Contents

[2. React and JSX Fundamentals 3](#_Toc173084509)

[1. Create a react app using Vite: 3](#_Toc173084510)

[2. If-else statement in JSX using ternary Operator 4](#_Toc173084511)

[3. Anonymous Function 5](#_Toc173084512)

[4. JSX For-Loop 6](#_Toc173084513)

[5. JSX Conditional Rendering Using IF-Else: 7](#_Toc173084514)

[6. JSX Conditional Rendering Using Switch Statement: 8](#_Toc173084515)

[7. JSX Conditional Rendering Using && Operator: 8](#_Toc173084516)

[8. Passing Properties to Child Component: 9](#_Toc173084517)

[Passing a String 9](#_Toc173084518)

[Passing an Object: 10](#_Toc173084519)

[Passing a Function: 12](#_Toc173084520)

[9. Managing C lick Event: 14](#_Toc173084521)

[Wrong Way: 14](#_Toc173084522)

[Correct Way By Using Arrow Function: 15](#_Toc173084523)

[Correct Way By Using Regular Function: 15](#_Toc173084524)

[10. File structure of this chapter: 16](#_Toc173084525)

[11. App.jsx of this chapter: 17](#_Toc173084526)

[12. Main.jsx of this chapter: 17](#_Toc173084527)

[13. Output Result of this chapter: 18](#_Toc173084528)

[3. React Hook and State Manager: 19](#_Toc173084529)

[1. useRef 19](#_Toc173084530)

[Changing the innerText 19](#_Toc173084531)

[Changing the innerHtml 20](#_Toc173084532)

[Compile Results: 21](#_Toc173084533)

[Changing the innerText or HTML by arrow function: 22](#_Toc173084534)

[Setting Attributes: 23](#_Toc173084535)

[Input Elements: 24](#_Toc173084536)

[Working with Css: 25](#_Toc173084537)

[UseRef and Caching 27](#_Toc173084538)

[File structure of useRef Section: 29](#_Toc173084539)

[App.jsx of this chapter: 30](#_Toc173084540)

[Main.jsx of this chapter: 30](#_Toc173084541)

[Output Result of this chapter: 31](#_Toc173084542)

[2. useState 33](#_Toc173084543)

[Immutable objects: 34](#_Toc173084544)

[Immutable Arrays (Adding to the Array) 36](#_Toc173084545)

[Immutable Arrays (Delete Array Element) 37](#_Toc173084546)

[Immutable Arrays (Delete Array Element)\*common method: 38](#_Toc173084547)

[Managing forms: 39](#_Toc173084548)

[File structure of useRef Section: 42](#_Toc173084549)

[App.jsx of this chapter: 43](#_Toc173084550)

[Main.jsx of this chapter: 43](#_Toc173084551)

[Output Result of this chapter: 44](#_Toc173084552)

[3. useEffect 45](#_Toc173084553)

# React and JSX Fundamentals

## Create a react app using Vite:

My node version on the Ubuntu WSL was outdated so I ran this command to install the latest long term support version of it:

**

In order to create a Vite project this command should be used and the Vite should be followed:



In order not having to install npm packages every time for each project and using the computer hard disk unnecessary und inefficient, pnpm instead will be used. Pnpm uses hard links and symlinks to save one version of a module only ever once on a disk. for installing it, this command have been used:



After that, instead of npm install, pnpm install will be used.

For running the program using pnpm, the following command should be used:



For running the program using npm, the following command should be used:



Project can also be run by using the Vite command:



## If-else statement in JSX using ternary Operator

In React, component names should start with an uppercase letter



## Anonymous Function



## JSX For-Loop

Using map function and then calling an anonymous function inside the map-function:

* Don’t forgot to add ‘return’ to the map function!



## JSX Conditional Rendering Using IF-Else:

In this code, more efficient and cleaner way, of implementing If-Else by using functions is illustrated: 

## JSX Conditional Rendering Using Switch Statement:



## JSX Conditional Rendering Using && Operator:

If condition is true, it will execute the code after the ‘&&’ operator, otherwise it won’t execute something else.



## Passing Properties to Child Component:

It has a unidirectional flow, so you cannot pass components from child to parent.

### Passing a String

Like html we will use Attributes:

***Parent Component:***



***Child Component:***



### Passing an Object:

***Parent Component:***



***Child Component:*** 

### Passing a Function:

***Parent Component:***



***Child Component:*** 

## Managing C lick Event:

### Wrong Way:

If you implement a function in this way, as illustrated in the code below, the browser will constantly keep running this function whenever the user refreshes the page instead of running it only when the button is clicked.

***App Component:***



***Button function in ‘Managing\_Click\_Event’ Component:*** 

### Correct Way By Using Arrow Function:

***Button function in ‘Managing\_Click\_Event’ Component:*** 

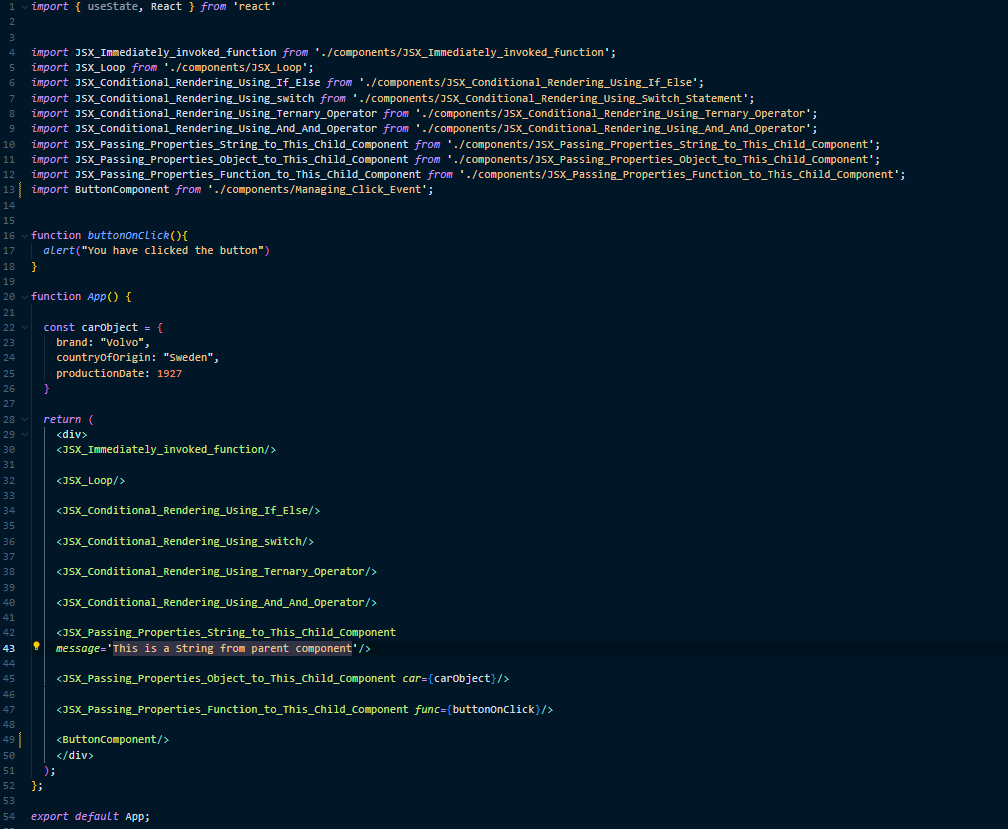
### Correct Way By Using Regular Function:



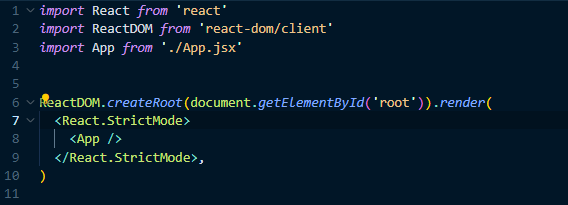
## File structure of this chapter:



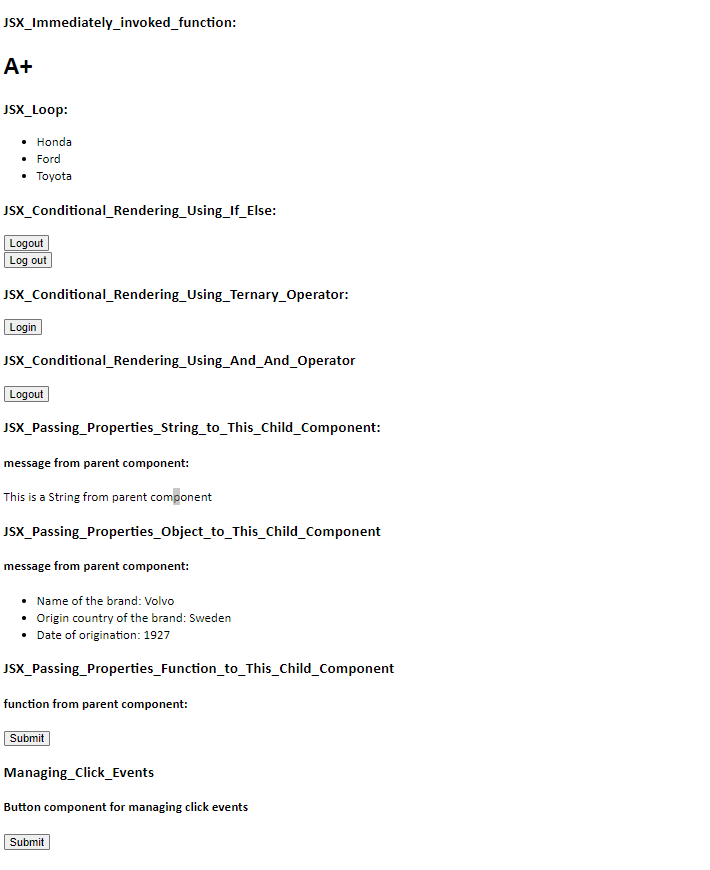
## App.jsx of this chapter:



## Main.jsx of this chapter:



## Output Result of this chapter:



# React Hook and State Manager:

## useRef

### Changing the innerText

***1\_useRef\_InnerText.jsx file:***



***App\_components2\_react\_hook.jsx file:***



***Main.jsx file:***

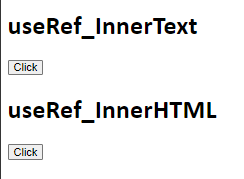


### Changing the innerHtml

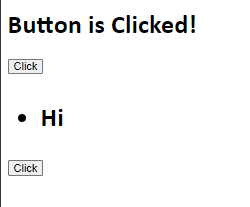


### Compile Results:

**Befor:**



**After:**



### Changing the innerText or HTML by arrow function:

* Compile results will be same as before
* Don’t forgot to remove “current” keyword



### Setting Attributes:

Changing the ‘src’ and Height of an image, when a button is clicked:



### Input Elements:

This code will get the first name and last name, which are entered in the placeholder, and then show them when a button is clicked



### Working with Css:

For this section bootstrap is used, so make sure to install it and then import in in the main.jsx file

**Main.jsx:**



Clicking the button will change the header’s color. The initial color is green, and it will be changed to red:



### UseRef and Caching



For instance, we want to reuse the result of an API multiple times without recomputing it every time the component renders. We can use useRef to store the API data, eliminating the need to fetch it again.

**UseRef\_Caching Component**

This component demonstrates the use of the useRef hook to cache data and manipulate DOM elements.

**useRef Hook**

* data: Stores fetched data.
* pTag: References the <p> tag.

**fetchData Function**

This asynchronous function fetches data from an API and stores it in the data ref. If the fetch operation fails, it catches the error and handles it.

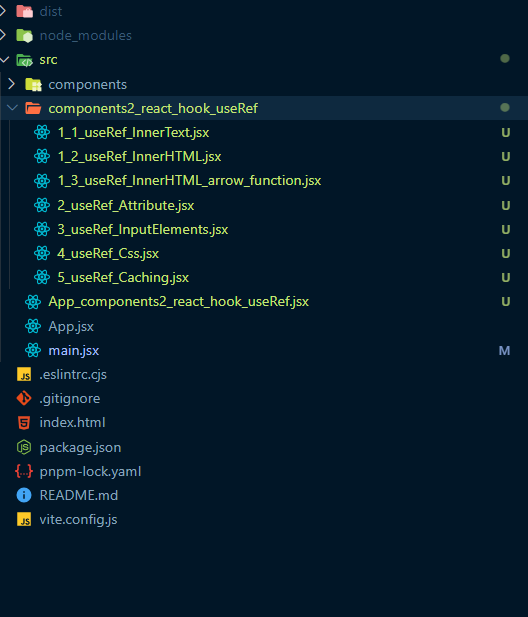
**showData Function**

This function displays the fetched data inside the <p> tag by updating its innerHTML.

**Return Statement**

* <h1>: Displays the title “UseRef\_Caching”.
* <p>: Placeholder for displaying fetched data.
* <button>: Triggers fetchData function.
* <button>: Triggers showData function.

### File structure of useRef Section:

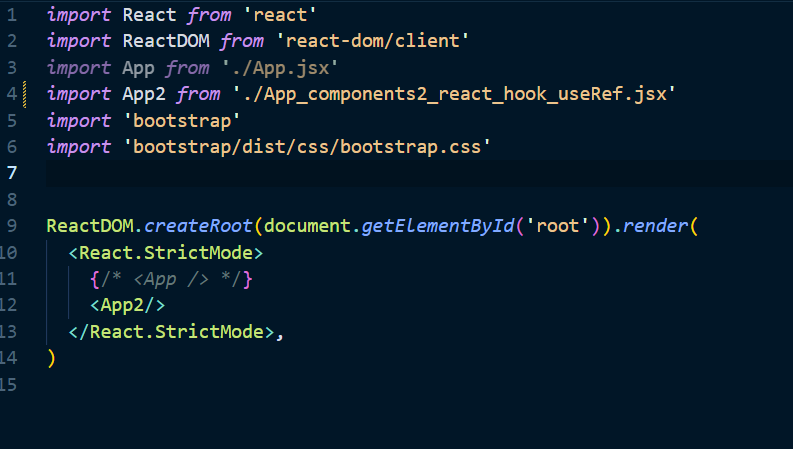


### App.jsx of this chapter:

App.jsx of this chapter is called `App\_components2\_react\_hook\_useRef `

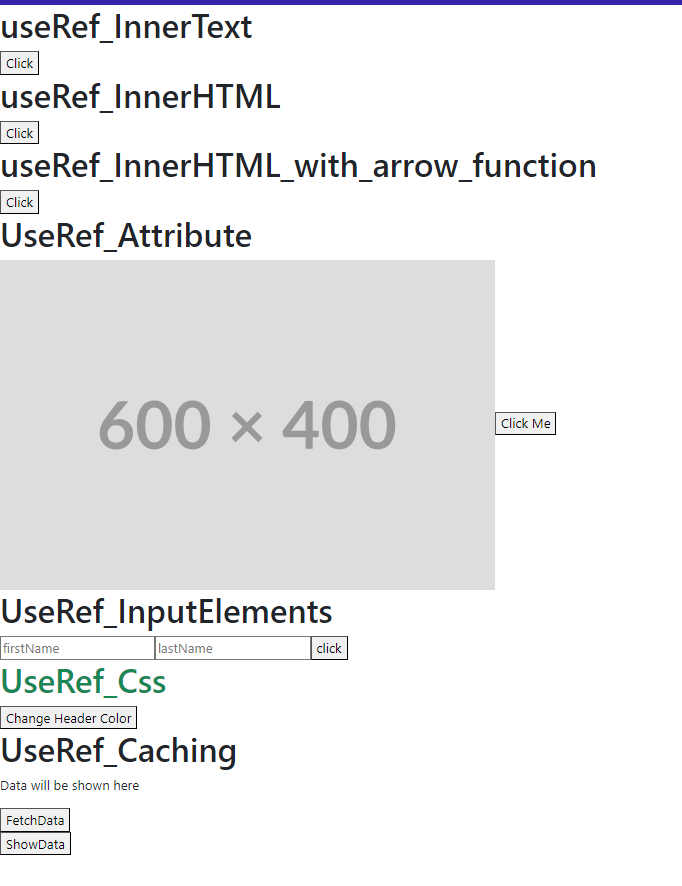


### Main.jsx of this chapter:

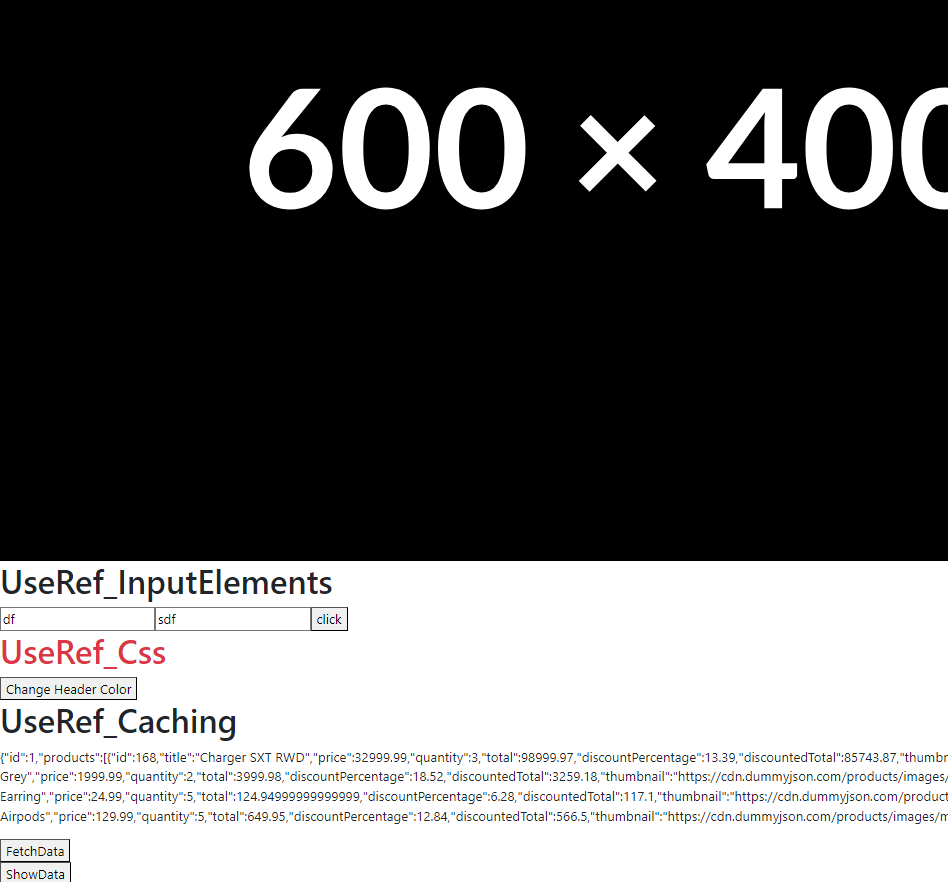


### Output Result of this chapter:

**Befor clicking:**



**After clicking:**



## useState

useState is a built in react object, which is used to contain data or information of an component.

The state can also be modified based on users inputs or network changes

This example code will increment the number by one after clicking the button:



### Immutable objects:

**Example code of creating and showing an Object:**



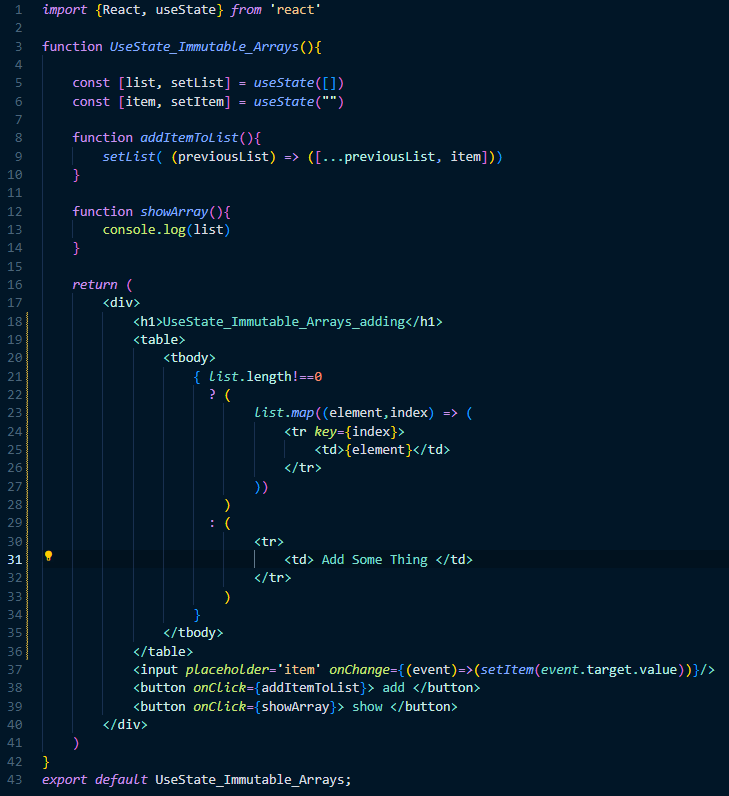
**Example code of changing an Object:**

This example code will change the `origin` key, after the button is clicked:



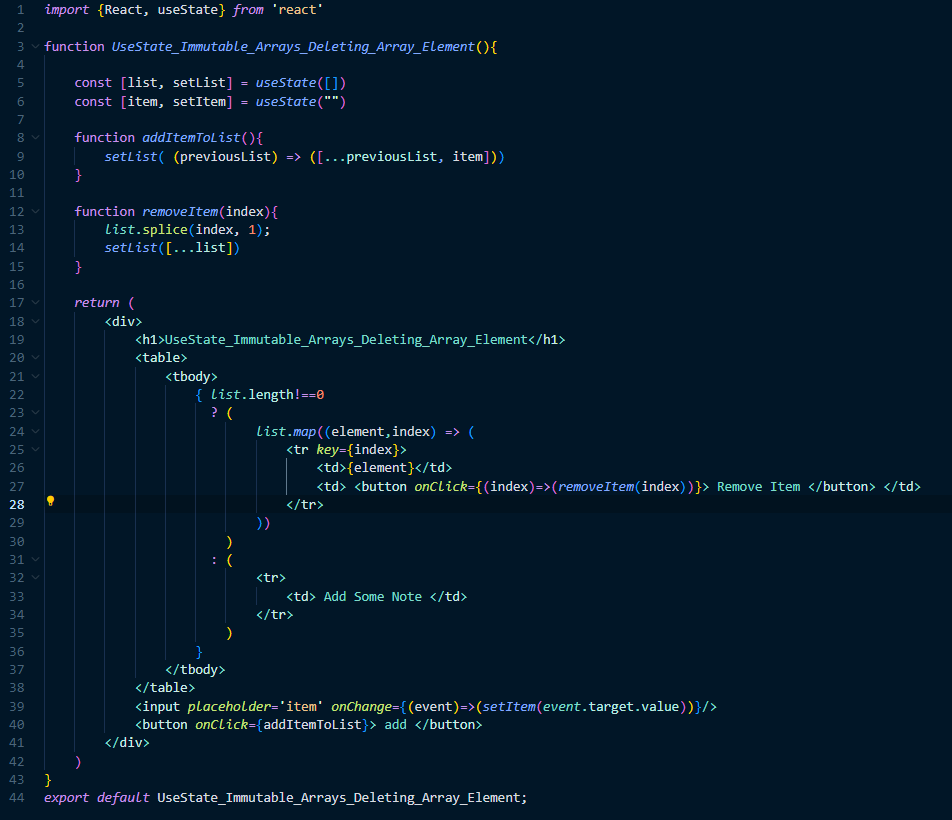
### Immutable Arrays (Adding to the Array)

In this code below a simple ‘To-Do’ app is created:



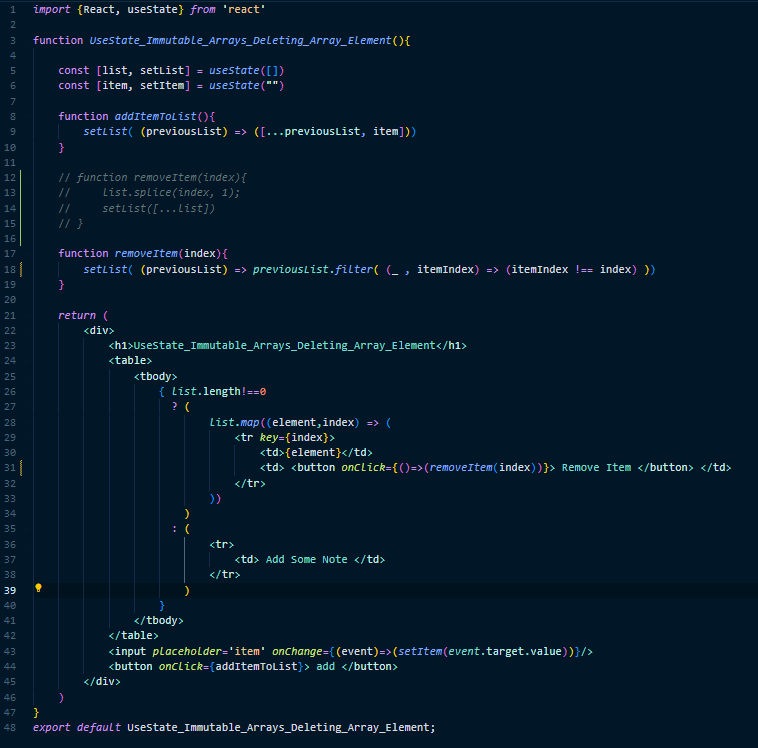
### Immutable Arrays (Delete Array Element)

In the code below, a new button called ‘Remove Item’ is added, which will trigger a function to delete that specific element from the array holding the to-do list elements.



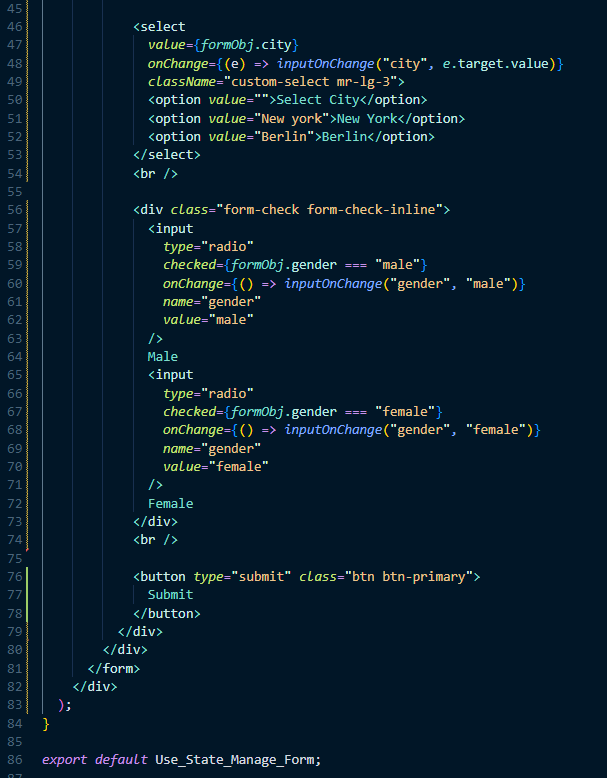
### Immutable Arrays (Delete Array Element)\*common method:

In the code below, the removeItem function has been updated to demonstrate a more efficient and commonly used method for deleting an element from an array

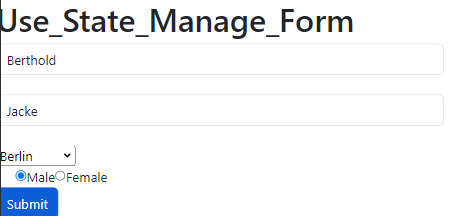


### Managing forms:

* Define an Object in useRef:
* Write a Function to Manage Input Changes:
* Create a Form
* Add Values to Inputs Using the Reference:
* Add onChange Attribute to Inputs:
* Write an Function to Manage the Form on Submit

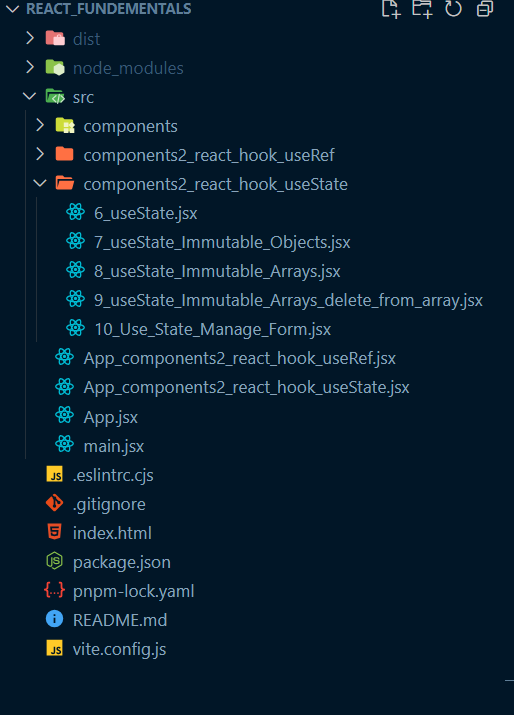


**Compile Result:**

****

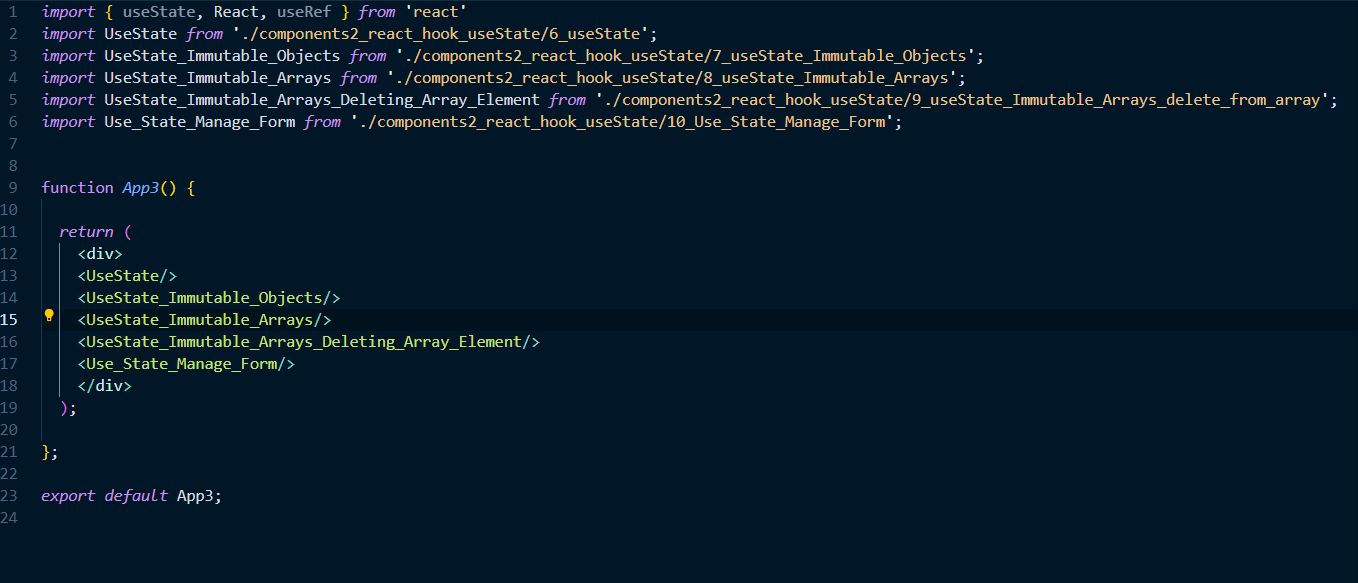


### File structure of useRef Section:

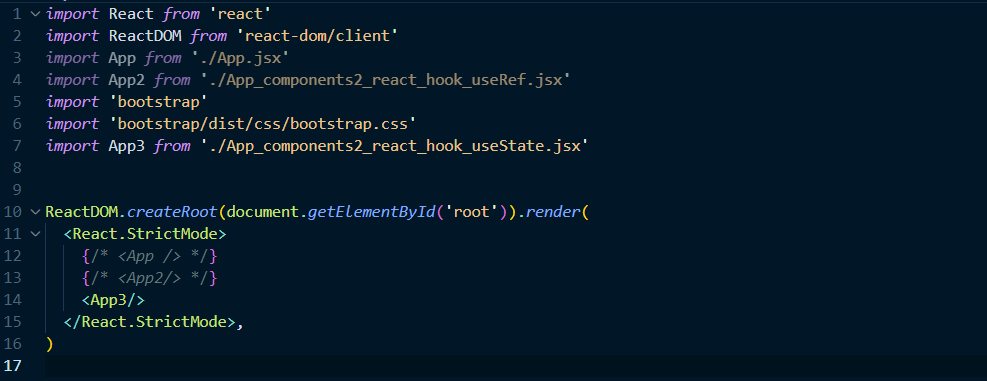


### App.jsx of this chapter:

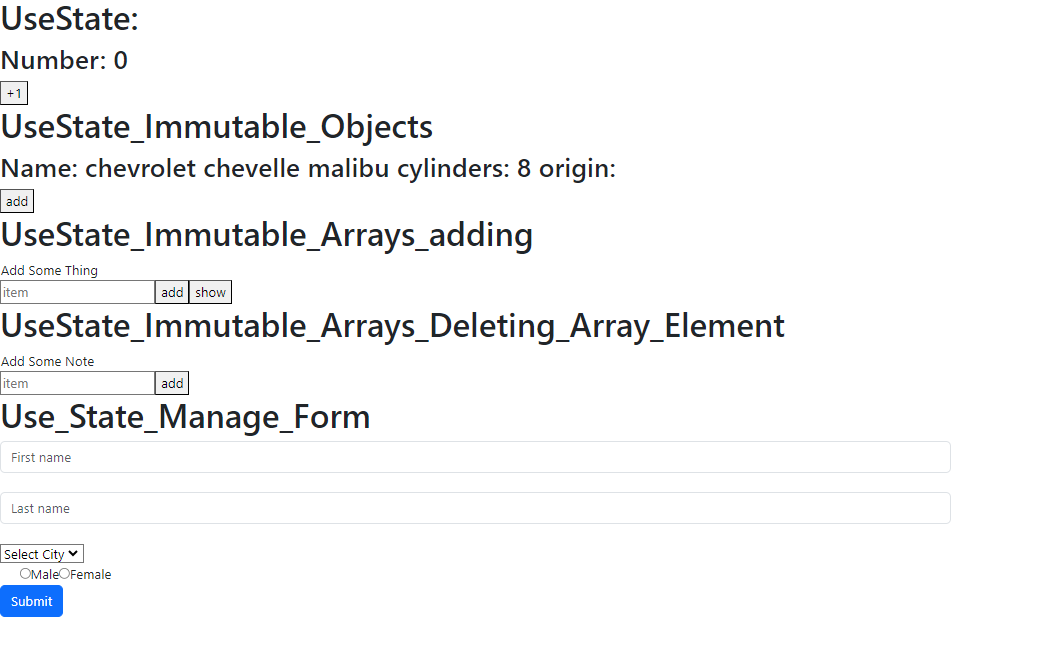
App.jsx of this chapter is called `App\_components2\_react\_hook\_useState`



### Main.jsx of this chapter:



### Output Result of this chapter:



## useEffect

Synchronizing with Effects[[1]](#footnote-1)

* Some components need to synchronize with external systems. For example, you might want to control a non-React component based on the React state, set up a server connection, or send an analytics log when a component appears on the screen. Effects let you run some code after rendering so that you can synchronize your component with some system outside of React.
* Sometimes this isn’t enough. Consider a ChatRoom component that must connect to the chat server whenever it’s visible on the screen. Connecting to a server is not a pure calculation (it’s a side effect) so it can’t happen during rendering. However, there is no single particular event like a click that causes ChatRoom to be displayed.
* Effects let you specify side effects that are caused by rendering itself, rather than by a particular event. Sending a message in the chat is an event because it is directly caused by the user clicking a specific button. However, setting up a server connection is an Effect because it should happen no matter which interaction caused the component to appear. Effects run at the end of a commit after the screen updates. This is a good time to synchronize the React components with some external system (like network or a third-party library).
* Don’t rush to add Effects to your components. Keep in mind that Effects are typically used to “step out” of your React code and synchronize with some external system. This includes browser APIs, third-party widgets, network, and so on. If your Effect only adjusts some state based on other state, you might not need an Effect.

1. https://react.dev/learn/synchronizing-with-effects [↑](#footnote-ref-1)