PROYECTO REACT-GRAPHQL

Aplicación de React para consumir APIs mediante el servicio Graphql

Revisar el documento del proyecto graphql-server: graphql-server.doc

Crear proyecto React

- *Crear proyecto con vitejs y con plantilla de react*

  _> npm init vite@latest react-graphql -- --template react

```
loper@wlopera MINGW64 /c/A_CURSOS/2024/GraphQL
$ npm init vite@latest react-graphql -- --template react
npm WARN exec The following package was not found and will be installed: create-vite@5.3.0

Scaffolding project in C:\A_CURSOS\2024\GraphQL\react-graphql...

Done. Now run:

  cd react-graphql
  npm install
  npm run dev
```

- *Instalar librerías (cd rteact-graphql)*

  _> npm install

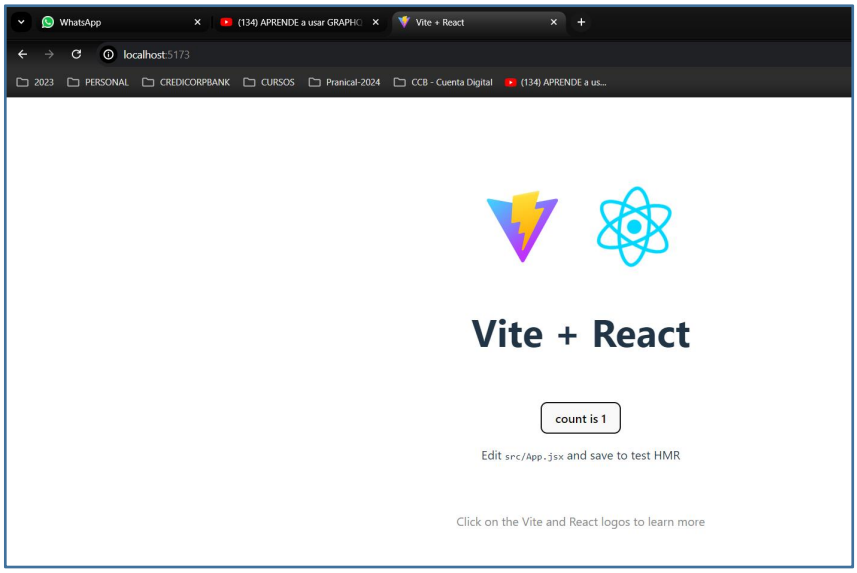
```
loper@wlopera MINGW64 /c/A_CURSOS/2024/GraphQL/react-graphql
$ npm install
npm WARN deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it
more comprehensive and powerful.
npm WARN deprecated @humanwhocodes/config-array@0.11.14: Use @eslint/config-array instead
npm WARN deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm WARN deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm WARN deprecated @humanwhocodes/object-schema@2.0.3: Use @eslint/object-schema instead

added 278 packages, and audited 279 packages in 2m

103 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- *Levantar y probar APP de react*

  _> npm run dev

```
loper@wlopera MINGW64 /c/A_CURSOS/2024/GraphQL/react-graphql
$ npm run
Scripts available in react-graphql@0.0.0 via `npm run-script`:
  dev
    vite
  build
    vite build
  lint
    eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0
  preview
    vite preview


loper@wlopera MINGW64 /c/A_CURSOS/2024/GraphQL/react-graphql
$ npm run dev

> react-graphql@0.0.0 dev
> vite


  VITE v5.3.3  ready in 142 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
```

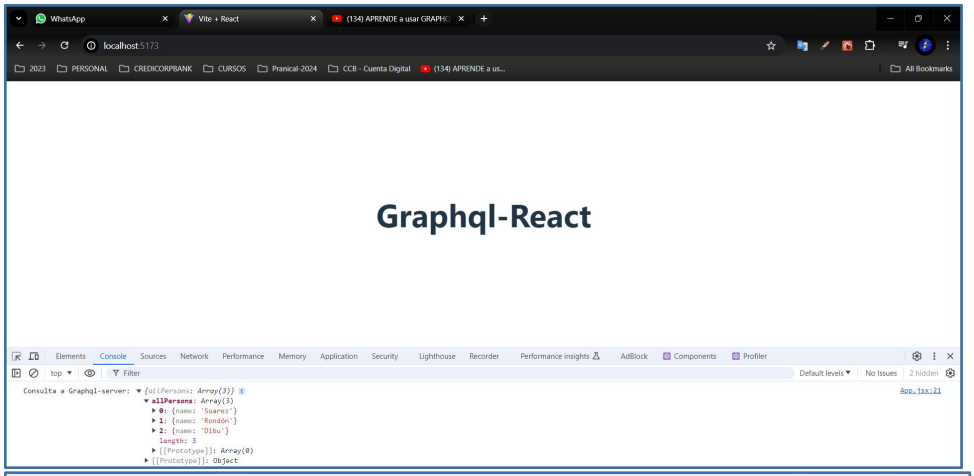
- *Mostrar APP*

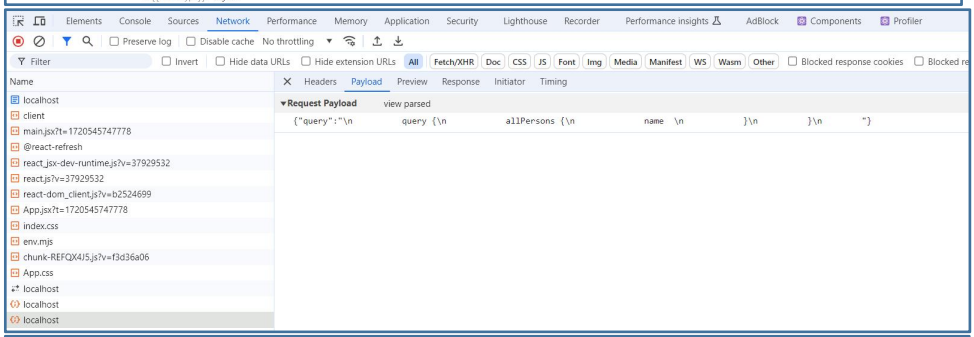


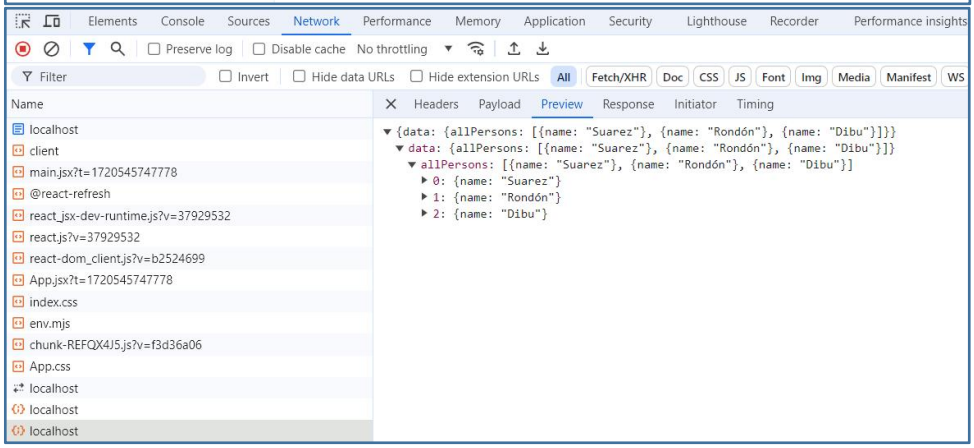
- *Limpiar el código base de react y realizar llamado a servicio de Graphql-server*

```
import { useEffect } from 'react'
import './App.css'

function App() {

  useEffect(() => {
    const queryGraphqlData = async()=>{
      const response = await fetch('http://localhost:4000', {
        method: 'POST',
        headers: {'Content-Type':'application/json'},
        body: JSON.stringify({query: `
          query {
            allPersons {
              name
            }
          }
        `})
      })
      const jsonData = await response.json()
      console.log("Consulta a Graphql-server:", jsonData.data)
    }
    queryGraphqlData()
  }, [])

  return (
    <>
      <h1>Graphql-React</h1>
    </>
  )
}

export default App
```

El servicio graphql-server se puede consultar desde cualquier cliente API Rest


WhatsApp
Vite + React
(134) APRENDE a usar GRAPHQ
localhost:5173
Graphql-React
Consulta a Graphql-server:
App.jsx:21


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
AdBlock
Components
Profiler
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
Hide extension URLs
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
WS
Wasm
Other
Name
Headers
Payload
Preview
Response
Initiator
Timing
Request Payload
view parsed
localhost
client
main.jsx?t=1720545747778
@react-refresh
react_jsx-dev-runtime.js?v=37929532
react.js?v=37929532
react-dom_client.js?v=b2524699
App.jsx?t=1720545747778
index.css
env.mjs
chunk-REFQX4J5.js?v=f3d36a06
App.css


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Performance insights
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
Hide extension URLs
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
WS
Name
Headers
Payload
Preview
Response
Initiator
Timing
localhost
client
main.jsx?t=1720545747778
@react-refresh
react_jsx-dev-runtime.js?v=37929532
react.js?v=37929532
react-dom_client.js?v=b2524699
App.jsx?t=1720545747778
index.css
env.mjs
chunk-REFQX4J5.js?v=f3d36a06
App.css
{data: {allPersons: [{name: "Suarez"}, {name: "Rondón"}, {name: "Dibu"}]}}
data: {allPersons: [{name: "Suarez"}, {name: "Rondón"}, {name: "Dibu"}]}
allPersons: [{name: "Suarez"}, {name: "Rondón"}, {name: "Dibu"}]
0: {name: "Suarez"}
1: {name: "Rondón"}
2: {name: "Dibu"}

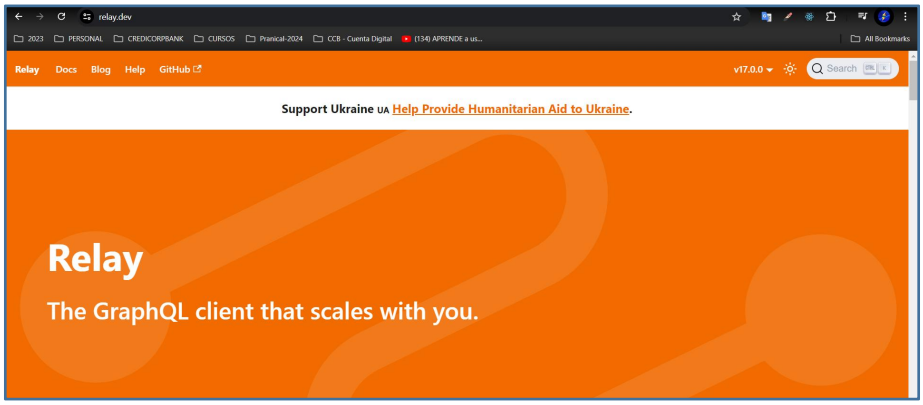
- Pero para existen clientes de graphql que permite un mejor control y desarrollo de las Aplicaciones del ado del cliente.

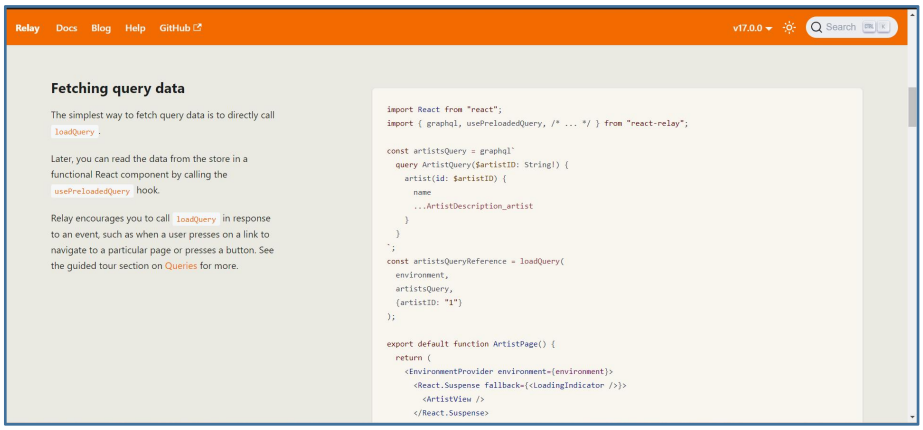
    - Relay_dev: Utilizado por Facebook para proyectos muy grandes

Diseñado para escalar

Relay está diseñado para un alto rendimiento a cualquier escala. Relay facilita la gestión de la obtención de datos, ya sea que su aplicación tenga decenas, cientos o miles de componentes. Y gracias al compilador incremental de Relay, mantiene la velocidad de iteración rápida incluso a medida que su aplicación crece.

https://relay.dev/

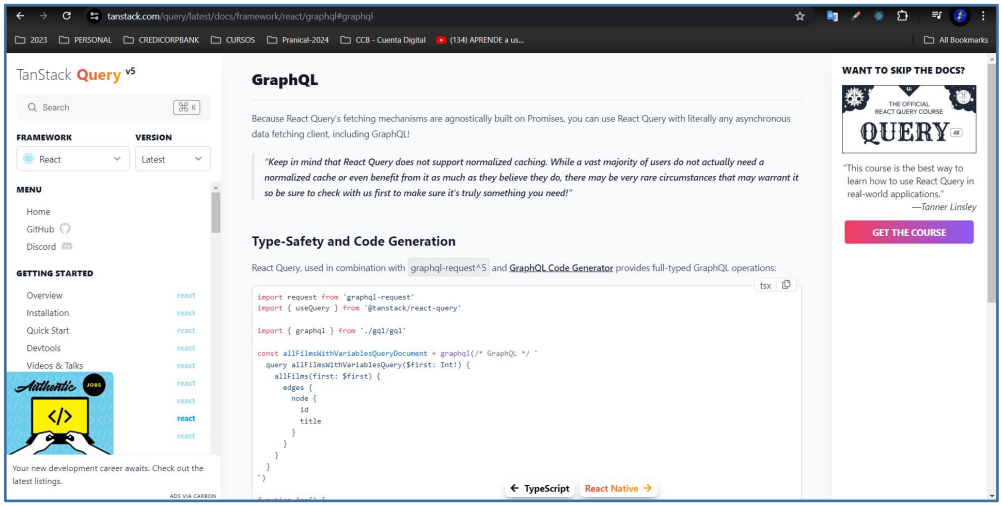




https://developers.facebook.com/videos/2019/building-the-new-facebookcom-with-react-graphql-and-relay/

- *React-query*

TanStack Query te ofrece consultas y mutaciones declarativas, siempre actualizadas y gestionadas automáticamente que mejoran directamente tanto tu experiencia de desarrollador como la de usuario .



https://tanstack.com/query/v3

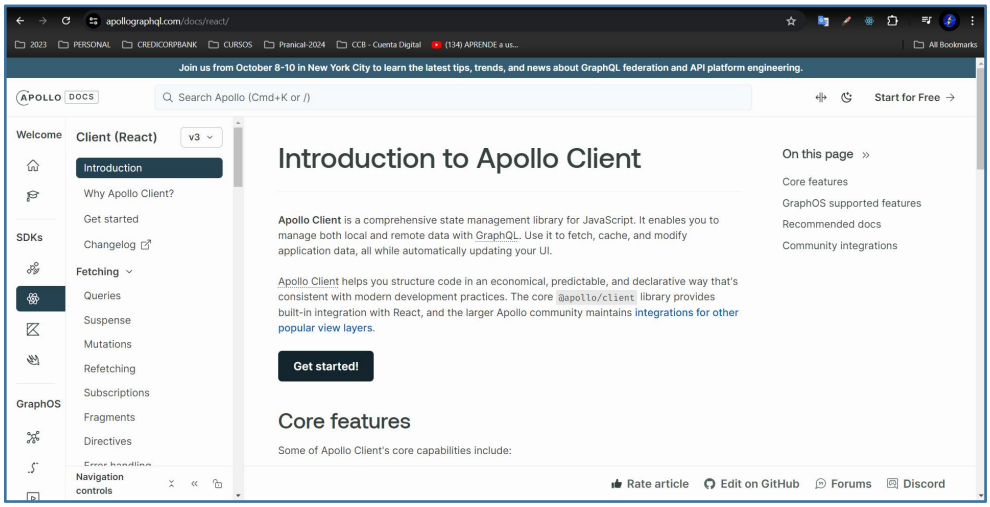
https://tanstack.com/query/latest/docs/framework/react/graphql#graphql

Apollo Client Graphql

Apollo Client es una biblioteca de administración de estado integral para JavaScript. Le permite administrar datos locales y remotos con GraphQL . Úselo para obtener, almacenar en caché y modificar datos de la aplicación, todo mientras actualiza automáticamente su interfaz de usuario.

Apollo Client te ayuda a estructurar el código de una manera económica, predecible y declarativa que es coherente con las prácticas de desarrollo modernas. La @apollo/clientbiblioteca principal proporciona integración incorporada con React, y la comunidad más grande de Apollo mantiene integraciones para otras capas de vista populares .

https://www.apollographql.com/docs/react/

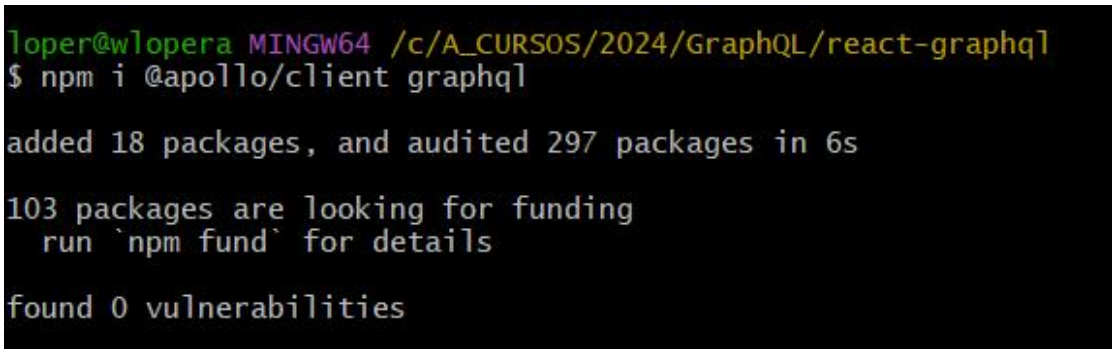


Vamos a utilizar el cliente de Apollo

- Instalar librerías

  _> npm install @apollo/client

  _> npmk install graphql



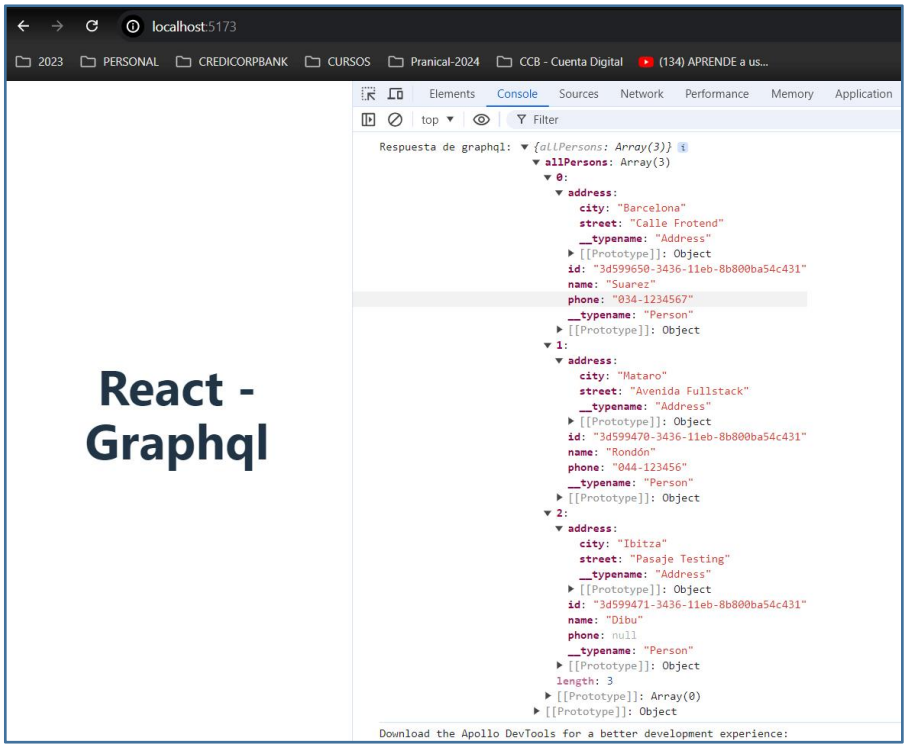
En main.js

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import {ApolloClient, HttpLink, InMemoryCache, gql} from '@apollo/client'

import './index.css'

// Debo pasar esa uri a variables de ambiente
const client = new ApolloClient({
  cache: new InMemoryCache(),
  link: new HttpLink({
    uri: 'http://localhost:4000'
  })
})

const query = gql`
  query {
    allPersons {
      id
      name
      phone
      address {
        street
        city
      }
    }
  }
`

client.query({query})
  .then(response => console.log("Respuesta de graphql:", response.data))

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Apollo client maneja cache así cuando realizo peticiones tengo a la mano la data sin necesidad de volver a consultar cuando la data requerida no cambia.

Otra forma de es:

Salida:


localhost:5173
2023
PERSONAL
CREDICORPBANK
CURSOS
Pranical-2024
CCB - Cuenta Digital
(134) APRENDE a us...
Elements
Console
Sources
Network
Performance
Memory
Application
top
Filter
React - Graphql
Respuesta de graphql: {allPersons: Array(3)}
allPersons: Array(3)
0:
address:
city: "Barcelona"
street: "Calle Frotend"
__typename: "Address"
[[Prototype]]: Object
id: "3d599650-3436-11eb-8b800ba54c431"
name: "Suarez"
phone: "034-1234567"
__typename: "Person"
[[Prototype]]: Object
1:
address:
city: "Mataro"
street: "Avenida Fullstack"
__typename: "Address"
[[Prototype]]: Object
id: "3d599470-3436-11eb-8b800ba54c431"
name: "Rondón"
phone: "044-123456"
__typename: "Person"
[[Prototype]]: Object
2:
address:
city: "Ibitza"
street: "Pasaje Testing"
__typename: "Address"
[[Prototype]]: Object
id: "3d599471-3436-11eb-8b800ba54c431"
name: "Dibu"
phone: null
__typename: "Person"
[[Prototype]]: Object
length: 3
[[Prototype]]: Array(0)
[[Prototype]]: Object
Download the Apollo DevTools for a better development experience:

Configurar variables de ambiente para manejo de variables y constantes

Agregar accesoa variable en mi main.js

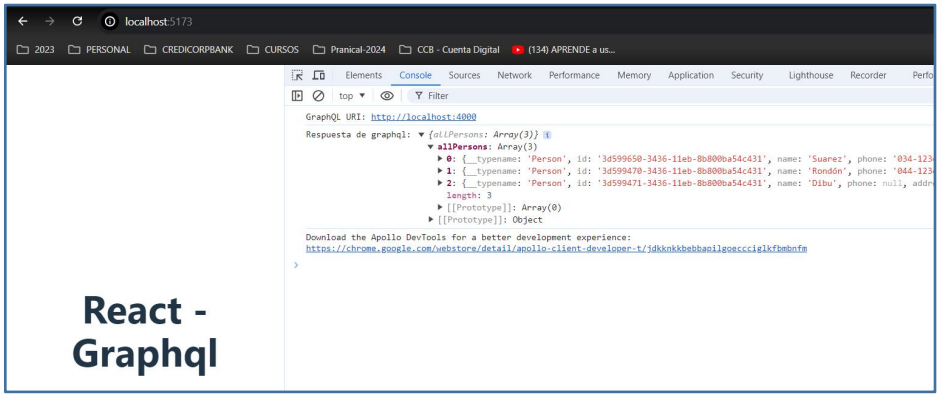
```
...
const GRAPHQL_URI = import.meta.env.VITE_GRAPHQL_URI;
console.log('GraphQL URI:', GRAPHQL_URI);
...
```

- *Crear ambiente desarrollo*

.env:

```
VITE_GRAPHQL_URI=http://localhost:4000
```

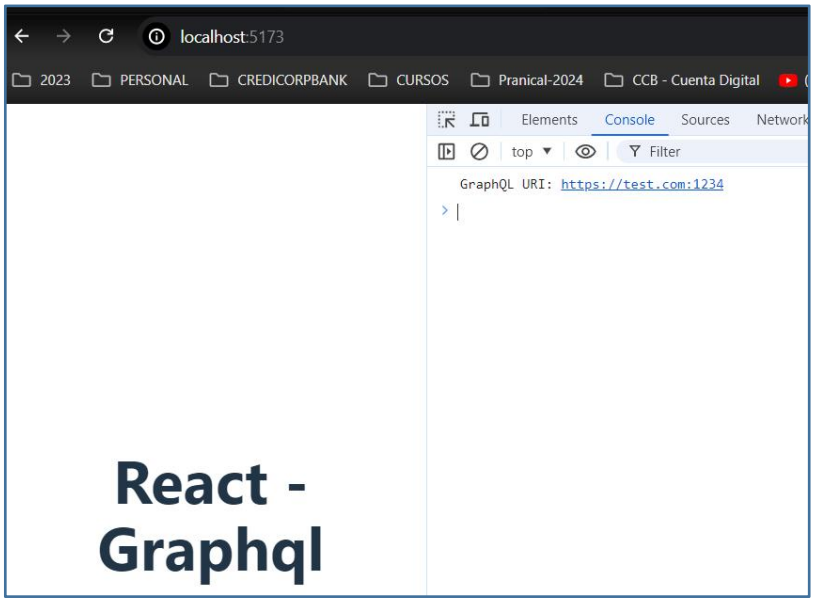
  - *Levantar react como desarrollo (npm run dev)*



- Crear ambiente producción (npm run dev -- --mode production)

.env.production

```
VITE_GRAPHQL_URI=https://test.com:1234
```



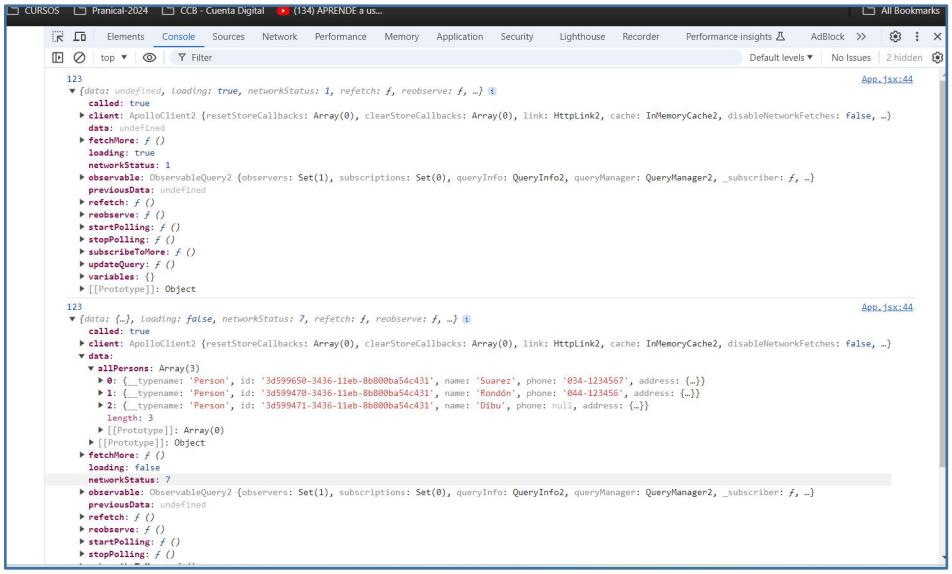
- *Agregamos un provider en main.js:*

```
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import {ApolloClient, ApolloProvider, HttpLink, InMemoryCache} from '@apollo/client'

import './index.css'

const GRAPHQL_URI = import.meta.env.VITE_GRAPHQL_URI;

// console.log('GraphQL URI:', GRAPHQL_URI);
// Debo pasar esa uri a variables de ambiente

const client = new ApolloClient({
  cache: new InMemoryCache(),
  link: new HttpLink({
    uri: GRAPHQL_URI
  })
})

// const request = gql`
//   query {
//     allPersons {
//       id
//       name
//       phone
//       address {
//         street
//         city
//       }
//     }
//   }
// `

// client.query({query: request})
//   .then(response => console.log("Respuesta de graphql:", response.data))
//   .catch((error) => console.error("Error en la consulta GraphQL:", error));

ReactDOM.createRoot(document.getElementById('root')).render(
  <ApolloProvider client={client}>
    <App />
  </ApolloProvider>,
)
```

- *En App.js se realiza la consulta*

```
// import { useEffect } from 'react'
import { gql, useQuery } from '@apollo/client'
import './App.css'

const ALL_PERSONS = gql`
  query {
    allPersons {
      id
      name
      phone
      address {
        street
        city
      }
    }
  }
`

function App() {

  // useEffect(() => {
  //   const queryGraphqlData = async()=>{
  //     const response = await fetch('http://localhost:4000', {
  //       method: 'POST',
  //       headers: {'Content-Type':'application/json'},
  //       body: JSON.stringify({query: `
  //         query {
  //           allPersons {
  //             name
  //           }
  //         }
  //       `})
  //     })

  //     const jsonData = await response.json()
  //     console.log("Consulta a Graphql-server:", jsonData.data)
  //   }

  //   queryGraphqlData()
  // }, [])

  const result = useQuery(ALL_PERSONS)
  console.log(123, result)

  return (
    <>
      <h1>React - Graphql</h1>
    </>
  )
}

export default App
```

- Se genera dos resultados de la consulta



En uno loading true y data undefined y la segunda loading false y data con información solicitada. Se envían diferentes estados en diferentes momentos para control de la data. (También trae un objeto error, entre otros)

Creamos un componente Person.jsx al cual le pasamos la logica de la consulta para mostrarla en la pantalla.
Agregamos un click que permite consultar otro graphql para buscar una persona dado su nombre y mostrar otro salida que despliega los datos consultados (uso de query con parámetros)
Uso de useLazyQuery que permite llamar un query graphql pasandole parámetros

```
/* eslint-disable react/prop-types */

import { gql, useLazyQuery } from "@apollo/client";
import { useEffect, useState } from "react";

const FIND_PERSON = gql`
  query findQueryByName($nameToSearch: String!) {
    findPerson(name: $nameToSearch) {
      id
      name
      phone
      address {
        street
        city
      }
    }
  }
`;

export const Persons = ({ persons }) => {
  const [person, setPerson] = useState(null);
  const [getPerson, result] = useLazyQuery(FIND_PERSON);
```

```
  useEffect(() => {
    if (result.data) {
      setPerson(result.data.findPerson);
    }
  }, [result]);
```

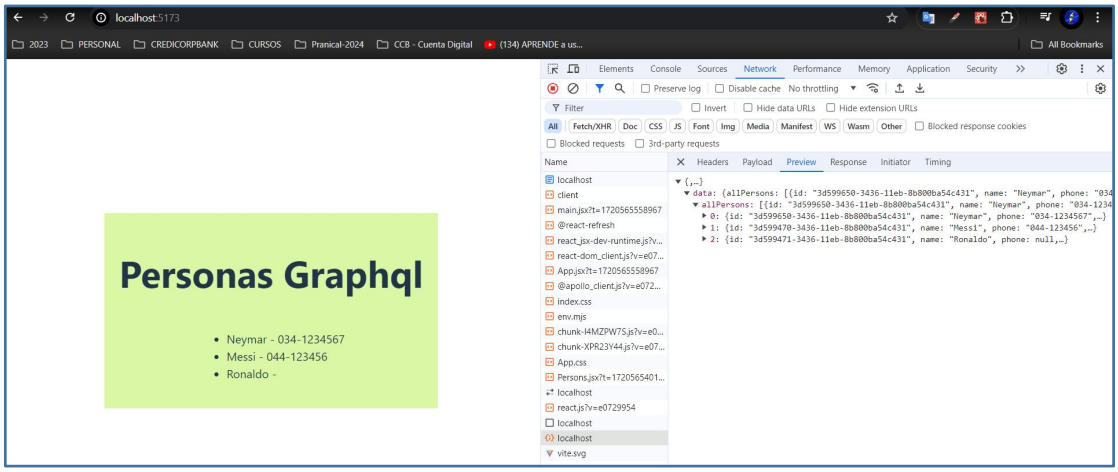
```
  const handlePerson = (name) => {
    getPerson({ variables: { nameToSearch: name } });
  };

  if (person) {
    return (
      <div>
        <h2>{person.name}</h2>
        <div>{person.id}</div>
        <div>{person.phone}</div>
        <div>
          {person.address.street} - {person.address.city}
        </div>
        <button onClick={() => setPerson(null)}>Regresar</button>
      </div>
    );
  }

  if (persons === null) {
    return null;
  }

  return (
    <div
      style={{
        display: "flex",
        flexDirection: "column",
        justifyContent: "center", // Centra horizontalmente
        alignItems: "center", // Centra verticalmente
        backgroundColor: "#DAF7A6",
        padding: "20px",
      }}
    >
      <h1>Personas Graphql</h1>
      <ul>
        {persons.map((person, index) => (
          <li
            key={index}
            style={{ textAlign: "start" }}
            onClick={() => handlePerson(person.name)}
          >
            {person.name} - {person.phone}
          </li>
        ))}
      </ul>
    </div>
  );
};
```

- Ajustamos el componente principal para que llame al componente Person.jsx
- Uso de useQuery para realizar consulta de servicio graphql-server

```
// import { useEffect } from 'react'
import { gql, useQuery } from "@apollo/client";
import "./App.css";
import { Persons } from "./Persons";

const ALL_PERSONS = gql`
  query {
    allPersons {
      id
      name
      phone
      address {
        street
        city
      }
    }
  }
`;

function App() {

  const { loading, data, error } = useQuery(ALL_PERSONS);

  if (error) {
    return <span style={{ color: "red" }}>{error}</span>;
  }

  return (
    <>{loading ? <p>Cargando...</p> : <Persons persons={data?.allPersons} />}</>
  );
}

export default App;
```

- Se ajusto la data dummy del Grapghql-server para que tengan la misma data que la del db.json

Salida:



- Click en usuario de nombre Neymar