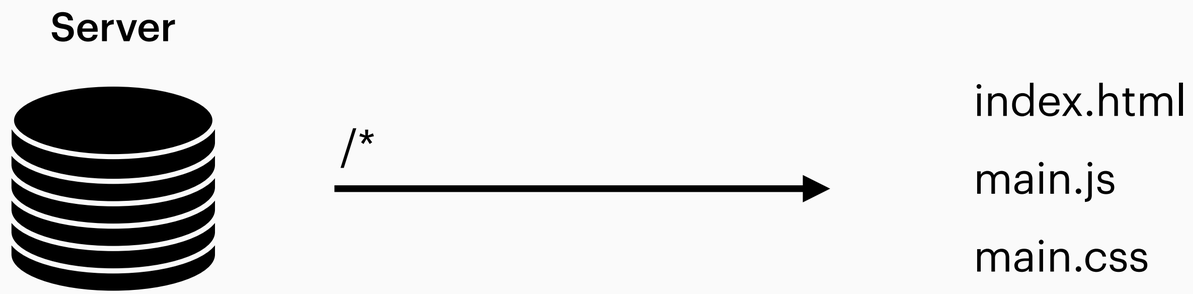
Server Side Rendering

with React in general and with NextJS in particular

Tobias Lohse • July 13, 2022

Single Page Applications (SPA)

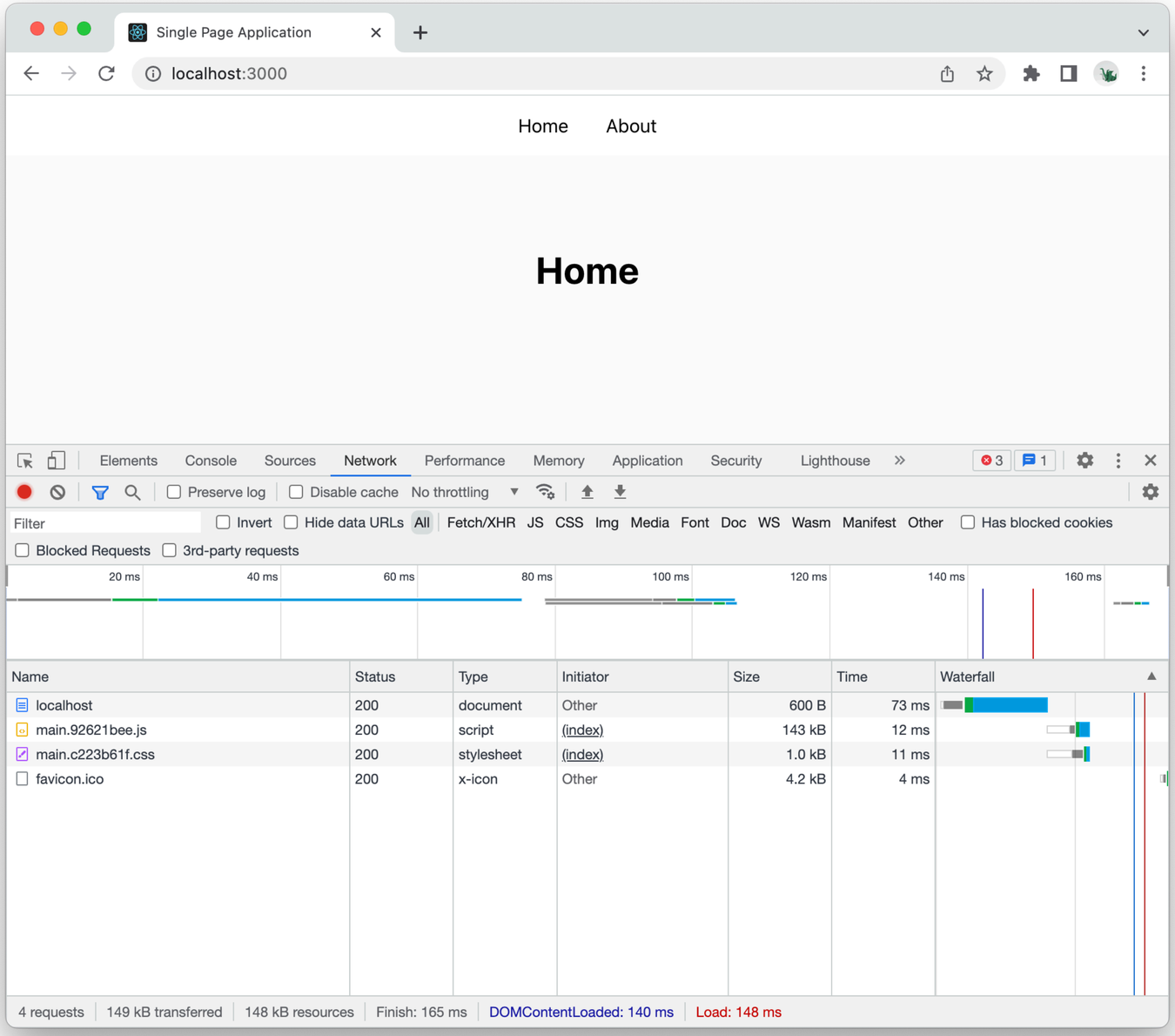
rendering HTML with JS and routing on the client



All routes get the same HTML/CSS/JS

Routing happens on the client side

React inserts HTML into the DOM


Single Page Application
localhost:3000
Home
About
Home
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
3
1
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
20 ms
40 ms
60 ms
80 ms
100 ms
120 ms
140 ms
160 ms
Name
Status
Type
Initiator
Size
Time
Waterfall
localhost 200 document Other 600 B 73 ms
main.92621bee.js 200 script (index) 143 kB 12 ms
main.c223b61f.css 200 stylesheet (index) 1.0 kB 11 ms
favicon.ico 200 x-icon Other 4.2 kB 4 ms
4 requests
149 kB transferred
148 kB resources
Finish: 165 ms
DOMContentLoaded: 140 ms
Load: 148 ms

Websites are still HTML pages, but in a SPA the HTML is boring

index.html (typical html template with CSS and JS links inserted by webpack)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" href="static/favicon.ico" />
    <title>Single Page Application</title>

    <script defer="defer" src="/static/js/main.2f544898.js" />
    <link rel="stylesheet" href="/static/css/main.612981ed.css" />

  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
      <!-- react renders into this -->
    </div>
  </body>
</html>
```

What does the Server for a SPA look like?

server.ts (a simple fastify server to serve the SPA bundle that was generated by webpack)

```
1  const server = fastify({ logger: true });
2
3  // serve static content
4  server.register(staticPlugin, {
5    root: path.join(__dirname, '../build/static'),
6    prefix: '/static',
7  });
8
9  // serve the same index.html for every route
10 server.get('/*', async (req, res) => {
11   const html: string = await readFile(path.join(__dirname, '../build/index.html'), 'utf8');
12   res.type('text/html').send(html);
13 });
14
15 server.listen({ port: 3000 });
```

How is the right HTML generated for a specific route?

index.tsx (minimal React app with client side routing, doesn't handle browser back button, use router lib in practice)

```tsx
function App() {
  const [currentPath, setCurrentPath] = useState<string>(window.location.pathname);

  const goToRoute = (e: MouseEvent<HTMLAnchorElement>): void => {
    e.preventDefault();
    const path = e.target.getAttribute('href')!;
    setCurrentPath(path);
    window.history.pushState({}, '', path);
  };

  const routes: Route[] = [
    { path: '/', render: () => <h1>Home</h1>, label: 'Home' },
    { path: '/about', render: () => <h1>About</h1>, label: 'About' },
  ];
  const currentRoute: Route | undefined = routes.find(({ path }) => path === currentPath);

  return (<>
    <nav>
      {routes.map(({ path, label }) => (
        <a key={path} href={path} onClick={goToRoute}>{label}</a>
      ))}
    </nav>
    <main>
      {currentRoute ? currentRoute.render() : <h1>404</h1>}
    </main>
  </>);
};
ReactDOM.createRoot(document.getElementById('root')!).render(<App />);
```

Downsides of SPAs?

Nothing shows until React finishes rendering

Search engines cannot crawl the page

⇒ `Server Side Rendering`

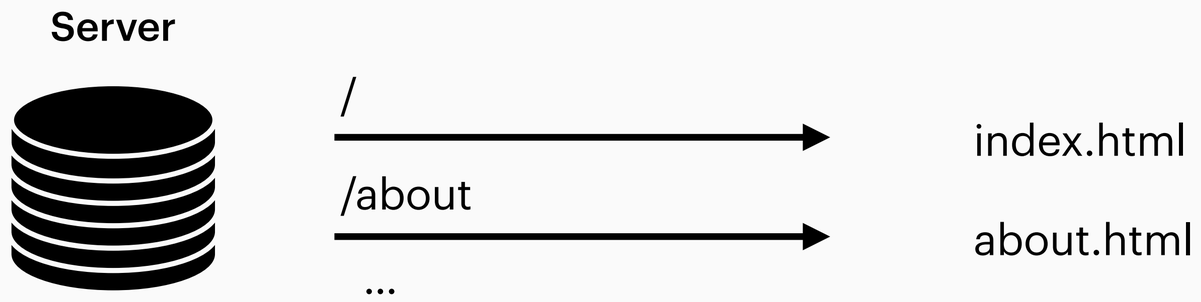
Whole bundle loads for each route

Logic in the client increases bundle size

⇒ `Code Splitting & Server Components`

Server Side Rendering (SSR)

rendering the initial HTML on the Server



Initial HTML for each route is generated on the server

React hydrates page with matching HTML on client

Routing still happens on client after initial request

All routes still get same CSS/JS bundle

How do we render React to HTML on the server?

App.tsx (shared component used for both client and server side rendering)

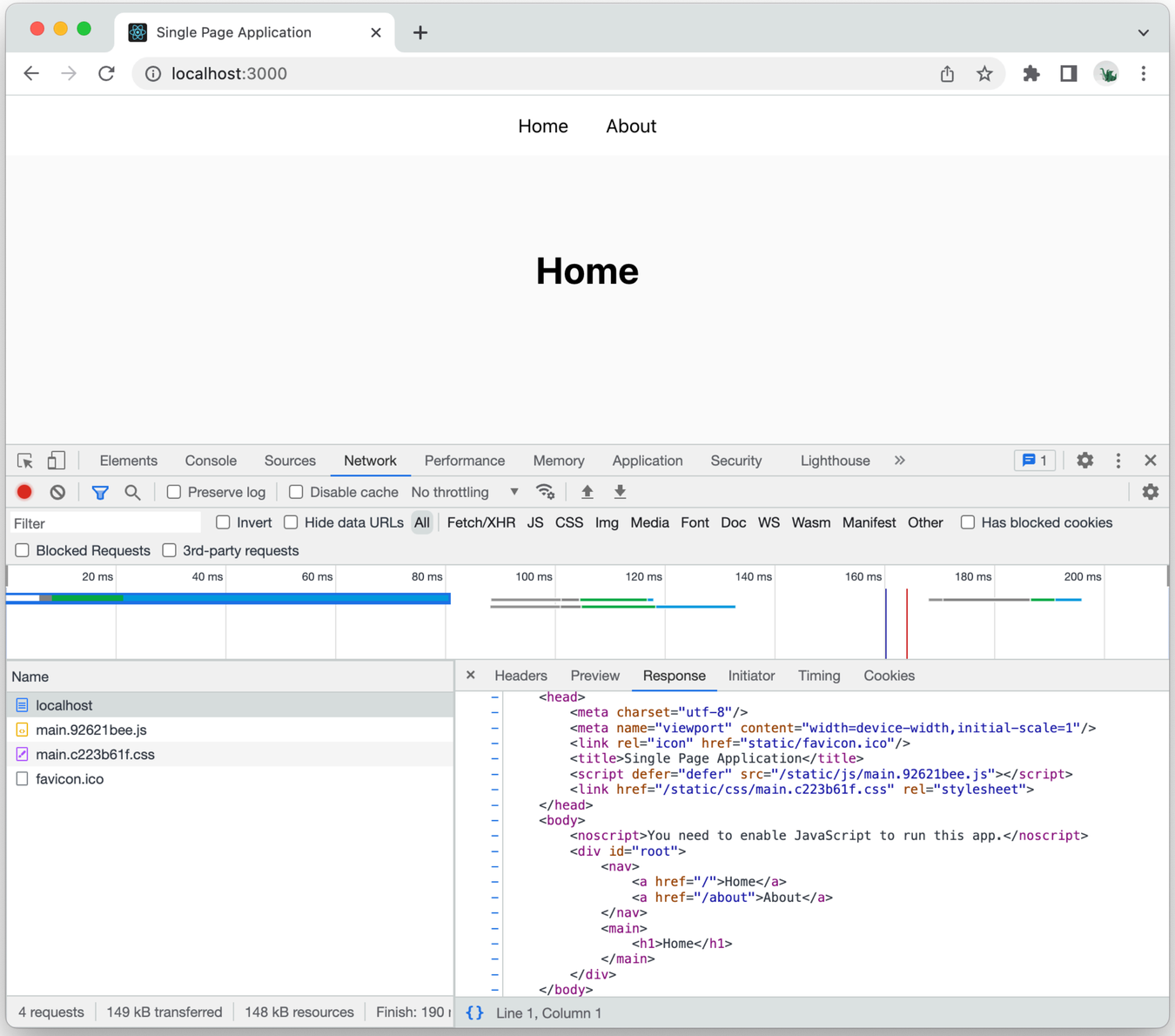
```tsx
export const routes: Route[] = […]

export default function App({ startPath = window.location.pathname }: { startPath?: string }) {
  const [currentPath, setCurrentPath] = useState<string>(startPath);
  …
}
```

server.ts (server rendering page to HTML string for initial request)

```ts
import App, { routes } from './App';

server.get<{ Params: { path: string } }>('/:path', async (req, res) => {
  const startPath: string = '/' + req.params.path;
  if (!routes.find(({ path }) => path === startPath)) res.status(404);

  const htmlWrapper: string = await readFile(path.join(__dirname, '../build/index.html'), 'utf8');
  const appElement: ReactElement = React.createElement(App, { startPath });
  const appHtml: string = ReactDOMServer.renderToString(appElement);
  const html: string = htmlWrapper.replace('<div id="root"></div>', `<div id="root">${appHtml}</div>`);

  res.type('text/html').send(html);
});
```

index.tsx (hydrate server rendered HTML with the same HTML on the client side)

```tsx
import App from './App';

ReactDOM.hydrateRoot(document.getElementById('root')!, <App />);
```


Single Page Application
localhost:3000
Home
About
Home
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
20 ms
40 ms
60 ms
80 ms
100 ms
120 ms
140 ms
160 ms
180 ms
200 ms
Name
localhost
main.92621bee.js
main.c223b61f.css
favicon.ico
Headers
Preview
Response
Initiator
Timing
Cookies
<head>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width,initial-scale=1"/>
<link rel="icon" href="static/favicon.ico"/>
<title>Single Page Application</title>
<script defer="defer" src="/static/js/main.92621bee.js"></script>
<link href="/static/css/main.c223b61f.css" rel="stylesheet">
</head>
<body>
<noscript>You need to enable JavaScript to run this app.</noscript>

<nav>
Home
About
</nav>
<main>
<h1>Home</h1>
</main>

</body>
4 requests
149 kB transferred
148 kB resources
Finish: 190
Line 1, Column 1

Client

Chromium (V8), SpiderMonkey, WebKit (JSCore), Chakra

- Limited free compute power
- Window Object (e.g. history)
- State/Effects
- CSS Animations
- Local storage & Client Cookies
- Web APIs (e.g. location)
- Browser Language Preferences

vs

Server

Node (V8), Deno (V8), Bun (JSCore)

- Scalable paid compute power
- Environment Variables
- DB connections
- File systems
- Response headers
- IP Location
- Server only Cookies

Zero Config Defaults and Optimized Bundling

Opinionated File System based Router

Server Side Rendering and Static Site Generation

Automatic Code Splitting and Prefetching

Server and Client Side rendered Header elements

Standardized Server Side Data Fetching

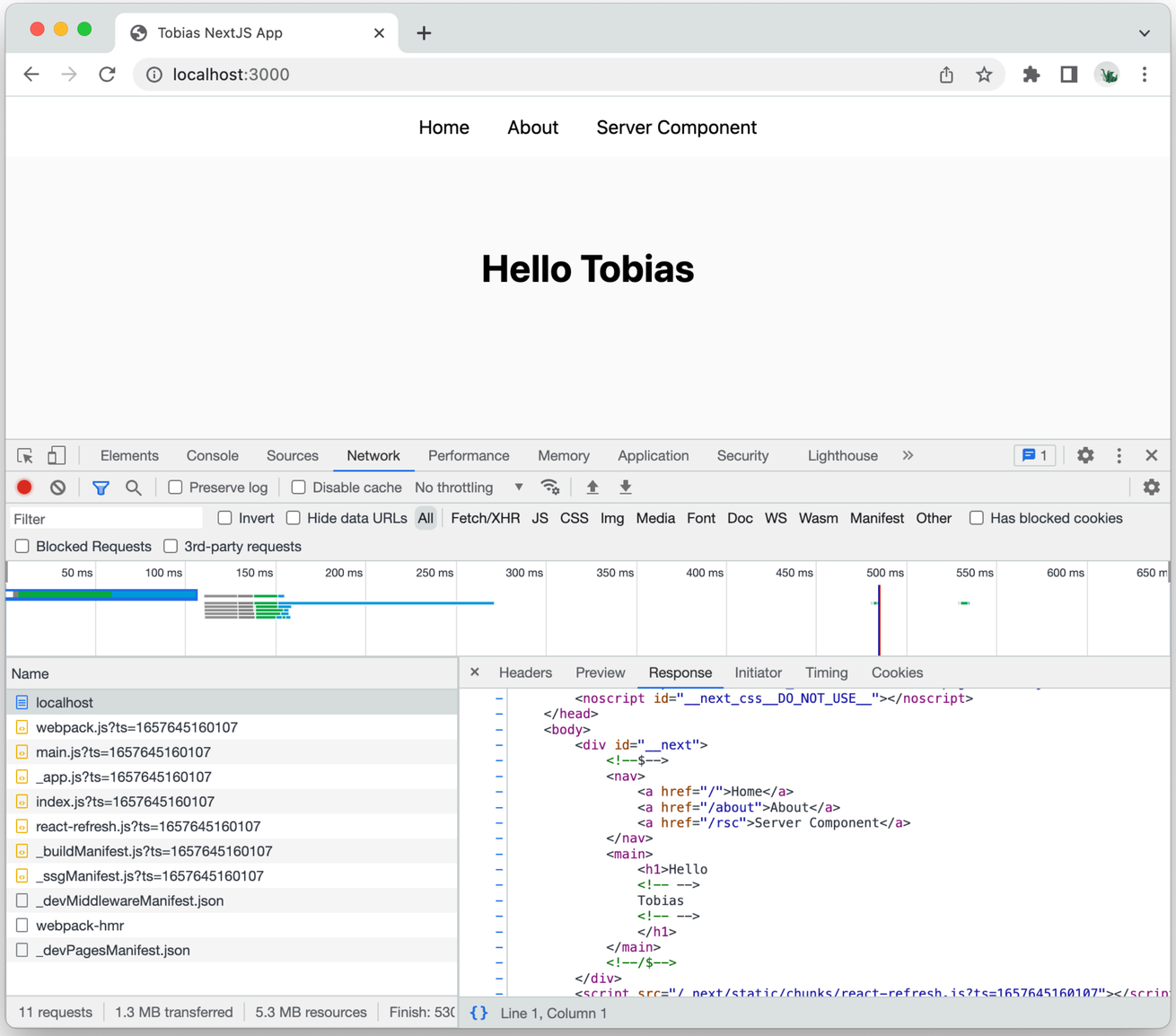
Image Optimization and CDN Support

Declare API data dependencies for initial render

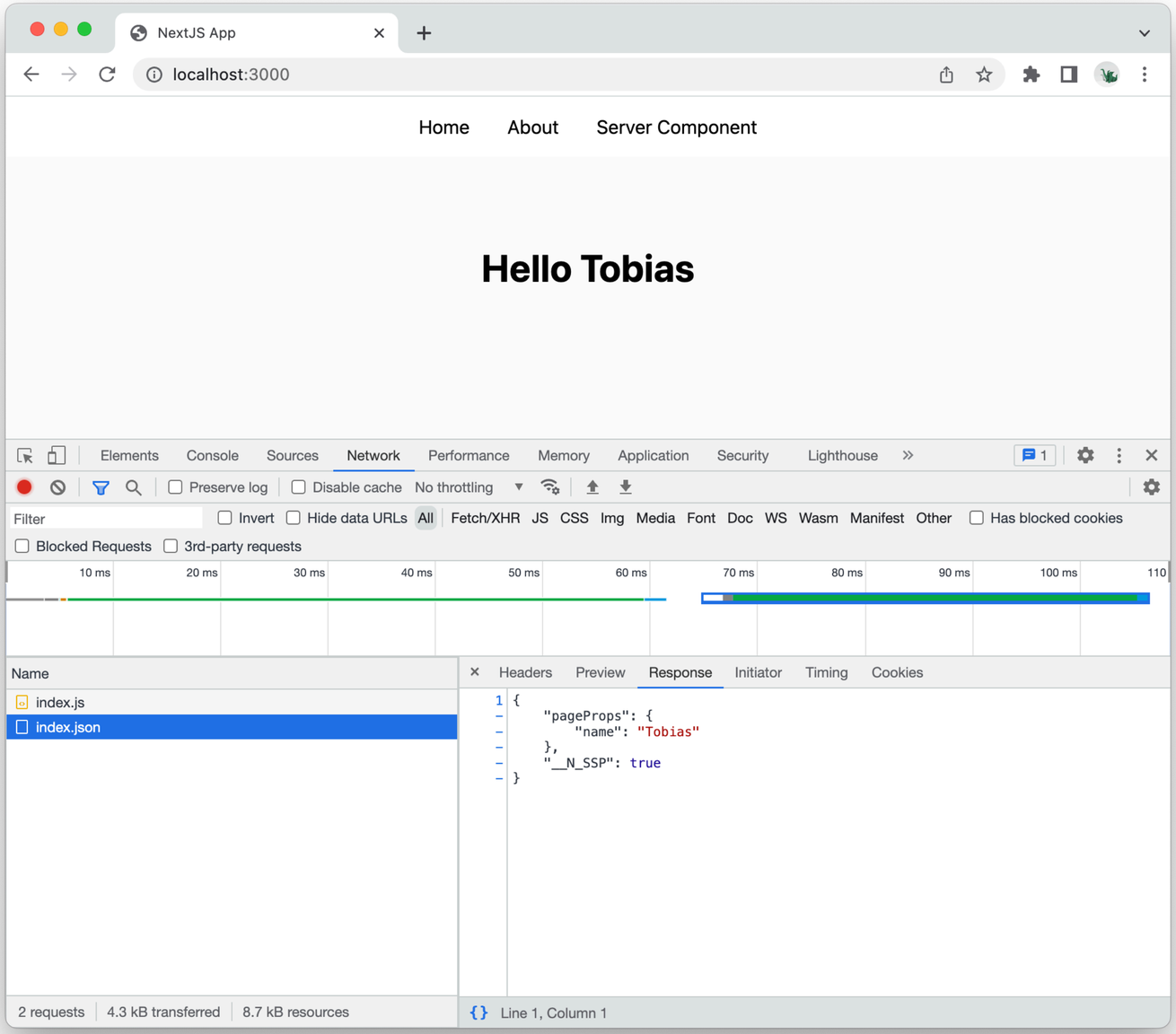
page.tsx (server side rendered page with data fetching on the server)

```tsx
type ServerProps = { name: string };

export async function getServerSideProps(
  ctx: GetServerSidePropsContext
): Promise<GetServerSidePropsResult<ServerProps>> {
  ctx.res.setHeader('Cache-Control', 'public, s-maxage=10, stale-while-revalidate=59');

  const data = await fetchJSON<{ name: string }>('/api/name');
  return { props: data };
}

export default function Page({ name }: ServerProps) {
  return (<>
    <Head>
      <title>{name} NextJS App</title>
    </Head>
    <h1>Hello {name}</h1>
  </>);
}
```

API data is fetched during server side rendering.

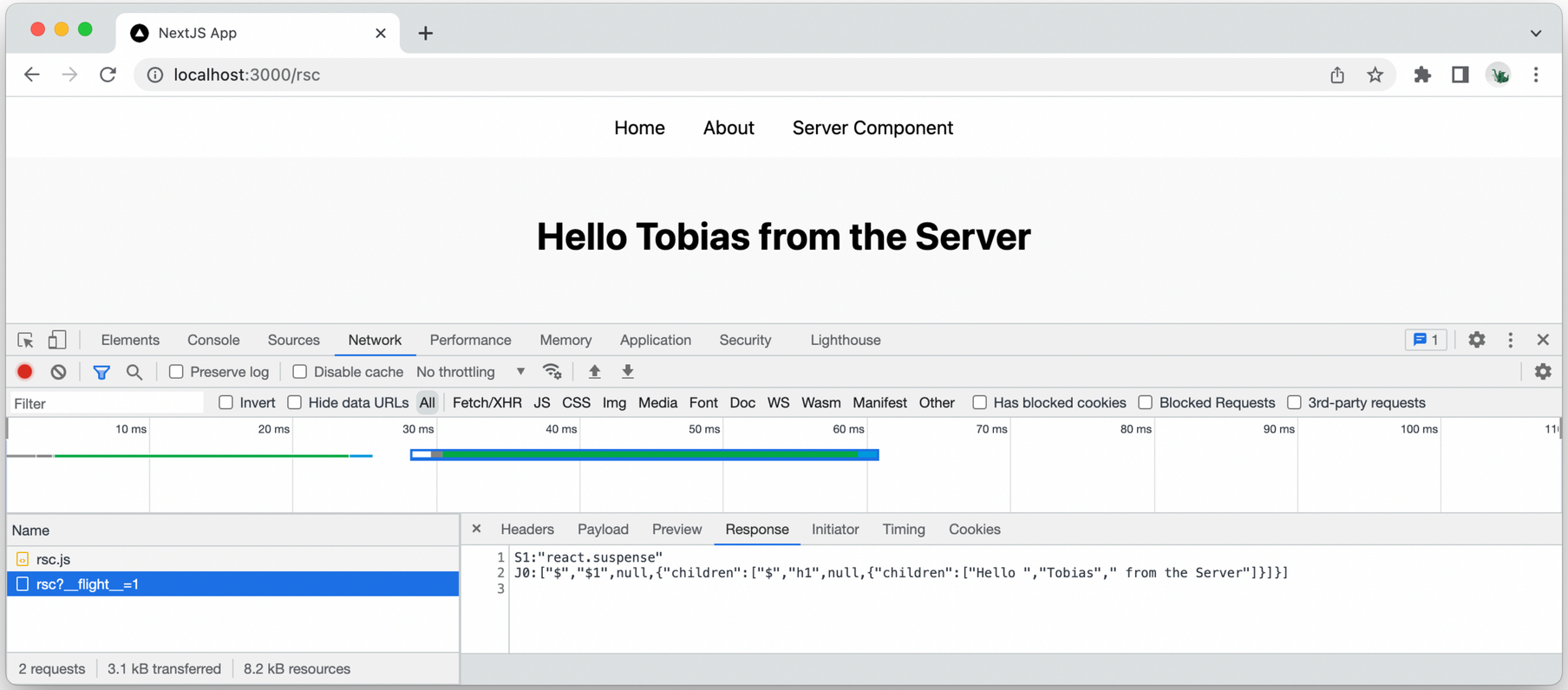


What happens during client side routing?



IN THE FUTURE:

Use React Server Components to Reduce Client Bundle Size



page.server.tsx (experimental react server component page that uses suspense data fetching hook)

```tsx
export default function ServerPage() {
  const { data, error } = useData<{ name: string }>('/api/name');
  if (error) return <div>{error}</div>;
  return <h1>Hello {data!.name} from the Server</h1>;
}
```