Simplificando el manejo de estado de tu app con React Query

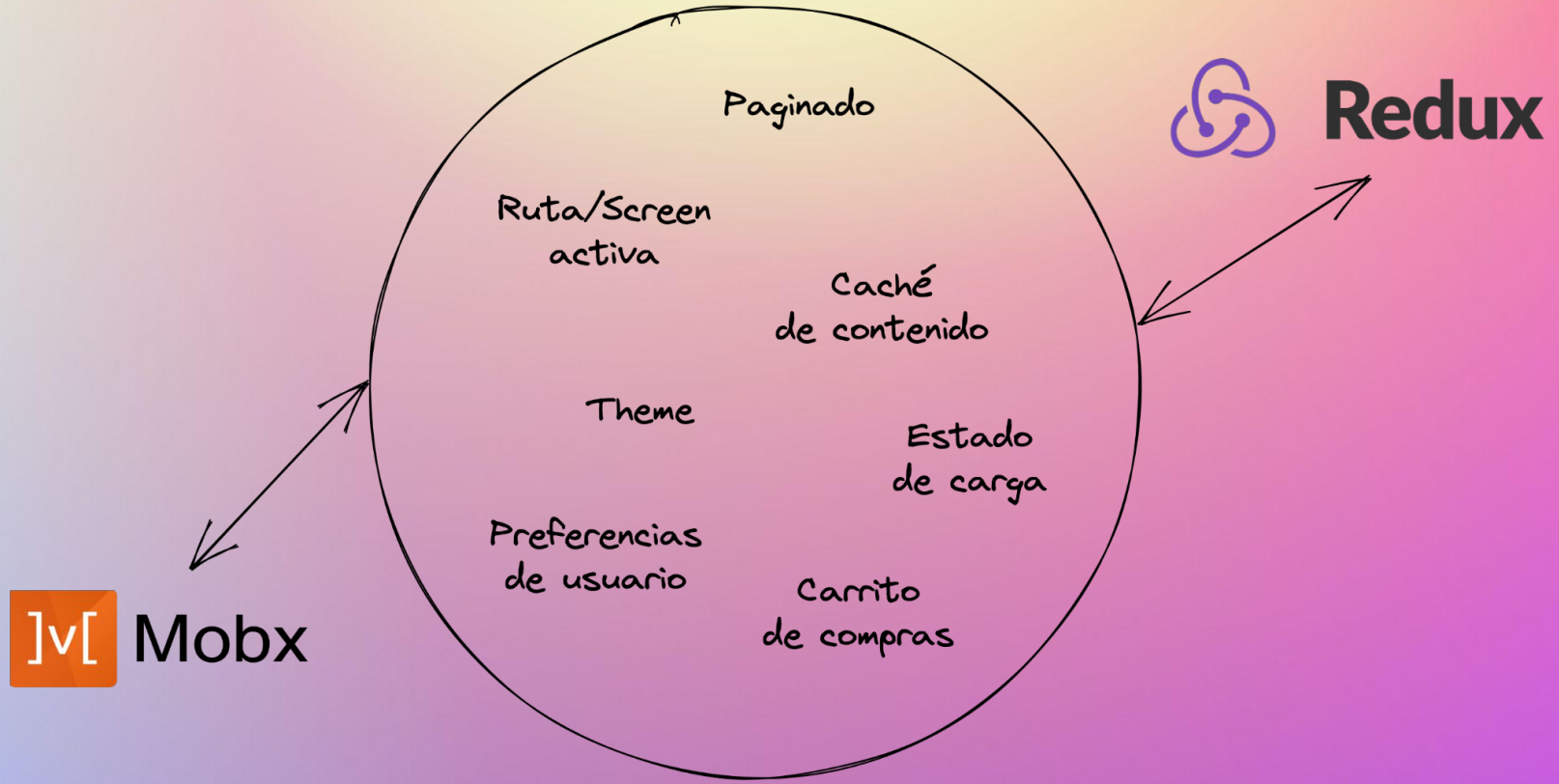
UNDERSCOPE

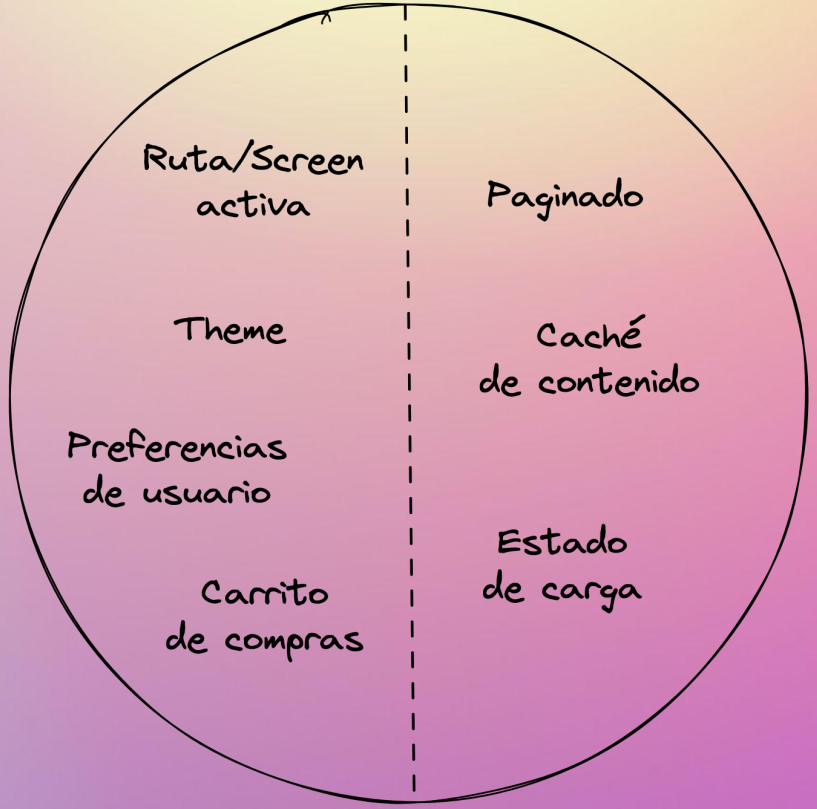
Navent | Diciembre 2022

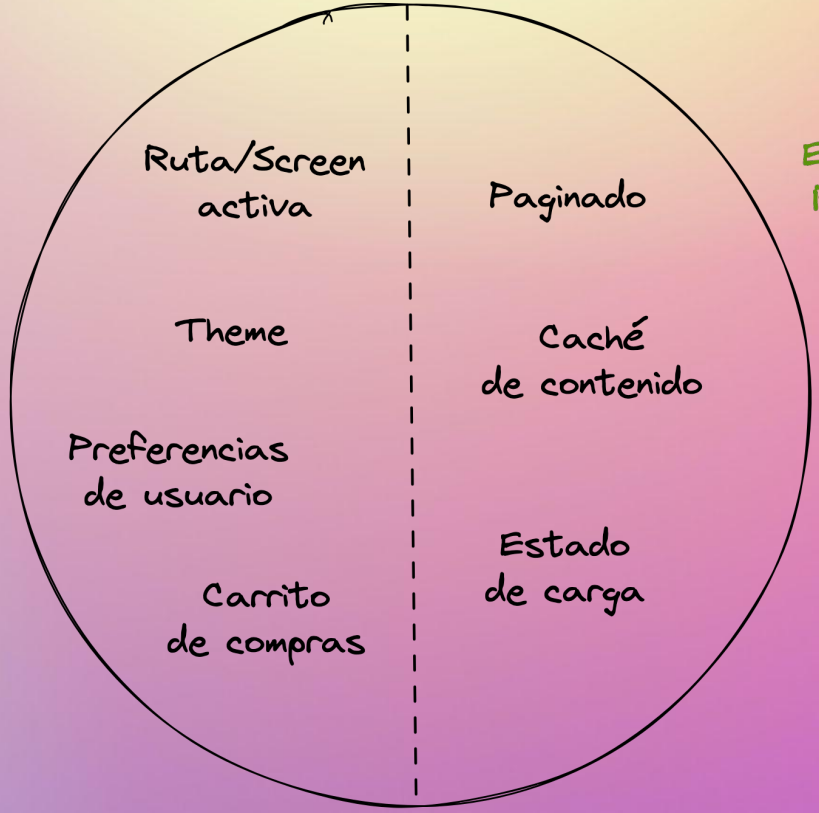
¿Qué es el **estado global** y como solíamos **manejarlo?**


Estado
Global


Paginado
Ruta/Screen activa
Caché de contenido
Theme
Estado de carga
Preferencias de usuario
Carrito de compras


Paginado
Ruta/Screen activa
Caché de contenido
Theme
Estado de carga
Preferencias de usuario
Carrito de compras
Redux
Mobx


Ruta/Screen activa
Theme
Preferencias de usuario
Carrito de compras
Paginado
Caché de contenido
Estado de carga



Estado Remoto

Estado Local

Estado Local

- Siempre está actualizado
- Pertenece al cliente

Estado Remoto

- Puede alterarse en función del tiempo
- No le pertenece al cliente
- Se consume asincrónicamente

¿Qué desafíos presenta el manejo del **estado remoto**?

- **Administrar el estado** de los pedidos al servidor
- **Guardar los datos devueltos** de dichos pedidos
- **Manejar un caché** de dichos datos y su expiración
- **Invalidar el caché** ante cambios en el cliente

¿Y si intentamos implementar esto con **Redux**?

- Solución artesanal
- Mucho boilerplate
- Código complejo
- Alto riesgo de introducir bugs

Necesitamos darle un **trato especial al estado remoto**

React Query

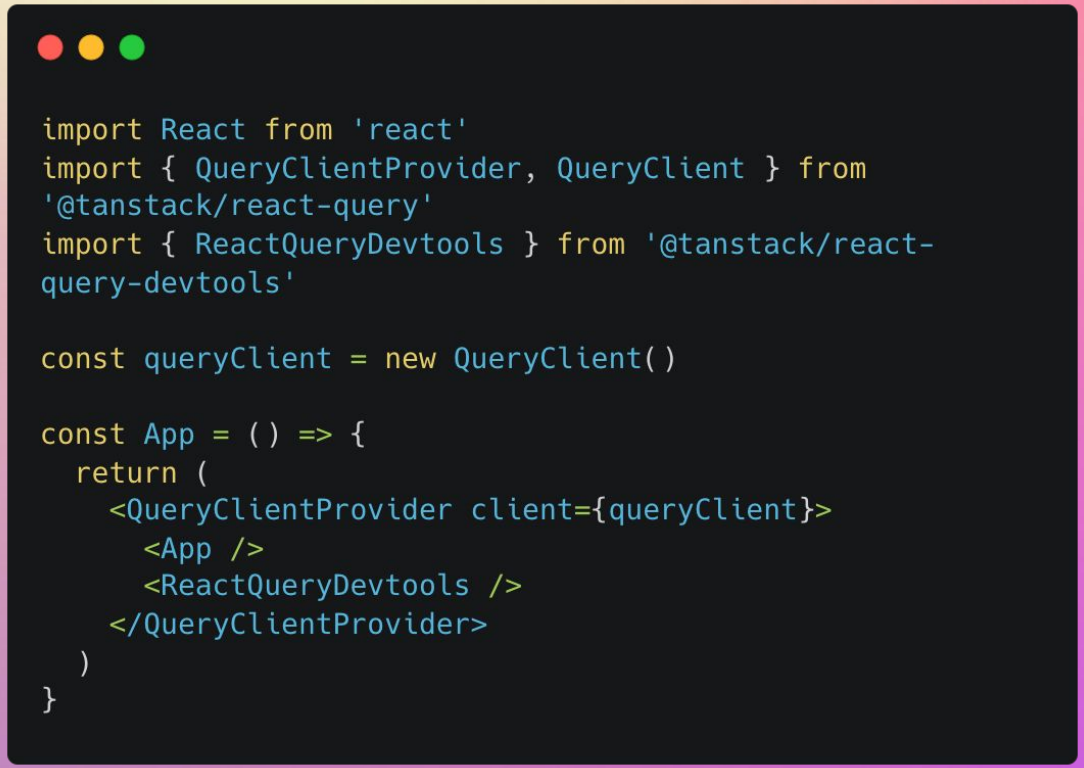
Una librería para sincronizar datos entre cliente y servidor

React Query

- Implementa **Stale While Revalidate (SWR)**
- Provee un **Caché**
- Acceso al estado del request (data, error, isLoading, etc.)
- Mejor experiencia de usuario
- Menor carga para nuestros servidores
- Código mas simple

Ejemplos de **Código**

- **Setup**
- useQuery
- useMutation



```
import React from 'react'
import { QueryClientProvider, QueryClient } from
'@tanstack/react-query'
import { ReactQueryDevtools } from '@tanstack/react-
query-devtools'

const queryClient = new QueryClient()

const App = () => {
  return (
    <QueryClientProvider client={queryClient}>
      <App />
      <ReactQueryDevtools />
    </QueryClientProvider>
  )
}
```

- Setup
- **useQuery**
- useMutation

```
import { useQuery } from '@tanstack/react-query'

import * as api from '../api'

const Intro = () => {
  const { data: todos, isLoading } = useQuery({
    queryKey: ['todos'],
    queryFn: () => api.getTodos(),
    staleTime: 60000
  })

  return (
    <>
      You currently have {isLoading ? 'loading…' :
todos.length} undone tasks
    </>
  )
}
```

- Setup
- useQuery
- **useMutation**

```
import { useMutation, useQueryClient } from '@tanstack/react-query'

import * as api from '../api'
import TodoForm from '../components/TodoForm'

const Todos = () => {
  const queryClient = useQueryClient()

  const add = useMutation({
    mutationFn: (todo) => api.addTodo(todo),

    onSettled: () => {
      queryClient.invalidateQueries({ queryKey: ['todos'] })
    },
  })

  const handleAddTodo = (todo) => {
    add.mutate(todo)
  }

  return (
    <TodoForm onAddTodo={handleAddTodo} isLoading={add.isLoading} />
  )
}

export default Todos
```

Demo

- Existen alternativas a React Query
- Eviten manejar estado local y remoto utilizando la misma herramienta
- Siempre desafíen al estado local

¡Gracias!

UNDERSCOPE

Navent | Diciembre 2022