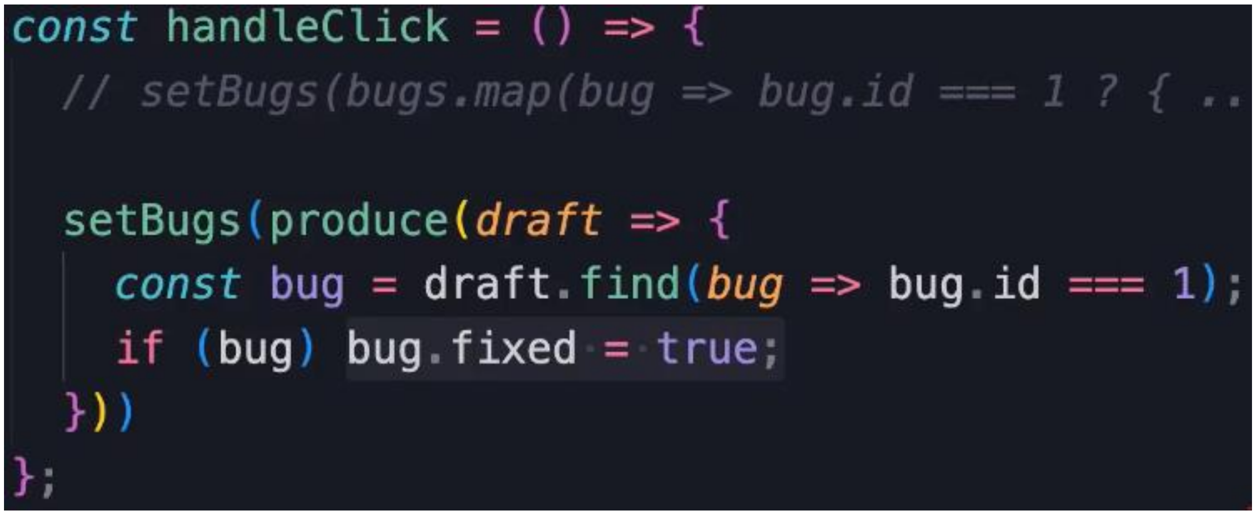
To install the immer library using npm: ***npm i immer***.

Where produce-function is from immer-library.



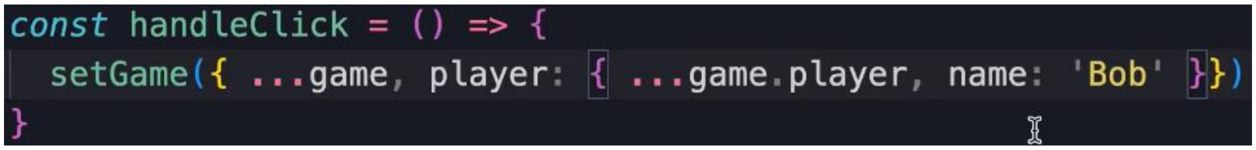
```
const handleClick = () => {
  // setBugs(bugs.map(bug => bug.id === 1 ? { ..

  setBugs(produce(draft => {
    const bug = draft.find(bug => bug.id === 1);
    if (bug) bug.fixed = true;
  }))
};
```

Here draft is like the proxy-pattern where it represents the object that we want to update in the array.

****

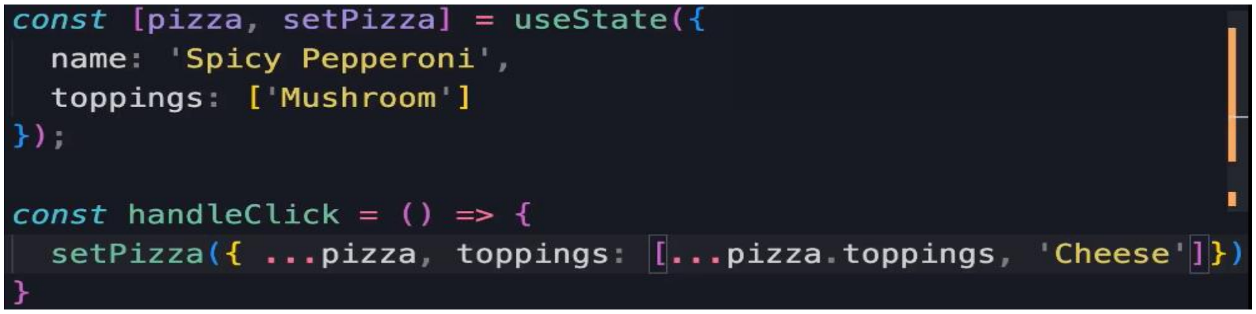
When passing the data between multiple components we must set the managing of state inside the main component.

****



```
const handleClick = () => {
  setGame({ ...game, player: { ...game.player, name: 'Bob' }})
}
```

****



```
const [pizza, setPizza] = useState({
  name: 'Spicy Pepperoni',
  toppings: ['Mushroom']
});

const handleClick = () => {
  setPizza({ ...pizza, toppings: [...pizza.toppings, 'Cheese']})
}
```

****

Another hook that we can use for reference fields of Forms is: useRef.

****

```
const nameRef= useRef<HTMLInputElement>(null);
const ageRef = useRef<HTMLInputElement>(null);
```

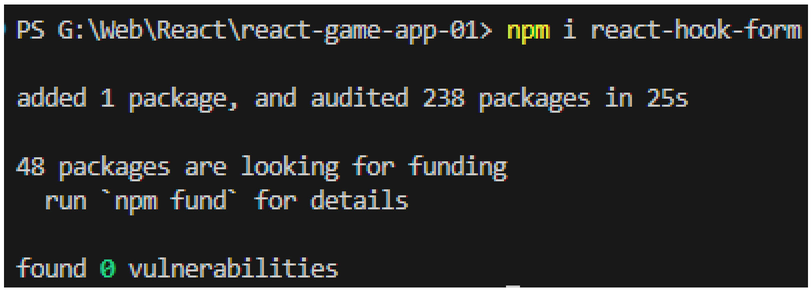
Then to use them with input fields:

```
<input type="text" className="form-control" id='name' ref={nameRef} />
<input type="number" className="form-control" id='age' ref={ageRef} />
```

```
const handleSubmit = (event: FormEvent) => {
  event.preventDefault();
  console.log("Data Has Been Submitted");

  console.log(nameRef.current);
  console.log(nameRef.current?.value);

  console.log(ageRef.current);
  console.log(ageRef.current?.value);
}
```

****

To simplify the use of states with forms we can install: react-hook-form.



```
PS G:\Web\React\react-game-app-01> npm i react-hook-form

added 1 package, and audited 238 packages in 25s

48 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

****

```
import { useForm } from "react-hook-form";
```

```
const { register } = useForm();
```

```
<input type="text" className="form-control" id='name' { ...register('name') } />
```

```
<input type="number" className="form-control" id='age' { ...register('age') } />
```

****

To Handle the Submit of the form we can use handleSubmit from useForm-hook:

```
const { register, handleSubmit, formState } = useForm();
```

To use it we can pass function to it to handle the data that we submit to the server:

```
const onSubmit = (data: FieldValues) => console.log(data);
```

```
<form onSubmit={ handleSubmit(onSubmit) }> ...Here we set the form </form>
```

****

To get the state of current form we can use formState it has many parameters: formState.errors to get the full errors of validation.

```
console.log(formState);
console.log(formState.errors)
```

****

To use nested destruction of Object:

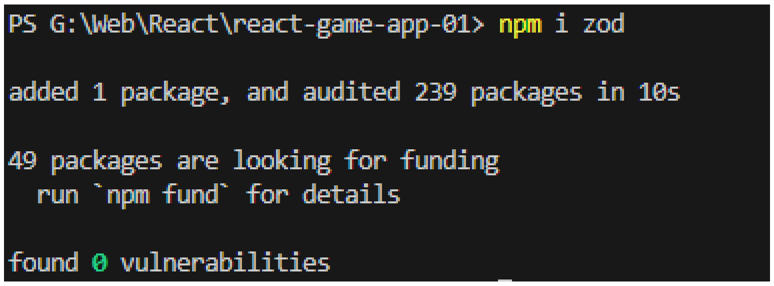
```
const { register, handleSubmit, formState: { errors } } = useForm();
```

****

To render error messages, we can set:

```
{ errors.name?.type === 'required' && <p>This Field is Required</p>}
{ errors.name?.type === 'minLength' && <p>This Field At Least 3-chars</p>}
```

****



```
PS G:\Web\React\react-game-app-01> npm i zod

added 1 package, and audited 239 packages in 10s

49 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

****

The Schema based validation method: zod-module.

In This way we can define the validation schema of our data.

****

```
import { z } from "zod";


const schema = z.object({
  name: z.string().min(3),
  age: z.number().min(18),
});


type FormDataValue = z.infer<typeof schema>;
```

By using ***z.object*** we define the schema of our Validation.

By using ***infer*** we can define the shape of our form.

****

Note (To Remember): To get the value of input field we can access to it using: ***event.target.value*** OR ***event.currentTarget.value***.

****

Note (To Remember): To get the input-element with validation using react-hook-form:

```
{ ...register('name', { required: true, minLength: 3}) }
```

****

```
const schema = z.object({
  // name: z.string().min(3).optional(),
  name: z.string().min(3, "This Field must be at least 3-characters"),
  age: z.number({ invalid_type_error: "You must enter Your Age" }).min(18, "You must be 18 or older"),
});
```

****

To get if the form state is valid or not we can use isValid from FormState of useForm:

```
const { register, handleSubmit, formState: { errors, isValid } } = useForm<FormDataValue>({ resolver:
zodResolver(schema)});
```

****

Then to disable the submit button if the form is not valid:

```
<button disabled={!isValid} className="btn btn-primary" type="submit">Save The Values</button>
```

****

For making Code More Readable, we can use _ with numbers: like we use: 1000_000:

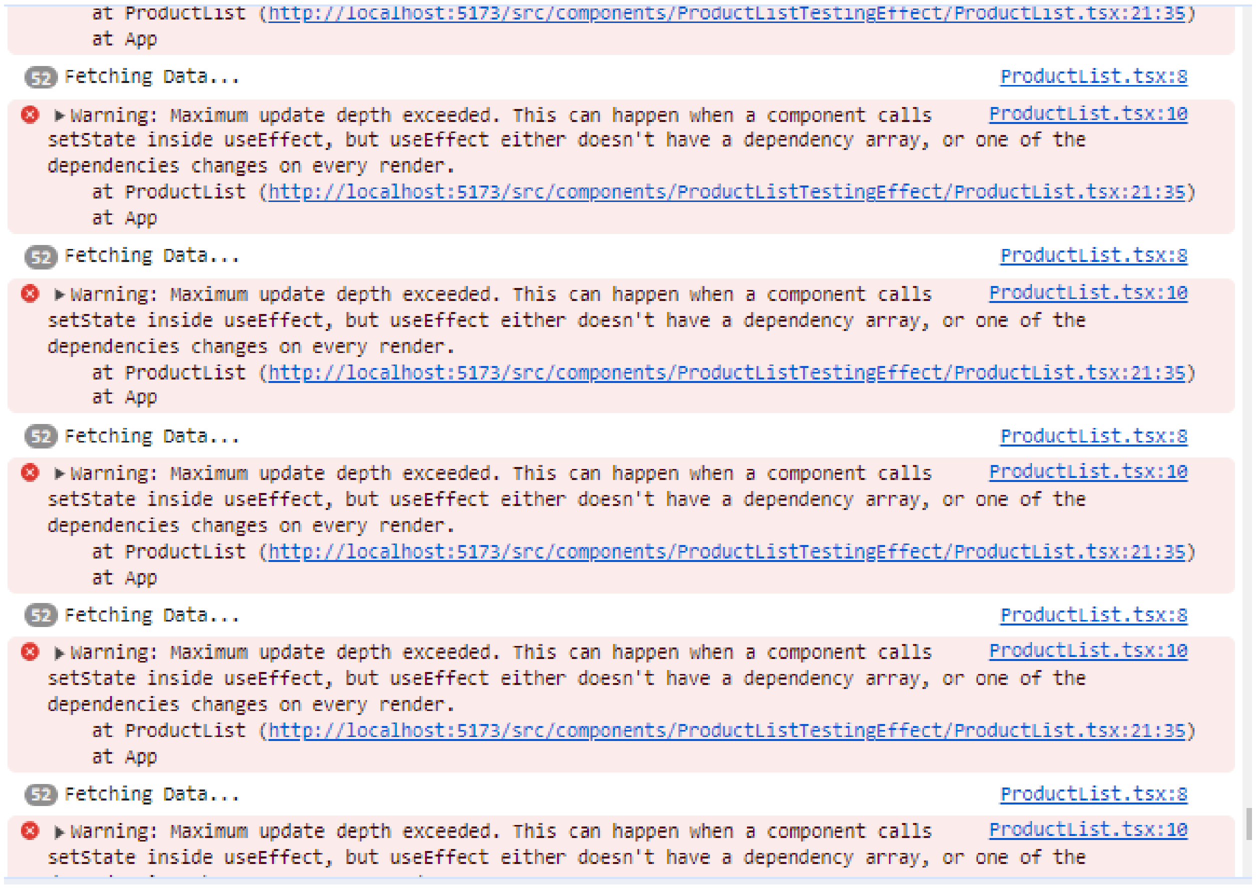
```
amount: z
```

```
  .number({ invalid_type_error: "You must enter Product Amount." })
  .min(10_000, "You must enter value > 1000.")
  .max(1000_000, "You must enter value < 1000000."),
```
****

To make the array constant with constant values we can declare it like:

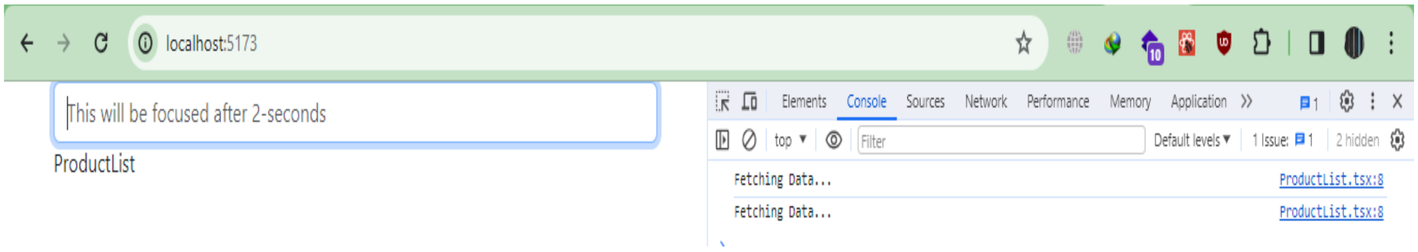
```
export const categories = [
 { name: "J-L-01", val: "Jafar-Loka-01" },
 { name: "J-L-02", val: "Jafar-Loka-02" },
 { name: "J-L-03", val: "Jafar-Loka-03" },
 { name: "J-L-04", val: "Jafar-Loka-04" },
] as const;
```
****

When using Side Effect Hook, we must be careful to the dependencies of the Side Effect Hook, else; A max depth of browser stack will be exceeded.



```
    at ProductList (http://localhost:5173/src/components/ProductListTestingEffect/ProductList.tsx:21:35)
    at App
52 Fetching Data...                                                              ProductList.tsx:8
▶Warning: Maximum update depth exceeded. This can happen when a component calls  ProductList.tsx:10
setState inside useEffect, but useEffect either doesn't have a dependency array, or one of the
dependencies changes on every render.
    at ProductList (http://localhost:5173/src/components/ProductListTestingEffect/ProductList.tsx:21:35)
    at App
52 Fetching Data...                                                              ProductList.tsx:8
▶Warning: Maximum update depth exceeded. This can happen when a component calls  ProductList.tsx:10
setState inside useEffect, but useEffect either doesn't have a dependency array, or one of the
dependencies changes on every render.
    at ProductList (http://localhost:5173/src/components/ProductListTestingEffect/ProductList.tsx:21:35)
    at App
52 Fetching Data...                                                              ProductList.tsx:8
▶Warning: Maximum update depth exceeded. This can happen when a component calls  ProductList.tsx:10
setState inside useEffect, but useEffect either doesn't have a dependency array, or one of the
dependencies changes on every render.
    at ProductList (http://localhost:5173/src/components/ProductListTestingEffect/ProductList.tsx:21:35)
    at App
52 Fetching Data...                                                              ProductList.tsx:8
▶Warning: Maximum update depth exceeded. This can happen when a component calls  ProductList.tsx:10
setState inside useEffect, but useEffect either doesn't have a dependency array, or one of the
dependencies changes on every render.
    at ProductList (http://localhost:5173/src/components/ProductListTestingEffect/ProductList.tsx:21:35)
    at App
52 Fetching Data...                                                              ProductList.tsx:8
▶Warning: Maximum update depth exceeded. This can happen when a component calls  ProductList.tsx:10
setState inside useEffect, but useEffect either doesn't have a dependency array, or one of the
```

If we pass empty array [], to useEffect-hook it will only fetch the data twice.



This will be focused after 2-seconds

ProductList

```
Fetching Data...                                  ProductList.tsx:8
Fetching Data...                                  ProductList.tsx:8
```

****

```
const ProductList = () => {
  const [products, setProducts] = useState<string[]>([]);
  useEffect(() => {
    console.log('Fetching Data...');
    setProducts(['Jafar-Loka-01', 'Jafar-Loka-02', 'Jafar-Loka-03']);
  }, []);
  return (
    <div>ProductList</div>
  )
}
```
****

<u>Note (To Remember)</u>: When the strict mode is on React will Render the Component Twice.

<u>Note</u>: This Behavior happened only on development mode, not in production mode.

****

When we use useState with useEffect we should be attention to call the data, because useState fire new render, and useEffect execute After Rendering data, so we finish in infinite loop.

****

So, to solve the above problem we must set the dependencies array, so when we pass empty array, it means there are no dependency at all.

But if we set any parameter, it will be fired each time the parameter is changed.

****

When we use useEffect to fetch our data, then we should be attention to dependency parameter.

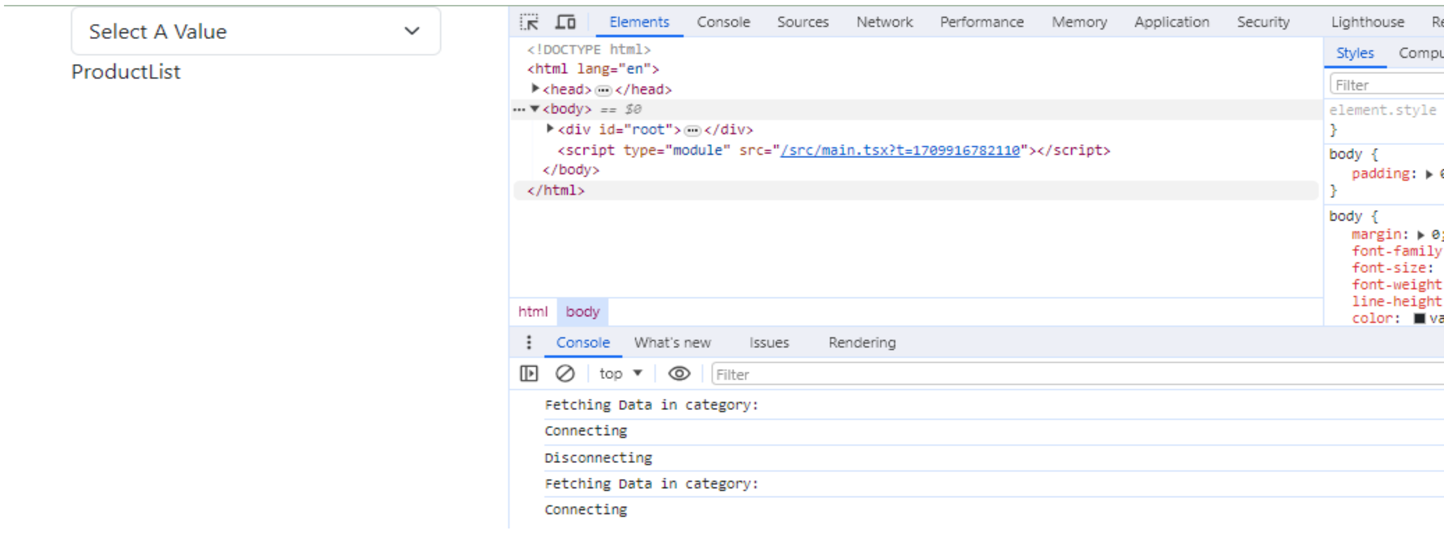
****

```
const [products, setProducts] = useState<string[]>([]);
  useEffect(() => {
    console.log('Fetching Data in category: ', category);
    setProducts(['Jafar-Loka-01', 'Jafar-Loka-02', 'Jafar-Loka-03']);
  }, [category]);
```
****

To clean the data of useEffect we should return a function at the end of useEffect:

```
const connect = () => console.log('Connecting');
  const disconnect = () => console.log('Disconnecting');

  useEffect(() => {
    console.log('Fetching Data in category: ', category);
    setProducts(['Jafar-Loka-01', 'Jafar-Loka-02', 'Jafar-Loka-03']);
    connect();
    return () => disconnect();
  }, [category]);
```
****

When unmounting the component from screen the clean function will be called:



```
<!DOCTYPE html>
<html lang="en">
  <head>…</head>
… <body> == $0
    <div id="root">…</div>
    <script type="module" src="/src/main.tsx?t=1709916782110"></script>
  </body>
</html>
```

```
Fetching Data in category:
Connecting
Disconnecting
Fetching Data in category:
Connecting
```

Here, as we use StrictMode, each component will be rendered twice, at the first time: the component will be rendered then cleaned from screen, then a second render, the component will be on the screen.

****

Note: All Time the Dependency parameters are changed the component will be unmounted, and the clean function will be called.

****

Note: The Empty Array []-of dependencies mean that, the useEffect will be executed only once, because there are no dependencies.

****

Note (From Me): When we use useEffect && useState in the same file, we must be attention to the useEffect dependency parameters, else; it will be executed in infinite loop.

****

Note: The useEffect()-hook doesn't accept async functions.

Note 2: to use async/await with useEffect, we can declare function that is using async and await, then inside the body of useEffect we can call the function.

Note (From Me): When we define the controller as Main Parameter inside the React Component then all the requests will be cancelled!!! (Search For it).

****

Note: The Best Place to set the state of loading data is inside the finally()-method.

Note 2: In Strict Mode the finally-method may be not worked.

****