CS385 Lecture 17

More useEffect, working with object changes, summarising object arrays, the Date Picker

JSON.stringify

- The **JSON.stringify()** static method converts a JavaScript value to a JSON string
- **Arrays are serialized as arrays** (enclosed by square brackets). Only array indices between 0 and length - 1 (inclusive) are serialized; other properties are ignored.
- **Can be very useful for debugging** – replaces the need to write a map function to render the current value of a variable (or an array)

```
4   function App() {
5     const basket = [
6       { pid: 10, name: 155 },
7       { pid: 12, name: 17 },
8       { pid: 10, name: 14 },
9       { pid: 10, name: 14 },
10    ];
11
12    return <>{JSON.stringify(basket)}</>;
13  }
14
```

localhost:3000

[{"pid":10,"name":155},{"pid":12,"name":17},{"pid":10,"name":14},{"pid":10,"name":14}]

- That's all – helpful way to check the contents of arrays as they move between components (for example)

Working with `useEffect` when state variables are involved

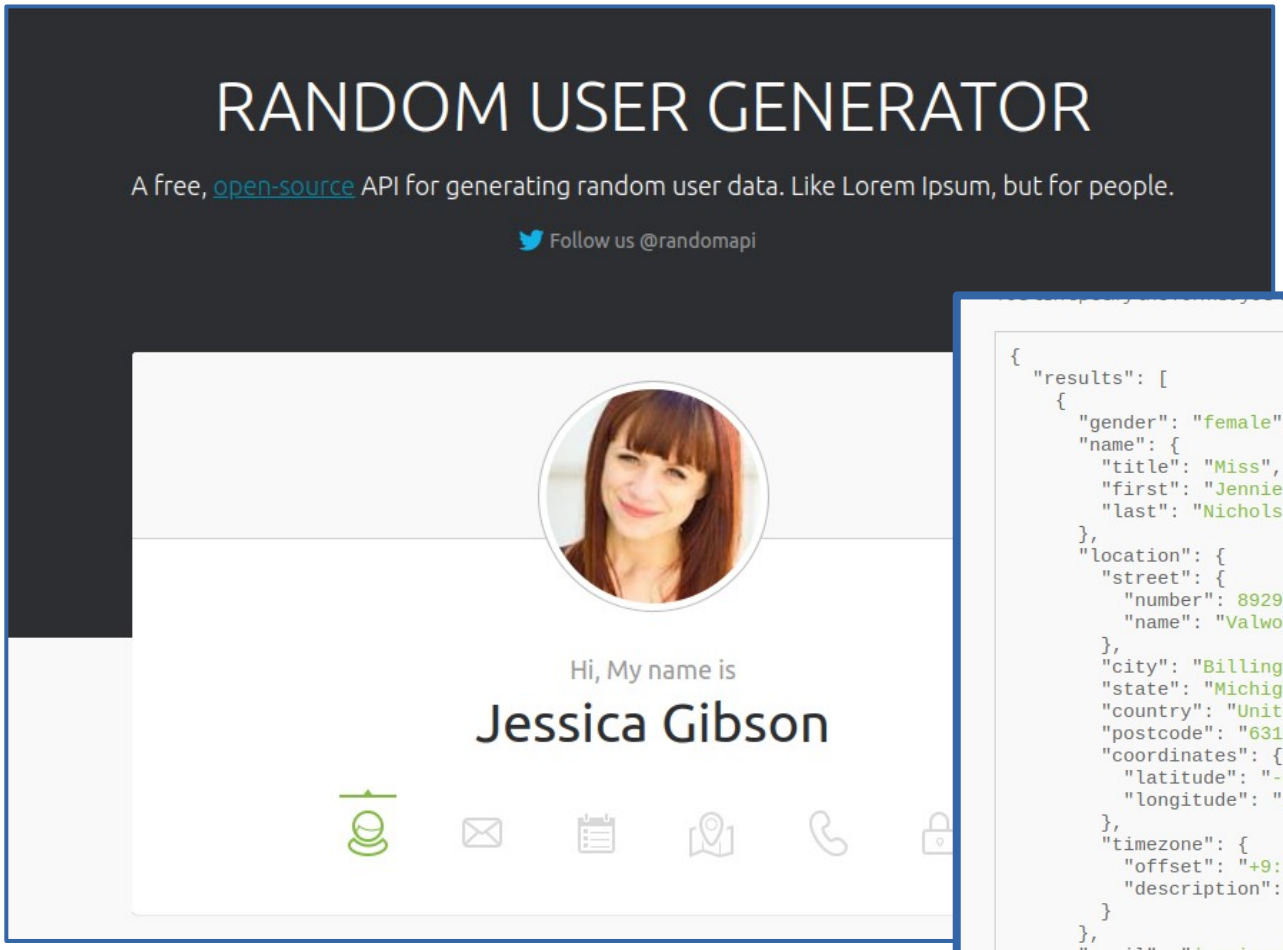
- Line 30 – the array is an empty array []

```
11  useEffect(() => {
12    // Change the URL to your chosen JSON file as per Lab 4
13    // we can choose ANY of the JSON files in the trains folder.
14    const URL =
15      "https://raw.githubusercontent.com/petermooney/cs385/main/trains/trainData25.json";
16
17    async function fetchTrainData() {
18      try {
19        const response = await fetch(URL);
20        const trainDataJson = await response.json(); // wait for the JSON response
21        setLoading(true);
22        setData(trainDataJson.trainService);
23      } catch (e) {
24        setError(e); // take the error message from the system
25        setLoading(false);
26      } // end try-catch block
27    } // end of fetchData
28
29    fetchTrainData(); // invoke fetchTrainData in useEffect
30  }, []); // end of useEffect
```

You can pass state variables to useEffect within this array

- This means, that as state variables change value, useEffect will be executed every time that these state variables change.
- This is very useful in useEffect where you have **API URLs** which need to be created using data collected from the user.
- Let's look at an example where we use the array to pass dependent state variables to the useEffect function.

We'll use the Random User API




```
{
  "results": [
    {
      "gender": "female",
      "name": {
        "title": "Miss",
        "first": "Jennie",
        "last": "Nichols"
      },
      "location": {
        "street": {
          "number": 8929,
          "name": "Valwood Pkwy",
        },
        "city": "Billings",
        "state": "Michigan",
        "country": "United States",
        "postcode": "63104",
        "coordinates": {
          "latitude": "-69.8246",
          "longitude": "134.8719"
        },
        "timezone": {
          "offset": "+9:30",
          "description": "Adelaide, Darwin"
        }
      },
      "email": "jennie.nichols@example.com",
      "login": {
        "uuid": "7a0eed16-9430-4d68-901f-c0d4c1c3bf00",
        "username": "yellowpeacock117",
        "password": "addison",
        "salt": "sld1yGtd",
        "md5": "ab54ac4c0be9480ae8fa5e9e2a5196a3",
        "sha1": "edcf2ce613cbdea349133c52dc2f3b83168dc51b",
        "sha256": "48df5229235ada28389b91e60a935e4f9b73eb4bdb855ef9258a1751f10bdc5d"
      },
      "dob": {
        "date": "1992-03-08T15:13:16.688Z",
        "age": 30
```

Setting up our code (like before)

- API URL Structure
- https://randomuser.me/api/?gender=female&results=100

```
function App() {
  // the data response from the API - initially empty array
  const [data, setData] = 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);
  // allow user to specify gender of random users
  const [gender, setGender] = useState("female");
  // allow user specify the number of results.
  const [choice, setChoice] = useState(7);
  // number of user options for drop down list
  const options = [5, 10, 15, 20, 25, 30];
```

```
<form>
  Pick number of users
  <select onChange={handleListChange}>
    {options.map((s, key) => (
      <option key={key} value={s}>
        <strong> {s}</strong>
      </option>
    ))}
  </select>
</form>

<form>
  Pick gender
  <select onChange={handleGenderListChange}>
    <option key="1" value="female">Female</option>
    <option key="2" value="male">Male</option>
  </select>
</form>
```

Our useEffect code – with the ability to change gender and results variables

```
24  useEffect(() => {
25    // create the URL for specific gender and number of results.
26    const URL =
27      "https://randomuser.me/api/?gender=" + gender + "&results=" + choice;
28
29    async function fetchStrData() {
30      try {
31        const response = await fetch(URL);
32        const strDataJson = await response.json(); // wait for the JSON response
33        setLoading(true);
34        setData(strDataJson.results);
35      } catch (e) {
36        setError(e); // take the error message from the system
37        setLoading(false);
38      } // end try-catch block
39    } // end of fetchData
40
41    fetchStrData(); // invoke fetchStrData in useEffect
42  }, [choice, gender]); // end of useEffect with two state variables
```

```
// allow user to specify gender of random users
const [gender, setGender] = useState("female");
// allow user specify the number of results.
const [choice, setChoice] = useState(7);
```

- Each time the user changes the drop-down-list selection, there is a state update – this causes useEffect to be re-executed

Pick number of users 10
Pick gender Female

Random User Display: 10 Users

User (1): Ms Tina Tucker
9232,Hillcrest Rd
Victorville,Alabama,United States

User (2): Ms Leslie Anderson
6505,E North St
Queanbeyan,New South Wales,Australia

User (3): Ms Micaela Guerrero
8640,Continuación Yucatán
San Jerónimo Coyula,Chiapas,Mexico

User (4): Mrs Lena Sødal
4013,Armauer Hansens gate
Namnå,Troms - Romsa,Norway

User (5): Ms Sharon Frazier
2794,High Street
Nottingham,County Down,United Kingdom

User (6): Miss Alberte Christiansen
2457,Hjortevænget
Odense Sv,Hovedstaden,Denmark

User (7): Mrs Riley Fletcher
2183,Bollinger Rd
Hartford,Utah,United States

User (8): Ms Rushali Dawangave
5935,Maharanipeta
Durg,Tripura,India

Pick number of users 15
Pick gender Male

Random User Display: 15 Users

User (1): Mr Ali Kuday
1564,Vatan Cd
Iğdır,İzmir,Turkey

User (2): Mr Wilfrido Rocha
8591,Cerrada Sur Solano
Chilchota,Tlaxcala,Mexico

User (3): Mr Breunis Wallet
1450,Brandwegske
Oude Niedorp,Overijssel,Netherlands

User (4): Mr Rubén Cabán
4818,Calle Baja California Sur
Ixmiquilpan,Sonora,Mexico

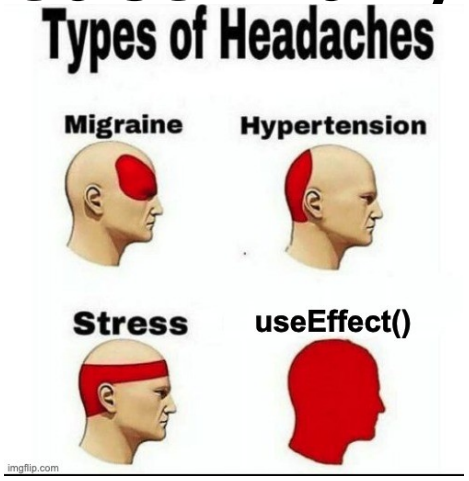
User (5): Mr Christian Ortega
8704,Calle Mota
Alcobendas,La Rioja,Spain

User (6): Mr Luis Monroy
7377,Continuación Franco
La Ventosa,Guerrero,Mexico

User (7): Mr Justin Dunn
4203,Kings Road
Nottingham,Merseyside,United Kingdom

Using useEffect with state variables

- Beaware that useEffect will be re-executed **every time a state variable** (passed using the dependency array) **is changed**.
- This needs careful design planning – you must be sure that your application needs “this effect”
- **If a state variable is changing very frequently this could cause a plethora of issues with your API service or API provider.**



Updating the properties of objects in React Javascript

Making changes to a single object in React Javascript

```
4   function App() {
5     let myObject = { name: "Peter", course: "CS385" };
6     // change an existing property
7     myObject.name = "Peter Mooney";
8     // add a new property to the object
9     myObject.semester = "1";
10    // delete a property from an object
11    delete myObject.course;
12    // create an new object property
13    myObject.module = "CS385";
14
15    return (
16      <>
17        <h1>
18          {myObject.name},{myObject.module}
19        </h1>
20        <h1>{JSON.stringify(myObject)}</h1>
21      </>
22    );
23  }
```

Peter Mooney,CS385

{"name":"Peter Mooney","semester":"1","module":"CS385"}

In a similar way – we can make changes to objects in an array

```
4   function App() {
5     let myObjects = [
6       { name: "Peter", course: "CS385" },
7       { name: "Jessica", course: "CS478" },
8     ];
9     // make changes to the individual objects
10    myObjects[0].name = "Peter Mooney";
11    myObjects[1].name = "Jessica Smith";
12
13    delete myObjects[0].course;
14    myObjects[0].module = "CS385";
15
16    myObjects[1].credits = "15";
17
18    return (
19      <>
20        <h1>{JSON.stringify(myObjects)}</h1>
21      </>
22    );
23  }
```

- Hardcoding changes
- Not very flexible
- **Does not scale well for large arrays** (or frequently changing objects/arrays)

[{"name":"Peter Mooney","module":"CS385"},{"name":"Jessica Smith","course":"CS478","credits":"15"}]

Updating a specific object with an object array (Part 1)

- Note how we use the spread operator on an individual object (Lab Exam 3 question)

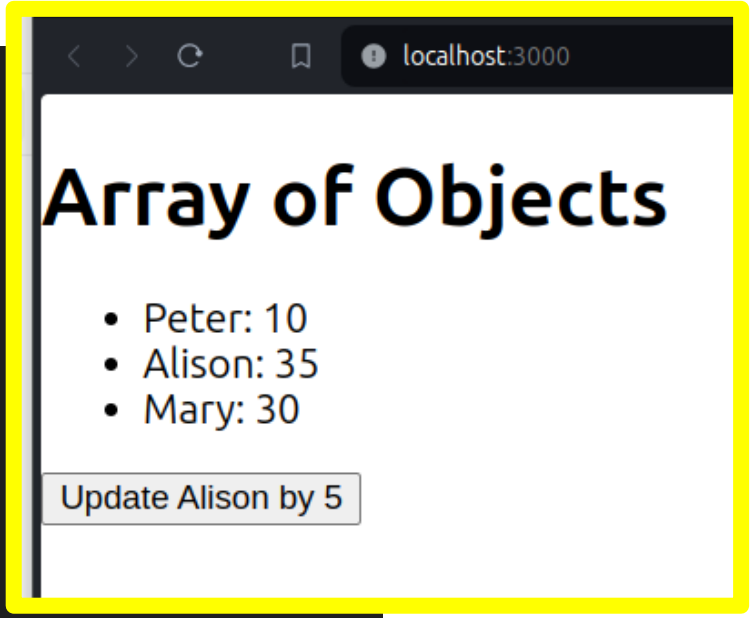
```
3   function App() {
4     // Sample state with an array of objects
5     const [data, setData] = useState([{ id: 1, name: "Peter", value: 10 },
6       { id: 2, name: "Alison", value: 20 },{ id: 3, name: "Mary", value: 30 }]);
7
8     function updateSpecificObjectProperty(id, newValue) {
9       // Use map to create a new array with the updated object
10      const updatedData = data.map((item) => {
11        if (item.id === id) {
12          // If it's the object we want to update, create a new object with the updated value
13          // The spread operator also works for individual object properties.
14          return { ...item, value: item.value + newValue };
15        }
16        // If it's not the object we want to update, keep it as it is
17        return item;
18      });
19      // Update the state with the new array
20      setData(updatedData);
21    }
```

Here we ensure that we do not mutate state by using the useState hook

Updating a specific object with an object array (Part 2)

- Here, a specific object is hardcoded into the button click

```
26  return (
27    <>
28      <h1>Array of Objects</h1>
29      <ul>
30        {data.map((item, index) => (
31          <li key={index}>
32            {item.name}: {item.value}
33          </li>
34        ))}
35      </ul>
36
37      <button onClick={() => updateSpecificObjectProperty(2, 5)}>
38        Update Alison by 5
39      </button>
40    </>
41  );
42 }
```

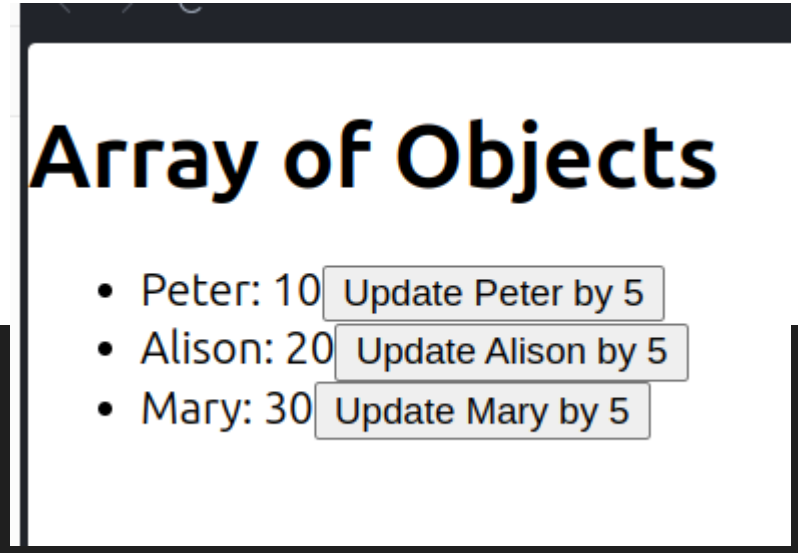


Updating object properties in an array (example 2)

- What we really need is an example where we do not code in the reference or index of a specific object (unless you think it is a good idea)
- We would like a more flexible approach – we can again use a map function for this.
- In this example, **we allow every object presented to the user to be updated for a specific property and with a specific amount or value**.
- This type of functionality is very common!

Updating object properties in an array (example 2) - code

- This approach simulates what we see in things like shopping carts online




```
29        <h1>Array of Objects</h1>
30        <ul>
31          {data.map((item, index) => (
32            <li key={index}>
33              {item.name}: {item.value}
34              <button onClick={() => updateSpecificObjectProperty(item.id, 5)}>
35                Update {item.name} by 5
36              </button>
37            </li>
38          ))}
39        </ul>
40      </>
41    );
```

```
// Sample state with an array of objects
const [data, setData] = useState([
  { id: 1, name: "Peter", value: 10 },
  { id: 2, name: "Alison", value: 20 },
  { id: 3, name: "Mary", value: 30 },
]);
```

Summarising an array of objects

- Very often when we create arrays of objects (such as a shopping cart) we can have many duplicated objects.
- It is a nicer user-experience if we can summarise the contents of the ‘basket’ or ‘cart’. For example – object A (3 times), object B (5 times), object D (2 times), object K (1 time)
- We need to write an algorithm to summarise our array of objects like this.

Summarising an array of objects (example)

- There are 5 objects, but there are actually only two unique objects
- **`{pid: 10, name: 14}`** appears three times
- **`{pid: 12, name: 17}`** appears twice.

```
5   const [basket, setBasket] = useState([
6     { pid: 10, name: 14 },
7     { pid: 12, name: 17 },
8     { pid: 10, name: 14 },
9     { pid: 10, name: 14 },
10    { pid: 12, name: 17 },
11  ]);
```

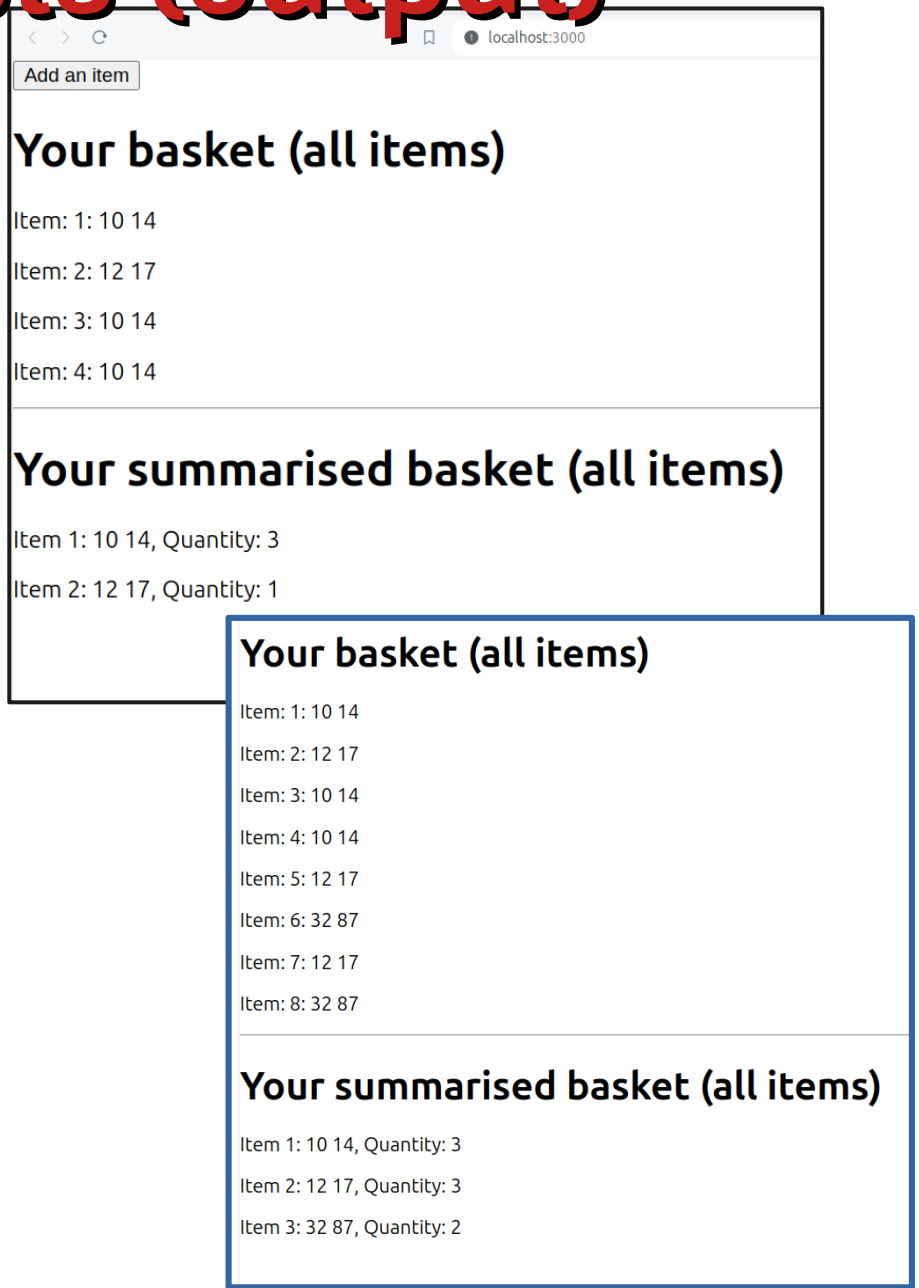
Our algorithm to summarise the array of objects

- If there is only one object, create qty property. Let qty = 1.
- If there is at least two objects.
  - Create a summary object from 1st object. Let qty = 1
  - Iterate through the array of objects. Use your own 'search' or 'find' to identify the objects that are duplicate.
  - Increment the qty in each case until the end of the array
  - Repeat for all other objects

```
5   const [basket, setBasket] = useState([
6     { pid: 10, name: 14 },
7     { pid: 12, name: 17 },
8     { pid: 10, name: 14 },
9     { pid: 10, name: 14 },
10    { pid: 12, name: 17 },
11  ]);
```

Our algorithm to summarise the array of objects (output)

- If there is only one object, create qty property. Let qty = 1.
- If there is at least two objects.
  - Create a summary object from 1st object. Let qty = 1
  - Iterate through the array of objects. Use your own 'search' or 'find' to identify the objects that are duplicate.
  - Increment the qty in each case until the end of the array
  - Repeat for all other objects



Step 1 – decide how you want “to find” duplicate objects

```
function SummariseBasket(props) {

  // This is our search helper
  // This helps us identify duplicate objects
  function searchBasket(needle) {
    return function (haystack) {
      return haystack.pid === needle.pid && haystack.name === needle.name;
    };
  }
```

```
const [basket, setBasket] = useState([
  { pid: 10, name: 14 },
  { pid: 12, name: 17 },
  { pid: 10, name: 14 },
  { pid: 10, name: 14 },
]);
```

Step 2 – deal with the default cases in the array

```
39  let summary = []; // empty summary basket.
40
41  if (props.stateBasket.length <= 0) {
42    summary = [];
43  } else if (props.stateBasket.length === 1) {
44    // only one product in here.
45    // no need to summarise except create a new property called qty
46    // we don't use spread as we can specify the properties in the
47    // summary object
48    summary[0] = {
49      qty: 1,
50      pid: props.stateBasket[0].pid,
51      name: props.stateBasket[0].name,
52    };
```

Step 3 – deal with the case where there are at least two objects

```
// there are at least two objects in this.state.basket.
// they might be duplicates.
// First element in the summary basket is the first element in
// this.state.basket
summary[0] = {
  qty: 1,
  pid: props.stateBasket[0].pid,
  name: props.stateBasket[0].name,
};
// iterate through the rest of this.state.basket.
// we have to check if the current object/product is already
// in the summary basket. If it is - we need to find the "index"
// where it is in the summary basket. Then to add 1 to the qty
```

Step 3 – deal with the case where there are at least two objects (part 2)

```
for (let i = 1; i < props.stateBasket.length; i++) {
  let indexPos = summary.findIndex(searchBasket(props.stateBasket[i]));

  if (indexPos >= 0) {
    // this object/property is in the summary array.
    // it's at position indexPos
    // advance the qty by 1. Then copy the other properties
    let tempQty = summary[indexPos].qty;
    tempQty = tempQty + 1;
    summary[indexPos] = {
      qty: tempQty,
      pid: summary[indexPos].pid,
      name: summary[indexPos].name,
    };
  } else {
    // this object IS NOT in the summary basket.
    // so let's put it into the summary basket - at the end of the
    // array
    summary[summary.length] = {
      qty: 1,
      pid: props.stateBasket[i].pid,
      name: props.stateBasket[i].name,
    };
  } // end if
} // end for
```

Step 4 – render the results of the summarised array

```
return (
  <>
    {summary.length > 0 && (
      <>
        <h1>Your summarised basket (all items)</h1>
        {summary.map((v, index) => (
          <p key={index}>
            Item {index + 1}: {v.pid} {v.name}, Quantity: {v.qty}
          </p>
        ))}
      </>
    )}
  </>
);
```

Summarising objects (things to note)

- We did not mutate state – we summarised on a copy of the array (via props)
- The summary array is NOT stored in state.
- We performed the summary computation in a separate component
- **YOU define what a "duplicate object" means in your application.**
- All Javascript array functions then can be applied to your summary array (sort, map, reduce, filter, and so on)

Organic Shop – Summary Basket

- Create a new component for the Summary (using the code as before)
- Obtain the current Basket as a props variable
- We'll need to change how we calculate the reduce function!

```
5   function Summary(props) {
6     // create a call back for the reduce function
7     // note how we access the price of each object.
8     // now we must consider the quantity of each object in the summary.
9     function getBasketTotal(acc, obj) {
10      return acc + obj.plant.price*obj.qty;
11    }
```

Organic Shop – we use the spread operator in the summary

```
24    if (props.basket.length <= 0) {
25      summary = [];
26    } else if (props.basket.length === 1) {
27      // only one product in here.
28      // no need to summarise except create a new property called qty
29      //
30      summary[0] = {
31        qty: 1,
32        ...props.basket[0]
33      };
34    } else {
35      // there are at least two objects in this.state.basket.
36      // they might be duplicates.
37      // First element in the summary basket is the first element in
38      // this.state.basket
39      summary[0] = {
40        qty: 1,
41        ...props.basket[0]
42      };
```

```
195          {basket.length > 0 && (
196            <>
197              <Summary
198                basket={basket}
199                removeItemFromBasket={removeItemFromBasket}
200                sorting={comparePriceAsc}
201              />
202            </>
203          )}
```

Organic Shop – outputs

Your shopping basket

Your basket has **6** items

Total cost: €150

Rhubarb Stalk roots,€5.00 Remove

Rhubarb Stalk roots,€5.00 Remove

Brambley Apple Tree,€35.00 Remove

Conference Pear Tree,€35.00 Remove

Conference Pear Tree,€35.00 Remove

Brambley Apple Tree,€35.00 Remove

Your shopping basket (summary)

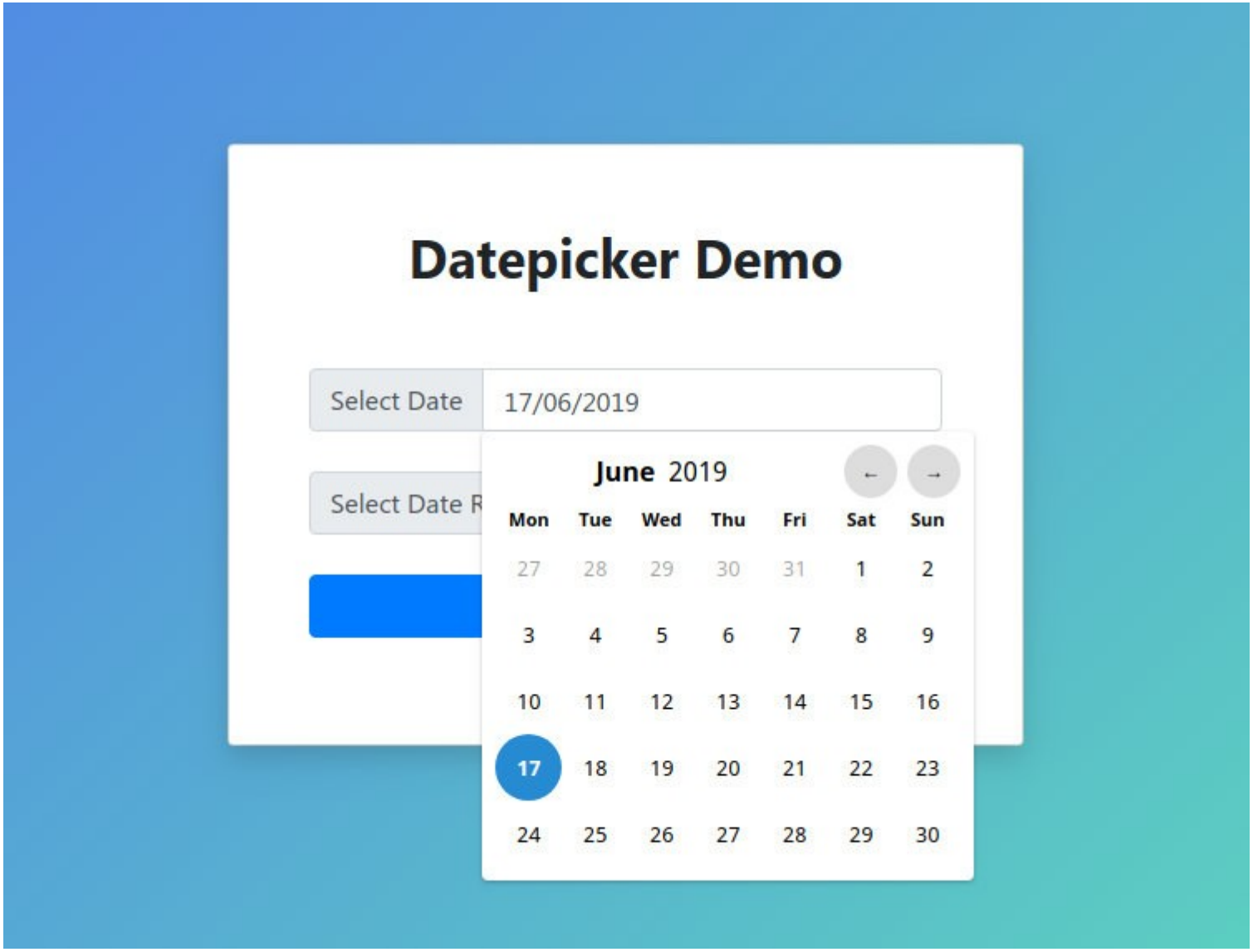
Total cost: €150

Rhubarb Stalk roots,€5.00,**Quantity = 2** Remove

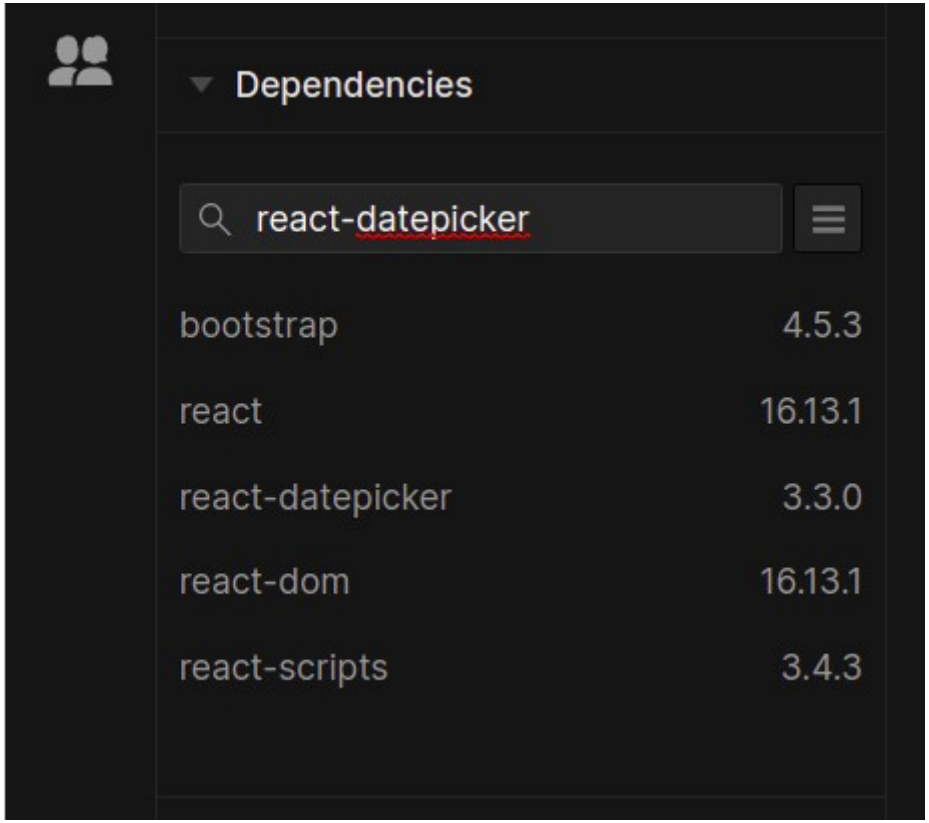
Brambley Apple Tree,€35.00,**Quantity = 2** Remove

Conference Pear Tree,€35.00,**Quantity = 2** Remove

How to add a ‘DATE PICKER’



We add a dependency for the react-datepicker



The react-datepicker package

react-datepicker DT

4.3.0 • Public • Published 25 days ago

Readme | Explore BETA | 6 Dependencies | 1,878 Dependents | 145 Versions

React Date Picker

npm package 4.3.0 | Test suite passing | Dependency Status | codecov 77% | downloads 4M/month

code quality: js/ts B | lgtm alerts 86

A simple and reusable Datepicker component for React (Demo)

Installation

The package can be installed via npm:

```
npm install react-datepicker --save
```

Or via yarn:

Install

```
> npm i react-datepicker
```

Repository

github.com/Hacker0x01/react-datepicker

Homepage

github.com/Hacker0x01/react-datepicker

Weekly Downloads

1,020,419

Version

174

License

MIT

Total Files

20

Pull Requests

14

Last publish

25 days ago

npmjs.com/package/react-datepicker

Configuration

The most basic use of the DatePicker can be described with:

```
<DatePicker selected={startdate} onChange={date => setStartDate(date)} />
```

You can use `onSelect` event handler which fires each time some calendar d
selected

```
<DatePicker
  selected={date}
  onSelect={handleDateSelect} //when day is clicked
  onChange={handleDateChange} //only when value has changed
/>
```

`onClickOutside` handler may be useful to close datepicker in `inline` mo

See here for a full list of props that may be passed to the component. Examples are given on the main website.

Time picker

You can also include a time picker by adding the showTimeSelect prop

```
<DatePicker
  selected={date}
  onChange={handleDateChange}
  showTimeSelect
  dateFormat="Pp"
/>
```

- Good Node packages (such as react-datepicker) will have EXTENSIVE DOCUMENTATION

Step 1 – import DatePicker class and the associated CSS

JS App.js

```
1  import React, { Component } from "react";
2  import DatePicker from "react-datepicker";
3  import "react-datepicker/dist/react-datepicker.css";
4
```

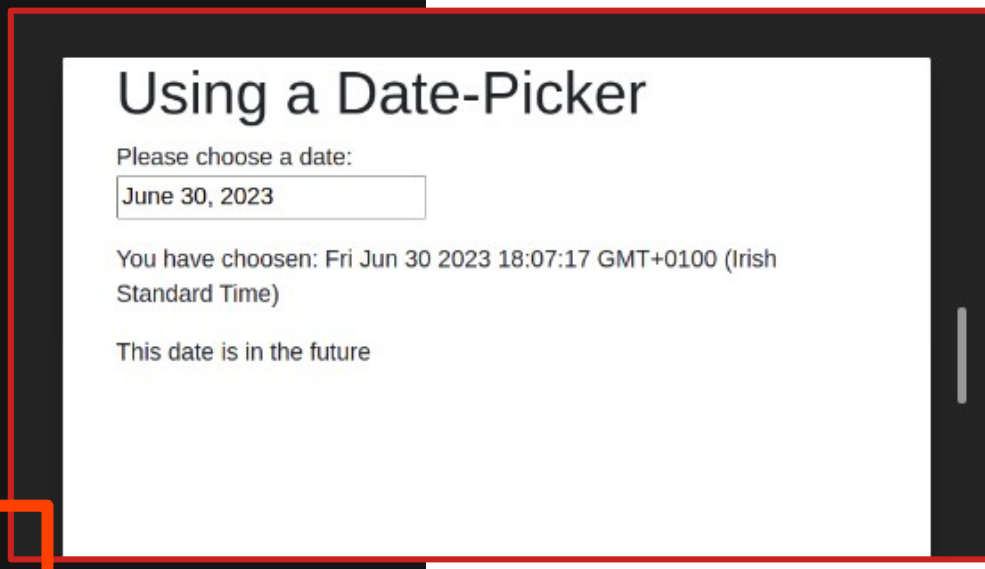
- This is pretty standard for most react dependencies (we have seen this with Bootstrap and its CSS)

Step 2: Basic config requires two state variables or properties

```
6     constructor(props) {
7       super(props);
8       // isSelected is false until we select a date
9       // we set the selected date initially as NOW (new Date())
10      this.state = {
11        selectedDate: new Date(),
12        isSelected: false
13      };
14      this.handleDateChange = this.handleDateChange.bind(this);
15    } // end constructor
16
17    // The callback for the date handler.
18    // There is a parameter which provides us with the date
19    // we can process the date in any way we require
20    handleDateChange(theDate) {
21      this.setState({ isSelected: true });
22      this.setState({ selectedDate: theDate });
23    }
24    // boolean function to indicate if this date is in the future.
25    isFuture() {
26      return this.state.selectedDate > new Date();
27    }
```

Step 3: Using the Date-Picker

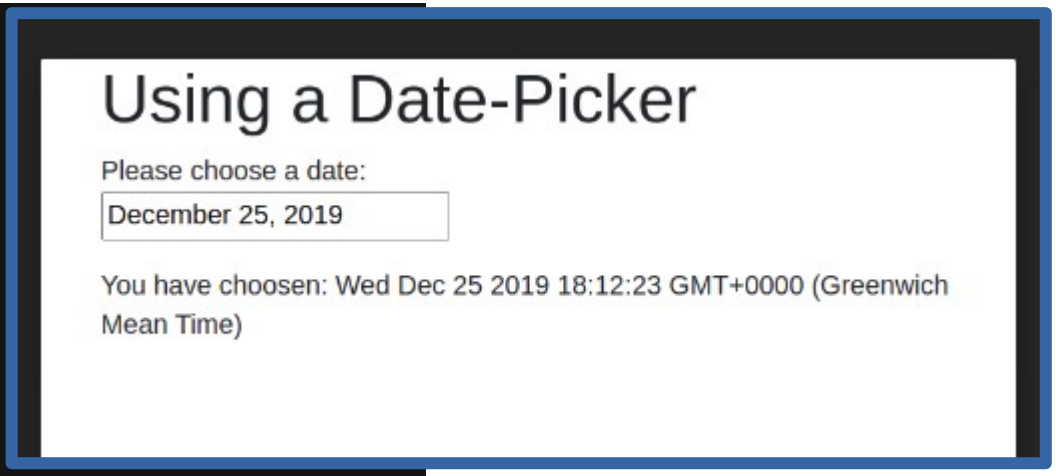
```
28    render() {
29      return (
30        <div className="App">
31          <div className="container">
32            <h1>Using a Date-Picker</h1>
33            <form>
34              Please choose a date:{" "}
35              <div className="form-group">
36                <DatePicker
37                  dateFormat="MMMM d, yyyy"
38                  closeOnScroll={true}
39                  selected={this.state.selectedDate}
40                  onChange={this.handleDateChange}
41                />
42              </div>
43            </form>
44
45            {this.state.isSelected && (
46              <p>You have choosen: {this.state.selectedDate.toString()}</p>
47            )}
48            {this.isFuture() && <p>This date is in the future</p>}
49          </div>
50        </div>
51      ); // end of return statement
52    } // end of render function
```



- We use the documentation to find out how to use the Date-Picker

Step 4 – we add some conditional rendering based on the date chosen

```
29    render() {
30      return (
31        <div className="App">
32          <div className="container">
33            <h1>Using a Date-Picker</h1>
34            <form>
35              Please choose a date:{" "}
36              <div className="form-group">
37                <DatePicker
38                  dateFormat="MMMM d, yyyy"
39                  closeOnScroll={true}
40                  selected={this.state.selectedDate}
41                  onChange={this.handleDateChange}
42                />
43              </div>
44            </form>
45
46            {this.state.isSelected && (
47              <p>You have choosen: {this.state.selectedDate.toString()}
48            )}
49            {this.isFuture() && <p>This date is in the future</p>}
50          </div>
51        </div>
52      ); // end of return statement
53    } // end of render function
```



```
Date.UTC()
Date.now()
Date.parse()
Date.prototype.getDate()
Date.prototype.getDay()
Date.prototype.getFullYear()
Date.prototype.getHours()
Date.prototype.getMilliseconds()
Date.prototype.getMinutes()
Date.prototype.getMonth()
Date.prototype.getSeconds()
Date.prototype.getTime()
Date.prototype.getTimezoneOffset()
Date.prototype.getUTCDate()
Date.prototype.getUTCDay()
Date.prototype.getUTCFullYear()
Date.prototype.getUTCHours()
Date.prototype.getUTCMilliseconds()
```

```
45
46          {this.state.isSelected && (
47            <p>You have choosen: {this.state.selectedDate.toString()}
48            <br/>The year is {this.state.selectedDate.getFullYear()}
49            <br/>The month is {this.state.selectedDate.getMonth()}
50            </p>
51          )}
52          {this.isFuture() && <p>This date is in the future</p>}
```

- Months are indexed from 0 to 11 in Javascript

Using a Date-Picker

Please choose a date:

June 30, 2023

You have choosen: Fri Jun 30 2023 18:16:24 GMT+0100 (Irish Standard Time)
The year is 2023
The month is 5

This date is in the future

That’s the Date Picker

- Feel free to use, if required.
- You can have two or more Date Picker components on the same application (for example for choosing a departure and an arrival date for a flight or journey)
- **Using the Date Picker is an easy way to handle date inputs in a consistent and efficient manner.**

CS385 Lecture 17

More useEffect, working with object changes, summarising object arrays, the Date Picker