Consume APIs de GraphQL con Apollo Client en Reactjs

juandc 37531

Apollo Client en una herramienta que nos ayuda a hacer consultas a APIs de GraphQL; y lo usaremos para construir una aplicación que buscara **GIFs** gracias a la *API GraphQL* de Giphy.

Instalación

Lo primero que debemos hacer es, instalar algunos paquetes de *Apollo* y *GraphQL* que nos ayuden a para construir nuestra aplicación:

```
yarn add apollo-boost react-apollo graphql-tag graphql
```

El paquete `apollo-boost` es una combinación de algunos paquetes paquetes que tendríamos que instalar separadamente en caso de proferirlo. Al usarlo estamos *limitando* un poco nuestra *configuración*; pero para este tutorial la configuración que se nos permite es más que suficiente. El resto de paquetes simplemente los necesitamos.

Configuración de *Apollo Client*

Configuración de *Apollo Client*

Para empezar a construir nuestra aplicación, necesitamos decirle a **Apollo** qué *API* usaremos. Para esto usaremos `apollo-boost`, que nos ayudara a crear una configuración de forma muy sencilla, donde le diremos el **Endpoint** de nuestra *API*:

```
// config/apollo.js

import ApolloClient from "apollo-boost";

export const client = new ApolloClient({
  uri: 'https://www.graphqlhub.com/graphql' // por defecto la URL seria
'/graphql'
});
```

Si quisiéramos una configuración mucho mas avanzada necesitaríamos instalar por separado los paquetes necesarios de *Apollo*. Para aprender a hacerlo puedes seguir el siguiente tutorial: Apollo Boost Migration.

Configuración de Reactjs para usar *Apollo Client*

Ahora debemos decirle a nuestra aplicación que usaremos *Apollo Client*.
Para esto, usaremos el paquete `react-apollo`. Este paquete nos dará un componente llamado **Apollo Provider**, que usaremos para avisarle a nuestra aplicación que estamos usando *Apollo Client*:

```
// app/index.js

import { ApolloProvider } from 'react-apollo';
import { client } from '../config/apollo'; // la configuración de apollo
que acabamos de crear
import App from './application'; // el archivo donde crearemos nuestra
aplicación

export default () => (
  <ApolloProvider client={client}> {/* en "client" debe ir la
configuración de apollo */}
    <App />
  </ApolloProvider>
);
```

Ahora que tenemos nuestra configuración lista, podemos construir nuestra aplicación.

Creación de los componentes

Creación de los componentes

El componente Contenedor

Este componente **guardara los estados** de nuestra aplicación y nos avisara cuándo podemos mostrar o no los resultados:

```
// app/application.js

import SearchBar from './SearchBar'; // ya lo crearemos :D
import GIFs from './GIFs'; // tambien ya lo crearemos :P

class App extends React.Component {
  state = {
    searchText: '',
    showResults: false, // solo habrá resultados cuando el usuario pulse
el botón
  };

  handleInputChange = event => {
    /*
     * Para saber lo que el usuario quiere buscar, necesitamos
     * escuchar los cambios del input. Así cuando oprima el botón
     * de búsqueda, sabremos cuales GIFs espera encontrar.
     * "event.target.value" es el valor del input
    */
    this.setState({ searchText: event.target.value });
  };

  handleButtonClick = () => {
    // El componente que busca y muestra los GIFs solo se mostrara
    // cuando showResults sea true.
    this.setState({ showResults: true });
  }
```

```
  render() {
    const { searchText, showResults } = this.state;

    return (
      <React.Fragment>
        <SearchBar
          searchText={searchText}
          handleChange={this.handleChange}
          handleClick={this.handleClick}
        />
        {!!showResults && <GIFs search={searchText} />}
      </React.Fragment>
    );
  }
}
```

La Barra de Búsqueda

La *Barra de búsqueda* le servirá a nuestros usuarios para realizar la búsqueda que ellos quieran. Nuestro componente *Container* tiene toda la *lógica* nuestra barra de búsqueda. Ahora solo necesitamos usar los métodos que ya hemos a nuestro componente para cambiar el estado nuestra aplicación dependiendo de las acciones del usuario: **escribir**, o **buscar**.

```
// app/SearchBar.js

// En las props tenemos los métodos necesarios para interactuar con
nuestros usuarios:
function SearchBar(props) {
  return (
      <div className="SearchBar__container">
        <input
          type="text"
          className="SearchBar__input"
          value={props.searchText}
          onChange={e => props.handleChange(e)}
        />
        <button
          className="SearchBar__button"
          onClick={e => props.handleClick(e)}
        >
          Buscar
        </button>
      </div>
    );
}
```

Ahora solo necesitamos un componente, con el que gracias a *Apollo Client* podamos realizar la búsqueda de

GIFs.

El componente de GIFs

Ahora que la lógica de nuestra aplicación esta lista, solo nos hace falta lo mas importante: **Traer las Imagenes**.

Para esto haremos uso de un componente de `react-apollo` llamado **Query**.

Este componente **Query** nos ayudara a realizar nuestras peticiones, escribiendo las *queries* necesarias gracias al paquete `gql` que instalamos al inicio. Por ultimo, sabremos por medio de **Render Props** si nuestra petición ha sido *exitosa*, esta *cargando* o si hubo un *error*.

```
// app/GIFs.js

import { Query } from 'react-apollo';

const GET_GIPHY_IMAGES = gql`
  # Ya que no sabemos qué buscaran nuestros usuarios, tenemos que hacer
  # una petición con variables; para cada vez que hacemos una búsqueda
  # cambiar la petición y obtener el resultado correcto.

  query dog($search: String!) {
    giphy {
      search(query: $search, limit: 10, offset: 0, rating: pg) {
        id
        images {
          original {
            url
          }
        }
      }
    }
  }
`;

function GIFs({ search }) {
  return (
    <Query query={GET_GIPHY_IMAGES} variables={{ search }}>
      {({  loading, error, data }) => {
        if (loading) return <p>loading...</p>;
        if (error) return <p>Sorry, we have an error...</p>;

        // si no hay errores, ni estamos cargando, tenemos nuestras
imagenes!
        return (
          <div className="GIFs__container">
            {data.giphy.search.map(
              ({ id, images }) => (
                <img key={id} alt={id} src={images.url.original} /> {/*
Tenemos los GIFs! */}
              )
```

```
            )}
          </div>
        );
      }}
    </Query>
  );
}
```

Y listo! Eso es todo lo que debíamos hacer para construir nuestra aplicación!

Conclusion

Usar *Apollo Client* para consumir *APIs* de *GraphQL* es muy sencillo. Ahora te animo a seguir aprendiendo y compartir con nosotros qué tal te va usando Apollo con *Reactjs* o cualquier otro framework JavaScript. Hasta la próxima!

0 hace 10 meses Python

Escribe tu comentario +2

Entradas relacionadas

2

Como usar un API Rest con GraphQL

Hola chicos! El día de hoy les comparto un post que me encontré por Medium y me pareció muy interesante en el cual podemos aprender como pas

 .

2

Apollo Client GraphQL: Actualizar Apollo Client a la version 2.x.x

En el curso se utiliza la version 1.x.x de Apollo Client, especificamente la 1.4.2, la cual ha sufrido cambios en su API, estando actualment

 **ivanguerra10**

7

BD: Conectar GraphQL con PostgreSQL a través de knex

El proyecto del curso utiliza sqLite, lo cual me parece muy conveniente debido a la facilidad de implementarlo sin mayor complicación, sin e

 **jjyepez**