



Forms och Server



Agenda



Forms

Sist arbetade vi med refaktorering av en applikation med noteringar

- Bröt ut komponent (Note)
- Flyttade ut komponent till separat fil

```
import { useState } from 'react'
import Note from './components/Note'

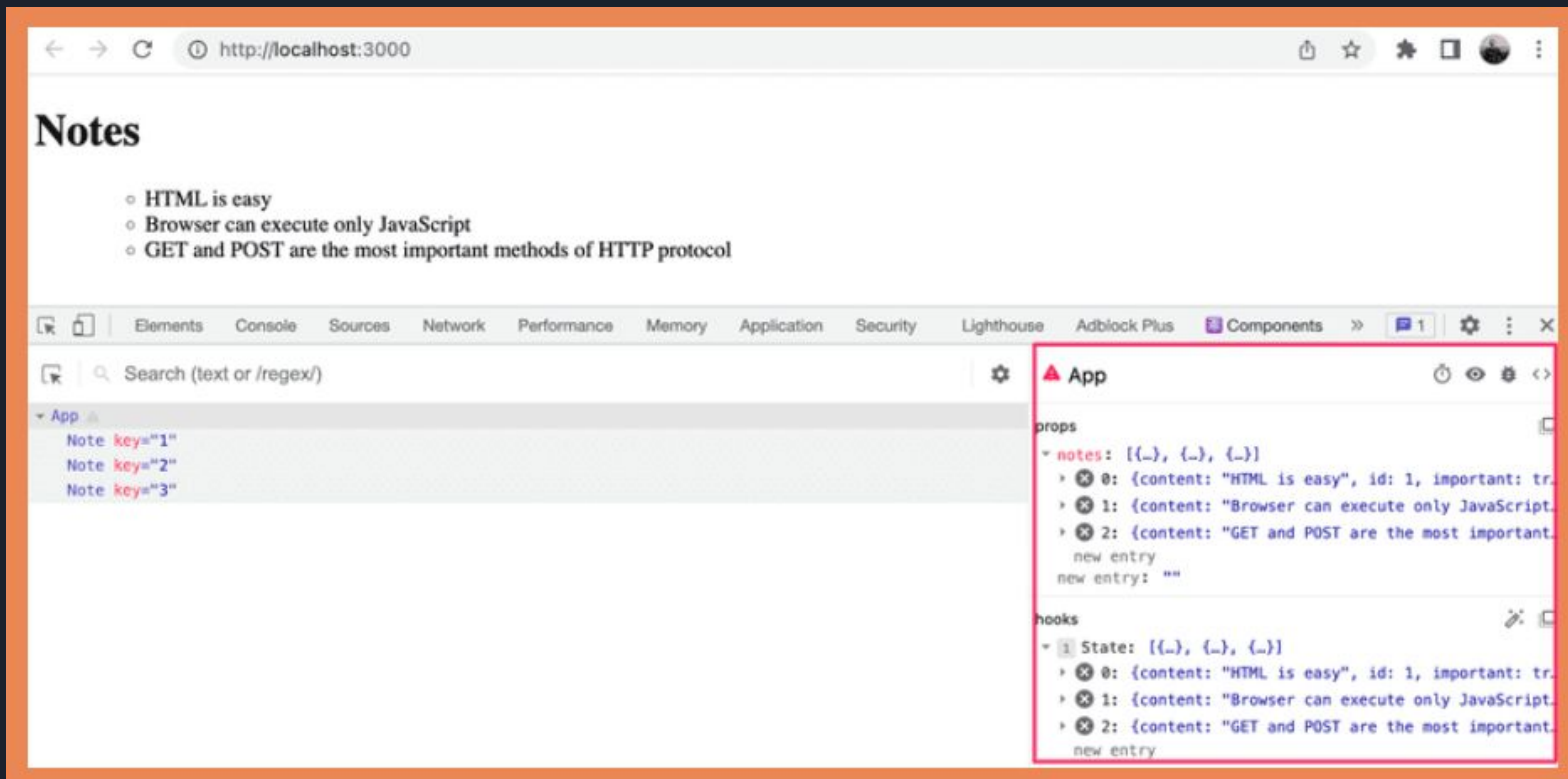
const App = (props) => {
  const [notes, setNotes] = useState(props.notes)

  return (
    <div>
      <h1>Notes</h1>
      <ul>
        {notes.map(note =>
          <Note key={note.id} note={note} />
        )}
      </ul>
    </div>
  )
}

export default App
```

Forms

Vi kan titta i developer tools vad som händer





Forms

Nu vill vi lägga till ett formulär för att lägga till noteringar...

```
const App = (props) => {  
  const [notes, setNotes] = useState(props.notes)  
  
  const addNote = (event) => {  
    event.preventDefault()  
    console.log('button clicked', event.target)  
  }  
  
  return (  
    <div>  
      <h1>Notes</h1>  
      <ul>  
        {notes.map(note =>  
          <Note key={note.id} note={note} />  
        )}  
      </ul>  
      <form onSubmit={addNote}>  
        <input />  
        <button type="submit">save</button>  
      </form>  
    </div>  
  )  
}
```

Forms

Resultatet blir detta

Notes

- HTML is easy
- Browser can execute only Javascript
- GET and POST are the most important methods of HTTP protocol

Elements Network Console Performance Sources Application Memory Security Audits

top Filter Default levels

button clicked ▶ `<form>...</form>`

>



Forms

Övning: Lägg till funktionalitet för att skapa nya notes...

- skapa ytterligare state (newNote, setNewNote)
- Lägg till onChange funktion på `<input onChange.... />` som ändrar newNote när man skriver
- Modifiera addNote funktion så att newNote blir tillagd i notes, `setNotes([...notes, newNote])`

```
const App = (props) => {  
  const [notes, setNotes] = useState(props.notes)  
  
  const addNote = (event) => {  
    event.preventDefault()  
    console.log('button clicked', event.target)  
  }  
  
  return (  
    <div>  
      <h1>Notes</h1>  
      <ul>  
        {notes.map(note =>  
          <Note key={note.id} note={note} />  
        )}  
      </ul>  
      <form onSubmit={addNote}>  
        <input />  
        <button type="submit">save</button>  
      </form>  
    </div>  
  )  
}
```



Forms

Exempel på hur addNote kan se ut...

```
const addNote = (event) => {  
  event.preventDefault()  
  const noteObject = {  
    content: newNote,  
    important: Math.random() < 0.5,  
    id: notes.length + 1,  
  }  
  
  setNotes(notes.concat(noteObject))  
  setNewNote('')  
}
```




Forms

Ibland vill vi kunna styra vad som ska visas.

Vi provar att skapa en knapp som styr om vi ska visa alla eller bara dom viktiga...

```
const App = (props) => {  
  const [notes, setNotes] = useState(props.notes)  
  const [newNote, setNewNote] = useState('')  
  const [showAll, setShowAll] = useState(true)  
  
  // ...  
}
```



Forms

Vi låter showAll state bestämma vilka notes som ska visas...

```
import { useState } from 'react'
import Note from './components/Note'

const App = (props) => {
  const [notes, setNotes] = useState(props.notes)
  const [newNote, setNewNote] = useState('')
  const [showAll, setShowAll] = useState(true)

  // ...

  const notesToShow = showAll
    ? notes
    : notes.filter(note => note.important === true)

  return (
    <div>
      <h1>Notes</h1>
      <ul>
        {notesToShow.map(note =>
          <Note key={note.id} note={note} />
        )}
      </ul>
      // ...
    </div>
  )
}
```



Forms

Vi låter en knapp styra ändringen av showAll state...

```
import { useState } from 'react'
import Note from './components/Note'

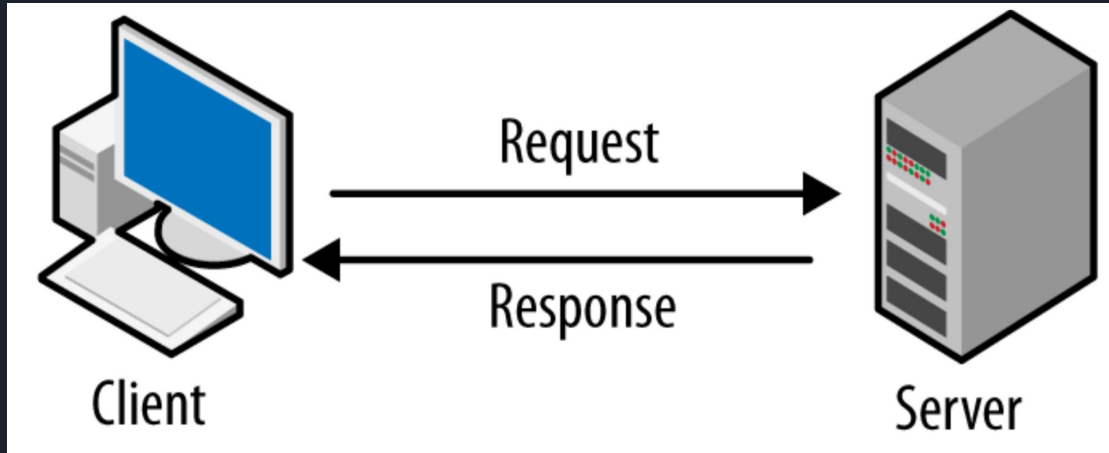
const App = (props) => {
  const [notes, setNotes] = useState(props.notes)
  const [newNote, setNewNote] = useState('')
  const [showAll, setShowAll] = useState(true)

  // ...

  return (
    <div>
      <h1>Notes</h1>
      <div>
        <button onClick={() => setShowAll(!showAll)}>
          show {showAll ? 'important' : 'all' }
        </button>
      </div>
      <ul>
        {notesToShow.map(note =>
          <Note key={note.id} note={note} />
        )}
      </ul>
      // ...
    </div>
  )
}
```

Hämta data från server


Vi modifierar koden lite för att hämta notes





Hämta data från server

Vi skapar en server (express) från vilken vi kan hämta notes istället....



```
import express from "express";

const app = express();

const notes = [
  {
    id: 1,
    content: "HTML is easy",
    important: true,
  },
  {
    id: 2,
    content: "Browser can execute only JavaScript",
    important: false,
  },
  {
    id: 3,
    content: "GET and POST are the most important methods of HTTP protocol",
    important: true,
  },
];

app.get("/notes", (req, res) => {
  res.json(notes);
});

app.listen(80, () => {
  console.log("Running on port 3000");
});
```



Async repetition

Om vi låter följande asynkrona anrop exekvera kommer appen fungera i 5 sekunder, sedan kommer allt att stanna...

```
setTimeout(() => {  
  console.log('loop..')  
  let i = 0  
  while (i < 500000000000) {  
    i++  
  }  
  console.log('end')  
}, 5000)
```



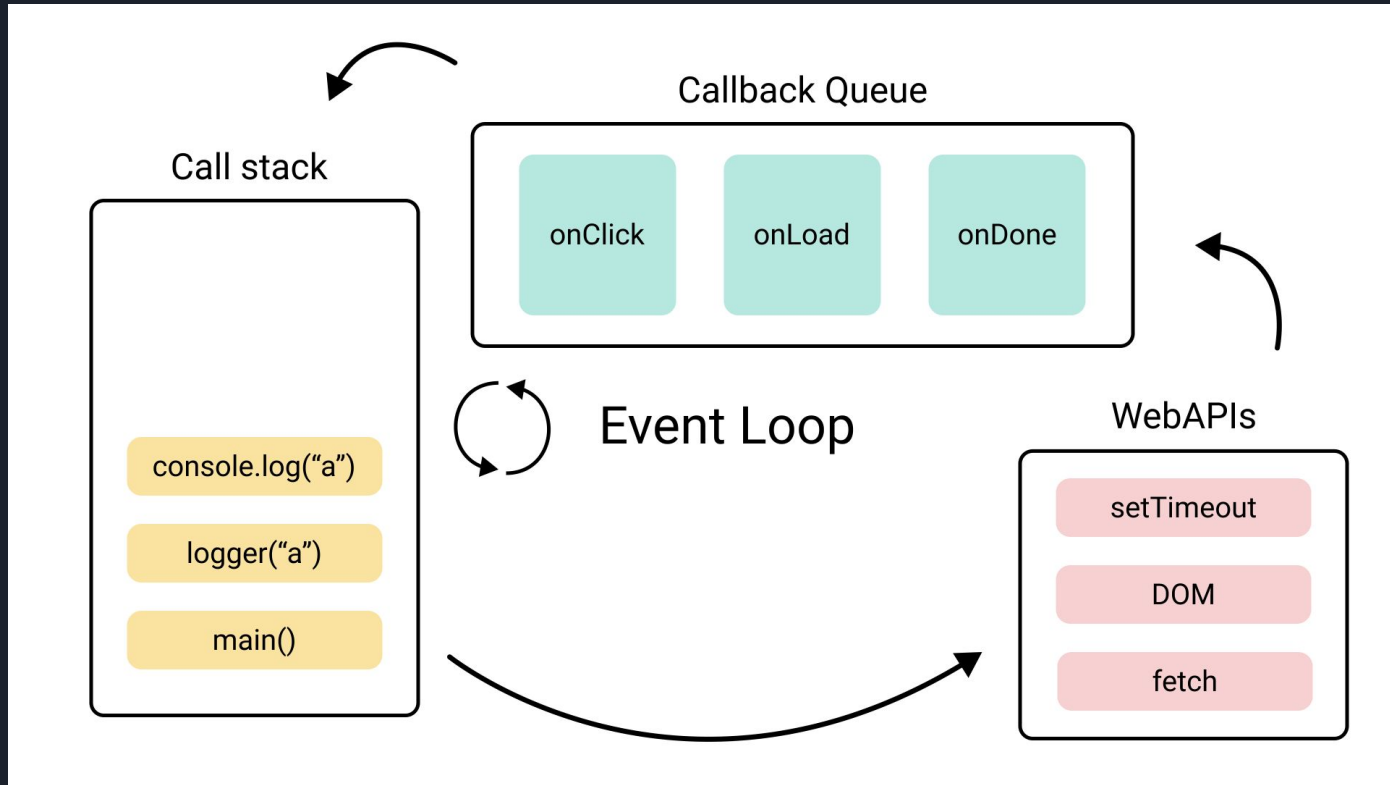
Async repetition

Detta beror på att webbläsare kan bara göra en sak i taget. Men hur är det möjligt? Det händer ju flera saker hela tiden i webbläsaren

- click
- timeouts
- fetch anrop
- etc..

```
setTimeout(() => {  
  console.log('loop..')  
  let i = 0  
  while (i < 500000000000) {  
    i++  
  }  
  console.log('end')  
}, 5000)
```


Async repetition



Async repetition





Async repetition

Så när 5 sekunder har gått hamnar
helt enkelt funktionen i callstack
och allt fryser!

```
setTimeout(() => {  
  console.log('loop..')  
  let i = 0  
  while (i < 500000000000) {  
    i++  
  }  
  console.log('end')  
}, 5000)
```



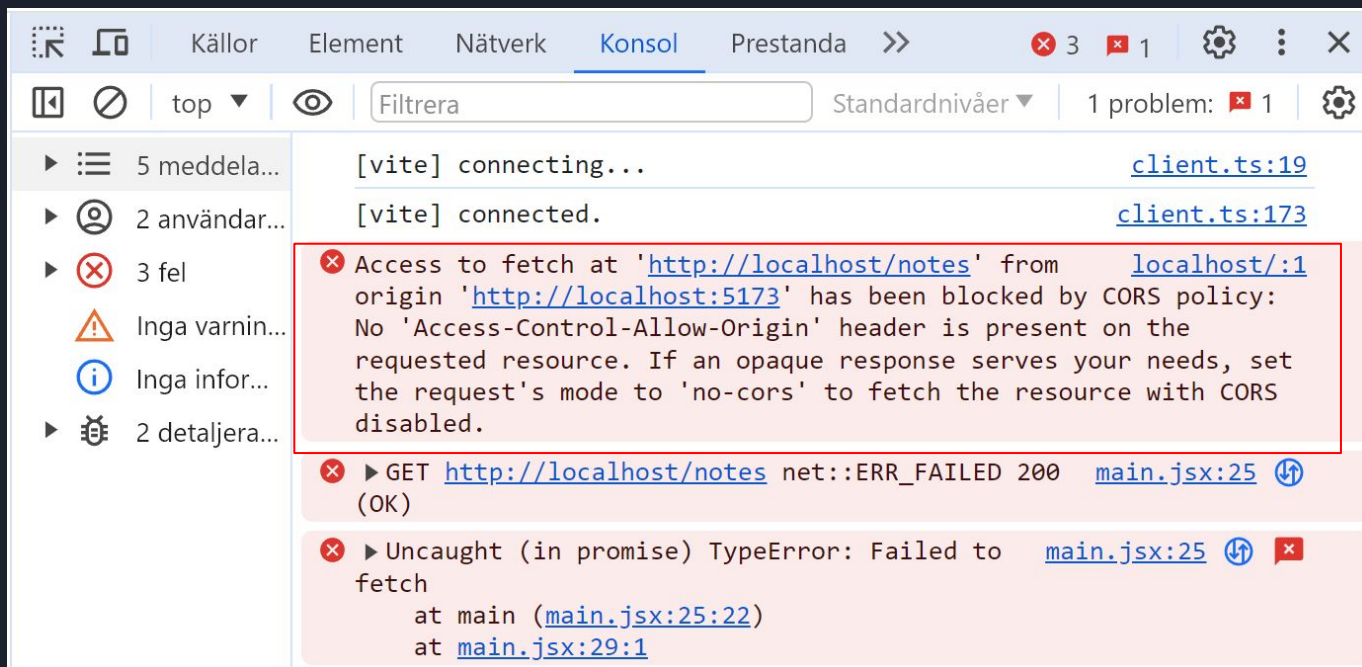
Hämta data från server

Vi provar att hämta notes i klient (react)

```
async function main() {  
  const resp = await fetch("http://localhost:80/notes");  
  const data = await resp.json();  
  console.log(data);  
}  
main();
```

Hämta data från server

Men vi får problem...



The screenshot shows a web browser's developer console with the 'Konsol' (Console) tab selected. The console displays several log messages and error messages. A red box highlights a specific error message related to a CORS policy violation.

Console Log:

- [vite] connecting... [client.ts:19](#)
- [vite] connected. [client.ts:173](#)
- Access to fetch at '<http://localhost/notes>' from origin '<http://localhost:5173>' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.
- GET <http://localhost/notes> net::ERR_FAILED 200 [main.jsx:25](#) (OK)
- Uncaught (in promise) TypeError: Failed to fetch
at main ([main.jsx:25:22](#))
at [main.jsx:29:1](#)



Hämta data från server

Vi måste tala om för vår server vilka klienter som tillåts att tala med den...

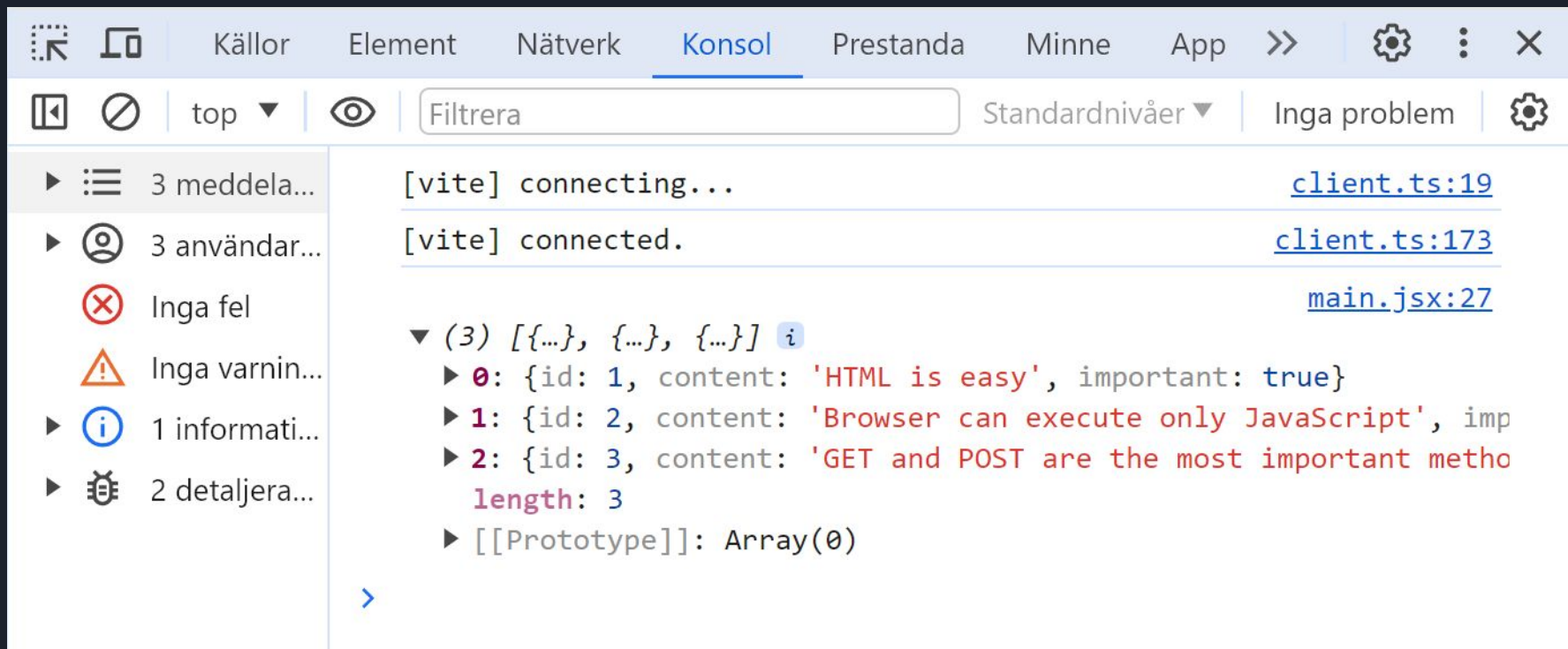
OBS: Cors middlewären tillåter alla default. Om man vill specificera <https://expressjs.com/en/resources/middleware/cors.html>

```
import express from "express";
import cors from "cors";

const app = express();
app.use(cors());

// ...
```

Hämta data från server



The screenshot shows a web browser's developer console with the 'Konsol' (Console) tab selected. The console displays the following logs:

- `[vite] connecting...` (client.ts:19)
- `[vite] connected.` (client.ts:173)
- A JSON array of 3 objects (main.jsx:27):
 - `{id: 1, content: 'HTML is easy', important: true}`
 - `{id: 2, content: 'Browser can execute only JavaScript', imp`
 - `{id: 3, content: 'GET and POST are the most important metho`

The JSON array is expanded, showing the following details:

- `length: 3`
- `[[Prototype]]: Array(0)`

The left sidebar of the developer console shows the following sections:

- 3 meddela...
- 3 användar...
- Inga fel
- Inga varnin...
- 1 informati...
- 2 detaljera...



Hämta data från server

Om vi vill använda datan från server skulle vi kunna modifiera koden på följande sätt

```
✓ import React from "react";
  import ReactDOM from "react-dom/client";
  import App from "./App";

✓ async function main() {
  |   const resp = await fetch("http://localhost:80/notes");
  |   const data = await resp.json();
✓   ReactDOM.createRoot(document.getElementById("root")).render(
  |     <App notes={data} />
  |   );
  }
  main();
```




Hämta data från server

Men bättre är att använda en hook (useEffect) som är gjord för asynkrona anrop som

- timeouts
- fetch anrop
- osv..



```
import { useState, useEffect } from "react";
import Note from "../Note";

const App = () => {
  const [notes, setNotes] = useState([]);
  const [newNote, setNewNote] = useState("");
  const [showAll, setShowAll] = useState(true);

  useEffect(() => {
    async function main() {
      const resp = await fetch("http://localhost:80/notes");
      const data = await resp.json();
    }
    main();
  }, []);

  return (
    <div>
      <h1>Notes</h1>
      <ul>
        {notes.map((note) => (
          <Note key={note.id} note={note} />
        ))}
      </ul>
    </div>
  );
};

export default App;
```



Hämta data från server

- Prova lägg till lite `console.log` för att se vad som händer...



Hämta data från server

Async förståelse

- What the heck is the eventloop anyway?
<https://www.youtube.com/watch?v=8aGhZQkoFbQ&t=1s&pp=ygUKZXZlbnQgbG9vcA%3D%3D>
- Learn useEffect hook in 13 min
<https://www.youtube.com/watch?v=0ZJgljluY7U&pp=ygUJdXNIRWZmZWNO>