# 02 Working with States

In the last tutorial, we have built a neat e-commerce website with a navigation bar and a product list. However, a real application involves a lot of interaction with the user, which means the website should be reactive to the user's action.

In this tutorial, you will learn how to add interactivity with **state**. State refers to updatable data that would change over time. Defining and managing states in a web application is probably the most important topic of development with React.

## 1 React-Bootstrap

Before we dive into React state, let's first learn something about **Bootstrap**.

Building front-end website with pure CSS is sometimes difficult and time-consuming when you have to write styling information by yourself. As a result, there are many open-source libraries that can simplify the developement.

Bootstrap is one of the most popular framework for developing responsive UI design. In this tutorial, we will use some of the exising component in the `react-bootstrap` library to develop beautiful UI very quickly.

Before we start to use `react-bootstrap`, it's better to take a look at the official document first: https://react-bootstrap.github.io/getting-started/introduction/

### Installation

First, you should install the dependencies for `react-bootstrap` with npm. Paste and run the following command in your terminal:

```
npm install react-bootstrap bootstrap
```

You also need to import the Bootstrap CSS into your project. Paste this line of code at the begining of your `App.js` file:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

### Adding a Carousel

Now, we can start to use some components from Bootstrap. Let's add a **carousel** first. A **carousel** is a slideshow component for cycling through images or slides of text. You can find more with the document at https://react-bootstrap.github.io/components/carousel/


Let's first take a look at a example from the document:

```
import Carousel from 'react-bootstrap/Carousel';

function CarouselFadeExample() {
  return (
    <Carousel fade>
      <Carousel.Item>
        <img
          className="d-block w-100"
          src="holder.js/800x400?text=First slide&bg=373940"
          alt="First slide"
        />
        <Carousel.Caption>
          <h3>First slide label</h3>
          <p>Nulla vitae elit libero, a pharetra augue mollis interdum.</p>
        </Carousel.Caption>
      </Carousel.Item>
      <Carousel.Item>
        <img
          className="d-block w-100"
          src="holder.js/800x400?text=Second slide&bg=282c34"
          alt="Second slide"
        />

        <Carousel.Caption>
          <h3>Second slide label</h3>
          <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
        </Carousel.Caption>
      </Carousel.Item>
      <Carousel.Item>
```

```
        <img
          className="d-block w-100"
          src="holder.js/800x400?text=Third slide&bg=20232a"
          alt="Third slide"
        />

        <Carousel.Caption>
          <h3>Third slide label</h3>
          <p>
            Praesent commodo cursus magna, vel scelerisque nisl consectetur.
          </p>
        </Carousel.Caption>
      </Carousel.Item>
    </Carousel>
  );
}

export default CarouselFadeExample;
```

In this example, we import the `<Carousel>` component, inside of which there are two types of sub-components: `<Carousel.Item>` and `<Carousel.Caption>`. For each `<Carousel.Item>`, there is a `<img>` element for the slide image and a `<Carousel.Caption>` component for the text captions. With these components that already built in the `react-bootstrap` library, we can just make several modifications to develop our own carousel.

The `fade` attribute inside `<Carousel>` indicates that this carousel use crossfade transition.

`className="d-block w-100"` is equivalent to `display: block` and `width: 100%` in CSS. This is how you can use Bootstrap to directly style your HTML elements. We won't go deeper with this. You can read more about it at https://www.w3schools.com/bootstrap/default.asp.

Now, let's change a little bit and put it into our home page. Create a file called `CarouselFade.js` under the `components` folder and paste the following codes:

```
import Carousel from "react-bootstrap/Carousel";
import classes from "./CarouselFade.module.css";
import image1 from "../assets/carousel-img-1.png";
import image2 from "../assets/carousel-img-2.png";
import image3 from "../assets/carousel-img-3.png";

function CarouselFade() {
  return (
    <Carousel fade interval={1000} className={classes.carousel}>
      <Carousel.Item>
        <img className="d-block w-100" src={image1} alt="First slide" />
        <Carousel.Caption>
          <h3>Build your career with iBookStore</h3>
          <p>Unlock knowledge for your future.</p>
        </Carousel.Caption>
```

```
      </Carousel.Item>
      <Carousel.Item>
        <img className="d-block w-100" src={image2} alt="Second slide" />
        <Carousel.Caption>
          <h3>Unlimited resources</h3>
          <p>Explore books and workshops developed by experienced engineers</p>
        </Carousel.Caption>
      </Carousel.Item>
      <Carousel.Item>
        <img className="d-block w-100" src={image3} alt="Third slide" />
        <Carousel.Caption>
          <h3>Join our community</h3>
          <p>
            Ask your questions and share your ideas. We are here to help you.
          </p>
        </Carousel.Caption>
      </Carousel.Item>
    </Carousel>
  );
}

export default CarouselFade;
```

In this modified version, we first add a attribute called `interval` for the `Carousel` component. This is an API defined within the Bootstrap framework, which allows us to change the time to delay between images when automatically cycling them. Here we set the value to 1000, which means the interval is 1 second. You can find more API from the document: https://react-bootstrap.github.io/components/carousel/

Also, we need some custom styles with CSS modules. So we import a `carousel` class with `className` for the `Carousel` component.

For the rest of the codes, we just change the `<h3>`, `<p>` and image `src` to our own content.

Next, let's create the `CarouselFade.module.css` file under the same directory:

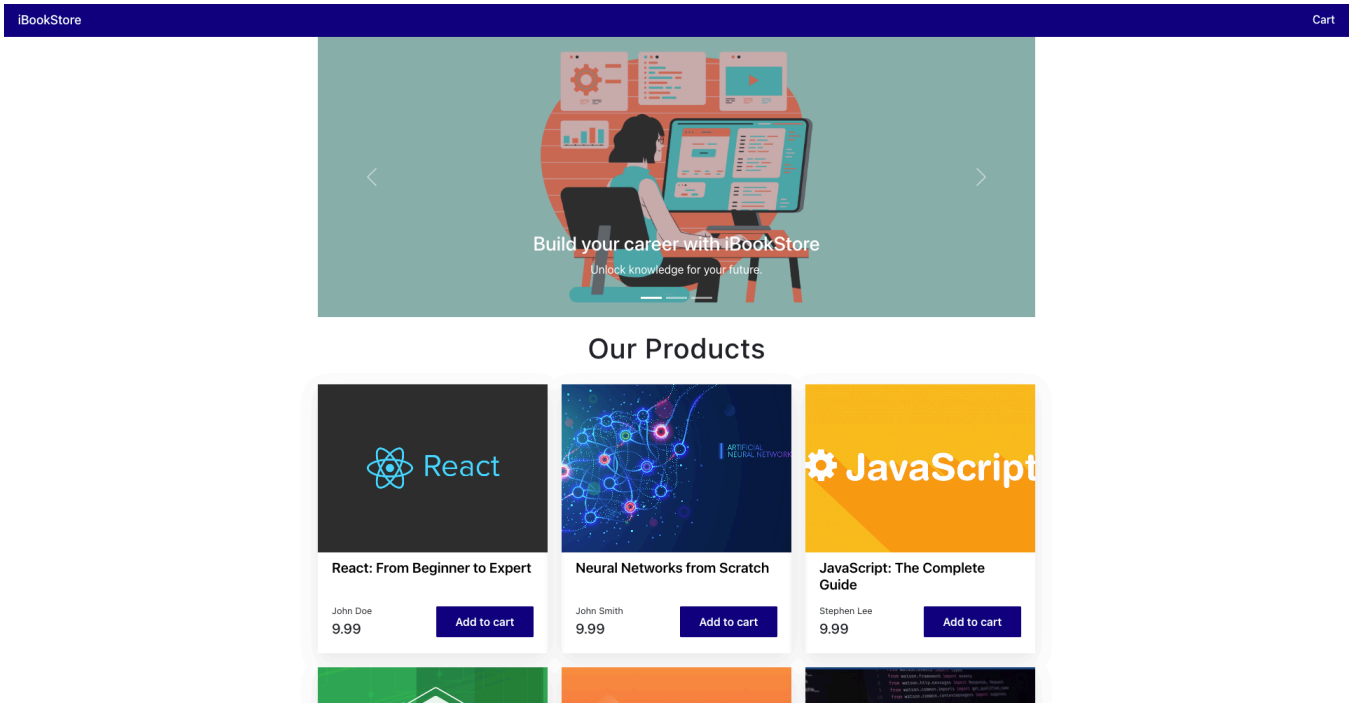
```
.carousel {
  width: 100%;
}

.carousel img {
  height: 400px;
  object-fit: cover;
  filter: brightness(80%);
}
```

The last step is importing the images for the carousel. You should download the images from our code repository: https://github.com/lvyin1122/ecommerce-tutorial-code/tree/02-working-with-state/src/assets

Put the images under the `src/assets/` directory.

After you finish these codes, it's time to put it on our home page. It is a very simple task, so you may try to do it yourself as an exercise. Feel free to reference the final codes if you get stuck.

Now, you should have a beatiful carousel at the top of your home page.




With just a few adjustments, we can attain a very pretty component. This is how the third-party component library saves your developement time.

## Adding a Pagination

Our current product list shows too many produt in one page. We are going to add a **pagination** component in this section to solve this problem, which can separate the whole product list into several pages.

The first thing is always about reading the official document: [https://react-bootstrap.github.io/components/pagination/](https://react-bootstrap.github.io/components/pagination/). Let's take a look at the official example first:




```
import Pagination from 'react-bootstrap/Pagination';

let active = 2;
let items = [];
```

```
for (let number = 1; number <= 5; number++) {
  items.push(
    <Pagination.Item key={number} active={number === active}>
      {number}
    </Pagination.Item>,
  );
}

const paginationBasic = (
  <div>
    <Pagination>{items}</Pagination>
    <br />

    <Pagination size="lg">{items}</Pagination>
    <br />

    <Pagination size="sm">{items}</Pagination>
  </div>
);

render(paginationBasic);
```

To create a complete pagination element, we can first create a list of `<Pagination.Item>`. Each of pagination item shows one page number. Therefore, to create the list, we can simpliy use a for loop to give each item a corresponding page number. In this example, three pagination components with different sizes are created. For each `<Pagination.Item>`, when the value of attribute `active` is true, the item will be highlighted, as shown in the image above. For this example, the active page number is fixed to 2.

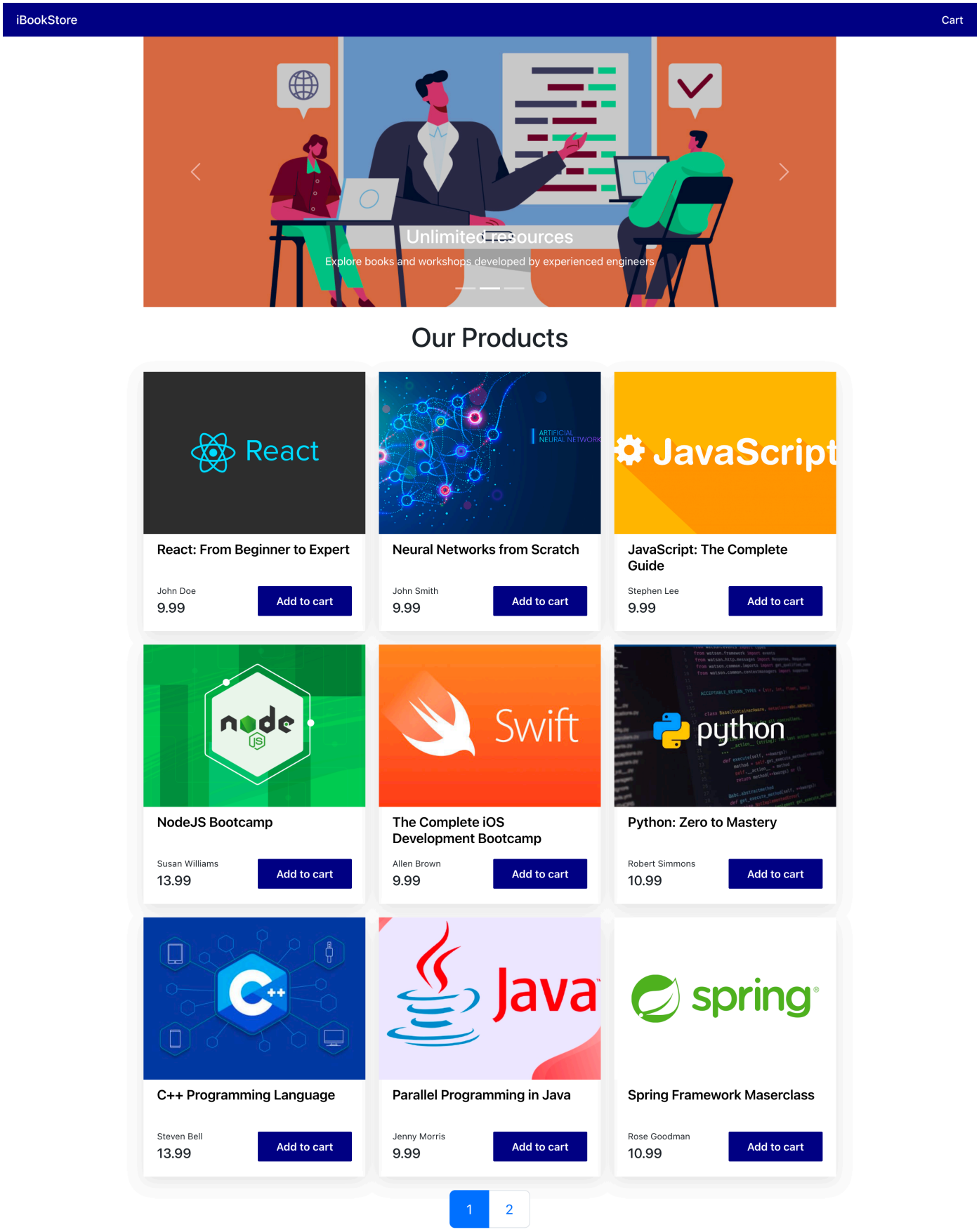
Now, let's put a pagination component into our project. You may try to finish it yourself first.

The complete code is shown below:

```
import React from "react";
import ProductItem from "../../components/ProductItem";
import CarouselFade from "../../components/CarouselFade";
import classes from "./Home.module.css";
import data from "../../productsData";

import Pagination from "react-bootstrap/Pagination";

function Home() {
  // Active page number
  let active = 1;
  // Create a list
  let items = [];
  // limit = 6, meaning we will have 6 products in one page at most
  const limit = 6;
  // Compute the number of pages from the number of products
  const pageNum = Math.ceil(data.length / limit);
```

```
  // Create a list of Pagination.Item using a for loop
  for (let number = 1; number <= pageNum; number++) {
    // Use push() method to add new item to the list
    items.push(
      <Pagination.Item key={number} active={number === active}>
        {number}
      </Pagination.Item>
    );
  }

  // Create the Pagination component
  const pagination = <Pagination size="lg">{items}</Pagination>;

  const productsList = data.map((data) => (
    <ProductItem
      key={data.id}
      id={data.id}
      name={data.name}
      author={data.author}
      price={data.price}
      image={data.image}
    />
  ));

  // Put the pagination below product list.
  return (
    <div className={classes.home}>
      <div className={classes.container}>
        <CarouselFade />
        <h1>Our Products</h1>
        <div className={classes.productsContainer}>{productsList}</div>
        {pagination}
      </div>
    </div>
  );
}

export default Home;
```

You should now see a Pagination showing two pages at the bottom of our home page. But currently, it is just a static UI without any functionalities. In the following sections, we will learn how to generate some interactive elements in our project by learning about two core concepts of React: state and hooks.




# 2 Managing States with "useState" Hook

"State" means any data or properties that need to be tracking in your application.

Let's first look at a simple example:

```
import React from "react";

function Counter() {
  let count = 0;

  return (
      <button>You have clicked {count} times.</button>
  );
}

export default Counter;
```

Here, we want to make a counter that counts the number of clicks the button from user.

The application should be working like this:

1. Handle the click event when the user click the button
2. Change the value of `count`
3. Re-render the button element to show the latest value of `count`

The React-style way of doing these tasks is using the `useState` Hook. Let's take a look at the codes first:

```
import React from "react";
import { useState } from "react";

function Counter() {
  // Initilize the count state with 0
  const [count, setCount] = useState(0);

  const clickHandler = () => {
    // Update the value by adding one
    setCount(count + 1);
  };

  return (
    // Add clickHandler to the button
    <button onClick={clickHandler}>You have clicked {count} times.</button>
  );
}

export default Counter;
```

First, we should use the `useState` hook to initilize the state of count.

The `useState` hook receives one argument: the initial value of the state. Here, we initilze the `count` state as a number 0. A state can also be initilized in any other types, such as string and object.

The `useState` hook will return a pair of values: (1) the state variable: `count`, and (2) the function that can update the value of the state: `setCount`. In this example, we get a `count` state with a value of 0 and the `setCount` function that updates it.

When we want to update the state to a new value, we can now simpliy call the `setCount` function. Say we want to set the `count` to 2, we just write `setCount(2)`.

The advantage of `useState` for changing variables is: when you update the state with the update function, React will "know" it and re-render the component at once.

Now, we can use the `setCount` function to describe the actions after the user clicks the button. Here, the `onClick` attribute of the button must receive a function variable as the event handler, so we should define a handler function `clickHandler` first. We let the handler function update the value of count by adding one to original value.

Instead of pass the state value to `setCount` function, you can also pass a function. This is often used to gurantee the state is changed strictly based on the previous state:

```
const clickHandler = () => {
  setCount(prevState =>  prevState + 1);
};
```

`prevState => prevState + 1` is an anonymous function (arrow function) that receives a `prevState` parameter and returns the value of `prevState + 1`. To check with more examples, you can go to https://reactjs.org/docs/hooks-reference.html#usestate.

After the value of `count` is change by `setCount`, React will re-render the page with the new value.

The magic of React is it will evaluate the differences and only re-render the parts that are changed instead of rebuilding the whole page, which lowers the cost and makes things faster. This is because React applys the virtual DOM to compare the differences between versions. Please reference the official documentation if you are intereted in the internals of React.

## Adding State to Pagination

Now, it's time to add state management to our Pagination componenet. It is quite challenging but still you can have a try first.

Let's build it step by step.

The first thing we need to do is add a state to the `active` attribute, so that we can change the active page. Let's name it as `page` to indicate the current page number.

```
const [page, setPage] = useState(1);
```

When when click on a `Pagination.Item`, it should set the page number to the page number shown on it, which is defined during the for loop. Therefore, we can write the codes like this:

```
for (let number = 1; number <= pageNum; number++) {
  items.push(
    <Pagination.Item key={number} active={number === page} onClick={() =>
 setPage(number)}>
      {number}
    </Pagination.Item>
  );
}
```

Note that, for different `Pagination.Item` generated by the loop, the handler function passed by `onClick` has different value of `number`, so that each pagination item is binded with its page number.

The next thing we need to do is make the product list react to the changes of page number. We can simpliy do a slicing to the data list according to the value of `page`. Here's how it works:

```
// Compute the start position of the products that should be shown in current page
const start = 0 + (page - 1) * limit;
const currentData = data.slice(start, start + limit);

const productsList = currentData.map((data) => (
  <ProductItem
    key={data.id}
    id={data.id}
    name={data.name}
    author={data.author}
    price={data.price}
    image={data.image}
  />
));
```

At last, remember that it is preferred to build you custom component and make it reusable, which can also bring simplicity and readablility to our codes. We can define our customized Pagination componenet as follows:

```
import React from "react";
import { Pagination } from "react-bootstrap";

function PaginationBasic(props) {
  let items = [];
  for (let number = 1; number <= props.pageCount; number++) {
    items.push(
      <Pagination.Item
        key={number}
        active={number === props.page}
        onClick={() => props.setPage(number)}
      >
        {number}
      </Pagination.Item>
```

```
    );
  }

  return (
    <div>
      <Pagination size="lg">{items}</Pagination>
    </div>
  );
}

export default PaginationBasic;
```

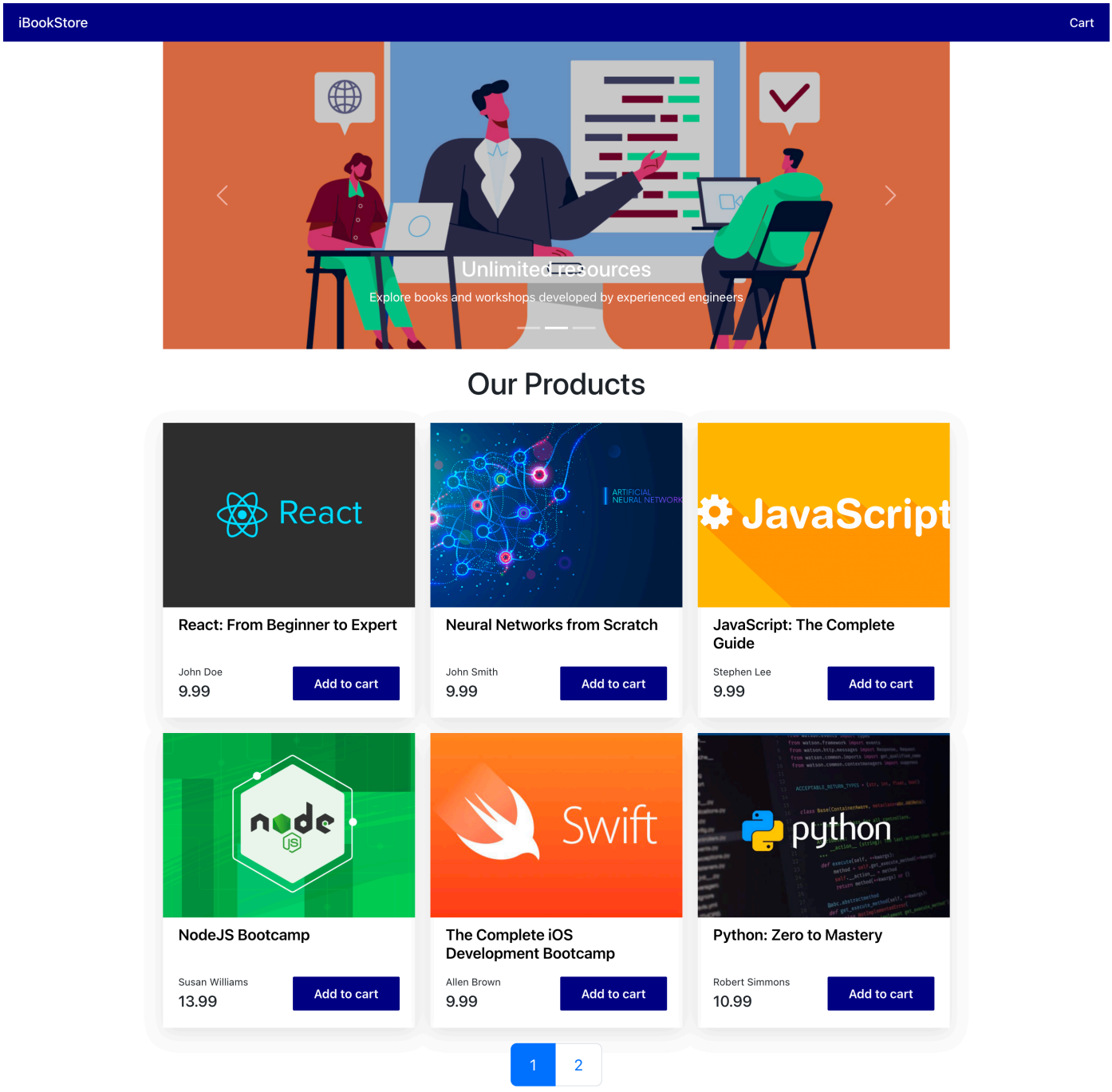
You may notice that we need to pass three props to this component: `pageCount`, `page` (the active page), and the `setPage` function. We should pass these props from `Home.js`:

```
import React, { useState } from "react";
import ProductItem from "../../components/ProductItem";
import CarouselFade from "../../components/CarouselFade";
import PaginationBasic from "../../components/PaginationBasic";
import classes from "./Home.module.css";
import data from "../../productsData";

function Home() {
  // Initialize the state indicating the active page
  const [page, setPage] = useState(1);
  // Each page contains 6 items
  const limit = 6;
  // Compute the number of pages from the number of items
  const pageCount = Math.ceil(data.length / limit);

  // Compute the products should be shown in current page
  const start = 0 + (page - 1) * limit;
  const currentData = data.slice(start, start + limit);

  const productsList = currentData.map((data) => (
    <ProductItem
      key={data.id}
      id={data.id}
      name={data.name}
      author={data.author}
      price={data.price}
      image={data.image}
    />
  ));

  return (
    <div className={classes.home}>
      <div className={classes.container}>
        <CarouselFade />
```

```
        <h1>Our Products</h1>
        <div className={classes.productsContainer}>{productsList}</div>
        <PaginationBasic pageCount={pageCount} page={page} setPage={setPage}/>
      </div>
    </div>
  );
}

export default Home;
```

If we want to change the number of products shown in one page, or create the same pagination for other pages, we don't need to change the codes inside the `PaginationBasic.js`.

Now, we have built a home page with a workable pagination bar. You can click on the page number to switch between pages.




You may notice a very interesting thing: the parent component `Home` tells the current active page number to the child component `PaginationBasic` through the `page` prop, while the child component `PagniationBasic` tells its parent component `Home` that the page number is changed by calling the `setPage` function passed by prop. This is a very common way of building a two-way communication between a pair of parent and child components.

However, this approach of passing information is sometimes verbose and inconvenient if you have to pass them through many components in the middle. Another case is that sometimes many components need to access the same global data. We will see how we can solve this problem in the next tutorial.

The complete code of Tutorial 2: [https://github.com/lvyin1122/ecommerce-tutorial-code/tree/02-working-with-state](https://github.com/lvyin1122/ecommerce-tutorial-code/tree/02-working-with-state)