React II Exercises

ex01: making an API call - async js revision

challenge

Use this URL - `https://example.com/api/products` to make a fake fetch call and list out all the items as an ordered list on the DOM. A fake fetch has been provided.

```
const fakeFetch = (url) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (url === "https://example.com/api/products") {
        resolve({
          status: 200,
          message: "Success",
          data: {
            products: [
              { name: "Pen", price: 30, quantity: 100 },
              { name: "Pencil", price: 50, quantity: 50 },
              { name: "Paper", price: 20, quantity: 30 }
            ]
          }
        });
      } else {
        reject({
          status: 404,
          message: "Items list not found."
        });
      }
    }, 2000);
  });
};
// Output on the DOM should be in the format, {name} -- INR {price} -- {quantity}:

// Pen -- INR 30 -- 100
// Pencil -- INR 50 -- 50
// Paper -- INR 20 -- 30
```

COPY

1. Pen -- INR 30 -- 100
2. Pencil -- INR 50 -- 50
3. Paper -- INR 20 -- 30

solution

https://replit.com/@tanaypratap/react-II-exercise-1

making API calls from React

ex02: API call on button click - console

understanding

`handleData` fetches data from a fake API endpoint using `fakeFetch`.

The function first tries to fetch the data using a `try-catch` block. If the data is successfully fetched, then the employee data is logged in the console.

If there's an error, then the error message is caught in the `catch` block.

challenge

Add an onclick event on the button with the text "get employee details" to call the handleData function. The handleData function should use the fakeFetch function to retrieve employee data from `https://example.com/api/employees`.

If the response status is 200, the function should log the employee data to the console.

https://codesandbox.io/s/api-call-on-button-click-console-challenge-etbb1h

solution

https://codesandbox.io/s/api-call-on-button-click-console-solution-udh6lw

ex03: API call on button click - view

understanding

`handleData` fetches data from a fake API endpoint using `fakeFetch`.

The function first tries to fetch the data using a `try-catch` block. If the data is successfully fetched, then the product data is logged in the console and the state is updated with the product data and displayed on the screen using a map.

If there's an error, then the error message is caught in the `catch` bloc.

challenge

Add an onclick event on the button with the text "get products details" to call the handleData function.

The handleData function should use the fakeFetch function to retrieve product data from `https://example.com/api/products`.

If the response status is 200, the function should update the state with the product data and display it on the screen. Each product item should display the name, price, and quantity.

https://codesandbox.io/s/api-call-on-button-click-view-challenge-rve802

solution

https://codesandbox.io/s/api-call-on-button-click-view-solution-xm0207

ex04: API call on button click - view w/ highlight

challenge

Click on the "highlight transactions > 1000" button should highlight all transactions with a amount greater than 1000.

https://codesandbox.io/s/api-call-on-button-click-view-w-highlight-challenge-ysev7b

solution

https://codesandbox.io/s/api-call-on-button-click-view-w-highlight-solution-zpn5hu

API call on load

Just like onClick, a load is also just an event.

ex05: useEffect - print on load

challenge

Predict the output with the order.

```
useEffect(cb, [])
function App() {
  useEffect(() => {
    console.log("from useEffect..."); // B
  },[]);

  console.log("before render..."); // A

  return (
    <div className="App">
      <h1 className="app-header">tanaypratap's box</h1>
    </div>
  );
}
```

COPY

understanding

1. "before render..."
2. "from useEffect..."

This is because the `console.log` statement inside `useEffect` is executed after the component has rendered. So first the `return` statement is executed and the component is rendered, logging "before render..." to the console. Then the `useEffect` hook is executed, logging "from useEffect..." to the console.

solution

https://codesandbox.io/s/useeffect-print-on-load-7gilg0

ex06: useEffect - predict the output and order

understanding

- It would be good to reason in your mind about the order of how things run in React.
- We had covered before that `useState()` triggers re-render.
- If you were unable to predict this, run the program and check.

challenge

- What will be the console for the initial render?
- What will be the console after the user clicks the increment button?

```
export default function App() {
  const [counter, setCounter] = useState(0);

        console.log("counter", counter);

  useEffect(() => {
    console.log("from useEffect...", counter);
  },[]);

  function incrementClickHandler() {
    setCounter((counter) => {
      console.log("from click handler...", counter);
      return counter + 1;
    });
  }

  console.log("before render...", counter);
  return (
    <div className="App">
      <h1>{counter} </h1>
      <button onClick={incrementClickHandler}>Increment </button>
    </div>
  );
}

/**
before render
from useEffect
USER CLICKS ON COUNTER
from click handler...
before render...
from useEffect ?? <--- NO

**/

/**
RENDER ZER0
counter 0
before render
from useEffect

USER CLICKS ON COUNTER - RENDER ONE
counter 1
before render
**/
```

COPY

output

1. Initially, the component is rendered and logs "before render... 0" to the console.
2. The `useEffect` hook runs and logs "from useEffect... 0" to the console.
3. The user clicks the button, and the `incrementClickHandler` function is called, which logs "from click handler... 0" to the console.
4. The `setCounter` function updates the state value of `counter` to 1.
5. The component is re-rendered, and logs "before render... 1" to the console.

solution

https://codesandbox.io/s/useeffect-print-on-load-ii-predict-output-6xlg4t

ex07: useEffect - success on console

understanding

Use `useEffect` to call the `fetchData` function when the component first renders.

challenge

In this challenge, you need to load product data using `useEffect` and `fakeFetch`

https://codesandbox.io/s/useeffect-success-on-console-challenge-iy2t9n

solution

https://codesandbox.io/s/useeffect-success-on-view-solution-74mc3f

ex08: useEffect - success on view

understanding

In the `useEffect` hook, call the `fetchData` function to fetch data when the component mounts. Use an empty dependency array `[]` to ensure that the `useEffect` hook only runs once.

challenge

In this challenge, you will use the `useEffect` and `useState` hooks to fetch data from an API using the `fakeFetch` function and display it on the screen.

https://codesandbox.io/s/useeffect-success-on-view-challenge-9gs79f

solution

https://codesandbox.io/s/useeffect-success-on-view-solution-5mctg1

Note: This is not how it's done in production, this is more about your understanding on how to load data from the server.

ex09: useEffect - loading state

understanding

Define an async function called `fetchData` that uses the `fakeFetch` function to fetch the product data from the API endpoint. Inside the function, set the `isLoading` state variable to `true` before making the API call.

If the call is successful, set the `data` state variable to the response data and set the `isLoading` state variable to `false`. If the call fails, log the error to the console.

challenge

In addition to the above solution, you need to display a loading state while the data is being fetched.

https://codesandbox.io/s/useeffect-success-on-view-solution-5mctg1

solution

https://codesandbox.io/s/useeffect-loading-state-solution-y35eqx

ex10: useEffect - error state

understanding

Use a try-catch block to catch any errors and set the **`isError`**state to `true` if there is an error. Set the `data` state to the product data from the response and set the `isLoading` state to `false`. Finally, set the **`isLoading`**state to `false` in a `finally` block.

challenge

In this challenge, you will need to fetch data from an API and display it on the screen. You will also need to show a loading state while the data is being fetched and an error message if the request fails.

https://codesandbox.io/s/useeffect-loading-state-challenge-eesj2s

solution

https://codesandbox.io/s/useeffect-error-state-solution-toor6q

challenges

ex11: API call on button click - view w/ filter

challenge

Your task is to modify the component to display cart items with a price greater than or equal to 50.

https://codesandbox.io/s/api-call-on-button-click-view-w-filter-question-003fef

solution

https://codesandbox.io/s/api-call-on-button-click-view-w-filter-0ypi3n

ex12: useEffect - success on view w/ filter

challenge

Your task is to modify the component to display wishlist items with a price greater than or equal to 100 but with useEffect - onload.

https://codesandbox.io/s/useeffect-success-on-view-w-filter-question-3ss7zg

solution

https://codesandbox.io/s/useeffect-success-on-view-w-filter-srx44h