Lazy loading

Let's focus on eager loading vs. lazy loading with React. First we'll examine our eager-loaded app, then implement lazy loading and take a look at the improvements.

Without lazy loading

1. Go `npm run build` and look in the dist folder. What's in the dist/assets folder? Write the file names down here.

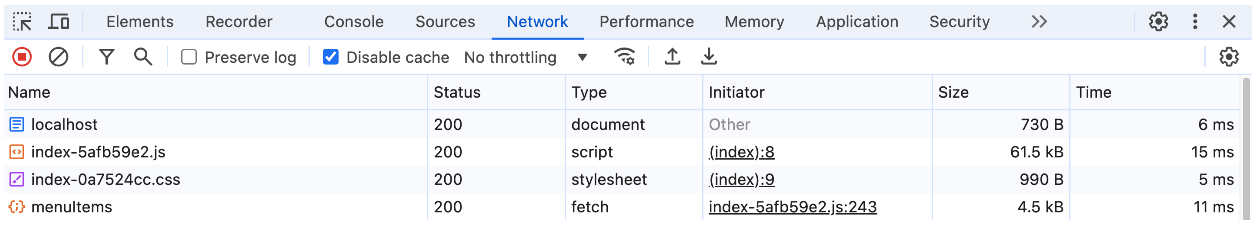
   ____

The JavaScript file is your bundle. It contains all of your JavaScript, React itself, the libraries React uses and the libraries that they use. And more. It's big. It's time-consuming and bandwidth-consuming. This gets downloaded every time someone visits the site, whether or not they use all of the resources in the bundle.

2. Run your app with `npm run preview`

3. Open the browser dev tools and look at the navigation tab.

4. Refresh the page. Make note of the files that were added. Look for the bundled JavaScript file. How big is it?



Name	Status	Type	Initiator	Size	Time
localhost	200	document	Other	730 B	6 ms
index-5afb59e2.js	200	script	(index):8	61.5 kB	15 ms
index-0a7524cc.css	200	stylesheet	(index):9	990 B	5 ms
menuItems	200	fetch	index-5afb59e2.js:243	4.5 kB	11 ms

5. Click the "Past Orders" menu option while looking at the network tab. See anything happen other than a fetch to "/orders"? You shouldn't because we've already got the Orders component in the bundle.

Notice that everyone comes to our app to order food. But how many will be looking at their old orders? Very few, right? We should probably not waste 99% of the people's time for the 1% of people who want to research old orders. Hey, let's lazy load the Orders component!

Creating a fallback message

Remember you're going to want to create a fallback and then wrap your components in a <Suspense>

6. Create a fallback component. It should have any type of UX affordance to let the user know they're waiting. Make it a simple 'Loading ...' message or go nuts and use an hourglass/beachball spinning indicator. It's up to you.

7. Wrap all of your <Routes> in a <Suspense> with the fallback you just created. (Hint: don't forget to import it).

8. Alter the imports of Order and Orders to be lazy. Here's a rough example of how you'd do Orders:

```
const Orders = lazy(() => import('./components/Orders.tsx'));
```

9. Run and test. You should see no difference at this point.

Did we just do anything?

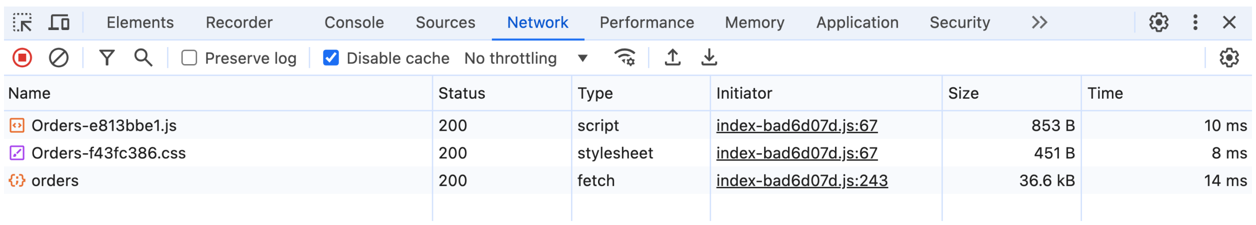
No difference? Did we just write that code for nothing? Hmmm. Let's look again to see if it's different.

10. Making sure you wrote down the files in the dist/assets folder, do another `npm run build`.

11. Now look at the files and compare them to before. You should see two JavaScript bundes instead of one.

12. Go `npm run preview` and look at the network debugger tab. Look for your bundle(s). Do you see them both? You shouldn't -- at least not yet.

13. Navigate to past orders and look in the tab again. You should see another JIT/lazy fetch for the much smaller bundle that has just orders.



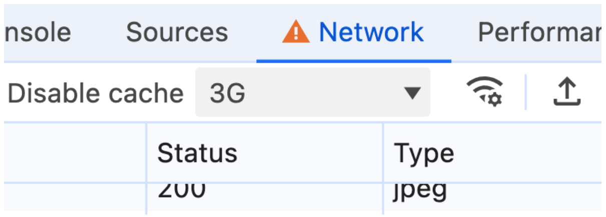
Name	Status	Type	Initiator	Size	Time
Orders-e813bbe1.js	200	script	index-bad6d07d.js:67	853 B	10 ms
Orders-f43fc386.css	200	stylesheet	index-bad6d07d.js:67	451 B	8 ms
orders	200	fetch	index-bad6d07d.js:243	36.6 kB	14 ms

Experiencing the laziness

Still not convinced? "Where's my 'loading' indicator?", you say? It's refreshing too fast for you to see. Here try this.

14. Navigate back to the main view. ie. Click the home link.

15. Still in your dev tools network tab, throttle the requests as far down as possible; probably to "3G" speeds.



16. Refresh your browser. You should see it be super slow to load.

17. Now click on the Order link/button and watch in the list of files. It should be way more obvious now. If you don't see your loading/fallback component, there's something wrong.

But if you do, you're finished. Congrats!