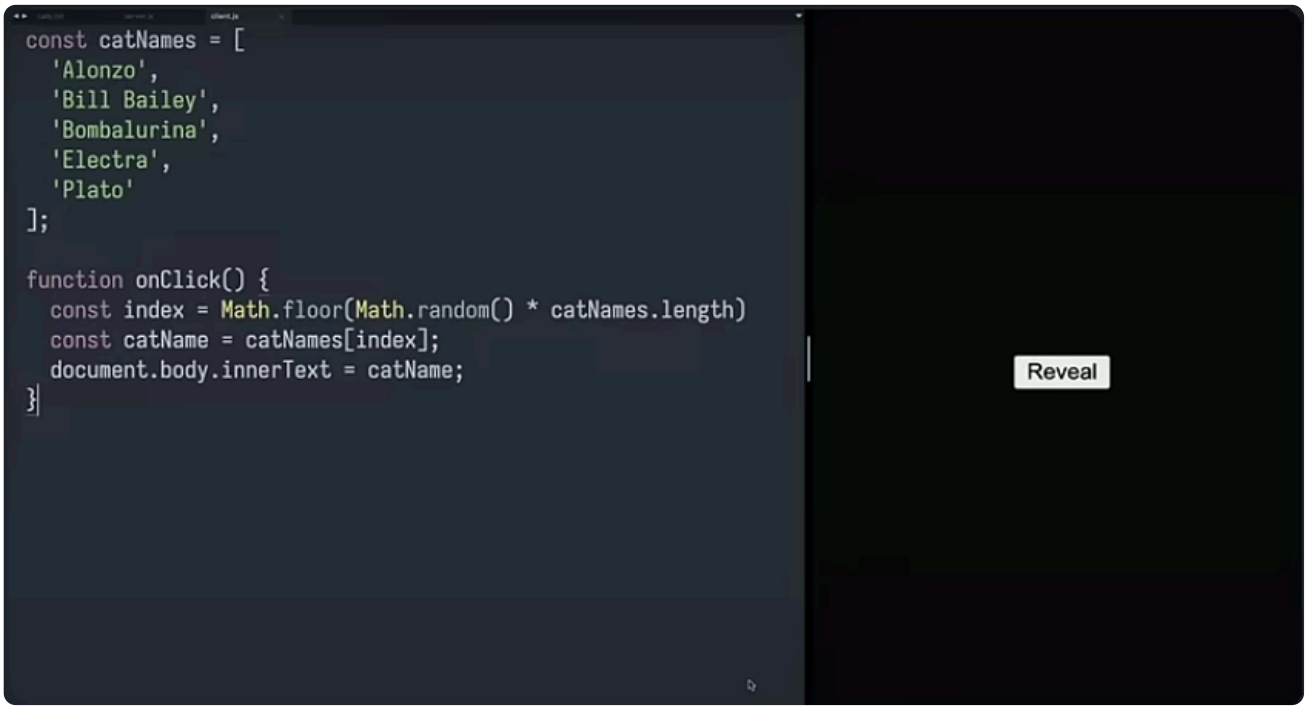
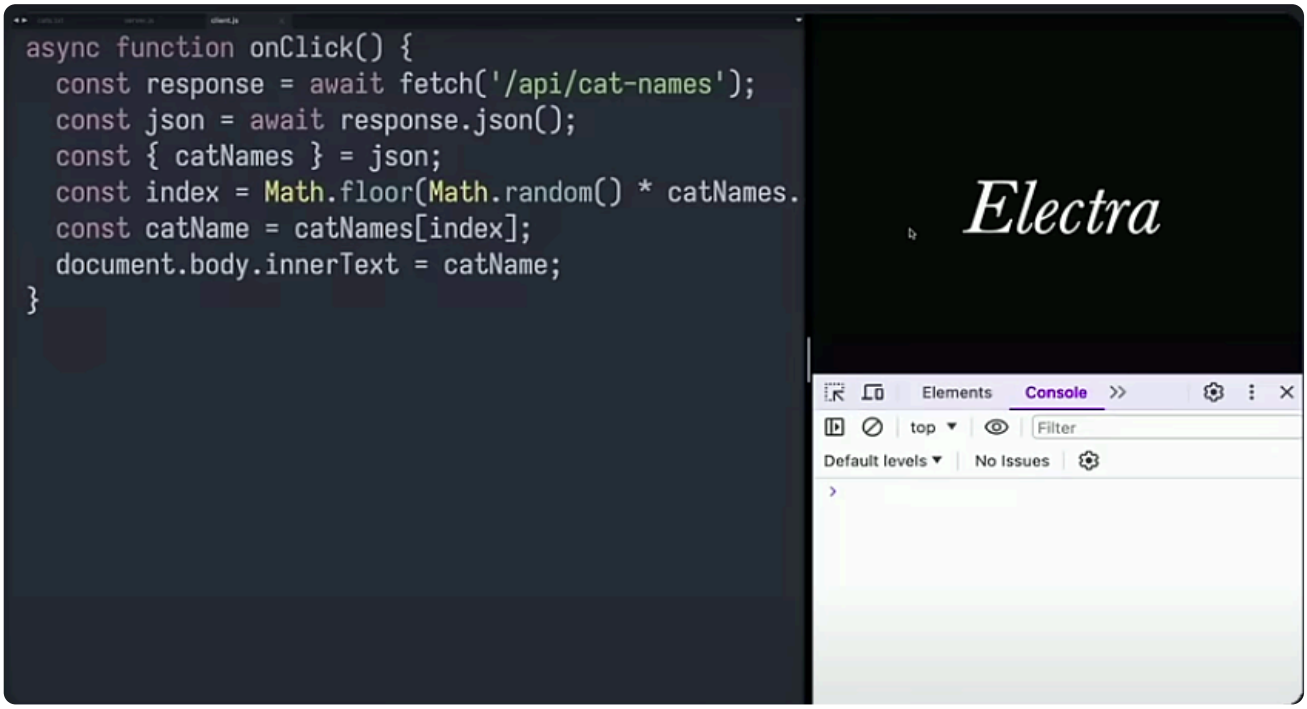
6.React for Two Computers

Talk about Server / Client relation i.e. request / response model.

We start from the following app.

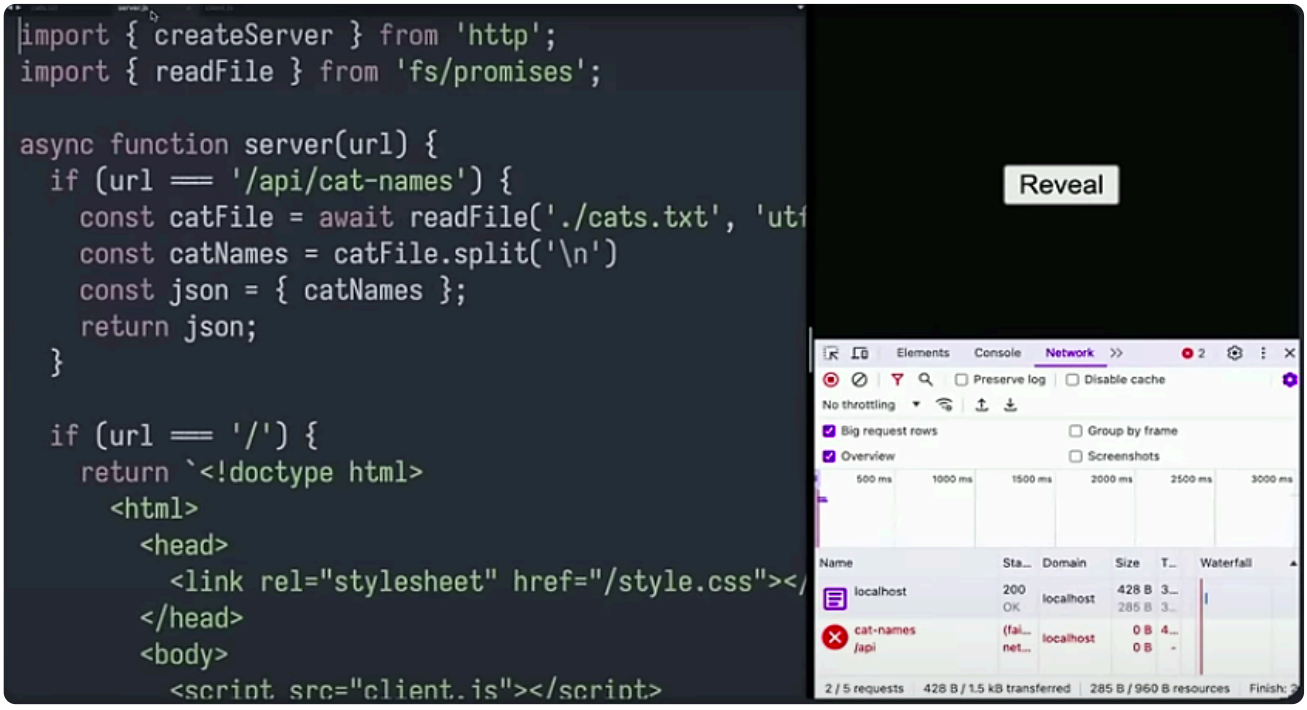


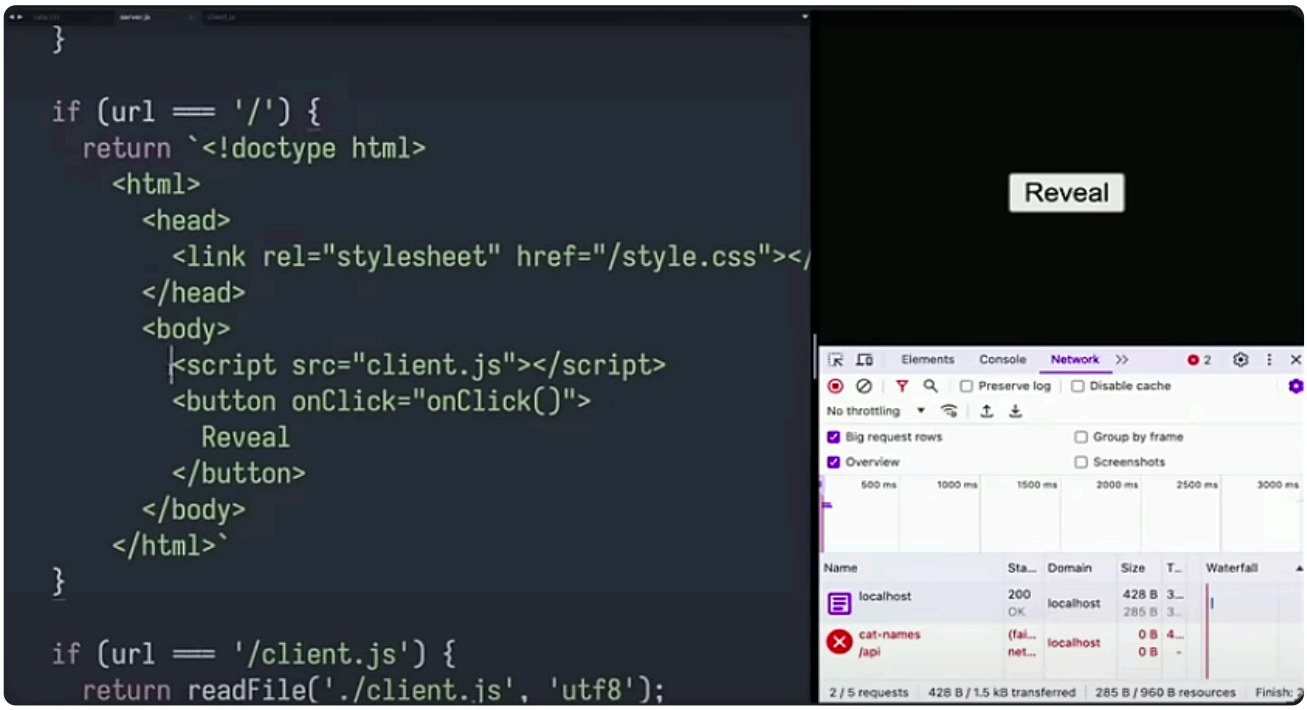
The data should not be hardcoded. We need to be able to fetch it from database / file system / etc. Traditionally we could do it in the following manner:



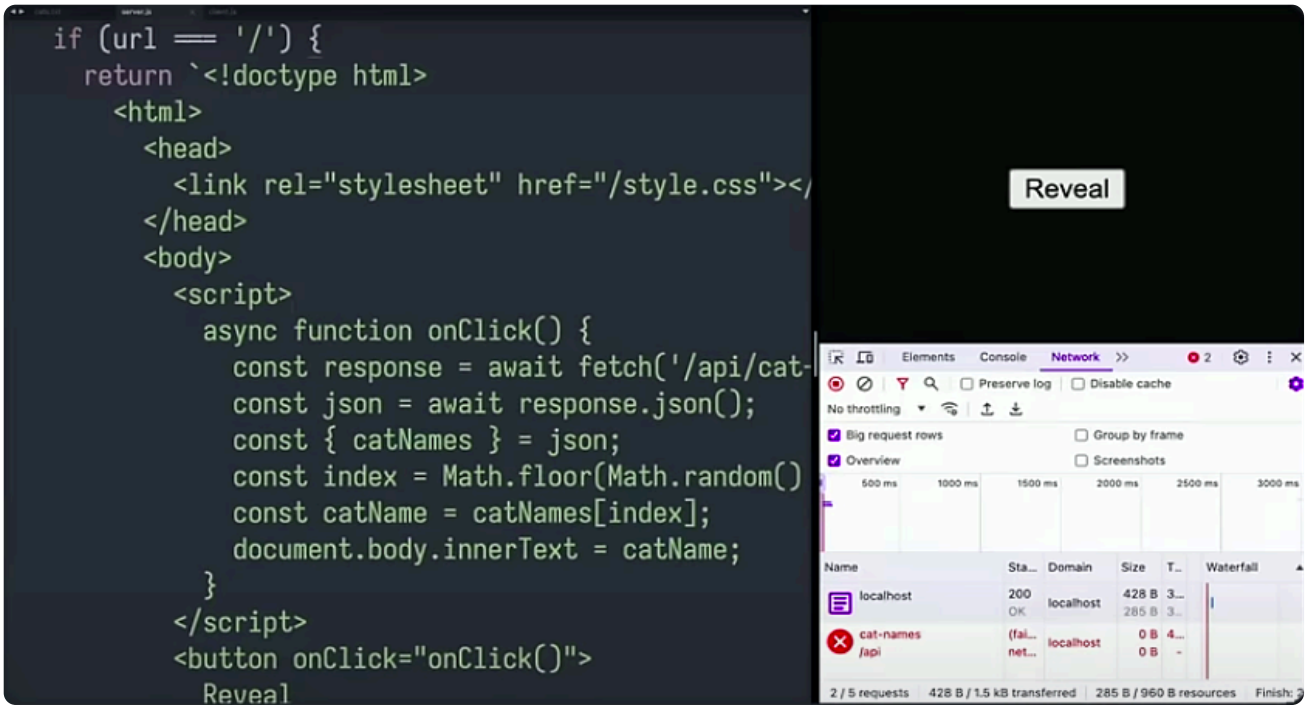
Now the behavior of the app is changed. On slow connections there is a bit of delay and it is not instantaneous anymore. That is beacuse it takes some time to load the data. For some tasks we expect to have some kind of delay in the UI if for example we submit a new blog post or save settings, etc. We want to be able to say that given interaction has to be synchronous i.e. we want the data to be already there and to not wait for it to load and visualize later. From client-side perspective there is not much we can do.

One thing that we can do is to look at the server code.





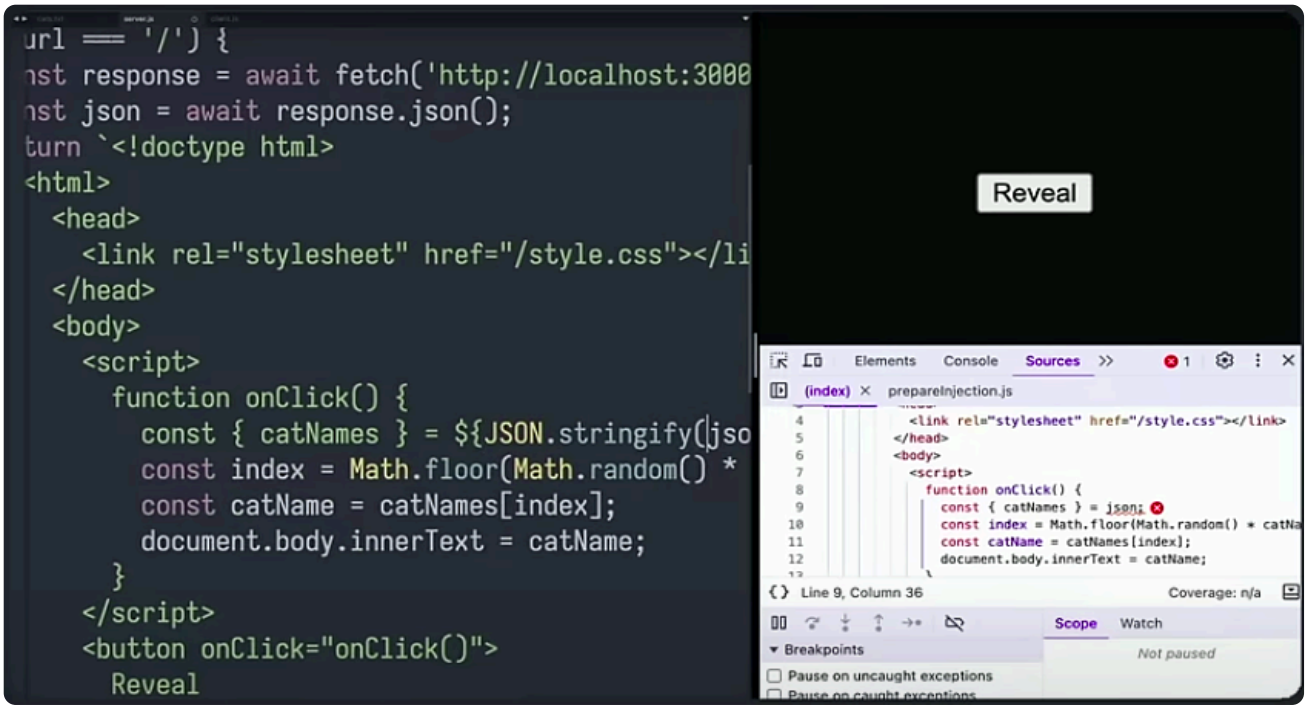
The interesting thing we notice is the client.js script.



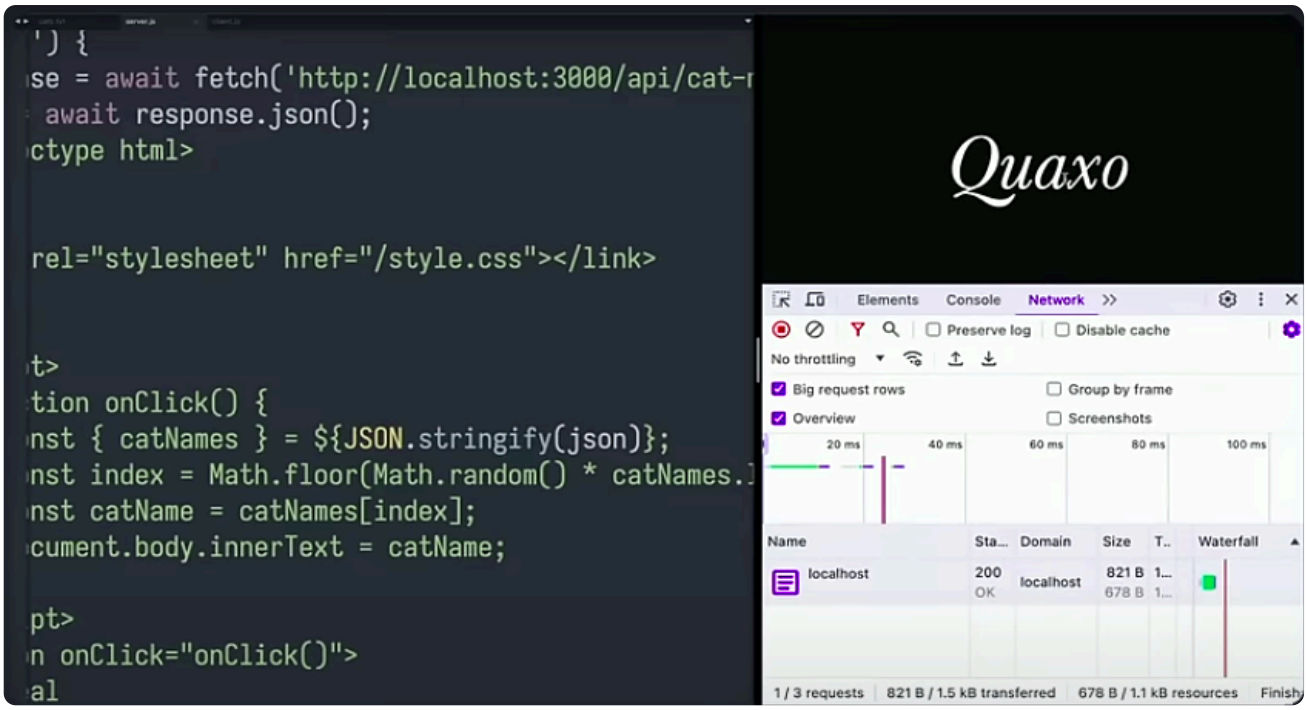
We may think that the server and client code are two separate programs that talk to each other. We could look at different perspective - a single program that spans to different devices / computers. The program is evaluated into steps. Thinking in that direction we could do some optimizations to our code. Things that could simplify it, we could not notice otherwise.

For example we want to make the handler synchronous. Then we could "lift the data up" (related with fecthing) in the same manner as React's "lift the state up" approach to parent.

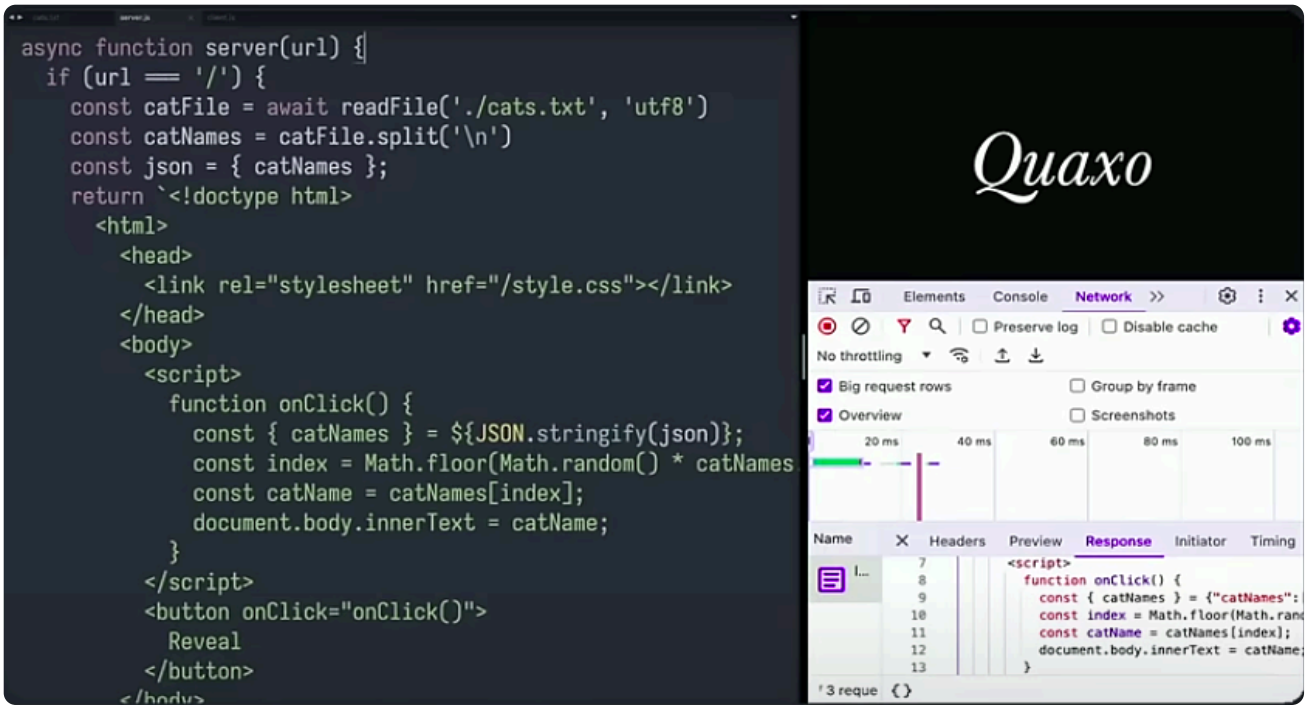
Personal Note: So here we talk about RSC and we move the fetching data in the parent Server Component.



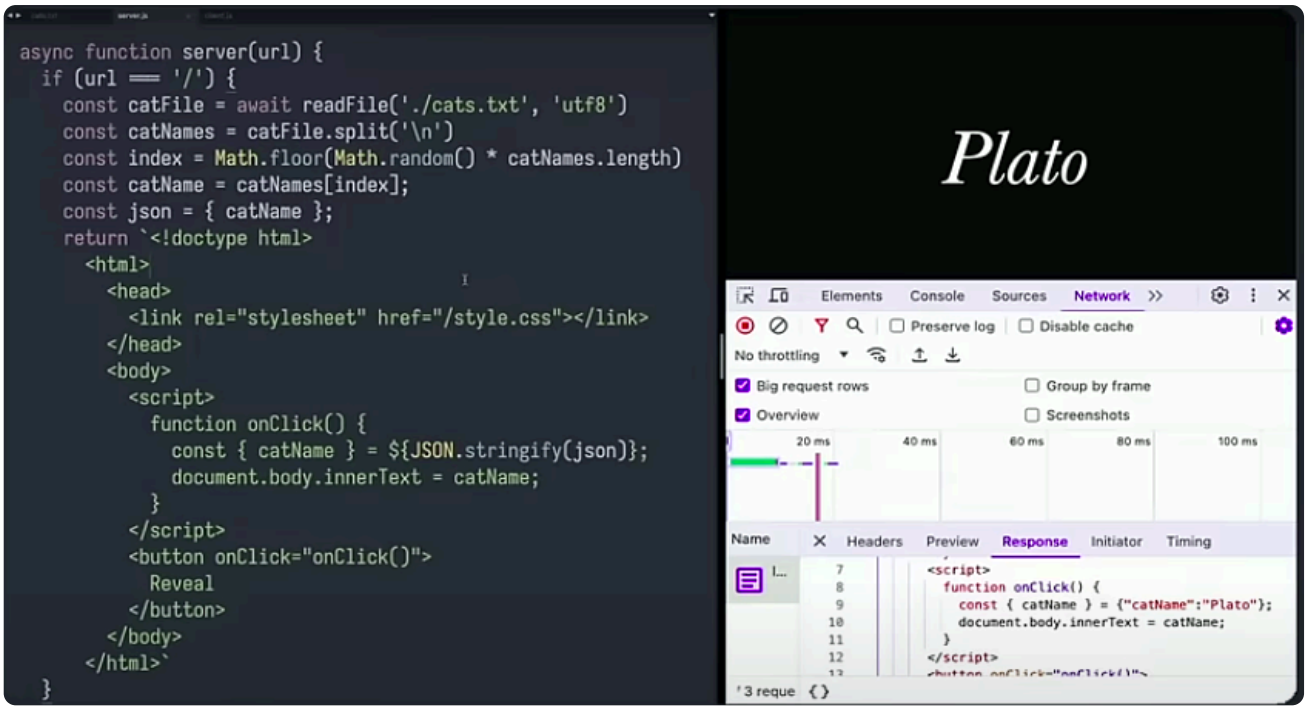
After the change when we press the button we notice that there is no network request. That is because the data is already there.



Other thing that we can do remove the api handler.

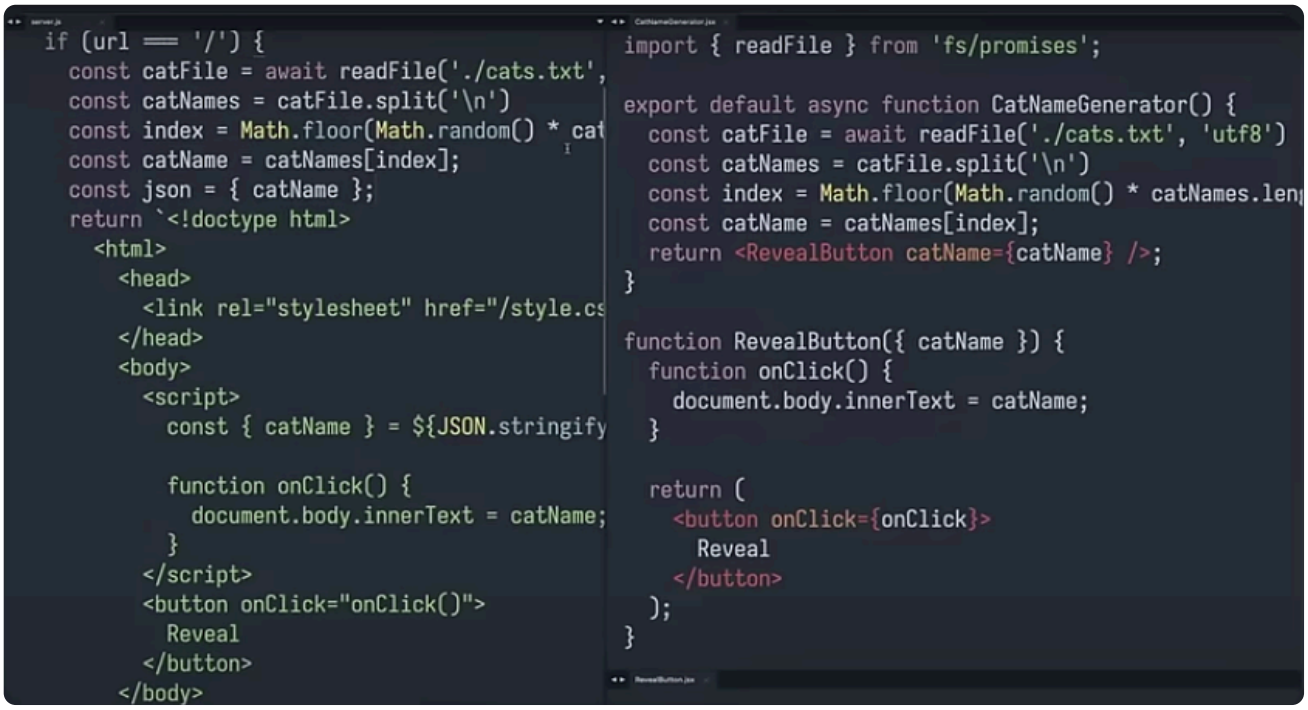


If we take a look at the response again we will notice that there is the whole data i.e. the array but we really need just one item from the array.



After the chages now we have code that executes first on one computer and emits code that will be send on another computer and we can pass some data together with this code. This is the fundamental shape of the problem and not matter which library or approach we are using. It kind of looks like this. There is a very clear separation of how the data flows through the application. But we don't want to write code like this i.e. in the template strings and without syntax highlighting / type checking / etc. Generally we

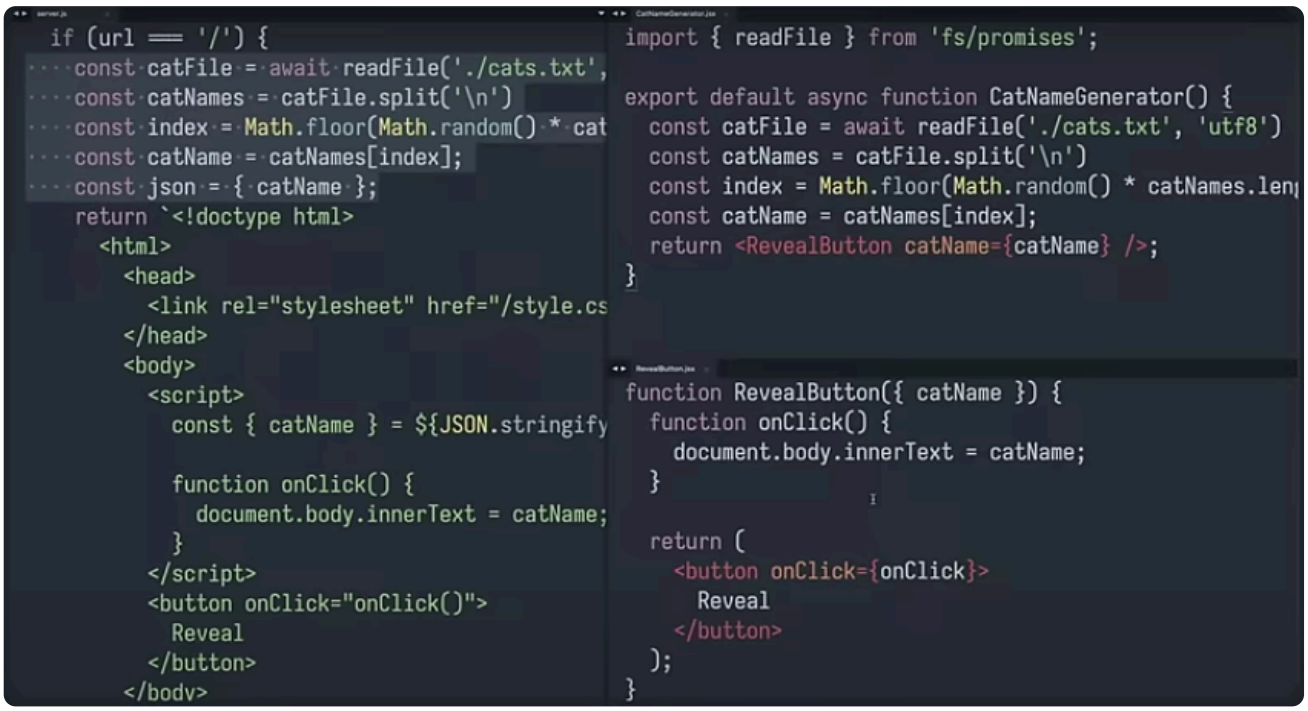
want to split the things and give them appropriate names. We have the concept of components and may use it.



```
if (url === '/') {
  const catFile = await readFile('./cats.txt',
  const catNames = catFile.split('\n')
  const index = Math.floor(Math.random() * cat
  const catName = catNames[index];
  const json = { catName };
  return `<!doctype html>
    <html>
      <head>
        <link rel="stylesheet" href="/style.cs
      </head>
      <body>
        <script>
          const { catName } = ${JSON.stringify

          function onClick() {
            document.body.innerText = catName;
          }
        </script>
        <button onClick="onClick()">
          Reveal
        </button>
      </body>
```

```
import { readFile } from 'fs/promises';

export default async function CatNameGenerator() {
  const catFile = await readFile('./cats.txt', 'utf8')
  const catNames = catFile.split('\n')
  const index = Math.floor(Math.random() * catNames.len
  const catName = catNames[index];
  return <RevealButton catName={catName} />;
}

function RevealButton({ catName }) {
  function onClick() {
    document.body.innerText = catName;
  }

  return (
    <button onClick={onClick}>
      Reveal
    </button>
  );
}
```

But actually one code could be executed on different computer and much earlier (i.e. CatNameGenerator is considered as a Server Component) and the other (i.e. ButtonReveal is considered as a Client Component) is executed on the client.

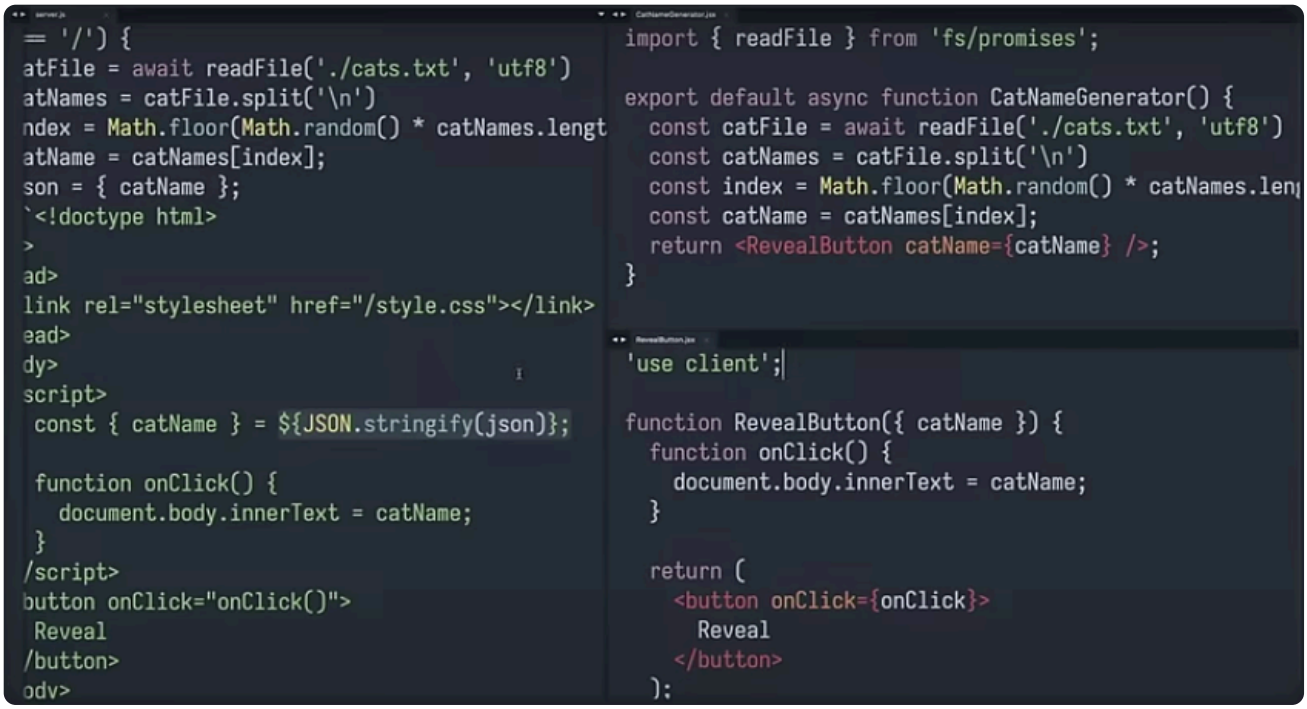


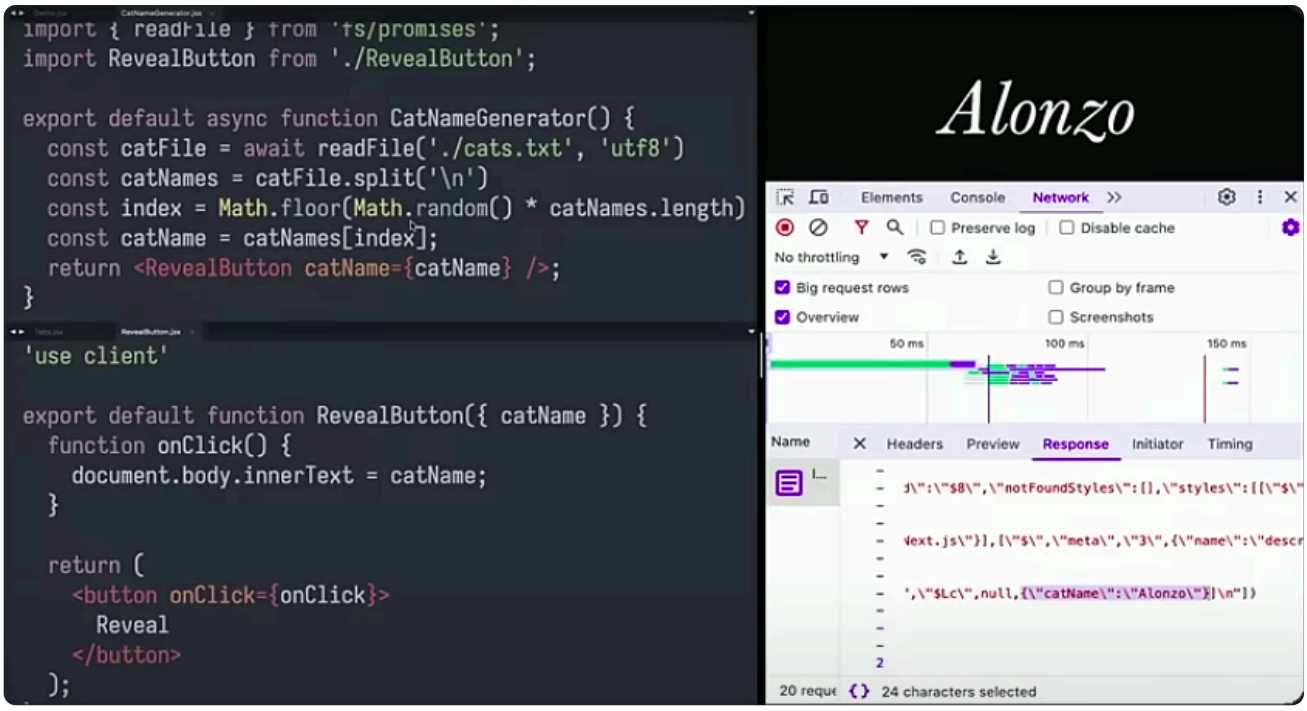
```
if (url === '/') {
  const catFile = await readFile('./cats.txt',
  const catNames = catFile.split('\n')
  const index = Math.floor(Math.random() * cat
  const catName = catNames[index];
  const json = { catName };
  return `<!doctype html>
    <html>
      <head>
        <link rel="stylesheet" href="/style.cs
      </head>
      <body>
        <script>
          const { catName } = ${JSON.stringify

          function onClick() {
            document.body.innerText = catName;
          }
        </script>
        <button onClick="onClick()">
          Reveal
        </button>
      </body>
```

```
import { readFile } from 'fs/promises';

export default async function CatNameGenerator() {
  const catFile = await readFile('./cats.txt', 'utf8')
  const catNames = catFile.split('\n')
  const index = Math.floor(Math.random() * catNames.len
  const catName = catNames[index];
  return <RevealButton catName={catName} />;
}
```

```
function RevealButton({ catName }) {
  function onClick() {
    document.body.innerText = catName;
  }

  return (
    <button onClick={onClick}>
      Reveal
    </button>
  );
}
```

We could pass data from the server component to the client component.





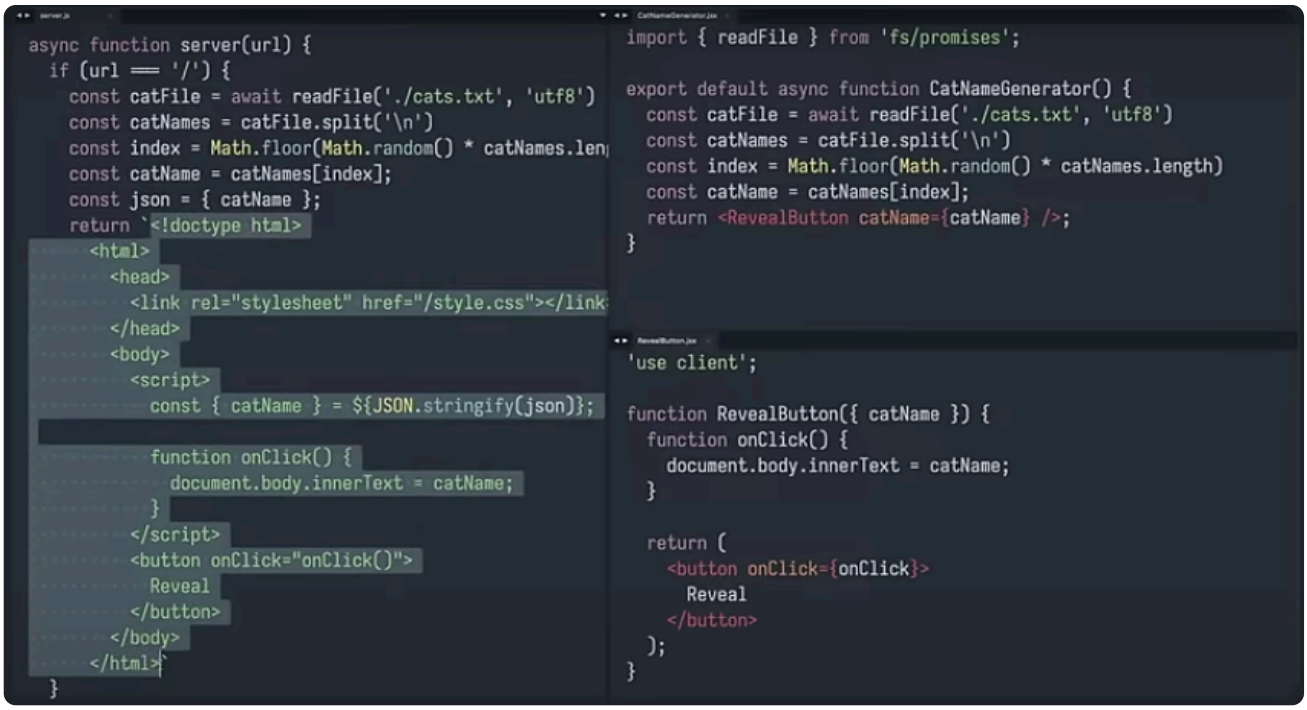
Recap: we used to think about the client and the server as a two separate programs. But we may also think about rendering a UI as a single program that is split between two computers. If we look at the vanilla JS representation the way it works is that this part executes in the server

```
async function server(url) {
  if (url === '/') {
    const catFile = await readFile('./cats.txt', 'utf8')
    const catNames = catFile.split('\n')
    const index = Math.floor(Math.random() * catNames.len
    const catName = catNames[index];
    const json = { catName };
    return `<!doctype html>
      <html>
        <head>
          <link rel="stylesheet" href="/style.css"></link
        </head>
        <body>
          <script>
            const { catName } = ${JSON.stringify(json)};

            function onClick() {
              document.body.innerText = catName;
            }
          </script>
          <button onClick="onClick()">
            Reveal
          </button>
        </body>
      </html>`
  }
```

```
import { readFile } from 'fs/promises';

export default async function CatNameGenerator() {
  const catFile = await readFile('./cats.txt', 'utf8')
  const catNames = catFile.split('\n')
  const index = Math.floor(Math.random() * catNames.length)
  const catName = catNames[index];
  return <RevealButton catName={catName} />;
}
```

```
'use client';

function RevealButton({ catName }) {
  function onClick() {
    document.body.innerText = catName;
  }

  return (
    <button onClick={onClick}>
      Reveal
    </button>
  );
}
```

And then it emits a program that execute in the client



```
async function server(url) {
  if (url === '/') {
    const catFile = await readFile('./cats.txt', 'utf8')
    const catNames = catFile.split('\n')
    const index = Math.floor(Math.random() * catNames.len
    const catName = catNames[index];
    const json = { catName };
    return `<!doctype html>
      <html>
        <head>
          <link rel="stylesheet" href="/style.css"></link
        </head>
        <body>
          <script>
            const { catName } = ${JSON.stringify(json)};

            function onClick() {
              document.body.innerText = catName;
            }
          </script>
          <button onClick="onClick()">
            Reveal
          </button>
        </body>
      </html>`
  }
```

```
import { readFile } from 'fs/promises';

export default async function CatNameGenerator() {
  const catFile = await readFile('./cats.txt', 'utf8')
  const catNames = catFile.split('\n')
  const index = Math.floor(Math.random() * catNames.length)
  const catName = catNames[index];
  return <RevealButton catName={catName} />;
}
```

```
'use client';

function RevealButton({ catName }) {
  function onClick() {
    document.body.innerText = catName;
  }

  return (
    <button onClick={onClick}>
      Reveal
    </button>
  );
}
```

And there is kind of like a hole through which we could pass an information i.e. serialized JSON.

In the React version we have Server and Client components. For practical reasons on the server we want to run both stages, because we want to generate the initial HTML so we run this stage with the initial state. But on the client we have the part that update and re-render in response to state changes.