# 69. SSR Cookie forwarding

#ssr #cookie #nextjs #urql #graphql #fronent #backend #context

---

- See Why is it not working? for how we arrived here

---

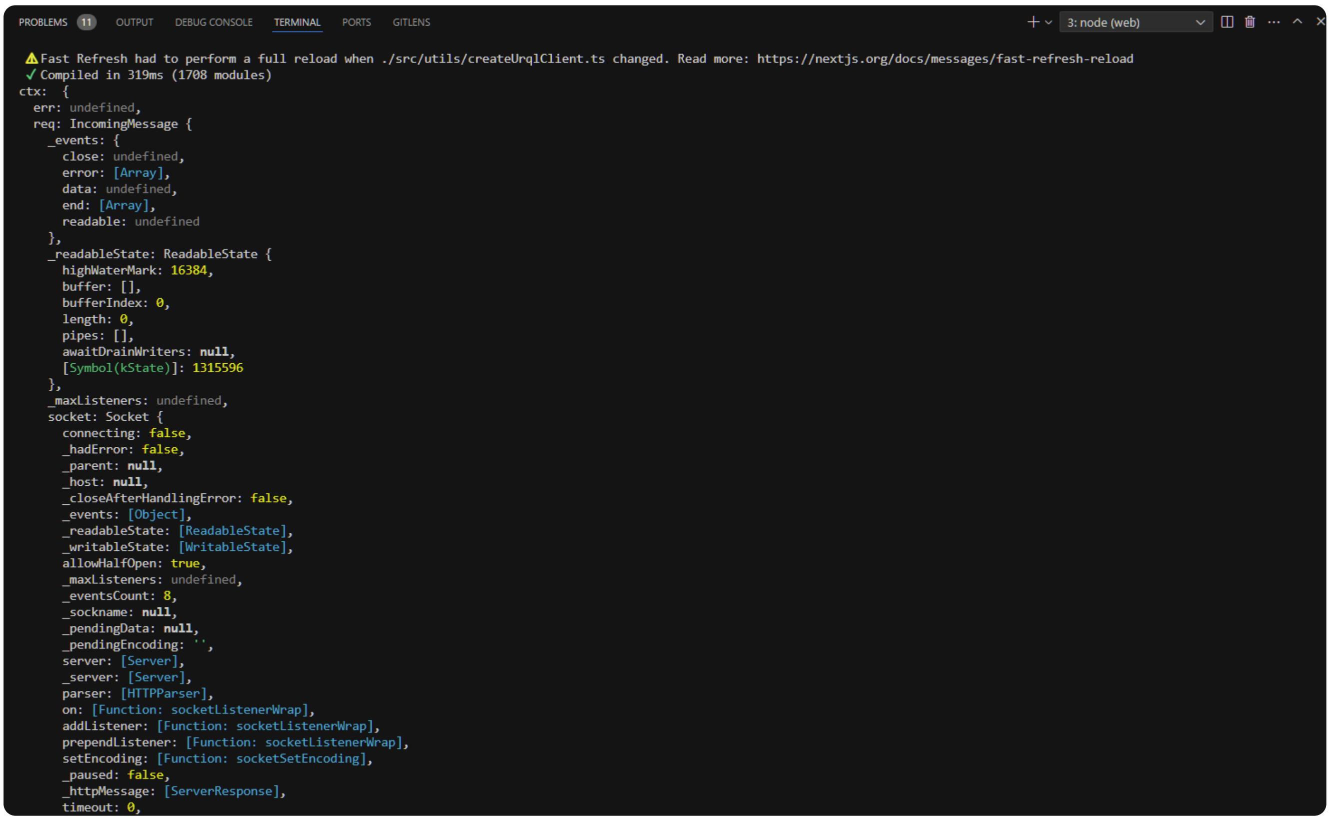
## Find the cookie in the next.js server

- In the createUrqlClient function, we can pass a ctx context object. So if we update the code as such:

/utils/createUrqlClient

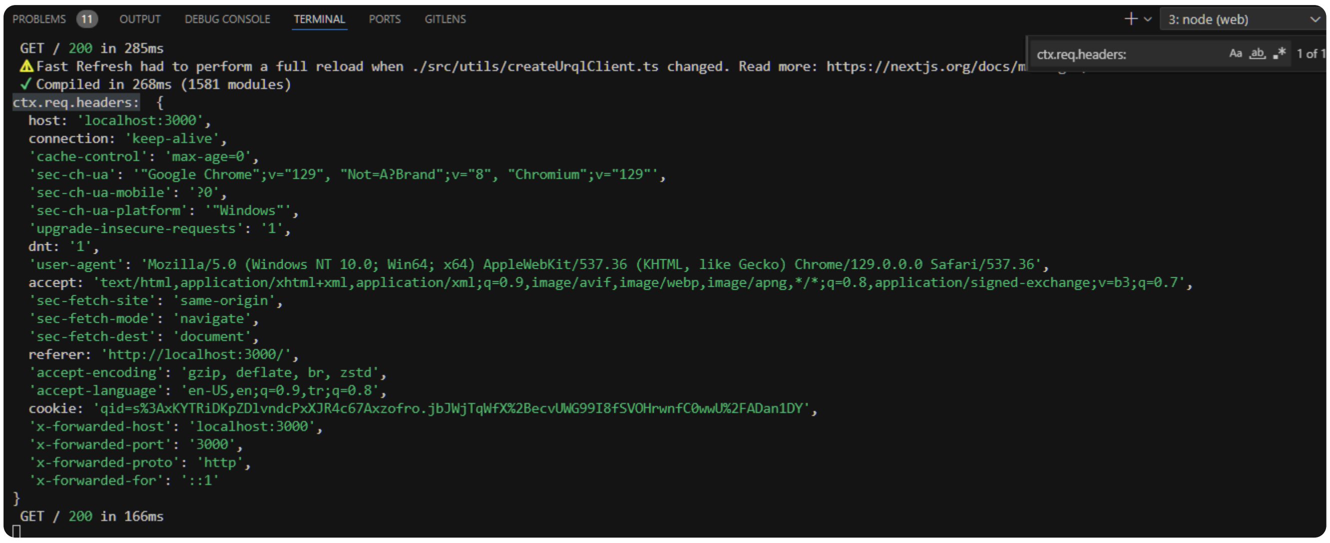
```
// this code runs both on the browser and the server
export const createUrqlClient = (ssrExchange: any, ctx: any) => {
  if (isServer()) { // we don't have the ctx object on the browser
     console.log(ctx)
   }
   return {
      // .... all the code remains the same .....
   }
}
```

We see that there's A LOT of stuff in the ctx object:



```
⚠Fast Refresh had to perform a full reload when ./src/utils/createUrqlClient.ts changed. Read more: https://nextjs.org/docs/messages/fast-refresh-reload
✓Compiled in 319ms (1708 modules)
ctx:  {
  err: undefined,
  req: IncomingMessage {
    _events: {
      close: undefined,
      error: [Array],
      data: undefined,
      end: [Array],
      readable: undefined
    },
    _readableState: ReadableState {
      highWaterMark: 16384,
      buffer: [],
      bufferIndex: 0,
      length: 0,
      pipes: [],
      awaitDrainWriters: null,
      [Symbol(kState)]: 1315596
    },
    _maxListeners: undefined,
    socket: Socket {
      connecting: false,
      _hadError: false,
      _parent: null,
      _host: null,
      _closeAfterHandlingError: false,
      _events: [Object],
      _readableState: [ReadableState],
      _writableState: [WritableState],
      allowHalfOpen: true,
      _maxListeners: undefined,
      _eventsCount: 8,
      _sockname: null,
      _pendingData: null,
      _pendingEncoding: '',
      server: [Server],
      _server: [Server],
      parser: [HTTPParser],
      on: [Function: socketListenerWrap],
      addListener: [Function: socketListenerWrap],
      prependListener: [Function: socketListenerWrap],
      setEncoding: [Function: socketSetEncoding],
      _paused: false,
      _httpMessage: [ServerResponse],
      timeout: 0,
```

- And if we scroll through this we see that there’s a req object and a res object in the ctx
- The cookie can be seen in req.headers (as well as req.rawHeaders and req,cookies.cookie)
- Note that in my implementation req.headers was not displayed when I `console.log` ’ged ctx. But when I `console.log` ’ged ctx.req.headers it displayed:



```
 GET / 200 in 285ms
 ⚠Fast Refresh had to perform a full reload when ./src/utils/createUrqlClient.ts changed. Read more: https://nextjs.org/docs/m
 ✓ Compiled in 268ms (1581 modules)
ctx.req.headers:  {
  host: 'localhost:3000',
  connection: 'keep-alive',
  'cache-control': 'max-age=0',
  'sec-ch-ua': '"Google Chrome";v="129", "Not=A?Brand";v="8", "Chromium";v="129"',
  'sec-ch-ua-mobile': '?0',
  'sec-ch-ua-platform': '"Windows"',
  'upgrade-insecure-requests': '1',
  dnt: '1',
  'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36',
  accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7',
  'sec-fetch-site': 'same-origin',
  'sec-fetch-mode': 'navigate',
  'sec-fetch-dest': 'document',
  referer: 'http://localhost:3000/',
  'accept-encoding': 'gzip, deflate, br, zstd',
  'accept-language': 'en-US,en;q=0.9,tr;q=0.8',
  cookie: 'qid=s%3AxKYTRiDKpZDlvndcPxXJR4c67Axzofro.jbJWjTqWfX%2BecvUWG99I8fSVOHrwnfC0wwU%2FADan1DY',
  'x-forwarded-host': 'localhost:3000',
  'x-forwarded-port': '3000',
  'x-forwarded-proto': 'http',
  'x-forwarded-for': '::1'
}
 GET / 200 in 166ms
```

- And more specifically, `console.log("cookie: ", ctx.req.headers.cookie)` gives:

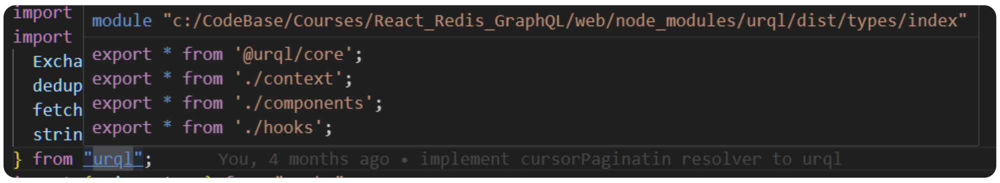
cookie: 'qid=s%3AxKYTRiDKpZDlvndcPxXJR4c67Axzofro.jbJWjTqWfX%2BecvUWG99I8fSVOHrwnfC0wwU%2FADan1DY'

- So all we want to do is to send this cookie from next.js to graphql

---

## Find headers field in fetchOptions()

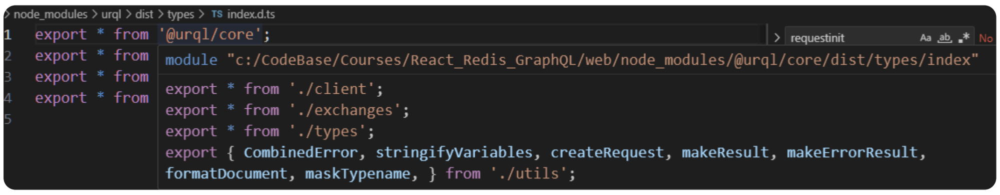
- First ctrl+click `“urql”` import



```
import
import
  Excha
  dedup
  fetch
  strin
} from "urql";        You, 4 months ago • implement cursorPaginatin resolver to urql
```

```
module "c:/CodeBase/Courses/React_Redis_GraphQL/web/node_modules/urql/dist/types/index"
export * from '@urql/core';
export * from './context';
export * from './components';
export * from './hooks';
```

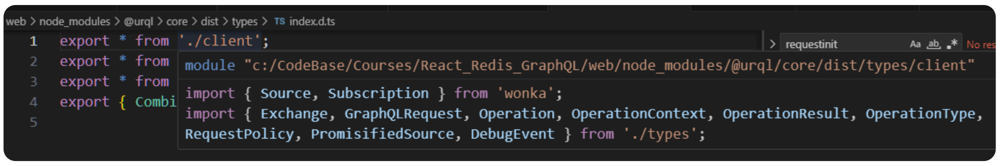
- Then ctrl+click `“@urql/core”` import



```
export * from '@urql/core';
export * from
export * from
export * from
```

```
module "c:/CodeBase/Courses/React_Redis_GraphQL/web/node_modules/@urql/core/dist/types/index"
export * from './client';
export * from './exchanges';
export * from './types';
export { CombinedError, stringifyVariables, createRequest, makeResult, makeErrorResult,
formatDocument, maskTypename, } from './utils';
```

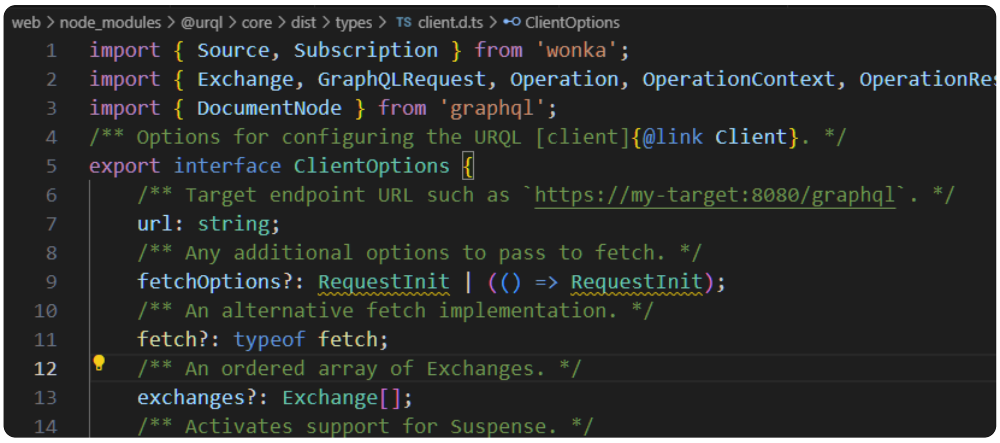
- Then ctrl+click `"./client"` import



```
web > node_modules > @urql > core > dist > types > TS index.d.ts
export * from './client';
export * from
export * from
export { Combi
```

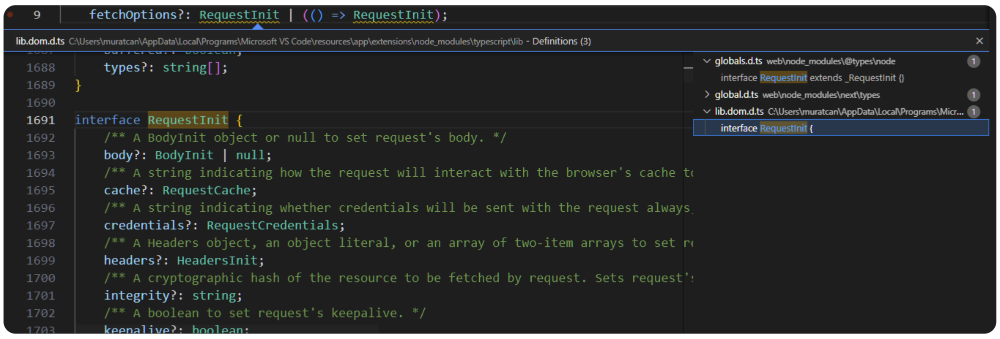
```
module "c:/CodeBase/Courses/React_Redis_GraphQL/web/node_modules/@urql/core/dist/types/client"

import { Source, Subscription } from 'wonka';
import { Exchange, GraphQLRequest, Operation, OperationContext, OperationResult, OperationType,
RequestPolicy, PromisifiedSource, DebugEvent } from './types';
```

- we see that in client.d.ts we have fetchOptions in ClientOptions



```
web > node_modules > @urql > core > dist > types > TS client.d.ts > ClientOptions
import { Source, Subscription } from 'wonka';
import { Exchange, GraphQLRequest, Operation, OperationContext, OperationRe
import { DocumentNode } from 'graphql';
/** Options for configuring the URQL [client]{@link Client}. */
export interface ClientOptions {
    /** Target endpoint URL such as `https://my-target:8080/graphql`. */
    url: string;
    /** Any additional options to pass to fetch. */
    fetchOptions?: RequestInit | (() => RequestInit);
    /** An alternative fetch implementation. */
    fetch?: typeof fetch;
    /** An ordered array of Exchanges. */
    exchanges?: Exchange[];
    /** Activates support for Suspense. */
```

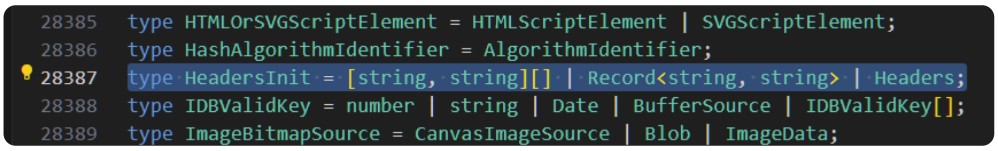
- ctrl+click on RequestInit to see that the definition in lib.dom.d.ts has the headers field in it



```
    fetchOptions?: RequestInit | (() => RequestInit);
lib.dom.d.ts C:\Users\muratcan\AppData\Local\Programs\Microsoft VS Code\resources\app\extensions\node_modules\typescript\lib - Definitions (3)
    types?: string[];
}

interface RequestInit {
    /** A BodyInit object or null to set request's body. */
    body?: BodyInit | null;
    /** A string indicating how the request will interact with the browser's cache tc
    cache?: RequestCache;
    /** A string indicating whether credentials will be sent with the request always
    credentials?: RequestCredentials;
    /** A Headers object, an object literal, or an array of two-item arrays to set re
    headers?: HeadersInit;
    /** A cryptographic hash of the resource to be fetched by request. Sets request's
    integrity?: string;
    /** A boolean to set request's keepalive. */
    keepalive?: boolean;
```

```
globals.d.ts web\node_modules\@types\node
    interface RequestInit extends _RequestInit {}
global.d.ts web\node_modules\next\types
lib.dom.d.ts C:\Users\muratcan\AppData\Local\Programs\Micr...
    interface RequestInit {
```

- And if we ctrl+click HeadersInit type we also see it's definition:



```
type HTMLOrSVGScriptElement = HTMLScriptElement | SVGScriptElement;
type HashAlgorithmIdentifier = AlgorithmIdentifier;
type HeadersInit = [string, string][] | Record<string, string> | Headers;
type IDBValidKey = number | string | Date | BufferSource | IDBValidKey[];
type ImageBitmapSource = CanvasImageSource | Blob | ImageData;
```

- Note that if it didn't have the headers field in it, we could've added it ourselves

---

## Send cookie to backend via fetchOptions()

- Now we just send the cookie to the backend in the headers field of fetchOptions()
- Note that we send it as an object

**/utils/createUrqlClient.ts**

```
// this code runs both on the browser and the server
export const createUrqlClient = (ssrExchange: any, ctx: any) => {
  let cookie = "";
  if (isServer()) {
    // we don't have the ctx object on the browser
    cookie = ctx.req.headers.cookie;
  }

  return {
    url: "http://localhost:4000/graphql",
    fetchOptions: {
      credentials: "include" as const,
      headers: cookie ? { cookie } : undefined,
    },
```

---

## Conclusion

- And now everything is working. The next.js server sends the cookie in the header and the graphql api receives the cookie retrieves the session.userId from it and even when we refresh the page the the userVote value is sent to frontend

---

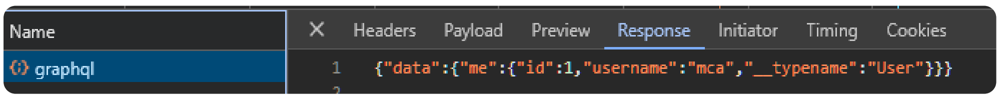
## (Optional) Enable server-side meQuery()

- We were preventing the execution of meQuery() on the server through these lines. So a request was made from the client for it,

**/components/NavBar.tsx**

```
const [{ data, fetching }] = useMeQuery({
  pause: isServer(), // this will prevent the query from running on the server
(there's no cookie on the server to look for)
});
```

- Request after a refresh:



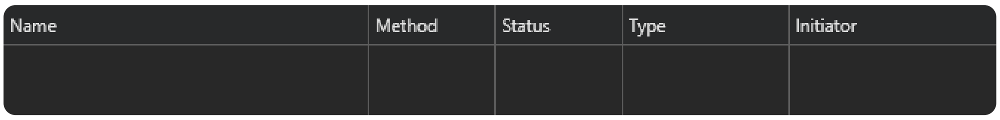
```
{"data":{"me":{"id":1,"username":"mca","__typename":"User"}}}
```

- We can actually enable it now since the server also receives the cookie. This way everything is done on the server side and we are not making any requests to the server from client side when we refresh the page

**/components/NavBar.tsx**

```
const [{ data, fetching }] = useMeQuery();
```

- No more requests after a refresh:



| Name | Method | Status | Type | Initiator |
|---|---|---|---|---|

- In the tutorial Ben leaves it client-side with `pause: isServer()`