

# React Hooks

Introduksjon til hooks

# Hva er hooks?

- Hooks ble introdusert i React versjon 16.8.0
- Før hooks var funksjonelle komponenter alltid “stateless”
- Legger til “state” og “lifecycle” for funksjonelle komponenter
- Gjør React kode enklere å skrive og enklere å lese.
- “Hvordan brukte man state før hooks?”

# Uten hooks

```
export class NoHooks extends Component {
  constructor(props) {
    super(props);
    this.state = {
      counter: 0,
    };
  }
  render() {
    const { counter } = this.state;
    return (
      <div className="App">
        <header className="App-header">
          The button is pressed: {counter} times.
          <button
            onClick={() => this.setState({ counter: counter + 1 })}
            style={{ padding: "1em 2em", margin: 10 }}
          >
            Click me!
          </button>
        </header>
      </div>
    );
  }
}
```

# Med hooks

```
export function Hooks() {
  const [counter, setCount] = useState(0);

  return (
    <div className="App">
      <header className="App-header">
        The button is pressed: {counter} times.
        <button
          onClick={() => setCount(counter + 1)}
          style={{ padding: "1em 2em", margin: 10 }}
        >
          Click me!
        </button>
      </header>
    </div>
  );
}
```

# Mer om hooks

- Hooks er funksjoner som lar oss “huke på” state og lifecycle m.m i funksjonelle komponenter
- React tilbyr mange hooks:
  - useState → Håndtering av state
  - useEffect → Håndtere lifecycle
  - useContext → Dele data mellom komponenter uten å bruke props
  - useRef -> Brukes ofte for å få tak i HTML elementer fra DOM
  - og mange mange fler...
  - Custom Hook → useMyCustomHook

# **useState og useEffect**

- "Bread & butter" av hooks
- Brukes ofte sammen
- Utgjør 90% av hooks brukt i prosjekter!!!

# useState

# useState

- Brukes for å definere en state som kan leses og endres (get & set)
- Endring av state forårsaker re-rendering av komponenten state brukes i.
- useState returnerer et array
  - første element → nåværende verdi
  - andre element → funksjon for å endre verdien
  - const [get, set] = useState(null)
  - “destructuring syntax” for å lagre array variablene returnert fra useState()

# useState

```
const [value, setValue] = useState(0)
```

- value → get
- setValue → set
- 0 → default verdien
- endre fra 0 til 1 → setValue(1)

# useState

```
const [name, setName] = useState("Captain Hook")
```

- I klasse-komponenter er state alltid et objekt
- useState støtter alle typer inkludert objekter

# useState eksempler

# useState - Counter

```
export function Counter() {
  const [counter, setCount] = useState(0);

  return (
    <>
      The button is pressed: {counter} times.
      <button
        onClick={() => setCount(counter + 1)}
        style={{ padding: "1em 2em", margin: 10 }}
      >
        Click me!
      </button>
    </>
  );
}
```

# useState – Name prompt

```
export function NamePrompt() {
  const [name, setName] = useState("Captain Hook");

  function handleClick() {
    const enteredName = prompt("What is your name?");
    setName(enteredName);
  }

  return (
    <>
      Name in state: {name}
      <button
        onClick={() => handleClick()}
        style={{ padding: "1em 2em", margin: 10 }}
      >
        Click to change name
      </button>
    </>
  );
}
```

# useState – Les mer/mindre

```
export function LessText({ text, maxLength }) {
  const [hidden, setHidden] = useState(true);

  if (text.LessText <= maxLength) {
    return <span>{text}</span>;
  }
  return (
    <span style={{ width: "75vw" }}>
      {hidden ? `${text.substr(0, maxLength).trim()} ...` : text}
      {hidden ? (
        <button onClick={() => setHidden(false)} className="Read-more-btn">
          {" les mer"}
        </button>
      ) : (
        <button onClick={() => setHidden(true)} className="Read-more-btn">
          {" les mindre"}
        </button>
      )}
    </span>
  );
}
```

# useEffect

# useEffect

- Brukes for å håndtere “lifecycle” i funksjonelle komponenter
- Fungerer som erstatning for:
  - componentDidMount → Komponenten instansieres/tas i bruk
  - componentDidUpdate → Komponenten ble oppdatert med nye props
  - componentWillUnmount → Komponenten er i ferd med å fjernes fra komponent-tree

# useEffect

- Skrives slik:

```
useEffect(() => {  
    // kode  
}, [ ])
```

# useEffect - struktur

- ```
useEffect(() => {  
    // code  
    return () => {  
        // cleanup  
    }  
}, []) // dependency array
```
- Tar inn en funksjon, en retur funksjon og et avhengighets array
  - funksjon → kode som skal kjøres
  - retur funksjon → brukes til opprydding
  - avhengighets liste → Når avhengighetene endres kjøres funksjonen på nytt

```
useEffect(() => {  
  // Kjører hver render  
});  
  
useEffect(() => {  
  // Kjører bare på første render  
, []);  
  
useEffect(() => {  
  // Kjører på første render  
  // og hver gang en avhengighets verdi endres  
, [prop, state]);
```

# useEffect

# useEffect – noen klassiske usecaser

- DOM manipulasjon
- Tidtakning
- Datahenting

# useEffect – DOM manipulasjon

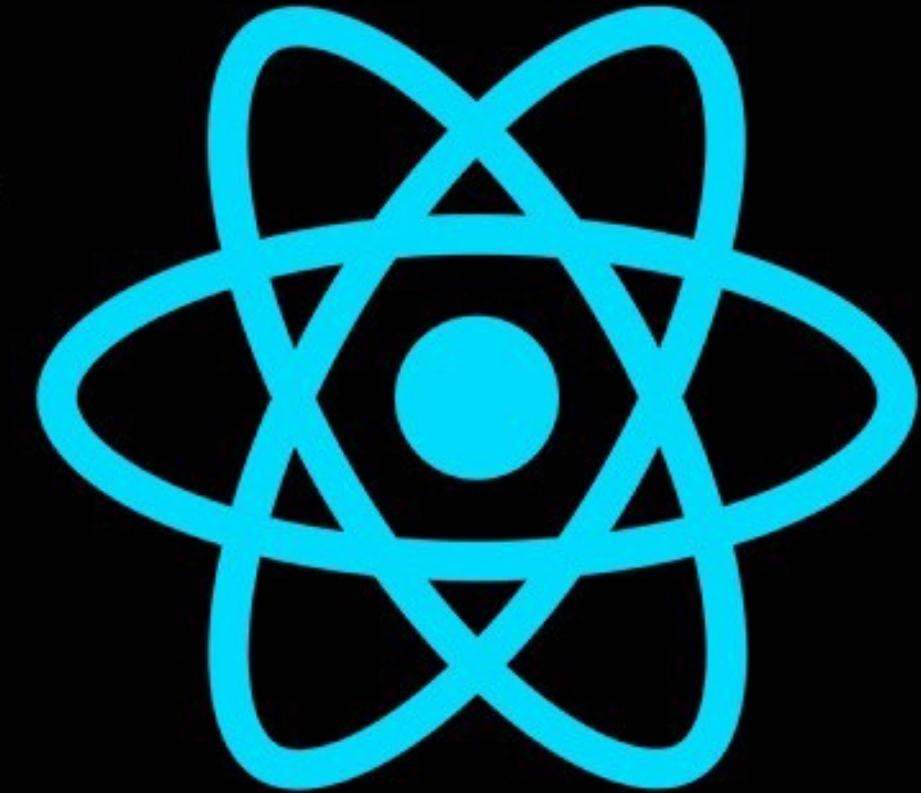
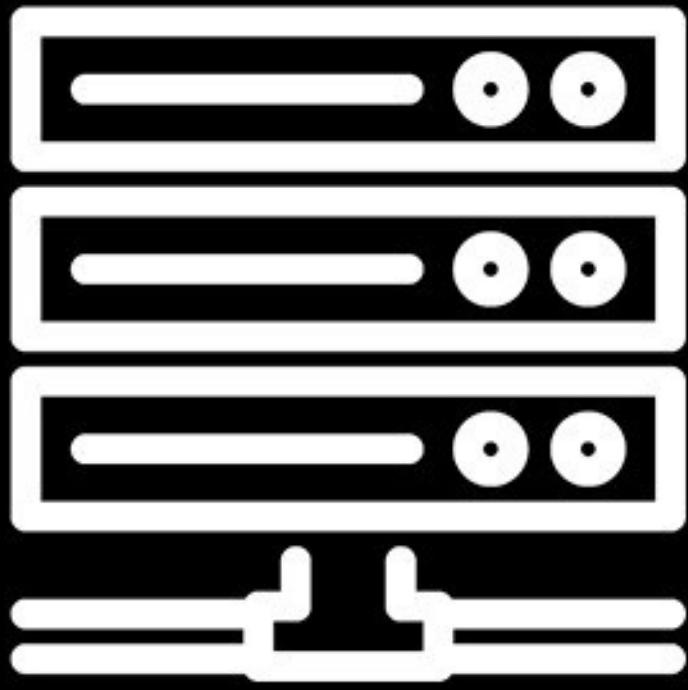
```
useEffect(() => {  
  document.title = `You clicked ${count} times`;  
, [count]);
```

# useEffect - Tidtakning

```
useEffect(() => {
  const timer = setTimeout(() => {
    setSeconds(seconds + 1);
  }, 1000);
  return () => clearTimeout(timer);
}, [seconds]);
```

# useEffect – Datahenting

```
useEffect(() => {
  fetch("https://api.chucknorris.io/jokes/random")
    .then((response) => response.json())
    .then((data) => {
      setChuckNorrisQuote(data.value);
    })
    .catch((e) => console.log("Fetch error: ", e));
}, []);
```



# Datahenting i React

Introduksjon til fetch

# Fetch

- `fetch('url')`
- Lar oss gjøre nettverkskall tilsvarende XMLHttpRequest (XHR)
  - XHR var standarden i årevis
- Hovedforskjellen er at Fetch API bruker promises
  - Enklere og ryddigere
  - Unngår "callback hell"
  - Slipper å huske APIet til XHR

# Fetch vs XHR

```
function reqListener() {
  var data = JSON.parse(this.responseText);
  console.log(data);
}

function reqError(err) {
  console.log("Fetch Error:", err);
}

var oReq = new XMLHttpRequest();
oReq.onload = reqListener;
oReq.onerror = reqError;
oReq.open("get", "https://api.chucknorris.io/jokes/random", true);
oReq.send();
```

# Fetch vs XHR

```
fetch("https://api.chucknorris.io/jokes/random")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((e) => {
    console.log("Fetch Error: ", e);
});
```

# Mer om Fetch()

- Hvorfor er "response.json()" nødvendig?
  - Parser responsen til JSON da det er foretrukket
  - json() returnerer også et promise
  - json(), text(), formData()
  - Mer om Response: <https://developer.mozilla.org/en-US/docs/Web/API/Response>

# Fetch – then, catch, finally

```
export const Get = fetch("https://api.chucknorris.io/jokes/random")
  .then((response) => response.json())
  .catch((error) => console.log("Fetch Error: ", error))
  .finally(() => console.log("Promise resolved"));
```

# Fetch – async, await

```
export const AsyncGet = async () => {
  try {
    let response = await fetch("https://api.chucknorris.io/jokes/random");
    let data = await response.json();
    return data;
  } catch (e) {
    console.log("Fetch Error: ", e);
  } finally {
    console.log("Promise resolved");
  }
};
```

# Fetch – post/put

```
export const Post = (data) => {
  fetch("URL du kan poste til", {
    method: "POST", // eller 'PUT'
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(data),
  })
  .then((response) => response.json())
  .then((data) => {
    console.log("Post Success:", data);
  })
  .catch((error) => {
    console.error("Post Error:", error);
  });
};
```

```
export const AsyncPost = async (data) => {
  try {
    let post = await fetch("URL du kan poste til", {
      method: "POST", // eller 'PUT'
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    });
    let response = await post.json();
    if (response) {
      console.log("Post Success: ", data);
    }
  } catch (error) {
    console.log("Post Error: ", error);
  }
};
```

# Eksempler og oppgaver

- <https://github.com/robertkittilsen/hooks-and-fetch-intro>
- Bare å spørre om dere trenger hjelp eller lurer på noe 😊

# Ressurser for videre lesing

- Hooks:
  - <https://reactjs.org/docs/hooks-reference.html>
  - <https://reactjs.org/docs/hooks-intro.html>
  - <https://reactjs.org/docs/hooks-state.html>
- Fetch:
  - [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)
  - <https://www.freecodecamp.org/news/fetch-data-react/>

Takk for meg!

