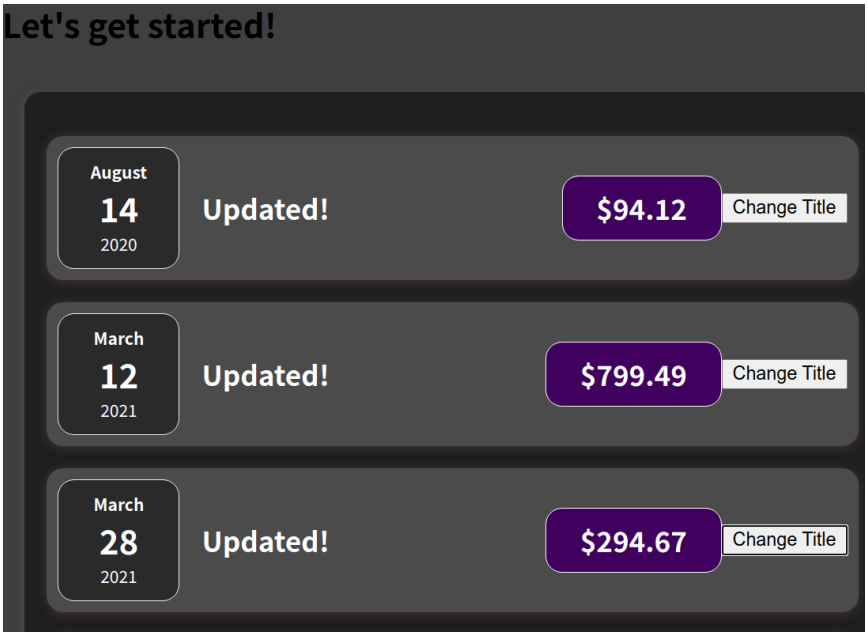
```
const [title, setTitle] = useState(props.title)
```

- this is now what the updated function looks like with `setTitle` in use

```
const ExpenseItem = (props) => {
  const [title, setTitle] = useState(props.title);

  const clickHandler = () => {
    setTitle("Updated!");
  };
  ...
}
```

- note that the `setTitle` function doesn't only assign the new value to a new variable, but it is also managed by React somewhere in memory because it's a special variable
- now when we click on a button, the title now updates it



5 A Closer Look at the "useState" Hook

5.1 Per-Component Basis

- `useState` registers some value as a state for the component in which it is being called
  - to be more precise, it registers it for a specific component instance

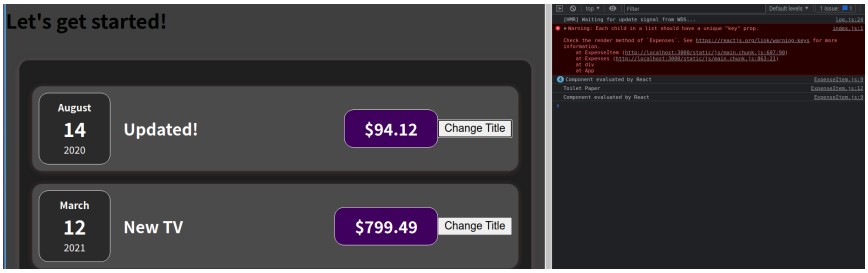
- in our `Expenses` component, our `ExpenseItem` component is used four times

```
// Expenses.js

// Note that the props passed in have 4 objects
const Expenses = (props) => {
  return (
    <Card className="expenses">
      {props.expenses.map(function (obj, i) {
        return (
          <ExpenseItem
            title={props.expenses[i].title}
            amount={props.expenses[i].amount}
            date={props.expenses[i].date}
          />
        );
      })}
    </Card>
  );
};
```

- the `useState` function is called 4 times when we create the 4 `ExpenseItem` components, and each are managed independently by React
  - that is why when we click on a button and title changes, the rest of the titles don't change, because they have their own state
- if we put a `console.log()` in our component, when we load up a page, this code will run 4 times; however, when we click on a button, it only runs once

```
// ExpenseItem.js
 const ExpenseItem = (props) => {
  const [title, setTitle] = useState(props.title);
  console.log("Component evaluated by React");
  const clickHandler = () => {
    setTitle("Updated!");
    console.log(title);
  };
  ...
```



- the takeaway is that *state is separated on a per-component basis*

5.2 Using `const`?

- why are we using `const` at the array-destructuring part when we eventually assign it a new value?
- note that we are not assigning a value with the = sign when we update the state
- instead, we call the state-updating function, and the value is managed somewhere else in React
- we never assign the value of `title` with the = operator; therefore, using `const` is fine

5.3 Closing Mentions

- knowing state is important, because in more complex apps, there could be times when a value doesn't update when it should
- using state is simple.
  1. You just register state with `useState`,
  2. you always get two values,
  3. you call the updating function, and
  4. you use the first value for outputs in JSX code

6 State Can Be Updated in Many Ways

Thus far, we update our state **upon user events**, (e.g. upon a click). That's very common but not required for state updates. *You can update states for whatever reason you many have.

Later in the course, we'll see HTTP requests where we want to update the state based on the HTTP response, but you could also be updating state because a timer (set with `setTimeout()`) expired for example