Jit Team™

Why is mutability such a pain in JavaScript?

https://jit.team

front.jit (29.03.2023)

Today's agenda

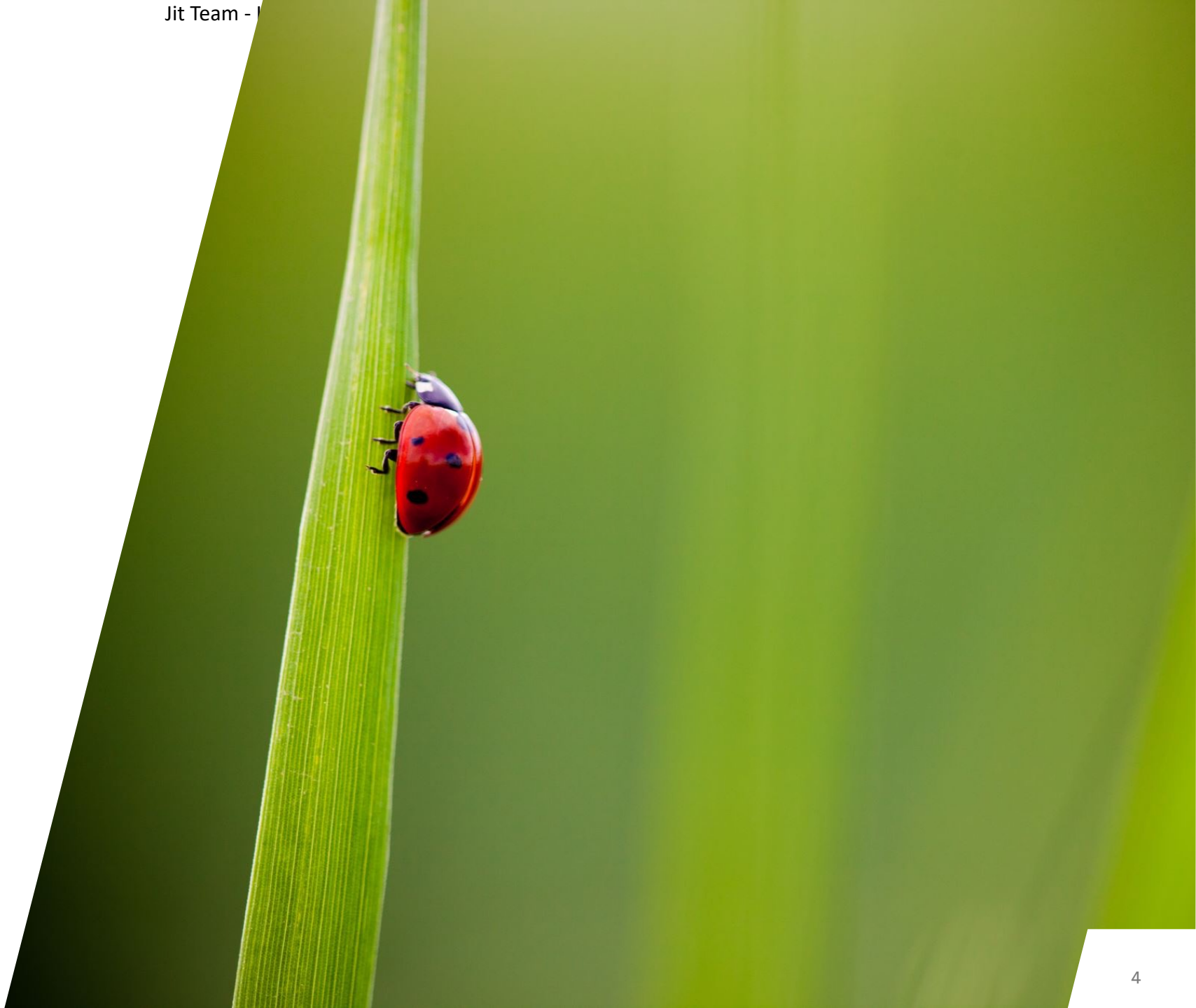
- Motivation and real-life examples
- Pros and cons of immutability
- How to handle immutability in JS

Maciej Kankowski

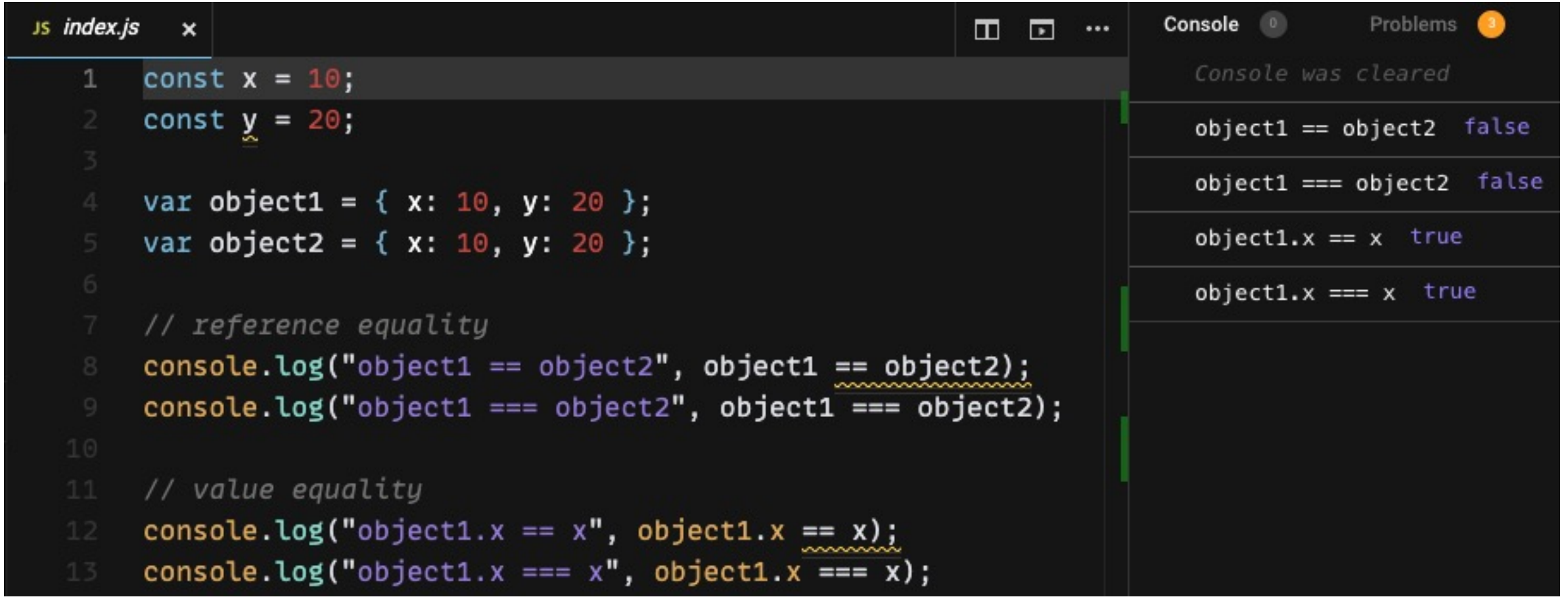
- In Jit Team for 5 years 🤠
- Frontend (React/Next/Vue + 🎨)
- Mobile (🍏)
- Internal projects / mentoring 👨‍🎓 👨‍🎓
- Travel, drones, books, cars 🚀

Motivation

- Bugs and hard debugging
- Discover problems quickly
- Ways to avoid accidental object mutations



Example: Reference vs value equality



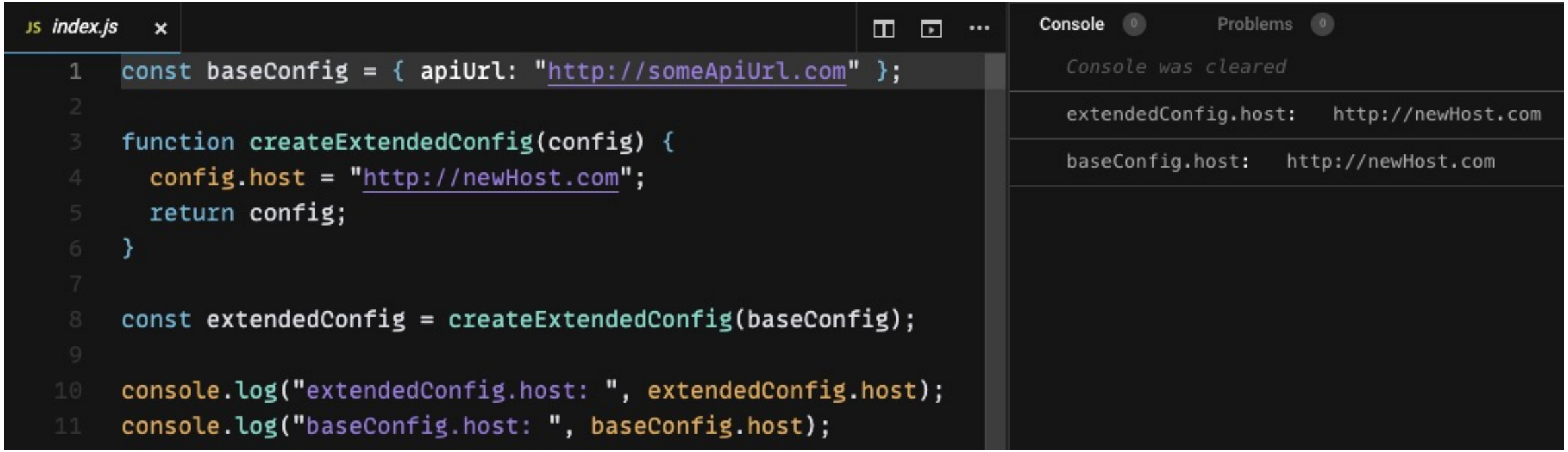
https://codesandbox.io/s/js-reference-vs-value-equality-zb2rdt?file=/src/index.js

Example: Object mutation

```
1  document.querySelector("#app header").innerHTML = `
2  <h1>JS - object mutation</h1>
3  `;
4
5  const car = { manufacture: "VW" };
6  const sameCar = car;
7  const newCar = { ...car };
8
9  sameCar.manufacture = "BMW";
10
11 document.querySelector("#app section").innerHTML = `<p>
12 Car: ${car.manufacture},
13 Same car: ${sameCar.manufacture},
14 New car: ${newCar.manufacture}
15 </p>`;
```

https://codesandbox.io/s/js-object-mutation-vcv37l?file=/src/index.js

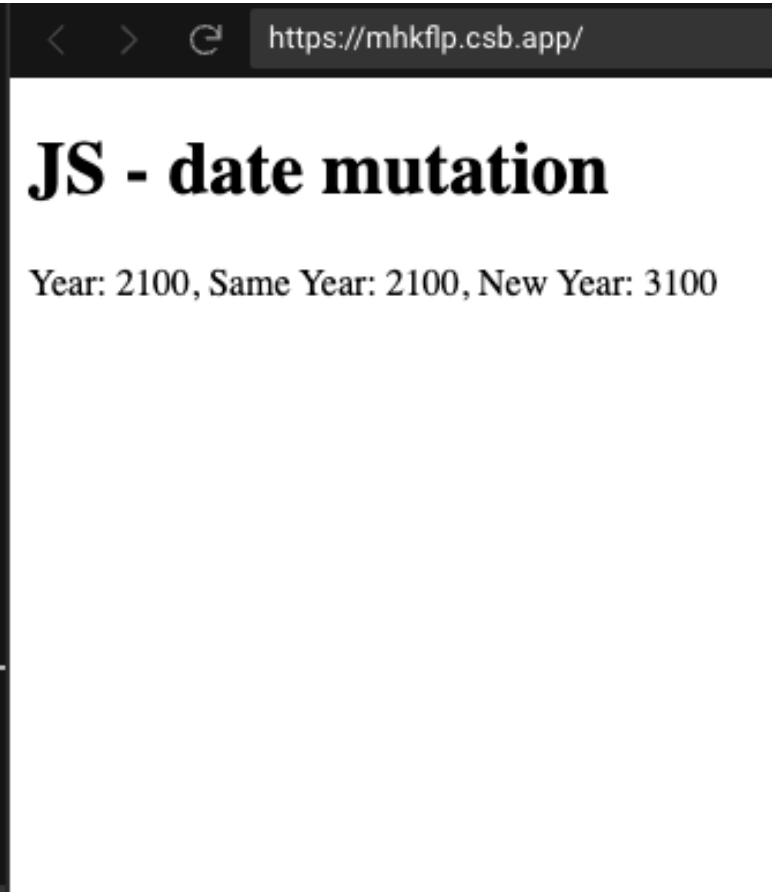
Example: Object mutation (from function)



https://codesandbox.io/s/js-object-mutation-from-function-k2uyen?file=/src/index.js

Example: Date object mutation

```
1  document.querySelector("#app header").innerHTML = `
2  <h1>JS - date mutation</h1>
3  `;
4
5  const date = new Date();
6  const sameDate = date;
7  const newDate = new Date(date);
8
9  sameDate.setFullYear(2100);
10 newDate.setFullYear(3100);
11
12 document.querySelector("#app section").innerHTML = `<p>
13 Year: ${date.getFullYear()},
14 Same Year: ${sameDate.getFullYear()},
15 New Year: ${newDate.getFullYear()}
16 </p>`;
```



https://codesandbox.io/s/js-date-mutation-mhkflp?file=/src/index.js

Pros & cons of immutability

- But before, what is data immutabiliy?
- Simpler programming and debugging
- Immutable data is slower than mutable
- DOM rendering, database quering etc. is even much slower
- Being aware of working with reference and values data types
- Being aware of working with shallow and deep comparision
- It matters in ”reactive” frameworks like React.js, Vue.js...

How to handle immutability in JS

- Reduce the usage of mutative functions like **push**, **unshift** or **splice**
- Use shallow-copying functions like **map**, **filter** or **reduce**
- const vs Object.freeze() (runtime)
- **readonly** - Typescript modificator (compile-time only)
- Immer.js, immutable.js
- React: **useDeepCompare**, **shouldComponentUpdate** (memo)
- Pure Function
- Deep & shallow clone methods...

Shallow clone methods

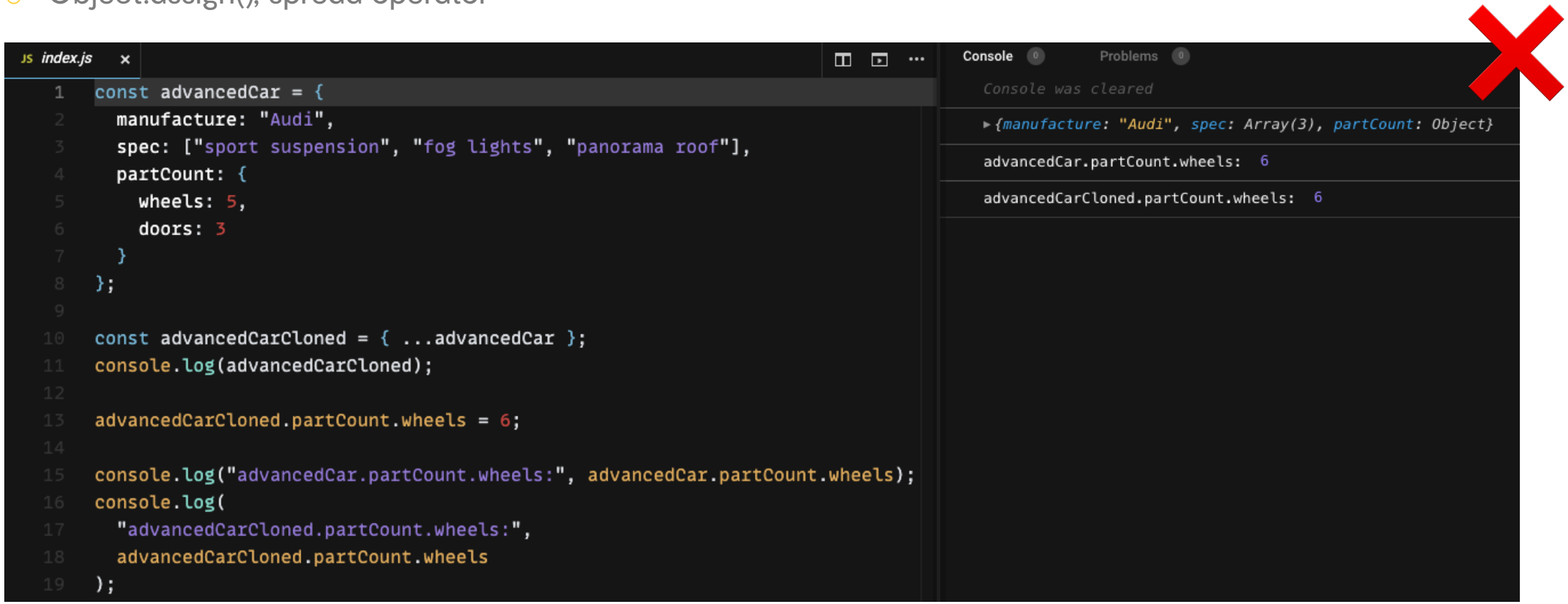
- Object.assign(), spread operator

```
const game = {
  title: "God of War",
}
// works fine since there is no nested object
const gameCopy = {...game}
```

```
const game = {
  title: "God of War",
  date: new Date(2022),
  genre: ["Action", "RPG"]
}

const gameCopy = {...game}

gameCopy.genre.push("Fighting")

gameCopy.date.setTime(2023)

// game.genre will be the same array as gameCopy
// game.date will be modified
```

Shallow clone methods

- Object.assign(), spread operator



https://codesandbox.io/s/js-shallow-merge-ow7jgq?file=/src/index.js

Shallow clone methods: JSON api

- JSON.stringify(), JSON.parse()
- Basic objects, arrays, primitives

```
let ingredients_list = ["noodles", { list: ["eggs", "flour", "water"] }];
let ingredients_list_deepcopy = JSON.parse(JSON.stringify(ingredients_list));

// Change the value of the 'list' property in ingredients_list_deepcopy.
ingredients_list_deepcopy[1].list = ["rice flour", "water"];
// The 'list' property does not change in ingredients_list.
console.log(ingredients_list[1].list);
// Array(3) [ "eggs", "flour", "water" ]
```

⚠️ Functions, Symbols, objects that represent HTML, recursive data... cannot be serialized

Shallow clone methods: JSON api

- JSON.stringify(), JSON.parse()
- Basic objects, arrays, primitives

```
const game = {
  title: "God of War",
  date: new Date(2022),
  genre: ["Action", "RPG"]
}

const gameCopy = Object.assign({}, game)

//  {
//    "title": "God of War",
//    "date": "1970-01-01T00:00:02.022Z",
//    "genre": [
//        "Action",
//        "RPG"
//    ]
//  }
```

⚠️ Functions, Symbols, objects that represent HTML, recursive data... cannot be serialized

Deep clone methods: structuredClone

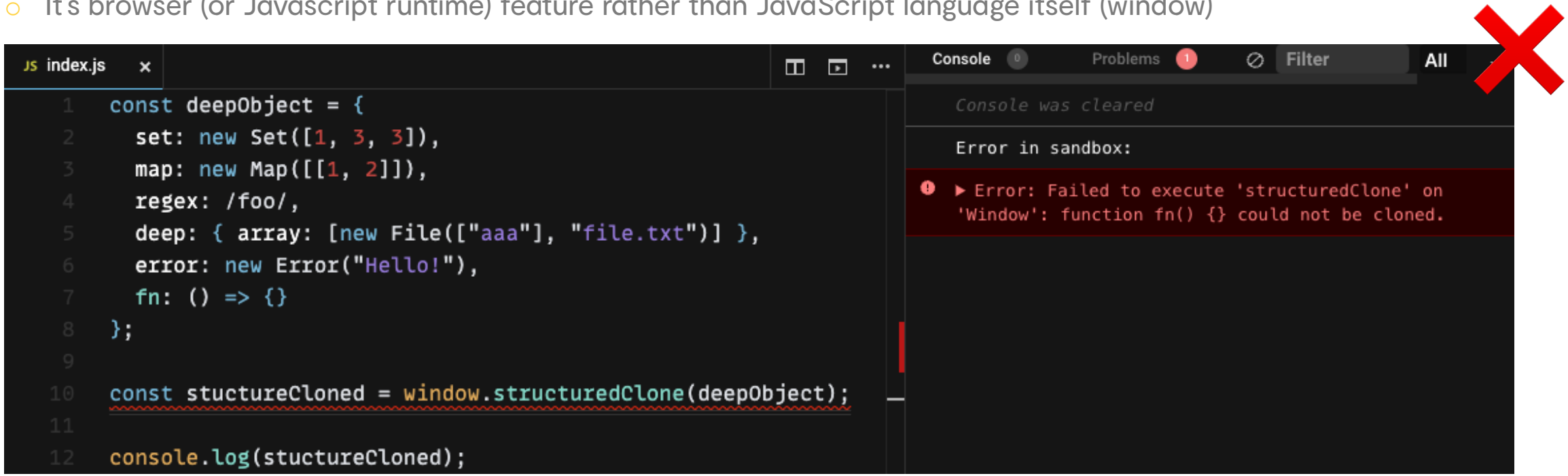
- Original values to be transferred rather than cloned to the new object.
- It's browser (or Javascript runtime) feature rather than JavaScript language itself (window)

```
// Create an object with a value and a circular reference to itself.
const original = { name: "MDN" };
original.itself = original;

// Clone it
const clone = structuredClone(original);

console.assert(clone !== original); // the objects are not the same (not same identity)
console.assert(clone.name === "MDN"); // they do have the same values
console.assert(clone.itself === clone); // and the circular reference is preserved
```

⚠️ The same serialization issues like JSON.stringify()

Deep clone methods: structuredClone

- Original values to be transferred rather than cloned to the new object.
- It's browser (or Javascript runtime) feature rather than JavaScript language itself (window)

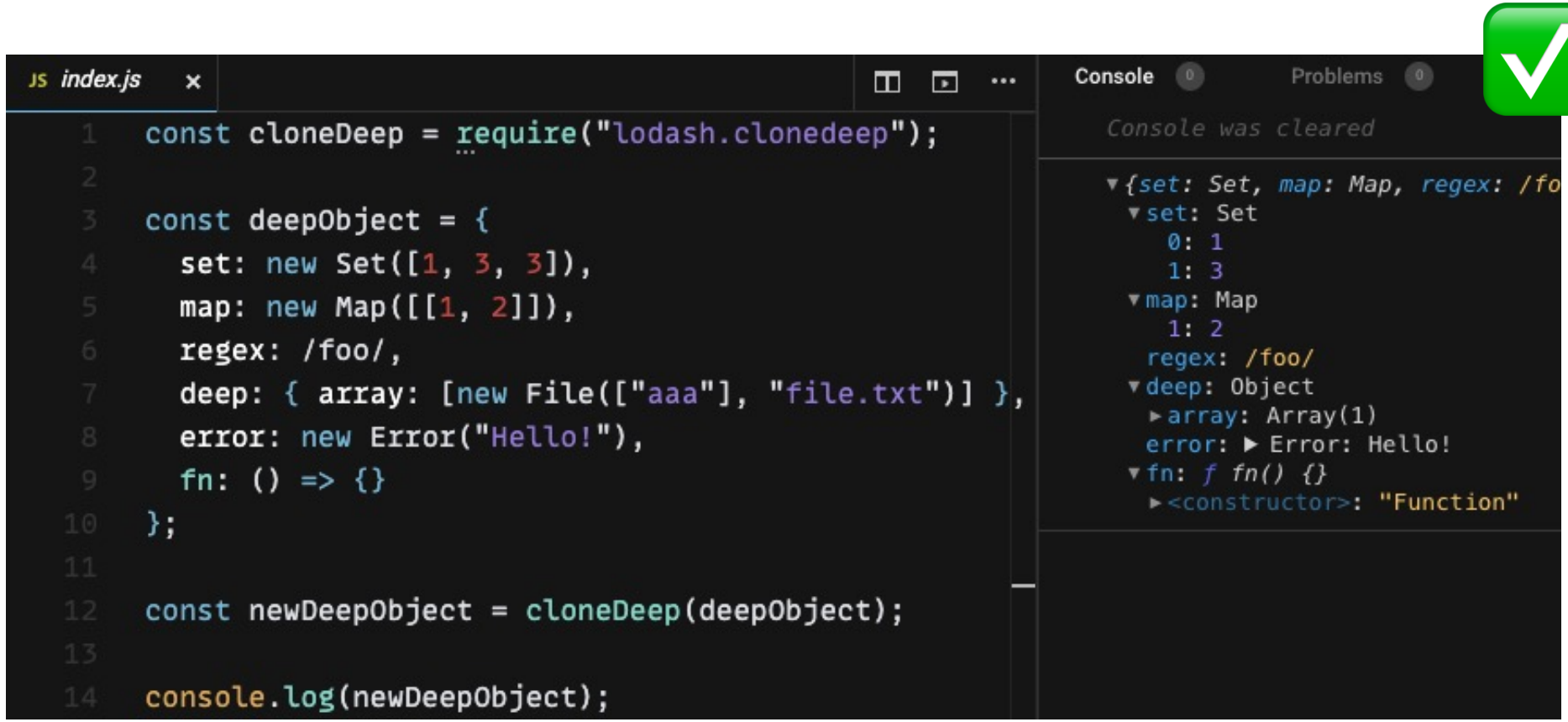


⚠️ The same serialization issues like JSON.stringify()

https://codesandbox.io/s/js-window-structuredclone-xkt0i7

Deep clone methods: external libraries

- Lodash (_.cloneDeep), Ramda
- Structural sharing
- Bundle size



Conclusions

- Bugs & debbuging: Working with immutable data prevents the developers to avoid accidental changs
- Performance impact: DOM renders is much slower than using immutable data
- Shallow (vs. Deep) comparition is a better performant so it’s a default solution for React, Redux etc.
- „Immutable libraries such as Immer can employ structural sharing, which effectively returns a new object that reuses much of the existing object being copied from”

WE WANT YOU
TO CONTRIBUTE
FRONT.JIT

References

- https://developer.mozilla.org/en-US/docs/Glossary/Shallow_copy
- https://developer.mozilla.org/en-US/docs/Glossary/Deep_copy
- https://redux.js.org/faq/immutable-data
- https://daveceddia.com/react-redux-immutability-guide/
- https://reactkungfu.com/2015/08/pros-and-cons-of-using-immutability-with-react-js/
- https://blog.klipse.tech/javascript/2021/02/26/structural-sharing-in-javascript.html
- https://javascript.plainenglish.io/deep-clone-objects-the-right-way-in-javascript-a7ded9d23860
- https://monsterlessons-academy.com/posts/mutable-and-immutable-data-in-javascript
- https://www.builder.io/blog/structured-clone