Implementació d'un Client Web Responsive amb un Front-End JavaScript

Alex Catalán Muñoz, Albert Gomáriz Sancha, Alexandre Gràcia Calvo i Marc Soler Colomer

Aplicacions i serveis web
17/12/20

Tardor 2020-2021

Índex de continguts

Informació de caire general **3**
Desplegament: 3
Seguiment tasques: 3

Canvis realitzats a la API **3**

Diagrames de seqüència **5**
Clicar enllaç per visualitzar la informació de l'usuari loguejat 5
Des del perfil d'usuari, canviar fer update de l'usuari 6

1. Informació de caire general

Desplegament

https://hackernews-asw-react.vercel.app/

Seguiment tasques

Observar apartat closed del repositori:
https://github.com/alexcatmu/hackernews-asw-react/issues

2. Canvis realitzats a la API

El primer canvi que vam realitzar a la API va ser canviar els endpoints per eliminar els likes de les contribucions, comentaris i rèpliques, tal com se'ns va indicar que hauria de ser en l'avaluació del segon lliurament.

El nostre endpoint original era DELETE /contributionlikes/{id_contributionlikes}, fent que necessitèssim retornar en alguna crida aquest id, per tal tenir-lo disponible per esborrar el like. L'hem canviat per DELETE /contributions/{id_contribution}/like, per tal de tenir el mateix endpoint que el seu antònim, el like, només canviant el mètode HTTP. Aquest canvi ens proporciona un millor disseny, que es tradueix amb més comoditat pel desenvolupador i un millor rendiment del front-end.

Després també vam afegir les répliques de les répliques. Al model de les rèpliques vam afegir-li un camp reply_id, que seria null en cas de ser la rèplica pare o l'identificador de la rèplica a la qual replicava.

La resta de canvis que hem realitzat segueixen aquesta filosofia, intentar alliberar el front-end de la responsabilitat de tractament de dades, i que la seva funcionalitat principal sigui la visualització d'aquestes, proporcionades pel back-end.

Els GET d'un llistat de contribucions/comentaris (New, threads, ask, upvoted submissions/comments, submitted, home, etc..)

- ➔ Les contribucions, comentaris i rèpliques, també retornen *l'username* de l'usuari que l'ha realitzat, així doncs el front-end no ha de fer una crida per cada element per conèixer el nom de l'autor.
- ➔ Les contribucions, comentaris i rèpliques, també retornen un array amb els identificadors d'usuari que l'han votada positivament, per tal de que el front-end sàpiga en tot moment qui ha fet like d'aquests elements i pugui controlar que la lògica funcioni correctament sense necessitat de crides addicionals.

- ➔ Els elements corresponents també retornen el nombre de comentaris que tenen per tal poder-los mostrar a les pàgines corresponents. D'aquesta manera no cal que passem tots els comentaris associats, ja que no s'han de mostrar en aquestes vistes.
- ➔ En el casos de les llistes de comentaris, també li passem el títol de la contribució a la qual pertany.

En els GET d'un sol element (/contributions/{user_id}, users/{users_id}, etc..)

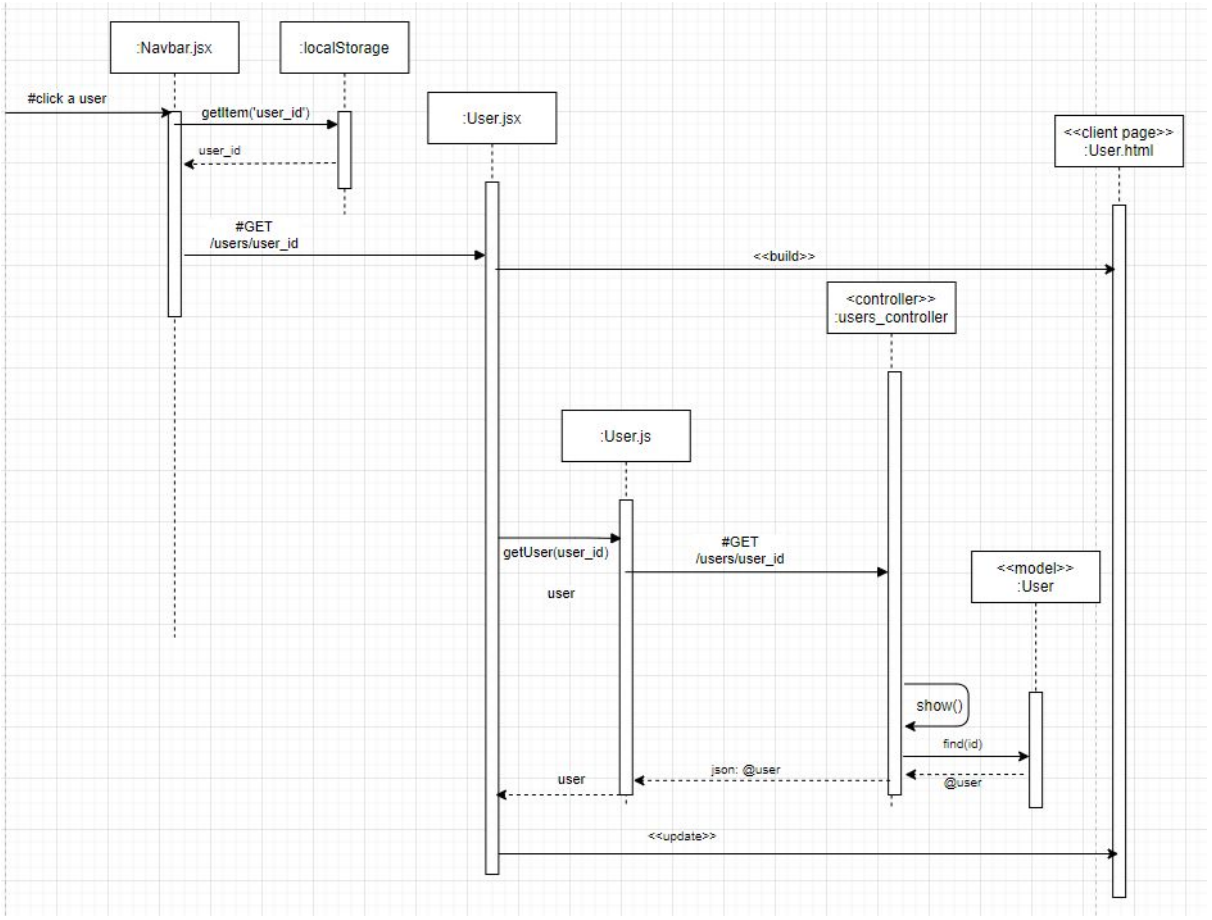
- ➔ Pel que fa al GET d'una sola contribució, retornem tot l'arbre de comentaris i rèpliques (recursives) associades a ella mateixa. D'aquesta manera el front-end s'estalvia moltes crides HTTP i pot dedicar els seus recursos a la recursivitat necessària durant la visualització.
- ➔ Hem afegit una funcionalitat a l'endpoint GET /users (que no s'utilitzava), que et retorni l'usuari corresponent a l'API KEY que li passem en el header. D'aquesta manera, hem dissenyat un petit login utilitzant aquesta crida, i el localStorage del navegador per guardar el token i l'id d'usuari que s'utilitzaran en altres funcionalitats de la web.

En el POST /contributions:

- ➔ Un petit canvi en el cos i l'status del cas quan intentes crear una contribució amb una url ja existent, per saber que s'ha de fer la redirecció a l'element concret. Retorna l'id de la contribució existent amb l'status 303 See other, acompanyada de la redirecció que hem comentat.

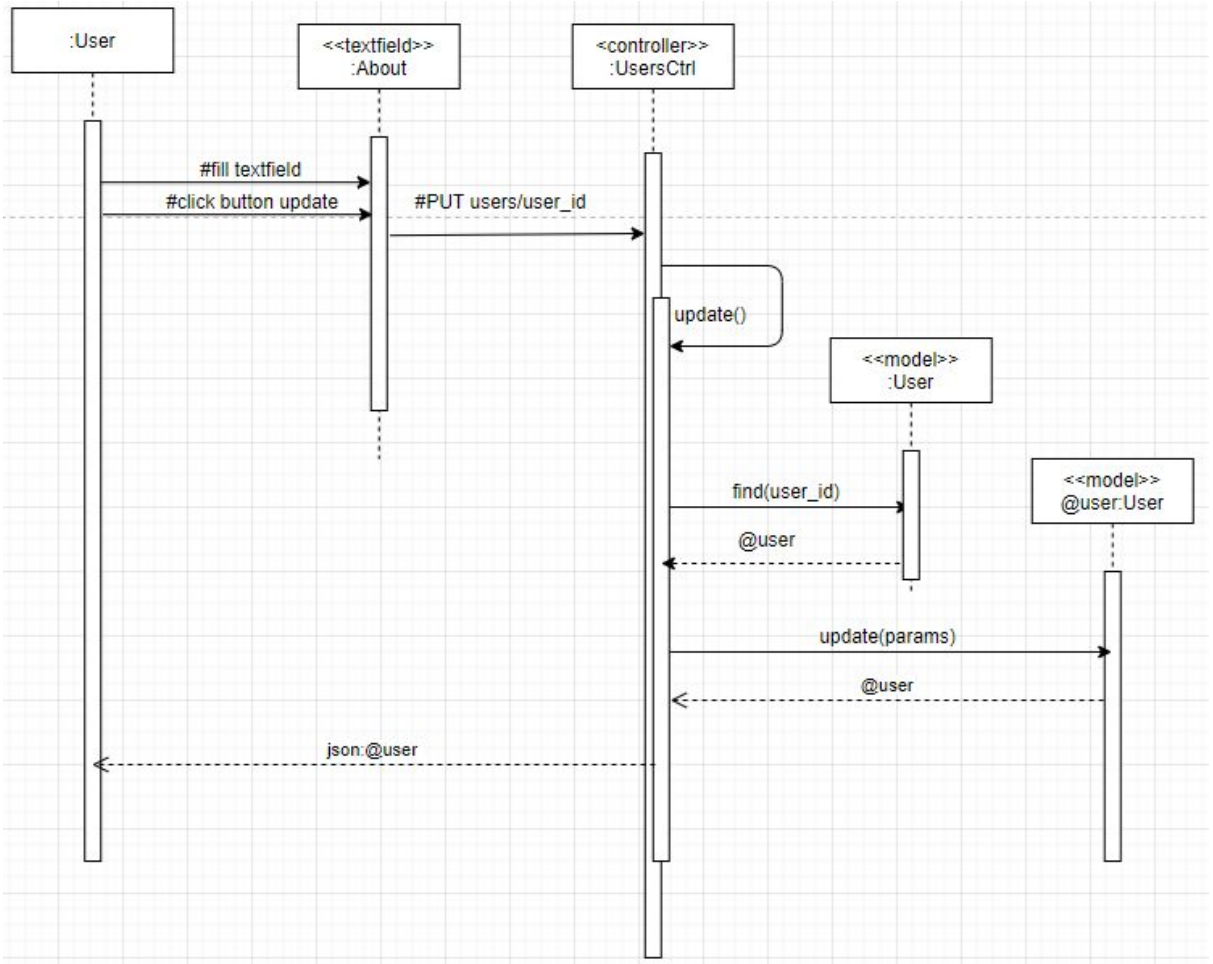
3. Diagrames de seqüència

Clicar enllaç per visualitzar la informació de l'usuari loguejat



La primera crida HTTP, és un GET a /users/user_id que ens porta al .jsx del component User. El {user_id} és agafat directament del localStorage. En cas que aquest sigui null, perquè l'usuari no s'ha loguejat, ens redirigirà al component de Login (no ho hem representat al diagrama ja que hem tractat el cas en que l'usuari ja està loguejat) . Altrament, ara farem la mateixa crida GET /users/user_id però cap al back-end, on recullirà l'id del path per buscar l'@user corresponent, i ens el tornarà en format JSON. El component el renderitzarà amb la seva funció render, compliant el jsx i l'usuari podrà visualitzar l'html corresponent.

Des del perfil d'usuari, canviar fer update de l'usuari



La crida HTTP, és un PUT a users/{user_id} amb els headers [Accept] = application/json i [token] = localStorage.getItem('token'), que és el token d'identificació de l'usuari que guardem al localStorage. Per últim també té el header [Content-Type] = application/json, ja que en el body li hem de passar un JSON amb l'about el qual haurem omplert en el "TextField". La resposta, serà un json amb l'@user actualitzat.