17. Authentication

Trouble shooting

graphQL과 Apollo의 버전이 맞지 않는 문제

```
 Generating query files with 'typescript' target
    → spurious results.
    Error: Cannot use GraphQLScalarType "String" from another module or
realm.

    Ensure that there is only one instance of "graphql" in the
node_modules
    directory. If different versions of "graphql" are the dependencies of
other
    relied on modules, use "resolutions" to ensure only one version is
installed.

    https://yarnpkg.com/en/docs/selective-version-resolutions

    Duplicate "graphql" modules cannot be used at the same time since
different
    versions may have different capabilities and behavior. The data from
one
    version used in the function from another could produce confusing and
    spurious results.
```

- `npm run apollo:codegen` 명령어 실행 시 위와 같은 에러 발생
  - apollographql의 버전과 graphql의 버전을 맞춰주면 된다.(아직 apollographql의 일부 패키지들이 업데이트가 되지 않아 graphql의 최신버전과 맞지 않는듯 함)
  - graphQL의 버전을 15.8.3으로 낮추고, 문제의 패키지 버전을 graphql의 버전과 강제로 맞추도록 함.
  - package.json에 글큐 버전 변경하여 `npm install` 후 `npm uninstall apollo`로 기존 패키지 삭제.
  - package.json에 아래 내용 추가

```
"overrides": {
  "@apollo/client": {
    "react": "$react"
  },
  "@apollographql/graphql-language-service-interface": {
    "graphql": "$graphql"
  },
  "@apollographql/graphql-language-service-parser": {
    "graphql": "$graphql"
  },
  "@apollographql/graphql-language-service-types": {
    "graphql": "$graphql"
  },
```

```
    "@apollographql/graphql-language-service-utils": {
      "graphql": "$graphql"
    }
  },
```

- `npm install apollo`로 아폴로 재설치

CORS 문제

```
Access to fetch at 'http://localhost:3030/' from origin
'http://localhost:3000' has been blocked by CORS policy: Response to
preflight request doesn't pass access control check: No 'Access-Control-
Allow-Origin' header is present on the requested resource. If an opaque
response serves your needs, set the request's mode to 'no-cors' to fetch
the resource with CORS disabled.
```

위와 같이 CORS 문제가 발생하여 백엔드의 main.ts에 `app.enableCors();`로 cors 설정을 추가하였다.

내용 정리

codegen을 이용한 DTO 자동 생성

```
const LOGIN_MUTATION = gql`
  mutation loginMutation($email: String!, $password: String!) {
    login(input: { email: $email, password: $password }) {
      ok
      token
      error
    }
  }
`;
```

위와 같이 mutation을 만들고 codegen을 실행하면

> codegen 은 `apollo client:codegen src/__generated__ --target=typescript --outputFlat`로 실행한다.(package.json에 scripts로 등록)

```
export interface loginMutation_login {
  __typename: "LoginOutput";
  ok: boolean;
  token: string | null;
  error: string | null;
}

export interface loginMutation {
  login: loginMutation_login;
}
```

```
export interface loginMutationVariables {
  email: string;
  password: string;
}
```

이렇게 DTO를 자동 생성해준다.👉 개발시 위 타입을 사용하여 자동으로 Type에 보호받을 수 있다. mutation의 Input까지 보호받기 위해서는

```
const LOGIN_MUTATION = gql`
  mutation loginMutation($loginInput: LoginInput!) {
    login(input: $loginInput) {
      ok
      token
      error
    }
  }
`;
```

위와 같이 Input Type을 적고 codegen을 실행하면 백엔드 graphQL의 스키마에 따라 아래처럼 새 타입이 생긴다.

```
export interface LoginInput {
  email: string;
  password: string;
}
```

따라서 `loginMutation` 실행 시 아래와 같이 변수로 LoginInput 타입을 넘겨줘야 한다.

```
const [loginMutation, { data }] = useMutation<
    loginMutation,
    loginMutationVariables
  >(LOGIN_MUTATION);
const onSubmit = () => {
    const { email, password } = getValues();
    loginMutation({
      variables: {
        loginInput: {
          email,
          password,
        },
      },
    });
  };
```

이제 백엔드에서 DTO를 수정하면 프론트의 타입스크립트가 해당 타입을 지적해 줄 것임.(버그 최소화 가능)

React Helmet

- 페이지별로 타이틀을 설정할 수 있는 라이브러리
- 설치 방법

```
npm i react-helmet
npm i --save-dev @types/react-helmet
```

- 페이지별로 아래와같이 추가하여 설정 가능

```
<Helmet>
  <title>Create Account | Nuber eats</title>
</Helmet>
```

React Helmet Async

- react-helmet이 비동기 처리에 문제가 생길 수 있음
  - thread-safe하지 않은 react-side-effect에 의존. 여러 쓰레드로부터 동시에 접근이 이루어져도 프로그램에 문제가 생기지 않음.
- `react-helmet-async` 설치 후 index.tsx에 `<HelmetProvider>` 추가하여 사용.

로그인 구현 방식

- 로그인 뮤테이션 호출 후 성공 여부에 따라 local storage에 토큰 저장.
- login.tsx

```
  const onCompleted = (data: loginMutation) => {
  const {
    login: { error, ok, token },
  } = data;
  if (ok && token) {
    localStorage.setItem(LOCALSTORAGE_TOKEN, token);
    authToken(token);
    isLoggedInVar(true);
  }
};

const [loginMutation, { data: loginMutationResult, loading }] =
useMutation<
  loginMutation,
  loginMutationVariables
>(LOGIN_MUTATION, {
  onCompleted,
});
```

- apollo.ts에서 로컬스토리지 토큰값 설정 여부를 받아와서 로그인 여부 확인.

```
import { ApolloClient, InMemoryCache, makeVar } from "@apollo/client";
import { LOCALSTORAGE_TOKEN } from "./constant";

const token = localStorage.getItem(LOCALSTORAGE_TOKEN);

export const isLoggedInVar = makeVar(Boolean(token));
export const authToken = makeVar(token);

export const client = new ApolloClient({
  uri: "http://localhost:3030/graphql",
  cache: new InMemoryCache({
    typePolicies: {
      Query: {
        fields: {
          isLoggedIn: {
            read() {
              return isLoggedInVar();
            },
          },
          token: {
            read() {
              return authToken();
            },
          },
        },
      },
    },
  }),
});
```

request header 설정

- apollo client의 `createHttpLink()`와 `setContext()` 활용

> apollo link 는 클라이언트와 서버 사이의 데이터 흐름을 사용자가 정의할 수 있게 해준다.(http request header 설정 가능)

- apollo.ts

```
const httpLink = createHttpLink({
  uri: "http://localhost:3030/graphql",
});

const authLink = setContext((_, { headers }) => {
  return {
    headers: {
      ...headers,
      "x-jwt": token || "",
    },
  };
```

```
});

export const client = new ApolloClient({
  link: authLink.concat(httpLink),
  ...
});
```

Router 설정(react-router-dom v5)

- path와 완전히 일치한 url로 라우팅하기 위해 `exact` 사용
- `<Switch></Switch>` 안에는 `<Route></Route>`태그만 사용가능하여 fragment로 감싸주면 에러 발생.

```
import { gql, useQuery } from "@apollo/client";
import React from "react";
import {
  BrowserRouter as Router,
  Redirect,
  Route,
  Switch,
} from "react-router-dom";
import { Header } from "../components/header";
import { meQuery } from "../__generated__/meQuery";
import { Restaurants } from "./client/restaurants";

const ClientRoutes = [
  <Route path="/" exact>
    <Restaurants />
  </Route>,
];

const ME_QUERY = gql`
  query meQuery {
    me {
      id
      email
      role
      verified
    }
  }
`;

export const LoggedInRouter = () => {
  const { data, loading, error } = useQuery<meQuery>(ME_QUERY);

  if (!data || loading || error) {
    return (
      <div className="h-screen flex justify-center items-center">
        <span className="font-medium text-xl tracking-wide">Loading...
</span>
      </div>
    );
```

```
  }
  return (
    <Router>
      <Header />
      <Switch>
        {data.me.role === "Client" && ClientRoutes}
        <Redirect to="/" />
      </Switch>
    </Router>
  );
};
```

Apollo cache를 활용한 hook

```
import { gql, useQuery } from "@apollo/client";
import { meQuery } from "../__generated__/meQuery";

const ME_QUERY = gql`
  query meQuery {
    me {
      id
      email
      role
      verified
    }
  }
`;
export const useMe = () => {
  return useQuery<meQuery>(ME_QUERY);
};
```

- 위와같이 여러 컴포넌트 안에서 필요한 정보는 hook으로 작성하여 사용하는 것이 편하다.
- 여러개의 컴포넌트 속에서 필요한 정보의 경우 redux와 같은 상태관리툴을 사용하는데, apollo client의 경우 위와같이 hook을 호출하여 사용하면 브라우저의 메모리에 캐시로 저장되어 실제로 graphql을 호출하지 않고 정보를 가져올 수 있다.

  > 여러 컴포넌트에서 `useMe()`를 호출하지만 실제로 Network탭에서 graphql이 호출된 것을 살펴보면 최초 1회만 호출된 것을 확인할 수 있다.

18. USER PAGES

Trouble shooting

내용 정리

Apollo Client 캐시 직접 수정

- 사용자의 이메일이 인증되면 client cache의 verified가 변경되어야 한다.
- 이 부분은 apollo client의 fragment를 이용하여 graphQL 쿼리를 던지지 않고 Apollo Client 의 캐시를 수정할 수 있다.

```
const onCompleted = (data: verifyEmail) => {
  const {
    verifyEmail: { ok },
  } = data;
  if (ok && userData?.me.id) {
    client.writeFragment({
      id: `User:${userData?.me.id}` + "",
      fragment: gql`
        fragment VerifiedUser on User {
          verified
        }
      `,
      data: {
        verified: true,
      },
    });
  }
};
const [verifyEmail, { loading: verifyingEmail }] = useMutation<
  verifyEmail,
  verifyEmailVariables
>(VERIFT_EMAIL_MUTATION, {
  onCompleted,
});
```

- email update시에도 똑같이 object의 id를 이용하여 캐시에 새로운 email과 verified 를 업데이트.

refetch를 이용한 캐시 수정

- fragment를 이용한 방법은 이메일 수정이 성공하면 user 정보를 재호출하지 않고 직접 캐시를 수정하는 방법이다.
- refetch를 이용하면 수정 후 userMe쿼리의 refetch를 실행하여 새 정보를 다시 호출해오고, 캐시도 함께 업데이트 해준다. 이 경우 api를 다시 한번 더 호출하게 되지만 구현은 더 간단하다.(refetch는 promise를 반환한다.)
- 위에서 fragment를 사용하여 처리한 부분은 아래와 같이 처리할 수도 있다.

```
const { data: userData, refetch } = useMe();
const onCompleted = async (data: verifyEmail) => {
  const {
    verifyEmail: { ok },
  } = data;
  if (ok && userData?.me.id) {
    await refetch();
  }
};
```

19. Restaurants

Trouble shooting

N/A

내용 정리

Pagination 구현 방식

- 현재 page 정보를 state로 갖고 사용자가 페이지 이동 화살표를 클릭할 때마다 변경
  - page 상태가 바뀌면 해당 페이지의 내용들을 api로 다시 호출(같은 페이지 재 호출시 cache의 값 가져옴. network 탭을 확인해보면 graphql 쿼리를 다시 호출하지 않는다.) 참고

> Whenever Apollo Client fetches query results from your server, it automatically caches those results locally. This makes later executions of that same query extremely fast.

```
const [page, setPage] = useState(1);
const { data, loading } = useQuery<
  restaurantsPageQuery,
  restaurantsPageQueryVariables
>(RESTAURANTS_QUERY, {
  variables: {
    input: {
      page: page,
    },
  },
});
const onNextPageClick = () => setPage((current) => current + 1);
const onPrevPageClick = () => setPage((current) => current - 1);
```

- page가 1이 아닌 경우에만 뒤로가기 버튼 노출, page가 restaurants 응답의 토탈 페이지(끝)이 아닐때만 다음 버튼 노출.

```
<div className="grid grid-cols-3 text-center max-w-md items-center mx-
auto mt-10">
  {page > 1 ? (
    <button
    onClick={onPrevPageClick}
    className="focus:outline-none font-medium text-2xl"
    >
      &larr;
    </button>
  ) : (
    <div></div>
  )}
  <span>
    {page} of {data?.restaurants.totalPages}
```

```
  </span>
    {page !== data?.restaurants.totalPages && (
    <button
      onClick={onNextPageClick}
      className="focus:outline-none font-medium text-2xl"
    >
      &rarr;
    </button>
  )}
</div>
```

조금 더 나은?코드를 위한 Restaurant 컴포넌트 분리

- 참고하기.

> 평소 워낙 코드가 더러운편,,,으로 좋은 코드 기록용입니다.🤦 😱

- components/restaurant.tsx

```
import React from "react";

interface IRestaurantProps {
  coverImg: string | null;
  name: string;
  categoryName?: string;
  id: string;
}

export const Restaurant: React.FC<IRestaurantProps> = ({
  coverImg,
  name,
  categoryName,
}) => (
  <div className="flex flex-col">
    <div
      style={{
        backgroundImage: `url(${coverImg})`,
      }}
      className="bg-cover bg-center mb-3 py-28"
    ></div>
    <h3 className="text-xlg font-medium">{name}</h3>
    <span className="border-t mt-2 py-2 text-xs opacity-50 border-
gray-300">
      {categoryName}
    </span>
  </div>
);
```

- pages/client/restaurants.tsx의 `Restaurant` 컴포넌트 사용부분

```
<div className="grid mt-16 grid-cols-3 gap-x-5 gap-y-10">
  {data?.restaurants.results?.map((restaurant) => (
    <Restaurant
      id={restaurant.id + ""}
      coverImg={restaurant.coverImg}
      name={restaurant.name}
      categoryName={restaurant.category?.name}
    />
  ))}
</div>
```

search에 react-router-dom(v5) 적용

- `useHistory()`활용
- `search`로 검색어를 push하면 path 뒤에 붙음.(URL query string)
- `state`로 전달하면 브라우저의 메모리(stack)에 저장되어 뒤로가기 누를 때 해당페이지의 state를 기억하지만 검색 결과 페이지를 url로 공유하는 경우 검색어가 전달되지 않아 여기선 `search`를 쓴다.

```
const { register, handleSubmit, getValues } = useForm<IFormProps>();
const history = useHistory();
const onSearchSubmit = () => {
  const { searchTerm } = getValues();
  history.push({
    pathname: "/search",
    search: `?term=${searchTerm}`,
  });
};
```

```
<form
  onSubmit={handleSubmit(onSearchSubmit)}
  className="bg-gray-800 w-full py-40 flex items-center justify-
center"
>
```

- query가 없는 경우 /로 replace
  - push의 경우 뒤로가기를 누르면 현재 페이지(`/search`)로 돌아오지만 replace의 경우 현재로 돌아오지 않고 네비게이션에 저장된 가장 최근 페이지로 돌아간다.(`\search`를 들어오기 전의 페이지)

```
const [_, query] = location.search.split("?term=");
if (!query) {
  return history.replace("/");
}
```

Lazy Query

- url의 query를 얻어오는 것이 늦어졌을 때 `useQuery()`는 쿼리를 바로 실행하지만, `useLazyQuery()`를 사용하면 query function을 반환받아 쿼리가 준비 된 이후에 실행할 수 있다.

```
const [queryReadyToStart, { loading, data, called }] = useLazyQuery<
  searchRestaurant,
  searchRestaurantVariables
>(SEARCH_RESTAURANT);
useEffect(() => {
  const [_, query] = location.search.split("?term=");
  if (!query) {
    return history.replace("/");
  }
  queryReadyToStart({
    variables: {
      input: {
        page: 1,
        query,
      },
    },
  });
}, [history, location]);
```

Apollo fragment

- gql 쿼리의 중복되는 부분은 fragment로 분리하여 사용하면 편리하다.
- src/fragments.ts

```
export const RESTAURANT_FRAGMENT = gql`
  fragment RestaurantParts on Restaurant {
    id
    name
    coverImg
    category {
      name
    }
    address
    isPromoted
  }
`;
```

- 사용(src/pages/client/restaurants.tsx)

```
const RESTAURANTS_QUERY = gql`
  query restaurantsPageQuery($input: RestaurantsInput!) {
    allCategories {
      ok
      error
      categories {
```

```
          ...CategoryParts
        }
      }
      restaurants(input: $input) {
        ok
        error
        totalPages
        totalResults
        results {
          ...RestaurantParts
        }
      }
    }
    ${RESTAURANT_FRAGMENT}
    ${CATEGORY_FRAGMENT}
  `;
```

useParam

- path parameter를 얻어오기 위해 사용.
- 라우터에 아래와 같이 추가(slug를 키로 parameter를 얻어올 수 있다.)

```
<Route key={5} path="/category/:slug" exact>
  <Category />
</Route>
```

- useLocation을 사용하여 search의 값으로 url의 path param을 읽어와도 되는데 useParam이 더 간편하여 사용하였다.

```
const params = useParams<ICategoryParams>();
const { data, loading } = useQuery<category, categoryVariables>(
    CATEGORY_QUERY,
    {
      variables: {
        input: {
          page: 1,
          slug: params.slug,
        },
      },
    }
  );
```

- `useLocation()` 사용시 location의 pathname을 활용하여야 한다.
  - useLocation()

```
{pathname: '/category/fooood', search: '', hash: '', state:
undefined, key: 'r0lr2u'}
```

- useParam()

```
{slug: 'fooood'}
```

20. Testing react components

Trouble shooting

LoggedInRouter테스트 실패

> 강의 촬영시와 버전이 많이 달라져서 생긴 에러로 추정

- `isLoggedInVar(true);`로 로그인여부가 업데이트 되기 전에 테스트가 수행되어 로그인된 결과가 아니라 로그아웃된 결과가 나타났다.
- 기존 코드

```
it("renders LoggedInRouter", async () => {
  render(<App />);
  await waitFor(() => {
    isLoggedInVar(true);
  });
  screen.getByText("logged in");
});
```

- `waitFor()` 안에서 테스트까지 수행하도록 변경.

> `waitFor()`은 state 변경을 기다리는 역할을 한다.

- 변경 코드

```
it("renders LoggedInRouter", async () => {
  render(<App />);
  await waitFor(() => {
    isLoggedInVar(true);
    expect(screen.getByText("logged in")).toBeInTheDocument();
  });
});
```

내용정리

React Testing Library

- Jest를 사용하여 React 컴포넌트 Mock

```
jest.mock("../../routers/logged-out-router", () => {
  return {
    LoggedOutRouter: () => <span>logged out</span>,
  };
});
```

- Test 실행
  - `render()`는 DOM 컴포넌트를 렌더링 해주는 역할

```
describe("<App />", () => {
  it("renders LoggedOutRouter", () => {
    render(<App />);
    screen.getByText("logged out");
  });
});
```

- rerender하여 다른 테스트 결과를 확인하는 것도 가능하다.

```
it("should render OK with props", () => {
  const { rerender } = render(
    <Button canClick={true} loading={false} actionText={"test"} />
  );
  screen.getByText("test");
  rerender(<Button canClick={true} loading={true} actionText={"test"}
/>);
  screen.getByText("Loading...");
});
```

- 컨테이너를 활용해 컴포넌트의 클래스를 테스트할 수 있다.

```
it("should display loading", () => {
  const {
    container: { firstChild },
  } = render(<Button canClick={false} loading={true} actionText=
{"test"} />);
  screen.getByText("Loading...");
  expect(firstChild).toHaveClass("pointer-events-none");
});
```

Apollo client를 사용하는 컴포넌트에 대한 테스트

- `MockedProvider`를 사용
  - 개별 쿼리에 대한 mock response 를 정의할 수 있어 GraphQL 서버와 통신할 필요가 사라져 외부 종속성이 제거된다.(안정적인 테스트 가능)

```
it("renders verify banner", async () => {
  render(
    <MockedProvider
      mocks={[
        {
          request: {
            query: ME_QUERY,
          },
          result: {
            data: {
              me: {
                id: 1,
                email: "",
                role: "",
                verified: false,
              },
            },
          },
        },
      ]}
    >
      <Router>
        <Header />
      </Router>
    </MockedProvider>
  );
  await waitFor(async () => {
    await new Promise((resolve) => setTimeout(resolve, 5));
    screen.getByText("Please verify your email.");
  });
});
```

- `getByText()` vs `queryByText()`
  - `getByText()`는 테스팅 시점에 해당 text가 렌더링되었는지 확인(해당 element가 사라졌는지는 확인 불가)

```
await waitFor(async () => {
  await new Promise((resolve) => setTimeout(resolve, 5));
  screen.getByText("Please verify your email.");
});
```

  - `queryByText()`는 해당 Element가 없으면 null을 반환해줘서 `.toBeNull()`을 활용하여 해당 요소가 사라졌는지 확인할 수 있다.

```
await waitFor(async () => {
  await new Promise((resolve) => setTimeout(resolve, 5));
  expect(screen.queryByText("Please verify your
```

```
  email.")).toBeNull();
  });
```

jest 참고

- jest.spyOn
  - 함수의 구현을 가짜로 대체하지 않고 함수의 호출 여부와 어떤 인자로 호출되었는지만 필요할 때 사용한다.

```
jest.spyOn(Storage.prototype, "setItem");
...
expect(localStorage.setItem).toHaveBeenCalledWith("nuber-token",
"XXX");
```

- jest.fn()
  - 가짜함수를 생성할 수 있다.
  - return 값이나 promise로 resolved 되는 값을 지정할 수 있다.

```
const mockedMutationResponse = jest.fn().mockResolvedValue({
  data: {
    login: { ok: true, token: "XXX", error: "mutation-error" },
  },
});
mockedClient.setRequestHandler(LOGIN_MUTATION,
mockedMutationResponse);
...
userEvent.type(email, formData.email);
userEvent.type(password, formData.password);
userEvent.click(submitBtn);

await waitFor(() => {
  expect(mockedMutationResponse).toHaveBeenCalledTimes(1);
});

expect(mockedMutationResponse).toHaveBeenCalledWith({
  loginInput: {
    email: formData.email,
    password: formData.password,
  },
});
```

- jest.requireActual()
  - 테스트를 위해 해당 모듈의 모든 함수를 mocking할 수 있다.
  - 해당 모듈에서 사용되는 함수를 일일이 mock하지 않아도 되서 편리하다.

```
const mockPush = jest.fn();
```

```
jest.mock("react-router-dom", () => {
  const realModule = jest.requireActual("react-router-dom");
  return {
    ...realModule,
    useHistory: () => {
      return {
        push: mockPush,
      };
    },
  };
});
```

react testing을 위한 custom render

- 테스트를 위한 컴포넌트를 mocking할 때 공통으로 자주 사용되는 패턴이 있다면 custom render를 만들어서 사용하면 편리하다.

```
import { render } from "@testing-library/react";
import React from "react";
import { HelmetProvider } from "react-helmet-async";
import { BrowserRouter as Router } from "react-router-dom";

const AllTheProviders: React.FC = ({ children }) => {
  return (
    <HelmetProvider>
      <Router>{children}</Router>
    </HelmetProvider>
  );
};

const customRender = (ui: React.ReactElement, options?: any) =>
  render(ui, { wrapper: AllTheProviders, ...options });

export * from "@testing-library/react";

export { customRender as render };
```

- 사용

```
import { render, waitFor } from "../../test-utils";
...
    render(
      <ApolloProvider client={mockedClient}>
        <CreateAccount />
      </ApolloProvider>
    );
```

21. E2E React testing

Trouble shooting

로그인시 Loading... 화면에서 다음 화면으로 넘어가지 않는 현상

> 새로고침해야 home으로 연결되고 최초 로그인시에는 Loading... 화면에서 멈춰있고, apollo에서 useMe query에 대해 Forbidden resource 응답.

- apollo 요청시 x-jwt 헤더에 전역으로 관리하는 token 변수인 authToken을 사용해야 하는데 로컬스토리지의 토큰값을 사용하여 빈 토큰값 전달.
- `makeVar()`로 가져온 전역변수토큰값 이용하도록 헤더 수정

> apollo.ts의 setContext()에서 `console.log("token: " + token, "authToken: " + authToken());` 찍었을 때 token만 null인 이유는....?

내용 정리

Cypress

> E2E test를 위한 툴

- 설치

```
npm install cypress --save-dev
```

- 실행

```
npx cypress open
```

- 설정
  - cypress/tsconfig.json

```
{
  "compilerOptions": {
    "allowJs": true,
    "baseUrl": "../node_modules",
    "types": ["cypress"],
    "outDir": "#"
  },

  "include": ["./**/*.*"]
}
```

- cypress/e2e/[testfile].cy.ts 파일 안에 테스트 내용 작성(jest와 비슷한 형태)

- intercept
  - 서버로 보내는 요청을 중간에 가로챌 수 있다. 여기선 계정 생성이 실제로 되지 않으면서 응답을 받을 수 있다.

```
user.intercept("http://localhost:3030/graphql", (req) => {
  const { operationName } = req.body;
  if (operationName && operationName === "createAccountMutation") {
    req.reply((res) => {
      res.send({
        data: {
          createAccount: {
            ok: true,
            error: null,
            __typename: "CreateAccountOutput",
          },
        },
      });
    });
  }
});
```

Cypress Testing Library

> cypress 테스트를 더 편리하게 해주는 라이브러리

- 사용 전

```
describe("First Test", () => {
  it("can fill out the form", () => {
    cy.visit("/")
      .get('[name="email"]')
      .type("1234512345")
      .get('[name="password"]')
      .get(".text-lg")
      .should("not.have.class", "pointer-events-none");
  });
});
```

- 사용 후
  - find~함수 자동완성 가능

```
describe("First Test", () => {
  it("can fill out the form", () => {
    cy.visit("/");
    cy.findByPlaceholderText(/Email/i).type("asdf@adfda.co");
    cy.findByPlaceholderText(/password/i).type("1234512345");
    cy.findByRole("button").should("not.have.class", "pointer-events-
none");
```

```
  });
});
```

Cypress custom commands

- 자주 사용되는 공통적인 테스트 부분을 명령어로 만들 수 있다.
- cypress/support/commands.ts 파일 내부에 명령어를 정의하면 된다.

```
Cypress.Commands.add("assertLoggedIn", () => {
  cy.window().its("localStorage.nuber-token").should("be.a",
"string");
});

Cypress.Commands.add("assertLoggedOut", () => {
  cy.window().its("localStorage.nuber-token").should("be.undefined");
});

Cypress.Commands.add("login", (email, password) => {
  // @ts-ignore
  cy.assertLoggedOut();
  cy.visit("/");
  cy.title().should("eq", "Login | Nuber eats");
  cy.findByPlaceholderText(/Email/i).type(email);
  cy.findByPlaceholderText(/password/i).type(password);
  cy.findByRole("button")
    .should("not.have.class", "pointer-events-none")
    .click();
  // @ts-ignore
  cy.assertLoggedIn();
});
```

- 사용

```
it("can fill out the form", () => {
  // @ts-ignore
  user.login("yellow2041@naver.com", "1212121212");
  user.window().its("localStorage.nuber-token").should("be.a",
"string");
});
```

22. OWNER DASHBOARD

Trouble shooting

백엔드 typeORM 잘못 작성한 후기,,,

- 강의때와 typeorm의 버전이 달라서 쿼리 작성하는 방식이 조금 달라졌다.

  > 오랜만에 백엔드를 수정하다보니 강의 그대로 적고 왜 안되는지 30분 넘게 헤맨듯ㅠ

- 잘못된 코드

```
const restaurants = await this.restaurants.find({
  where: { ownerId: owner.id },
});
```

  - ownerId는 테이블 간의 관계를 정의하기 위해 사용하는 것인데, 이를 기준으로 사용하여 쿼리 실행시 백엔드에서 ownerId가 없다는 에러가 발생하고 있었다. 아래와 같이 owner object의 id를 기준으로 레스토랑을 찾아오도록 수정하였다.

- 수정 코드

```
const restaurants = await this.restaurants.find({
  where: { owner: { id: owner.id } },
});
```

createRestaurant mutation 실행 시 name 글자 수 제한

> 백엔드의 Restaurant의 name에 `@Length(5)`을 적용해두고 자꾸 5글자 미만으로 요청을 보내고 있었다...🤦‍♀️

- 프론트에서 createRestaurant input의 name을 3글자로 채워서 보낼 시

```
index.ts:59 Uncaught (in promise) ApolloError: Bad Request Exception
  at new ApolloError (index.ts:59:1)
  at QueryManager.ts:256:1
  at both (asyncMap.ts:30:1)
  at asyncMap.ts:19:1
  at new Promise (<anonymous>)
  at Object.then (asyncMap.ts:19:1)
  at Object.next (asyncMap.ts:31:1)
  at notifySubscription (module.js:132:1)
  at onNotify (module.js:176:1)
  at SubscriptionObserver.next (module.js:225:1)
```

  에러 발생...
  - apollo 요청시 뮤테이션이 잘못되었을 때 400을 던진다고 한다.
- 백엔드에 최소글자 설정을 해둬서 발생한 에러...

  > 에러 내용이 친절하지 않네요... mutation 오타난줄알고 열심히 찾았는데...ㅠ.ㅠ

학습 내용

AWS s3를 이용한 파일업로드 구현(BE)

- aws에 AmazonS3FullAccess정책을 가진 사용자 생성
- `aws-sdk` 설치
- `nest g mo uploads`으로 모듈 생성 후 `controllers: [UploadsController]` 추가
- controller 생성

```
import {
Controller,
Post,
UploadedFile,
UseInterceptors,
} from '@nestjs/common';
import { FileInterceptor } from '@nestjs/platform-express';
import * as AWS from 'aws-sdk';

const BUCKET_NAME = 'janabinubereats';

@Controller('uploads')
export class UploadsController {
  @Post('')
  @UseInterceptors(FileInterceptor('file'))
  async uploadFile(@UploadedFile() file) {
    AWS.config.update({
      credentials: {
        accessKeyId: process.env.S3_ACCESS_KEY_ID,
        secretAccessKey: process.env.S3_SECRET_ACCESS_KEY,
      },
    });
    try {
      // AWS.config.update({ region: 'ap-northeast-2' });
      const objectName = Date.now() + file.originalname;
      await new AWS.S3()
        .putObject({  // 초기에 createBucket 실행 필요
          Body: file.buffer,
          Bucket: BUCKET_NAME,
          Key: objectName,
          ACL: 'public-read',
        })
        .promise();
      const url =
`https://${BUCKET_NAME}.s3.amazonaws.com/${objectName}`;
      return { url };
    } catch (e) {
      return null;
    }
  }
}
```

- 포스트맨에서 Header의 Content-Type을 multipart/form-data로 설정 후 Body의 form-data로 key를 file로하여 사진 업로드 테스트 가능

레스토랑 추가 후 refetch를 이용한 refresh 방법

- apollo는 레스토랑 추가 후 원래 페이지로 돌아가면 기존 캐시에 있던 정보들을 그대로 보여주기 때문에 실제로 서버엔 레스토랑이 추가되었지만 화면엔 추가되기 전의 상태를 보여준다.
- `refetchQueries`를 이용하여 특정 뮤테이션이 발생하였을때 특정 쿼리를 refetch해오도록 만들 수 있다.
    - 하지만 레스토랑의 갯수가 많아지는 경우 refetch하여 가져와야 할 데이터가 많아지기때문에 효율적이지 않다.
        - 효율적으로 바꾸는것은 뒷부분에 진행 예정

```
const [createRestaurantMutation, { data }] = useMutation<
  createRestaurant,
  createRestaurantVariables
>(CREATE_RESTAURANT_MUTATION, {
  onCompleted,
  refetchQueries: [{ query: MY_RESTAURANTS_QUERY }],
});
const { register, getValues, formState, handleSubmit } =
useForm<IFormProps>({
  mode: "onChange",
});
```

readQuery와 writeQuery를 이용한 fake refresh 방법

- readQuery로 아폴로 캐시에 저장되어있는 기존 쿼리 결과를 읽어오고, writeQuery로 추가된 정보를 캐시에 기록하여 api호출 없이 새 데이터를 추가할 수 있다.
- fake로 writeQuery에 add된 레스토랑 결과를 추가할 때는 아폴로 캐시에 저장된 형태와 동일한 형태로 저장하면 된다.

```
const client = useApolloClient();
const history = useHistory();

const onCompleted = (data: createRestaurant) => {
  const {
    createRestaurant: { ok, restaurantId },
  } = data;
  if (ok) {
    const { file, name, categoryName, address } = getValues();
    setUploading(false);
    const queryResult = client.readQuery({
      query: MY_RESTAURANTS_QUERY,
    });
    client.writeQuery({
      query: MY_RESTAURANTS_QUERY,
      data: {
        ...queryResult.myRestaurants,
        myRestaurants: [
          {
            address,
            category: {
              name: categoryName,
              __typename: "Category",
              __proto__: Object,
```

```
            },
            coverImg: imageUrl,
            id: restaurantId,
            isPromoted: false,
            name,
            __typename: "Restaurant",
          },
          ...queryResult.myRestaurants.restaurants,
        ],
      },
    });
    history.push("/");
  }
};
```

메뉴별 세부 옵션 추가

- `Date.now()`를 key 값으로 활용하여 옵션 추가 및 삭제

```
const onSubmit = () => {
  const { name, price, description, ...rest } = getValues();
  const optionObjects = optionsNumber.map((theId) => ({
    name: rest[`${theId}-optionName`],
    extra: +rest[`${theId}-optionExtra`],
  }));
  createDishMutation({
    variables: {
      input: {
        name,
        price: +price,
        description,
        restaurantId: +restaurantId,
        options: optionObjects,
      },
    },
  });
  history.goBack();
};
const onAddOptionClick = () => {
  setOptionsNumber((current) => [Date.now(), ...current]);
};
const onDeleteClick = (idToDelete: Number) => {
  setOptionsNumber((current) => current.filter((id) => id !==
idToDelete));
  unregister(`${idToDelete}-optionName`);
  unregister(`${idToDelete}-optionExtra`);
};
```

- 추가 및 삭제하는 부분

```
{optionsNumber.length !== 0 &&
  optionsNumber.map((id) => (
    <div key={id} className="mt-5">
      <input
        {...register(`${id}-optionName`)}
        className="py-2 px-4 focus:outline-none mr-3 focus:border-
gray-600 border-2"
        type="text"
        placeholder="Option Name"
      />
      <input
        {...register(`${id}-optionExtra`)}
        className="py-2 px-4 focus:outline-none mr-3 focus:border-
gray-600 border-2"
        type="number"
        min={0}
        placeholder="Option Extra"
      />
      <span
        className="cursor-pointer text-white bg-red-500 ml-3 py-3 px-4
mt-5 bg-"
        onClick={() => onDeleteClick(id)}
      >
        Delete Option
      </span>
    </div>
  ))}
```

Victory를 활용한 차트 만들기

- Victory를 설치하여 차트를 만들 수 있다.
- 그래프 종류도 다양하고 속성도 많아서 공식문서 참고하면 좋을듯.
- 예시
  - 막대그래프

```
<VictoryChart domainPadding={20}>
  <VictoryAxis
    tickFormat={(step) => `${step / 1000}K`}
    dependentAxis
  />
  <VictoryAxis label="Days" tickFormat={(step) => `Day ${step}`}
/>
  <VictoryBar data={chartData} />
</VictoryChart>
```

  - 원형그래프

```
<VictoryPie data={chartData} />
```

24. Making an order

Trouble shooting

학습 내용