Open Opened 5 days ago by **Rubén Montero**

Paginación... ¡más elaborada!

Resumen

- Comprenderemos que paginación con `offset` y `size` tiene sus limitaciones
- Hablaremos de paginación con `filter` y `size`
- Añadiremos un botón *Cargar más elementos* al final de la lista de preguntas en `DashboardDetail.js`
- Cuando lo pulsemos, se solicitará una nueva página de preguntas al API REST y se concatenará a las `questions` que se muestran en ese momento

Descripción

Problemas de una paginación sencilla (`offset` y `size`)

La paginación vista en la tarea anterior es una simple y adecuada para un caso donde los datos no cambian.

Pero es problemática si los datos se alteran (añaden o borran) con frecuencia.

Imagina que un cliente HTTP consume una primera página:

- GET http://servidorDeEjemplo:8081/mensajes?offset=0&size=3 *(página 1)*

Posición	Contenido	Fecha
1	¡Empezamos la semana!	Lunes (9:00)
2	Ni te cases ni te embarques	Martes (9:00)
3	Media semana, ¡qué bien!	Miércoles (9:00)

...y, entonces, alguien *añade* un nuevo comentario entre las posiciones `2` y `3` :

Posición	Contenido	Fecha
1	¡Empezamos la semana!	Lunes (9:00)
2	Ni te cases ni te embarques	Martes (9:00)
2,5	*(nuevo)* ¡Ah, no! Eso sólo es los martes 13	Martes (12.00)
3	Media semana, ¡qué bien!	Miércoles (9:00)
4	El día de Júpiter	Jueves (9:00)
5	Hoy empieza el fin de semana 😇	Viernes (9:00)
6	¡Un sábado de deporte!	Sábado (9:00)
7	¡Bonito domingo de callos!	Domingo (9:00)

Entonces, ¡la petición para la seguna página será errónea!

- GET http://servidorDeEjemplo:8081/mensajes?offset=3&size=3 *(página 2)*

Posición	Contenido	Fecha
3 ⚠	*(se mostraría 2 veces al usuario)* Media semana, ¡qué bien!	Miércoles (9:00)
4	El día de Júpiter	Jueves (9:00)
5	Hoy empieza el fin de semana 😇	Viernes (9:00)

Paginación (ahora con `filter` y `size`)

Una versión más elaborada implica el uso de:

- `filter` : Condición para filtrar los datos
- `size` : Número de datos máximo a devolver

Consiste en usar algún dato (id, fecha,...) del último elemento de la página actual para decidir dónde comienza la página siguiente.

Por ejemplo:

- GET http://servidorDeEjemplo:8081/mensajes?size=3 *(página 1)*

Posición	Contenido	Fecha
1	¡Empezamos la semana!	Lunes (9:00)
2	Ni te cases ni te embarques	Martes (9:00)
3	Media semana, ¡qué bien!	Miércoles (9:00)

- GET http://servidorDeEjemplo:8081/mensajes?filter=masRecienteQueMiercoles&size=3 *(página 2)*

Posición	Contenido	Fecha
4	El día de Júpiter	Jueves (9:00)
5	Hoy empieza el fin de semana 😇	Viernes (9:00)
6	¡Un sábado de deporte!	Sábado (9:00)

- GET http://servidorDeEjemplo:8081/mensajes?filter=masRecienteQueSabado&size=3 *(página 3)*

Posición	Contenido	Fecha
7	¡Bonito domingo de callos!	Domingo (9:00)

Gracias a `filter=masRecienteQue` esta implementación es más robusta.

¿Puedes imaginarte cómo, ante el caso problemático, esta versión de la paginación no daría problemas? 🌈

📦 La tarea

En `DashboardDetail.js` , **añade** un nuevo estado:

```
const [newPageOlderThan, setNewPageOlderThan] = useState('');
```

Luego, después *(debajo)* del `<div data-cy="list_of_questions">{...}</div>` de la lista de preguntas, **añade** este botón:

```
<button data-cy='loadMoreButton' onClick={onClickLoadMore}>Cargar más elementos</button>
```

...y **crea** el `onClickLoadMore` , **así**:

```
const onClickLoadMore = (event) => {
  const lastQuestion = questions[questions.length - 1];
  setNewPageOlderThan(lastQuestion.created_at);
}
```

Hemos conseguido que, en el estado `newPageOlderThan` se guarde la fecha de la última *pregunta*.

Análogamente al ejemplo anterior, cuando hagamos *click* (la primera vez) en `Cargar más elementos` , `newPageOlderThan` *pasaría a valer* `Miércoles` .[1]

Reaccionando y pidiendo una nueva página

Ahora, **modifica** tu `useEffect` en `DashboardDetail.js` :

```
  useEffect(() => {
-   axios.get('http://raspi:8082/api/v2/dashboards/' + params.dashboardId + '?page_size=5').then(response => {
+   axios.get('http://raspi:8082/api/v2/dashboards/' + params.dashboardId + '?page_size=5&older_than='+newPageOlderThan).then(respor
```

```
      setDashboardTitle(response.data.title);
      setDashboardDescription(response.data.description);
      setQuestions(response.data.questions);
    })
- }, [])
+ }, [newPageOlderThan])
```

Con ello, *reusamos* el efecto para que la petición ahora mande `page_size` y `older_than` (análogos a `size` y `filter`).

Todas las piezas están perfectamente encajadas para que, al clicar en `Cargar más elementos`, el estado `newPageOlderThan` adquiera el valor de la fecha de la última *pregunta* mostrada, y automáticamente se reenvíe la petición adecuada para obtener las `5` siguientes.

Una última mejora... ¡Concatenar[2] (en vez de sustituir) las *preguntas* a las que ya tenemos! Así, conseguimos un efecto de *scroll* infinito.

Modifica el `useEffect` nuevamente así:

```
  useEffect(() => {
    axios.get('http://raspi:8082/api/v2/dashboards/' + params.dashboardId + '?page_size=5&older_than='+newPageOlderThan).then(respor
      setDashboardTitle(response.data.title);
      setDashboardDescription(response.data.description);
-     setQuestions(response.data.questions);
+     const newQuestions = questions.concat(response.data.questions);
+     setQuestions(newQuestions);
    })
  }, [newPageOlderThan])
```

¡Listo!

Verifica cómo se comporta http://localhost:3000/dashboards/2.

Repasa lo que has implementado para asegurarte de que lo entiendes.

🏆 Por último

Sube tus cambios al repositorio en un nuevo *commit*.

---

1. En el ejemplo se filtar por `masRecienteQue`, y en nuestro caso por `older_than`, ya que van a la inversa (de más reciente a más antiguo, en vez de al revés). ⮌
2. ¿Recuerdas que hay una regla sobre modificar un estado que es un *array*? Debemos evitar *mutarlo* directamente. ⮌

Rubén Montero @ruben.montero changed milestone to %Sprint 5 5 days ago