JSX and JavaScript Concepts Sprint Documentation

---

0. App.tsx — Parent Component

Screenshot Placeholder:

Description

The **App.tsx** file is the parent component that integrates all other components:

- FilterControls for toggling filters
- NumberList for displaying data
- Logger, HoistingDemo, and ConstructorDemo for interactive demonstrations

It manages state (showEven, showDoubled) and uses useMemo to efficiently recalculate filtered/mapped data when toggles change.

Key Code Snippet

```
import React, { useMemo, useState } from "react";
import NumberList, { NumberItem } from "./components/NumberList";
import FilterControls from "./components/FilterControls";
import Logger from "./components/Logger";
import HoistingDemo from "./components/HoistingDemo";
import ConstructorDemo from "./components/ConstructorDemo";

const baseNumbers: NumberItem[] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].map((v) =>
({ value: v }));

export default function App() {
  const [showEven, setShowEven] = useState(false);
  const [showDoubled, setShowDoubled] = useState(false);

  const processed = useMemo(() => {
    let arr = baseNumbers.slice();

    if (showEven) {
      arr = arr.filter((n) => n.value % 2 === 0);
    }

    if (showDoubled) {
      arr = arr.map((n) => ({ value: n.value * 2 }));
    }

    return arr;
  }, [showEven, showDoubled]);

  return (
    <div className="max-w-3xl mx-auto p-6 font-sans">
      <h1 className="text-2xl font-bold mb-4">
        JSX & JavaScript Concepts Sprint — Solution
      </h1>

      <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
        {/* Left side: filters + number list */}
        <div className="rounded-lg border">
          <FilterControls
            showEven={showEven}
            setShowEven={setShowEven}
            showDoubled={showDoubled}
            setShowDoubled={setShowDoubled}
          />
          <NumberList numbers={processed} />
        </div>

        {/* Right side: logger + demos */}
        <div className="rounded-lg border p-4">
          <Logger numbers={processed} />
          <div className="my-4">
            <HoistingDemo />
          </div>
          <div className="my-4">
            <ConstructorDemo />
          </div>
        </div>
      </div>

      <footer className="mt-6 text-sm text-gray-600">
        <div>How to run:</div>
        <ol className="list-decimal pl-6">
          <li>
            Create a new React + TypeScript project using{" "}
            <code>npx create-react-app my-app --template typescript</code>.
          </li>
          <li>
            Add the five components in <code>src/components/</code>.
          </li>
          <li>Replace <code>src/App.tsx</code> with this code.</li>
          <li>Run <code>npm start</code> and open your browser at
<code>http://localhost:3000</code>.</li>
        </ol>
      </footer>
    </div>
  );
}
```

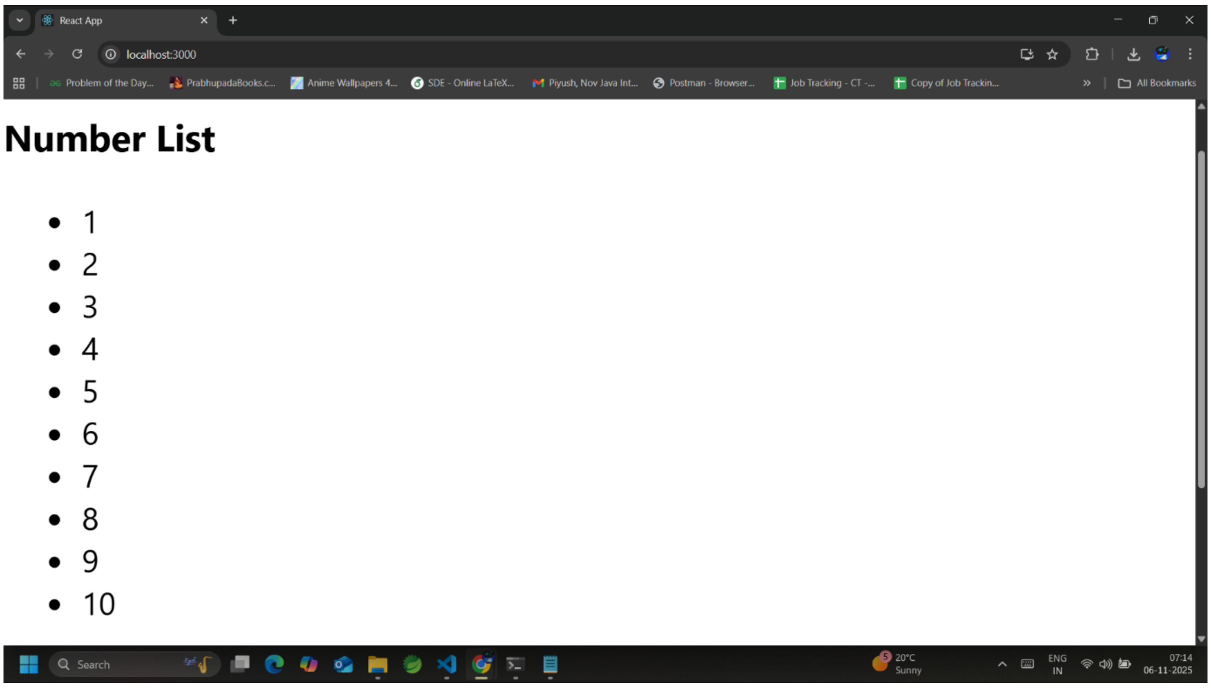
Explanation (2–3 lines)

- Acts as the central hub combining all components.
- Uses **React Hooks (useState, useMemo)** for dynamic UI updates.
- Demonstrates clean state management and functional component composition.

---

1. NumberList Component

Screenshot Placeholder:



Number List

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Description

The **NumberList** component dynamically renders a list of numbers and automatically updates based on the filters applied in the UI. It reflects real-time changes when “even numbers” or “doubled values” filters are toggled.

Key Code Snippet

```
import React from "react";

export interface NumberItem {
    value: number;
}

export default function NumberList({ numbers }: { numbers: NumberItem[] }) {
    return (
        <div className="p-4">
            <h3 className="text-lg font-semibold">Number List</h3>
            <ul className="list-disc pl-6 mt-2">
                {numbers.map((n, idx) => (
                    <li key={idx}>{n.value}</li>
                ))}
            </ul>
        </div>
    );
}
```

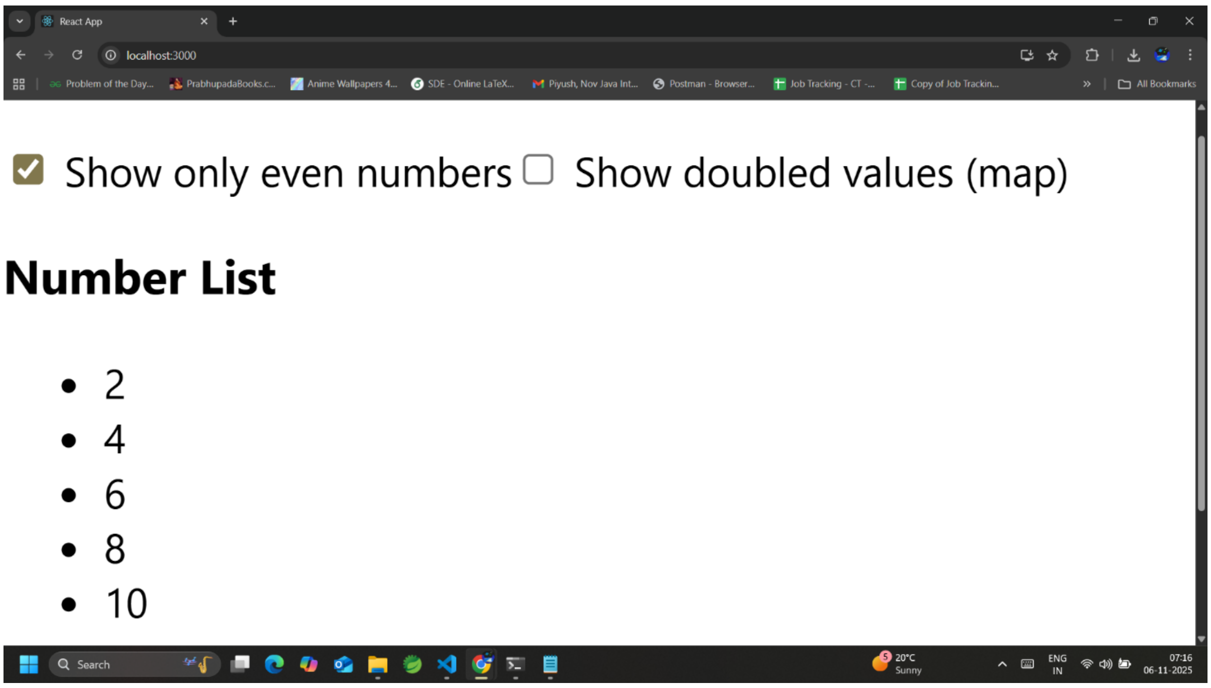
Explanation (2–3 lines)

- This component takes an array of numbers as props.
- It maps through each number and renders it as a list item (<li>).
- React automatically re-renders the list when filters are toggled.

---

2. FilterControls Component

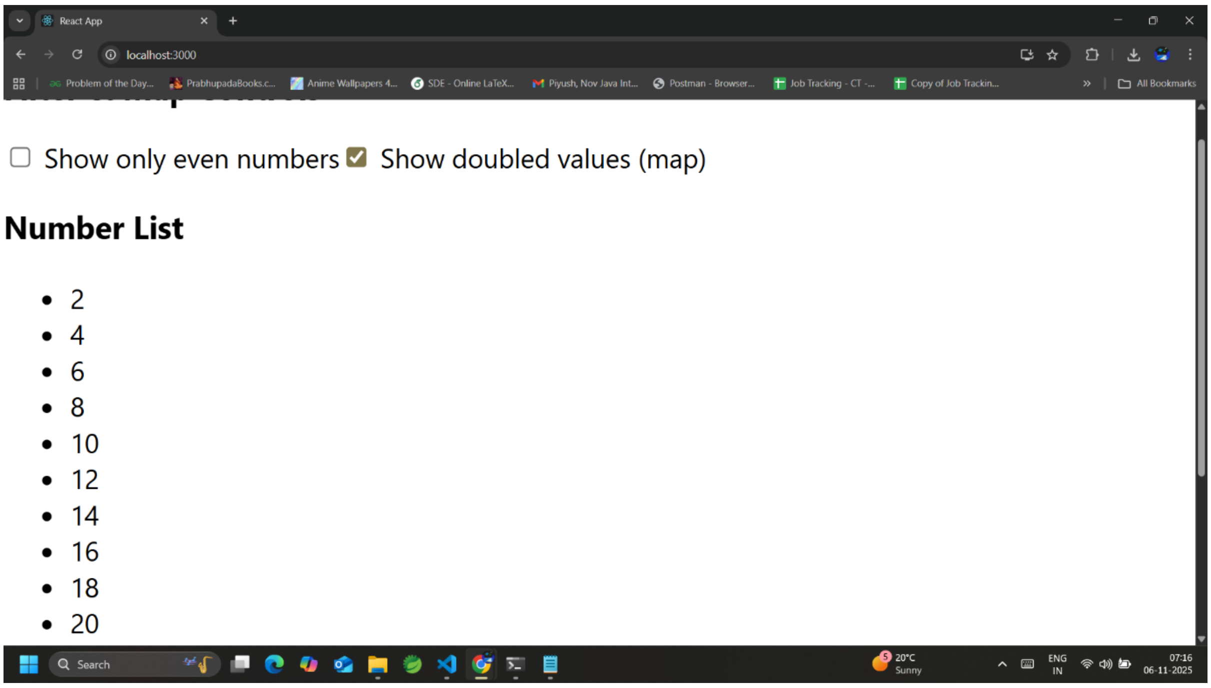
Screenshot Placeholder:



☑ Show only even numbers ☐ Show doubled values (map)

Number List

- 2
- 4
- 6
- 8
- 10



☐ Show only even numbers ☑ Show doubled values (map)

Number List

- 2
- 4
- 6
- 8
- 10
- 12
- 14
- 16
- 18
- 20

Description

The **FilterControls** component provides checkboxes that allow users to toggle filters. Users can choose to display only even numbers or to double the list values before rendering.

Key Code Snippet

```
import React from "react";

interface FilterProps {
    showEven: boolean;
    setShowEven: (b: boolean) => void;
    showDoubled: boolean;
    setShowDoubled: (b: boolean) => void;
}

export default function FilterControls({
    showEven,
    setShowEven,
    showDoubled,
    setShowDoubled,
}: FilterProps) {
    return (
        <div className="p-4">
            <h3 className="text-lg font-semibold">Filter & Map Controls</h3>
            <div className="mt-2 space-x-4">
                <label>
                    <input
                        type="checkbox"
                        checked={showEven}
                        onChange={(e) => setShowEven(e.target.checked)}
                    />{" "}
                    Show only even numbers
                </label>
                <label>
                    <input
                        type="checkbox"
                        checked={showDoubled}
                        onChange={(e) => setShowDoubled(e.target.checked)}
                    />{" "}
                    Show doubled values (map)
                </label>
            </div>
        </div>
    );
}
```

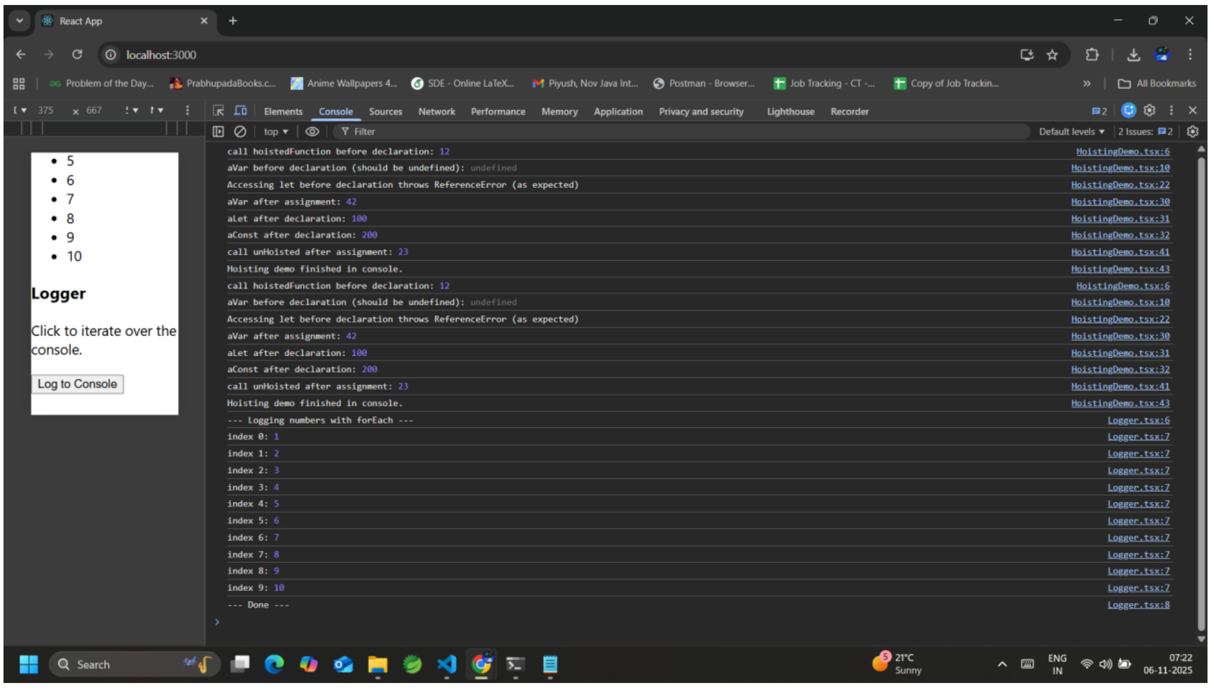
Explanation (2–3 lines)

- Uses controlled checkboxes to toggle state in the parent component.
- Each toggle triggers an update, filtering or transforming the displayed numbers.
- Demonstrates React’s state and event handling.

---

3. Logger Component

Screenshot Placeholder:



- 5
- 6
- 7
- 8
- 9
- 10

Logger

Click to iterate over the console.

Log to Console

```
call hoistedFunction before declaration: 12                         HoistingDemo.tsx:6
aVar before declaration (should be undefined): undefined            HoistingDemo.tsx:10
Accessing let before declaration throws ReferenceError (as expected) HoistingDemo.tsx:22
aVar after assignment: 42                                           HoistingDemo.tsx:30
aLet after declaration: 100                                         HoistingDemo.tsx:31
aConst after declaration: 200                                       HoistingDemo.tsx:32
call unHoisted after assignment: 23                                 HoistingDemo.tsx:41
Hoisting demo finished in console.                                  HoistingDemo.tsx:43
call hoistedFunction before declaration: 12                         HoistingDemo.tsx:6
aVar before declaration (should be undefined): undefined            HoistingDemo.tsx:10
Accessing let before declaration throws ReferenceError (as expected) HoistingDemo.tsx:22
aVar after assignment: 42                                           HoistingDemo.tsx:30
aLet after declaration: 100                                         HoistingDemo.tsx:31
aConst after declaration: 200                                       HoistingDemo.tsx:32
call unHoisted after assignment: 23                                 HoistingDemo.tsx:41
Hoisting demo finished in console.                                  HoistingDemo.tsx:43
--- Logging numbers with forEach ---                                Logger.tsx:6
index 0: 1                                                          Logger.tsx:7
index 1: 2                                                          Logger.tsx:7
index 2: 3                                                          Logger.tsx:7
index 3: 4                                                          Logger.tsx:7
index 4: 5                                                          Logger.tsx:7
index 5: 6                                                          Logger.tsx:7
index 6: 7                                                          Logger.tsx:7
index 7: 8                                                          Logger.tsx:7
index 8: 9                                                          Logger.tsx:7
index 9: 10                                                         Logger.tsx:7
--- Done ---                                                        Logger.tsx:8
```

Description

The **Logger** component demonstrates array iteration using JavaScript's forEach. When the "Log to Console" button is clicked, each value from the number list is logged to the browser console.

Key Code Snippet

```
import React from "react";
import { NumberItem } from "./NumberList";

export default function Logger({ numbers }: { numbers: NumberItem[] }) {
    function handleLog() {
        console.log("--- Logging numbers with forEach ---");
        numbers.forEach((n, i) => console.log(`index ${i}:`, n.value));
        console.log("--- Done ---");
    }

    return (
        <div className="p-4">
            <h3 className="text-lg font-semibold">Logger</h3>
            <p className="mt-2">Click to iterate over the list and log values
to the console.</p>
            <button className="mt-2 px-3 py-1 rounded border"
onClick={handleLog}>
                Log to Console
            </button>
        </div>
    );
}
```

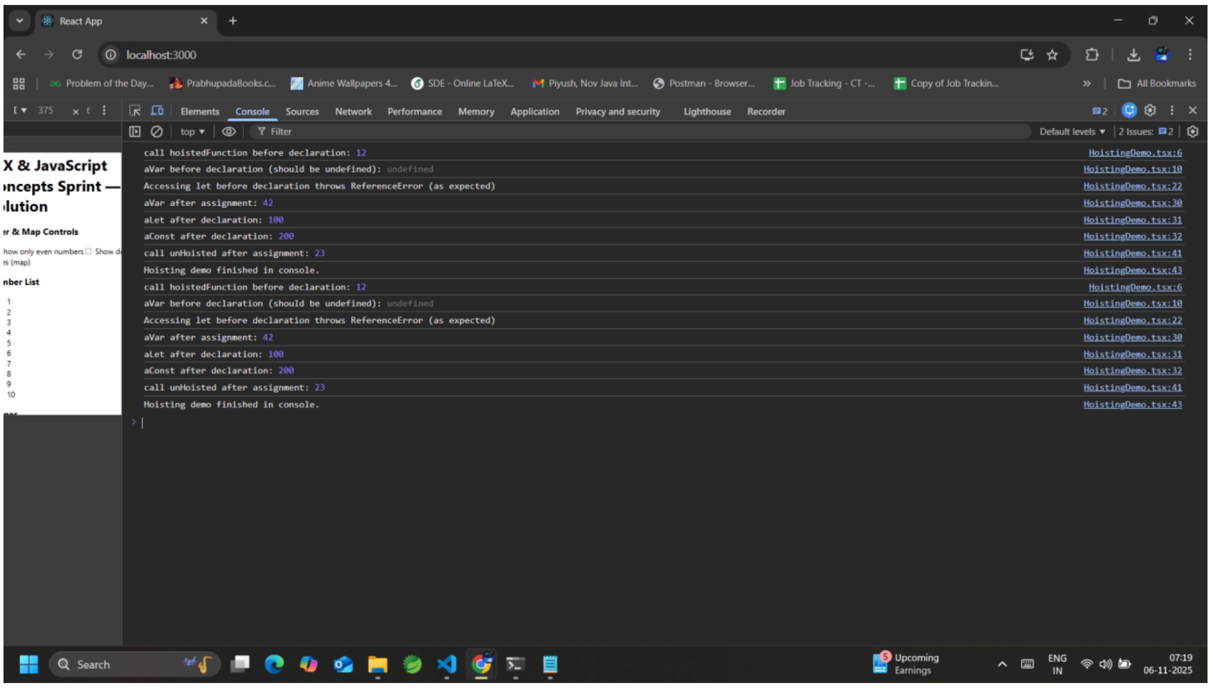
Explanation (2–3 lines)

- Iterates over array elements using forEach.
- Demonstrates event-driven logging in React components.
- Useful for debugging or showing iteration behavior in the console.

---

4. HoistingDemo Component

Screenshot Placeholder:



```
call hoistedFunction before declaration: 12                                    HoistingDemo.tsx:6
aVar before declaration (should be undefined): undefined                        HoistingDemo.tsx:10
Accessing let before declaration throws ReferenceError (as expected)            HoistingDemo.tsx:22
aVar after assignment: 42                                                       HoistingDemo.tsx:30
aLet after declaration: 100                                                     HoistingDemo.tsx:31
aConst after declaration: 200                                                   HoistingDemo.tsx:32
call unHoisted after assignment: 23                                             HoistingDemo.tsx:41
Hoisting demo finished in console.                                              HoistingDemo.tsx:43
call hoistedFunction before declaration: 12                                    HoistingDemo.tsx:6
aVar before declaration (should be undefined): undefined                        HoistingDemo.tsx:10
Accessing let before declaration throws ReferenceError (as expected)            HoistingDemo.tsx:22
aVar after assignment: 42                                                       HoistingDemo.tsx:30
aLet after declaration: 100                                                     HoistingDemo.tsx:31
aConst after declaration: 200                                                   HoistingDemo.tsx:32
call unHoisted after assignment: 23                                             HoistingDemo.tsx:41
Hoisting demo finished in console.                                              HoistingDemo.tsx:43
```

Description

The **HoistingDemo** component illustrates JavaScript hoisting — how function and variable declarations are processed before code execution. It logs examples of var, let, const, and function hoisting in the console.

Key Code Snippet

```
import React, { useEffect } from "react";

export default function HoistingDemo() {
    useEffect(() => {
        // Function hoisting demo
        console.log("call hoistedFunction before declaration:",
hoistedFunction(2));

        // Variable hoisting demo
        // var variables are hoisted (declared but initialized to undefined)
        console.log("aVar before declaration (should be undefined):", (window
as any).aVar);

        // let/const variables are not hoisted in the same way (temporal dead
zone)
        try {
            // Accessing aLet before declaration throws ReferenceError
            // We'll simulate it using an IIFE so it doesn’t crash the whole
script
            (function () {
                // @ts-expect-error intentional TDZ access
                // eslint-disable-next-line no-unused-expressions
                console.log("aLet before declaration:", aLet);
            })();
        } catch {
            console.log("Accessing let before declaration throws
ReferenceError (as expected)");
        }

        // Now declare variables
        var aVar = 42;
        let aLet = 100;
        const aConst = 200;

        console.log("aVar after assignment:", aVar);
        console.log("aLet after declaration:", aLet);
        console.log("aConst after declaration:", aConst);

        // Function declaration (hoisted)
        function hoistedFunction(x: number) {
            return x + 10;
        }

        // Function expression (not hoisted)
        const unHoisted = (x: number) => x + 20;
        console.log("call unHoisted after assignment:", unHoisted(3));

        console.log("Hoisting demo finished in console.");
    }, []);

    return (
        <div className="p-4">
            <h3 className="text-lg font-semibold">Hoisting Demo</h3>
            <p className="mt-2">
                Open the browser console to see the hoisting behavior of
<code>var</code>,{" "}
                <code>let</code>, <code>const</code>, and function
declarations.
            </p>
            <ul className="list-disc pl-6 mt-2">
                <li><b>Function declarations</b> are hoisted (can be called
before they appear).</li>
                <li><b>var</b> is hoisted but initialized as
<code>undefined</code>.</li>
                <li><b>let</b> and <b>const</b> exist in a “temporal dead
zone” until declared.</li>
                <li><b>Function expressions</b> (arrow functions) are not
hoisted.</li>
            </ul>
        </div>
    );
```

```
}
```

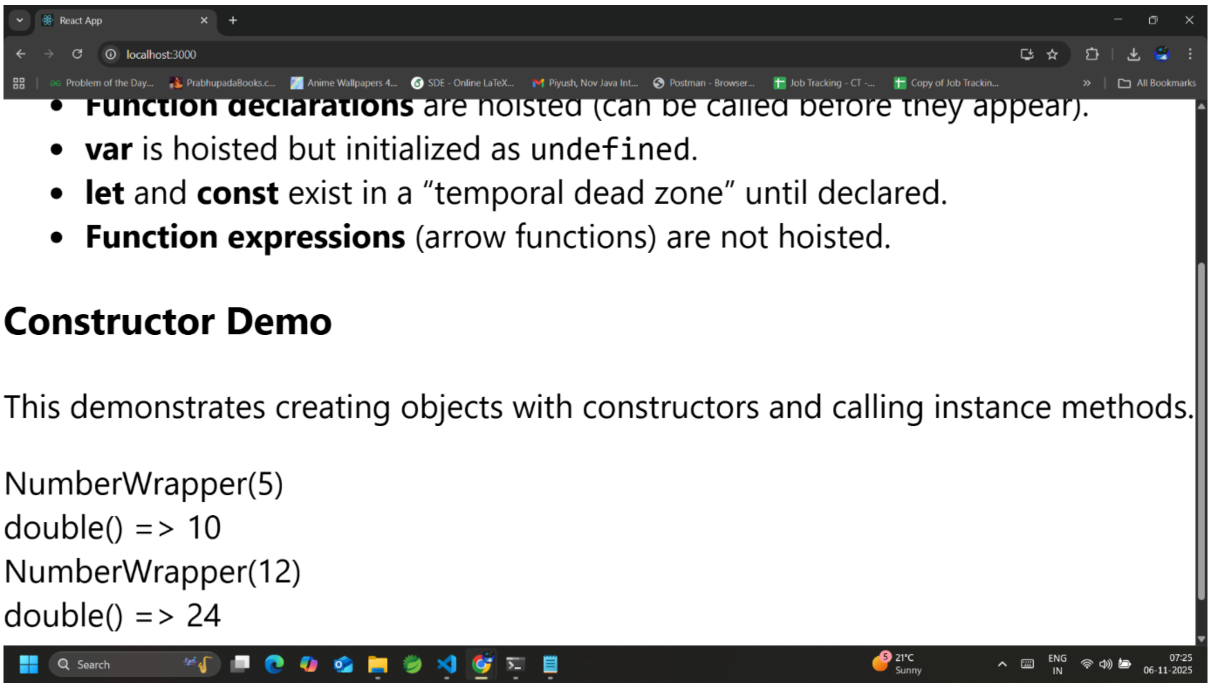
Explanation (2–3 lines)

- Demonstrates the difference between function and variable hoisting.
- Shows that var is hoisted as undefined, while let/const are not accessible before declaration.
- Helps visualize temporal dead zone and function declaration behavior.

---

5. ConstructorDemo Component

Screenshot Placeholder:



- **Function declarations** are hoisted (can be called before they appear).
- **var** is hoisted but initialized as `undefined`.
- **let** and **const** exist in a “temporal dead zone” until declared.
- **Function expressions** (arrow functions) are not hoisted.

Constructor Demo

This demonstrates creating objects with constructors and calling instance methods.

NumberWrapper(5)
double() => 10
NumberWrapper(12)
double() => 24

Description

The **ConstructorDemo** component demonstrates JavaScript classes, constructors, and instance methods. It creates NumberWrapper objects and calls methods like .double() to show class-based logic in React.

Key Code Snippet

```
import React, { useMemo } from "react";

class NumberWrapper {
    value: number;
    constructor(value: number) {
        this.value = value;
    }
    double() {
        return this.value * 2;
    }
    toString() {
        return `NumberWrapper(${this.value})`;
    }
}

export default function ConstructorDemo() {
    const examples = useMemo(() => {
        const a = new NumberWrapper(5);
        const b = new NumberWrapper(12);
        return [a, b];
    }, []);

    return (
        <div className="p-4">
            <h3 className="text-lg font-semibold">Constructor Demo</h3>
            <p className="mt-2">This demonstrates creating objects with
constructors and calling instance methods.</p>
            <div className="mt-2">
                {examples.map((e, i) => (
                    <div key={i} className="mb-2">
                        <div>{e.toString()}</div>
                        <div>double() =&gt; {e.double()}</div>
                    </div>
                ))}
            </div>
        </div>
    );
}
```

Explanation (2–3 lines)

- Defines a NumberWrapper class with constructor and method.
- Creates objects to demonstrate OOP in JavaScript.
- Renders instance data and computed results in JSX.

Conclusion

This project demonstrates key React and JavaScript concepts:

- JSX rendering and props passing.
- State management using React hooks (useState, useMemo).
- Array methods: map, filter, forEach.

- JavaScript fundamentals: hoisting and constructors.