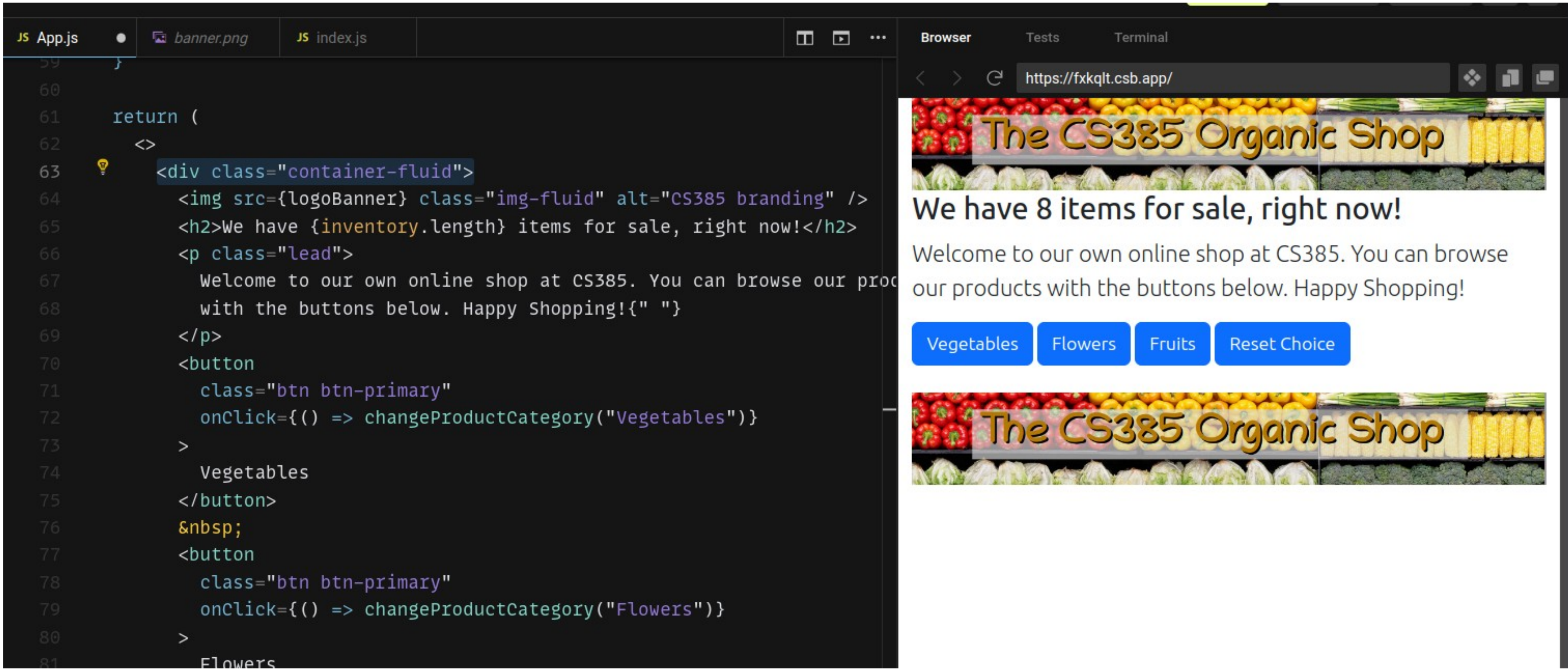
CS385 Lecture 14 – The CS385 Organic Shop app

Adding Bootstrap, sorting of objects, drop-down-list and an API link

Bootstrap – get started on some minor edits to our JSX code



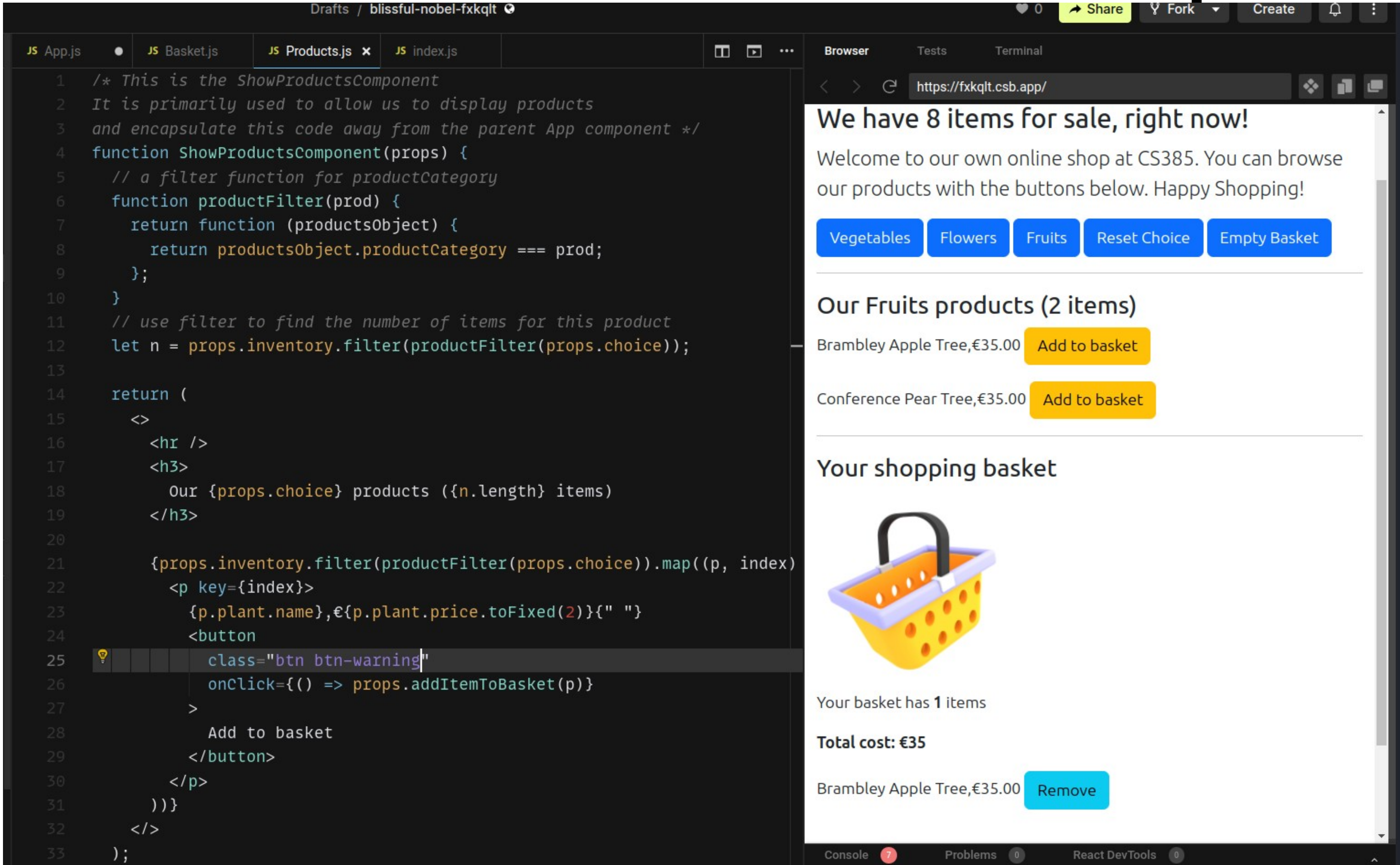
```
    }

    return (
      <>
        <div class="container-fluid">
          <img src={logoBanner} class="img-fluid" alt="CS385 branding" />
          <h2>We have {inventory.length} items for sale, right now!</h2>
          <p class="lead">
            Welcome to our own online shop at CS385. You can browse our proc
            with the buttons below. Happy Shopping!{" "}
          </p>
          <button
            class="btn btn-primary"
            onClick={() => changeProductCategory("Vegetables")}
          >
            Vegetables
          </button>
          &nbsp;
          <button
            class="btn btn-primary"
            onClick={() => changeProductCategory("Flowers")}
          >
            Flowers
```

We have 8 items for sale, right now!

Welcome to our own online shop at CS385. You can browse our products with the buttons below. Happy Shopping!

You will only have to make minor changes – all you are doing is changing the STYLE of your User Interface. You are not changing functionality!

TIP – get functionality working first – then include Bootstrap



```
/* This is the ShowProductsComponent
It is primarily used to allow us to display products
and encapsulate this code away from the parent App component */
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
            class="btn btn-warning"
            onClick={() => props.addItemToBasket(p)}
          >
            Add to basket
          </button>
        </p>
      ))}
    </>
  );
```

We have 8 items for sale, right now!

Welcome to our own online shop at CS385. You can browse our products with the buttons below. Happy Shopping!

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

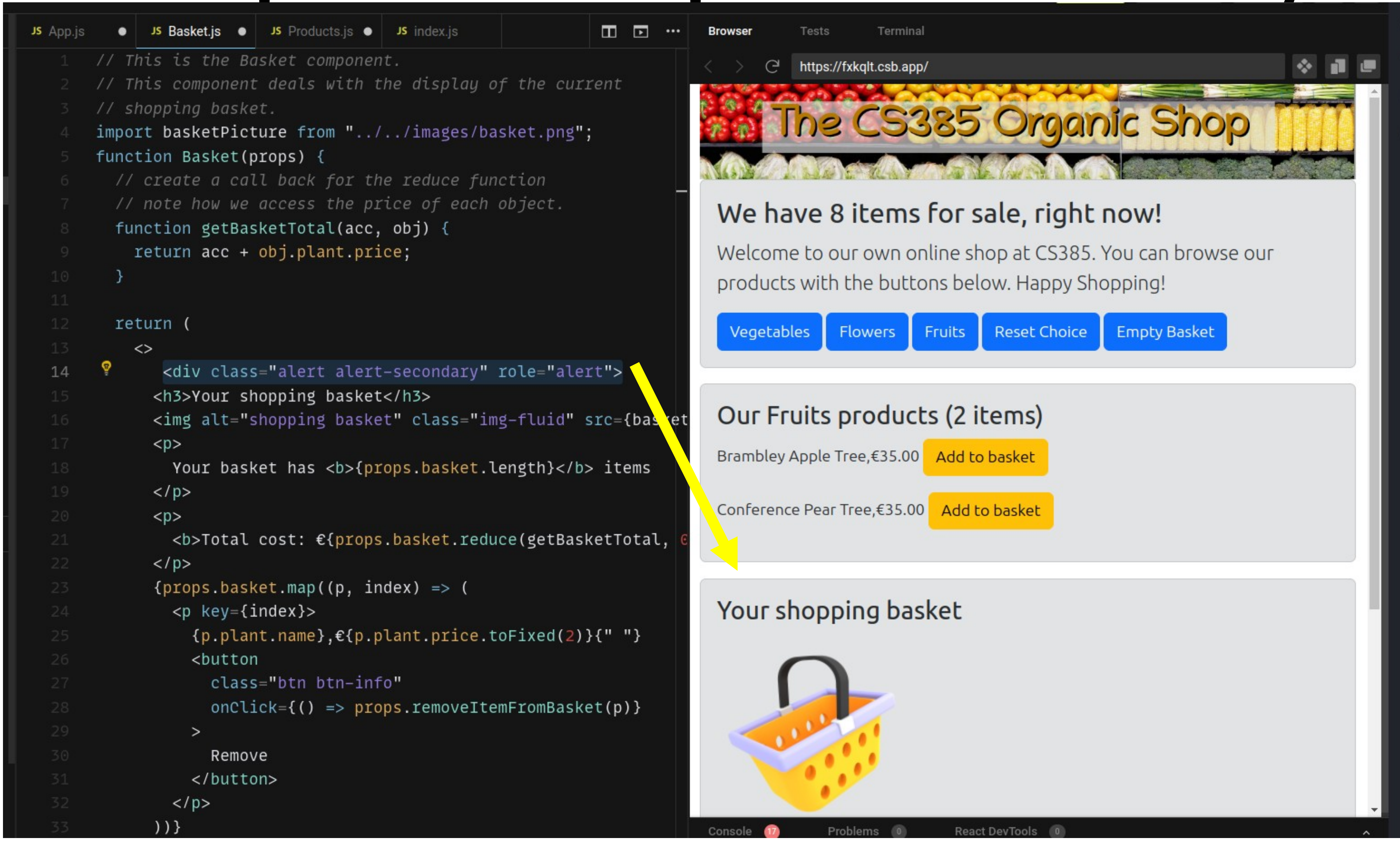
Your shopping basket

Your basket has **1** items

Total cost: €35

Brambley Apple Tree,€35.00 Remove

The `<div>` alerts are an easy way to separate components visually



```
// 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;
  }

  return (
    <>
      <div class="alert alert-secondary" role="alert">
        <h3>Your shopping basket</h3>
        <img alt="shopping basket" class="img-fluid" src={basket
        <p>
          Your basket has <b>{props.basket.length}</b> items
        </p>
        <p>
          <b>Total cost: €{props.basket.reduce(getBasketTotal,
        </p>
        {props.basket.map((p, index) => (
          <p key={index}>
            {p.plant.name},€{p.plant.price.toFixed(2)}{" "}
            <button
              class="btn btn-info"
              onClick={() => props.removeItemFromBasket(p)}
            >
              Remove
            </button>
          </p>
        ))}
```

The CS385 Organic Shop

We have 8 items for sale, right now!

Welcome to our own online shop at CS385. You can browse our products with the buttons below. Happy Shopping!

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

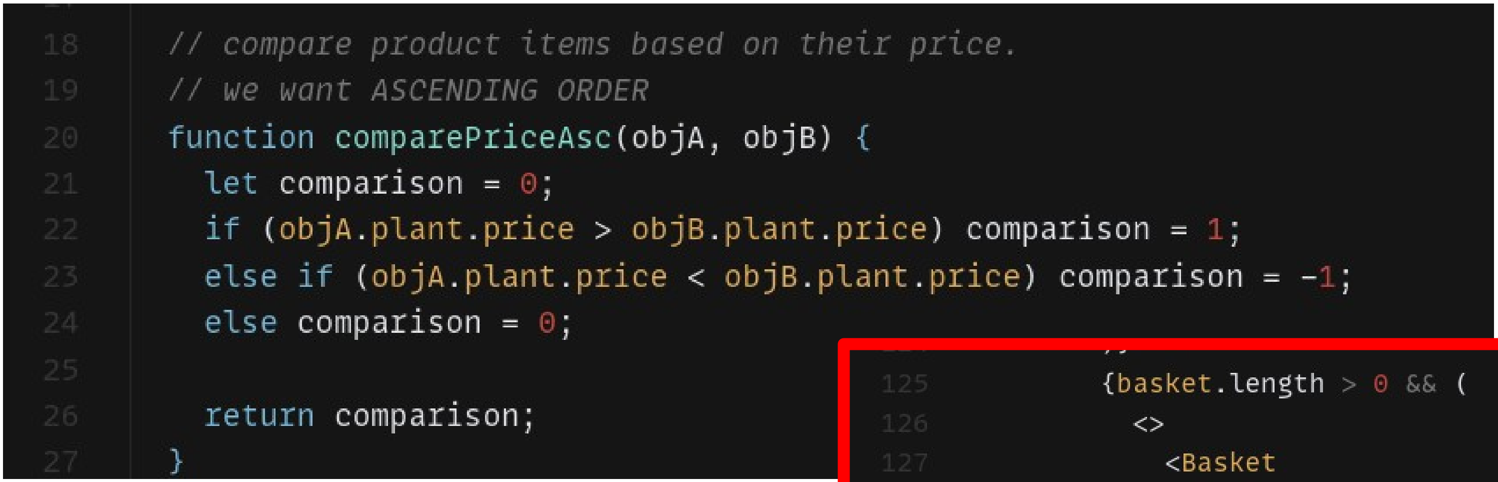
Bootstrap in CS385

- There’s no need to spend hours and hours working on Bootstrap for your project.
- Work on functionality first – then work on integrating Bootstrap.
- Keep your user-interface simple – then you will just need a minimal Bootstrap implementation.
- **That’s all I’ll be showing you for Bootstrap in CS385. Feel free to explore the documentation and even the other CSS frameworks.**

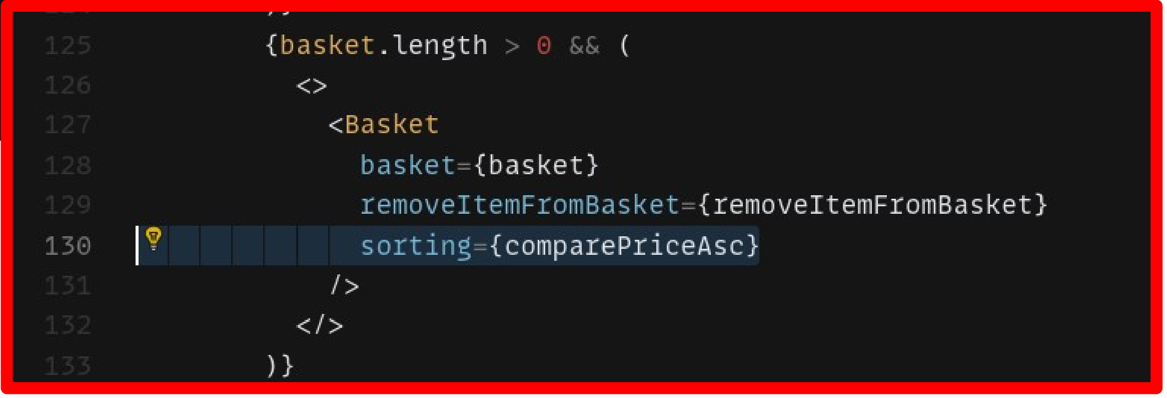
Implementing sorting

Let’s sort by PRICE in ascending order within Basket component

- It is a good idea to define your sorting comparator functions in the PARENT or highest component – then pass via props.
  Write once – use many times

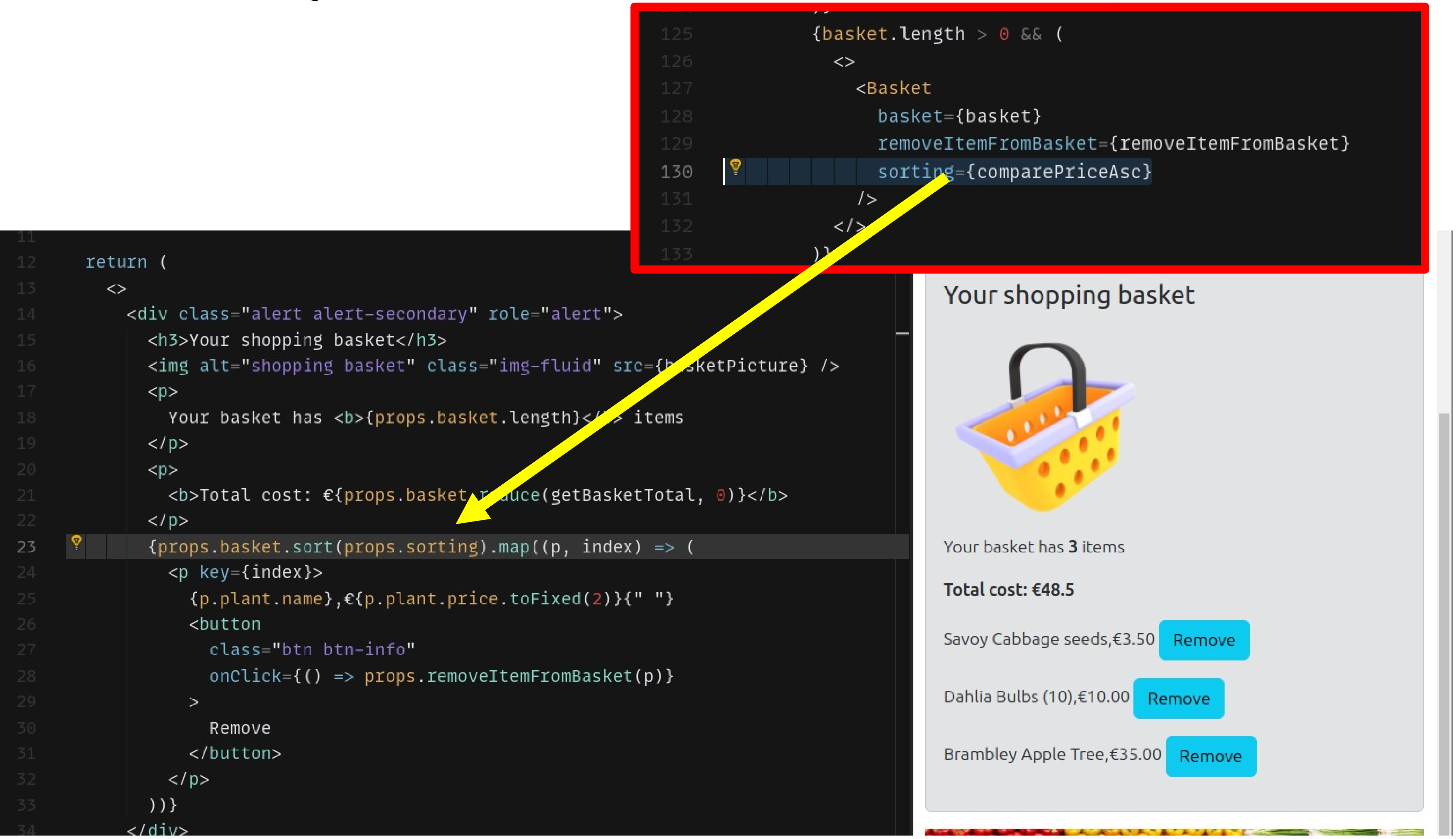


```
// compare product items based on their price.
// we want ASCENDING ORDER
function comparePriceAsc(objA, objB) {
  let comparison = 0;
  if (objA.plant.price > objB.plant.price) comparison = 1;
  else if (objA.plant.price < objB.plant.price) comparison = -1;
  else comparison = 0;

  return comparison;
}
```



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

Sort by price – Basket component



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

```
  return (
    <>
      <div class="alert alert-secondary" role="alert">
        <h3>Your shopping basket</h3>
        <img alt="shopping basket" class="img-fluid" 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.sort(props.sorting).map((p, index) => (
          <p key={index}>
            {p.plant.name},€{p.plant.price.toFixed(2)}{" "}
            <button
              class="btn btn-info"
              onClick={() => props.removeItemFromBasket(p)}
            >
              Remove
            </button>
          </p>
        ))}
      </div>
```

Your shopping basket

Your basket has **3** items

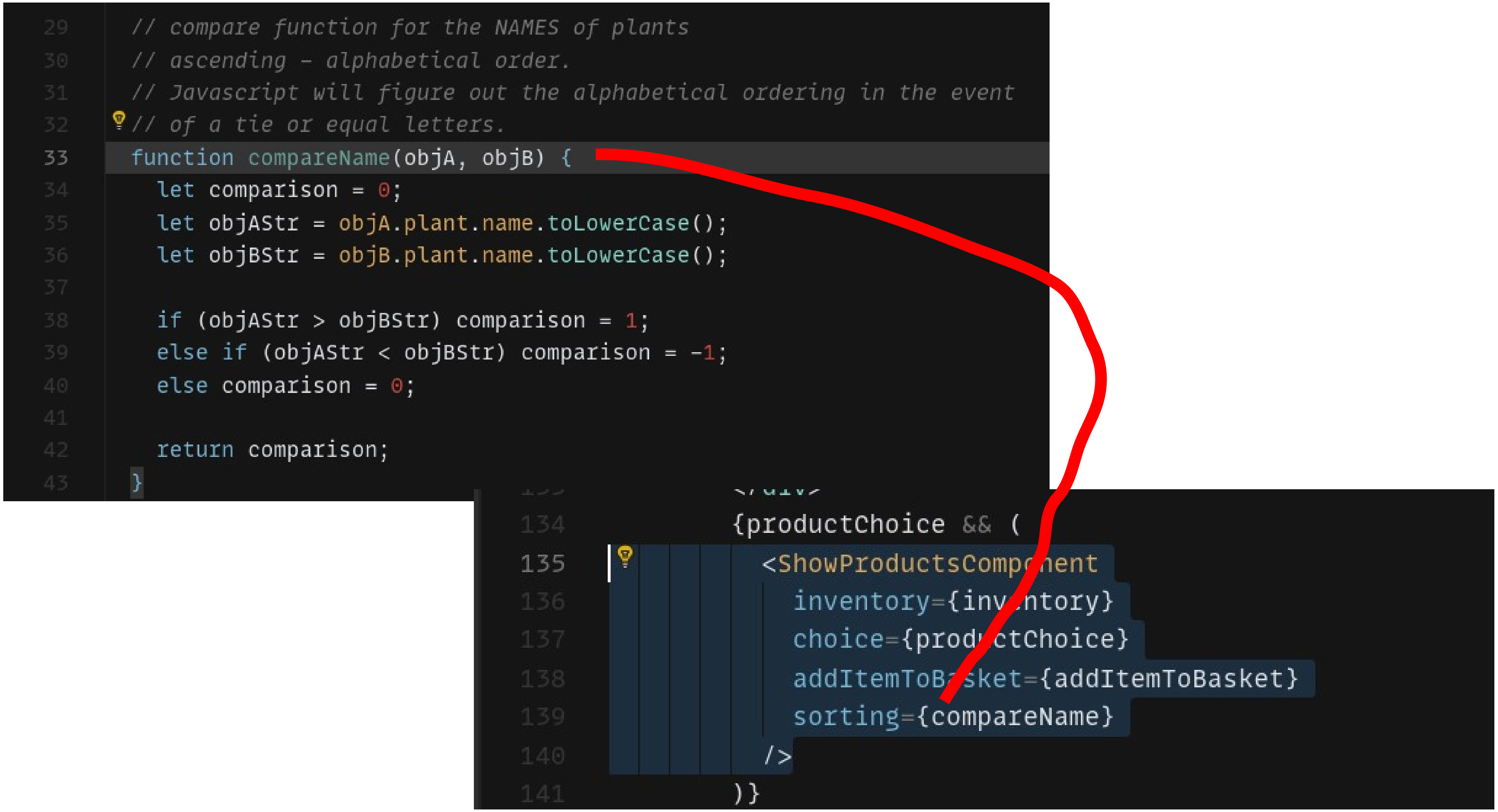
Total cost: €48.5

Savoy Cabbage seeds,€3.50 Remove

Dahlia Bulbs (10),€10.00 Remove

Brambley Apple Tree,€35.00 Remove

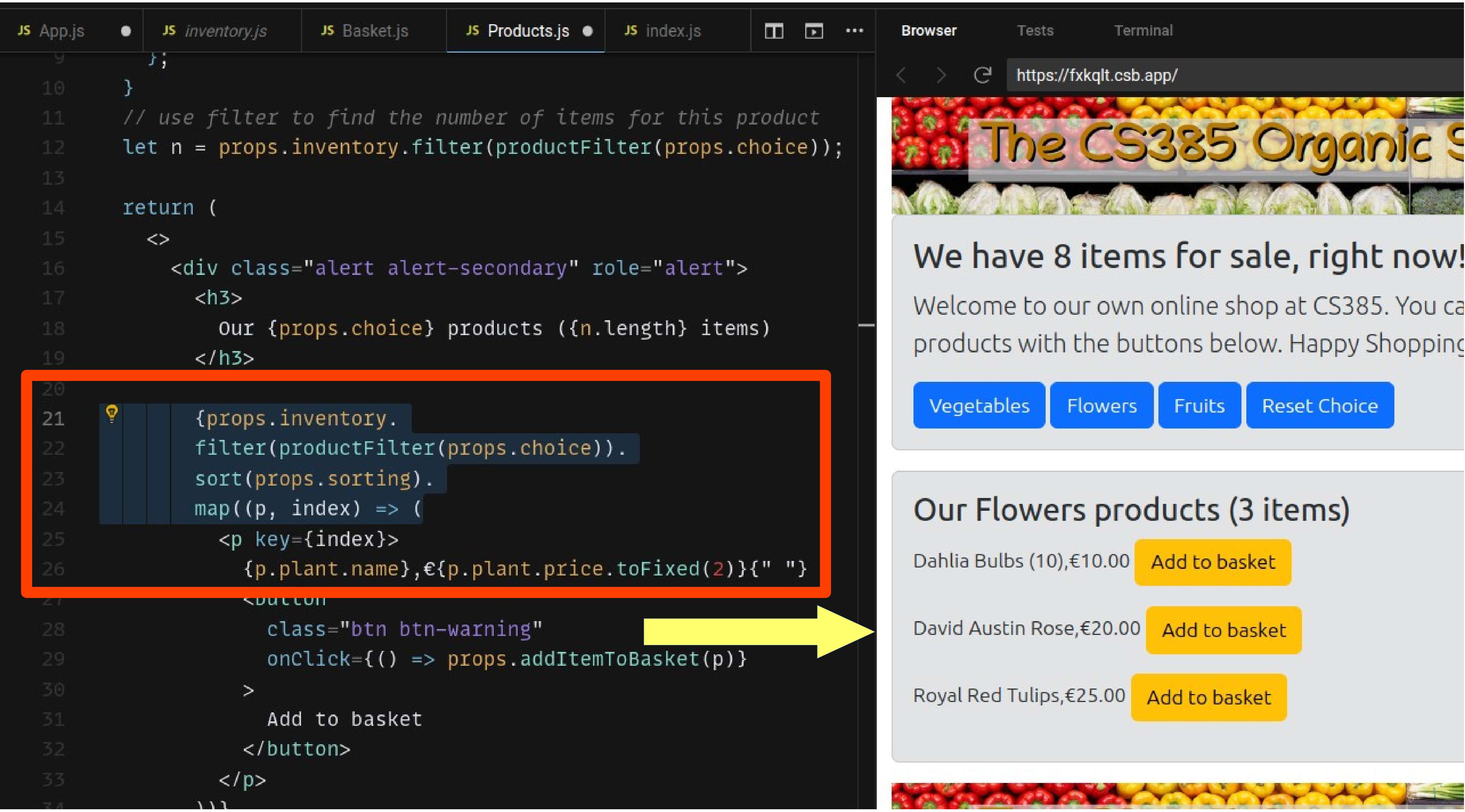
In ShowProductsComponent – let's sort in alphabetical order



```
// compare function for the NAMES of plants
// ascending - alphabetical order.
// Javascript will figure out the alphabetical ordering in the event
// of a tie or equal letters.
function compareName(objA, objB) {
  let comparison = 0;
  let objAStr = objA.plant.name.toLowerCase();
  let objBStr = objB.plant.name.toLowerCase();

  if (objAStr > objBStr) comparison = 1;
  else if (objAStr < objBStr) comparison = -1;
  else comparison = 0;

  return comparison;
}
```

```
          </div>
          {productChoice && (
            <ShowProductsComponent
              inventory={inventory}
              choice={productChoice}
              addItemToBasket={addItemToBasket}
              sorting={compareName}
            />
          )}
```

ShowProductsComponent – we compose filter, sort, & map



```
    };
  }
  // use filter to find the number of items for this product
  let n = props.inventory.filter(productFilter(props.choice));

  return (
    <>
      <div class="alert alert-secondary" role="alert">
        <h3>
          Our {props.choice} products ({n.length} items)
        </h3>

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

We have 8 items for sale, right now!

Welcome to our own online shop at CS385. You ca
products with the buttons below. Happy Shoppin

Vegetables | Flowers | Fruits | Reset Choice

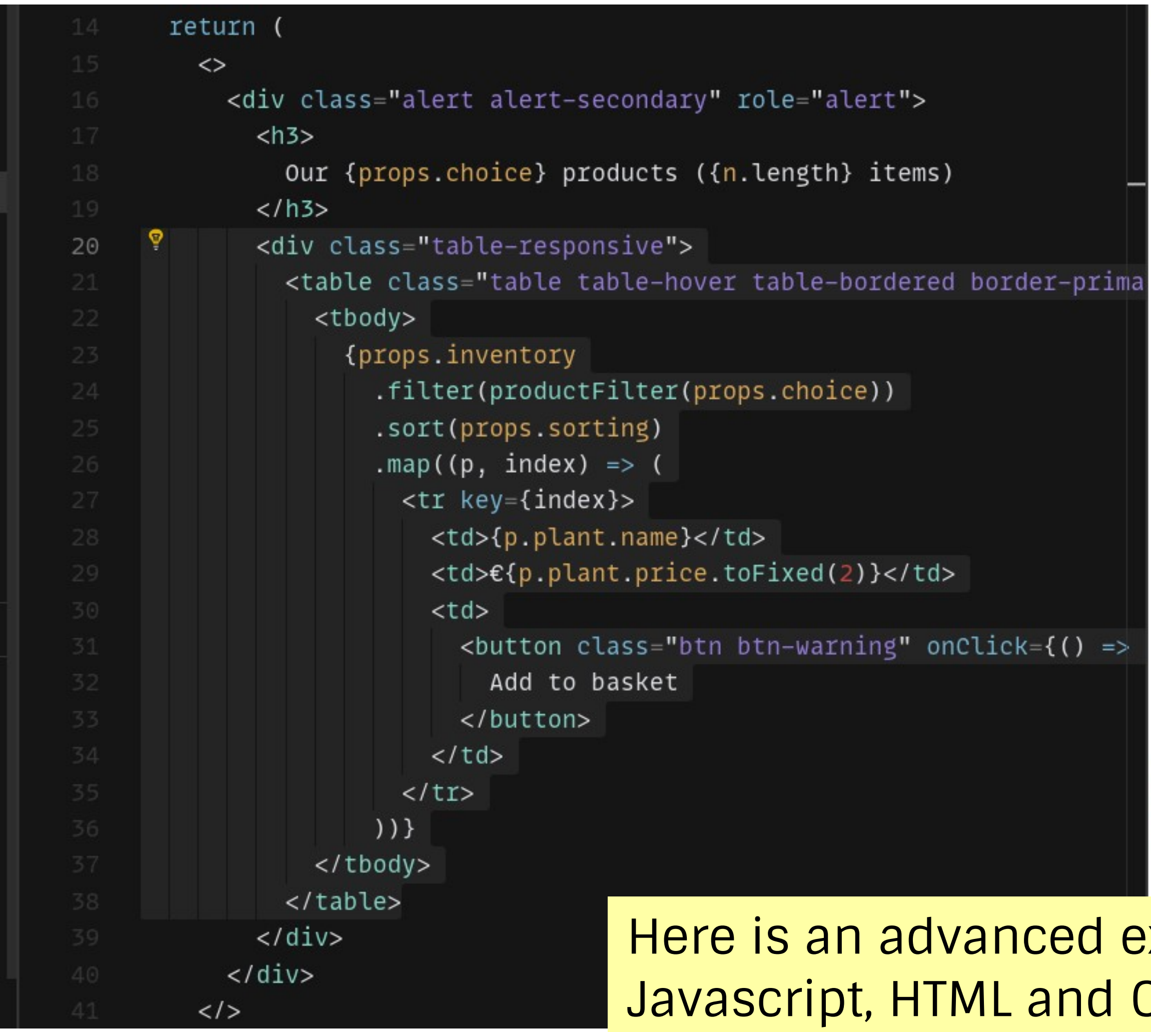
Our Flowers products (3 items)

Dahlia Bulbs (10),€10.00 Add to basket

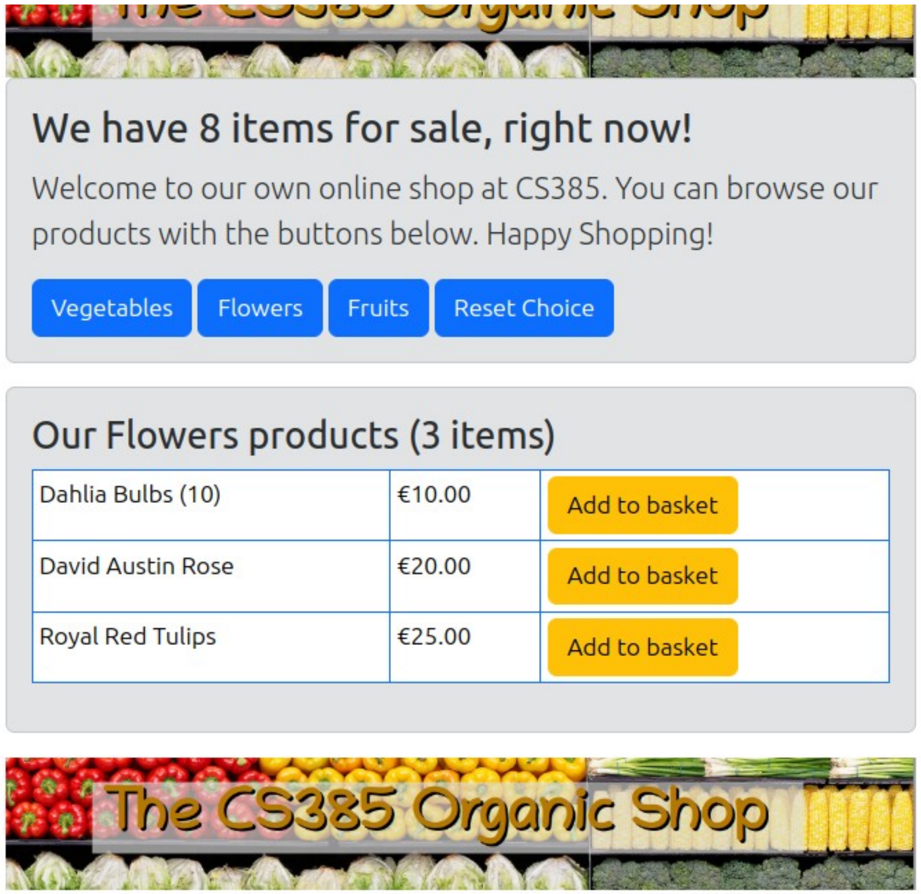
David Austin Rose,€20.00 Add to basket

Royal Red Tulips,€25.00 Add to basket

ShowProductsComponent – using a table to display products/items



```
return (
  <>
    <div class="alert alert-secondary" role="alert">
      <h3>
        Our {props.choice} products ({n.length} items)
      </h3>
      <div class="table-responsive">
        <table class="table table-hover table-bordered border-prima
          <tbody>
            {props.inventory
              .filter(productFilter(props.choice))
              .sort(props.sorting)
              .map((p, index) => (
                <tr key={index}>
                  <td>{p.plant.name}</td>
                  <td>€{p.plant.price.toFixed(2)}</td>
                  <td>
                    <button class="btn btn-warning" onClick={() =>
                      Add to basket
                    </button>
                  </td>
                </tr>
              ))}
          </tbody>
        </table>
      </div>
    </div>
  </>
```



We have 8 items for sale, right now!

Welcome to our own online shop at CS385. You can browse our products with the buttons below. Happy Shopping!

Our Flowers products (3 items)

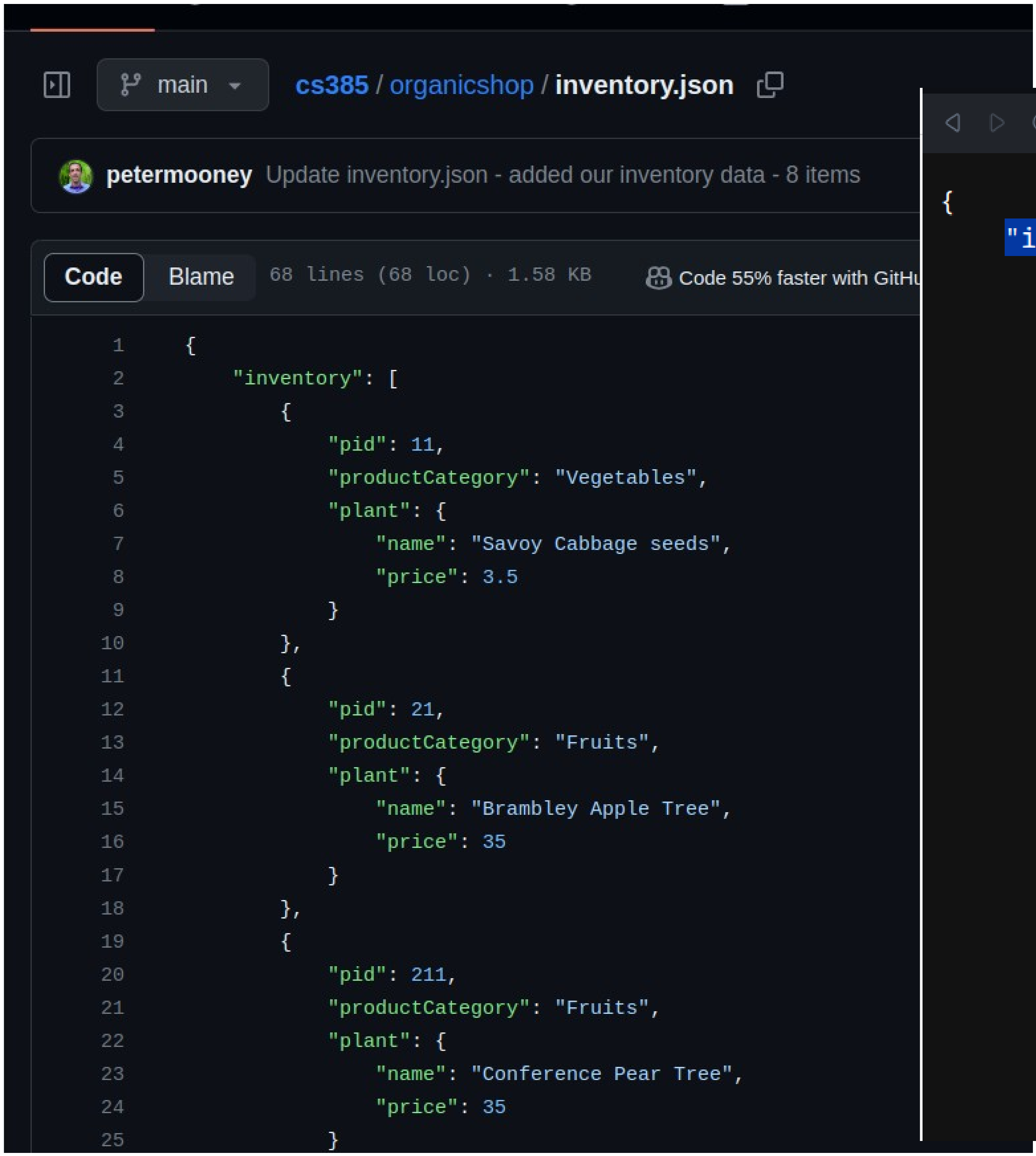
Dahlia Bulbs (10)	€10.00	Add to basket
David Austin Rose	€20.00	Add to basket
Royal Red Tulips	€25.00	Add to basket

Here is an advanced example of JSX – we have Javascript, HTML and CSS (from bootstrap) included in this code to render our table

Adding an API rather than using our local “inventory” JS file

- There are a few things to consider
  - You don’t want to have to re-write lots of code
  - Your data model should be the same (local file JSON vrs the API JSON)
  - The simple approach is to have your **useEffect** API code in the parent (top level) component
  - The API will have to store the data into a state variable (so there will be a change in how we store the data)

Our JSON from the GitHub API



main

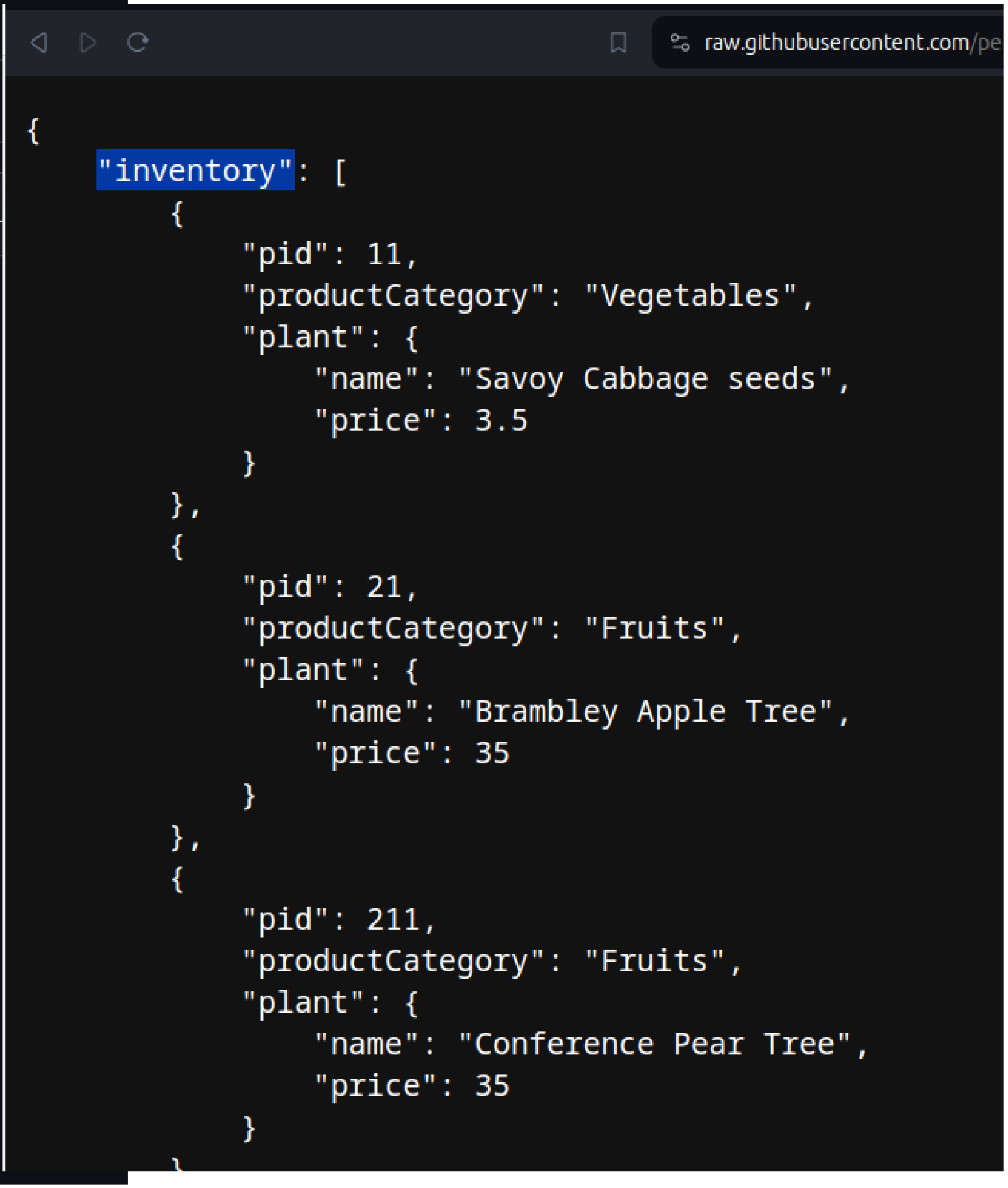
cs385 / organicshop / inventory.json

petermooney Update inventory.json - added our inventory data - 8 items

Code Blame 68 lines (68 loc) · 1.58 KB

Code 55% faster with GitHu

```
{
    "inventory": [
        {
            "pid": 11,
            "productCategory": "Vegetables",
            "plant": {
                "name": "Savoy Cabbage seeds",
                "price": 3.5
            }
        },
        {
            "pid": 21,
            "productCategory": "Fruits",
            "plant": {
                "name": "Brambley Apple Tree",
                "price": 35
            }
        },
        {
            "pid": 211,
            "productCategory": "Fruits",
            "plant": {
                "name": "Conference Pear Tree",
                "price": 35
            }
```

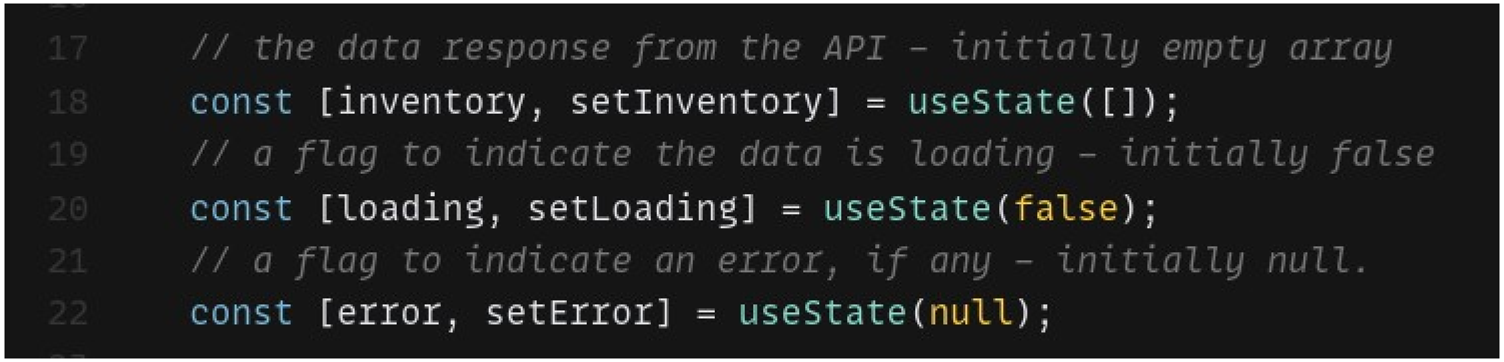


raw.githubusercontent.com/pe

```
{
    "inventory": [
        {
            "pid": 11,
            "productCategory": "Vegetables",
            "plant": {
                "name": "Savoy Cabbage seeds",
                "price": 3.5
            }
        },
        {
            "pid": 21,
            "productCategory": "Fruits",
            "plant": {
                "name": "Brambley Apple Tree",
                "price": 35
            }
        },
        {
            "pid": 211,
            "productCategory": "Fruits",
            "plant": {
                "name": "Conference Pear Tree",
                "price": 35
            }
```

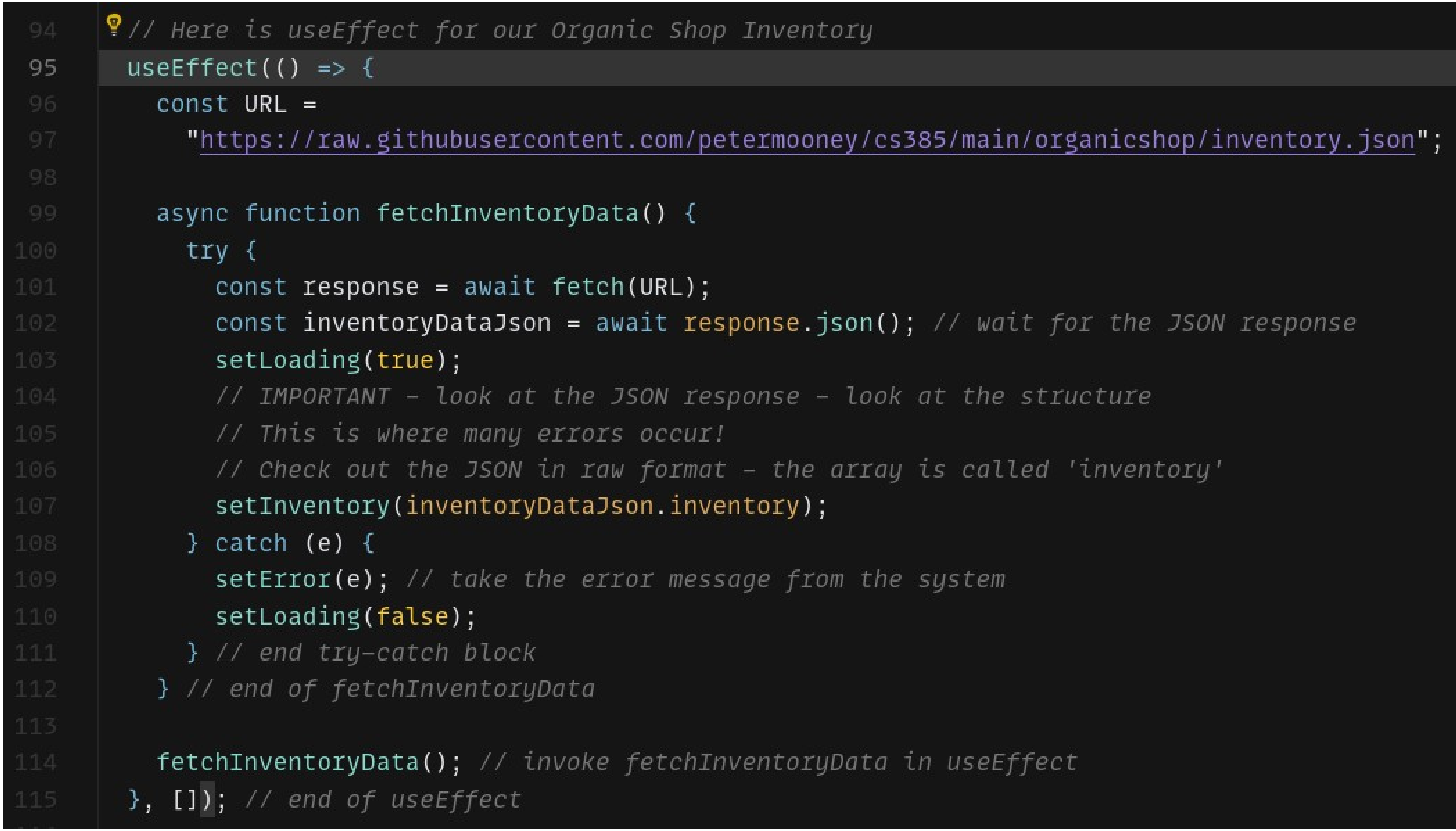
Step 1 – create the state variables for the API

- We previously used '**inventory**' as the name of the input data array – so let's keep that. This will keep code changes to a minimum



```
// the data response from the API - initially empty array
const [inventory, setInventory] = useState([]);
// a flag to indicate the data is loading - initially false
const [loading, setLoading] = useState(false);
// a flag to indicate an error, if any - initially null.
const [error, setError] = useState(null);
```

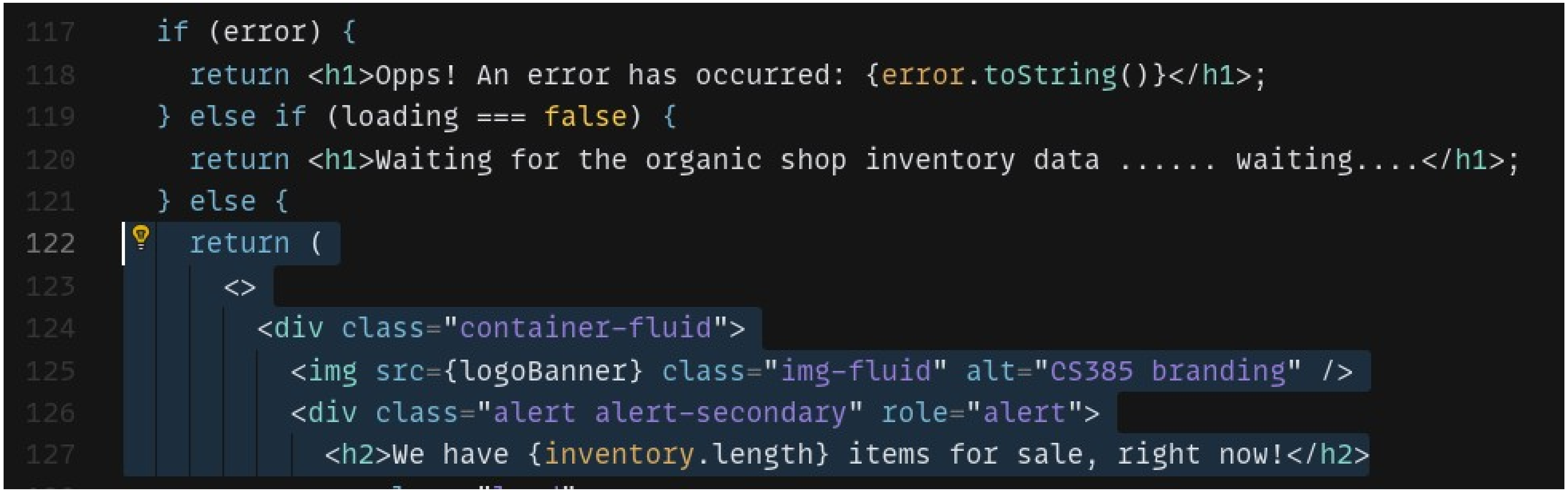
Step 2 – include the useEffect code from previous examples



```
// Here is useEffect for our Organic Shop Inventory
useEffect(() => {
  const URL =
    "https://raw.githubusercontent.com/petermooney/cs385/main/organicshop/inventory.json";

  async function fetchInventoryData() {
    try {
      const response = await fetch(URL);
      const inventoryDataJson = await response.json(); // wait for the JSON response
      setLoading(true);
      // IMPORTANT - look at the JSON response - look at the structure
      // This is where many errors occur!
      // Check out the JSON in raw format - the array is called 'inventory'
      setInventory(inventoryDataJson.inventory);
    } catch (e) {
      setError(e); // take the error message from the system
      setLoading(false);
    } // end try-catch block
  } // end of fetchInventoryData

  fetchInventoryData(); // invoke fetchInventoryData in useEffect
}, []); // end of useEffect
```

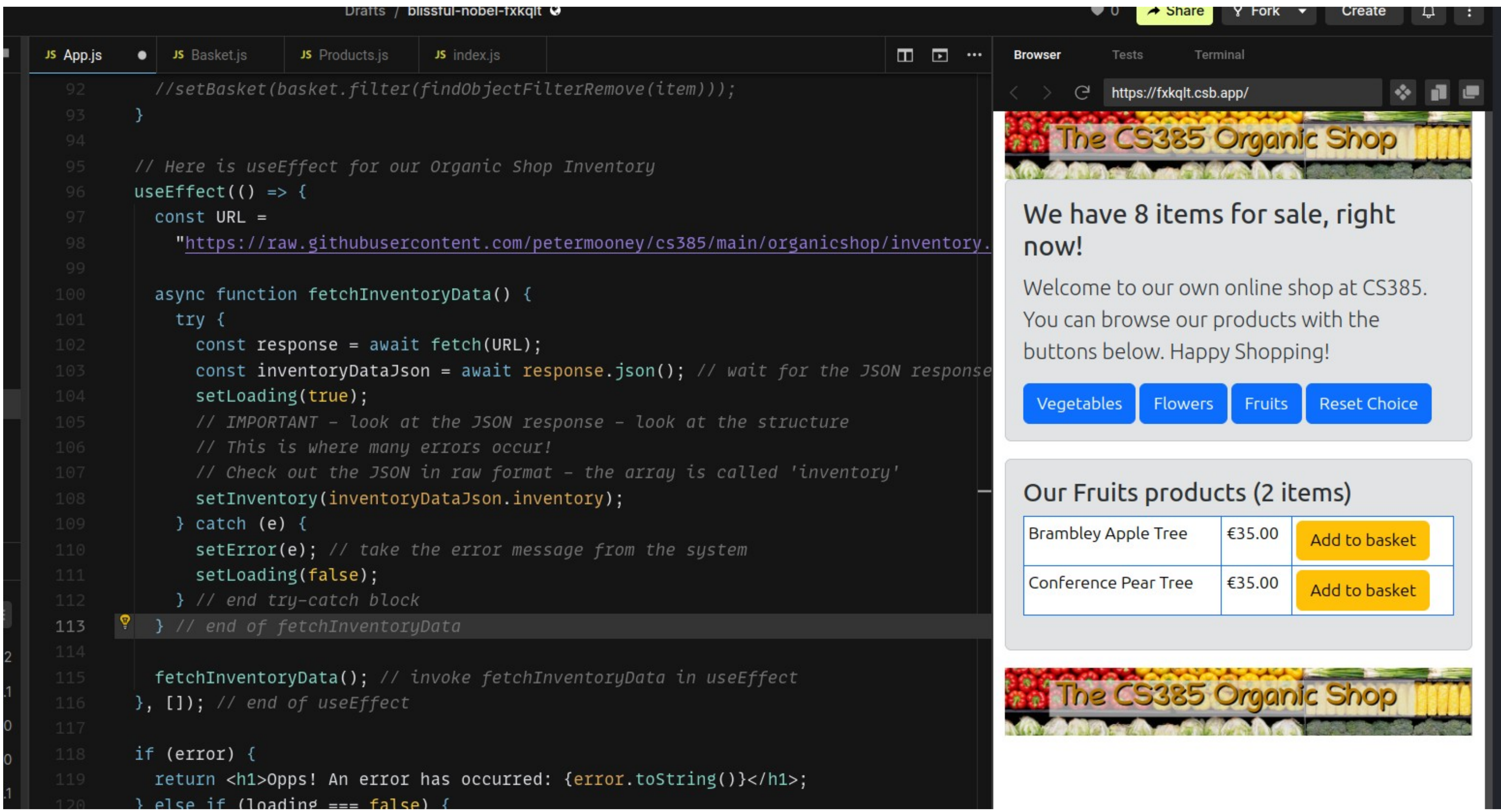
Step 3 – modify the return statement for the App component

- Notice we have added the **if statement**
- You might like to have a properly designed error message/component



```
  if (error) {
    return <h1>Opps! An error has occurred: {error.toString()}</h1>;
  } else if (loading === false) {
    return <h1>Waiting for the organic shop inventory data ...... waiting....</h1>;
  } else {
    return (
      <>
        <div class="container-fluid">
          <img src={logoBanner} class="img-fluid" alt="CS385 branding" />
          <div class="alert alert-secondary" role="alert">
            <h2>We have {inventory.length} items for sale, right now!</h2>
```

Step 4 – use the API



```
      //setBasket(basket.filter(findObjectFilterRemove(item)));
    }

    // Here is useEffect for our Organic Shop Inventory
    useEffect(() => {
      const URL =
        "https://raw.githubusercontent.com/petermooney/cs385/main/organicshop/inventory.

      async function fetchInventoryData() {
        try {
          const response = await fetch(URL);
          const inventoryDataJson = await response.json(); // wait for the JSON response
          setLoading(true);
          // IMPORTANT - look at the JSON response - look at the structure
          // This is where many errors occur!
          // Check out the JSON in raw format - the array is called 'inventory'
          setInventory(inventoryDataJson.inventory);
        } catch (e) {
          setError(e); // take the error message from the system
          setLoading(false);
        } // end try-catch block
      } // end of fetchInventoryData

      fetchInventoryData(); // invoke fetchInventoryData in useEffect
    }, []); // end of useEffect

    if (error) {
      return <h1>Opps! An error has occurred: {error.toString()}</h1>;
    } else if (loading === false) {
```

We have 8 items for sale, right now!

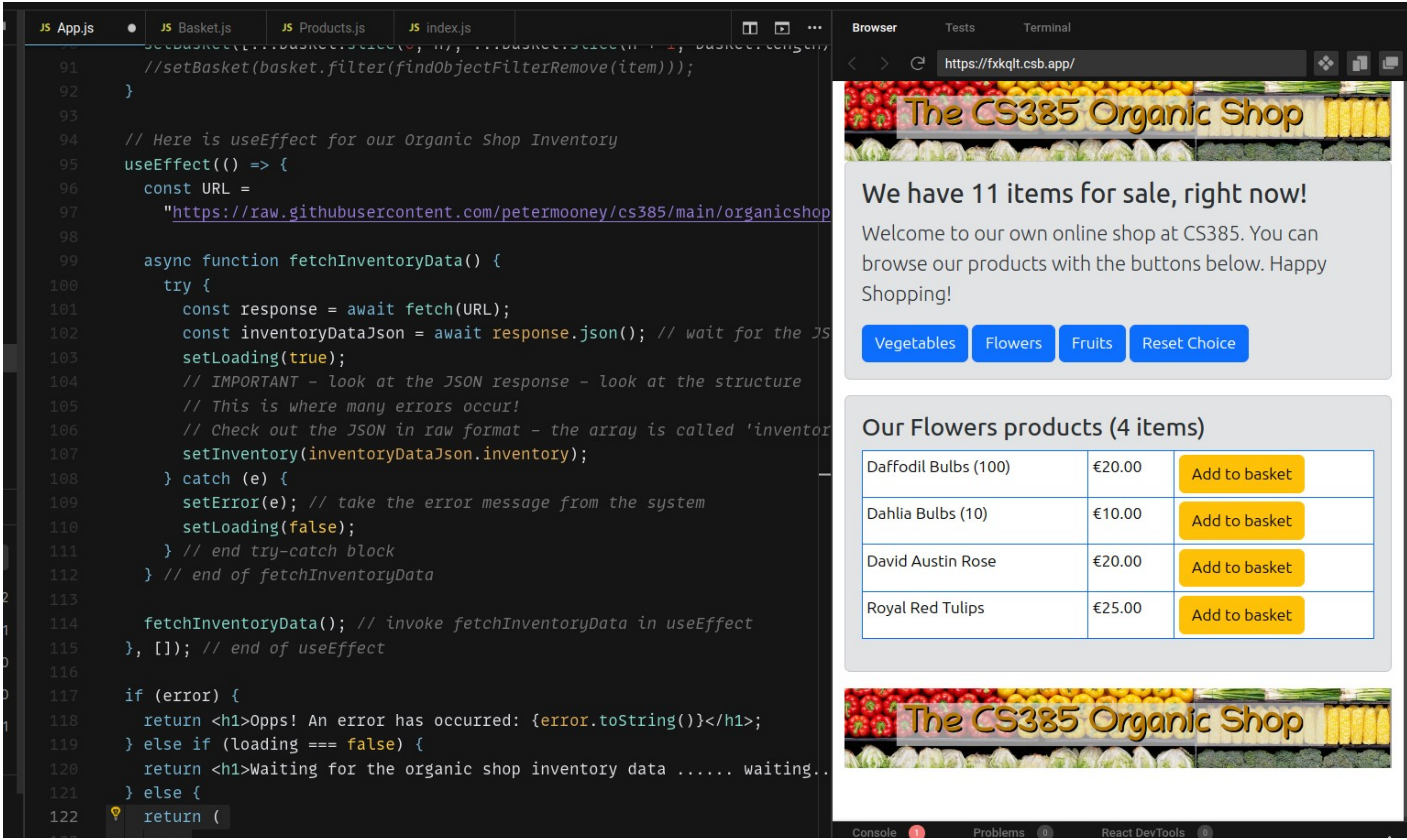
Welcome to our own online shop at CS385. You can browse our products with the buttons below. Happy Shopping!

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

Step 4 – use the API (more items)



```
      //setBasket(basket.filter(findObjectFilterRemove(item)));
    }

    // Here is useEffect for our Organic Shop Inventory
    useEffect(() => {
      const URL =
        "https://raw.githubusercontent.com/petermooney/cs385/main/organicshop

      async function fetchInventoryData() {
        try {
          const response = await fetch(URL);
          const inventoryDataJson = await response.json(); // wait for the JS
          setLoading(true);
          // IMPORTANT - look at the JSON response - look at the structure
          // This is where many errors occur!
          // Check out the JSON in raw format - the array is called 'inventor
          setInventory(inventoryDataJson.inventory);
        } catch (e) {
          setError(e); // take the error message from the system
          setLoading(false);
        } // end try-catch block
      } // end of fetchInventoryData

      fetchInventoryData(); // invoke fetchInventoryData in useEffect
    }, []); // end of useEffect

    if (error) {
      return <h1>Opps! An error has occurred: {error.toString()}</h1>;
    } else if (loading === false) {
      return <h1>Waiting for the organic shop inventory data ...... waiting..
    } else {
      return (
```

We have 11 items for sale, right now!

Welcome to our own online shop at CS385. You can browse our products with the buttons below. Happy Shopping!

Our Flowers products (4 items)

Daffodil Bulbs (100)	€20.00	Add to basket
Dahlia Bulbs (10)	€10.00	Add to basket
David Austin Rose	€20.00	Add to basket
Royal Red Tulips	€25.00	Add to basket

Summary

- The API code was included but no other code in our application required any changes.
- The API works well – it only needs to be involved once when the application loads
- The GitHub page can simulate the addition, editing, or deletion of items from the inventory

CS385 Lecture 14 – PROJECT UPDATE

PROJECT UPDATE

A CS385 tradition ….. the annual choice of Team Names

This applies to groups as well as solo projects

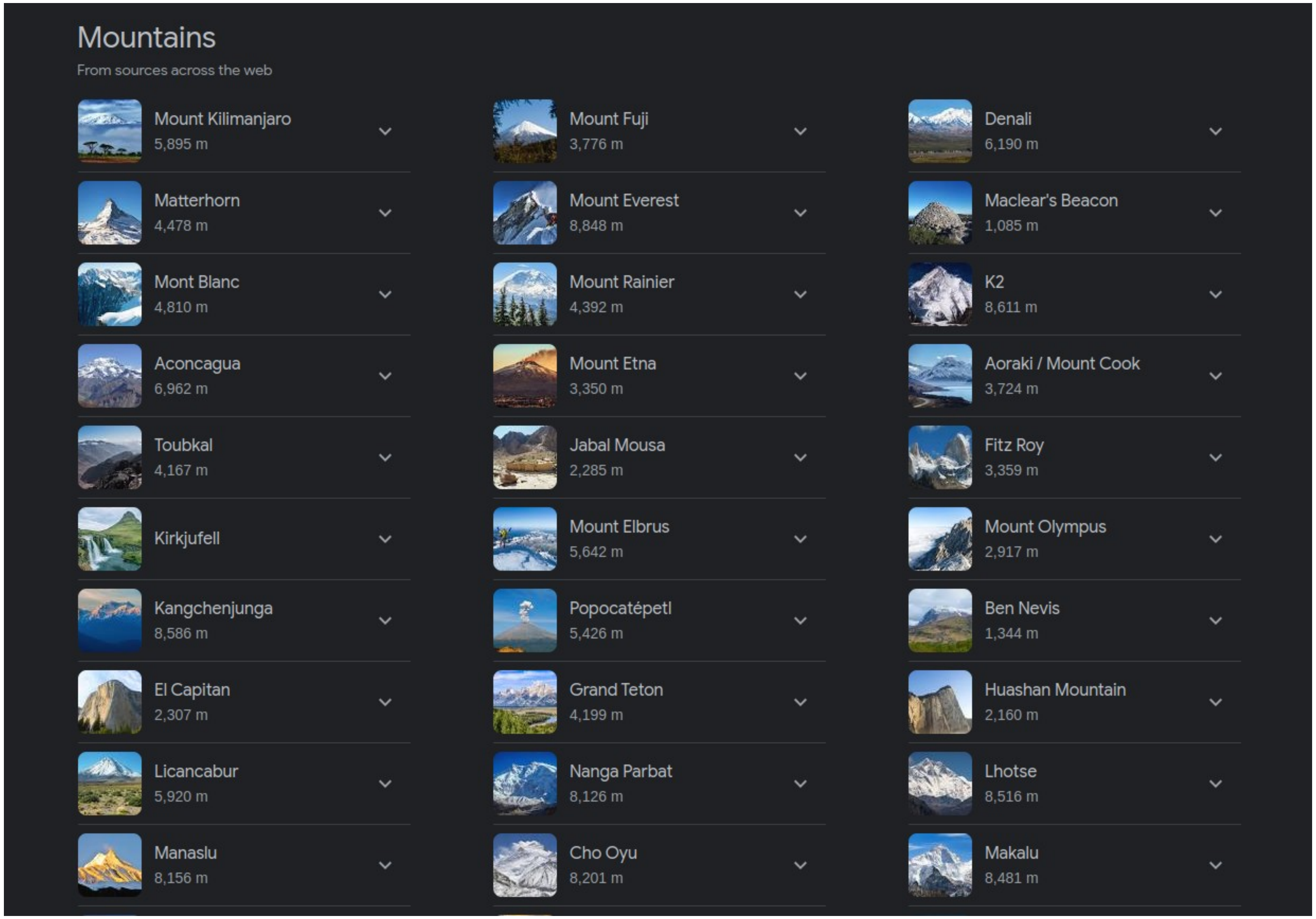
This really helps me and the demonstrator team when organising the project labs!

In the past, in CS385, we have had ….

- Teams named after **famous rivers of the world** (2021)
- Teams named after **famous types of trees** (2022)
- Teams named after **objects from our solar system and beyond** (2018)
- Teams named after **famous cities** (2019)
- Teams named after **elements of the periodic table** (2020)

This year's team name selection will be taken from .....

- Famous mountains of the world (and Ireland)



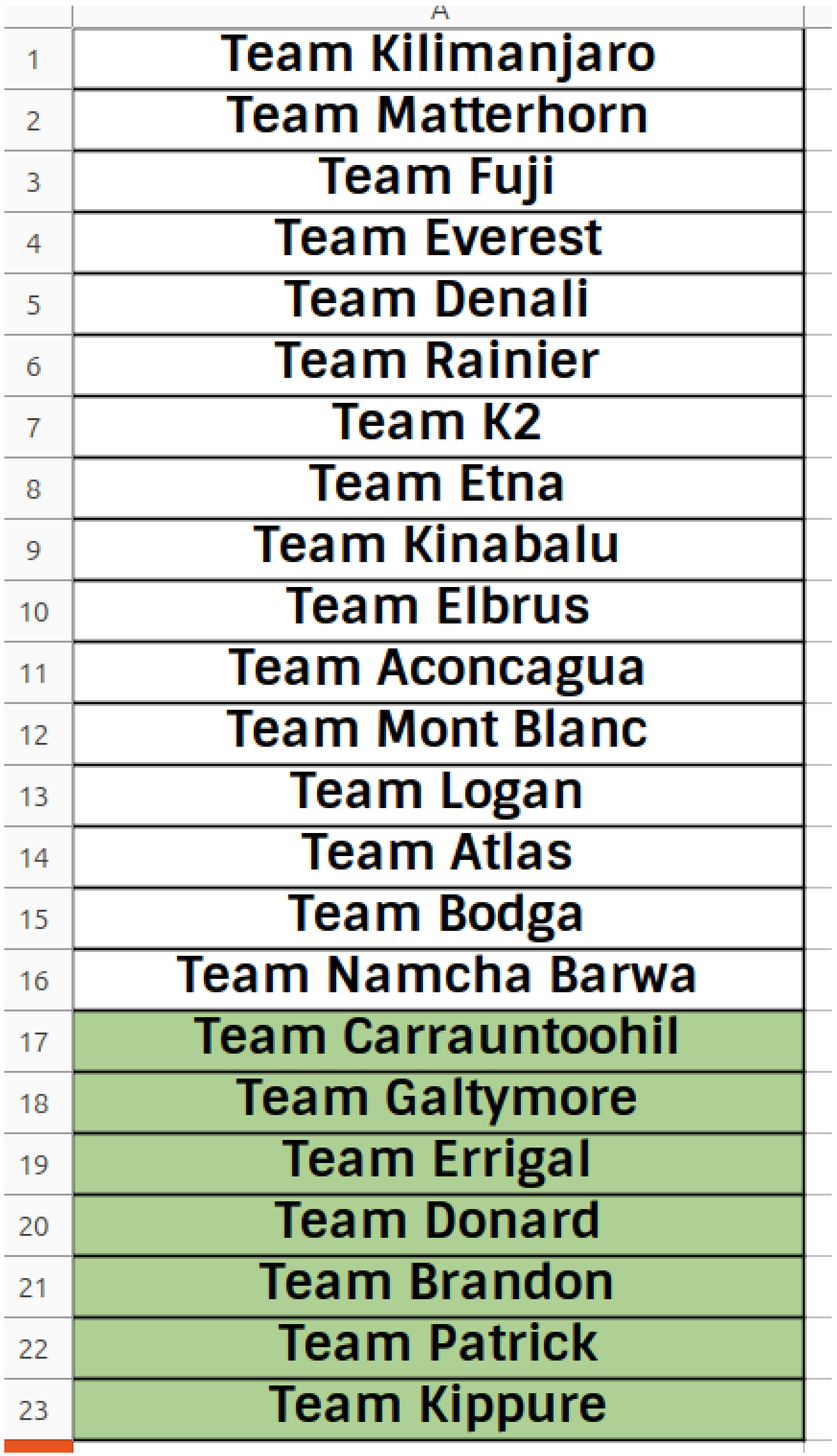
Mountains

From sources across the web

Mount Kilimanjaro	5,895 m	Mount Fuji	3,776 m	Denali	6,190 m
Matterhorn	4,478 m	Mount Everest	8,848 m	Maclear's Beacon	1,085 m
Mont Blanc	4,810 m	Mount Rainier	4,392 m	K2	8,611 m
Aconcagua	6,962 m	Mount Etna	3,350 m	Aoraki / Mount Cook	3,724 m
Toubkal	4,167 m	Jabal Mousa	2,285 m	Fitz Roy	3,359 m
Kirkjufell		Mount Elbrus	5,642 m	Mount Olympus	2,917 m
Kangchenjunga	8,586 m	Popocatépetl	5,426 m	Ben Nevis	1,344 m
El Capitan	2,307 m	Grand Teton	4,199 m	Huashan Mountain	2,160 m
Licancabur	5,920 m	Nanga Parbat	8,126 m	Lhotse	8,516 m
Manaslu	8,156 m	Cho Oyu	8,201 m	Makalu	8,481 m

The List for 2023

- Choose ONE of these – OR if you can make a convincing case for another mountain (must be verifiable)
- **You need to tell the demonstrators your chosen mountain**



	A
1	Team Kilimanjaro
2	Team Matterhorn
3	Team Fuji
4	Team Everest
5	Team Denali
6	Team Rainier
7	Team K2
8	Team Etna
9	Team Kinabalu
10	Team Elbrus
11	Team Aconcagua
12	Team Mont Blanc
13	Team Logan
14	Team Atlas
15	Team Bodga
16	Team Namcha Barwa
17	Team Carrauntoohil
18	Team Galtymore
19	Team Errigal
20	Team Donard
21	Team Brandon
22	Team Patrick
23	Team Kippure

Every team will be assigned a demonstrator as PROJECT MANAGER

- Your team MUST check in with the PROJECT MANAGER every week (this is taken as an attendance at a project meeting!)
- **Friday 17th November** – choose your team name AND describe your the planning for your application.
- PROJECT MANAGER list available on Friday morning