01 Getting Started with React.js

In this tutorial, you will learn React by building an e-commerce website step-by-step.

To learn React better and understand this tutorial faster, you'd better first have some basic knowledge about web development using HTML, CSS, and JavaScript. Here are some useful resources to have a quick review of the knowledge, make sure you can have a basic idea about these concepts:

- HTML: https://www.w3schools.com/html/
- CSS: https://www.w3schools.com/css/
- JavaScript (espectially **functions** and **modules**): https://www.w3schools.com/js/

It is also recommended to go with our tutorials first and reference these websites when you need help.

Each of the section covers a concept of React. You will first be given some simple examples of the concept from React. Then you will learn how it will help us to build specific function in the application.

There are two ways of learning this tutorial. You can either write the codes with the tutorial step-by-step, or jump to read the complete codes first and go back to this tutorial when you get stuck with some part. The complete code is given at the end of each tutorial.

1 React Introduction

React is a open-srouce front-end JavaScript library for building user interfaces, developed and maintained by Facebook.

As a modern front-end framework, React can boost the process of development with the following advantages:

- Declarative
- Easy to learn and use
- Reusable components
- Cross-platform

2 Getting Started

Create a React Project

Before we start building application with React, make sure you have installed Node.js and npm (Node Package Manager) on your computer. You can download the installer from this page: https://nodejs.org/en/download/

After installation, you can open your command prompt (or Terminal on Mac) and enter the following:

```
node -v
npm -v
```

The system should display the versions of Node and npm.

Now, it's time to create your first React application by running the following command:

```
npx create-react-app ecommerce
```

`npx` is an npm package runner. Here, we use `npx` to execute the `create-react-app` package, which creates a boilerplate project with React for us.

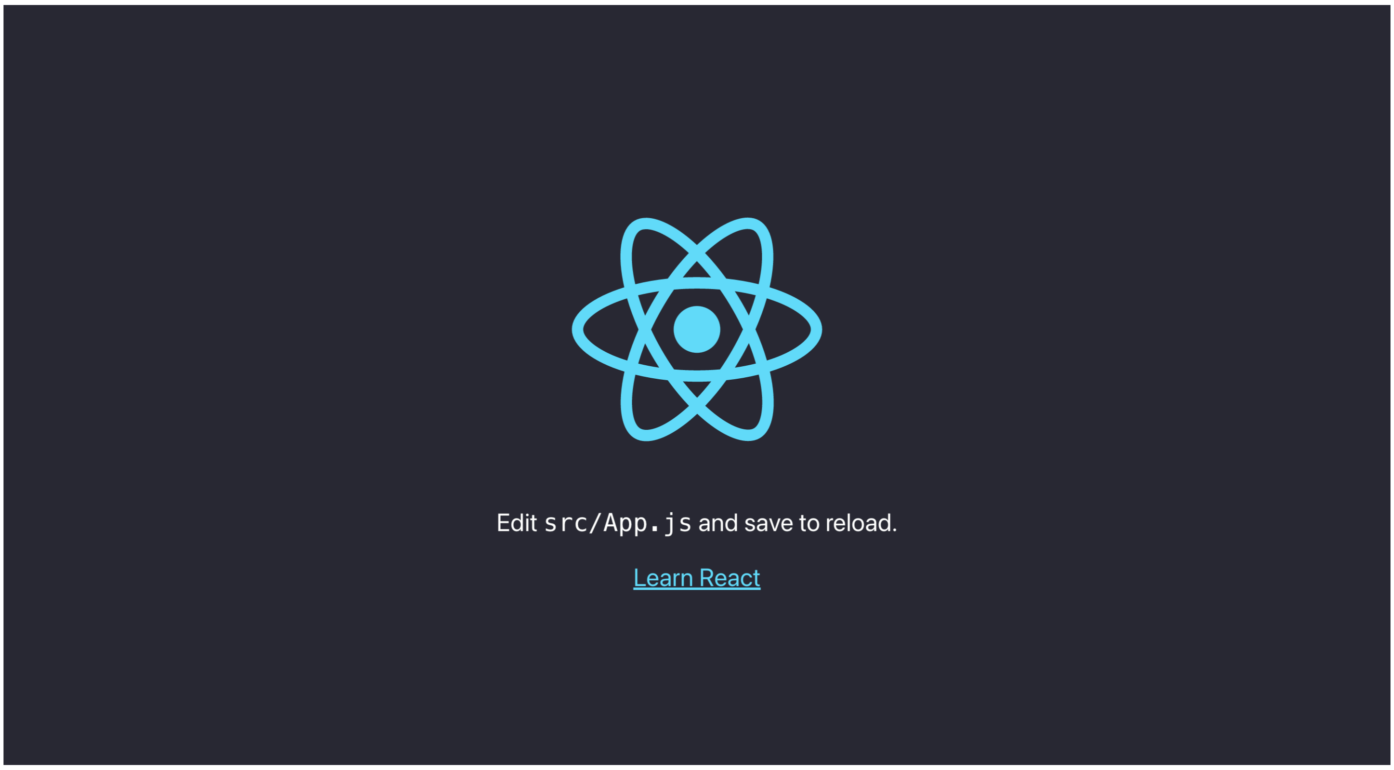
`ecommerce` is the name of the project we created. You can also change it to other names if you like.

Run the React Application

Now, you can move to the directory and run the application.

```
cd ecommerce
npm start
```

Your computer will pop up a new browser window called *React App*. You can also access the website by entering the address `localhost:3000`. Right now, the website looks like this:



We will build our own web application in the following steps.

3 JSX

JSX is a syntax extension to JavaScript.

In React, we use JSX to describe what the UI should look like instead of using pure HTML. It allows us to write HTML directly in JavaScript. Also, we can put JavaScript expressions into the HTML elements. `const`

Here's an example:

```
const element = <h1>Hello World!</h1>;
```

This statement declares a constant variable called `element`, whose value is written with JSX. It looks like HTML but actually produces React "elements" that can be rendered by React. We can also embed JavaScript expressions inside JSX using curly braces:

```
const hello = "Hello World!";
const element = <h1>{hello}</h1>;
```

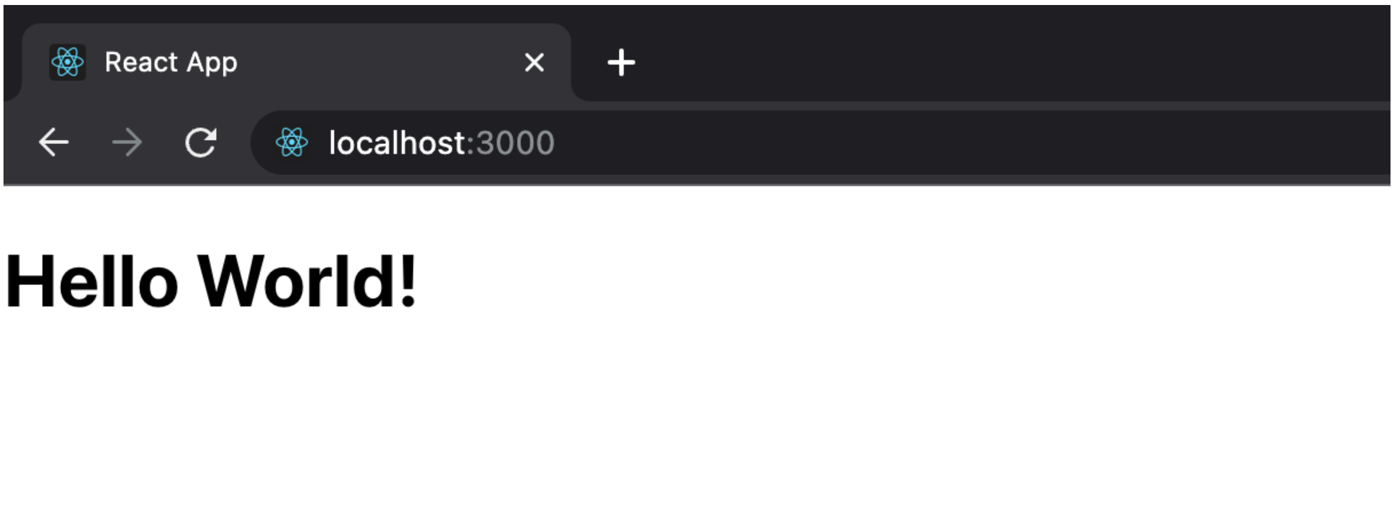
Here, we embed the value of the string variable `hello` into an `<h1>` element with curly braces, which can produce the same result as the previous example.

Now, it's time to see how we can use JSX in the project we just created. Let's find the file called `App.js` under the `ecommerce/src` folder.

To make it simple, delete all the existing codes and paste the following:

```
function App() {
  const hello = "Hello World!"
  return (
    <div>
      <h1>{hello}</h1>
    </div>
  );
}

export default App;
```

Save the file, and you should see the text "Hello World!" displayed on the website.



Hello World!

In this example, we created a JavaScript function called `App`, which is a root **component** of our application. This component returns a `<div>` element with JSX syntax, which is exactly the same as our previous example. The returned elements will be rendered by React and displayed on the screen.

We can also write it like this, where we create a new element variable and embed it to the `<div>` element:

```
function App() {
  const hello = "Hello World!";
  const element = <h1>{hello}</h1>;
  return (
    <div>
      {element}
    </div>
  );
}

export default App;
```

4 Components and Props

Components and **props** are the core concepts of React. They are the foundation upon which you build your user interface.

Component allows you to combine your HTML, CSS styling, and JavaScript into one custom reusable UI element.

We just created the root component `App` for our application. Now, let's learn more about components by building a simple UI element on our website: a top navigation bar.

First, let's create a new folder called `components` under the `src` folder. We will put all of our components in this folder.

Now, we can create a new JavaScript file called `Navbar.js` and paste the following:

```
function Navbar() {
  return (
    <div>
      <div>iBookStore</div>
      <div>Cart</div>
    </div>
  );
}

// Export the component
export default Navbar;
```

This `Navbar` component will show two pieces of text. One is for our brand name (iBookStore as an example), and one is for a button that should redirect the website to the shopping cart.

Now, we can import the `Navbar` component into `App` and put it into the `return` function to render it.

```
// Import the component
import Navbar from "./components/Navbar"

function App() {
  return (
    <div>
      <Navbar/>
    </div>
  );
}

export default App;
```

In modern React, every componet is usually defined as a function. A React component can accept input data through arguments, just like a function. The object argument of a component is called "props" (which stands for properties).

Let's take a look at how we can use props in `Navbar.js`:

```
function Navbar(props) {
  return (
    <div>
      <div>{props.brand}</div>
      <div>Cart</div>
    </div>
  );
}
```

Here, the `props` is an object argument that passes data to the `Navbar` component. In the returned element, we retrieve a `brand` property from the props, which should be specified when we want to use the component.

Let's specify the value of the brand property from `App` component.

```
import Navbar from "./components/Navbar"

function App() {
  return (
    <div>
      <Navbar brand="iBookStore"/>
    </div>
  );
}

export default App;
```

We add a `brand` attribute to the `<Navbar>`, so that the `Navbar` component will receive the string value through `props`.

Now, if we want to modify the brand name or create another website with the same navigation bar, we don't need to change the codes in `Navbar.js`. This is how components and props make it reusable and improve development speed. We will learn more about it in the following tutorials.

5 CSS Styling

Currently, our `Navbar` only shows two lines of texts. To make it look like a navigation bar, we need to add some styles.

In React, there are many ways to add styling information to the UI elements. In our project, we will choose a more common and convenient method: **CSS Modules**.

Create a new file called `Navbar.module.css` and paste the following codes:

```css
.navbar {
  height: 50px;
  background-color: #000080;
  display: flex;
  padding: 0 20px;
  align-items: center;
  justify-content: space-between;
}

.brand {
  font-weight: 500;
  font-size: large;
  cursor: pointer;
  text-decoration: none;
  color: white;
}

.cart {
  font-weight: 500;
  cursor: pointer;
  text-decoration: none;
  color: white;
}
```

The syntax of CSS Modules has no difference with CSS. Each block of code is a CSS **class**. And each class is used to style the corresponding element we created perviously.

Now, let's import these styles to our `Navbar` component:

```js
import React from "react";
// Import the styles from the CSS file
import classes from "./Navbar.module.css";
```

```
function Navbar(props) {
    return (
      <div className={classes.navbar}>
        <div className={classes.brand}>{props.brand}</div>
        <div className={classes.cart}>Cart</div>
      </div>
    );
  }

export default Navbar;
```

Here, we first import the stylesheet as a `classes` object variable. Then we assign the CSS classes to the corresponding elements through the `className` attribute. For example, if we want to use the `.navbar` class style, we just write `className={classes.navbar}`.

Now, we have a neat navigation bar on the top of our website.

6 Lists

Sometimes, we need to render one component for multiple times in an ordered format. For example, a list of products. In React, we generally use the JavaScript `map()` method. Let's first look at a simple example:

```
const numbers = [1, 2, 3, 4, 5]
const list = numbers.map((number)
=>
  <p>{number}</p>
);
```

In this example, the `map()` method will map each of the number in the `numbers` list to a `<p>` element. The result is equivalent to this:

```
<p>1</p>
<p>2</p>
<p>3</p>
<p>4</p>
<p>5</p>
```

Similarly, we're gonna add a product list to our home page.

For this tutorial, the products we're selling are e-books. Each e-book has 5 properties: id, name, author, price, and image.

First, let's create a `ProductItem` component under the `components` file:

```
import React from "react";
import classes from "./ProductItem.module.css";
```

```
function ProductItem(props) {
  return (
    <div className={classes.productItem}>
      <img src={props.image} alt="" />
      <div className={classes.productInfo}>
        <h1>{props.name}</h1>
        <div className={classes.infoBottom}>
          <div className={classes.infoLeft}>
            <span className={classes.author}>{props.author}</span>
            <span className={classes.price}>${props.price}</span>
          </div>
          <button>Add to cart</button>
        </div>
      </div>
    </div>
  );
}

export default ProductItem;
```

Here, we defined a component called `ProductItem`. It requires product properties including `image`, `author`, and `price`. It uses the CSS classes to style the elements.

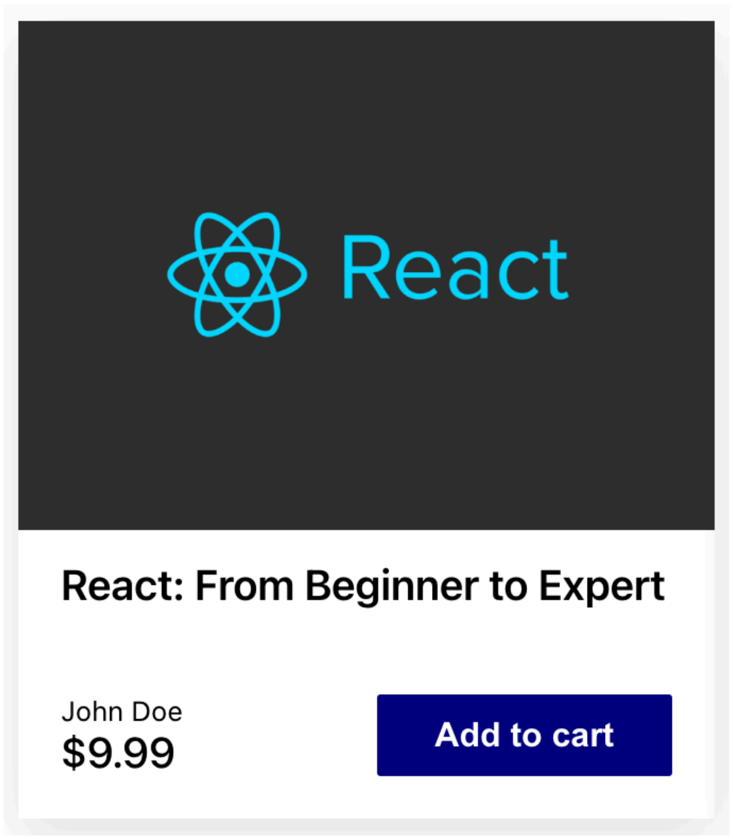
You may feel overwhelmed with this much of codes. Don't be afraid. Read it through and you will find that it is just a complicated example with the same concepts we just learned. It is also good to read the complete codes first and check the details afterwards.

Next, paste the following styling codes to a new file called `ProductItem.module.css` under the same folder. You may look up the details if you want to learn more about CSS.

```
.productItem {
    box-shadow: 0 .5rem 1.5rem rgba(0,0,0,.1);
    max-height: 383.5px;
}

.productItem img {
    width: 100%;
    height: 240px;
    object-fit: cover;
}

.productInfo {
    display: flex;
    flex-direction: column;
    justify-content: space-between;
    padding: 10px 20px 20px 20px;
    gap: 10px;
}
```

```
.productInfo h1 {
    width: 100%;
    height: 54px;
    margin: 0;
    font-size: 20px;
    text-decoration: none;
    font-weight: 600;
    color: black;
}

.author {
    font-size: small;
}

.price {
    font-size: 20px;
    font-weight: 500;
}

.infoBottom {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    align-items: center;
}

.infoBottom button {
    flex: 2;
    color: white;
    font-size: medium;
    font-weight: 600;
    padding: 10px 20px;
    border: none;
    border-radius: .1rem;
    cursor: pointer;
    text-decoration: none;
    background-color: #000080;
}
.infoLeft {
    flex: 3;
    display: flex;
    flex-direction: column;
}
```

With the codes above, a `ProductItem` will be look like this:



We have built the component for each product. Now, we need a list of product data. In a real application, the data is retrieved from back-end server. Here, we just put some fake data into a separate JavaScrtip file under the `src` folder. Let's call it `productsData.js`:

```
const data = [
    {
      id: 0,
      name: "React: From Beginner to Expert",
      author: "John Doe",
      price: 9.99,
      image:
        "https://cdn.thenewstack.io/media/2022/05/600b72f9-react-1024x680.png",
    },
    {
      id: 1,
      name: "Neural Networks from Scratch",
      author: "John Smith",
      price: 9.99,
      image:
        "https://7wdata.be/wp-content/uploads/2020/11/What-is-an-Artificial-Neural-
Networks.png",
    },
    {
      id: 2,
      name: "JavaScript: The Complete Guide",
      author: "Stephen Lee",
      price: 9.99,
      image:
        "https://repository-images.githubusercontent.com/229449376/3cfde400-5298-11ea-
9f39-aab161ef8f69",
    },
    {
      id: 3,
      name: "NodeJS Bootcamp",
      author: "Susan Williams",
      price: 13.99,
      image:
        "https://www.hkcodingclub.com/wp-content/uploads/2022/02/share-nodejs-logo-
1024x536.png",
    },
    {
      id: 4,
      name: "The Complete iOS Development Bootcamp",
      author: "Allen Brown",
      price: 9.99,
      image: "https://miro.medium.com/max/730/1*ND2d6CvH-Cz0dp5I_tYalQ.png",
    },
    {
      id: 5,
      name: "Python: Zero to Mastery",
      author: "Robert Simmons",
      price: 10.99,
      image:
        "https://bytesofintelligence.co.uk/wp-content/uploads/2021/06/python.jpeg",
    },
    {
      id: 6,
      name: "C++ Programming Language",
      author: "Steven Bell",
      price: 13.99,
      image:
        "https://res.cloudinary.com/practicaldev/image/fetch/s--Tu2z9cvG--
/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https://miro.medium.com/max/696/1%2
A6UpwEDOw04H5fKyaMGXpSw.png",
    },
    {
      id: 7,
      name: "Parallel Programming in Java",
      author: "Jenny Morris",
      price: 9.99,
      image:
        "https://mindqsystems.com/wp-content/uploads/2019/08/Core-Java-Training.jpg",
    },
    {
      id: 8,
      name: "Spring Framework Maserclass",
      author: "Rose Goodman",
      price: 10.99,
      image:
        "https://spring.io/images/OG-Spring.png",
    },
  ];

  export default data;
```

Now, we can finally build our product list.

Here, we create a new component called `Home` to display our home page.

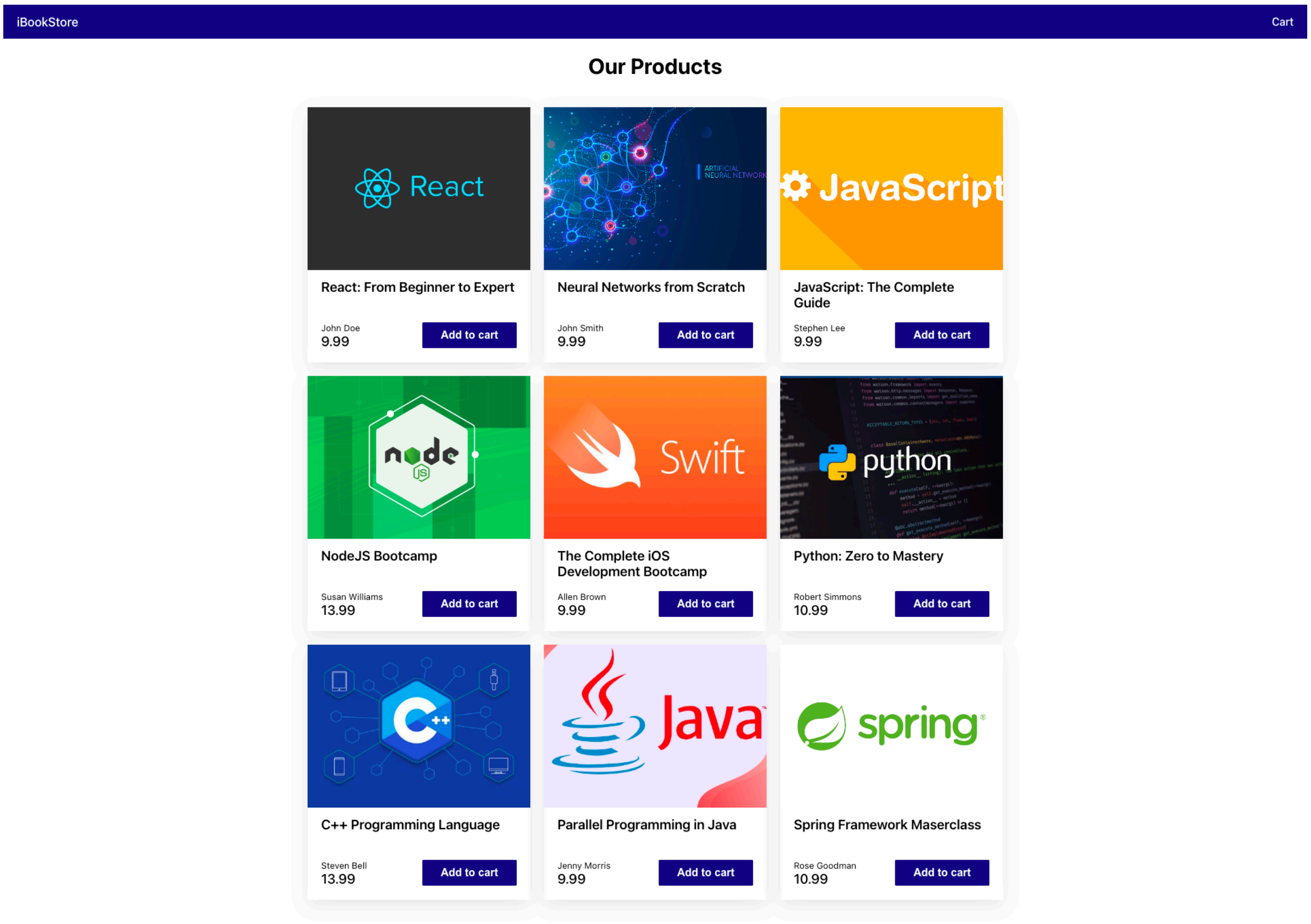
```
import React from "react";
import ProductItem from "../../components/ProductItem";
import classes from "./Home.module.css";
import data from "../../productsData";

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

  return (
    <div className={classes.home}>
      <div className={classes.container}>
        <h1>Our Products</h1>
        <div className={classes.productsContainer}>{productsList}</div>
      </div>
    </div>
  );
}

export default Home;
```

In the `Home` component, we create a product list with the `map()` method, which assign each of the value in the `data` to a corresponding `ProductItem`. For each `ProductItem`, the product information is passed with `props`. Note that when you use `map()` to create a list of components, you must pass a value for the `key` attribute, which should be a unique identifier that helps React identify the items. In our example, we can directly use the `id` in the data as the key value.

Also remember to paste these codes into `Home.module.css` for styling:

```css
.home {
    display: flex;
    justify-content: center;
}

.container {
    width: 100%;
    max-width: 1024px;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    gap: 20px;
    margin-bottom: 32px;
    min-height: 100vh;
}

.productsContainer {
    width: 100%;
    min-height: 787px;
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    grid-gap: 20px;
    grid-auto-rows: minmax(100px, auto);
}
```

Now, you should see a product list shown in the home page.



Our Products

React: From Beginner to Expert John Doe 9.99 Add to cart	Neural Networks from Scratch John Smith 9.99 Add to cart	JavaScript: The Complete Guide Stephen Lee 9.99 Add to cart
NodeJS Bootcamp Susan Williams 13.99 Add to cart	The Complete iOS Development Bootcamp Allen Brown 9.99 Add to cart	Python: Zero to Mastery Robert Simmons 10.99 Add to cart
C++ Programming Language Steven Bell 13.99 Add to cart	Parallel Programming in Java Jenny Morris 9.99 Add to cart	Spring Framework Maserclass Rose Goodman 10.99 Add to cart

The complete code of Tutorial 1: https://github.com/lvyin1122/ecommerce-tutorial-code/tree/01-getting-started