Change Port Number

You can change the port number for your React application using a few methods:

.env (Recommended for Create React App):

- Create a **.env** file in your project root.
- **Add PORT=3001** (or your desired port) to this file.
- Create React App automatically picks this up when you run npm start.

cross-env (For package.json scripts):

- Install cross-env as a dev dependency (**npm install cross-env**).
- Modify your start script in **package.json** like: "**start**": "**cross-env PORT=3001 reactscripts start**".
- cross-env is a package that allows you to set environment variables uniformly across different operating systems (Windows, macOS, Linux) within your package.json scripts.

set (Windows only):

- **update start script in package.json**
- "**start**": "**set PORT=4001 && react-scripts start**",
- This solution using set is only for Windows. It will not work on macOS or Linux. If you need a cross-platform solution, you should use cross-env as mentioned previously.

Virtual DOM

When something on the web page needs to change, React first updates this **Virtual DOM**. Then, it efficiently compares the updated Virtual DOM with its previous version to find out exactly what changed. Finally, it applies **only the minimal necessary changes** to the **Real DOM** that the user sees in the browser.

This process makes updating web pages much faster and smoother than directly manipulating the complex Real DOM every time something changes.

Imagine an IPL scoreboard page.

1. **Initial View:** You see the **Navbar**, **Footer**, Team A's score, current batsman's name, and their score. React has a complete (**Virtual DOM**) of this.
2. **Batsman Hits a Four:**
   - React gets the update: "batsman's score +4, Team score +4."
   - It quickly makes a **new Virtual DOM** reflecting these *only* these two changes.
   - React compares the new **Virtual DOM** to the old one. It instantly sees that only batsman's individual score and the team's total score numbers have changed.

3. **Screen Update:** React tells the browser: "Just update batsman's score and the team's total. Leave everything else alone."

The scores update **instantly and smoothly** on your screen without the whole page reloading.

SPA

In a React Single Page Application (SPA), smooth navigation relies on the **Virtual DOM** and the <Link> component from react-router-dom.

How it works for your website with a consistent navbar and footer:

- **Initial Load:** React builds the entire UI (navbar, Home content, footer) as a **Virtual DOM** blueprint. The browser then draws the **Real DOM** from this blueprint.
- **Navigation (e.g., clicking "About"):**
    - You **must use <Link to="/about">**, not <a>. <Link> prevents a full page reload.
    - react-router-dom updates the URL, triggering React.
    - React creates a **NEW Virtual DOM** for the "About" page.
    - React's "diffing" algorithm compares the new "About" Virtual DOM with the old "Home" one. It sees the navbar and footer are identical, but the content has changed.
    - React then tells the browser to **only update the changed part of the Real DOM** (just the content area).

Add images in your component

Image is in public/ folder

Place the image in public/, for example: public/bird.png

Usage in JSX:

```
<img src="/images/bird.png" alt="Bird" />
```

Do **not** use import for public files

Use **relative path from public root** (starts with /)

Using Images from the src/ Folder

Place the image in src/assets/, for example: src/assets/bird.png

Usage in JSX (must import):

```
import birdImg from './assets/bird.png';
<img src={birdImg} alt="Bird" />
```

This goes through Webpack bundling

Useful for React components and when you want to optimize images

export default

- **Only one default export allowed per file.**
- Used when the module exports a single main thing (function, class, object, etc.).
- When importing, you **do NOT use curly braces**.
- You can rename it freely when importing.

export (Named Export)

- You can have **multiple named exports per file**.
- Each export has its own name.
- When importing, you **MUST use curly braces** and the exact exported name.
- You can rename during import using as.

Export	Code Example	Import Syntax
Named Export (Arrow Func)	export const **MyComponent** = () => {}	import **{ MyComponent }** from ...
Named Export (Regular Func)	export function **MyComponent**() {}	import **{ MyComponent }** from ...
Default Export (Arrow Func)	const **MyComponent** = () => {}; **export default MyComponent** JavaScript syntax does **not** allow combining export default with const (or let/var) in a single statement.	import **MyComponent** from ...
Default Export (Regular Func)	**export default** function **MyComponent**() {}	import **MyComponent** from ...

The choice between arrow vs regular function is just JavaScript syntax preference and doesn't affect import/export behavior.

Examples

Named Export (Arrow Function)

MyComponent.js

```
export const MyComponent = () => {
  return <h1>Hello from Named Export!</h1>;
};
```

Myapp.js

```
import { MyComponent } from './MyComponent';

export default function Myapp() {
  return <MyComponent />;
}
```

Named Export (Regular Function)

MyComponent.js

```
export function MyComponent() {
  return <h1>Hello from Regular Named Export!</h1>;
}
```

Myapp.js

```
import { MyComponent } from './MyComponent';
export default function Myapp() {
  return <MyComponent />;
}
```

Default Export (Arrow Function)

MyComponent.js

```
const MyComponent = () => {
  return <h1>Hello from Default Export!</h1>;
};
export default MyComponent;
```

Myapp.js

```
import MyComponent from './MyComponent';
export default function Myapp() {
  return <MyComponent />;
}
```

You can't do: export default const MyComponent = () => {} That's a syntax error.

Default Export (Regular Function)

MyComponent.js

```
export default function MyComponent() {
  return <h1>Hello from Default Export (Regular Function)!</h1>;
}
```

Myapp.js

```
import MyComponent from './MyComponent';
export default function Myapp() {
  return <MyComponent />;
}
```

map() and filter()

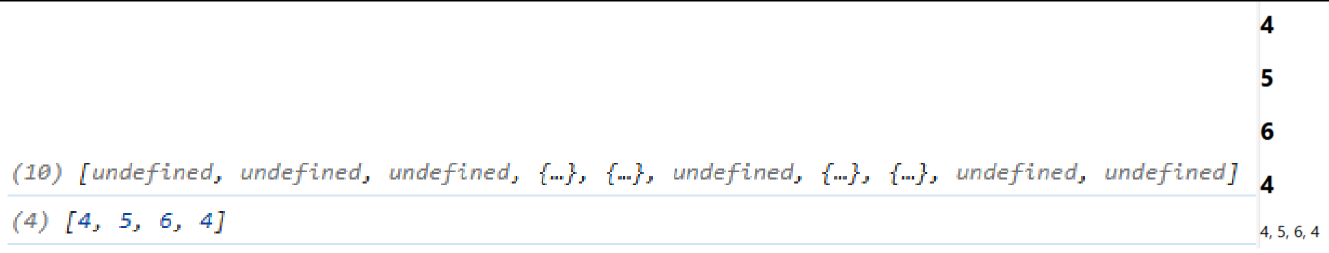
map() always returns a new array with the same number of elements as the original array. **For console**, When val is 5, 6, 4, 4 (from arr), it returns an <h2> JSX element. When val is 1, 2, 3, 3, 3, 1 (from arr), the if (val > 3) condition is false. Since there's no else block or any other return statement, the function implicitly returns undefined.
When React renders <div>{t}</div>, it will render the <h2> elements and **simply ignore (not render anything for)** the undefined values.

The filter() method creates a **new array containing only the elements for which the provided callback function returns a "true" value.**

```
function Task() {
  const arr=[1,2,3,4,5,3,6,4,3,1];
  var t = arr.map((val,i) => {
        if (val > 3) return (
          <h2 key={i}>{val}</h2>
        )
      })
      console.log(t)

   var t1 = arr.filter((val,i) => {
     if (val > 3) return (
       <h2 key={i}>{val}</h2>
     )
    })
    console.log(t1)

    return(
      <>
      <div>{t}</div>
      <p>{t1.join(', ')}</p>
      </>
    )
}
export default Task
```



```
(10) [undefined, undefined, undefined, {…}, {…}, undefined, {…}, {…}, undefined, undefined]
(4) [4, 5, 6, 4]
```

```
4

5

6

4

4, 5, 6, 4
```

The double console output is a feature of **React Strict Mode** in development to help you write robust components. It's not an error in itself, but rather an indication that your component's rendering logic is being run twice.

You remove <React.StrictMode> from your index.js