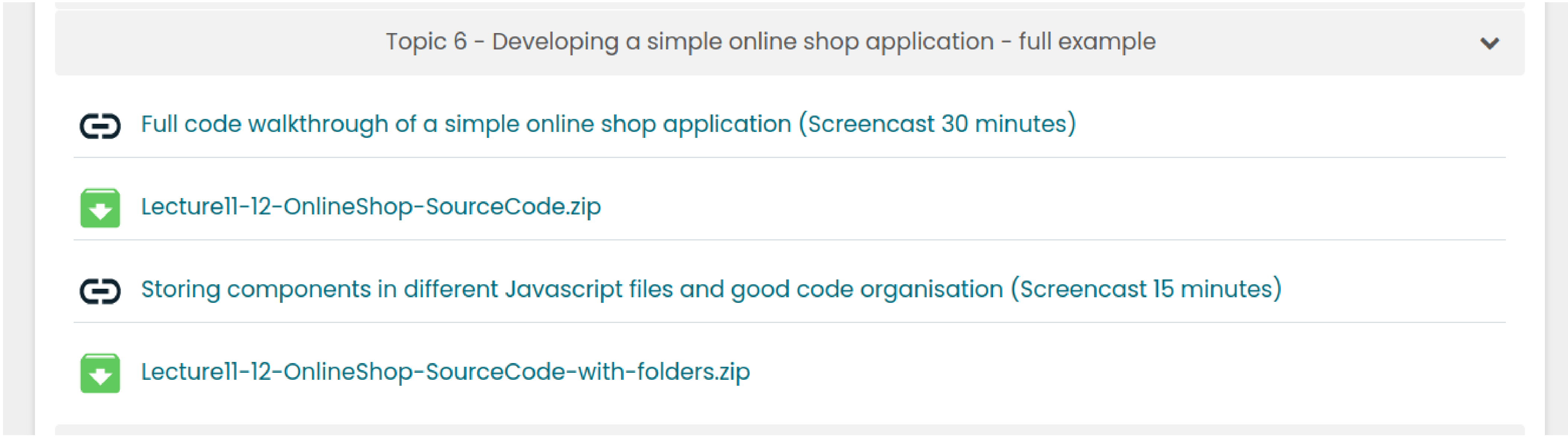
CS385 Lecture 11

Lecture 11 considers developing a full application (online store)

- We will utilise concepts we have already learned in our course
  - **Using multiple components**
  - **Using parent child communication**
  - **Event driven programming**
  - **Managing variables in state**
- We will **learn three new and important concepts**:
  - **The spread operator** in Javascript (Lab Exam 2)
  - **Avoiding mutation of state variables**
  - Using **the splice operator for arrays** in Javascript (Lab Exam 2)

Lecture 11 – 12 – full source code available AND screencasts

- Look at Topic 6 on Moodle



Topic 6 - Developing a simple online shop application - full example

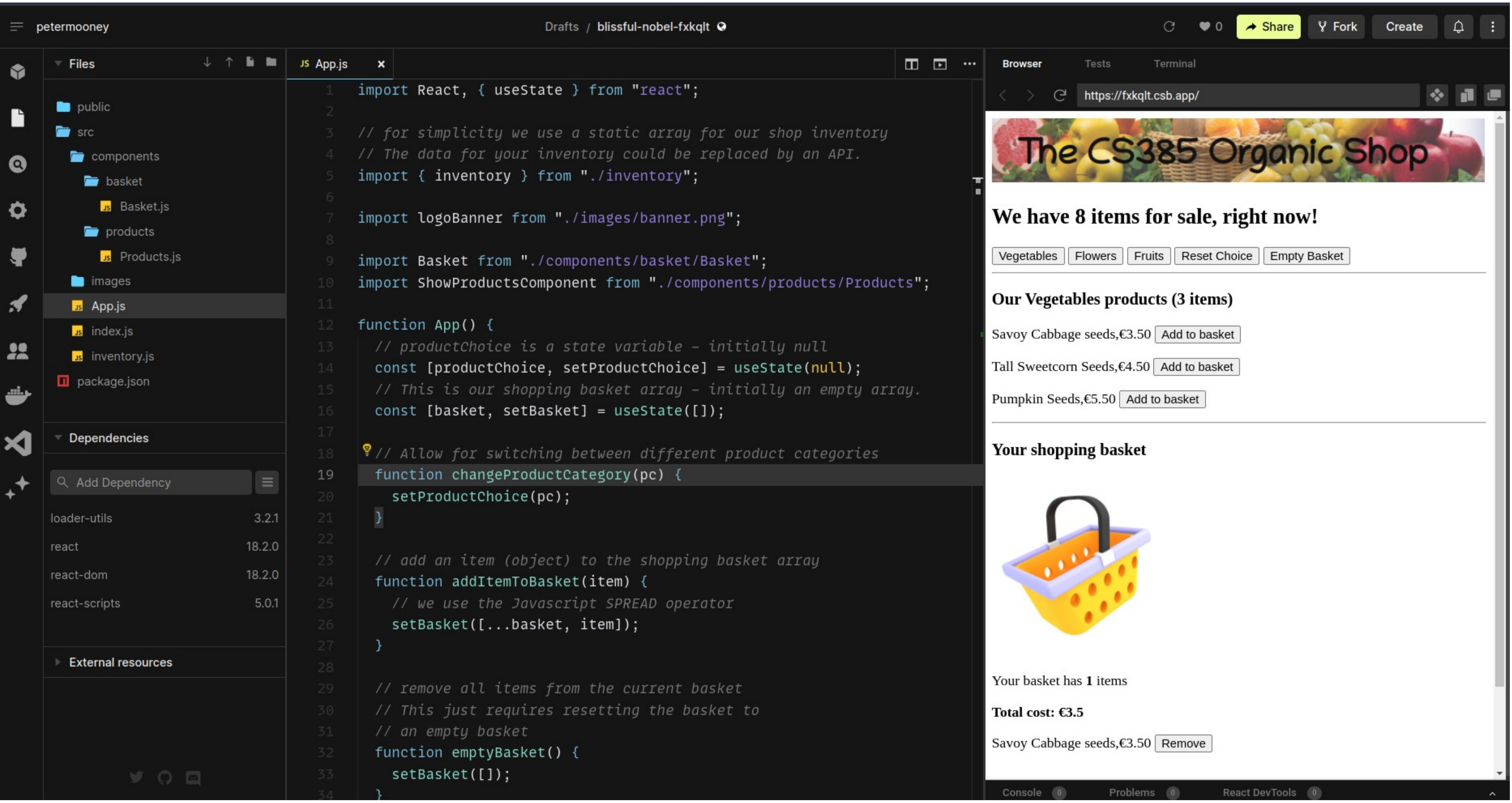
Full code walkthrough of a simple online shop application (Screencast 30 minutes)

Lecture11-12-OnlineShop-SourceCode.zip

Storing components in different Javascript files and good code organisation (Screencast 15 minutes)

Lecture11-12-OnlineShop-SourceCode-with-folders.zip

The CS385 Organic Shop



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

// for simplicity we use a static array for our shop inventory
// The data for your inventory could be replaced by an API.
import { inventory } from "./inventory";

import logoBanner from "./images/banner.png";

import Basket from "./components/basket/Basket";
import ShowProductsComponent from "./components/products/Products";

function App() {
  // productChoice is a state variable - initially null
  const [productChoice, setProductChoice] = useState(null);
  // This is our shopping basket array - initially an empty array.
  const [basket, setBasket] = useState([]);

  // Allow for switching between different product categories
  function changeProductCategory(pc) {
    setProductChoice(pc);
  }

  // add an item (object) to the shopping basket array
  function addItemToBasket(item) {
    // we use the Javascript SPREAD operator
    setBasket([...basket, item]);
  }

  // remove all items from the current basket
  // This just requires resetting the basket to
  // an empty basket
  function emptyBasket() {
    setBasket([]);
  }
```

We have 8 items for sale, right now!

Vegetables | Flowers | Fruits | Reset Choice | Empty Basket

Our Vegetables products (3 items)

Savoy Cabbage seeds,€3.50 Add to basket

Tall Sweetcorn Seeds,€4.50 Add to basket

Pumpkin Seeds,€5.50 Add to basket

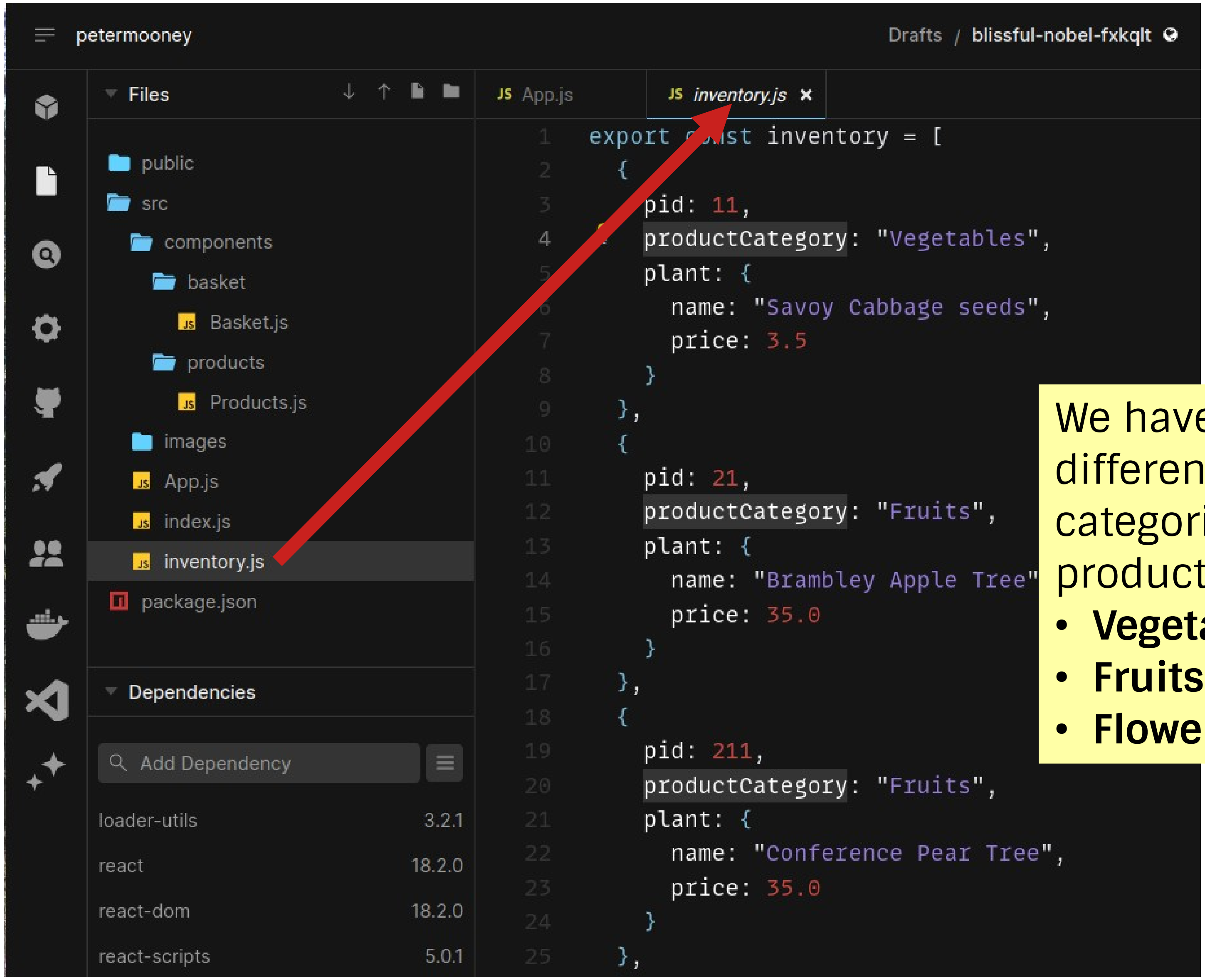
Your shopping basket

Your basket has **1** items

Total cost: €3.5

Savoy Cabbage seeds,€3.50 Remove

For simplicity – we use a local JS file to hold our "inventory" of products (as JSON)



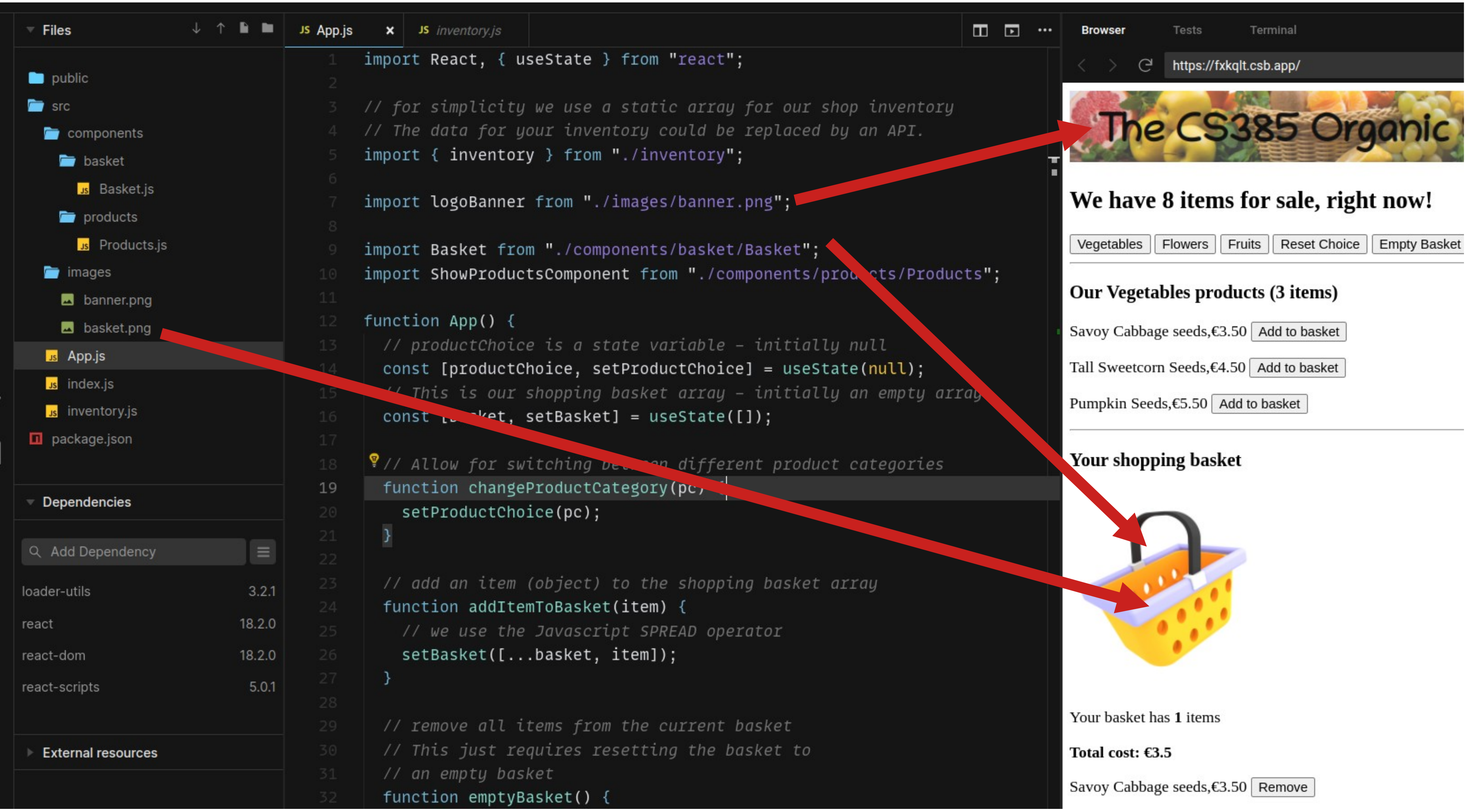
loader-utils	3.2.1
react	18.2.0
react-dom	18.2.0
react-scripts	5.0.1

```
export const inventory = [
  {
    pid: 11,
    productCategory: "Vegetables",
    plant: {
      name: "Savoy Cabbage seeds",
      price: 3.5
    }
  },
  {
    pid: 21,
    productCategory: "Fruits",
    plant: {
      name: "Brambley Apple Tree",
      price: 35.0
    }
  },
  {
    pid: 211,
    productCategory: "Fruits",
    plant: {
      name: "Conference Pear Tree",
      price: 35.0
    }
  },
```

We have three different categories of products

- **Vegetables**
- **Fruits**
- **Flowers**

We store two images for our User Interface in a sub-folder of src



Files

- public
- src
  - components
    - basket
      - Basket.js
    - products
      - Products.js
  - images
    - banner.png
    - basket.png
  - App.js
  - index.js
  - inventory.js
- package.json

Dependencies

loader-utils	3.2.1
react	18.2.0
react-dom	18.2.0
react-scripts	5.0.1

External resources

App.js · inventory.js

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

// for simplicity we use a static array for our shop inventory
// The data for your inventory could be replaced by an API.
import { inventory } from "./inventory";

import logoBanner from "./images/banner.png";

import Basket from "./components/basket/Basket";
import ShowProductsComponent from "./components/products/Products";

function App() {
  // productChoice is a state variable - initially null
  const [productChoice, setProductChoice] = useState(null);
  // This is our shopping basket array - initially an empty array
  const [basket, setBasket] = useState([]);

  // Allow for switching between different product categories
  function changeProductCategory(pc) {
    setProductChoice(pc);
  }

  // add an item (object) to the shopping basket array
  function addItemToBasket(item) {
    // we use the Javascript SPREAD operator
    setBasket([...basket, item]);
  }

  // remove all items from the current basket
  // This just requires resetting the basket to
  // an empty basket
  function emptyBasket() {
```

Browser · Tests · Terminal

https://fxkqlt.csb.app/

The CS385 Organic

We have 8 items for sale, right now!

Vegetables · Flowers · Fruits · Reset Choice · Empty Basket

Our Vegetables products (3 items)

Savoy Cabbage seeds,€3.50 Add to basket

Tall Sweetcorn Seeds,€4.50 Add to basket

Pumpkin Seeds,€5.50 Add to basket

Your shopping basket

Your basket has **1** items

Total cost: €3.5

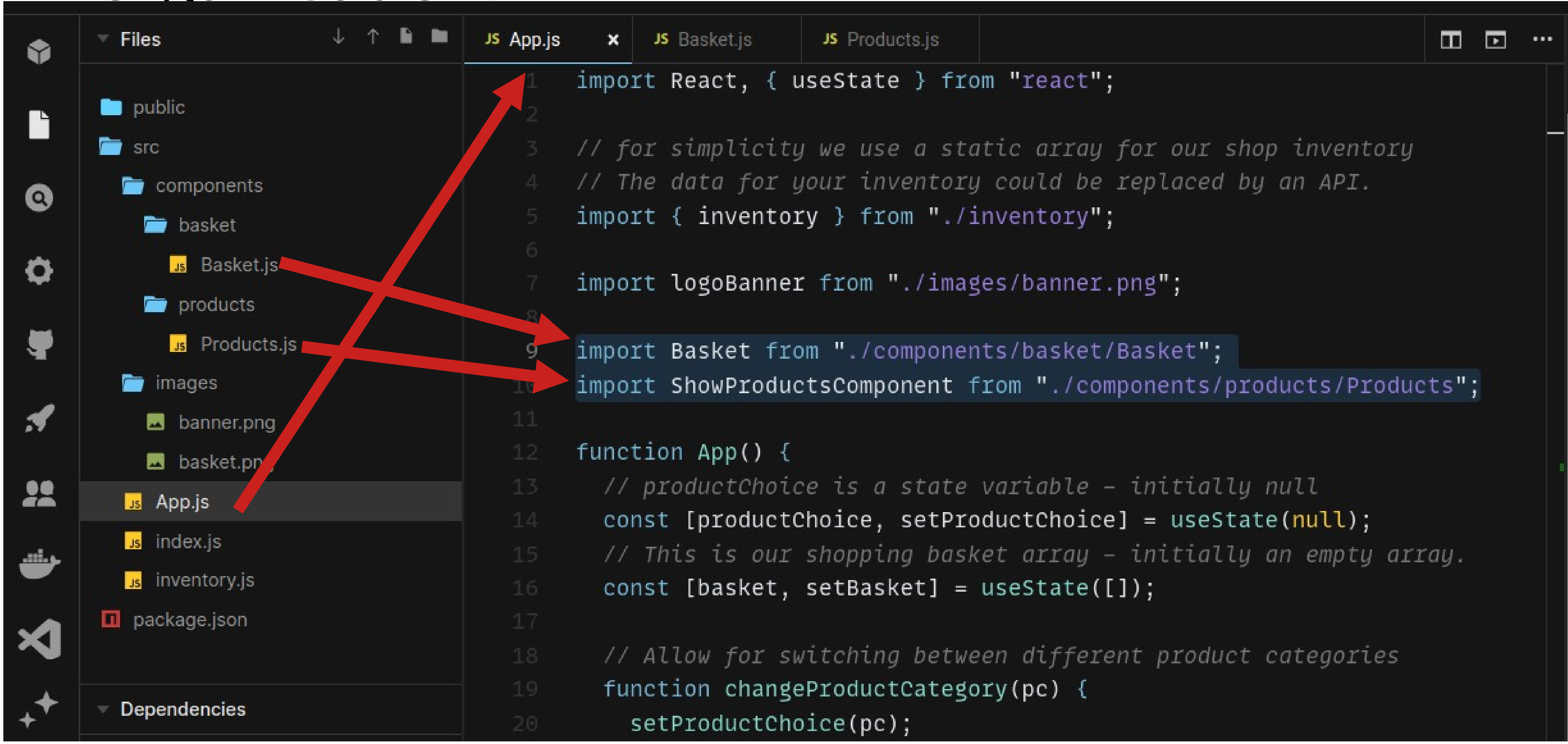
Savoy Cabbage seeds,€3.50 Remove

We will have THREE components in our application

- The App() **component (PARENT)** – the parent will manage state variables for the basket and the array of JSON objects (inventory)
- The **ShowProductsComponent (CHILD)** – used to display the items in each product category
- The **Basket Component (CHILD)** – used to manage and display the content of the user’s shopping basket or shopping cart.

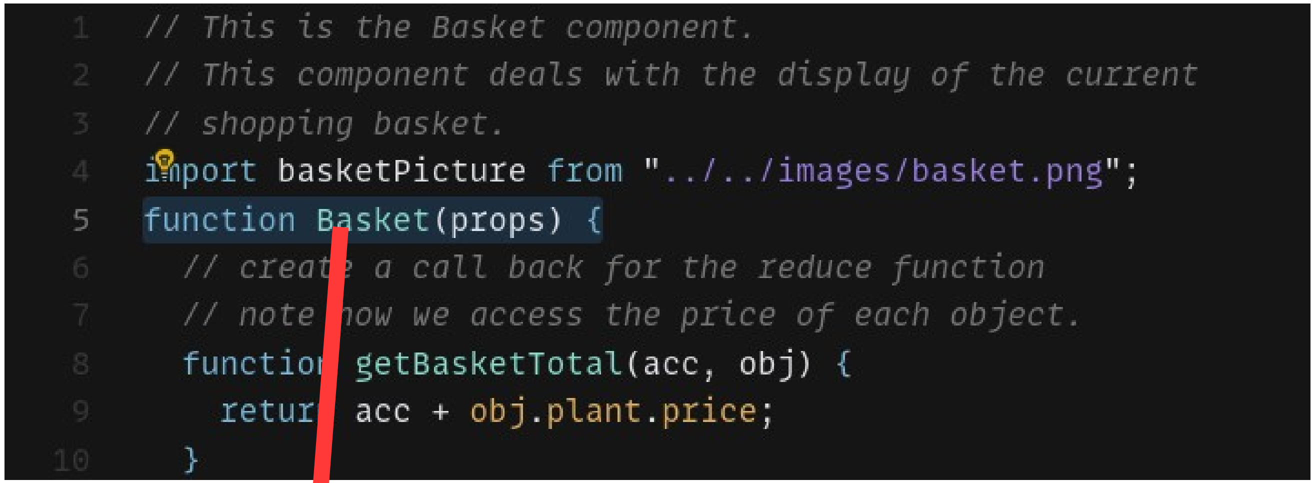
In this app – we have separated our components into THREE different files

- This helps code readability and code organisation.

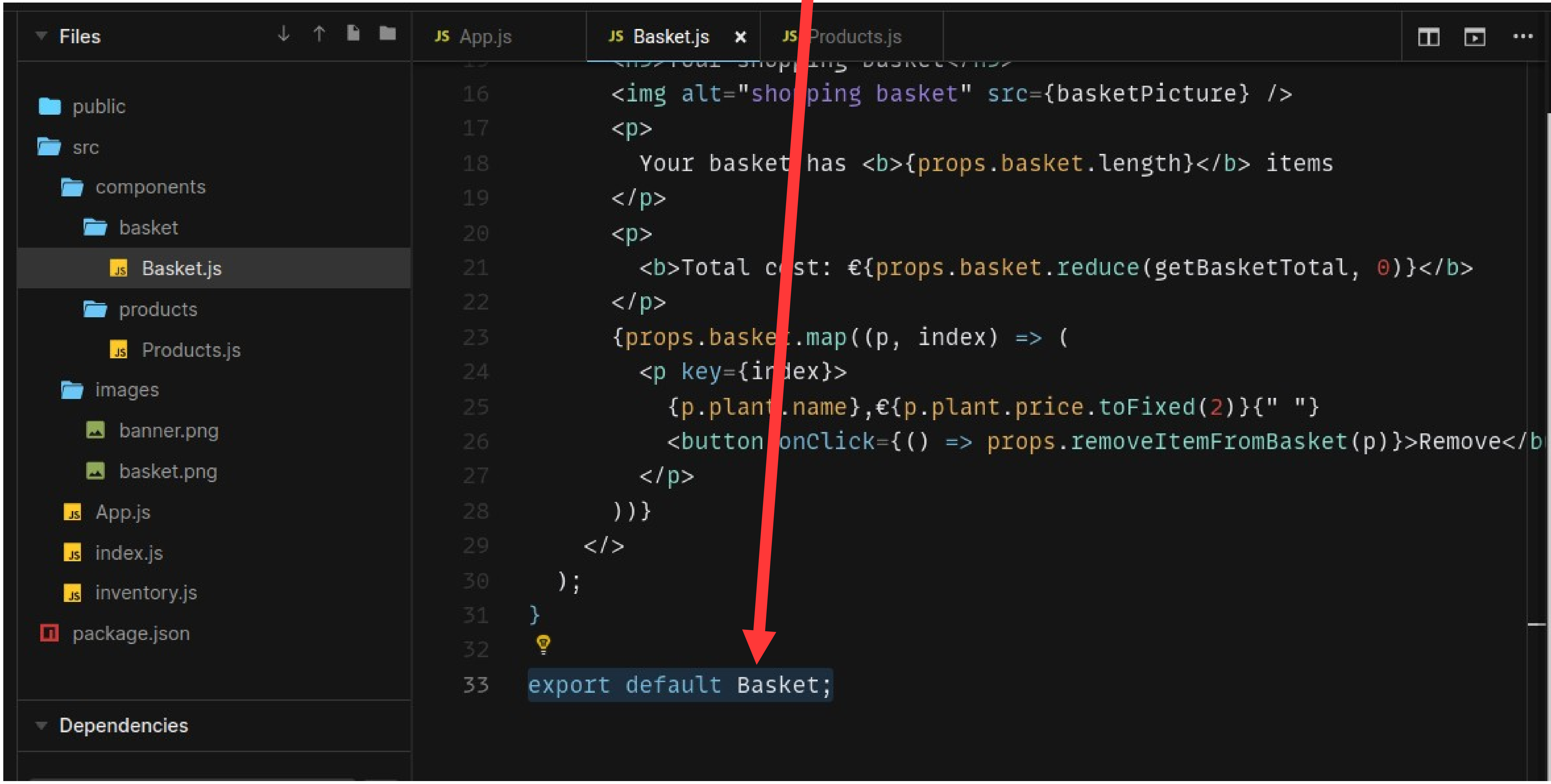


```
import React, { useState } from "react";

// for simplicity we use a static array for our shop inventory
// The data for your inventory could be replaced by an API.
import { inventory } from "./inventory";

import logoBanner from "./images/banner.png";

import Basket from "./components/basket/Basket";
import ShowProductsComponent from "./components/products/Products";

function App() {
  // productChoice is a state variable - initially null
  const [productChoice, setProductChoice] = useState(null);
  // This is our shopping basket array - initially an empty array.
  const [basket, setBasket] = useState([]);

  // Allow for switching between different product categories
  function changeProductCategory(pc) {
    setProductChoice(pc);
```

Basket.js



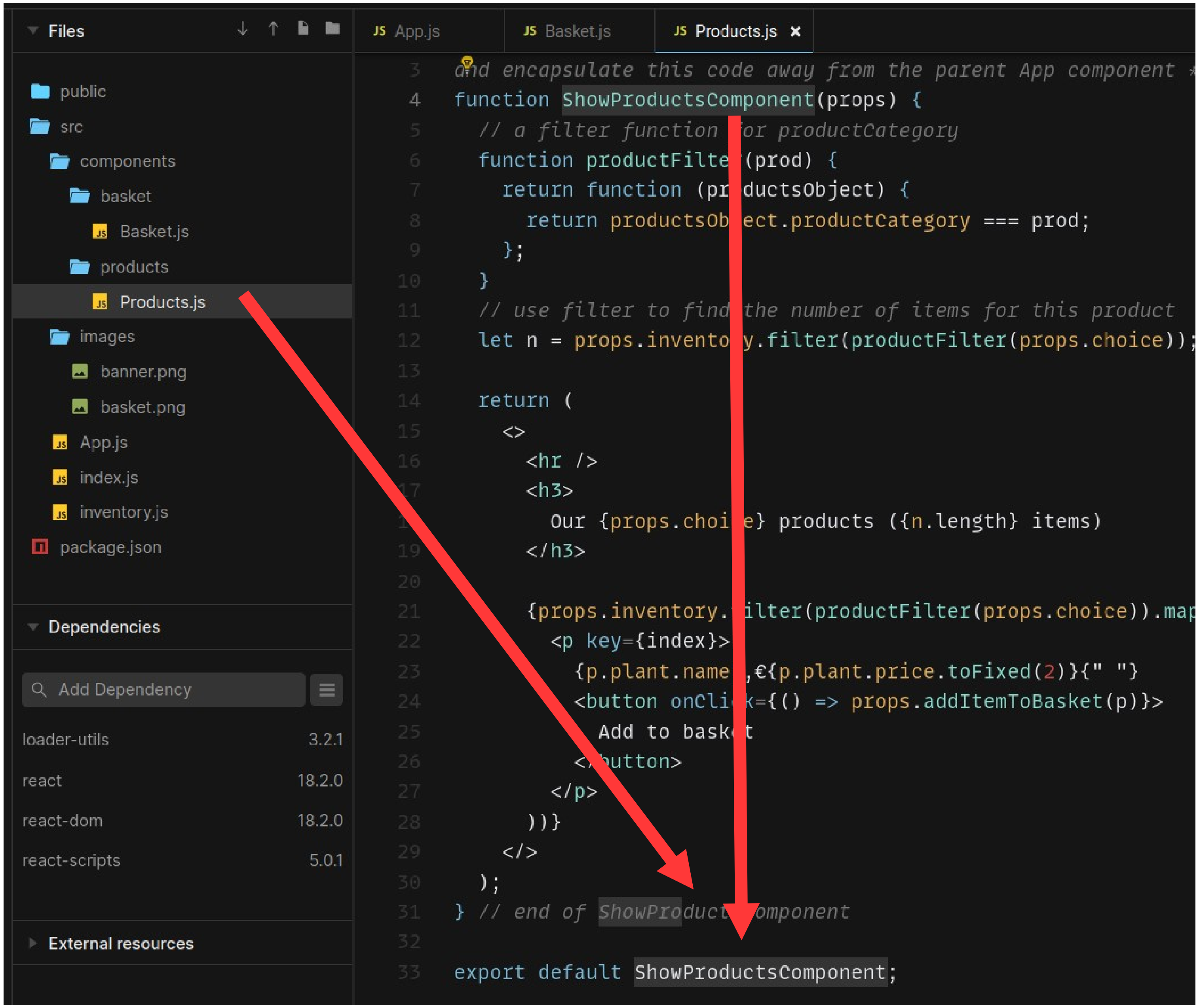
```
// This is the Basket component.
// This component deals with the display of the current
// shopping basket.
import basketPicture from "../../images/basket.png";
function Basket(props) {
  // create a call back for the reduce function
  // note how we access the price of each object.
  function getBasketTotal(acc, obj) {
    return acc + obj.plant.price;
  }
```



```
        <h3>Your shopping basket</h3>
        <img alt="shopping basket" src={basketPicture} />
        <p>
          Your basket has <b>{props.basket.length}</b> items
        </p>
        <p>
          <b>Total cost: €{props.basket.reduce(getBasketTotal, 0)}</b>
        </p>
        {props.basket.map((p, index) => (
          <p key={index}>
            {p.plant.name},€{p.plant.price.toFixed(2)}{" "}
            <button onClick={() => props.removeItemFromBasket(p)}>Remove</b
          </p>
        ))}
      </>
    );
}

export default Basket;
```

Products.js

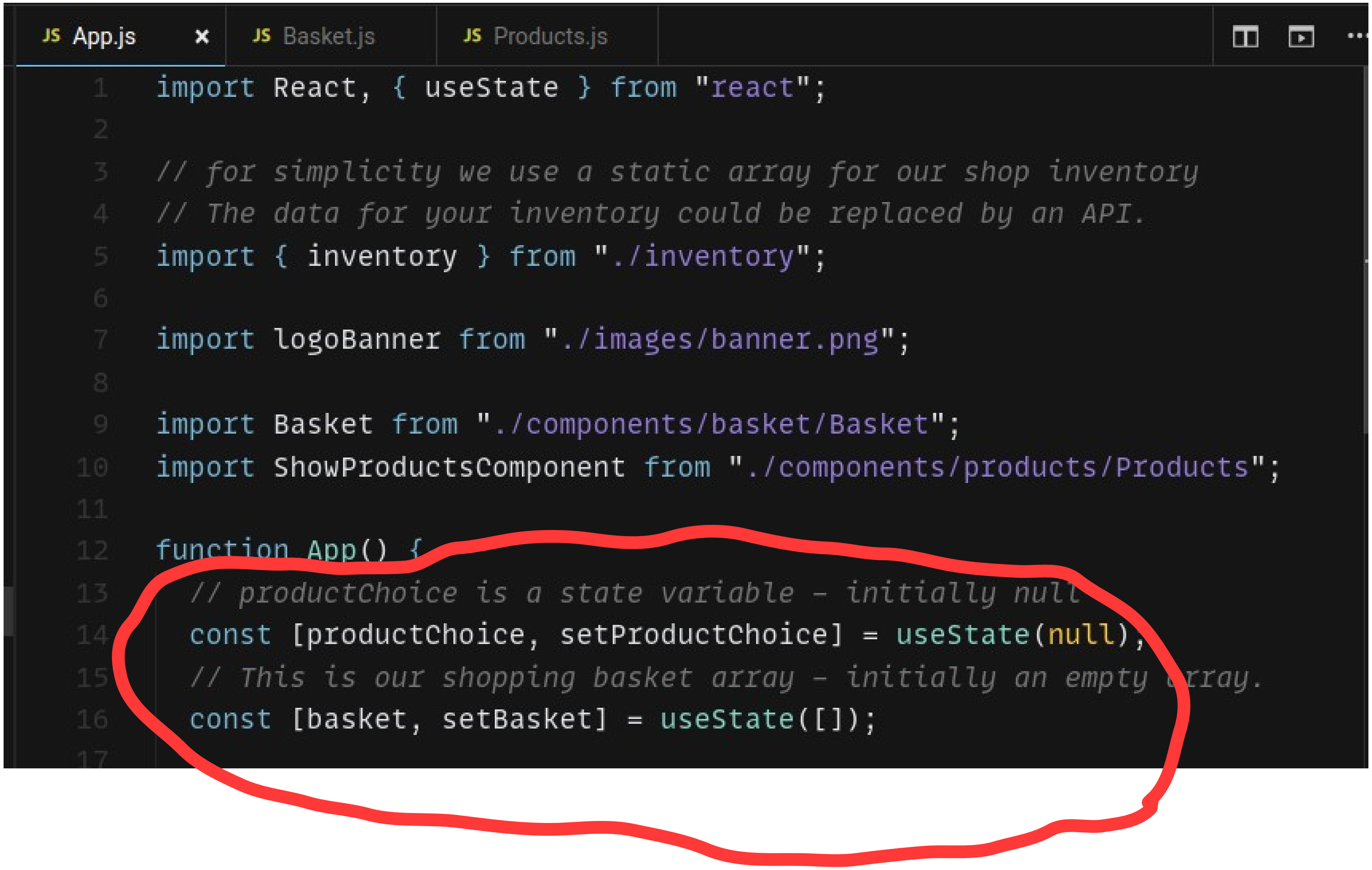
- Note the difference in the filename and the component name. **Note the use of the export on Line 33**



loader-utils	3.2.1
react	18.2.0
react-dom	18.2.0
react-scripts	5.0.1

```
3   and encapsulate this code away from the parent App component *
4   function ShowProductsComponent(props) {
5     // a filter function for productCategory
6     function productFilter(prod) {
7       return function (productsObject) {
8         return productsObject.productCategory === prod;
9       };
10    }
11    // use filter to find the number of items for this product
12    let n = props.inventory.filter(productFilter(props.choice));
13
14    return (
15      <>
16        <hr />
17        <h3>
18          Our {props.choice} products ({n.length} items)
19        </h3>
20
21        {props.inventory.filter(productFilter(props.choice)).map
22          <p key={index}>
23            {p.plant.name},€{p.plant.price.toFixed(2)}{" "}
24            <button onClick={() => props.addItemToBasket(p)}>
25              Add to basket
26            </button>
27          </p>
28        ))}
29      </>
30    );
31  } // end of ShowProductsComponent
32
33  export default ShowProductsComponent;
```

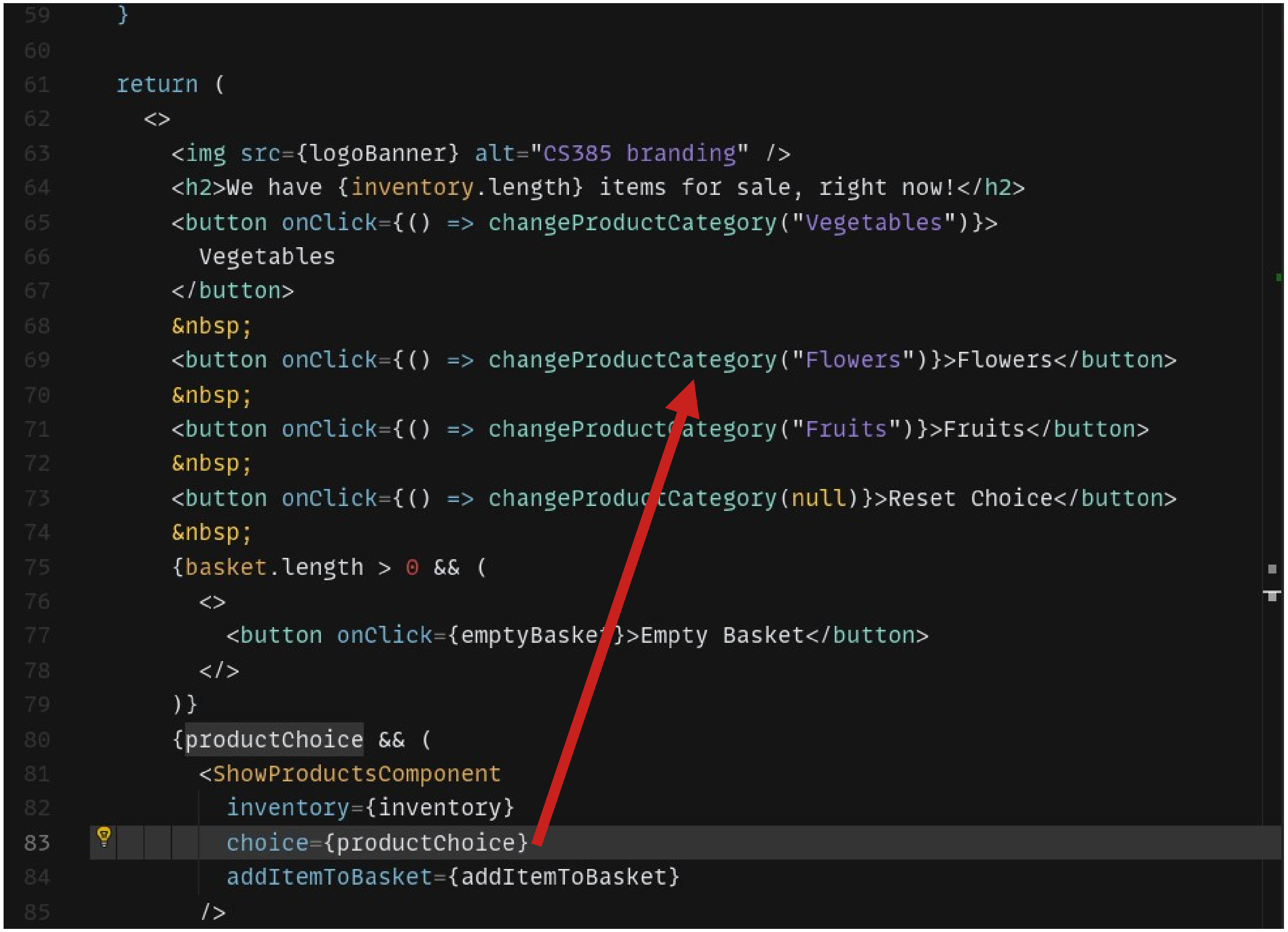
CS385 Organic Shop – our `state variables` Line 14 - 16



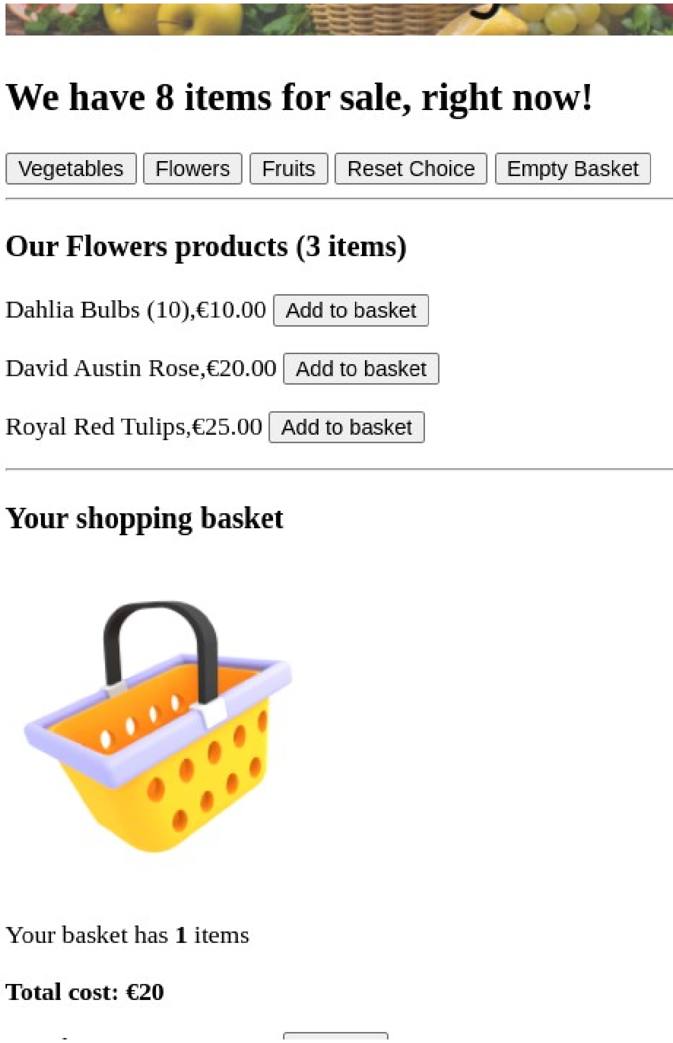
```
1   import React, { useState } from "react";
2
3   // for simplicity we use a static array for our shop inventory
4   // The data for your inventory could be replaced by an API.
5   import { inventory } from "./inventory";
6
7   import logoBanner from "./images/banner.png";
8
9   import Basket from "./components/basket/Basket";
10  import ShowProductsComponent from "./components/products/Products";
11
12  function App() {
13    // productChoice is a state variable - initially null
14    const [productChoice, setProductChoice] = useState(null);
15    // This is our shopping basket array - initially an empty array.
16    const [basket, setBasket] = useState([]);
```

We use conditional rendering to display products and buttons

- For example, the empty basket button Line 77



```
59    }
60
61    return (
62      <>
63        <img src={logoBanner} alt="CS385 branding" />
64        <h2>We have {inventory.length} items for sale, right now!</h2>
65        <button onClick={() => changeProductCategory("Vegetables")}>
66          Vegetables
67        </button>
68        &nbsp;
69        <button onClick={() => changeProductCategory("Flowers")}>Flowers</button>
70        &nbsp;
71        <button onClick={() => changeProductCategory("Fruits")}>Fruits</button>
72        &nbsp;
73        <button onClick={() => changeProductCategory(null)}>Reset Choice</button>
74        &nbsp;
75        {basket.length > 0 && (
76          <>
77            <button onClick={emptyBasket}>Empty Basket</button>
78          </>
79        )}
80        {productChoice && (
81          <ShowProductsComponent
82            inventory={inventory}
83            choice={productChoice}
84            addItemToBasket={addItemToBasket}
85          />
```



We have 8 items for sale, right now!

Vegetables | Flowers | Fruits | Reset Choice | Empty Basket

Our Flowers products (3 items)

Dahlia Bulbs (10),€10.00 Add to basket

David Austin Rose,€20.00 Add to basket

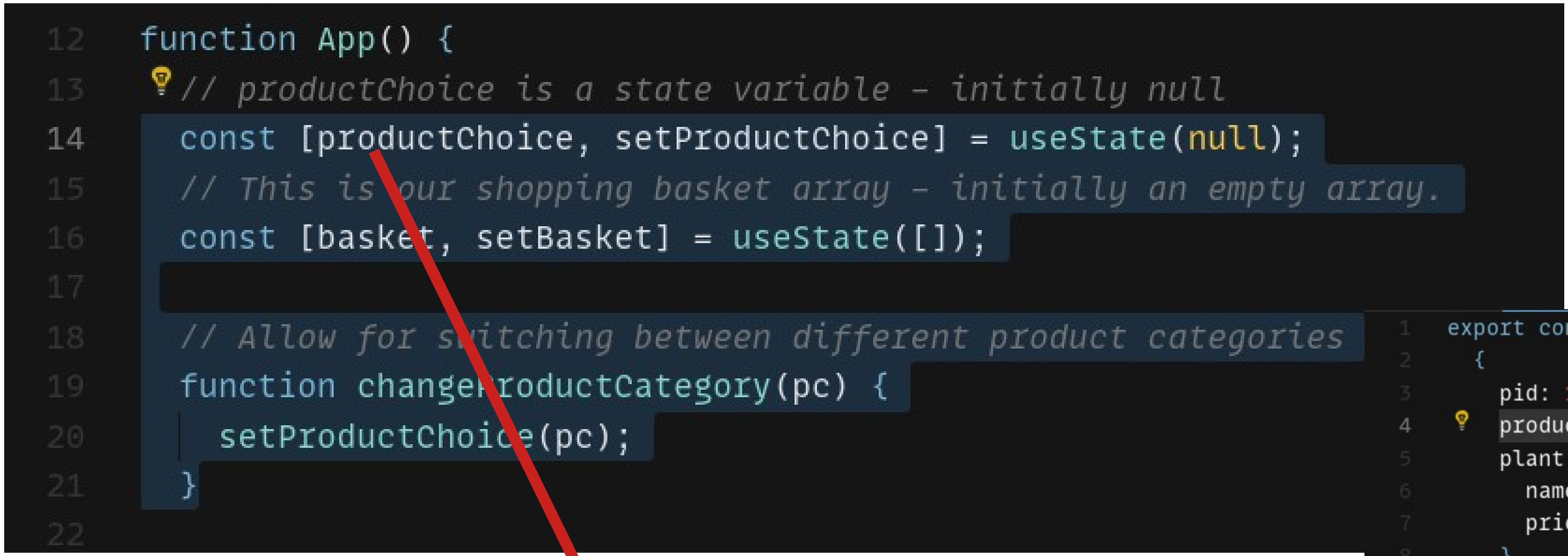
Royal Red Tulips,€25.00 Add to basket

Your shopping basket

Your basket has **1** items

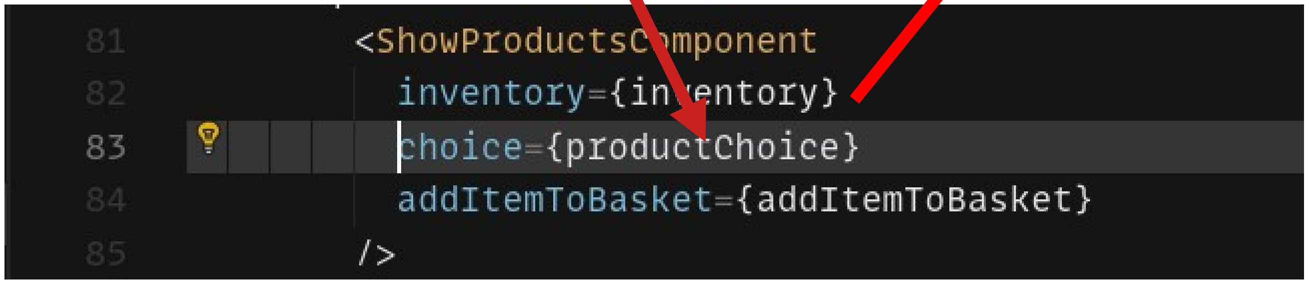
Total cost: €20

changeProductChoice

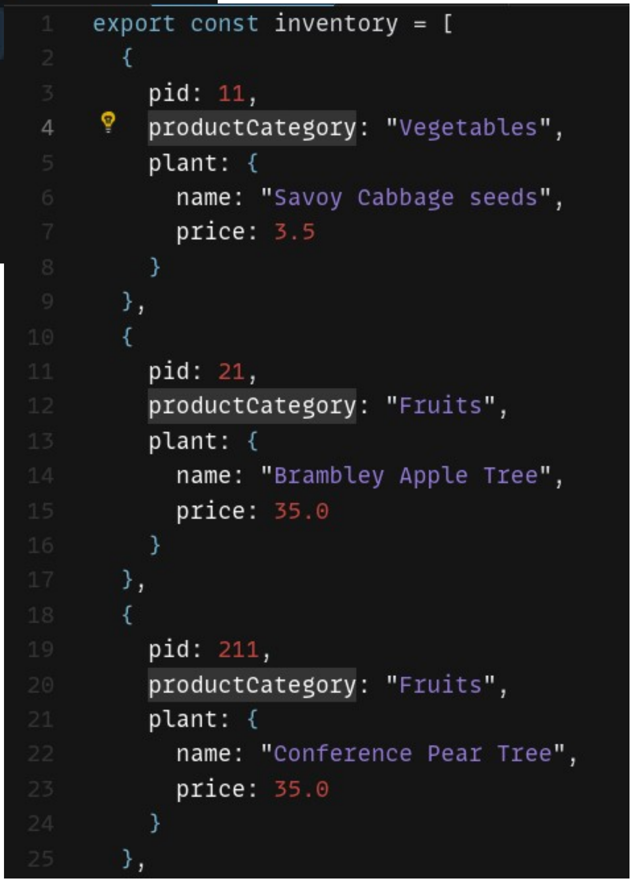


```
function App() {
  // productChoice is a state variable - initially null
  const [productChoice, setProductChoice] = useState(null);
  // This is our shopping basket array - initially an empty array.
  const [basket, setBasket] = useState([]);

  // Allow for switching between different product categories
  function changeProductCategory(pc) {
    setProductChoice(pc);
  }

```

Our inventory (array) of products

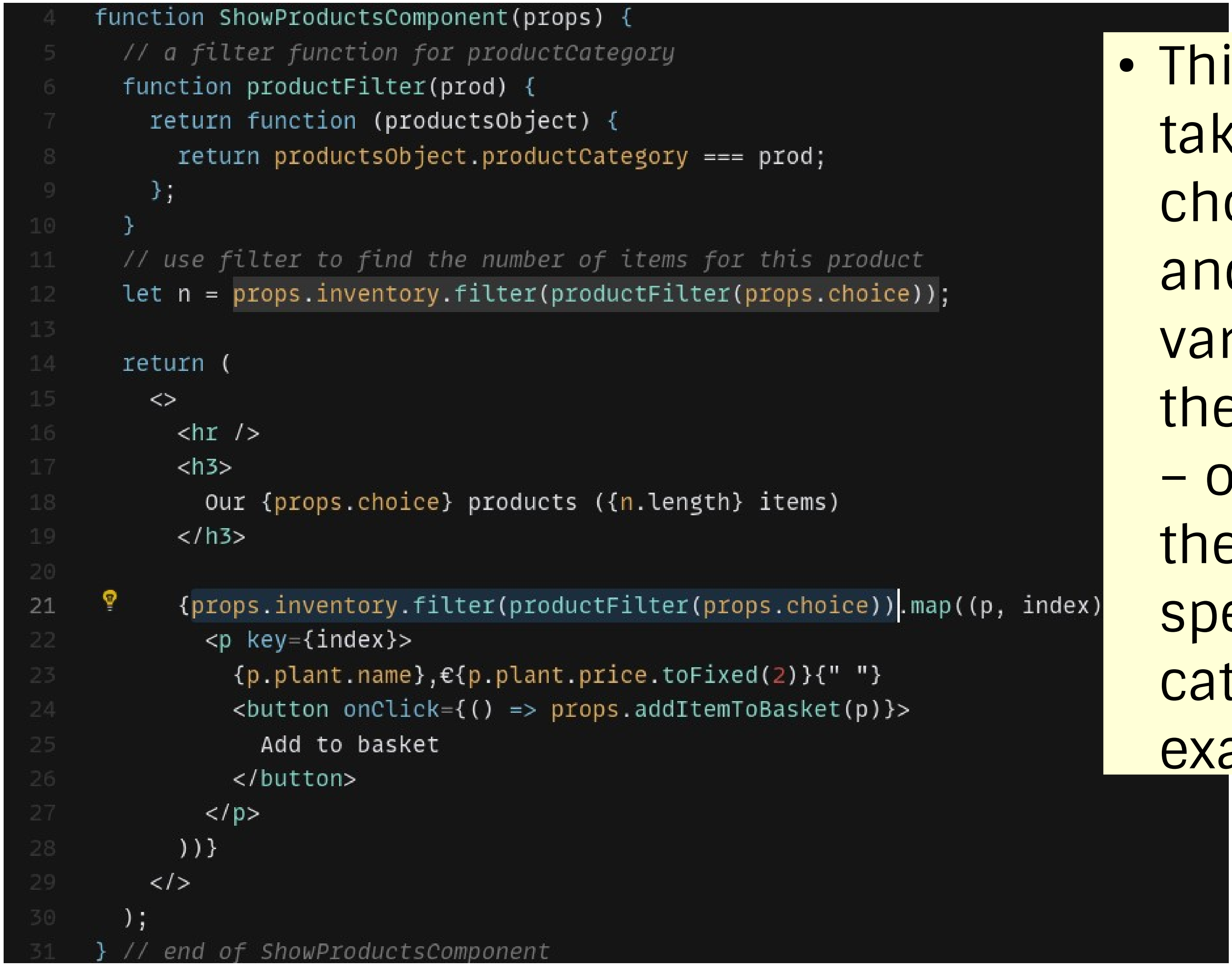


```
        <ShowProductsComponent
          inventory={inventory}
          choice={productChoice}
          addItemToBasket={addItemToBasket}
        />
```



```
export const inventory = [
  {
    pid: 11,
    productCategory: "Vegetables",
    plant: {
      name: "Savoy Cabbage seeds",
      price: 3.5
    }
  },
  {
    pid: 21,
    productCategory: "Fruits",
    plant: {
      name: "Brambley Apple Tree",
      price: 35.0
    }
  },
  {
    pid: 211,
    productCategory: "Fruits",
    plant: {
      name: "Conference Pear Tree",
      price: 35.0
    }
  },
```

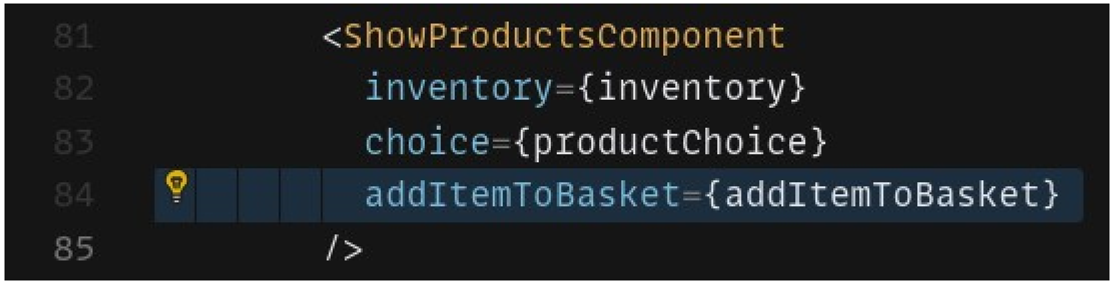
ShowProductsComponent



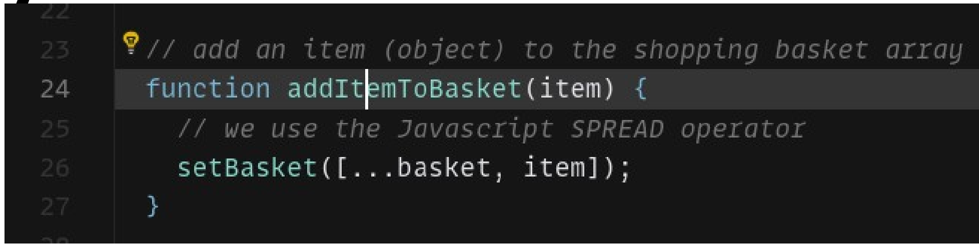
```
function ShowProductsComponent(props) {
  // a filter function for productCategory
  function productFilter(prod) {
    return function (productsObject) {
      return productsObject.productCategory === prod;
    };
  }
  // use filter to find the number of items for this product
  let n = props.inventory.filter(productFilter(props.choice));

  return (
    <>
      <hr />
      <h3>
        Our {props.choice} products ({n.length} items)
      </h3>

      {props.inventory.filter(productFilter(props.choice)).map((p, index)
        <p key={index}>
          {p.plant.name},€{p.plant.price.toFixed(2)}{" "}
          <button onClick={() => props.addItemToBasket(p)}>
            Add to basket
          </button>
        </p>
      ))}
    </>
  );
} // end of ShowProductsComponent
```

- This component takes the user’s choice of product and uses this variable to FILTER the inventory array – only displaying the items for the specific product category (for example “Flowers”)

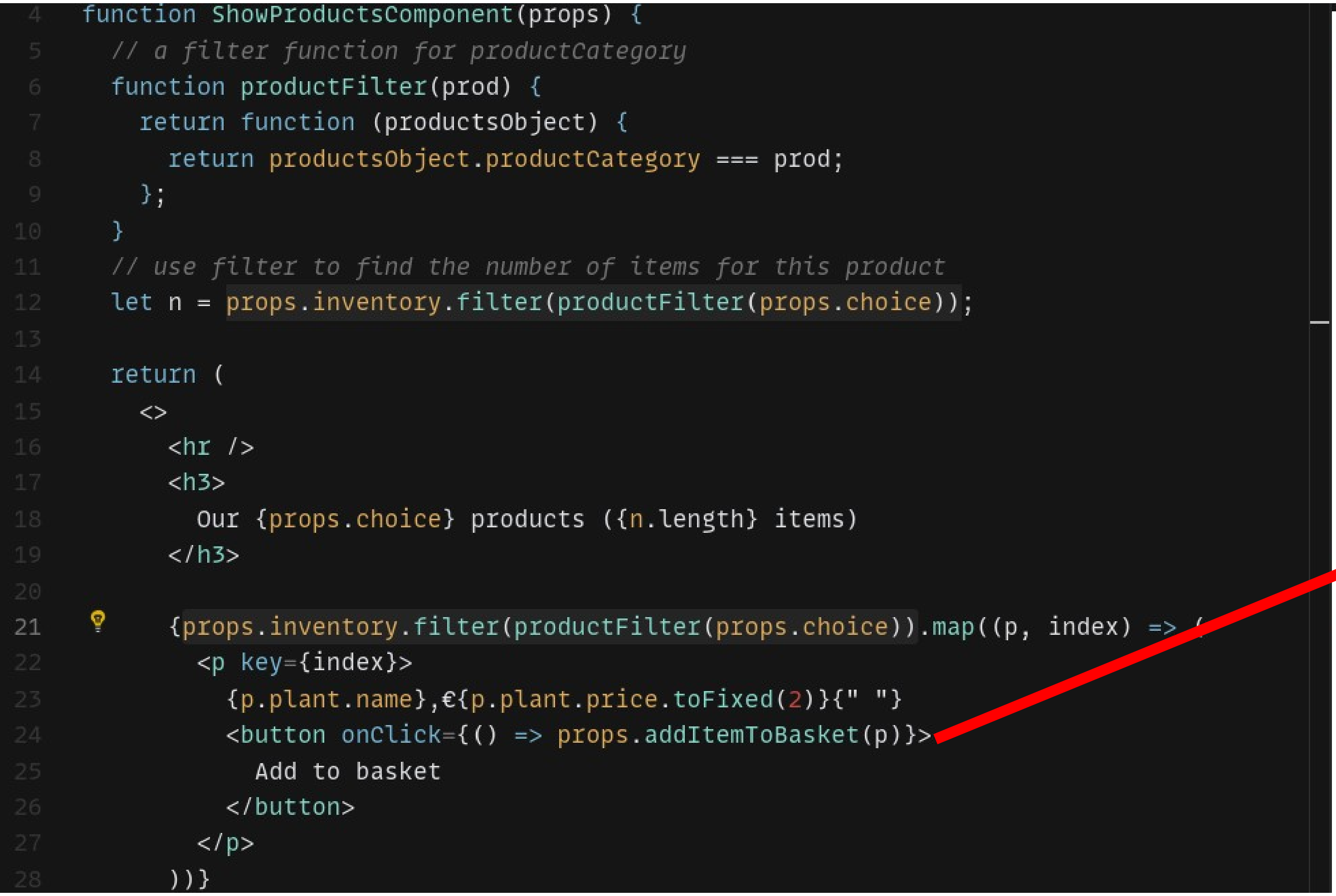
**** We use the map function to add a button functionality for each item



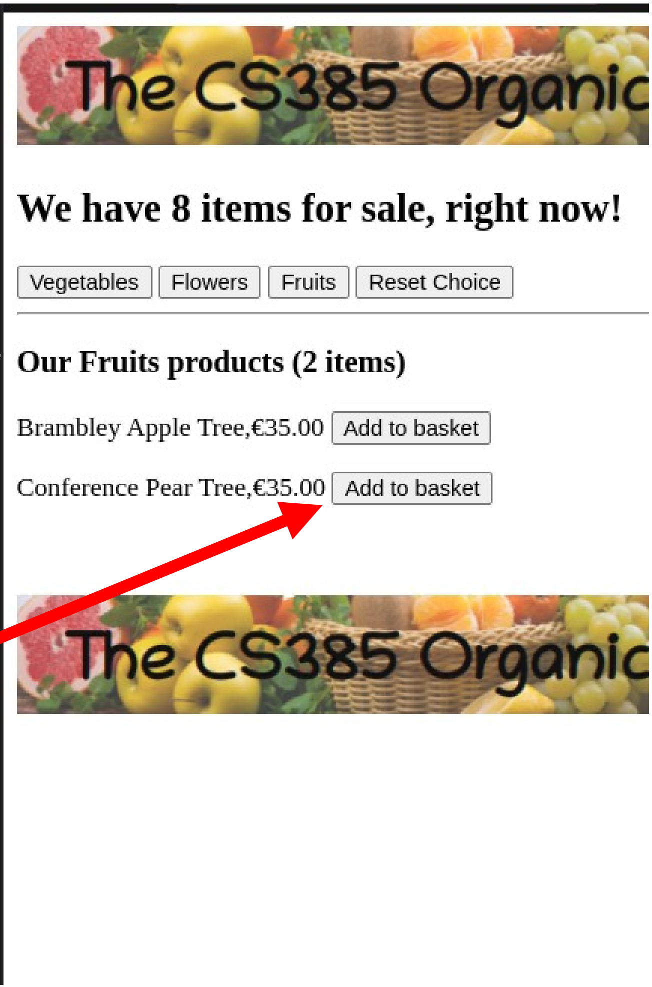
```
        <ShowProductsComponent
          inventory={inventory}
          choice={productChoice}
          addItemToBasket={addItemToBasket}
        />
```



```
// add an item (object) to the shopping basket array
function addItemToBasket(item) {
  // we use the Javascript SPREAD operator
  setBasket([...basket, item]);
}
```



```
function ShowProductsComponent(props) {
  // a filter function for productCategory
  function productFilter(prod) {
    return function (productsObject) {
      return productsObject.productCategory === prod;
    };
  }
  // use filter to find the number of items for this product
  let n = props.inventory.filter(productFilter(props.choice));

  return (
    <>
      <hr />
      <h3>
        Our {props.choice} products ({n.length} items)
      </h3>

      {props.inventory.filter(productFilter(props.choice)).map((p, index) => (
        <p key={index}>
          {p.plant.name},€{p.plant.price.toFixed(2)}{" "}
          <button onClick={() => props.addItemToBasket(p)}>
            Add to basket
          </button>
        </p>
      ))}
```



The CS385 Organic

We have 8 items for sale, right now!

Vegetables | Flowers | Fruits | Reset Choice

Our Fruits products (2 items)

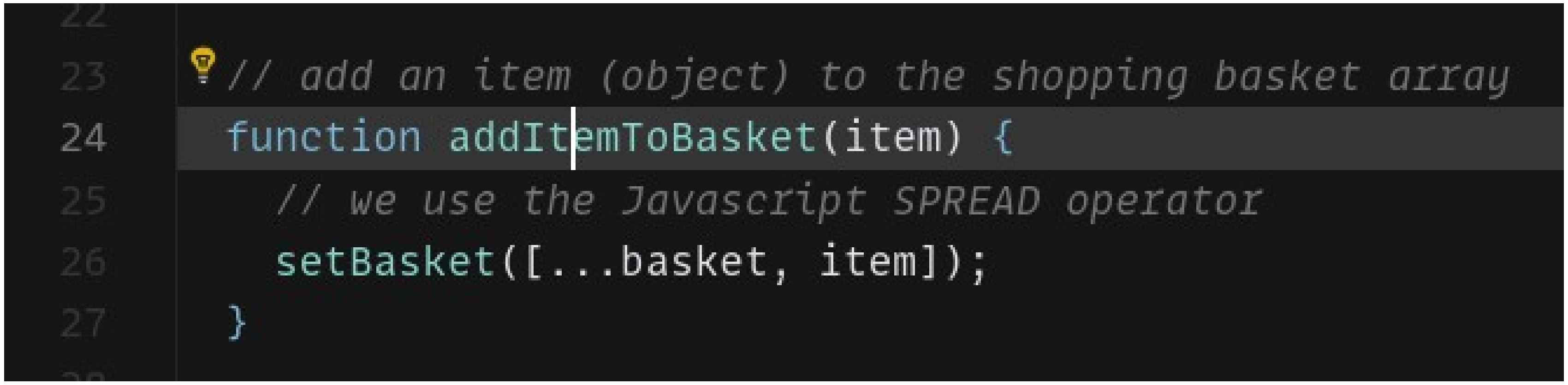
Brambley Apple Tree,€35.00 Add to basket

Conference Pear Tree,€35.00 Add to basket

The CS385 Organic

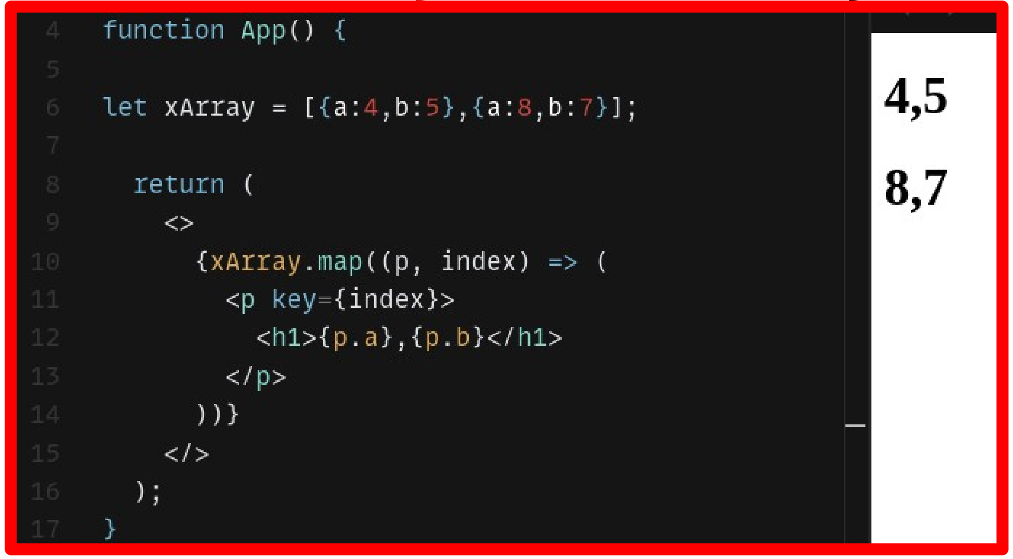
The SPREAD operator is used within the `addItemToBasket` function

- This is a very important operator in Javascript. It essentially allows us to concatenate arrays.
- **It is especially useful when we are working with arrays as state variables – it helps us to avoid the problem of state mutation.**



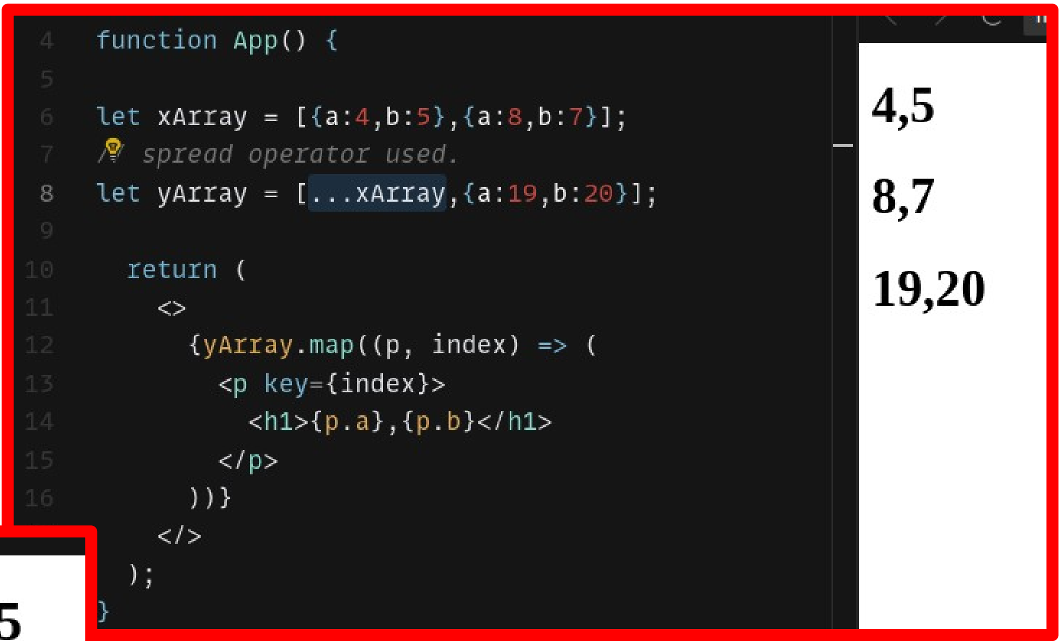
```
// add an item (object) to the shopping basket array
function addItemToBasket(item) {
  // we use the Javascript SPREAD operator
  setBasket([...basket, item]);
}
```

Spread operator example



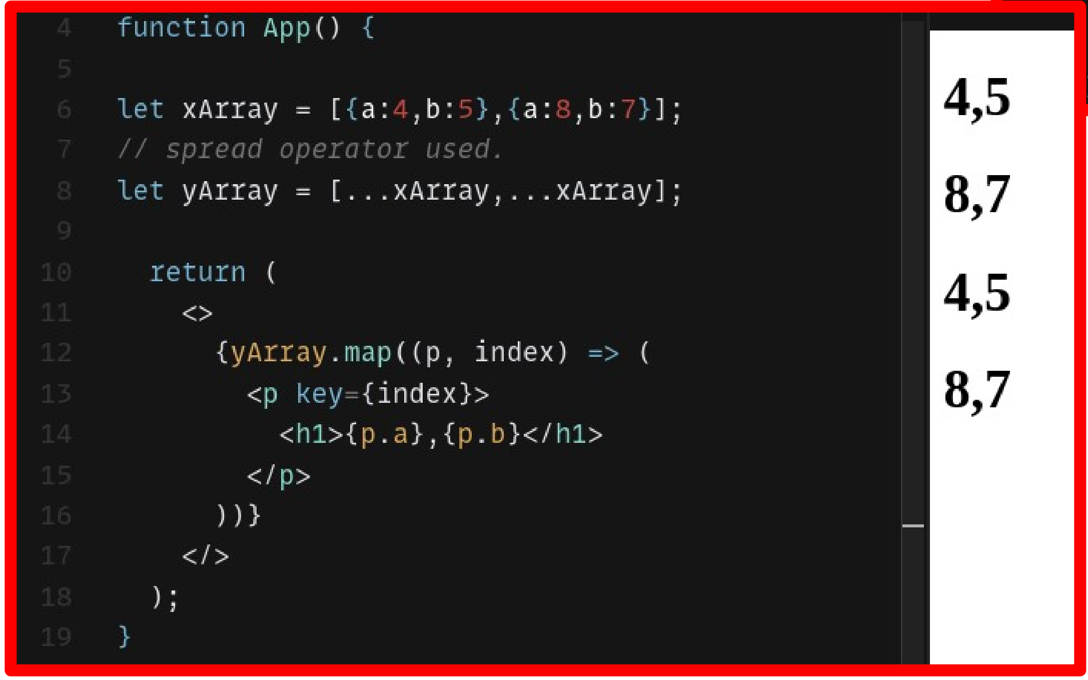
```
function App() {

let xArray = [{a:4,b:5},{a:8,b:7}];

  return (
    <>
      {xArray.map((p, index) => (
        <p key={index}>
          <h1>{p.a},{p.b}</h1>
        </p>
      ))}
    </>
  );
}
```

```
4,5

8,7
```



```
function App() {

let xArray = [{a:4,b:5},{a:8,b:7}];
// spread operator used.
let yArray = [...xArray,{a:19,b:20}];

  return (
    <>
      {yArray.map((p, index) => (
        <p key={index}>
          <h1>{p.a},{p.b}</h1>
        </p>
      ))}
    </>
  );
}
```

```
4,5

8,7

19,20
```



```
function App() {

let xArray = [{a:4,b:5},{a:8,b:7}];
// spread operator used.
let yArray = [...xArray,...xArray];

  return (
    <>
      {yArray.map((p, index) => (
        <p key={index}>
          <h1>{p.a},{p.b}</h1>
        </p>
      ))}
    </>
  );
}
```

```
4,5

8,7

4,5

8,7
```

Mutating state

- **Mutating state in functional React components can lead to unexpected behavior and bugs.**
- Modifying State Objects: If your state is an object or an array, directly modifying its properties or elements can lead to unexpected behavior.
- **To avoid mutating an array in state within a functional React component, you should always create a new array when you need to update or modify the state.**

Avoiding mutating state with arrays in React Javascript

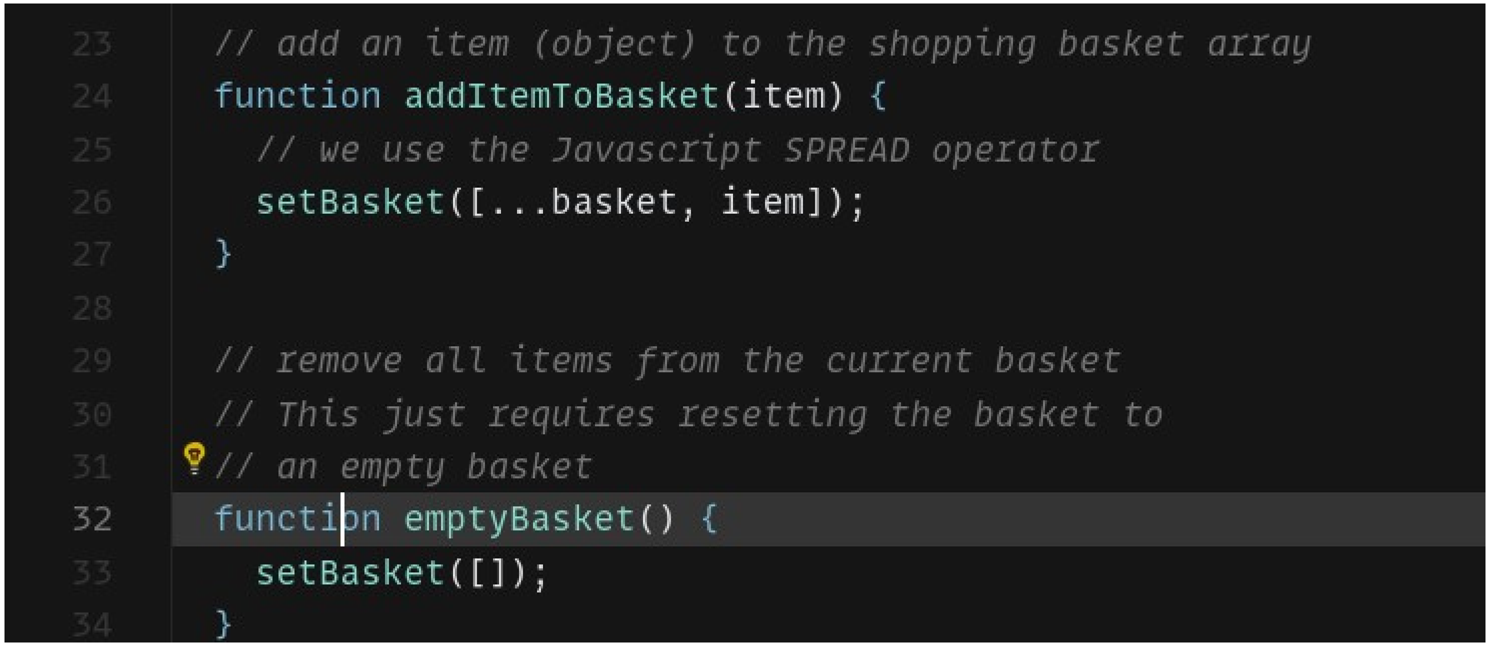
- **Creating a new array allows you to maintain the immutability of the state**, which is important for React to detect and handle state changes correctly.
- **You should create a copy of the array and update the copy instead of modifying the original array directly. You can use the spread operator (...) to create a new array with the desired changes.**
- **The key is to always create a new array or a new state object when updating the state within a functional React component.**

Mutating state TL;DR

- Always use the spread operator when working with arrays in state within React

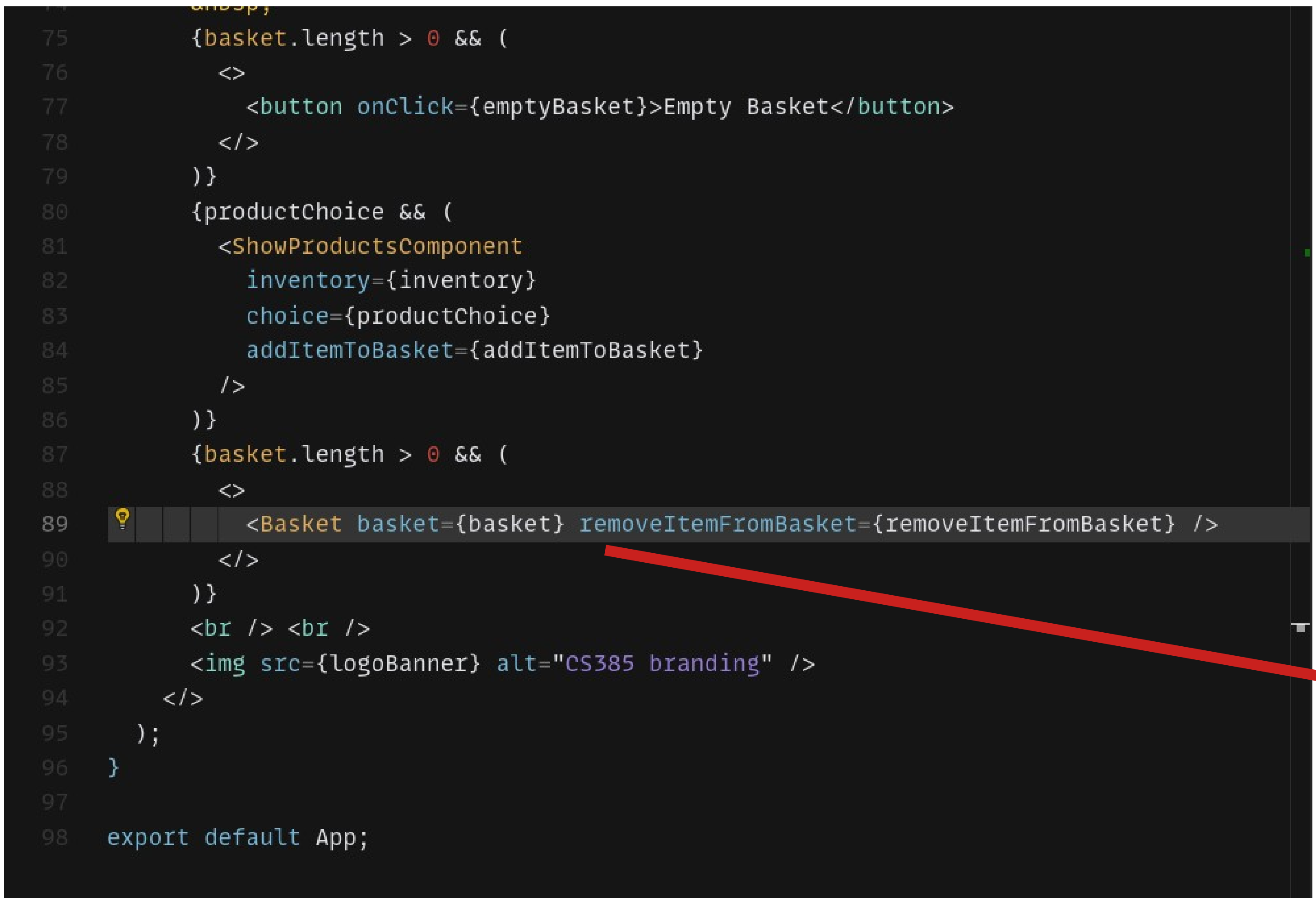
To empty the basket (or reset an array to empty) is very easy

- There are no issues around mutation of state here. We just set the array to an empty array []

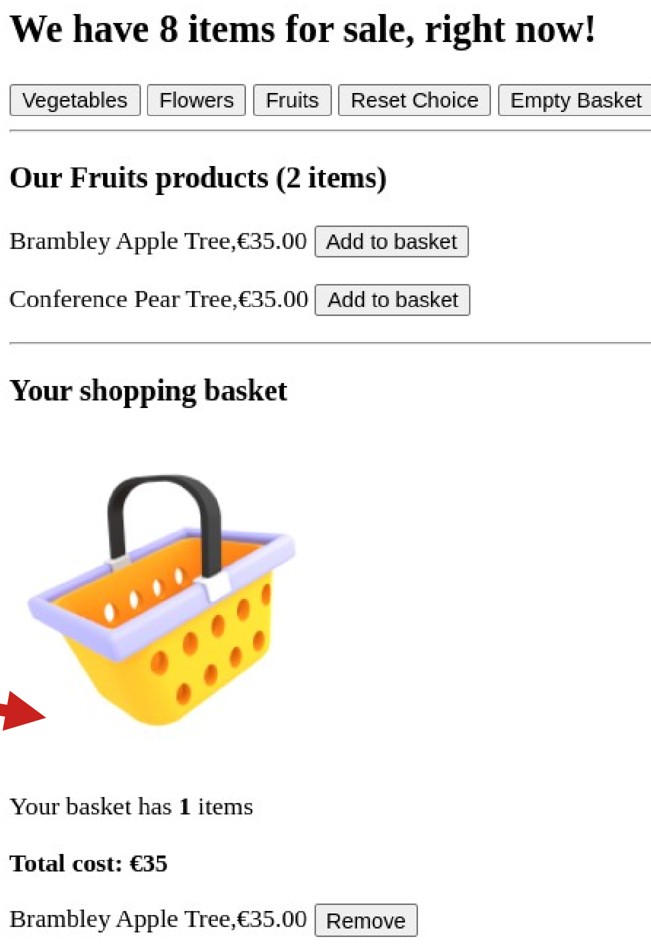


```
// add an item (object) to the shopping basket array
function addItemToBasket(item) {
  // we use the Javascript SPREAD operator
  setBasket([...basket, item]);
}

// remove all items from the current basket
// This just requires resetting the basket to
// an empty basket
function emptyBasket() {
  setBasket([]);
}
```

When we add one item to the basket – conditional rendering them invokes the Basket component



```
        {basket.length > 0 && (
          <>
            <button onClick={emptyBasket}>Empty Basket</button>
          </>
        )}
        {productChoice && (
          <ShowProductsComponent
            inventory={inventory}
            choice={productChoice}
            addItemToBasket={addItemToBasket}
          />
        )}
        {basket.length > 0 && (
          <>
            <Basket basket={basket} removeItemFromBasket={removeItemFromBasket} />
          </>
        )}
        <br /> <br />
        <img src={logoBanner} alt="CS385 branding" />
      </>
    );
  }

export default App;
```



We have 8 items for sale, right now!

Vegetables | Flowers | Fruits | Reset Choice | Empty Basket

Our Fruits products (2 items)

Brambley Apple Tree,€35.00 Add to basket

Conference Pear Tree,€35.00 Add to basket

Your shopping basket

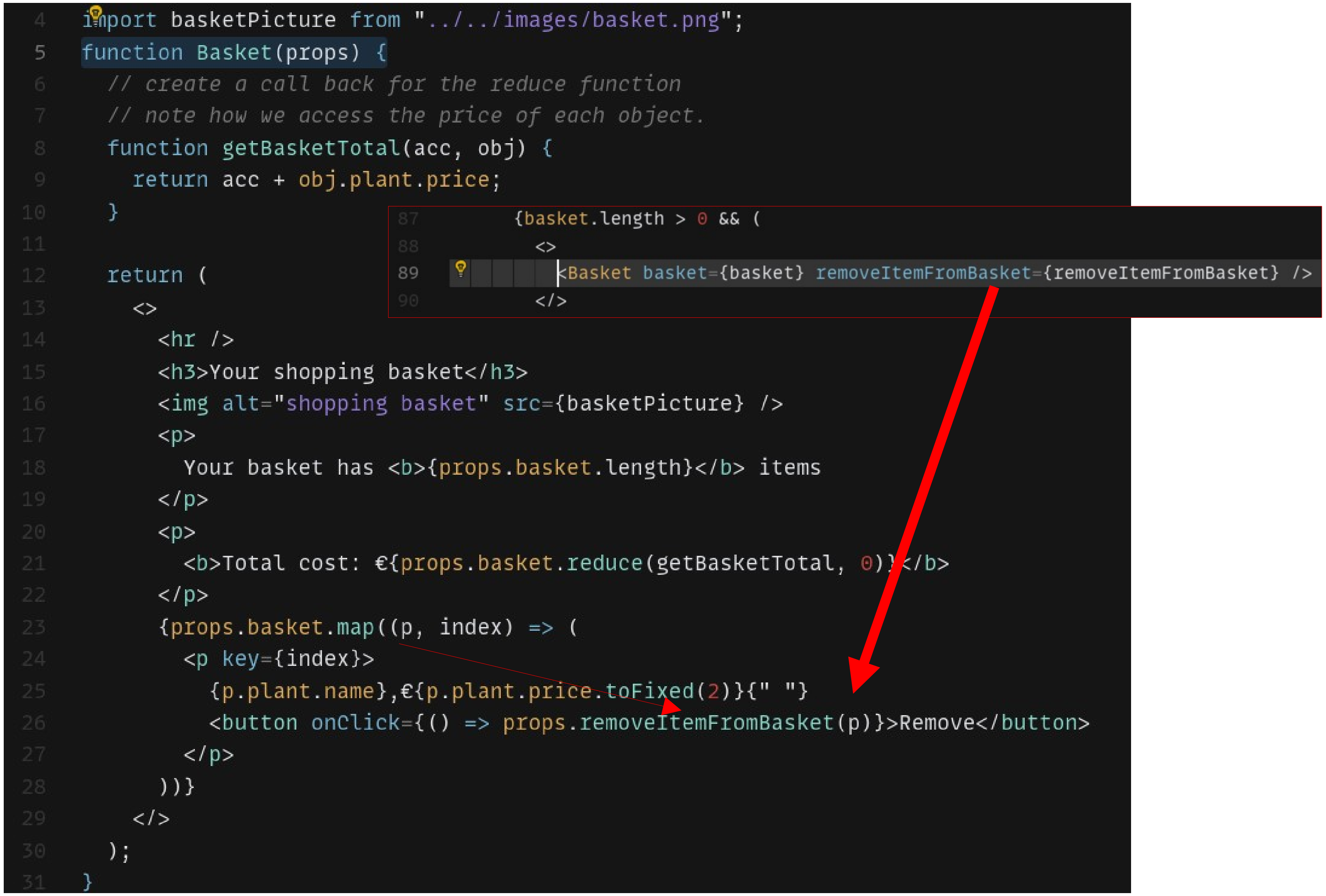
Your basket has **1** items

Total cost: €35

Brambley Apple Tree,€35.00 Remove

The Basket Component

- This is a simple component but we have a BUTTON to allow us REMOVE an item from the shopping basket



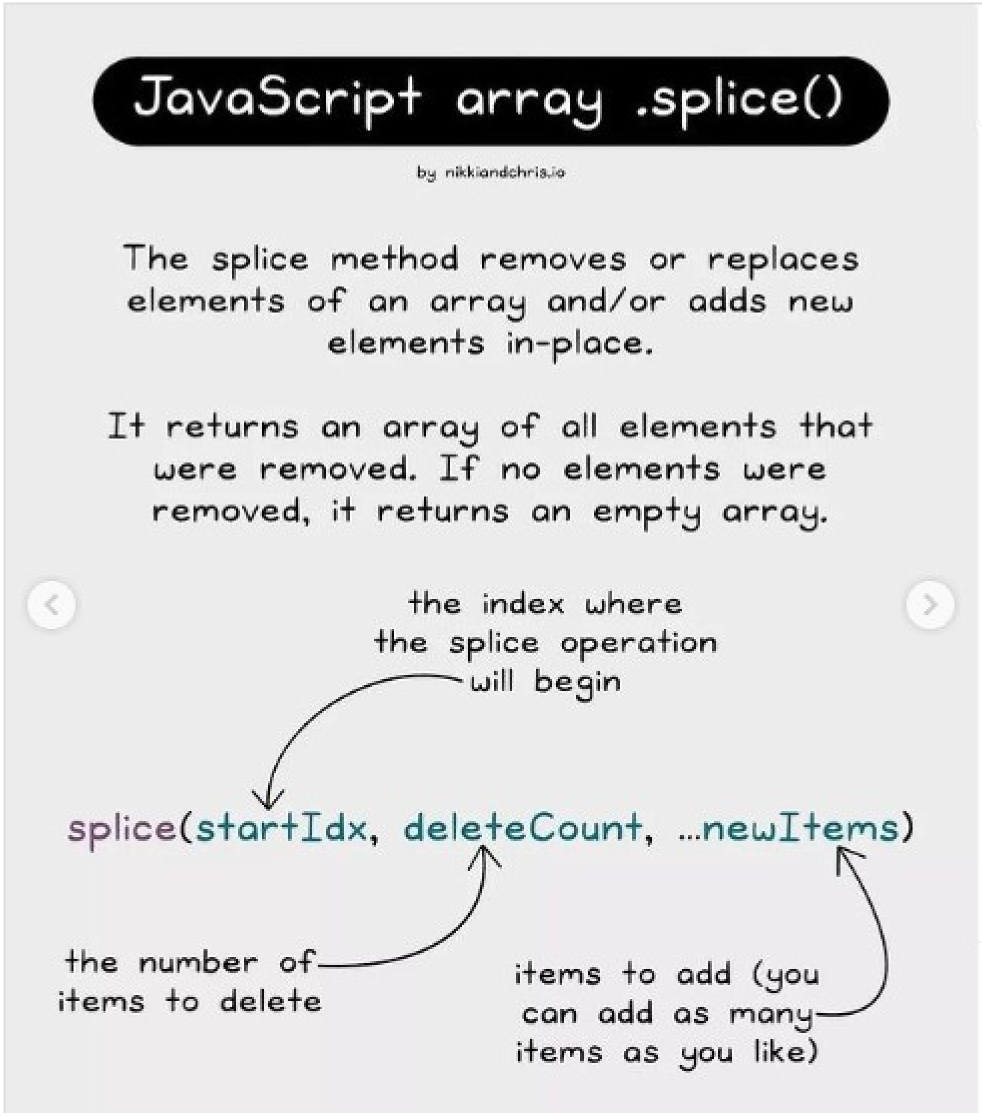
```
import basketPicture from "../../images/basket.png";
function Basket(props) {
  // create a call back for the reduce function
  // note how we access the price of each object.
  function getBasketTotal(acc, obj) {
    return acc + obj.plant.price;
  }

  return (
    <>
      <hr />
      <h3>Your shopping basket</h3>
      <img alt="shopping basket" src={basketPicture} />
      <p>
        Your basket has <b>{props.basket.length}</b> items
      </p>
      <p>
        <b>Total cost: €{props.basket.reduce(getBasketTotal, 0)}</b>
      </p>
      {props.basket.map((p, index) => (
        <p key={index}>
          {p.plant.name},€{p.plant.price.toFixed(2)}{" "}
          <button onClick={() => props.removeItemFromBasket(p)}>Remove</button>
        </p>
      ))}
    </>
  );
}
```

```
        {basket.length > 0 && (
          <>
            <Basket basket={basket} removeItemFromBasket={removeItemFromBasket} />
          </>
```

Removing objects from an array in state is complicated

- We have to avoid mutation of state (so we must create a new array)
- We must find the object's position in the array so that we can remove/delete it
- We must then effectively **SPLIT, SLICE or SPLICE** the array – by deleting the object and then "glueing" the two other parts of the array back together.
- It is a complicated process

How to splice an array



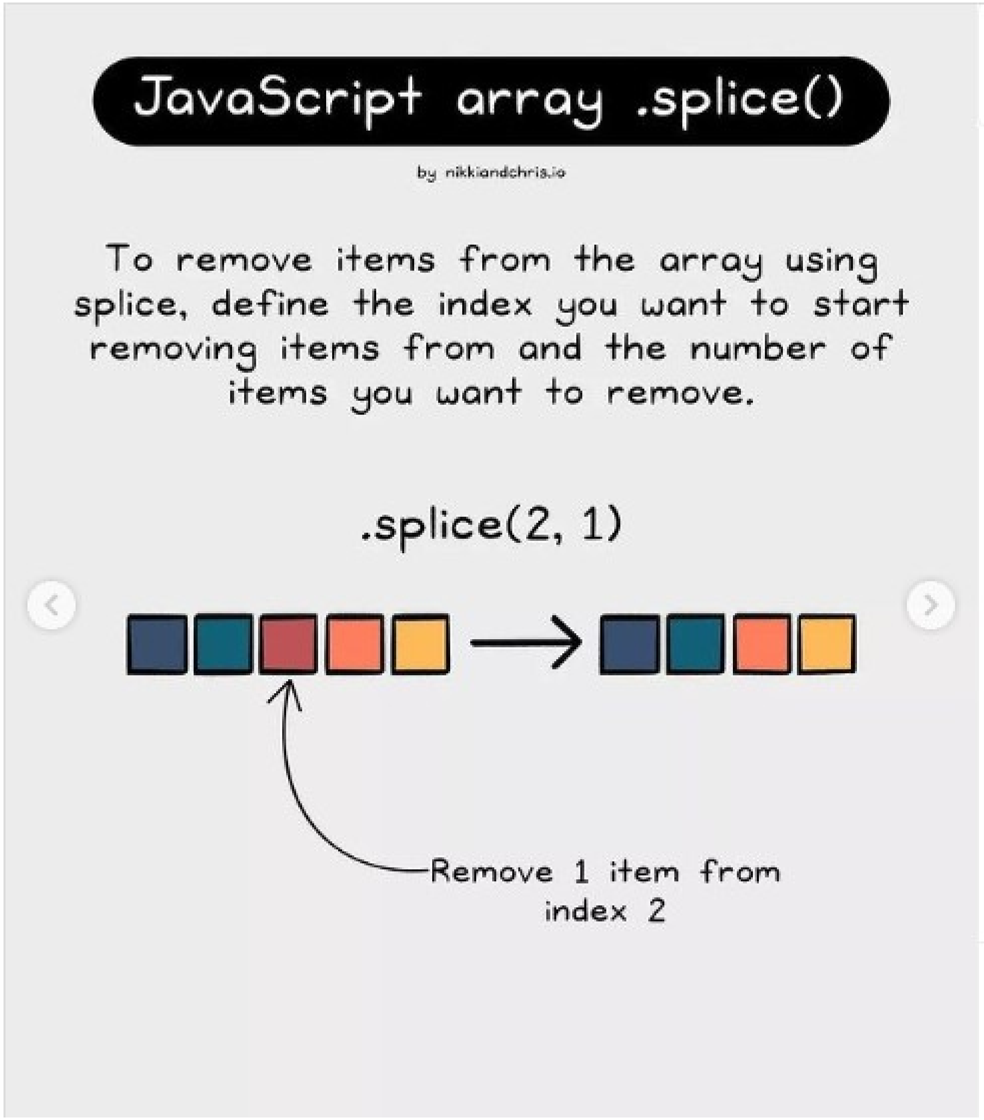
JavaScript array .splice()

by nikkiandchris.io

The splice method removes or replaces elements of an array and/or adds new elements in-place.

It returns an array of all elements that were removed. If no elements were removed, it returns an empty array.

the index where the splice operation will begin
splice(startIdx, deleteCount, ...newItems)
the number of items to delete
items to add (you can add as many items as you like)



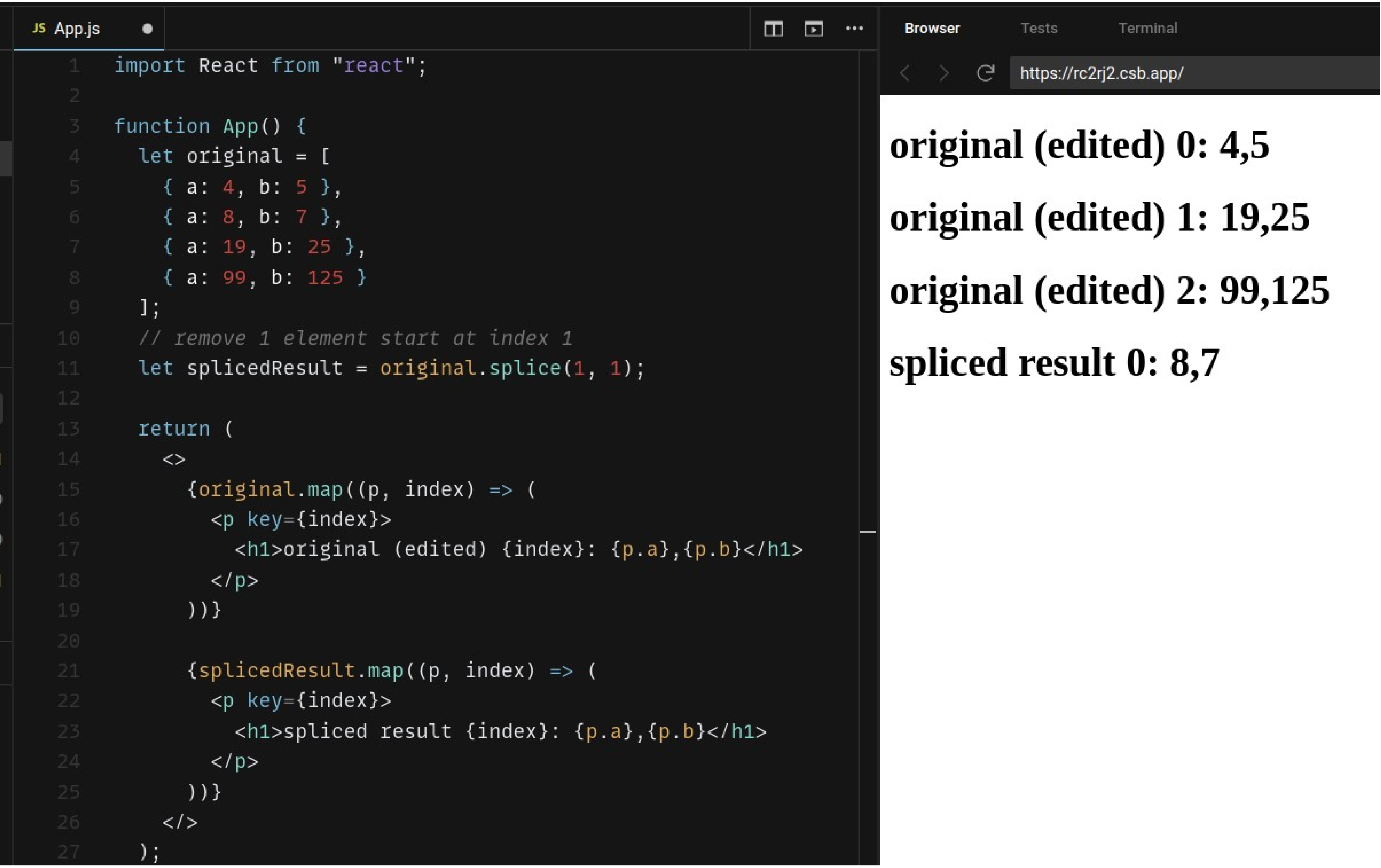
JavaScript array .splice()

by nikkiandchris.io

To remove items from the array using splice, define the index you want to start removing items from and the number of items you want to remove.

https://www.instagram.com/p/CxshYQiNJ8m/?img_index=3

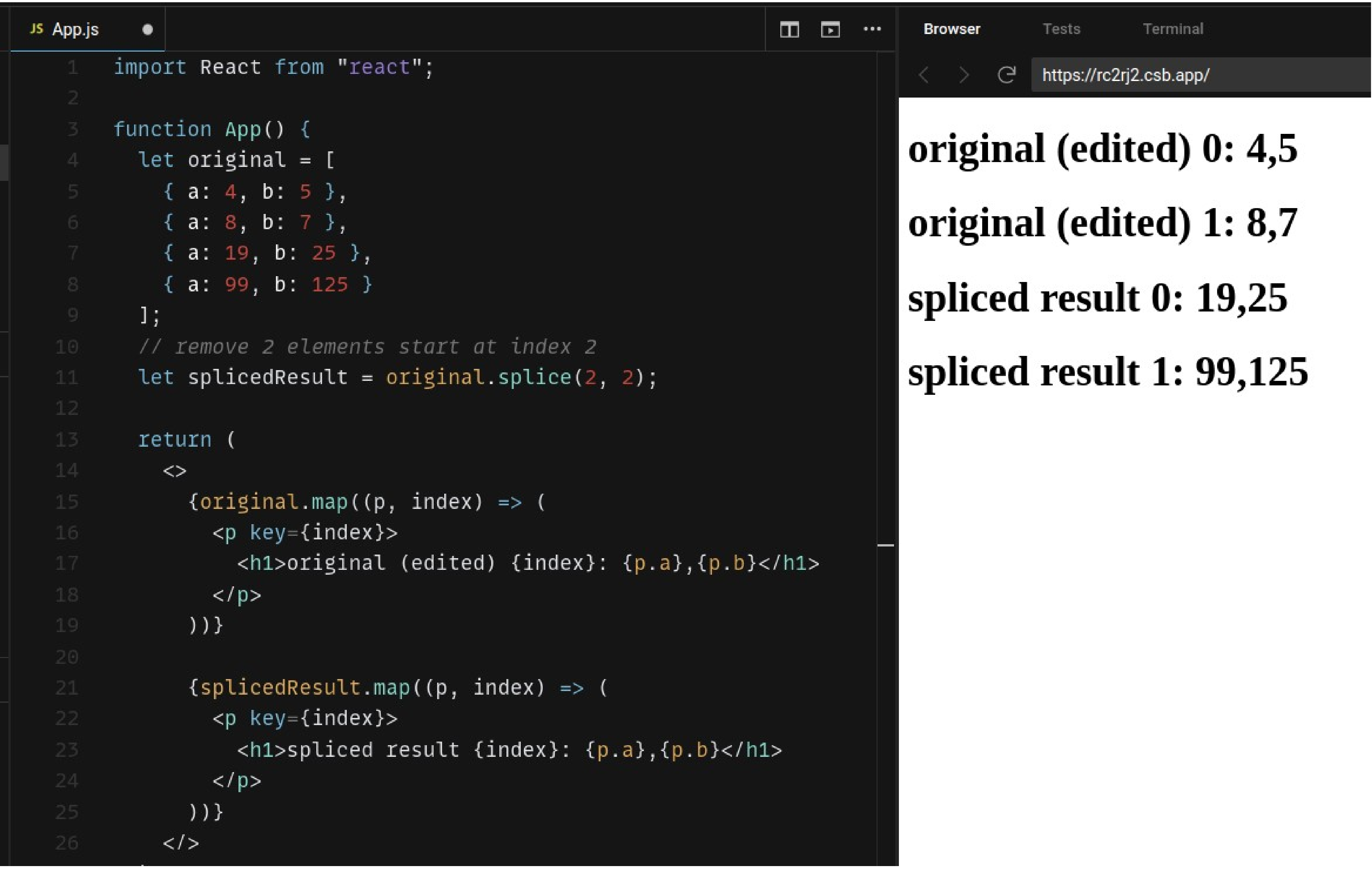
The splice operator – example 1



```
import React from "react";

function App() {
  let original = [
    { a: 4, b: 5 },
    { a: 8, b: 7 },
    { a: 19, b: 25 },
    { a: 99, b: 125 }
  ];
  // remove 1 element start at index 1
  let splicedResult = original.splice(1, 1);

  return (
    <>
      {original.map((p, index) => (
        <p key={index}>
          <h1>original (edited) {index}: {p.a},{p.b}</h1>
        </p>
      ))}

      {splicedResult.map((p, index) => (
        <p key={index}>
          <h1>spliced result {index}: {p.a},{p.b}</h1>
        </p>
      ))}
    </>
  );
```

```
original (edited) 0: 4,5

original (edited) 1: 19,25

original (edited) 2: 99,125

spliced result 0: 8,7
```

The splice operator – example 2



App.js

```
import React from "react";

function App() {
  let original = [
    { a: 4, b: 5 },
    { a: 8, b: 7 },
    { a: 19, b: 25 },
    { a: 99, b: 125 }
  ];
  // remove 2 elements start at index 2
  let splicedResult = original.splice(2, 2);

  return (
    <>
      {original.map((p, index) => (
        <p key={index}>
          <h1>original (edited) {index}: {p.a},{p.b}</h1>
        </p>
      ))}

      {splicedResult.map((p, index) => (
        <p key={index}>
          <h1>spliced result {index}: {p.a},{p.b}</h1>
        </p>
      ))}
    </>
```

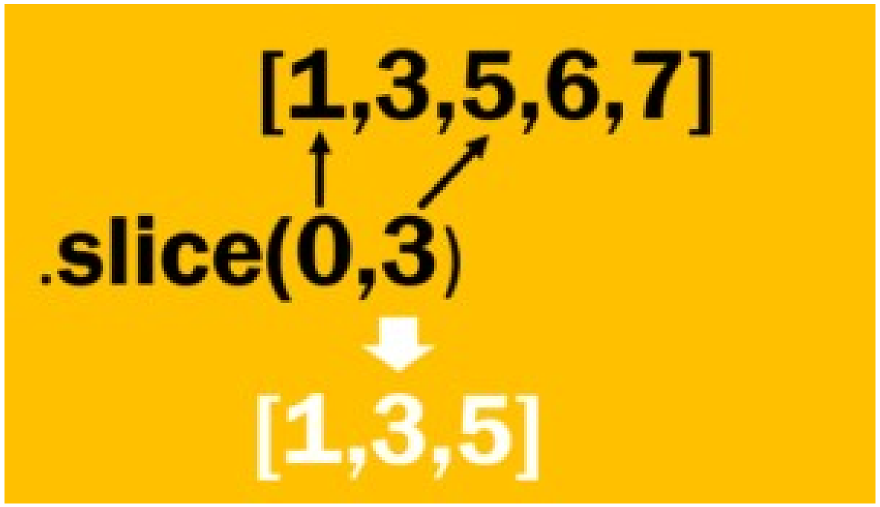
Browser Tests Terminal

https://rc2rj2.csb.app/

```
original (edited) 0: 4,5
original (edited) 1: 8,7
spliced result 0: 19,25
spliced result 1: 99,125
```

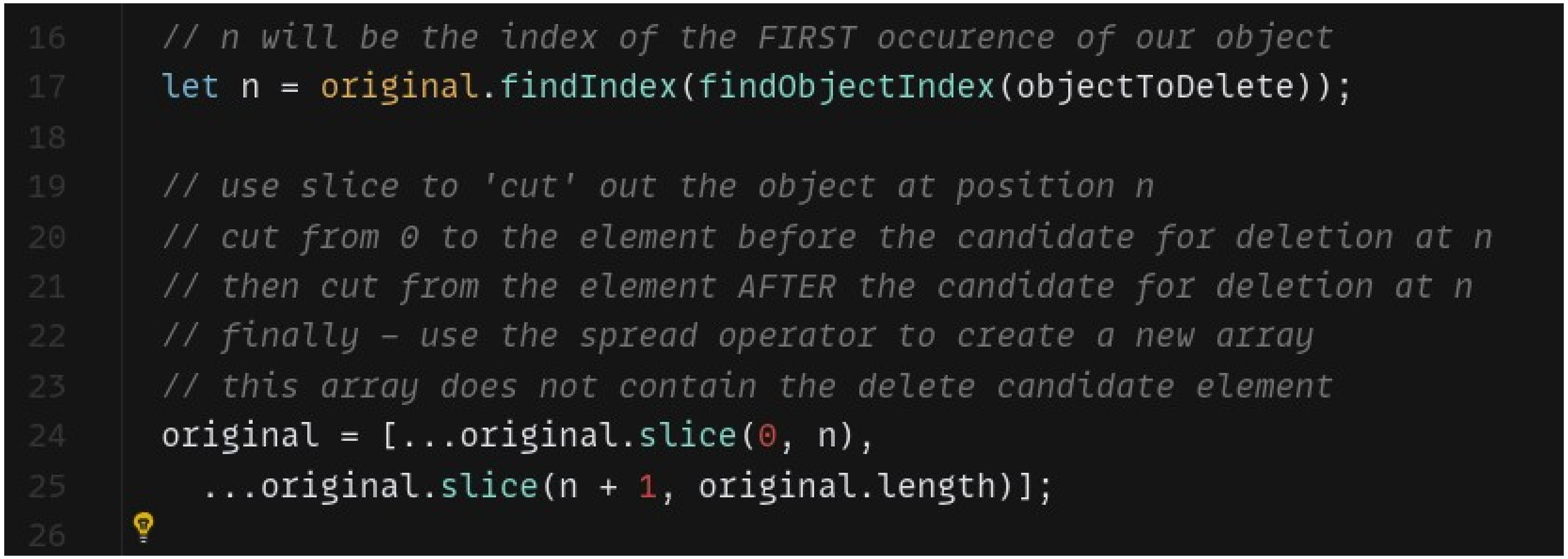
The slice operator

- The slice operator allows us to cut a portion away from a given array
- **It is very useful for removing elements in an array by removing the elements we want to keep or retain in the array**



The slice operator

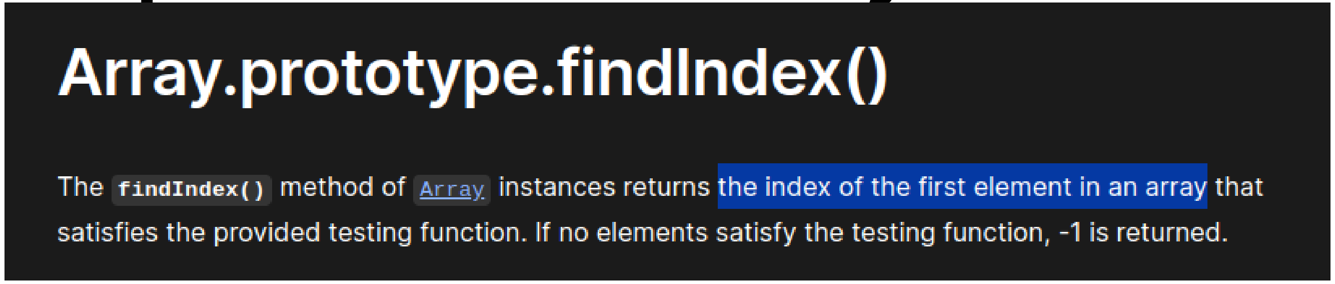
- Example to follow .....



```
// n will be the index of the FIRST occurence of our object
let n = original.findIndex(findObjectIndex(objectToDelete));

// use slice to 'cut' out the object at position n
// cut from 0 to the element before the candidate for deletion at n
// then cut from the element AFTER the candidate for deletion at n
// finally - use the spread operator to create a new array
// this array does not contain the delete candidate element
original = [...original.slice(0, n),
  ...original.slice(n + 1, original.length)];

```

Steps to remove an object from an array in state

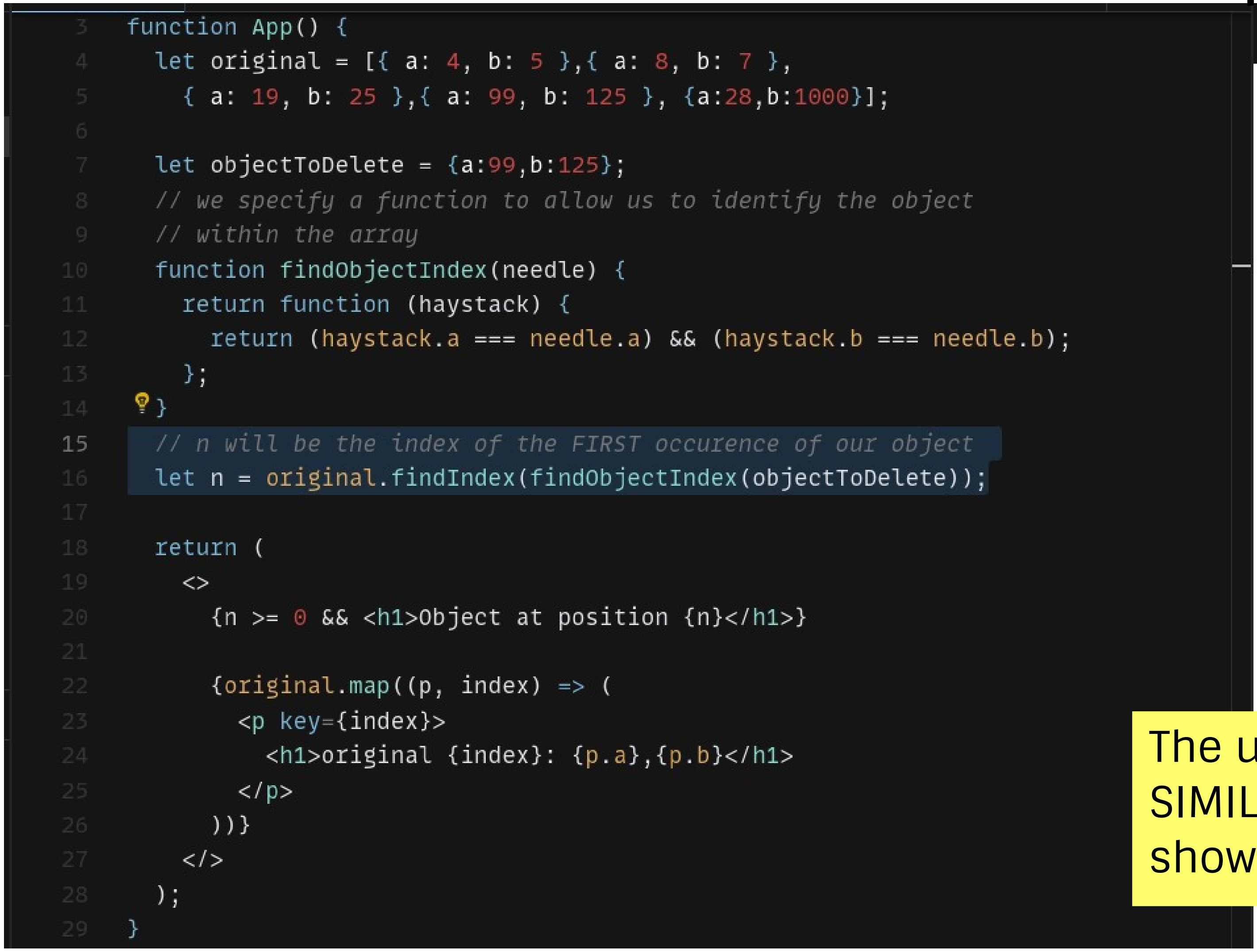
- **Step 1 – we need to FIND the location of the object in the array** – what index is the object currently located it. (use findIndex from Javascript)
- **Step 2** – if the object is in the array (findIndex returns >= 0) then **we need to use the splice operator to 'cut' or 'slice' the array at the index position of our object.**


Array.prototype.findIndex()

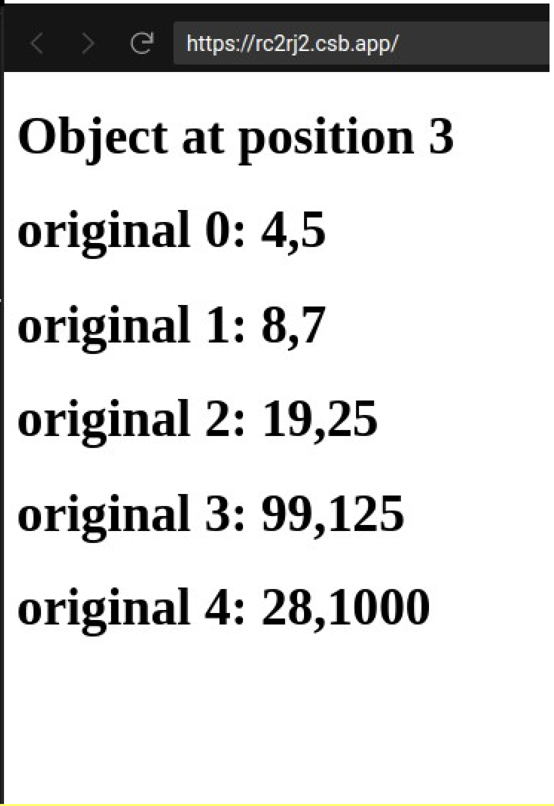
The `findIndex()` method of `Array` instances returns the index of the first element in an array that satisfies the provided testing function. If no elements satisfy the testing function, -1 is returned.

EXAMPLE: Part 1: Let's see a simple example of findIndex and slice

- We want to delete {a:99, b:25} at position 3



```
function App() {
  let original = [{ a: 4, b: 5 },{ a: 8, b: 7 },
    { a: 19, b: 25 },{ a: 99, b: 125 }, {a:28,b:1000}];

  let objectToDelete = {a:99,b:125};
  // we specify a function to allow us to identify the object
  // within the array
  function findObjectIndex(needle) {
    return function (haystack) {
      return (haystack.a === needle.a) && (haystack.b === needle.b);
    };
  }
  // n will be the index of the FIRST occurence of our object
  let n = original.findIndex(findObjectIndex(objectToDelete));

  return (
    <>
      {n >= 0 && <h1>Object at position {n}</h1>}

      {original.map((p, index) => (
        <p key={index}>
          <h1>original {index}: {p.a},{p.b}</h1>
        </p>
      ))}
    </>
  );
}
```



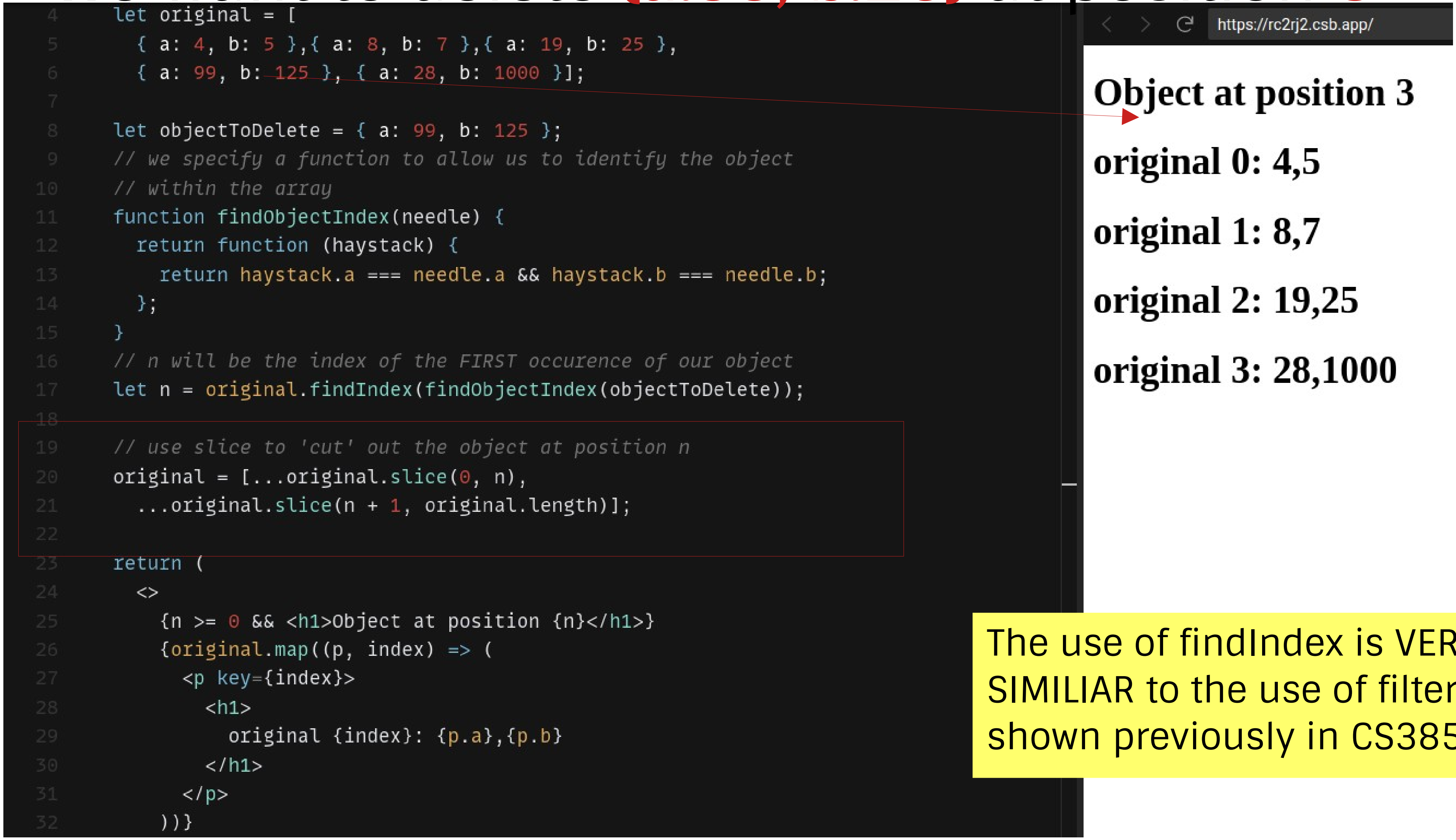
https://rc2rj2.csb.app/

```
Object at position 3
original 0: 4,5
original 1: 8,7
original 2: 19,25
original 3: 99,125
original 4: 28,1000
```

The use of findIndex is VERY SIMILIAR to the use of filter as shown previously in CS385

EXAMPLE: Part 2: Let's see a simple example of findIndex and slice

- We want to delete {a:99, b:25} at position 3



```
    let original = [
      { a: 4, b: 5 },{ a: 8, b: 7 },{ a: 19, b: 25 },
      { a: 99, b: 125 }, { a: 28, b: 1000 }];

    let objectToDelete = { a: 99, b: 125 };
    // we specify a function to allow us to identify the object
    // within the array
    function findObjectIndex(needle) {
      return function (haystack) {
        return haystack.a === needle.a && haystack.b === needle.b;
      };
    }
    // n will be the index of the FIRST occurence of our object
    let n = original.findIndex(findObjectIndex(objectToDelete));

    // use slice to 'cut' out the object at position n
    original = [...original.slice(0, n),
      ...original.slice(n + 1, original.length)];

    return (
      <>
        {n >= 0 && <h1>Object at position {n}</h1>}
        {original.map((p, index) => (
          <p key={index}>
            <h1>
              original {index}: {p.a},{p.b}
            </h1>
          </p>
        ))}
```

https://rc2rj2.csb.app/

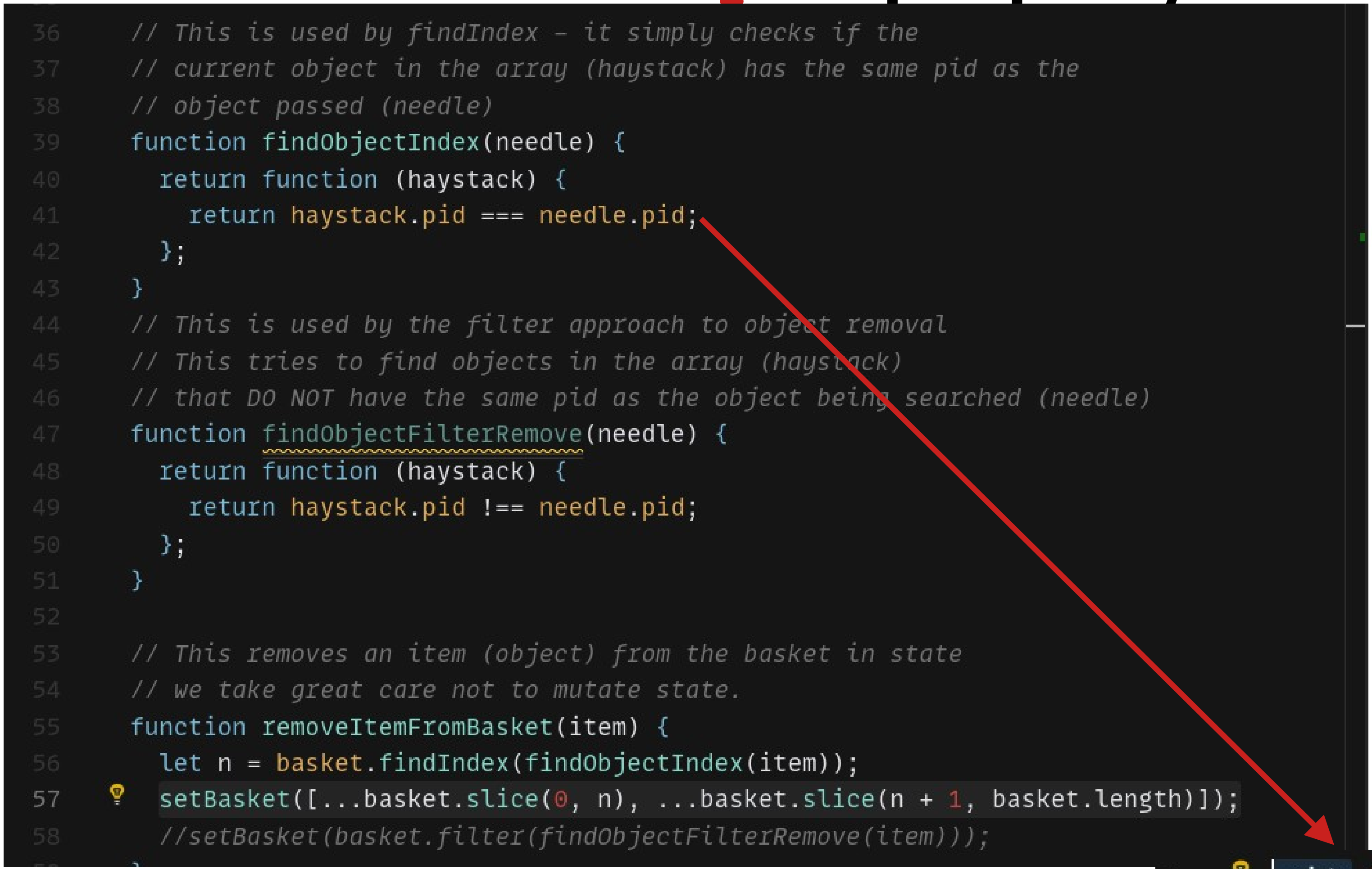
```
Object at position 3

original 0: 4,5

original 1: 8,7

original 2: 19,25

original 3: 28,1000
```

The use of findIndex is VERY SIMILIAR to the use of filter as shown previously in CS385

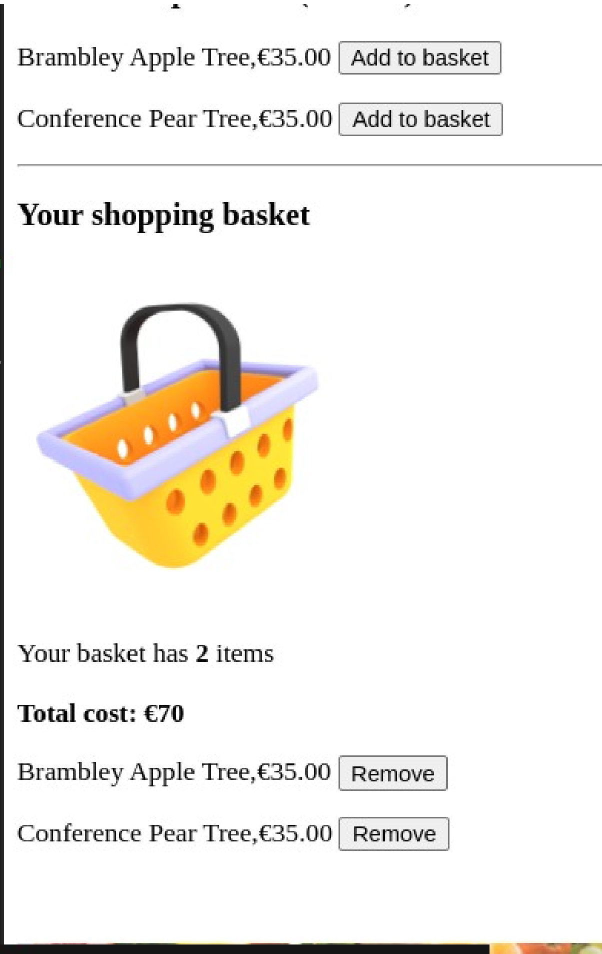
Let's return to the organic shop application code

- **We can see the REMOVING an element from an array is actually a very complicated process.**
- However, it works the same for ANY object array.
- All you need to do is to modify the code (especially for **findIndex**) to suit your application needs.

We find the object for deletion based on the `pid` property



```
36    // This is used by findIndex - it simply checks if the
37    // current object in the array (haystack) has the same pid as the
38    // object passed (needle)
39    function findObjectIndex(needle) {
40      return function (haystack) {
41        return haystack.pid === needle.pid;
42      };
43    }
44    // This is used by the filter approach to object removal
45    // This tries to find objects in the array (haystack)
46    // that DO NOT have the same pid as the object being searched (needle)
47    function findObjectFilterRemove(needle) {
48      return function (haystack) {
49        return haystack.pid !== needle.pid;
50      };
51    }
52
53    // This removes an item (object) from the basket in state
54    // we take great care not to mutate state.
55    function removeItemFromBasket(item) {
56      let n = basket.findIndex(findObjectIndex(item));
57      setBasket([...basket.slice(0, n), ...basket.slice(n + 1, basket.length)]);
58      //setBasket(basket.filter(findObjectFilterRemove(item)));
```



Brambley Apple Tree,€35.00 Add to basket

Conference Pear Tree,€35.00 Add to basket

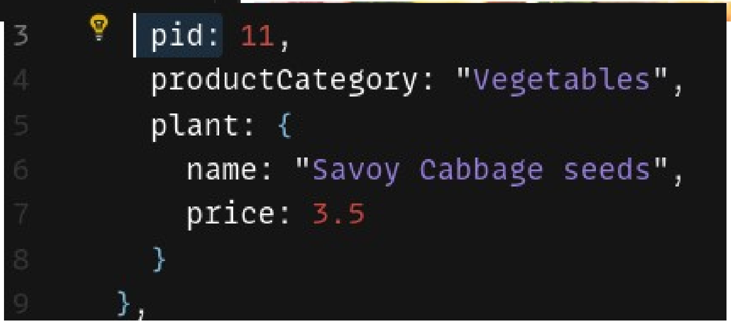
Your shopping basket

Your basket has **2** items

Total cost: €70

Brambley Apple Tree,€35.00 Remove

Conference Pear Tree,€35.00 Remove



```
3      pid: 11,
4      productCategory: "Vegetables",
5      plant: {
6        name: "Savoy Cabbage seeds",
7        price: 3.5
8      }
9    },
```

Line 57 is VERY IMPORTANT
Notice how setBasket is used – in combination with the SPREAD operator and the SLICE operator

Ways to improve the application

- **Start to use Bootstrap** (Lecture 13 – 14) for a more attractive user interface
- **Sort the elements by price, or by name**, etc (Lecture 13 – 14)
- **Use an API** (Lecture 13 – 14)
- **We'll use this example as the basis for the content in Lecture 13 - 14**

Lecture 11 – CS385 Organic Shop

- **Functionalities (what we achieved)**
  - **Conditional Rendering** – Basket component, ShowProductComponent
  - **Updating state** – finding objects, deleting objects, adding objects to state arrays **[NEW]**
  - Using Javascript functions: **spread, slice, splice**, **findIndex**. **[NEW]**
  - **Using parent child communications**
  - Adding our own **images** **[NEW]**
  - **Using filter, map, reduce**

Lecture 11 – gives the framework for many CS385 applications

- **Many CS385 projects will manage state**
- In state, there will (most likely) be arrays of objects.
- **In most applications you will ADD objects, FIND objects, and DELETE objects from the arrays in state.** Each array operation will have a function or functions (such as `addToBasket`) to perform these tasks for you – maybe connected to UI elements
- **You will also use multiple components** – sharing both the arrays but also the array operation functions around via props.

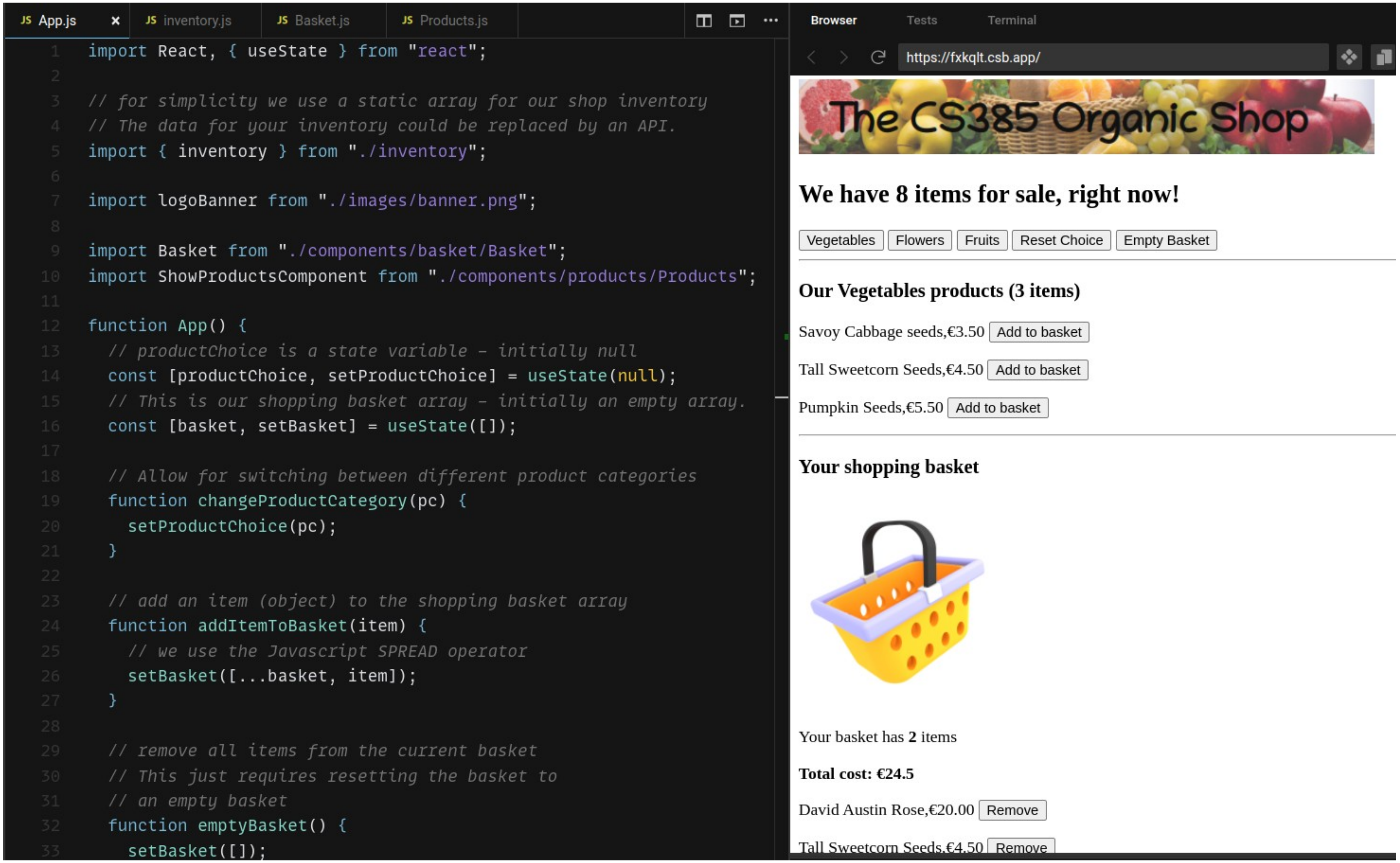
Try and test the application yourself

- **The best way to understand the code is to try it out for yourself**
- Lecture 11 source code is available on Moodle.
- Screencast also available

VERY IMPORTANT – Lab Exam 2

- **The SPREAD, SLICE and SPLICE operators will be tested in Lab Exam 2** (yes, there will be a demo lab exam 2, also)
- But the simple examples shown here will form the basis of Lab Exam 2 questions.
- You should also be familiar with the rules around avoiding mutating state with arrays

CS385 Lecture 11



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

// for simplicity we use a static array for our shop inventory
// The data for your inventory could be replaced by an API.
import { inventory } from "./inventory";

import logoBanner from "./images/banner.png";

import Basket from "./components/basket/Basket";
import ShowProductsComponent from "./components/products/Products";

function App() {
  // productChoice is a state variable - initially null
  const [productChoice, setProductChoice] = useState(null);
  // This is our shopping basket array - initially an empty array.
  const [basket, setBasket] = useState([]);

  // Allow for switching between different product categories
  function changeProductCategory(pc) {
    setProductChoice(pc);
  }

  // add an item (object) to the shopping basket array
  function addItemToBasket(item) {
    // we use the Javascript SPREAD operator
    setBasket([...basket, item]);
  }

  // remove all items from the current basket
  // This just requires resetting the basket to
  // an empty basket
  function emptyBasket() {
    setBasket([]);
```

We have 8 items for sale, right now!

Vegetables | Flowers | Fruits | Reset Choice | Empty Basket

Our Vegetables products (3 items)

Savoy Cabbage seeds,€3.50 Add to basket

Tall Sweetcorn Seeds,€4.50 Add to basket

Pumpkin Seeds,€5.50 Add to basket

Your shopping basket

Your basket has **2** items

Total cost: €24.5

David Austin Rose,€20.00 Remove

Tall Sweetcorn Seeds,€4.50 Remove