56. Pagination for Posts - Urql Client

#pagination #urql #typescript #frontend #exchange #graphql #graphcache #cache #resolver #error

---

Implement cursorPagination() resolver to pass into cacheExchange

- Right now, frontend is pulling and displaying the next batch of posts, but **the new batch replaces the previous batch.** We'll fix it so new batches are appended to the previous batches
- urql has a simplePagination() function that can be used when doing pagination with limit and offset, as well as a relay pagination version.

formidable.com/open-source/urql/docs/graphcache/computed-queries/#simple-pagination

- However we're using limit and cursor for our pagination, so we'll implement cursorPagination() function by altering the simplePagination()

github.com/urql-graphql/urql/blob/a7d2b21f5c1d456709ac9c520e9132ba6e2e857e/exchanges/graphcache/src/extras/simplePagination.ts

- More info on `cache.resolve()` → formidable.com/open-source/urql/docs/graphcache/computed-queries/

/utils/createUrqlClient.ts

```
import { Resolver, cacheExchange } from "@urql/exchange-graphcache";
import { Exchange, dedupExchange, fetchExchange, stringifyVariables } from
"urql";

const cursorPagination = (): Resolver => {
  return (_parent, fieldArgs, cache, info) => {
    const { parentKey: entityKey, fieldName } = info;
    // entityKey = Query, fieldName = posts, since we plug this Resolver into
cacheExchange like that (see below)

    const allFields = cache.inspectFields(entityKey);
    // Retrieves the fields of the cached queries - cache can contain different
queries so we will filter them
    // allFields: [
    //   {
    //     fieldKey: 'posts({"limit":10})',
    //     fieldName: 'posts',
    //     arguments: { limit: 10 }
```

```
    //   }
    // ]

    // filter allFields to get only the field infos related to the query we want
to work on
    const fieldInfos = allFields.filter((info) => info.fieldName === fieldName);

    const size = fieldInfos.length;
    if (size === 0) {
      return undefined;
    }

    // create a new fieldKey to check if the data is in the cache and return it
from cache, updating cache if needed
    // fieldArgs is the arguments passed into the current query, e.g. { limit:
10, cursor: "159734454958" }
    // fieldKey will have the form 'posts({limit:10,cursor:"159734454958"})' as
seen in allFields
    // so we use fieldName and fieldArgs to construct the most recent fieldKey
and check if it is in the cache
    const fieldKey = `${fieldName}(${stringifyVariables(fieldArgs)})`;

    const isItInTheCache = cache.resolve(
      cache.resolveFieldByKey(entityKey, fieldKey) as string,
      "posts"
    );

    info.partial = !isItInTheCache; // reload if new results are not in the cache

    // cache.readQuery() --> This will call the resolver again and enter an
infinite loop
    // so we use this:
    const results: string[] = [];
    let hasMore = true;
    fieldInfos.forEach((fi) => {
      const key = cache.resolveFieldByKey(entityKey, fi.fieldKey) as string;
      const data = cache.resolve(key, "posts") as string[];
      if (!(cache.resolve(key, "hasMore") as boolean)) {
        hasMore = false;
      }
```

```
      results.push(...data);
    });

    return {
      __typename: "PaginatedPosts", // NOT PUTTING THIS WAS CAUSING AN ERROR
graphql.tsx:374 Invalid resolver value: The field at `Query.posts({"limit":10})`
is a scalar (number, boolean, etc), but the GraphQL query expects a selection set
for this field.
      hasMore,
      posts: results,
    };
  };
};
```

Insert the cursorPagination() function into cacheExchange

- Here we add cursorPagination() as a client-side resolver to the cacheExchange so that it will be executed everytime the posts query is run

/utils/createUrqlClient.ts

```
cacheExchange({
  // to circumvent the error: Invalid key: The GraphQL query at the field at
`Query.posts({"limit":10})` has a
  // selection set, but no key could be generated
  keys: {
    PaginatedPosts: () => null,
  },
  resolvers: {
    Query: {
      // this will run whenever the posts query is run
      // name of it matches what we used in posts.graphql
      posts: cursorPagination(),
    },
  },
  updates: { ... }
```