19. GraphQL Mutation - register / graphql-codegen for simpler urql

#urql #graphql #graphql-codegen #mutation #frontend

We set up urql usage and register() mutation in 17. Urql for GraphQL requests but the response for the register() call is <any> and we don't want that

```
<Wrapper variant="small">
  <Formik
    initialV
    onSubmit
      const response = await register(values);
    }}
```

```
const response: OperationResult<any>
'response' is declared but its value is never read. ts(6133)
Quick Fix... (⌘.)
```

Install new packages

We will use `graphql-code-generator` (graphql-code-generator.com)

```
yarn add -D @graphql-codegen/cli
```

```
yarn graphql-codegen init
```

1. The application is of type React
2. The schema is at http://localhost:4000/graphql
3. The operations and fragments will be sotred at src/graphql/**/*.graphql
4. Plugins: TypeScript and TypeScript Operations
5. Output location: The default value should be fine
6. No instrospection file needed
7. Name of the config file codegen.,yml
8. Name of the script to run codegen: gen

After the codegen.,yml config file is created, edit it and add "typescript-urql" to plugins

codegen.yml

```yaml
overwrite: true
schema: "http://localhost:4000/graphql"
```

```
documents: "./src/graphql/**/*.graphql"
generates:
  src/generated/graphql.tsx:
    plugins:
      - "typescript"
      - "typescript-operations"
      - "typescript-urql"
```

- Install the package:

```
yarn add -D @graphql-codegen/typsecsript-urql
```

Add the mutation to generate

- Create the folders src/graphql/mutations and src/graphql/queries
- Install the GraphQLfor VSCode (by Kumar Harsh) extension if it's not already installed to get graphql syntaz highlighting
- And now create register.graphql file in mutations folder and copy our register mutation into it as such:
- Note that instead of giving parameters separately to Register() we make use of the userNamePasswordInput @InputType we defined in 8. Resolver - user.ts / Mutation - register()

/graphql/mutations/register.graphql

```
mutation Register($options: userNamePasswordInput!) {
  register(options: $options) {
    errors {
      field
      message
    }
    user {
      id
      username
    }
  }
}
```

Generate TypeScript code for the Mutation

```
yarn gen
```

- This will run the generator and place the generated TypeScript code in the src/generated/graphql.tsx file
- The most important bit here will be the **useRegisterMutation** custom hook, at the end of the file:

generated/graphql.tsx

```
export function useRegisterMutation() {
  return Urql.useMutation<RegisterMutation, RegisterMutationVariables>
(RegisterDocument);
};
```

Use the custom hook for the request

- Now we can update register.tsx to use this custom hook that was generated

pages/register.tsx

```
import React from "react";
import { Form, Formik } from "formik";
import { Button, Box } from "@chakra-ui/react";
import { Wrapper } from "../components/Wrapper";
import { InputField } from "../components/InputField";
import { useRegisterMutation } from "../generated/graphql";

const Register: React.FC<registerProps> = ({}) => {

  const [, register] = useRegisterMutation();

  return (
    <Wrapper variant="small">
      <Formik // initalValues, onSubmit, setErrors provided by Formik, values is
inferred from initialValues
        initialValues={{ username: '', email: '', password: '' }}
        onSubmit={(values) => {
          return register({ options: values })
        }}
```

```
      >
        {(
          { isSubmitting } // isSubmitting is provided by Formik
        ) => (
          <Form>
            <InputField
              name="username"
              label="Username"
              placeholder="Username"
            />
            <Box mt={4}>
              <InputField
                name="email"
                label="Email"
                placeholder="Email"
                type="email"
              />
            </Box>
            <Box mt={4}>
              <InputField
                name="password"
                label="Password"
                placeholder="Password"
                type="password"
              />
            </Box>
            <Button mt={4} type="submit" isLoading={isSubmitting} color="teal">
              Register
            </Button>
          </Form>
        )}
      </Formik>
    </Wrapper>
  );
};
```

The return type of the custom hook is well-defined

- Now, the type of the response object that is returned from the register() call is not <any> but <RegisterMutation>:

```
const Register: React.FC<registerProps> = ({}) => {
  const [, register] = useRegisterMutation();
  return (
    <Wrapper variant="small">
      <Formik
        initialV
        onSubmit
          const response = await register(values);
        }}
```

```
const response: OperationResult<RegisterMutation>
'response' is declared but its value is never read. ts(6133)
Quick Fix... (⌘.)
```

- And it is possible to look into the response object, since the IDE it knows what's in there:

```
  response.data.register.
  }}
>
  {({ isSubmitting }) => (
```

```
__typename?
errors?          (property) err…
user?
```

Conclusion

- Now everytime we want to add a new graphql query or mutation, we can put the code in the graphql/mutations or graphql/queries folder and easily generate the custom hook for that mutaion or query with `yarn gen`