**PROYECTO REACT-GRAPHQL**

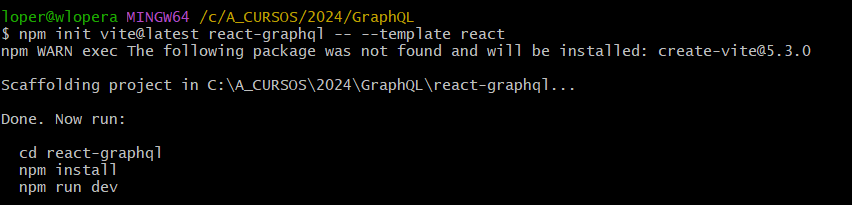
Aplicación de React para consumir APIs mediante el servicio Graphql

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

**Crear proyecto React**

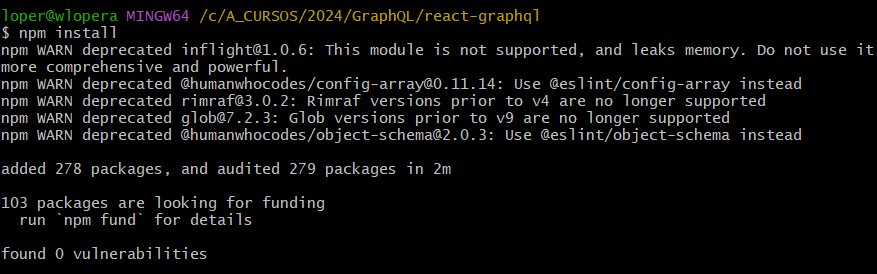
* Crear proyecto con vitejs y con plantilla de react

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



* Instalar librerías (cd rteact-graphql)

\_> npm install

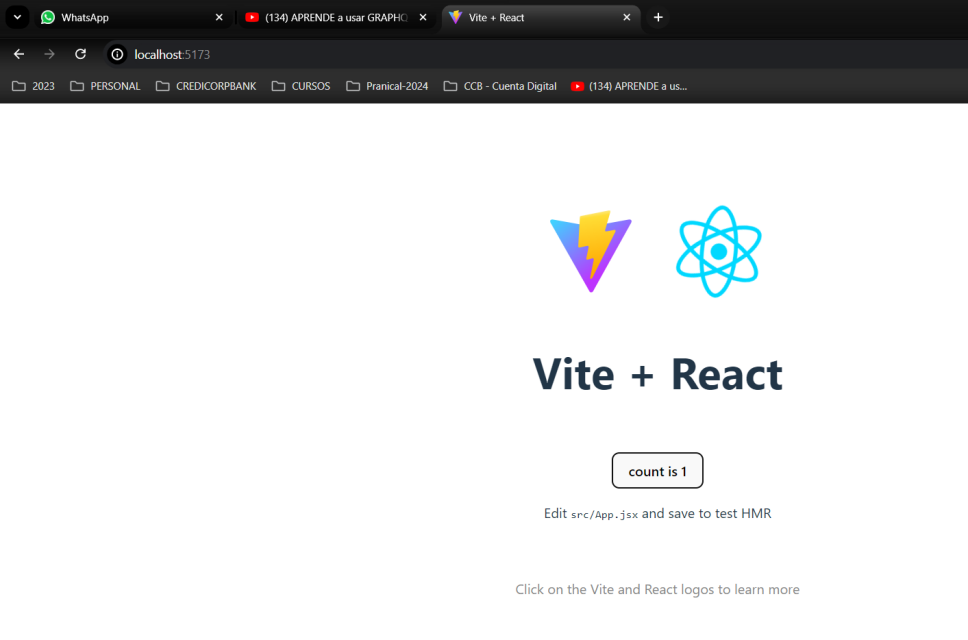


* Levantar y probar APP de react

\_> npm run dev



* 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>

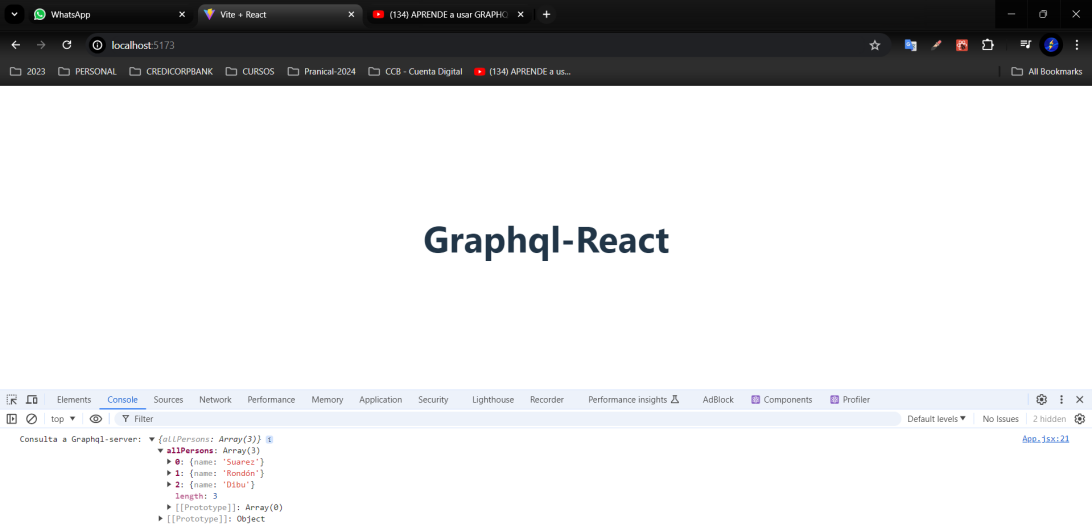
    </>

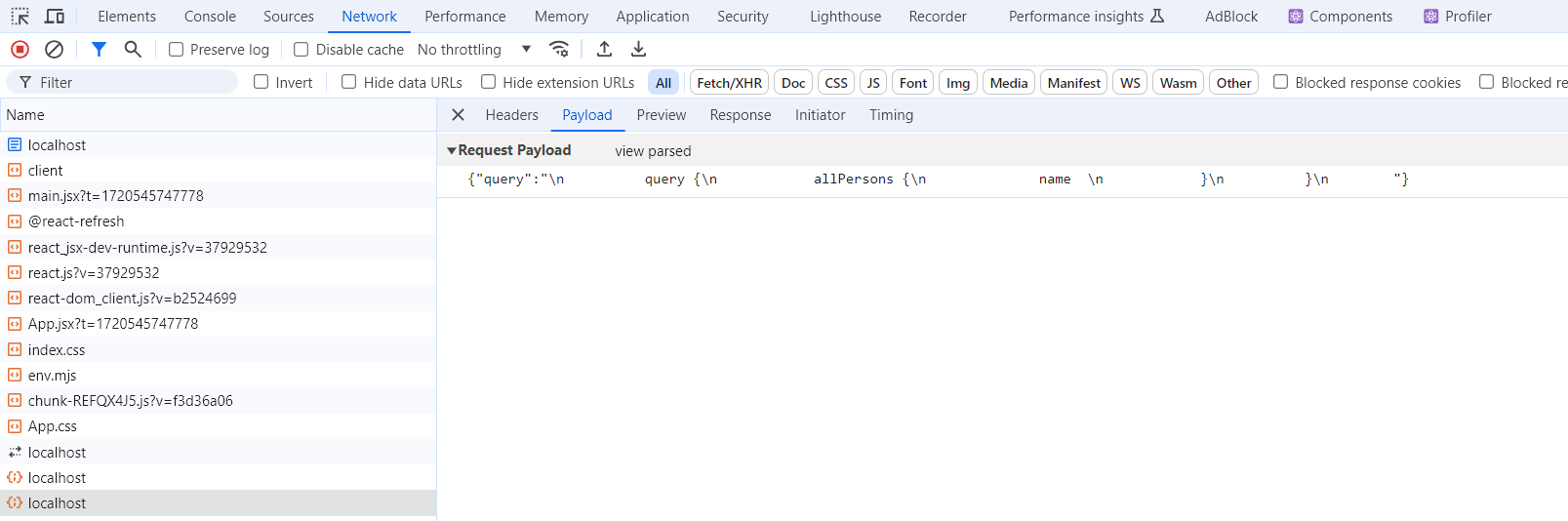
  )

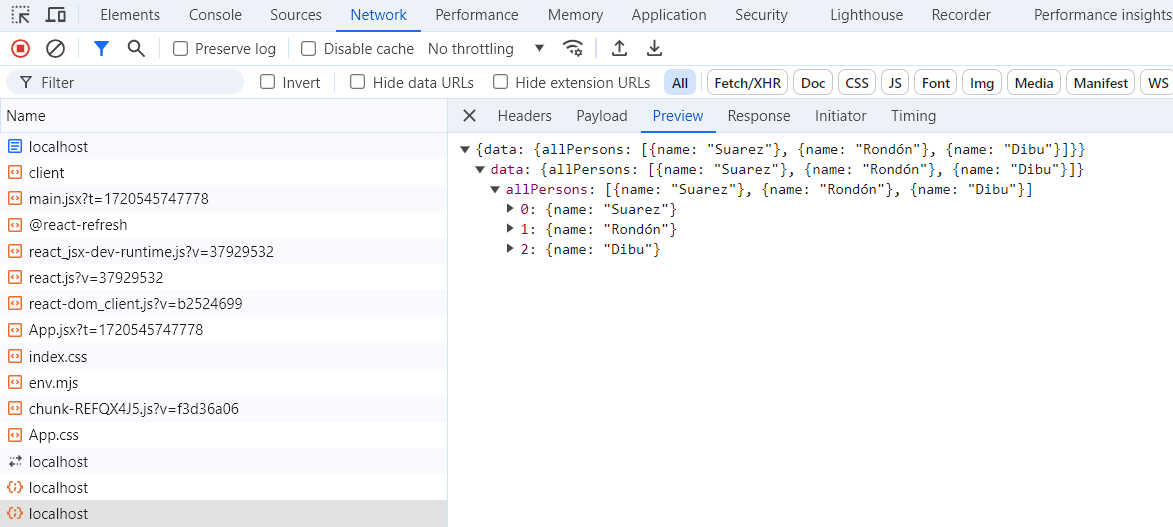
}

export default App

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





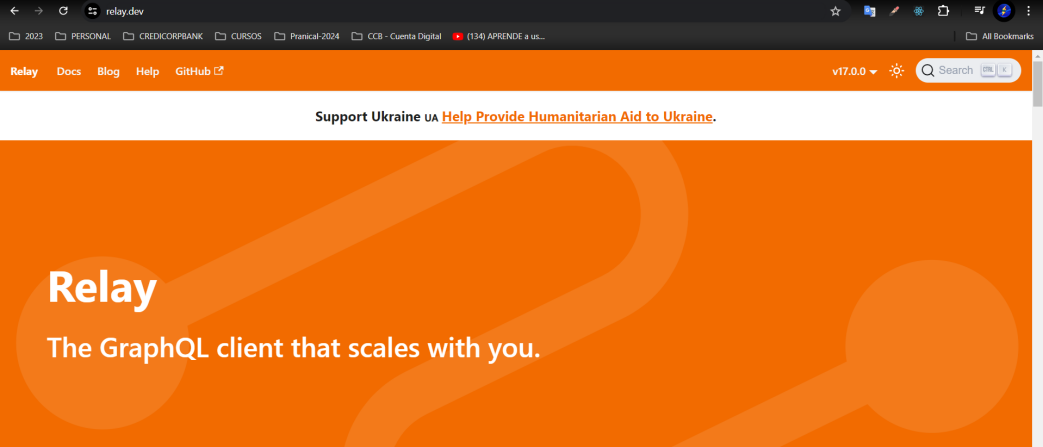


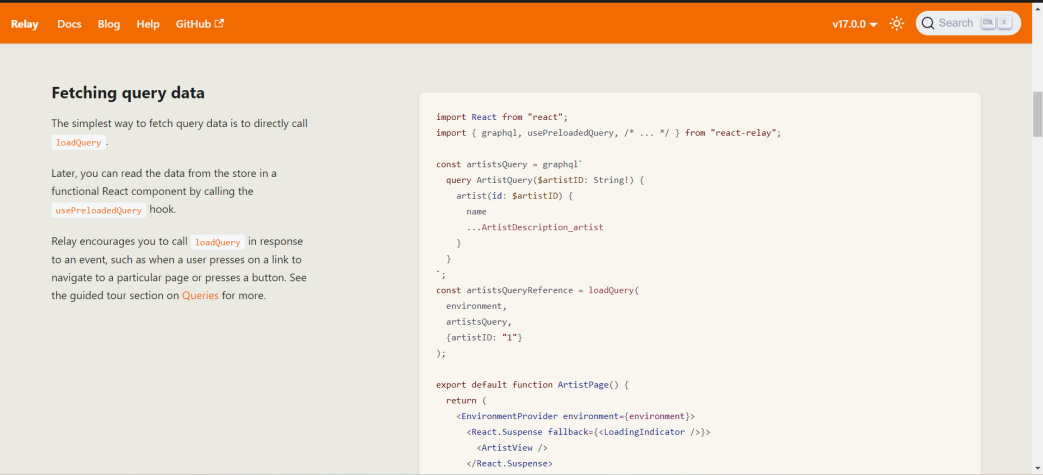
* Pero para existen clientes de graphql que permite un mejor control y desarrollo de las Aplicaciones del ado del cliente.
  + Relay\_dev: Utilizado por Facebook para proyectos muy grandes

## 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/>

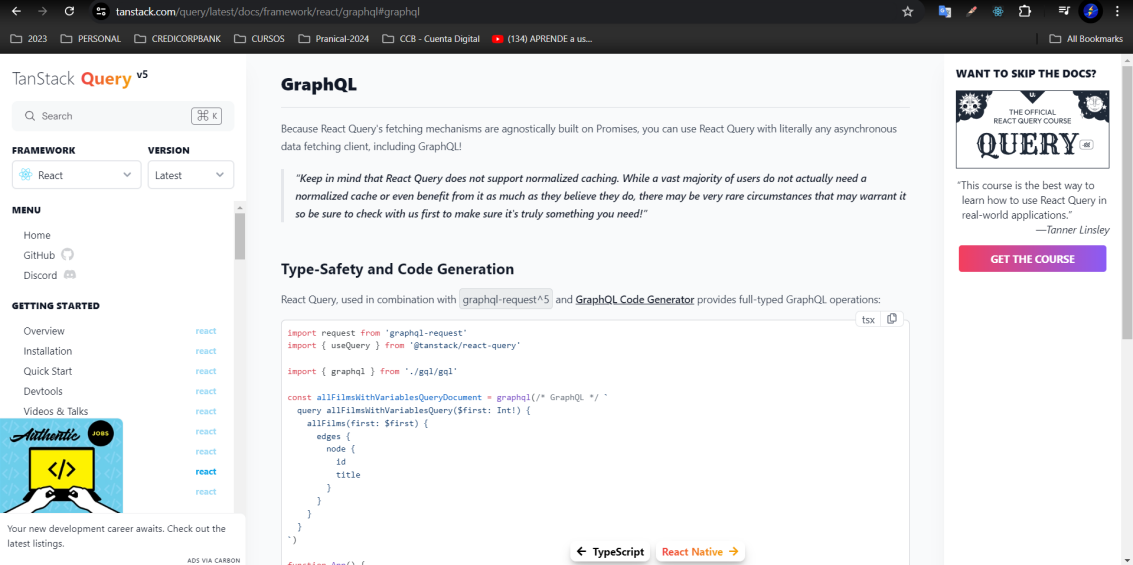




<https://developers.facebook.com/videos/2019/building-the-new-facebookcom-with-react-graphql-and-relay/>

* + 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>

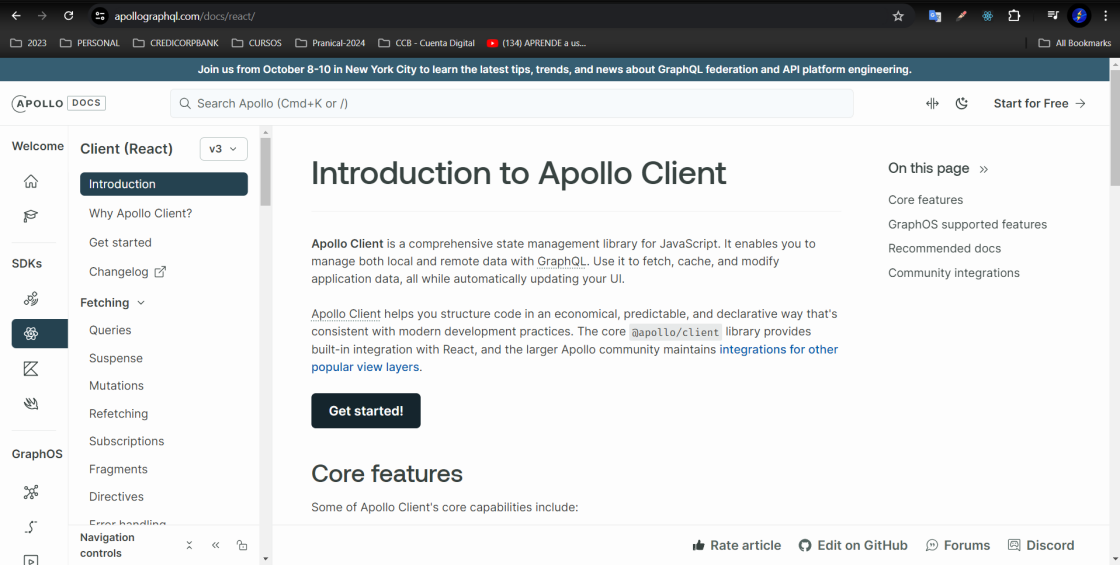
<https://tanstack.com/query/latest/docs/framework/react/graphql#graphql>

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/>

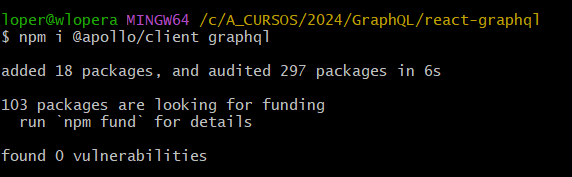


**Vamos a utilizar el cliente de Apollo**

* 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>,

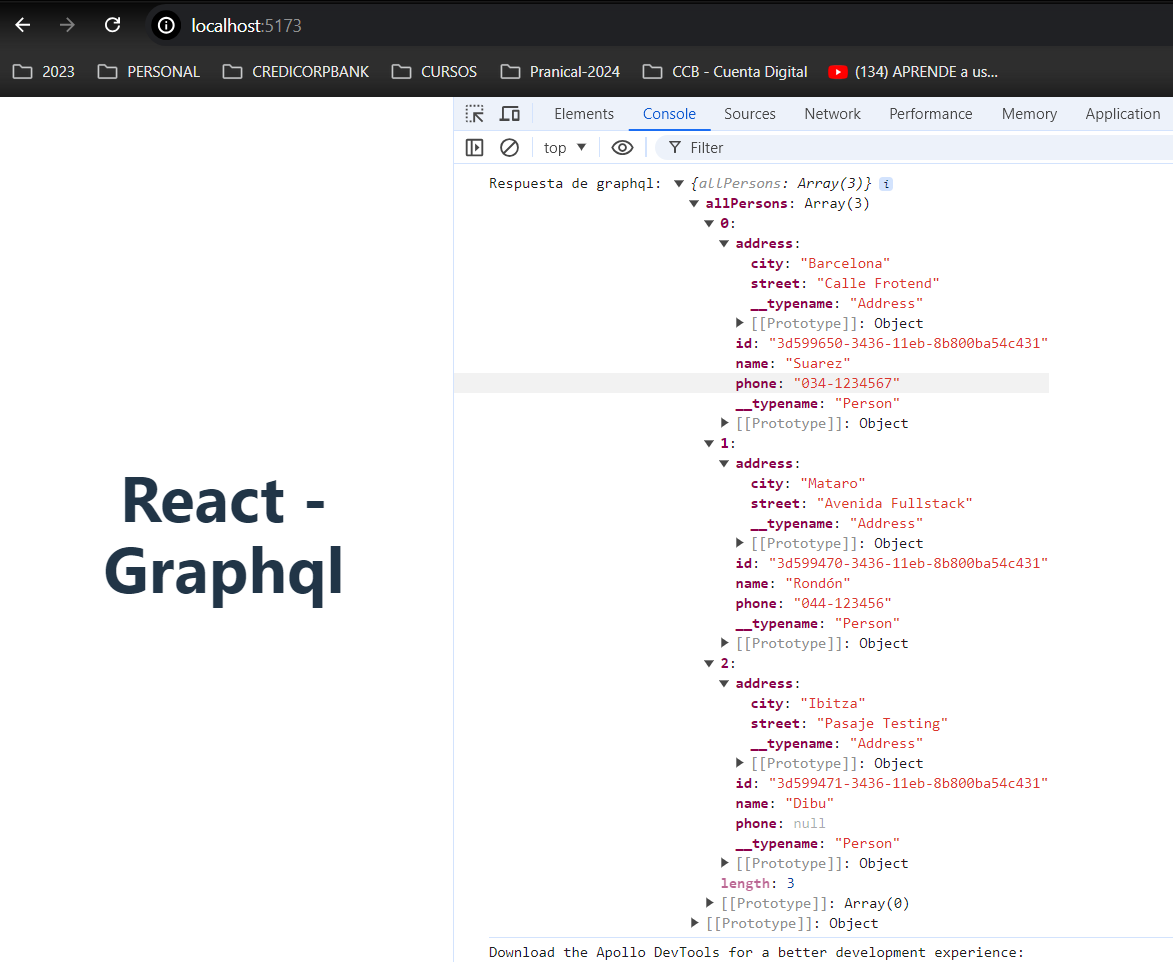
)

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:



**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);

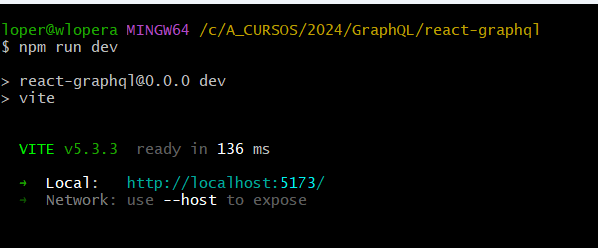
...

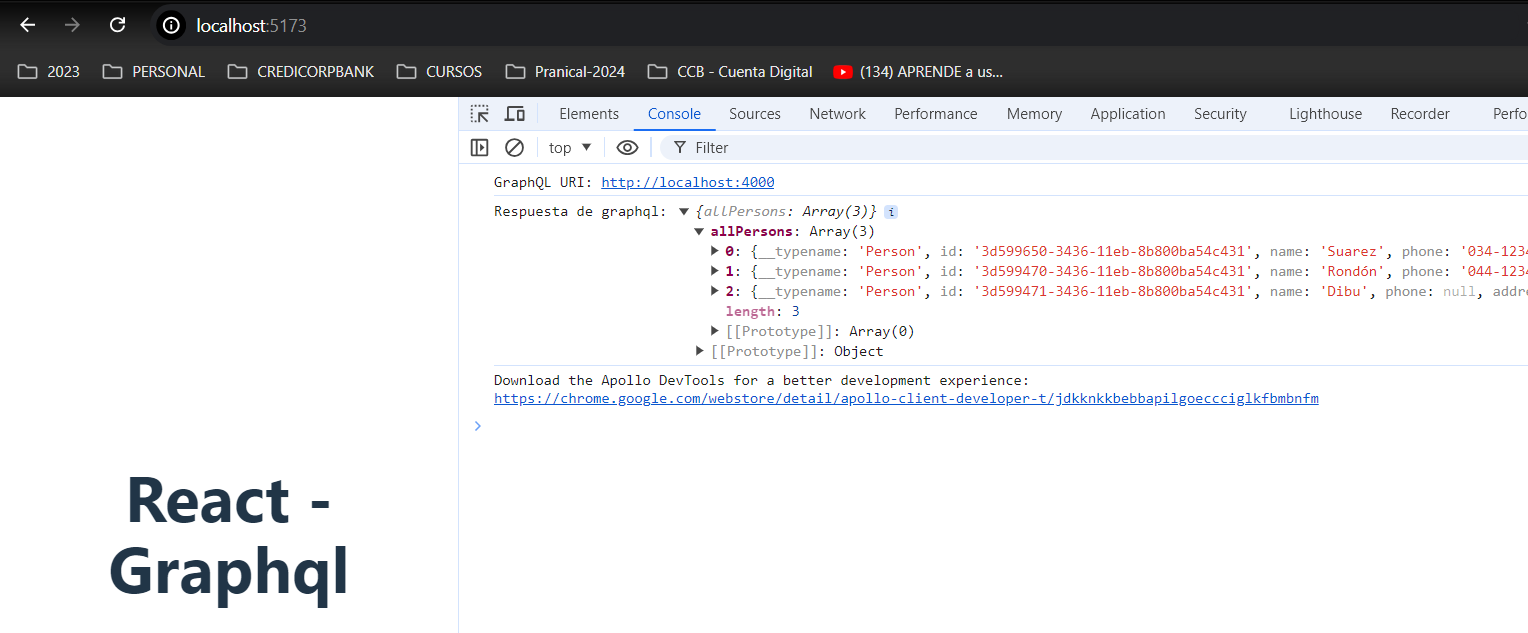
* Crear ambiente desarrollo

.env:

VITE\_GRAPHQL\_URI=http://localhost:4000

* Levantar react como desarrollo ( npm run dev )



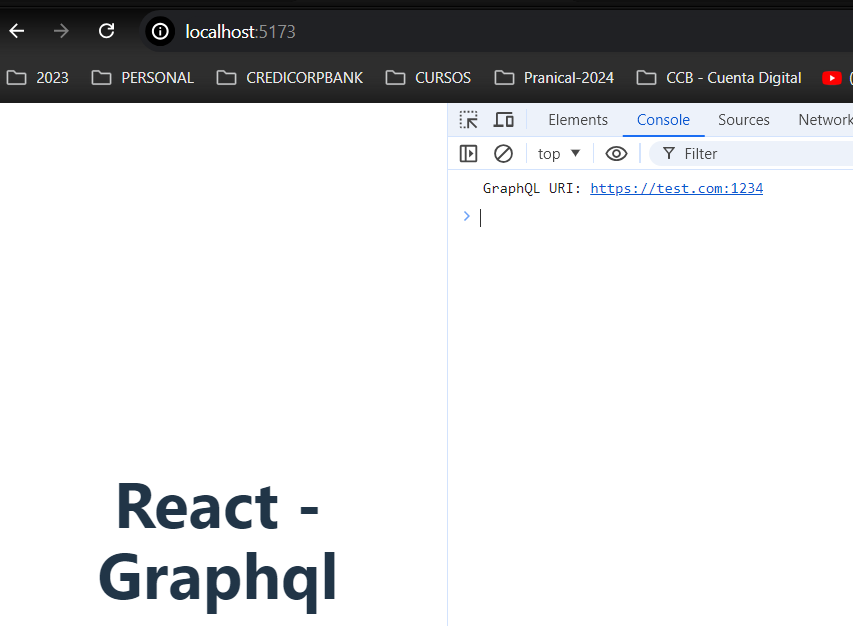


* + Crear ambiente producción (npm run dev -- --mode production)

.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>

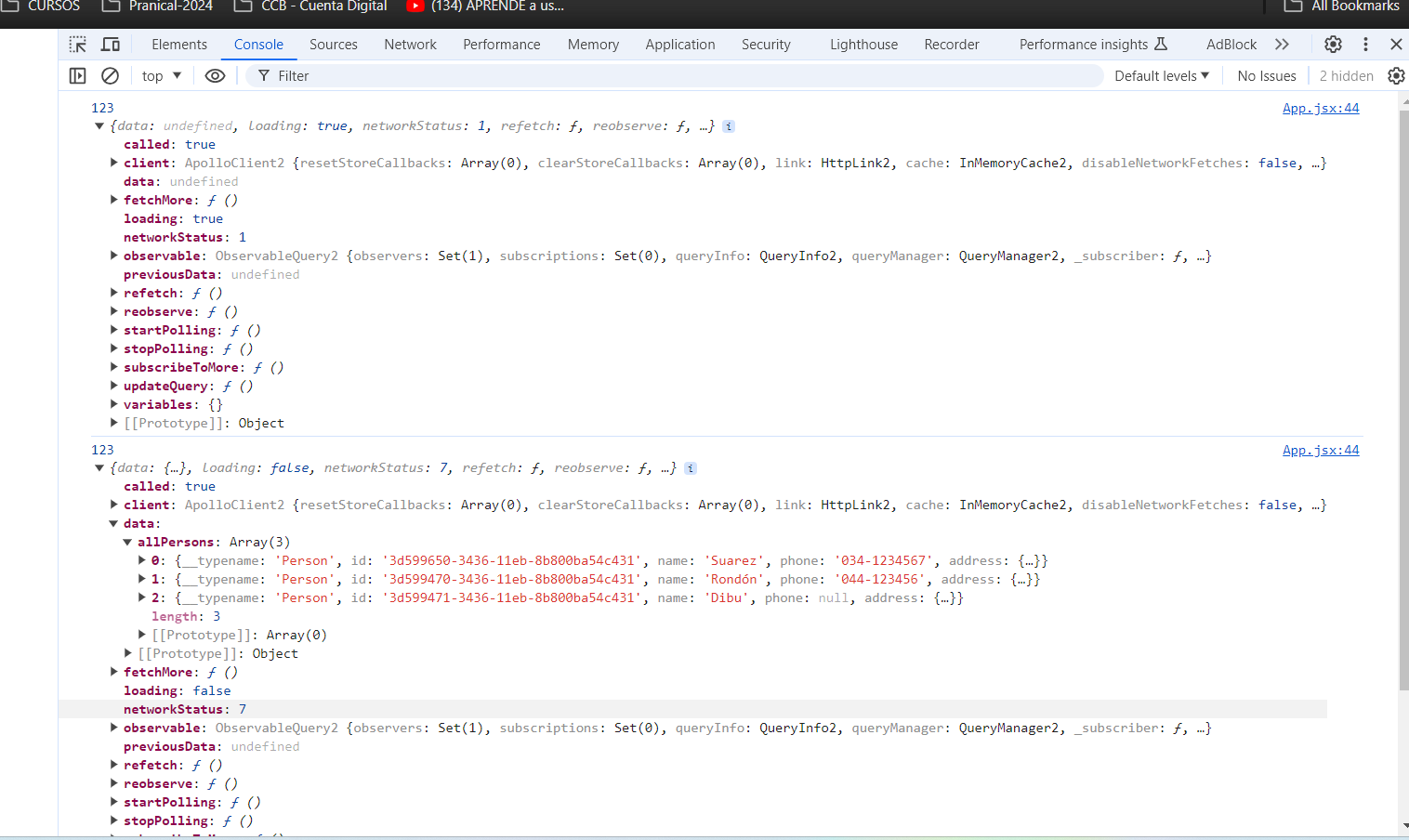
    </>

  )

}

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} />}</>

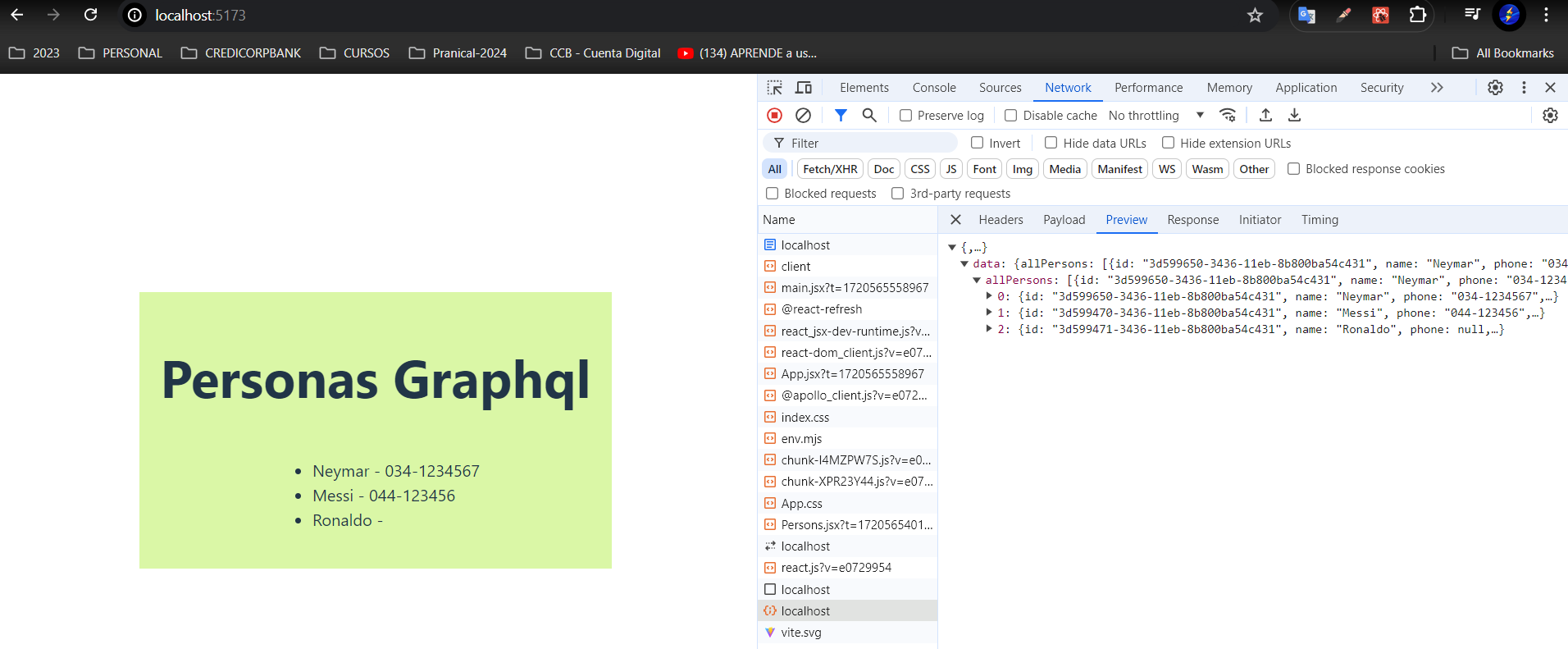
  );

}

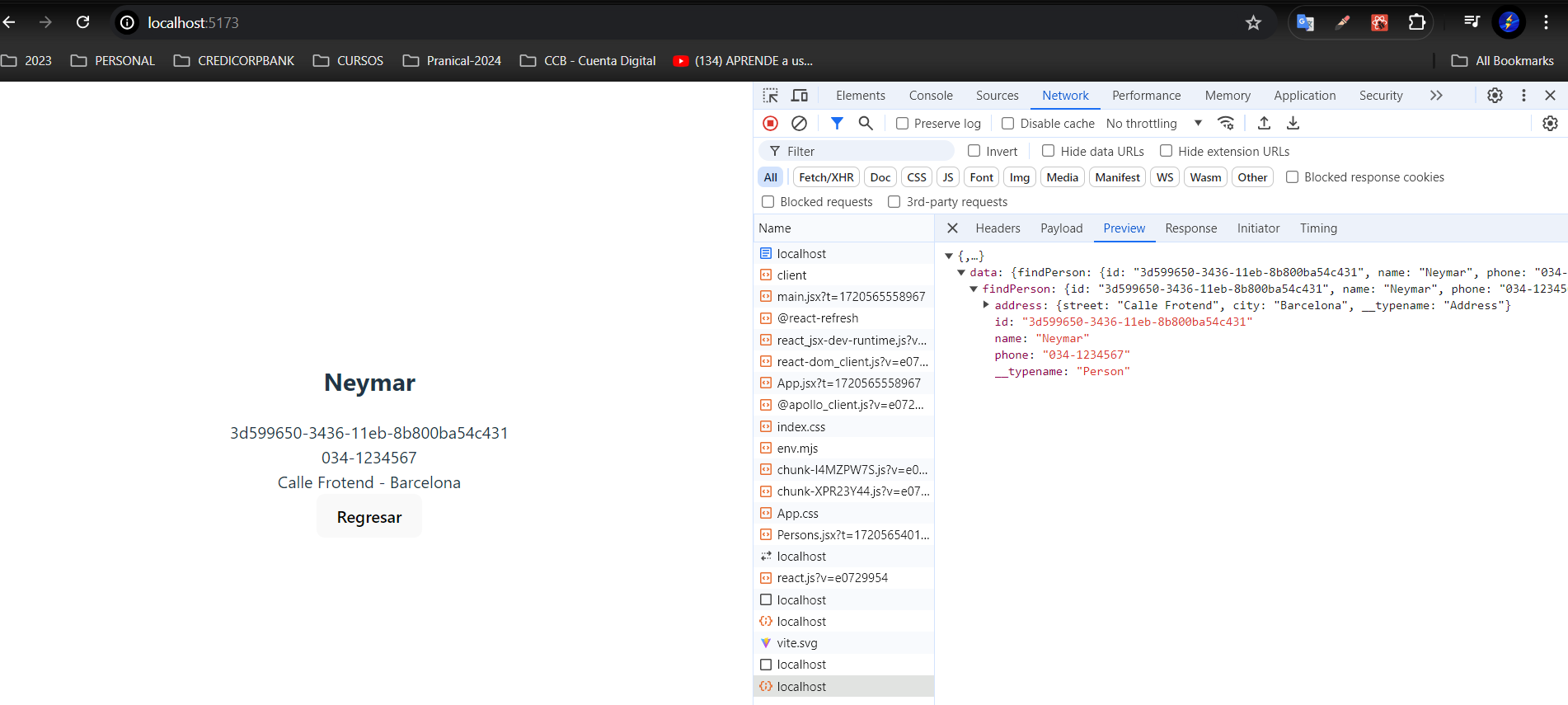
export default App;

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

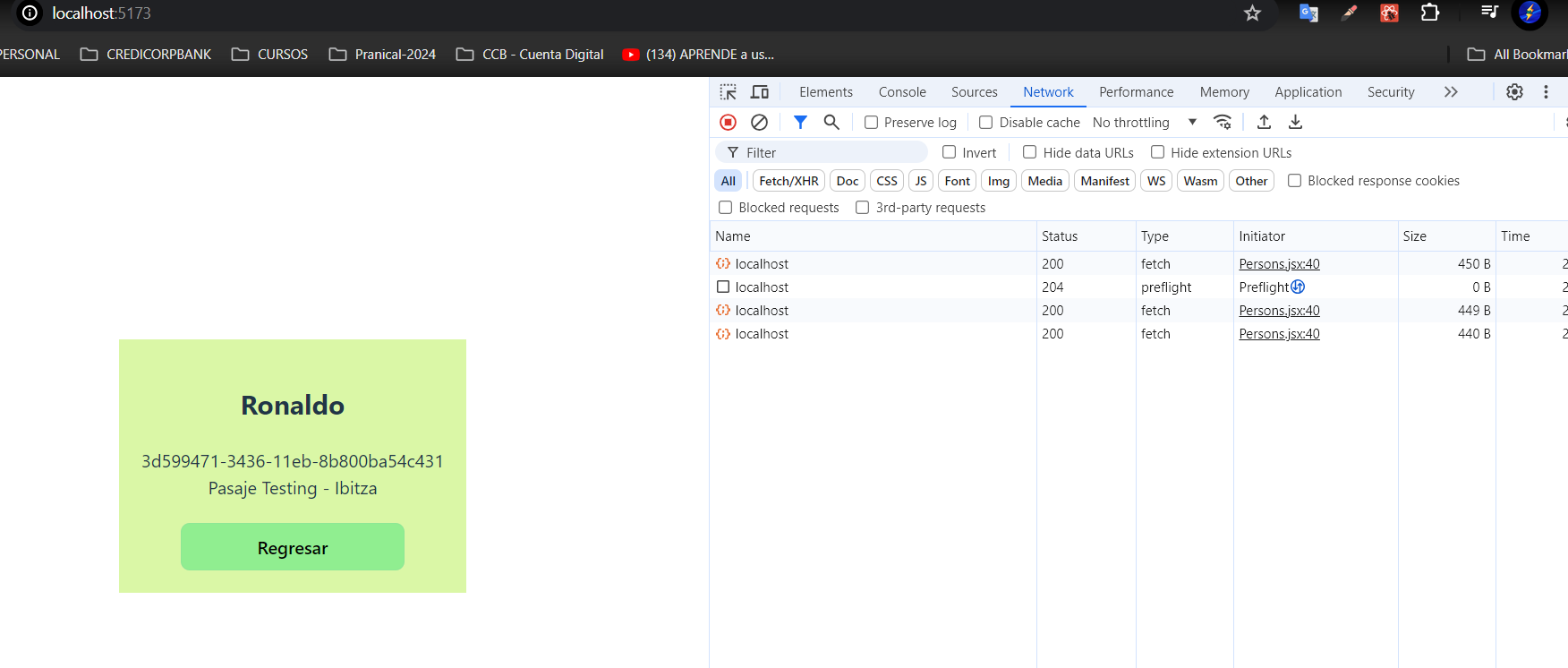
**Salida:**



* Click en usuario de nombre Neymar

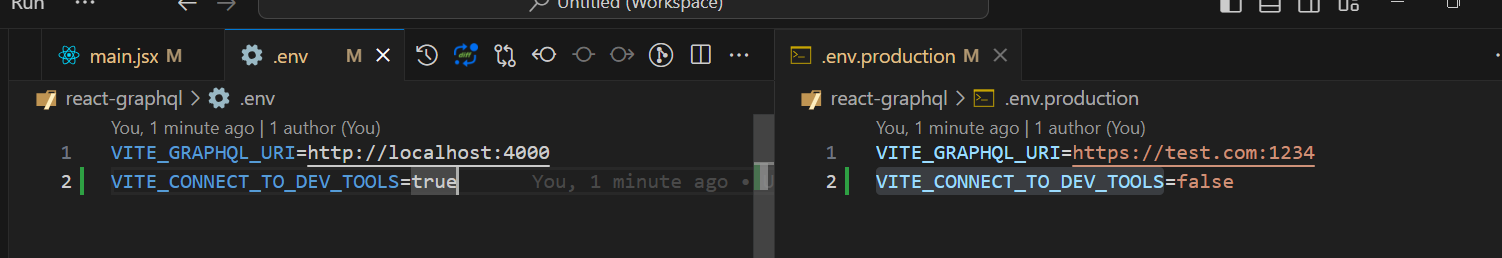


* Se puede observar que si una petición se vuelve a realizar, si es la misma, se retorna la data de la cache y no se consulta al servidor Graphql-server.



Como ya se invoco 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 Apolo - devTools



Main.jsx

...

const GRAPHQL\_URI = import.meta.env.VITE\_GRAPHQL\_URI;

const CONNECT\_TO\_DEV\_TOOLS = import.meta.env.VITE\_CONNECT\_TO\_DEV\_TOOLS;

...

const client = new ApolloClient({

  connectToDevTools: CONNECT\_TO\_DEV\_TOOLS,

  cache: new InMemoryCache(),

  link: new HttpLink({

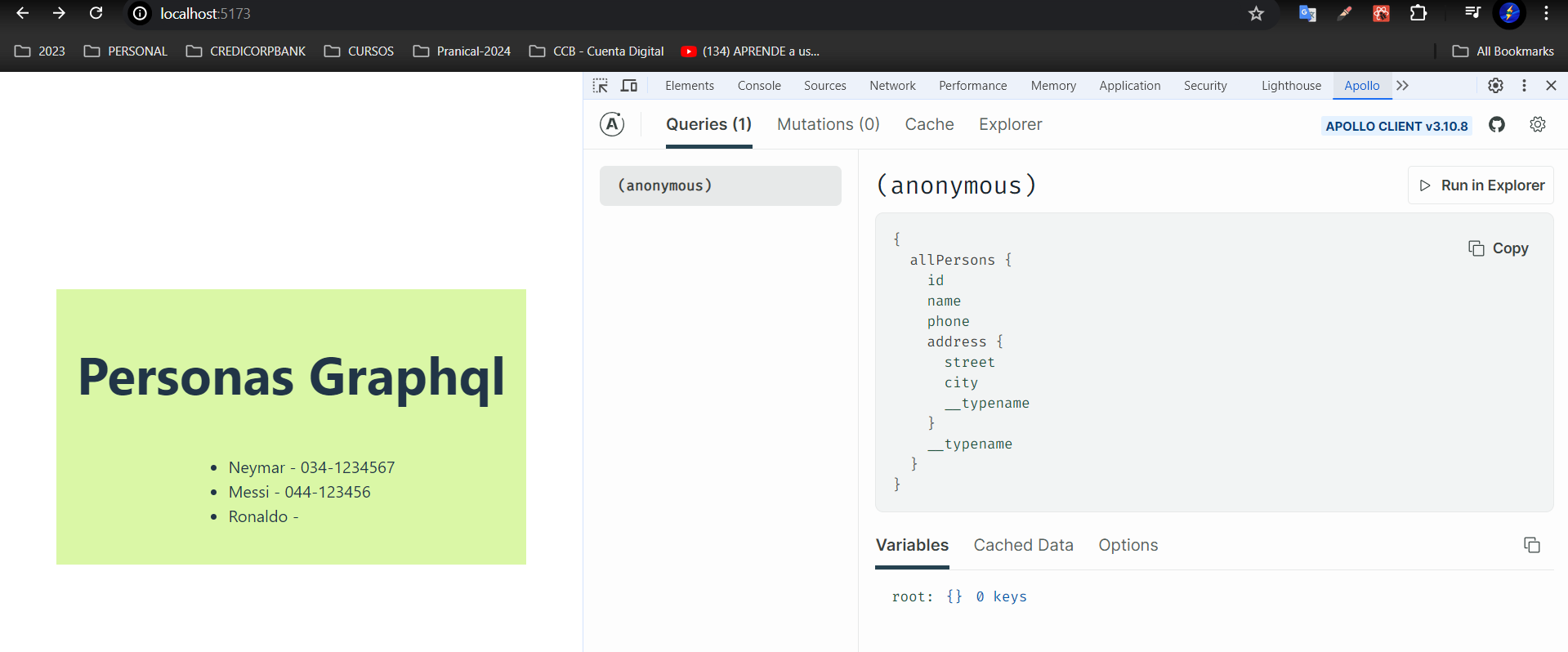
    uri: GRAPHQL\_URI,

  }),

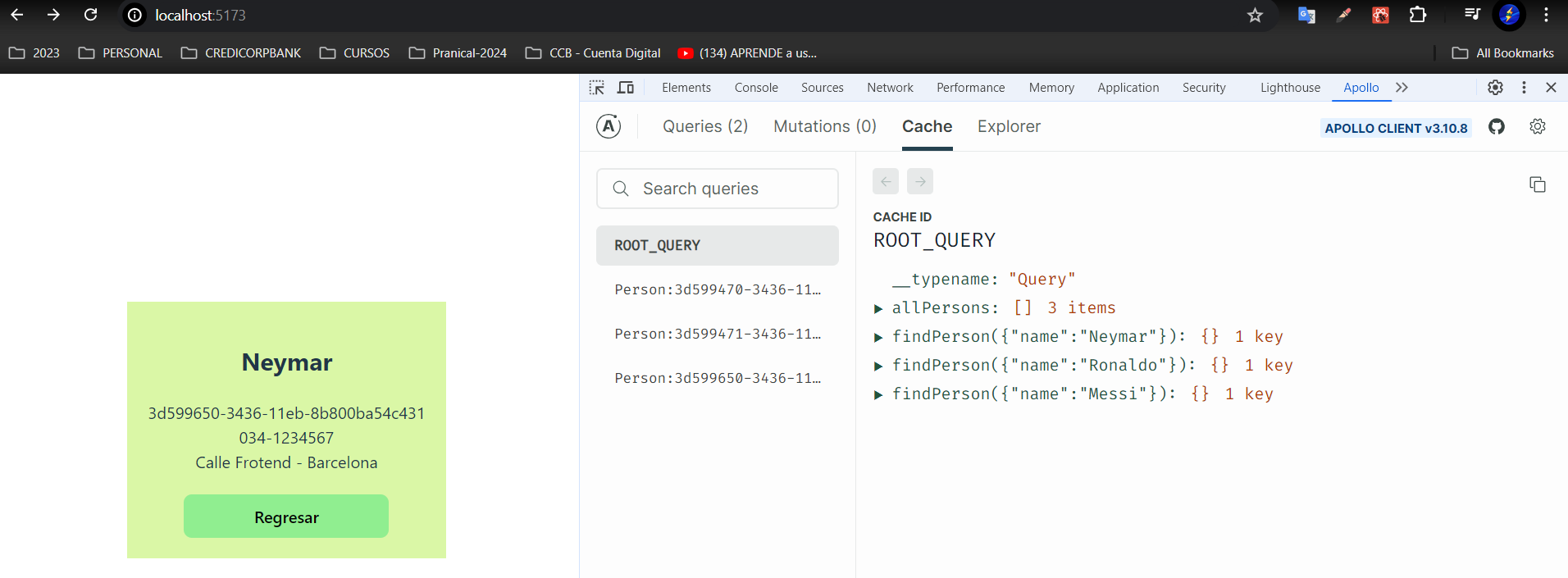
});

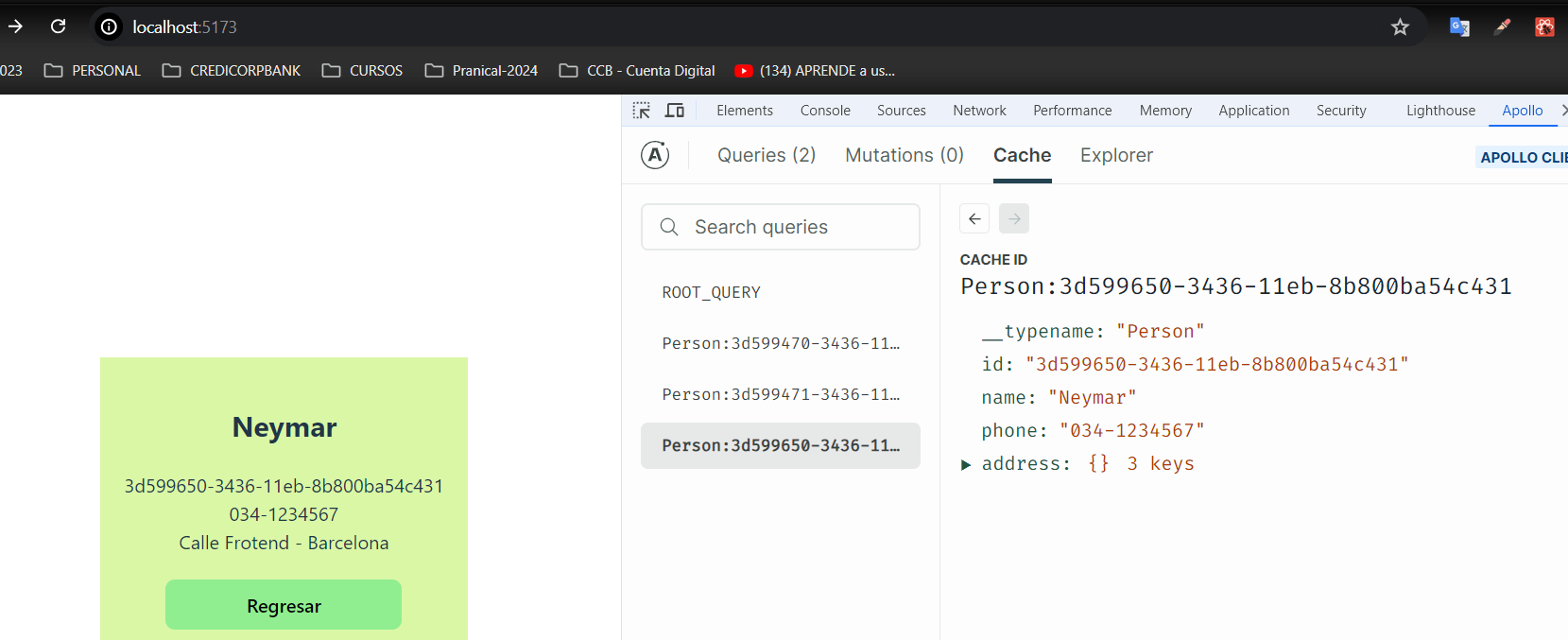
...

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 {

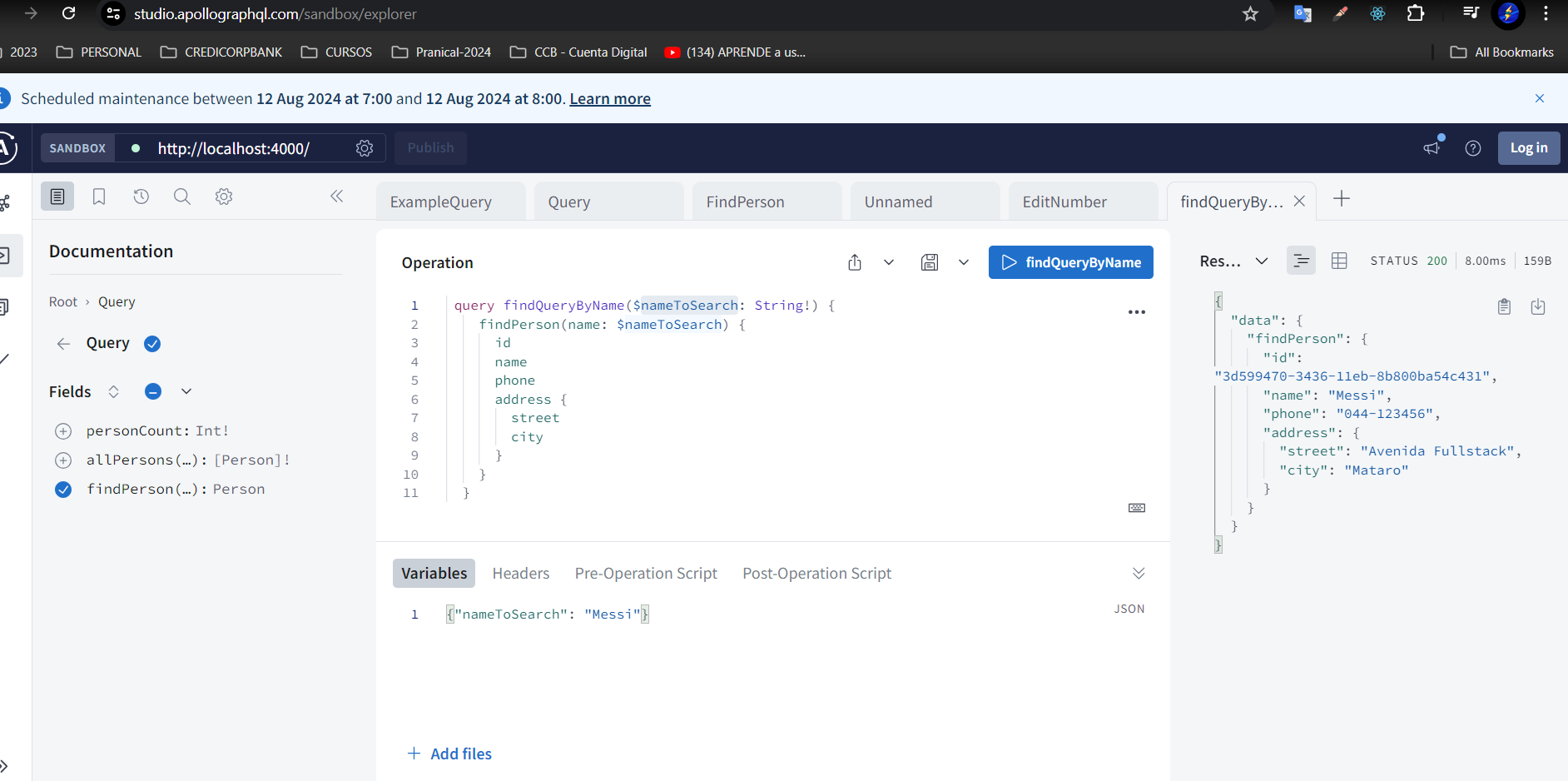
        street

        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 oder 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>

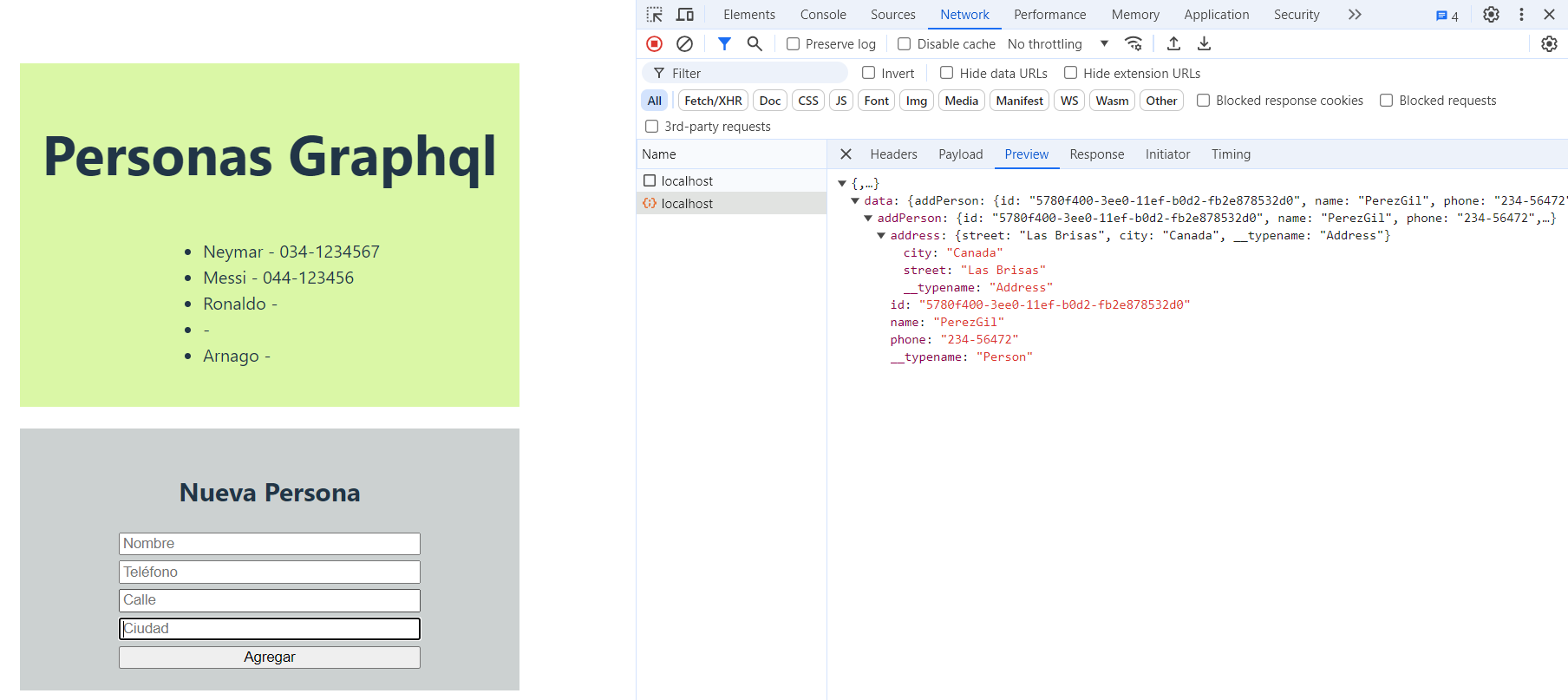
  );

};

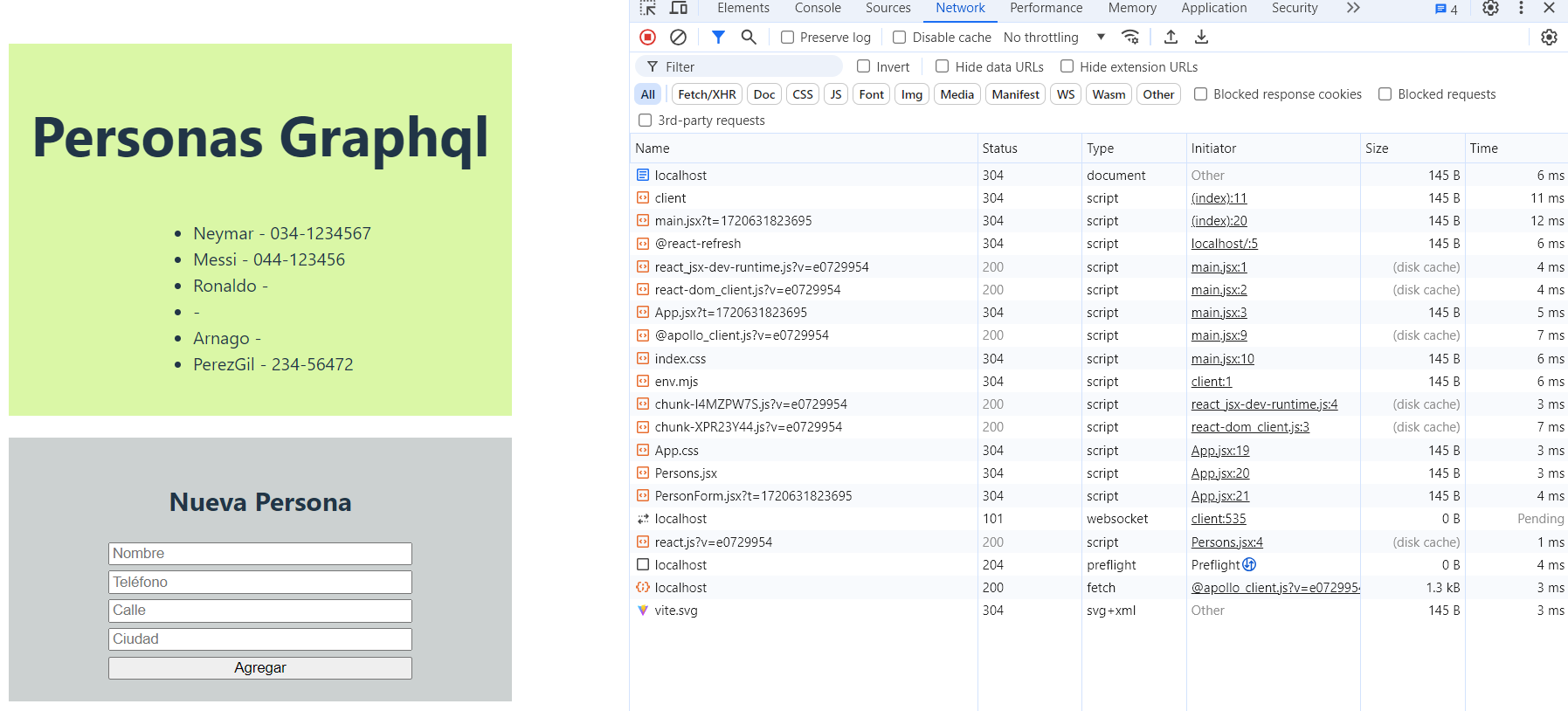
Se agrega la persona pero no se actualiza la data (si se observan nuevos datos, es porque se realizaron varias pruebas previamente)



* 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



* Si refresco la vista. Se ve el cambio



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

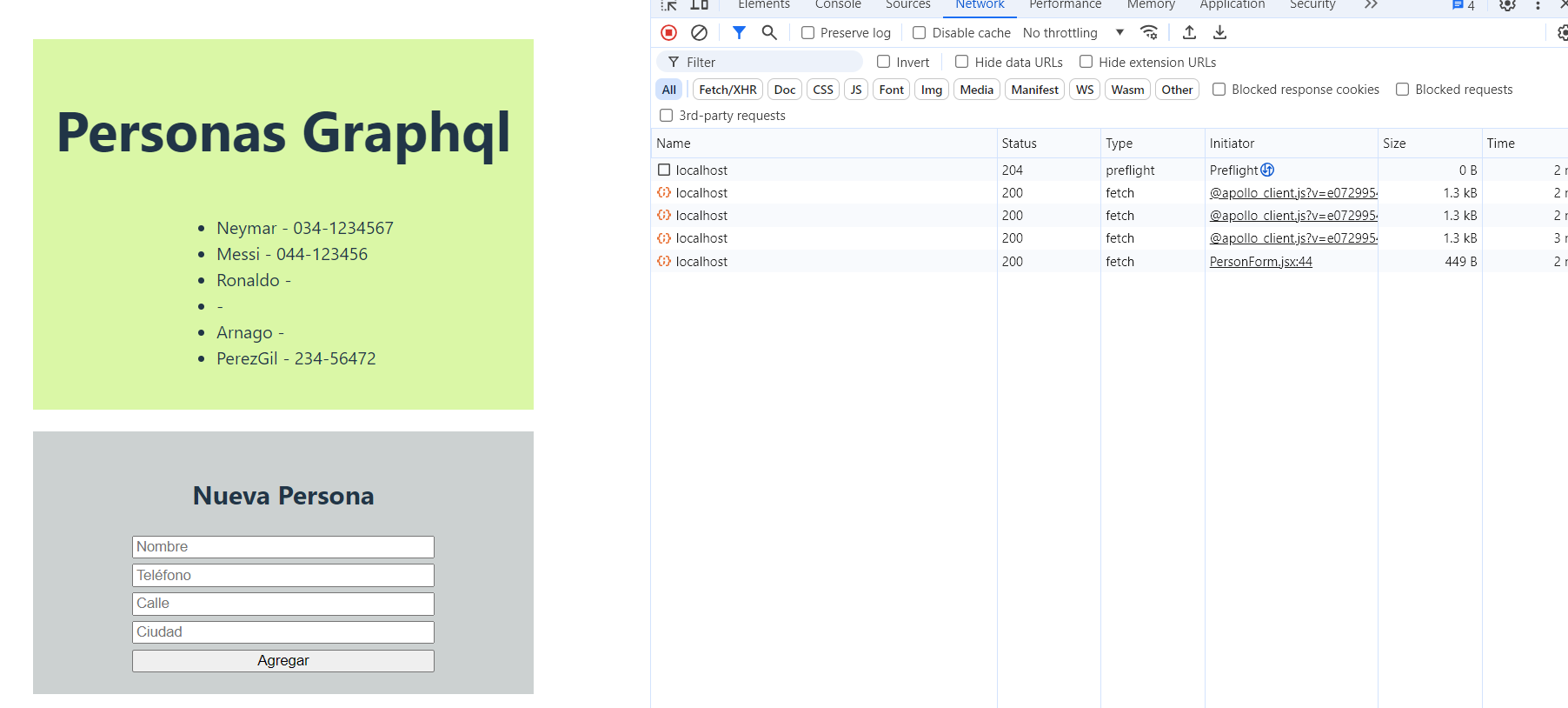
...

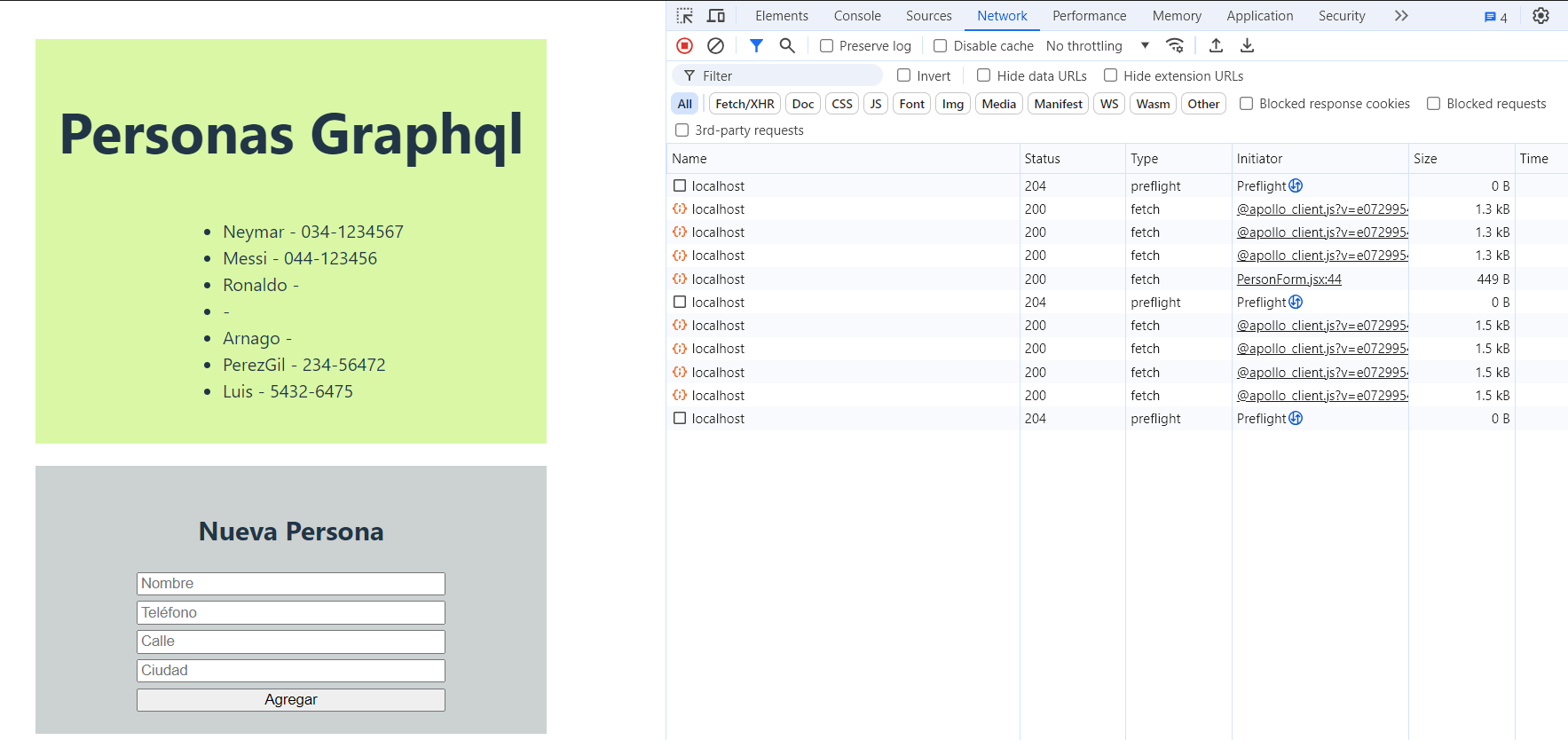
  const { loading, data, error } = useQuery(ALL\_PERSONS, {

    pollInterval: 2000,

  });

...





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



* Método 2: Agregar un parámetro a la Mutación para que refresque luego de agregar la nueva persona
  + Para no repetir el codigo 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

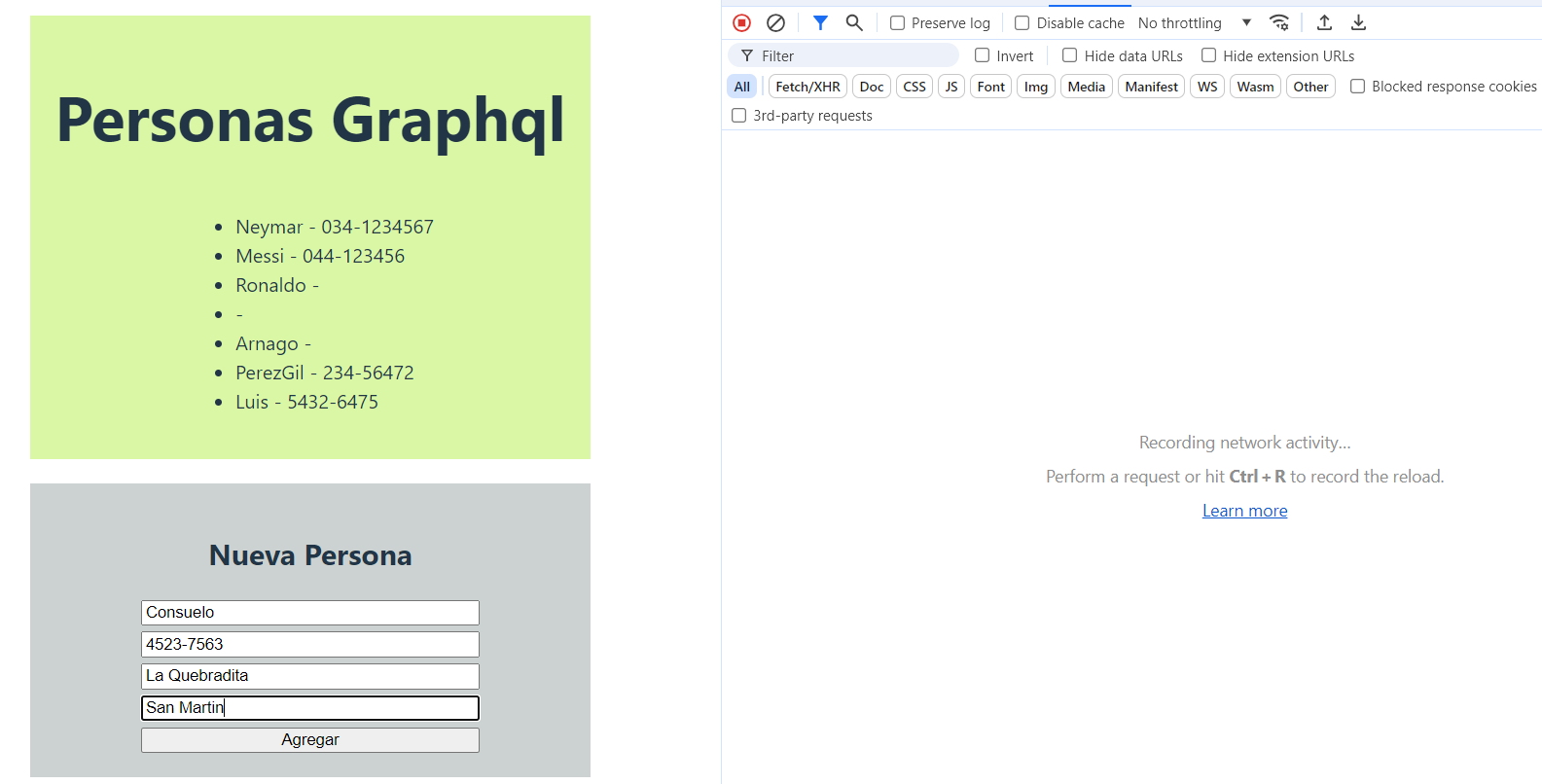
...

const [createPerson] = useMutation(CREATE\_PERSON, {

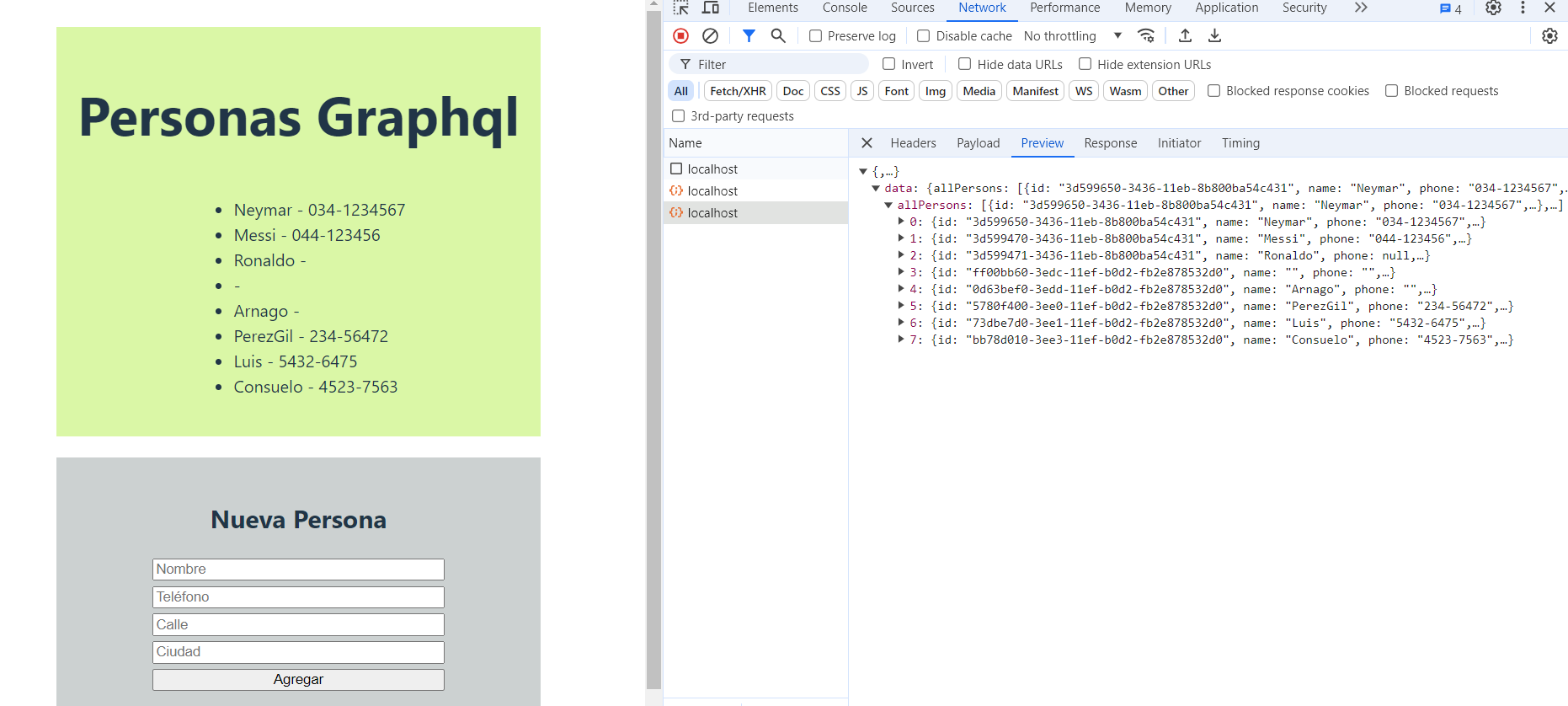
    refetchQueries: [{ query: ALL\_PERSONS }],

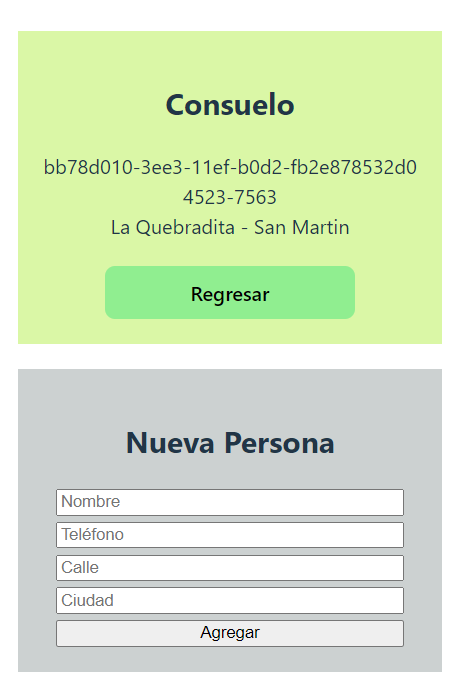
  });

...



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





Otra forma de refrescar automáticamente seria con suscripciones.