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

Búsquedas en APIs REST

Resumen

- Añadiremos un estado `searchText` a `QuestionDetail.js`
- Añadiremos un formulario para permitir introducir un texto de búsqueda
- Cuando se haga `submit` del formulario, mandaremos una petición al API REST para refrescar la lista de respuestas que encajen con el resultado

Descripción

Nuestro API REST soporta *buscar* mediante el *query param* `search` en los mismos *endpoints* que soportan paginación:

- `/api/v2/dashboards/<dashboardId>`
- `/api/v2/dashboards/<dashboardId>/questions/<questionId>`

Vamos, ahora, a usar el segundo *endpoint* para implementar *buscar respuestas*.

Por ejemplo, ¿qué datos devuelve esta petición?

- GET http://raspi:8082/api/v2/dashboards/1/questions/1

En contraposición, ¿qué datos devuelve esta otra? (al mismo *endpoint*)

- GET http://raspi:8082/api/v2/dashboards/1/questions/1?search=ayudarte

Nuestro objetivo...

...es permitir a los usuarios de nuestra web aprovechar esta funcionalidad, y que puedan filtrar las respuestas de `QuestionDetail.js` en base a un texto de búsqueda.

La tarea

Añade en `QuestionDetail.js` un *estado* que represente el texto *actual* en el campo de búsqueda:

```
const [searchText, setSearchText] = useState('');
```

Luego, **añade** un `<form>` con las siguientes características:

- Estará ubicado después del `<h3>`
- La etiqueta `<form>` tendrá `data-cy='searchForm'` y `onSubmit={onSubmit}` *(Escribiremos la función `onSubmit` a continuación)*
- Contendrá una primera etiqueta `<input>` con:
  - `data-cy='searchInput'`
  - `type='text'`
  - `placeholder='solucionado'`
  - `value={searchText}`
  - `onChange={onChange}` *(Escribiremos la función `onChange` a continuación)*
- Contendrá una segunda etiqueta `<input>` con:
  - `data-cy='inputSubmit'`
  - `type='submit'`
  - `value='Realizar búsqueda'`

Luego, **añade** la función `onChange`, que se encarga de reflejar en el estado `searchText` lo que el usuario escribe:

```
const onChange = (e) => {
  setSearchText(e.target.value);
}
```

Para terminar, **añade** la función `onSubmit` [1]:

```
const onSubmit = (e) => {
  e.preventDefault();
```

```
    // Aquí lanzamos la búsqueda
    axios.get('http://raspi:8082/api/v2/dashboards/' + params.dashboardId + '/questions/' + params.questionId + '?search=' + searchT
      setAnswers(response.data.answers)
    })
  }
```

Como puedes observar, construye la URL apropiada para la petición HTTP (incluyendo el parámetro `?search=` y lanza la petición para que el API REST devuelva las *respuestas* que coinciden con la búsqueda[2].

¡**Verifica** en http://localhost:3000/dashboards/1/questions/1 que funciona correctamente!

🏆 Por último

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

---

1. En contraposición al ejemplo de paginación, ahora *no* reutilizamos el `useEffect`. Así, la petición sólo se envía cuando hacemos `onSubmit` (y no con cada carácter que teclea el usuario). ¿Es mejor? ¿Es peor? ¿Qué opinas? ⮌
2. Esto es una búsqueda *lado servidor*. Si el conjunto de datos no fuera excesivo, podríamos plantearnos una búsqueda (filtrado) *lado cliente*. Para ello, descargaríamos todos los datos y los filtraríamos en local usando `.filter` ⮌

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