29. Server Side Rendering with NextJS and urql

#nextjs #urql #ssr #next-urql #frontend

Install packages

- This is also a good time to format the code and take the creation of urql client out of the _app.tsx, and put it into a utility function
- formidable.com/open-source/urql/docs/advanced/server-side-rendering/#nextjs
- What we're doing here is **legacy version** of ssr with urql! Check the docs for the modern implementation

```
yarn add next-urql react-is isomorphic-unfetch
```

Usage of withUrqlClient()

- This is how withUrqlClient() function is used to set up urql provider on the page and the `{ ssr: true }` is added as second parameter to enable SSR for the page:

```
export default withUrqlClient(ssrExchange => ({
  url: 'http://localhost:3000/graphql',
  exchanges: [cacheExchange, ssrExchange, fetchExchange],
}), { ssr: true })(Index);
```

Utility function to create the urql client

- So basically, we copy the code from createClient() in _app.tsx and implement the utility function createUrqlClient() function to pass into wirhUrqlClient() :
- **Note that we add** ssrExchange between cacheExchange and fetchExchange

/utils/createUrqlClient.ts

```
import { dedupExchange, fetchExchange } from "urql";
import {
  LogoutMutation,
  MeQuery,
  MeDocument,
  LoginMutation,
```

```
  RegisterMutation,
} from "../generated/graphql";
import { cacheExchange } from "@urql/exchange-graphcache";
import { betterUpdateQuery } from "./betterUpdateQuery";

export const createUrqlClient = (ssrExchange: any) => ({
  url: "http://localhost:4000/graphql",
  fetchOptions: {
    credentials: "include" as const,
  },
  exchanges: [
    dedupExchange,
    cacheExchange({
      // this will update the cache everytime the defined mutations are run run
      updates: {
        Mutation: {
          logout: (result, args, cache, info) => {
            betterUpdateQuery<LogoutMutation, MeQuery>(
              cache,
              { query: MeDocument }, // e.g. MeQuery's input type is MeDocument
              result,
              () => ({ me: null }) // clear the query
            );
          },
          login: (result, args, cache, info) => {
            // cache.updateQuery({ query: MeDocument }, (data: MeQuery) => { })
            betterUpdateQuery<LoginMutation, MeQuery>(
              cache,
              { query: MeDocument },
              result,
              (r, q) => {
                if (r.login.errors) {
                  return q; // return the current query if there's error
                } else {
                  return {
                    me: r.login.user, // return the user info received from
successful login
                  };
                }
              }
```

```
            );
          },
          register: (result, args, cache, info) => {
            betterUpdateQuery<RegisterMutation, MeQuery>(
              cache,
              { query: MeDocument },
              result,
              (r, q) => {
                if (r.register.errors) {
                  return q; // return the current query if there's error
                } else {
                  return {
                    me: r.register.user, // return the user info received from
successful register
                  };
                }
              }
            );
          },
        },
      },
    }),
    ssrExchange,
    fetchExchange,
  ],
});
```

Setting a page for urql provider

- Set login.tsx and register.tsx use urql provider (but not for SSR)

pages/login.tsx

```
export default withUrqlClient(createUrqlClient)(Login);
```

pages/register.tsx

```
export default withUrqlClient(createUrqlClient)(Register);
```

Setting a page for SSR

- Set index.tsx to use urql provider and to be rendered with SSR

pages/index.tsx

```
export default withUrqlClient(createUrqlClient, { ssr: true })(Index);
```