ADI_2324

Práctica evaluable 4: Frameworks Javascript

El objetivo de esta práctica es desarrollar una web que sirva de cliente al API de la práctica 1. En la implementación debéis usar un *framework JS*, en concreto **Vue**

Organización del código

- Usad una estructura de proyecto similar a la práctica anterior, con una carpeta `client` y una `server`
- Necesitaréis modificar el servidor **para que use CORS**, como se hizo en el servidor de la lista de la compra (mirar la **Sección titulada "CORS"** de este documento)
- Para crear el proyecto cliente usad la herramienta `create-vue`. Esto os permitirá emplear Single File Components, en los que cada componente se escribe en un archivo separado. Esta forma de estructurar el código suele tener más soporte en los IDEs que la que vimos la semana pasada en la minipráctica, y también es más apropiada para proyectos de tamaño mediano/grande. También podríais usar Parcel u otro *bundler*, pero `create-vue` es más recomendable por ser la herramienta oficial de desarrollo.

Requisitos mínimos

La implementación correcta de estos requisitos es necesaria para poder aprobar la práctica. Con estos requisitos podéis optar hasta un 6 si además de la implementación comentáis adecuadamente el código fuente, indicando para cada componente: qué hace en líneas generales, qué eventos genera o procesa, qué representa cada variable del estado (`data` en Vue, si lo tiene) , cada prop, y cada método (descripción de lo que hace, parámetros, efecto en el estado si lo tiene).

Debéis desarrollar una web en la que como mínimo se pueda hacer:

- Login/logout
- Listado de items: pueden ser todos o puede haber un cuadro de búsqueda asociado. Si permitís búsqueda e implementáis paginación en el cliente tendréis **0.5 puntos más**
- Eliminación de items (similar al botón "x" de la lista de la compra)
- Creación de items
- Ver detalles: al clicar en un item o en un botón asociado a cada item, se muestra más información sobre el mismo (sin cambiar de página)
- Edición de items: cada elemento del listado debería tener un botón o enlace "editar" para cambiar su contenido. Al pulsarlo aparecerá un formulario para escribir los nuevos datos. Dónde y cómo aparezca (y desaparezca al acabar la edición) queda a vuestra elección. Los formularios de edición también deben validarse.

> Pongo "items" como término genérico porque cada uno estáis haciendo una aplicación distinta, en vuestro caso concreto serán "productos" o "usuarios" o "películas" o lo que proceda.

Los formularios de login/creación de items deberían estar validados (por ejemplo login y password no pueden estar vacíos, si tenéis algún campo numérico hay que chequear que lo sea, etc), sin perjuicio de que los datos también se chequeen en el servidor. Para implementar la validación podéis usar el método que queráis. Por ejemplo se pueden usar atributos de HTML y JS estándar o emplear una librería de terceros, como por ejemplo VeeValidate para Vue.

> Ojo, la validación con HTML/JS estándar se dispara en algunos casos con el evento "submit" de un formulario. Cuando este evento llega al navegador, el comportamiento por defecto es cargar una página nueva, o recargar la actual, esto hará que el estado de los componentes se pierda, y no es una buena idea :). Si usáis este método tendréis que aseguraros de que el evento submit se anula en vuestro *listener* con un `preventDefault`.

Además, **se debe usar Pinia o Vuex para gestionar el estado de vuestra aplicación de manera centralizada**. En el *store* debería estar como mínimo:

- La información sobre el usuario autentificado y los métodos de negocio de login/logout/registrar usuario (si tenéis este último)
- Los datos de los listados que mostréis en la web
- Los métodos de comunicación con el servidor (lo que en los ejemplos de la lista de la compra siempre metemos en la clase `ClienteAPI`)

Cómo implementar el login/logout

Aunque en un API REST "puro" no existe formalmente la idea de "hacer login/hacer *logout*", para el usuario de la web es útil que existan las operaciones de *login* y el *logout*. De lo contrario tendría que estar introduciendo continuamente *login* y *password*. Debes implementar las siguientes funcionalidades:

- En algún sitio debes mostrar un formulario para hacer *login*.
- Una vez comprobado que *login* y *password* son correctos, y obtenido el *token* se guardará este último en el *local storage*
- Cada vez que se haga una petición al servidor con Javascript para una operación restringida enviaréis el *token* en la cabecera `Authorization`
- La operación de *logout* simplemente borrará el *token* del *local storage*.
- Una vez hecho el login con éxito se debería mostrar en algún sitio de la página el login del usuario o algún mensaje que le indique que ya está logueado y un botón/enlace para hacer *logout*. Cuando se hace el logout con éxito se debería mostrar el formulario de login.

Requisitos adicionales

Además de los 6 puntos de los requisitos mínimos, podéis implementar los siguientes requisitos adicionales:

- hasta *0.5 puntos* búsqueda y paginación de datos, evidentemente para esto tendría que estar implementado también en el servidor. Si no lo hicisteis en su día y queréis hacerlo ahora, podéis añadirlo (aunque no os puedo dar puntos extra por la parte del servidor).
- Hasta *0.5 puntos* introducir transiciones/animaciones cuando aparezcan/desaparezcan elementos de la página. Añadir además el típico efecto de que el botón/formulario "tiembla" cuando introducimos un login/password incorrectos. Vue incorpora ya integrada la funcionalidad para hacer estas transiciones
- hasta *1 punto* si usáis algún *framework* de componentes visuales. Para Vue tenéis por ejemplo vuetify o bootstrap-vue
- hasta *0.5 puntos*: usar un *router* para gestionar la navegación por la *app*. En el caso de Vue se usaría Vue Router
- hasta *0.5 puntos*: implementar el listado de otro recurso incluyendo también "ver detalles", "eliminar" y "editar".
- Hasta *1 punto*: Implementar SSR (Server Side Rendering) en los componentes usando el *framework* Nuxt). Al menos uno de los componentes "principales" del sitio debe usar SSR (con "principal" me refiero a que sea la lista de items, o algo similar, no un contador o un *widget* que no tenga que ver con el propósito principal de la web). El concepto de SSR se explicará en clase de teoría del 21 de noviembre.

Cualquier otra funcionalidad que queráis añadir consultadla antes conmigo para ver cuánto se podría valorar en el baremo.

Entrega

El plazo de entrega concluye el **viernes 22 de diciembre de 2023** a las 23:59 horas. La entrega se realizará como es habitual a través de Moodle. Entregad una carpeta comprimida con el proyecto de esta práctica y el del servidor (modificado para usar CORS) en dos subcarpetas, ya que ambos son necesarios para ejecutar la aplicación.

Entregad también un LEEME.txt donde expliquéis brevemente las partes optativas que habéis hecho y cualquier detalle que consideréis necesario.