

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 rreact-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

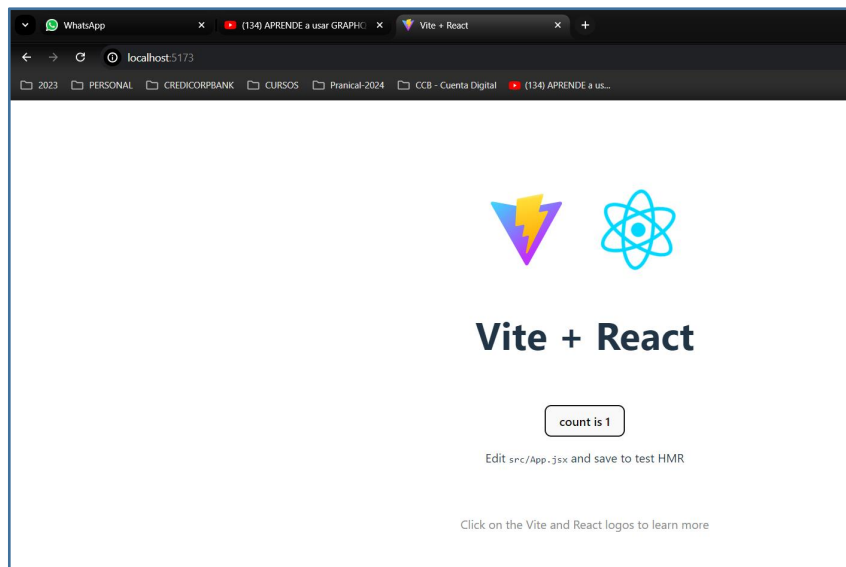
```
loper@wlopera MINGW64 /c/A_CURSOS/2024/GraphQL/react-graphql
$ npm run
Scripts available in react-graphql@0.0.0 via `npm run-script`:
  dev
  vite
  build
  vite build
  lint
  eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0
  preview
  vite preview

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'

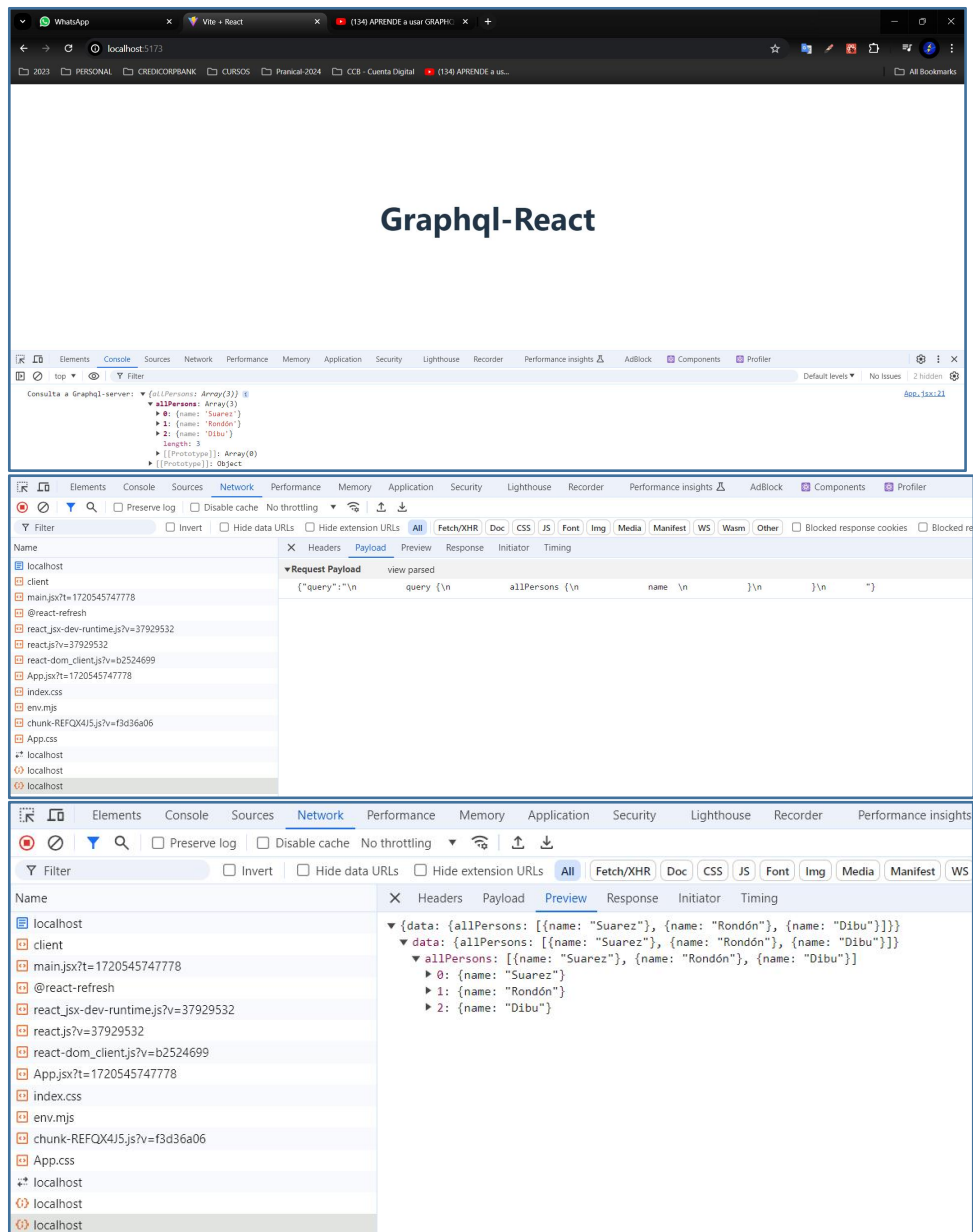
function App() {

  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

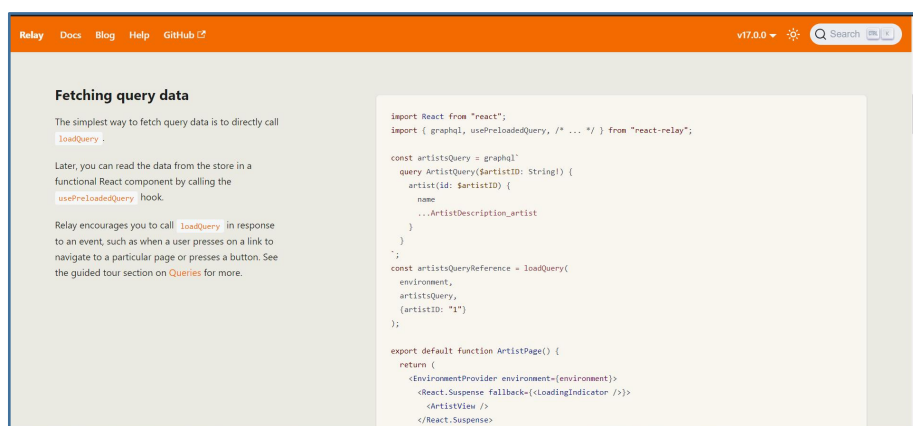
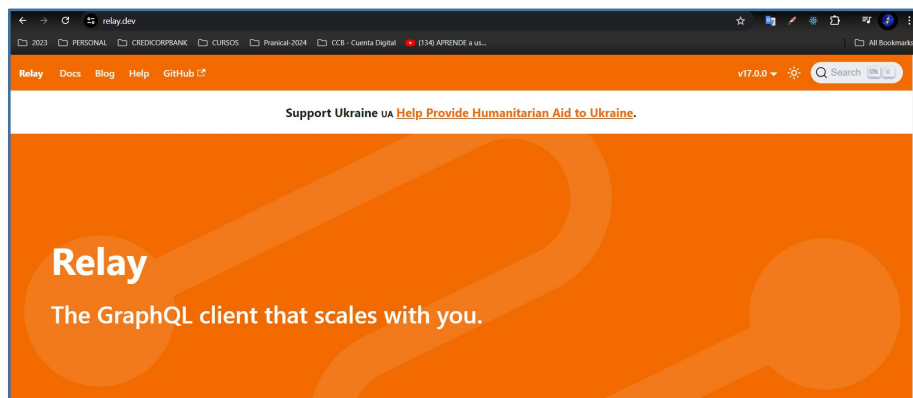


- Pero para existen clientes de graphql que permite un mejor control y desarrollo de las Aplicaciones del lado del cliente.
- Relay_dev: Utilizado por Facebook para proyectos muy grandes

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.

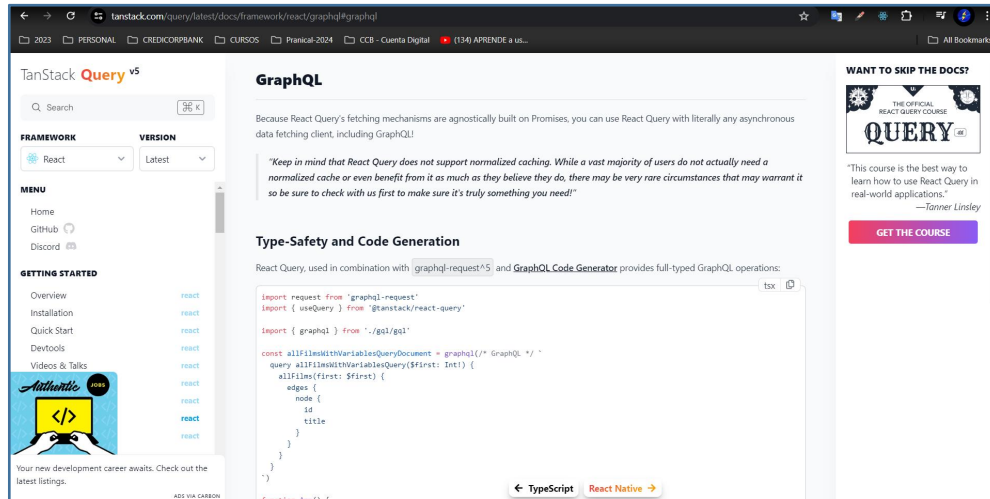
<https://relay.dev/>



<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>

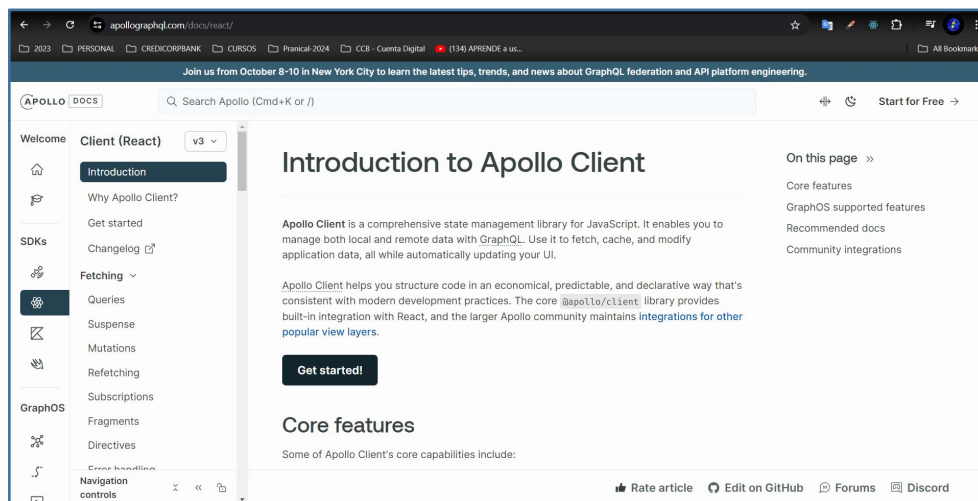
<https://tanstack.com/query/latest/docs/framework/react/graphql#graphql>

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/client biblioteca 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
```

```
_> npm install graphql
```

```
loper@wlopera MINGW64 /c/A_CURSOS/2024/GraphQL/react-graphql
$ npm i @apollo/client graphql

added 18 packages, and audited 297 packages in 6s

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

found 0 vulnerabilities
```

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:

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

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

Salida:

React - GraphQL

localhost:5173

2023 PERSONAL CREDICORPBANK CURSOS Pranical-2024 CCB - Cuenta Digital (134) APRENDE a us...

Elements Console Sources Network Performance Memory Application

top Filter

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: "Suauez"
    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 acceso a variable en mi main.js

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

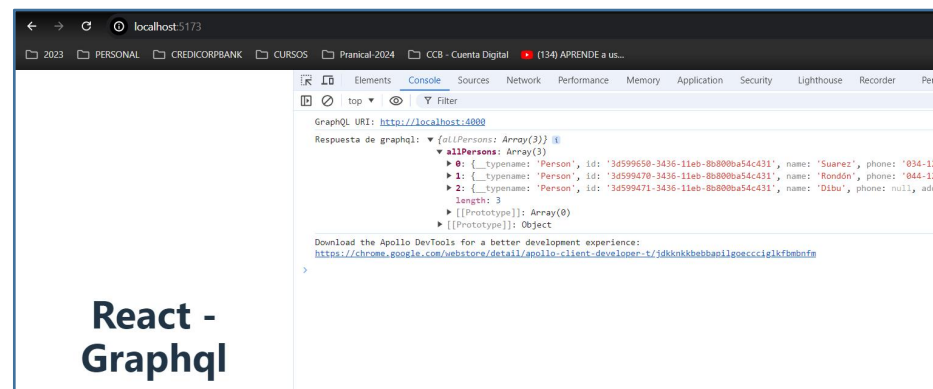
- Crear ambiente desarrollo

.env:

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

- Levantar react como desarrollo (npm run dev)

```
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 136 ms  
  
+ Local:   http://localhost:5173/  
+ Network: use --host to expose
```



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

`.env.production`

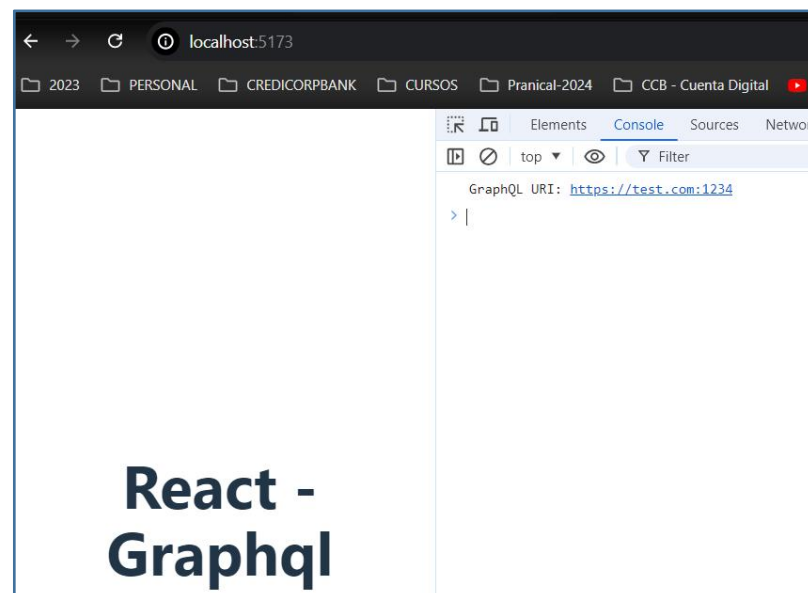
```
VITE_GRAPHQL_URI=https://test.com:1234
```

```
MINGW64:/c/A_CURSOS/2024/GraphQL/react-graphql

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

> react-graphql@0.0.0 dev
> vite --mode production

VITE v5.3.3 ready in 136 ms
+ Local:   http://localhost:5173/
+ Network: use --host to expose
```



- 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)
  //   }

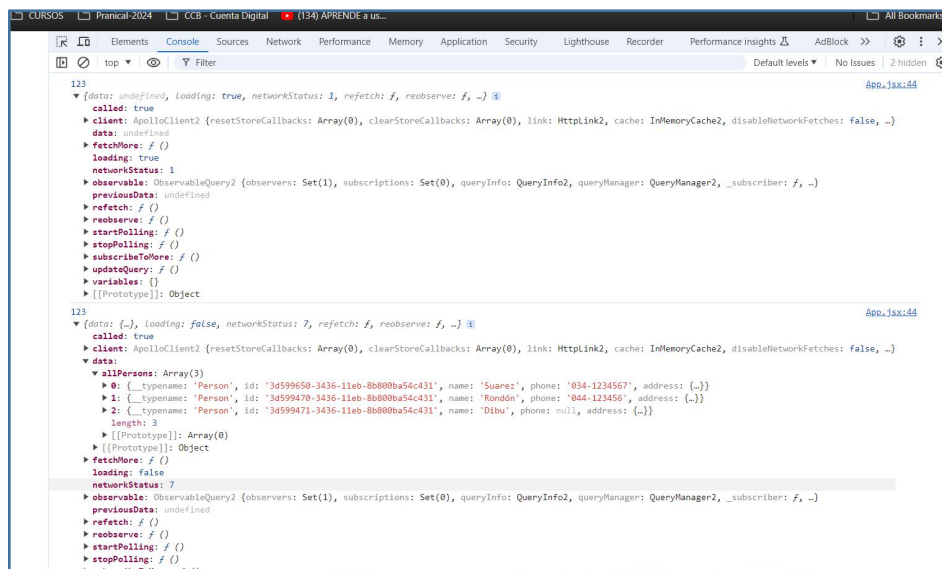
  //   queryGraphQLData()
  // }, [])

  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);

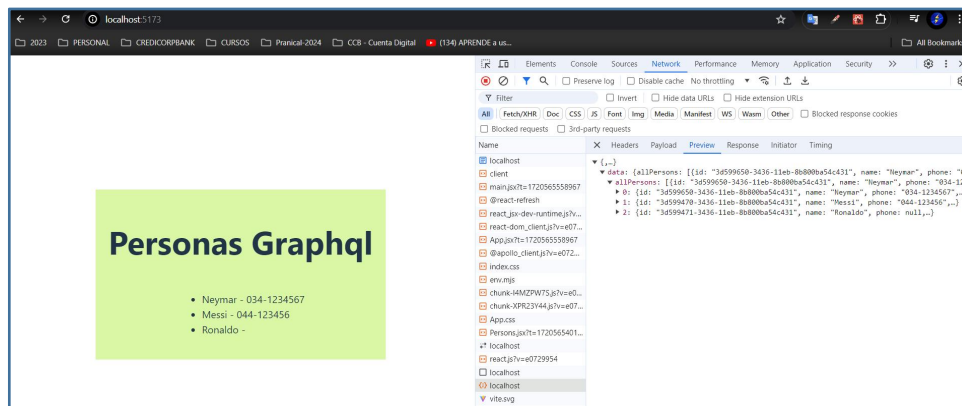
  if (error) {
    return <span style={{ color: "red" }}>{error}</span>;
  }

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

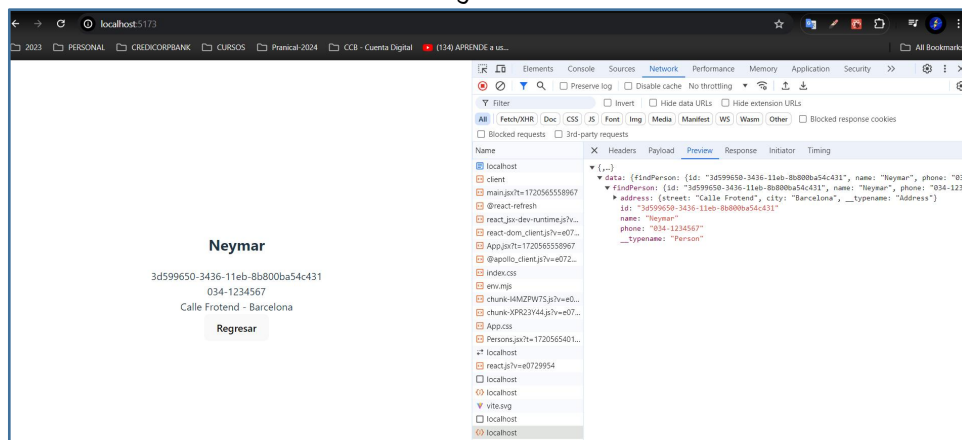
export default App;
```

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

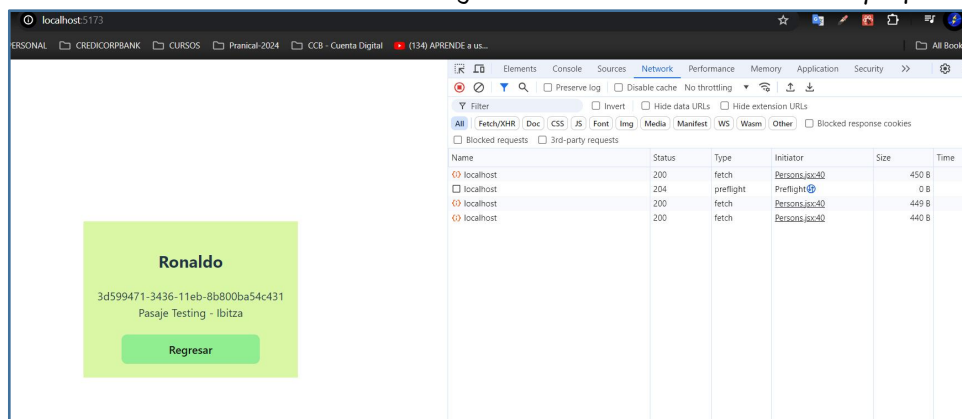
Salida:



- Click en usuario de nombre Neymar

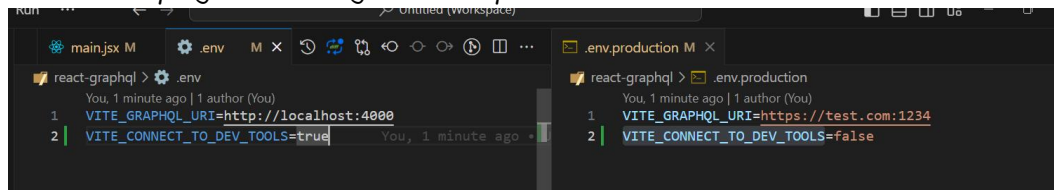


- Se puede observar que si una petición se vuelve a realizar, si es la misma, se retorna la data de la cache y no se consulta al servidor GraphQL-server.

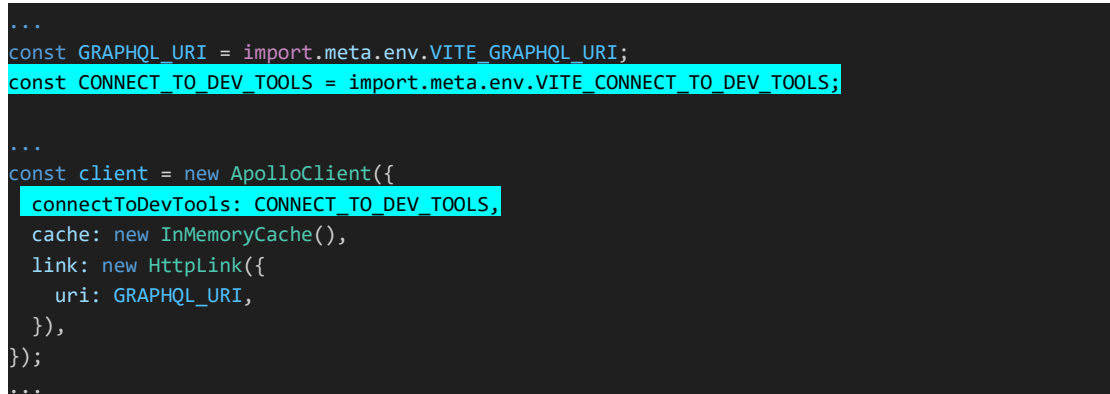


Como ya se invoca el llamado a la data Ronaldo se toma de la cache. Si refresco la vista se pierde por el tipo de cache. Se puede utilizar localStorage u otro tipo de manejo de cache (ex: base de datos) para que persista la data en cache

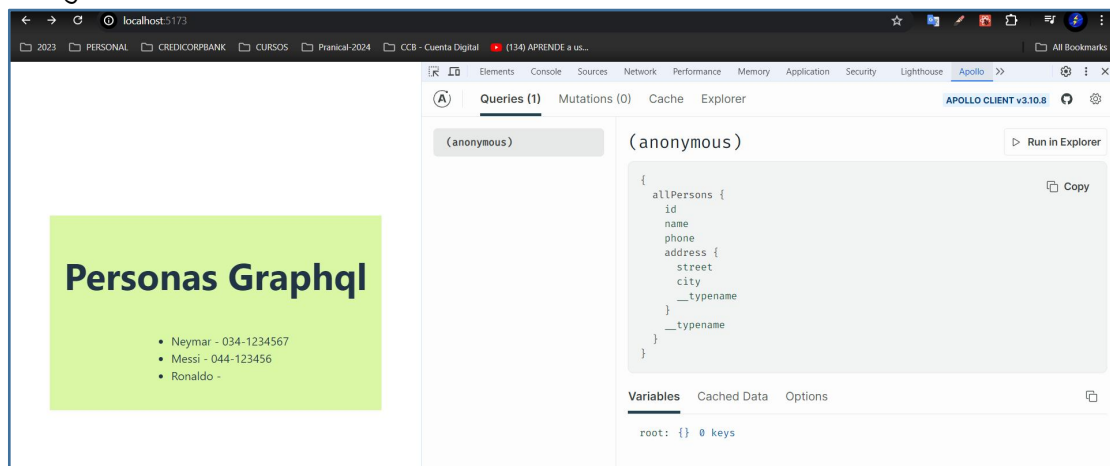
- *Uso de plugin del navegador de Apollo - devTools*



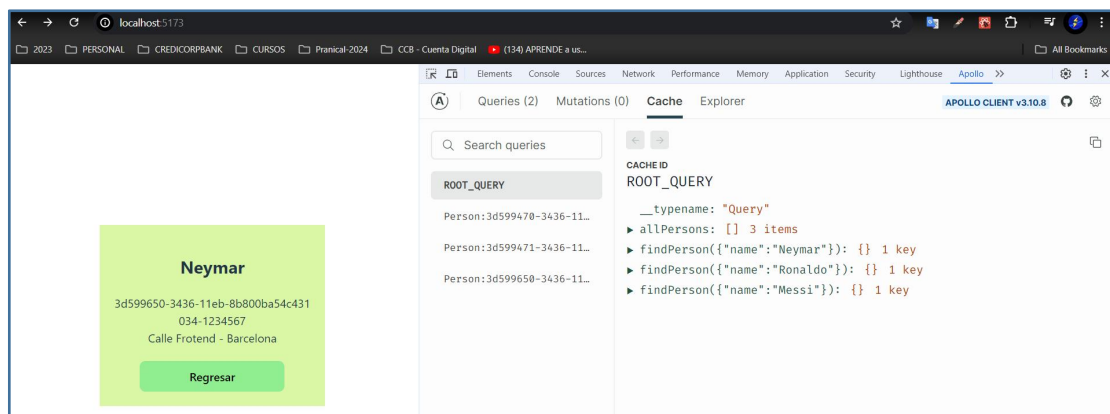
Main.jsx

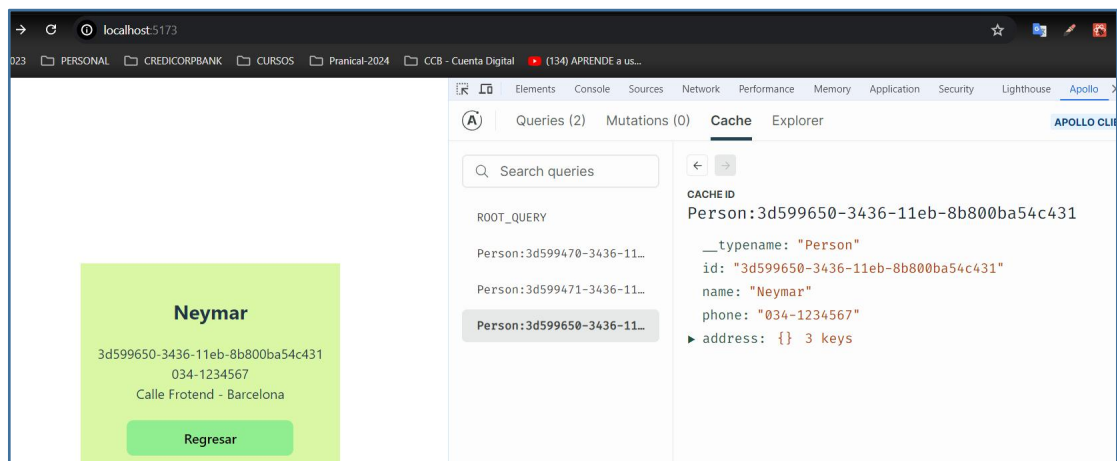


Levantar servidor como desarrollo para activar el devTools y probarlo en el navegador



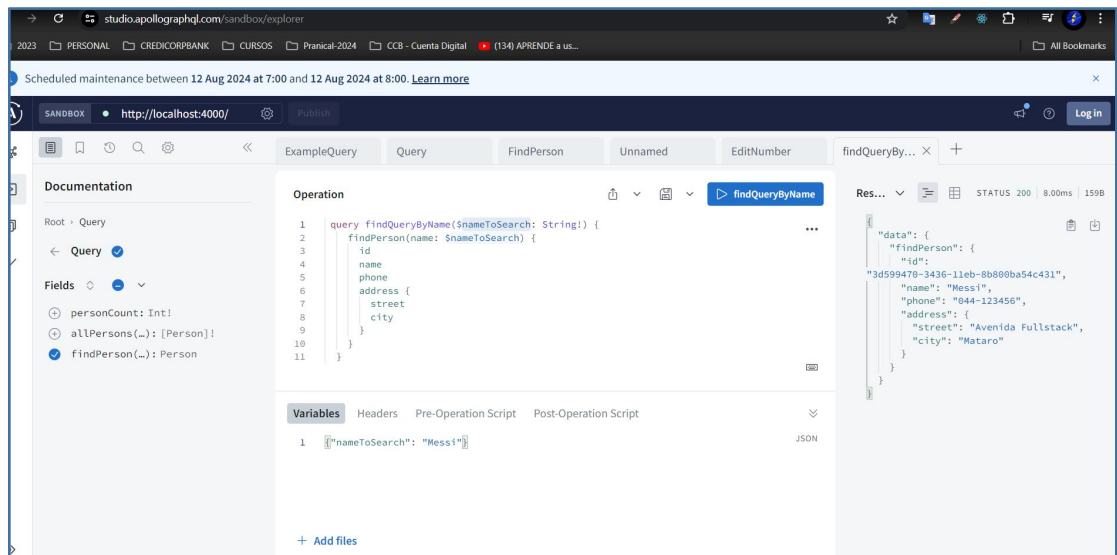
Aqui podemos ver las queries, las mutaciones y la cache que se genera y utiliza para consultas al vuelo





Uso de variables en el ambiente GraphQL-server:

```
query findQueryByName($nameToSearch: String!) {
  findPerson(name: $nameToSearch) {
    id
    name
    phone
    address {
      street
      city
    }
  }
}
```



Vamos a colocar la data de Personas en GraphQL-Server como dummy para poder probar las mutaciones ya que necesitamos haber definido un api en json-server para poder agregar Personas y no esta programado en este ejemplo

- Crear una mutación - agregar una persona

```
import { useState } from "react";
import { gql, useMutation } from "@apollo/client";

const CREATE_PERSON = gql`
  mutation createPerson(
    $name: String!
    $phone: String
    $street: String!
    $city: String!
  ) {
    addPerson(name: $name, phone: $phone, street: $street, city: $city) {
      id
      name
      phone
      address {
        street
        city
      }
    }
  }
`;

const divStyle = {
  display: "flex",
  flexDirection: "column",
  justifyContent: "center", // Centra horizontalmente
  alignItems: "center", // Centra verticalmente
  backgroundColor: "#CCD1D1",
  padding: "20px",
  marginTop: "20px",
};

export const PersonForm = () => {
  const [name, setName] = useState("");
  const [phone, setPhone] = useState("");
  const [street, setStreet] = useState("");
  const [city, setCity] = useState("");

  const [createPerson] = useMutation(CREATE_PERSON);

  const handleSubmit = (e) => {
    e.preventDefault();

    createPerson({ variables: { name, phone, street, city } });

    setName("");
    setPhone("");
    setStreet("");
    setCity("");
  };
};
```

```

return (
  <div style={divStyle}>
    <h2>Nueva Persona</h2>
    <form
      onSubmit={handleSubmit}
      style={{ display: "flex", flexDirection: "column", width: "40vh" }}
    >
      <input
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Nombre"
        style={{ marginBottom: "5px" }}
      />
      <input
        value={phone}
        onChange={(e) => setPhone(e.target.value)}
        placeholder="Teléfono"
        style={{ marginBottom: "5px" }}
      />
      <input
        value={street}
        onChange={(e) => setStreet(e.target.value)}
        placeholder="Calle"
        style={{ marginBottom: "5px" }}
      />
      <input
        value={city}
        onChange={(e) => setCity(e.target.value)}
        placeholder="Ciudad"
        style={{ marginBottom: "5px" }}
      />
      <input type="submit" value="Agregar" />
    </form>
  </div>
);
};

```

Se agrega la persona pero no se actualiza la data (si se observan nuevos datos, es porque se realizaron varias pruebas previamente)

Personas GraphQL

- Neymar - 034-1234567
- Messi - 044-123456
- Ronaldo -
-
- Arnago -

Nueva Persona

- Luego de agregar la nueva persona. Se observa que se agrego pero no se refleja el cambio en la data total a menos que se refresque la vista

Personas GraphQL

- Neymar - 034-1234567
- Messi - 044-123456
- Ronaldo -
-
- Arnago -

Nueva Persona

Nombre
Teléfono
Calle
Ciudad
Agregar

Network Tab:

```

{
  "data": {
    "addPerson": {
      "id": "5780f400-3ee0-11ef-b0d2-fb2e878532d0",
      "name": "PerezGil",
      "phone": "234-56472",
      "address": {
        "street": "Las Brisas",
        "city": "Canada",
        "__typename": "Address"
      }
    }
  }
}

```

- Si refresco la vista. Se ve el cambio

Personas GraphQL

- Neymar - 034-1234567
- Messi - 044-123456
- Ronaldo -
-
- Arnago -
- PerezGil - 234-56472

Nueva Persona

Nombre
Teléfono
Calle
Ciudad
Agregar

Network Tab:

Name	Status	Type	Initiator	Size	Time
localhost	304	document	Other	145 B	6 ms
client	304	script	index111	145 B	11 ms
main.jsx?t=1720631823695	304	script	index120	145 B	12 ms
@react-refresh	304	script	localhost/5	145 B	6 ms
react_jsx-dev-runtime.js?v=e0729954	200	script	main.jsx:1	(disk cache)	4 ms
react-dom_client.js?v=e0729954	200	script	main.jsx:2	(disk cache)	4 ms
App.jsx?t=1720631823695	304	script	main.jsx:3	145 B	5 ms
@apollo_client.js?v=e0729954	200	script	main.jsx:6	(disk cache)	7 ms
index.css	304	script	main.jsx:10	145 B	6 ms
env.mjs	304	script	client:1	145 B	6 ms
chunk-H4MZPW75.js?v=e0729954	200	script	react_jsx-dev-runtime.js:4	(disk cache)	3 ms
chunk-XPR23Y44.js?v=e0729954	200	script	react-dom_client.js:3	(disk cache)	7 ms
App.css	304	script	App.jsx:19	145 B	3 ms
Persons.jsx	304	script	App.jsx:20	145 B	3 ms
PersonForm.jsx?t=1720631823695	304	script	App.jsx:21	145 B	4 ms
localhost	101	websocket	client:535	0 B	Pending
react.js?v=e0729954	200	script	Persons.jsx:4	(disk cache)	1 ms
localhost	204	preflight	Preflight	0 B	4 ms
localhost	200	fetch	@apollo_client.js?v=e0729954	1.3 kB	3 ms
vite.svg	304	svg+xml	Other	145 B	3 ms

Voy a agregar dos formas de actualizar la data de manera automática para actualizar la nueva data

- *Método1: Agregar ajuste para que la data se refresque cada cierto tiempo (ex: 2 seg). Agregar parámetro de intervalo en mi useQuery de consulta en App.jsx*

```
...
const { loading, data, error } = useQuery(ALL_PERSONS, {
  pollInterval: 2000,
});
...
```

Personas GraphQL

- Neymar - 034-1234567
- Messi - 044-123456
- Ronaldo -
-
- Arnago -
- PerezGil - 234-56472

Nueva Persona

Nombre

Teléfono

Calle

Ciudad

Name	Status	Type	Initiator	Size	Time
localhost	204	preflight	Preflight	0 B	2
localhost	200	fetch	@apollo_client.js?v=e072995	1.3 kB	2
localhost	200	fetch	@apollo_client.js?v=e072995	1.3 kB	2
localhost	200	fetch	@apollo_client.js?v=e072995	1.3 kB	3
localhost	200	fetch	PersonForm.js:c44	449 B	2

Personas GraphQL

- Neymar - 034-1234567
- Messi - 044-123456
- Ronaldo -
-
- Arnago -
- PerezGil - 234-56472
- Luis - 5432-6475

Nueva Persona

Nombre

Teléfono

Calle

Ciudad

Name	Status	Type	Initiator	Size	Time
localhost	204	preflight	Preflight	0 B	
localhost	200	fetch	@apollo_client.js?v=e072995	1.3 kB	
localhost	200	fetch	@apollo_client.js?v=e072995	1.3 kB	
localhost	200	fetch	@apollo_client.js?v=e072995	1.3 kB	
localhost	200	fetch	PersonForm.js:c44	449 B	
localhost	204	preflight	Preflight	0 B	
localhost	200	fetch	@apollo_client.js?v=e072995	1.5 kB	
localhost	200	fetch	@apollo_client.js?v=e072995	1.5 kB	
localhost	200	fetch	@apollo_client.js?v=e072995	1.5 kB	
localhost	200	fetch	@apollo_client.js?v=e072995	1.5 kB	
localhost	204	preflight	Preflight	0 B	

El problema es que cada 2 seg se realiza una petición al servidor graphql-server

Luis

73dbe7d0-3ee1-11ef-b0d2-fb2e878532d0
5432-6475
Las Alemanas - Las Palmas

Nueva Persona

Nombre

Teléfono

Calle

Ciudad

- Método 2: Agregar un parámetro a la Mutación para que refresque luego de agregar la nueva persona
 - Para no repetir el código exporto el objeto, elimino el `pollInterval` (para que no se este llamando cada 2 seg) en `App.jsx`

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

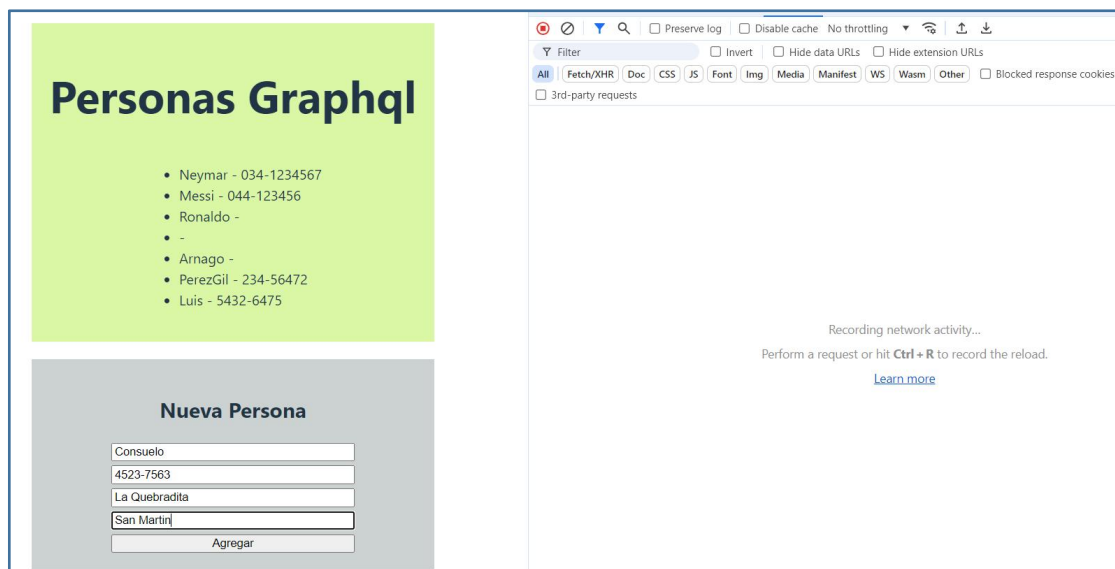
const { loading, data, error } = useQuery(ALL_PERSONS);
...

```

- Agregar el parámetro `refetchQueries` (importar el objeto `ALL_PERSON`) en la mutación en `PersonForm.jsx`

```
...
const [createPerson] = useMutation(CREATE_PERSON, {
  refetchQueries: [{ query: ALL_PERSONS }],
});
...

```



- Se realizan dos peticiones una mutación (agregar la persona) y luego consultar toda la data

Personas GraphQL

- Neymar - 034-1234567
- Messi - 044-123456
- Ronaldo -
-
- Arnago -
- PerezGil - 234-56472
- Luis - 5432-6475
- Consuelo - 4523-7563

Nueva Persona

Nombre
Teléfono
Calle
Ciudad
Agregar

Consuelo

bb78d010-3ee3-11ef-b0d2-fb2e878532d0
4523-7563
La Quebradita - San Martin

Regresar

Nueva Persona

Nombre
Teléfono
Calle
Ciudad
Agregar

Network Traffic:

Request 1 (POST):

```
{
  "data": {
    "addPerson": {
      "id": "3d599658-3436-11eb-8080ba54c431",
      "name": "Neymar",
      "phone": "034-1234567",
      "address": ""
    }
  }
}
```

Request 2 (GET):

```
{
  "data": {
    "allPersons": [
      {
        "id": "3d599658-3436-11eb-8080ba54c431",
        "name": "Neymar",
        "phone": "034-1234567",
        "address": ""
      },
      {
        "id": "3d599470-3436-11eb-8080ba54c431",
        "name": "Messi",
        "phone": "044-123456",
        "address": ""
      },
      {
        "id": "3d599471-3436-11eb-8080ba54c431",
        "name": "Ronaldo",
        "phone": null,
        "address": ""
      },
      {
        "id": "ff08bb68-3edc-11ef-b0d2-fb2e878532d0",
        "name": "",
        "phone": "",
        "address": ""
      },
      {
        "id": "0d63bef0-3edd-11ef-b0d2-fb2e878532d0",
        "name": "Arnago",
        "phone": "",
        "address": ""
      },
      {
        "id": "5780f400-3ae0-11ef-b0d2-fb2e878532d0",
        "name": "PerezGil",
        "phone": "234-56472",
        "address": ""
      },
      {
        "id": "73dbef40-3ee3-11ef-b0d2-fb2e878532d0",
        "name": "Luis",
        "phone": "5432-6475",
        "address": ""
      },
      {
        "id": "bb78d010-3ee3-11ef-b0d2-fb2e878532d0",
        "name": "Consuelo",
        "phone": "4523-7563",
        "address": ""
      }
    ]
  }
}
```

Otra forma de refrescar automáticamente sería con suscripciones.

