Fishing with React

Workshop

Martin Farkaš

Michal Gregor

11/1/2019

About Us

UNICORN

With Unicorn's IT solutions and services our customers gain an edge by exploiting the potential of modern IT technologies and innovations to support their business.

- Custom Solutions
- Vertical Solutions
- Skills & Resources

- Banking & Insurance
- Energy & Utilities
- Manufacturing & Trade

More than **300** EUROPEAN INDUSTRY LEADING COMPANIES trust **Unicorn** as their long-term reliable IT partner thanks to its flexibility and capability to deliver.

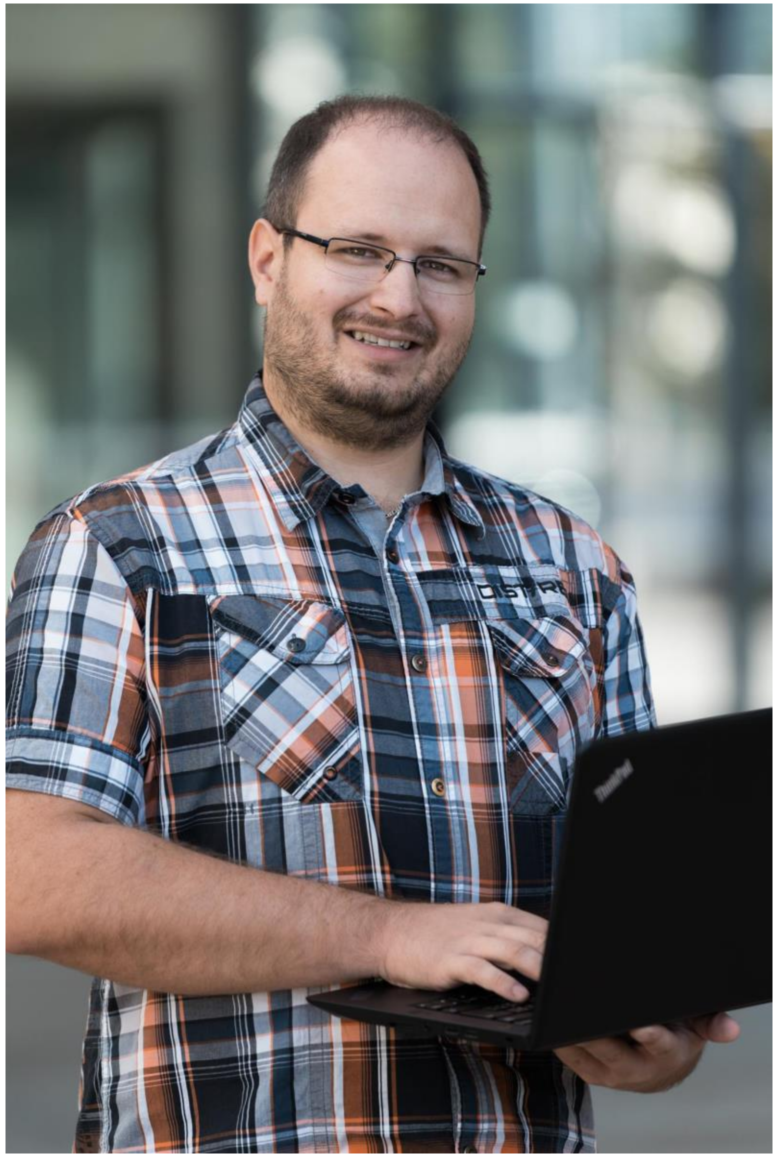
Martin

Software engineer at Unicorn where he has worked on numerous projects in banking, energy and insurance over the last 10 years.

He has recently been working as an evangelist helping developers transition to React, NodeJS, .NET Core and microservice architecture.

He spends his spare time with his two daughters and his wife while attempting to renovate their century-old house.

Michal



Software engineer, full-stack developer and an evangelist at Unicorn.

Always happy to help his co-workers with coding and architectural problems and issues, he dedicates a lot of his time to teaching others through mentoring and coaching activities both at Unicorn and at the University.

When he is not in the IT realm, he enjoys a good film or book, swimming and walking with a camera.

Concept

Workshop is about …

- What the hell are **React Hooks**?
- How can it help me to **improve my app**?
- How to **use it**?

You will need …

- 3 hours of your time
- Notebook + Internet
- **Chrome v77+**
  - https://www.google.com/chrome/
- **Node.js v10.16+**
  - https://nodejs.org
- **VS Code** / **Webstorm** is recommended
  - https://code.visualstudio.com/
  - https://www.jetbrains.com/webstorm/
- To know **JavaScript & Basics of React**
  - https://reactjs.org/tutorial/tutorial.html

Timeline

- Some really necessary theory [30m]
- Environment preparation [15m]
- Exercise 1 – useState [15m]
- Exercise 2 – useContext [15m]
- Exercise 3 – useEffect [15m]

- **Plus4U Coffee break** [15m]

- Exercise 4 - useReducer [25m]
- Exercise 5 - useRef [10m]
- Exercise 6 – useCallback [15m]
- Exercise 7 – Custom Hook [10m]
- Discussion [15m]

Theory

UNICORN

React Component

- Functional Component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- Class Component

```
class Welcome extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}
```

Function vs. Class

- Functional Component
  - Easy to learn
  - Quick to code
  - Quick to render
  - No state
  - No lifecycle
  - Error Boundaries
- Class Component
  - Hard to learn
  - More codes
  - Slower render
  - State
  - Lifecycle
  - Error Boundaries

React Hooks

- The official source says…
  - New addition in React 16.8
  - Let you use **state** and other React features **without** writing **class**
  - Completely **opt-in**
  - 100% **backwards-compatible**
  - **Ready** to use

- In other words…
  - **You don’t need class anymore!**

And what is a Hook?

- A Hook is a **special** function that lets you “**hook** into” React features.

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

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

Under the Hood

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

function DoubleCounter() {
  const [count, setCount] = useState(0);
  const [count2, setCount2] = useState(20);

  return (
    <div>
      <button
        onClick={() => setCount(count + 1)}>
        {count}
        </button>

      <button
        onClick={() => setCount2(count2 + 1)}>
        {count2}
      </button>
    </div>
  );
}
```

Rules of Hooks

1. **Only Call Hooks at the Top Level!**
   - Don’t call Hooks inside loops, conditions, or nested functions.

2. **Only Call Hooks from React Function!**
   - Don’t call Hooks from regular JavaScript functions.
   - Call Hooks from React function components.
   - Call Hooks from custom Hooks

React Hooks – List

- Basic Hooks
  - useState
  - useEfect
  - useContext
- Additional Hooks
  - useReducer
  - useCallback
  - useMemo
  - useRef
  - useImperativeHandle
  - useLayoutEffect
  - useDebugValue
- Custom Hooks
  - You can compose build-in and other custom hooks together!

The game changes…

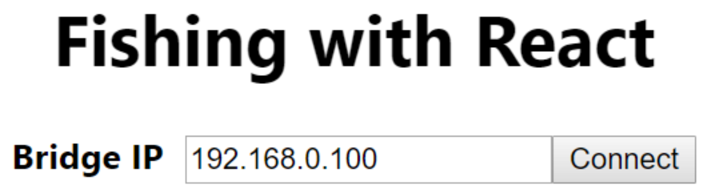
- Functional Component
  - Easy to learn
  - Quick to code
  - Quick to render
  - State
    - Easy to reuse state logic
  - Lifecycle
    - Simple and easy to use
  - Error Boundaries
    - Will be added soon

- Class Component
  - Hard to learn
  - More codes
  - Slower render
  - State
    - Hard to reuse state logic
  - Lifecycle
    - Complex and hard to learn
  - Error Boundaries

Environment

Setup

- Clone or download repository
  - Go to http://bit.ly/2r2PaR9
  - git clone https://github.com/UnicornUniverse/reactiveconf-fishing-with-react.git
- Open root folder in IDE
- Open terminal in IDE
  - cd hooked-on-hooks
  - npm install
  - npm start
- Check app is running on localhost:3000

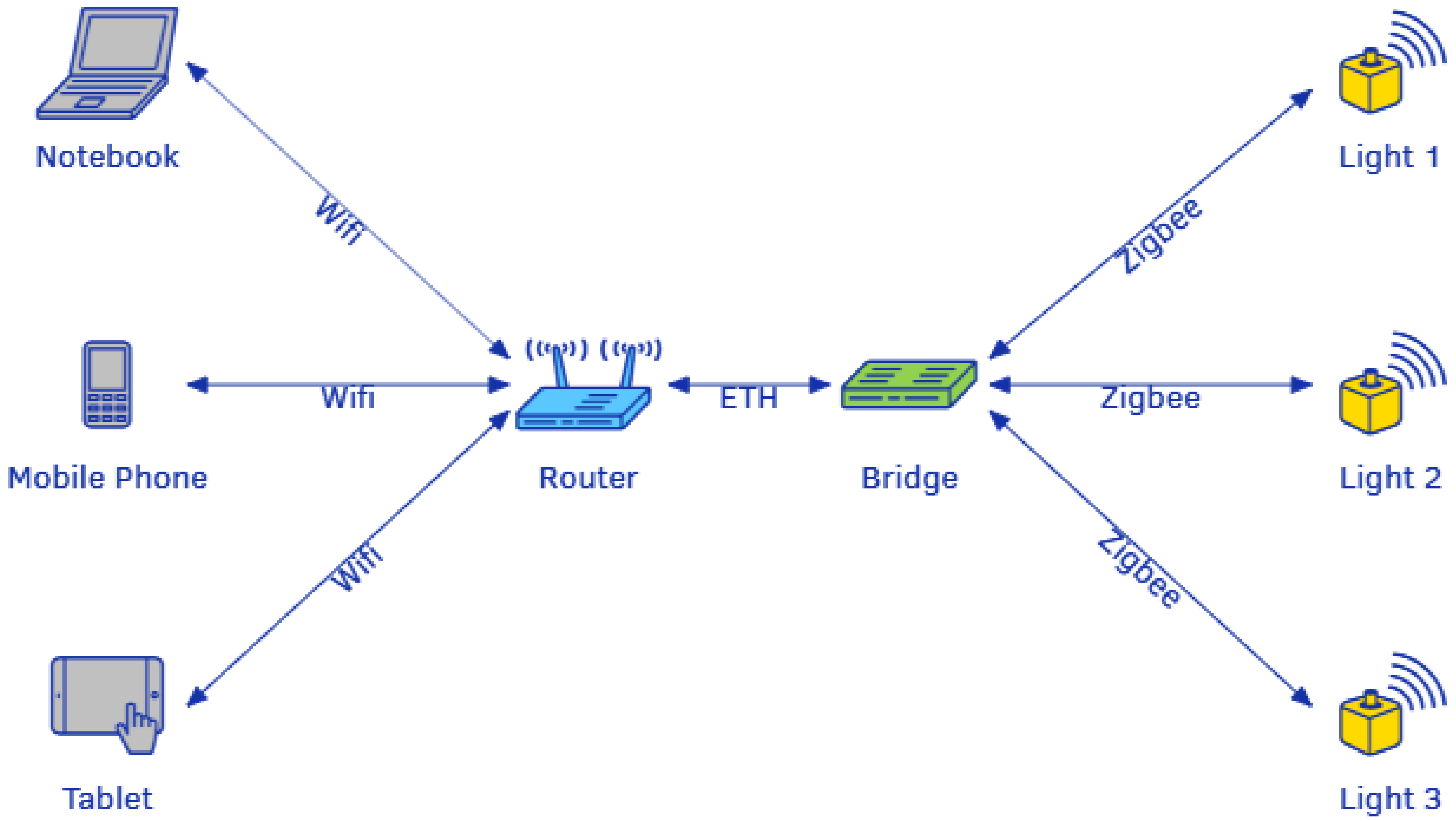


Fishing with React

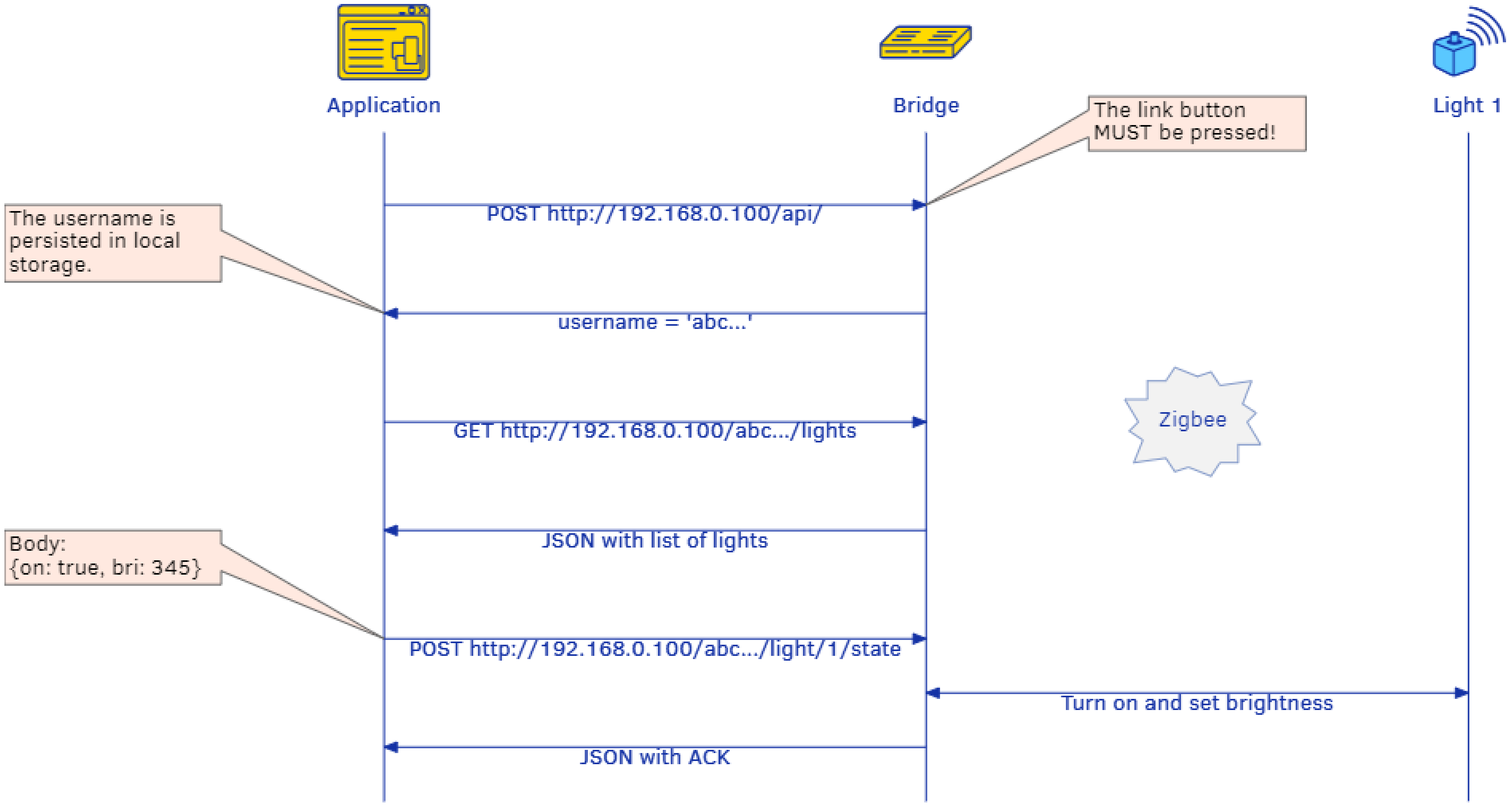
Bridge IP 192.168.0.100

Philips Hue

How it works?


Notebook
Wifi
Mobile Phone
Wifi
Tablet
Wifi
Router
ETH
Bridge
Zigbee
Zigbee
Zigbee
Light 1
Light 2
Light 3

How to use API?

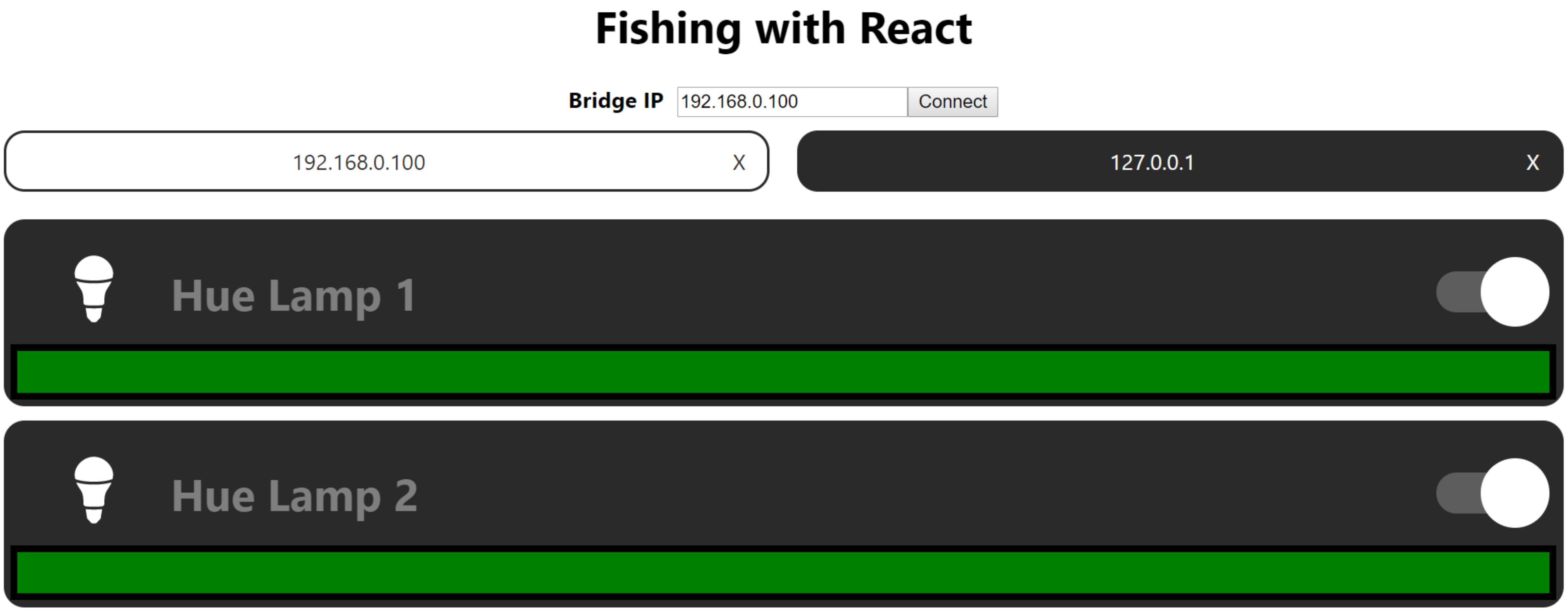


Application
Bridge
Light 1
The link button MUST be pressed!
The username is persisted in local storage.
POST http://192.168.0.100/api/
username = 'abc...'
GET http://192.168.0.100/abc.../lights
Zigbee
JSON with list of lights
Body: {on: true, bri: 345}
POST http://192.168.0.100/abc.../light/1/state
Turn on and set brightness
JSON with ACK

Connect to bridge

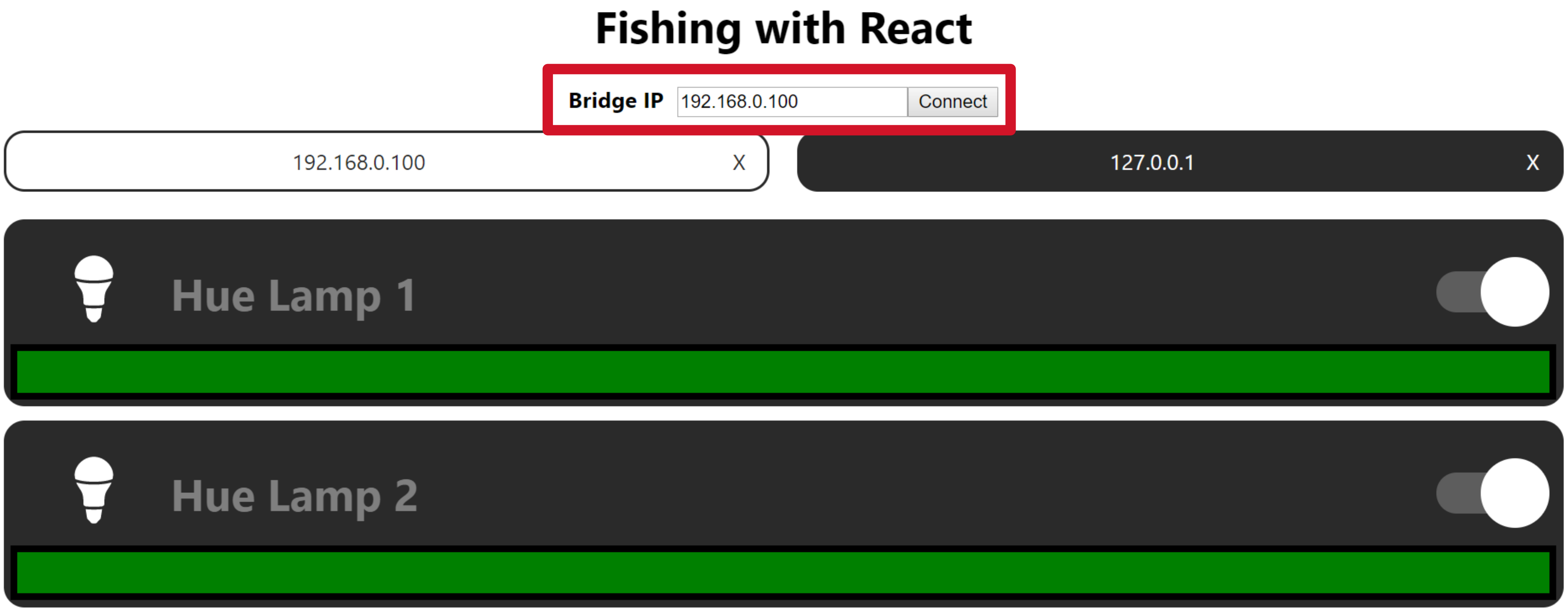
- Connect to Wifi
  - SSID „fishing-with-react“
  - Password is ReactHooks2019
  - No internet, sorry ;)
- Ask Michal Gregor to push link button on bridge.
- Press button Connect

Application



Exercises

Exercise 1 - useState


Fishing with React

Bridge IP 192.168.0.100 Connect

192.168.0.100 X

127.0.0.1 X

Hue Lamp 1

Hue Lamp 2

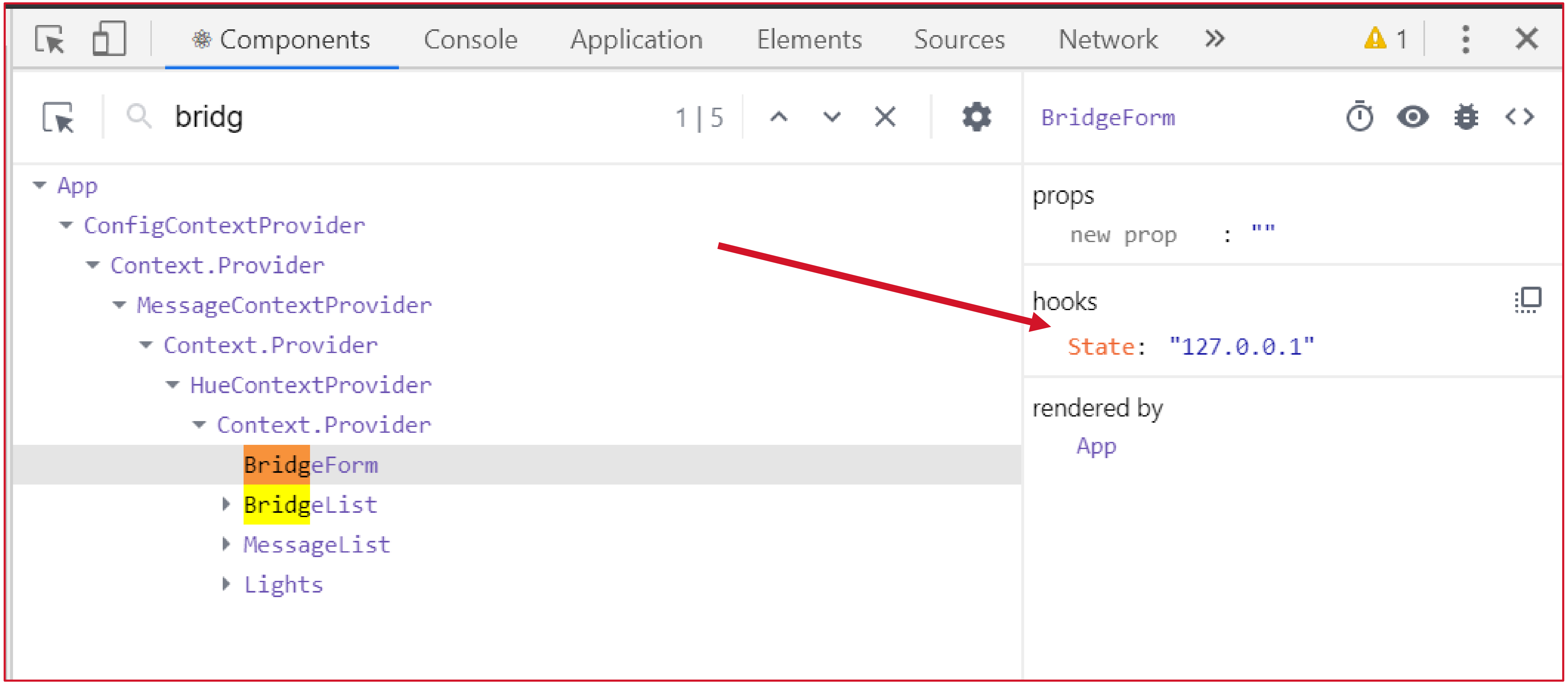
Exercise 1 – useState

```
export default class BridgeForm extends React.Component {
  constructor() {
    super();
    this.state = { bridgeIp: "192.168.0.100" };
    this._handleChange = this._handleChange.bind(this);
    this._handleClick = this._handleClick.bind(this);
  }

  _handleChange(e) {
    this.setState({ bridgeIp: e.target.value });
  }

  _handleClick() {
    alert(this.state.bridgeIp);
  }

  render() {
    return (
      <div>
        <span>Bridge IP</span>
        <input
          type="text"
          value={this.state.bridgeIp}
          onChange={this._handleChange}
        />
        <button onClick={this._handleClick}>Connect</button>
      </div>
    );
  }
}
```

Exercise 1 – useState

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

export default function BridgeForm() {
  const [bridgeIp, setBridgeIp] = useState("127.0.0.1");

  function _handleChange(e) {
    setBridgeIp(e.target.value);
  }

  function _handleClick() {
    alert(bridgeIp);
  }

  return (
    <div>
      <span>Bridge IP</span>
      <input type="text" value={bridgeIp} onChange={_handleChange} />
      <button onClick={_handleClick}>Connect</button>
    </div>
  );
}
```

React Developer Tools



```
▾ App
  ▾ ConfigContextProvider
    ▾ Context.Provider
      ▾ MessageContextProvider
        ▾ Context.Provider
          ▾ HueContextProvider
            ▾ Context.Provider
                BridgeForm
              ▸ BridgeList
              ▸ MessageList
              ▸ Lights
```

```
BridgeForm

props
  new prop : ""

hooks
  State: "127.0.0.1"

rendered by
  App
```

useState

- It does **not merge** new and previous state!
- You can **split state** to multiple variables.
- If the new state is computed using the previous state, you can pass a **function to setState.**
- If the initial state is the result of an expensive computation, you may provide a **function to useState.**
- If you update a State Hook to the **same value** as the current state, React will bail out **without rendering** the children **or** firing **effects**.
- React uses the **Object.is comparison algorithm**

Exercise 2 – useContext

```
export default class BridgeForm extends React.Component {
  constructor() {
    super();
    this.state = { bridgeIp: "192.168.0.100" };
    this._handleChange = this._handleChange.bind(this);
    this._handleClick = this._handleClick.bind(this);
  }
  _handleChange(e) { this.setState({ bridgeIp: e.target.value });}
  _handleClick() { this._addBridge({ ip: this.state.bridgeIp });}
  render() {
    return (
      <HueContext.Consumer>
        {({ addBridge }) => {
          this._addBridge = addBridge;
          return (
            <div>
              <span css={labelCss}>Bridge IP</span>
              <input
                type="text"
                value={this.state.bridgeIp}
                onChange={this._handleChange}
              />
              <button onClick={this._handleClick}>Connect</button>
            </div>
          );
        }}
      </HueContext.Consumer>
    );
  }
}
```

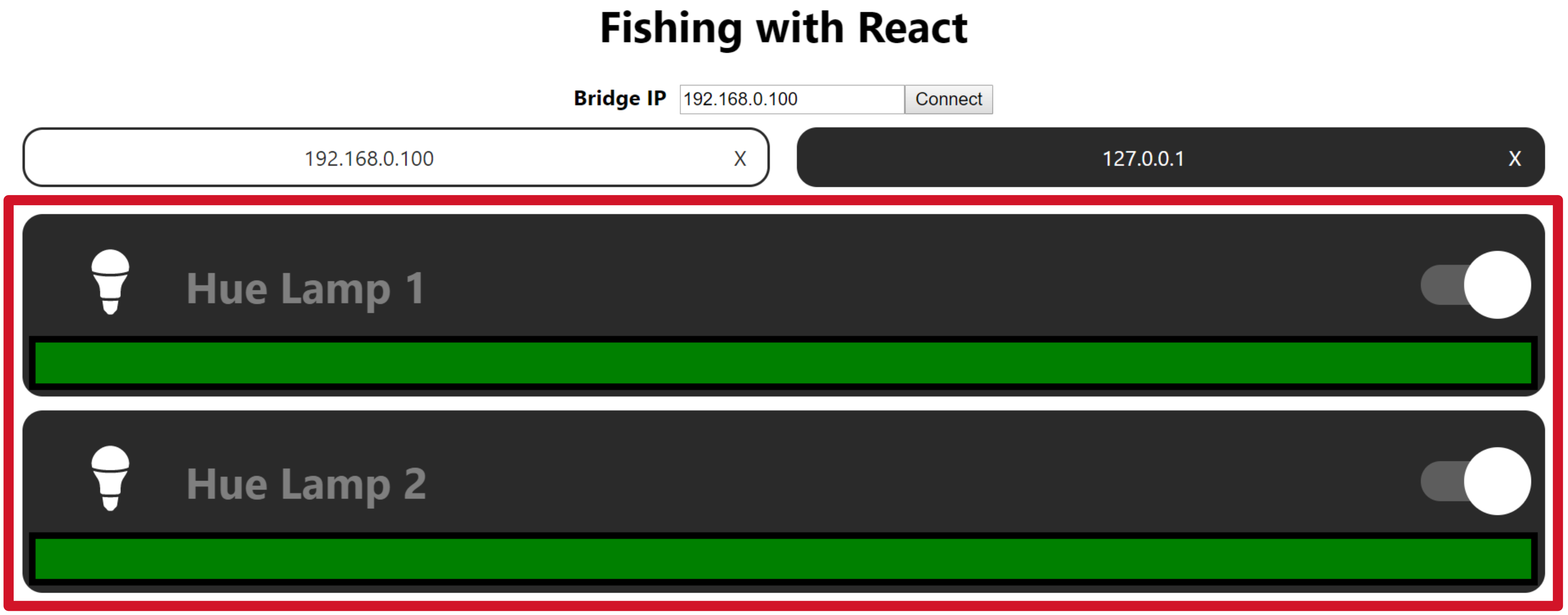
Exercise 2 - useContext

```
export default function BridgeForm() {
  const [bridgeIp, setBridgeIp] = useState("127.0.0.1");
  const { addBridge } = useContext(HueContext);

  function _handleChange(e) {
    setBridgeIp(e.target.value);
  }

  function _handleClick() {
    addBridge({ ip: bridgeIp });
  }

  return (
    <div>
      <input type="text" value={bridgeIp} onChange={_handleChange} />
      <button onClick={_handleClick}>Connect</button>
    </div>
  );
}
```

useContext

- Don’t forget **send context object** to Hook!
  - Correct: useContext(MyContext)
  - Incorrect: useContext(MyContext.Consumer)
  - Incorrect: useContext(MyContext.Provider)
- A component calling useContext will **always re-render** when the context value changes.
- If re-rendering the component is **expensive**, you can optimize it by using **memoization**.

Exercise 3 - useEffect



Exercise 3 - useEffect

```
export default class Lights extends React.Component {
  static contextType = HueContext;

  constructor() {
    super();
    this.state = { lights: [] };
  }

  componentDidMount() {
    if (this.context.status !== "ready") {
      return;
    }

    this.context.user.getLights().then(response => {
      const lights = transform(response);
      this.setState({ lights });
    });
  }

  componentDidUpdate() { /* DO SAME JOB AS MOUNT */  }

  render() {
    return <LightList lights={this.state.lights} />;
  }
}
```

```
{
  "1": {
    "modelid": "LCT001",
    "name": "Hue Lamp 1",
    "state": {
      "sat": 254,
      "bri": 254,
      "hue": 4444,
      "on": true
    },
    "uniqueid": "00:17:88:01"
  },
  "2": {
    "modelid": "LCT001",
    "name": "Hue Lamp 2",
    "state": {
      "sat": 144,
      "bri": 254,
      "hue": 23536,
      "on": true
    },
    "type": "Extended color light",
    "uniqueid": "00:17:88:02"
  }
}
```

Exercise 3 - useEffect

```
function Lights() {
  const hueContext = useContext(HueContext);
  const [lights, setLights] = useState([]);

  useEffect(() => {
    async function loadLights() {
      if (hueContext.status !== "ready") {
        return;
      }

      const response = await hueContext.user.getLights();

      const newLights = transform(response);

      setLights(newLights);
    }

    loadLights();

  }, [hueContext]);

  return <LightList lights={lights} />
}
```

```
{
  "1": {
    "modelid": "LCT001",
    "name": "Hue Lamp 1",
    "state": {
      "sat": 254,
      "bri": 254,
      "hue": 4444,
      "on": true
    },
    "uniqueid": "00:17:88:01"
  },
  "2": {
    "modelid": "LCT001",
    "name": "Hue Lamp 2",
    "state": {
      "sat": 144,
      "bri": 254,
      "hue": 23536,
      "on": true
    },
    "type": "Extended color light",
    "uniqueid": "00:17:88:02"
  }
}
```

Lifecycle mapping

- **constructor**
  - Initialize the state in the useState call
- **getDerivedStateFromProps**
  - While you probably don’t need it!
  - For expensive computation try memoziation
  - Prefer to write full controlled components
- **shouldComponentUpdate**
  - You can wrap a function component with React.memo
- **componentDidMount, componentDidUpdate, componentWillUnmount**
  - The useEffect Hook can express all combinations
- **componentDidCatch and getDerivedStateFromError**
  - The are no Hook equivalents now!
- **render**
  - This is the function component body itself

Plus4U Coffee break

Exercise 4 - useReducer

```
function Lights() {
  const hueContext = useContext(HueContext);
  const [lights, dispatch] = useReducer(LightReducer, []);

  useEffect(() => {
    async function loadLights() {
      if (hueContext.status !== "ready") return;

      const response = await hueContext.user.getLights();

      const newLights = Object.keys(response).map(key => {
        let light = response[key];
        light.id = key;
        return light;
      });

      dispatch({ type: "reset", payload: newLights });
    }

    loadLights();
  }, [hueContext]);
  return <LightList lights={lights} onDispatch={dispatch}  />
}
```

LightReducer

```
export default function LightReducer(lights, action) {
  switch (action.type) {
    case "toggleOn":
      return toggleOn(lights, action.payload);
    case "setBrightness":
      return setBrightness(lights, action.payload);
    case "setHue":
      return setHue(lights, action.payload);
    case "setSaturation":
      return setSaturation(lights, action.payload);
    case "setColor":
      return setColor(lights, action.payload);
    case "reset":
      return action.payload;
    default:
      return lights;
  }
}
```

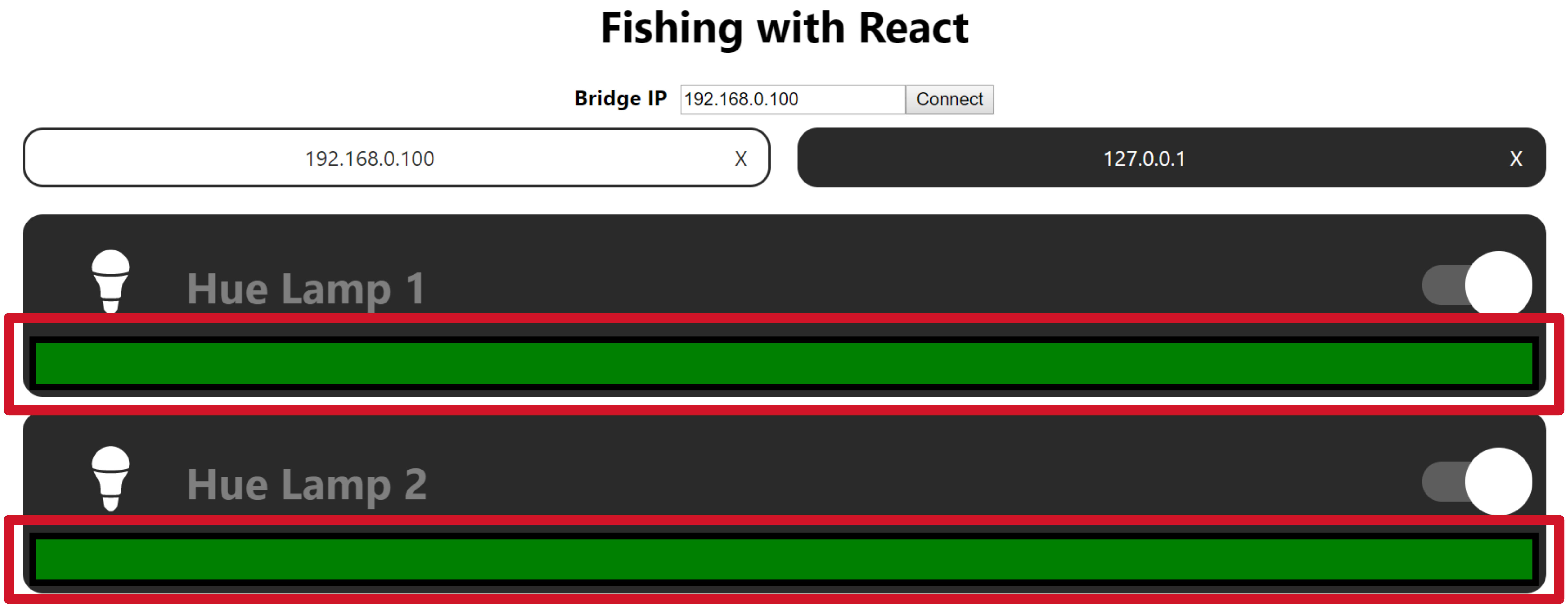
useReducer

- Suitable for
  - Reusable state logic
  - Complex state logic
  - State with multiple sub-values
  - Next state depends on the previous one
  - Optimize performance

How to turn light on? With side effect…

```
function Lights() {
  // ...

  function dispatchWrapper(dispatch) {
    return action => {
      switch (action.type) {
        case "toggleOn":
          hueContext.user.setLightState(action.payload.light.id, {
            on: action.payload.on
          });
          break;
        case "setBrightness":
          hueContext.user.setLightState(action.payload.light.id, {
            bri: action.payload.bri
          });
     }

      dispatch(action);
    };
  }

  return <LightList lights={lights} onDispatch={dispatchWrapper(dispatch)} />;
}
```

Exercise 5 - useRef



Exercise 5 - useRef

```
function LightSlider({ min, max, initValue, onChange }) {
  const ref = useRef(null);
  const [value, setValue] = useState(initValue);
  useEffect(() => { setValue(initValue); }, [initValue]);

  function _handleClick(event) {
    const newWidth = (event.clientX - ref.current.offsetLeft) / ref.current.offsetWidth;
    const newValue = Math.round(newWidth * (max - min) + min);
    setValue(newValue);


    onChange(newValue);
  }

  return (
    <div ref={ref} css={main} onClick={_handleClick} onMouseMove={_handleMouseMove}>
      <div css={css`${slider}; width: ${width}%;`}></div>
    </div>
  );
}
```

Exercise 5 - useRef

```
function LightSlider({ min, max, initValue, onChange }) {
  const ref = useRef(null);
  const isUpdate = useRef(false);
  const [value, setValue] = useState(initValue);
  useEffect(() => { setValue(initValue); }, [initValue]);

  useEffect(() => {
    isUpdate.current = true;
  },[]);

  useEffect(() => {
    if(isUpdate.current) {
      onChange(value);
    }
  }, [value]);

  function _handleClick(event) {  /* COMPUTE newValue */ setValue(newValue); }

  return (/* No changes in render */ );
}
```

useRef

- Not only for component references
- Can hold **any object**
- Similar to instance fields in classes
- **Doesn’t cause a re-render.**

```
function Timer() {
    const intervalRef = useRef();

    useEffect(() => {
      const id = setInterval(() => {
        // ...
      });

      intervalRef.current = id;

      return () => {
        clearInterval(intervalRef.current);
      };
    });

    // ...
}
```

Exercise 6 - useCallback

```
function HueContextProvider({ children }) {
  const _addBridge = useCallback(
     async ({ ip }) => {
      // ...
      dispatch({type: "addBridge",payload: {ip, name: ip,username}});
    },
    [dispatch]
  });

  const _removeBridge = useCallback(
    async bridge => {
      dispatch({type: "removeBridge",payload: {id: bridge.id}});
    },
    [dispatch]
  );

  return (
    <HueContext.Provider value={{...hueContext, addBridge: _addBridge, removeBridge: _removeBridge }} >
      {children}
    </HueContext.Provider>
  );
}
```

useCallback

- Returns a **memoized** callback.
- Only changes if one of the dependencies has changed
- Good for **optimization of reference equality** in children
- Try **useMemo** Hook for memoziation of results from expensive computations

Exercise 7 – Custom Hook

```
export default function useDebounceCallback(callback, delay) {
  const debounceFunction = useRef(callback);
  debounceFunction.current = callback;

  const timeoutHandler = useRef(null);

  const debounceCallback = (...args) => {
    clearTimeout(timeoutHandler.current);

    timeoutHandler.current = setTimeout(() => {
      debounceFunction.current(...args);
    }, delay);
  };

  return [debounceCallback];
}
```

How to use Custom Hook?

```
function LightSlider({ min, max, initValue, onChange }) {
  const [debounceOnChange] = useDebounceCallback(onChange, 100);

  function _handleMouseMove(event) {
    if (event.buttons === 0) return;

    const newWidth =
      (event.clientX - ref.current.offsetLeft) / ref.current.offsetWidth;

    const newValue = Math.round(newWidth * (max - min) + min);

    setValue(newValue);
    debounceOnChange(newValue);
  }

  const width = ((value - min) / (max - min)) * 100;

  return (
    <div ref={ref} css={main} onMouseMove={_handleMouseMove}>
      <div css={css`${slider}; width: ${width}%;`}></div>
    </div>
  );
```

Custom Hooks

- Name should always start with **use**!
- May call other **hooks**
- State and effects inside of it are fully **isolated**.
- Try to resist adding **abstraction too early**!

Discussion

UNICORN

Resources

- Official documentation
  - https://reactjs.org/docs/hooks-intro.html
- Side effects
  - https://gist.github.com/astoilkov/013c513e33fe95fa8846348038d8fe42
- Data fetching
  - https://www.robinwieruch.de/react-hooks-fetch-data