

|  |
| --- |
| **Informe Proyecto Integrador To-Do**  Front End II **Karin Stricker**30/07/2021 Durante el trayecto de la cursada, desarrollamos una página que permite a los usuarios gestionar sus tareas. Esta página tiene las siguientes funcionalidades: Ingresar a nuestra cuenta (login)Ver tareas pendientesVer tareas terminadasMarcar una tarea como terminadaCrear nuevas tareasVer la fecha de creación de una tarea En este informe se detalla brevemente cómo fue realizado cada uno de estos ítems, y cómo funciona su código. Al finalizar se incluye una breve apreciación sobre el proyecto. |
| Página 1 |

|  |  |  |
| --- | --- | --- |
| **Proyecto Integrador To-Do**  Front End II | |  |
| Ver tareas pendientes21Ingresar a la cuenta (Login)Ver tareas terminadasMarcar una tarea como terminada3Crear nuevas tareasVer tareas terminadas54Ver la fecha de creación de una tarea67Conclusiones8Comentarios finales | Ingresar a la Cuenta (Login)Datos necesarios En primer lugar, seleccionamos el elemento html “formulario” para luego poder ejecutar una función al suceder el evento “Submit” del mismo.  Dentro de la función del evento Submit, una vez detenido el evento por defecto, guardamos en variables los valores que ingresa el usuario en los campos email y contraseña, que son los campos que figuran en la página de login. Petición Con esto preparado, hacemos entonces uso del método fetch, que permite realizar peticiones asíncronas. A fetch le pasamos dos parámetros, el endpoint al cual queremos hacerle la petición del login (en este caso la url que termina en /login), y como segundo parámetro un objeto con la información que solicita la API. En este caso este objeto va a tener una propiedad “method” POST ya que enviamos una petición con datos del usuario; va a tener otra propiedad “headers” en la cual vamos a indicar que el contenido que enviaremos estará en formato json; y por último, la propiedad “body”, en la cual enviaremos los datos (en formato JSON) que nos solicita la API para hacer la petición, en este caso estaremos enviando el email y la contraseña que ingresó el usuario. El body permite que el backend corrobore que esta información pertenezca a un usuario existente. Trabajando con los datos Fetch devuelve una promesa que si se resuelve entonces nos permite trabajar con la información que nos devuelve el backend. Para poder hacer uso de esta data, utilizamos el método “.then”. | |
|  | | |
| Página 2 | | |

Como lo que se devuelve es un objeto response, hacemos un primer “.then” para transformar los datos del body en un objeto de JavaScript con el método “.json()”. Como el método .json() nos devuelve otra promesa, con un segundo “.then” ya podemos utilizar la información para lo que nos interesa. En este caso queremos guardar la propiedad “jwt” que nos devuelve la API en localStorage, y le ponemos como nombre de clave “token”. Este token es un código que permite a la aplicación saber que ese usuario está logueado, y que no necesite loguearse cada vez que realice una acción, por eso es que lo dejamos guardado en localStorage. Inmediatamente ya podemos redirigir al usuario a la página del listado de tareas.

## Errores

Podemos agregar el método “.catch()” para el caso que la promesa no se resuelva, o que ocurra un error, para informar al usuario del error que ocurrió.

# 2. Ver tareas pendientes

## Petición

Para consultar las tareas, creamos una función para obtener las tareas; nuevamente hacemos uso del método fetch, al cual le pasamos como primer parámetro el endpoint, y como segundo el objeto con los datos; en este caso no hace falta aclarar que el método es “GET” (o sea para obtener información de la API) ya que es el método por defecto que trae fetch. Es importante en “headers” enviar la autenticación que requiere la API, en este caso es el token que se dejó guardado en localStorage. Como la petición es Get, no hace falta enviar un body con datos.

## Trabajando con los datos

Cuando la promesa se resuelve, aplicamos .json() , y luego en el segundo then, hacemos uso de otra función que llamamos “crearTareas”. Esta función en primer lugar limpia las tareas que ya habían sido renderizadas anteriormente, para traer nuevamente desde el backend todas las tareas; y luego con el método forEach, recorre todo el array de tareas que se obtuvo de la respuesta de la API y se renderiza una por una en pantalla.

## Renderización

Para visualizar específicamente las tareas pendientes, en el template dentro de la función que renderiza las tareas, colocamos como variables la información de la API que indica si la tarea está completa o no, junto con su id, descripción de la tarea y fecha de la tarea. Esta información es parte de la respuesta que nos devuelve el backend, y por eso podemos utilizarla y ponerla ya sea como atributos en le html, o bien como parte de la información que se visualiza. Finalmente, con una sentencia if, podemos insertar las tareas (con inner.HTML) que aún no están completas en el contenedor de tareas pendientes.

# 3. Marcar una tarea como terminada

## Template

En la función que renderiza las tareas, agregamos al template un atributo dinámico “onclick” y como valor de este evento colocamos una función “completarTarea”. A esta función le pasamos como parámetros el id de la tarea, y el booleano de tarea completa que nos devuelve el backend. Esto nos permite que al hacer click en la tarea se ejecute esa función para esa tarea.

## Petición

La función de completar tarea tiene nuevamente el método fetch, al cual le pasamos como parámetros el endpoint, que en este caso termina en “/tasks/idTarea” (el id que entra por parámetro), y el objeto que contendrá información. Este objeto tiene el método PUT ya que en este caso se quiere modificar una tarea, en headers enviamos la autenticación, que es el token que obtenemos del localStorage, y el tipo de contenido que será JSON. En el body enviamos únicamente el dato a modificar que será la propiedad “completed”, y como valor enviamos el booleano que entra por parámetro, pero negado; esto lo hacemos así ya que si una tarea está incompleta (false) y le hacemos click, queremos que la modificación sea marcarla como completa (true). El body lo enviamos en formato String con el método JSON.stringify().

## Trabajando con los datos

Una vez que la promesa esté resuelta, convertimos esa información en formato JavaScript y luego ejecutamos nuevamente la función para obtener todas las tareas nuevamente.

# 4. Ver tareas terminadas

Al ejecutarse la función que obtiene todas las tareas, ésta realiza una petición un GET, luego limpia la pantalla de las que estaban ya renderizadas y vuelve a renderizar las tareas una por una en pantalla, colocando las tareas incompletas dentro del contenedor pendientes, y las completas dentro del contenedor terminadas. Así podemos ver todas las tareas en su correspondiente contenedor y saber si está completa o no.

# 5. Crear nuevas tareas

## Seteo de datos

Creamos la función agregarTareas en la cual obtenemos con un querySelector el valor que coloca el usuario en el contenedor para escribir las tareas nuevas. Guardamos este valor como la descripción de la tarea, y seteamos a la tarea como incompleta.

## Petición

Luego realizamos una petición con fetch, pasándole el endpoint de /tasks, método POST ya que queremos enviar datos a la API, en headers enviamos el token y el tipo de datos, y en el body enviamos la información sobre la nueva tarea que requiere la API (en este caso la descripción y el estado). Luego convertimos el objeto respuesta y finalmente renderizamos esta nueva tarea (aparece en el contenedor de tareas pendientes ya que la seteamos como incompleta).

# 6. Ver la fecha de creación de una tarea

En el template utilizado en la función que renderiza las tareas, uno de los datos que se colocó fue la fecha de creación de la tarea como un párrafo. De esta manera, al renderizarse podremos ver la fecha de creación obtenida desde el backend.

# 7. Conclusiones

En este proyecto se logró unificar y aplicar contenidos teóricos relacionado con selección de datos de una plantilla html, validación de datos, promesas, peticiones a una API, template String, local Storage y eventos. Se logró así crear una página que permite la gestión completa de tareas por parte del usuario, pudiendo en primer lugar loguearse, visualizar sus tareas en secciones diferentes según su estado, crear nuevas tareas, eliminar y modificar su estado.

# 8. Comentarios finales

El proyecto To-Do me resultó muy bueno para integrar todos los conocimientos en una página funcional, entender cómo funcionan las peticiones a una API, y cómo trabajar con la información que ésta devuelve. Me motivó e inspiró para seguir practicando con otras APIs, y lograr así pequeñas aplicaciones o páginas de valor para el usuario.

Me gustaron las dinámicas de las clases, fue muy valioso poder ir viendo en vivo cómo se codea, qué errores o problemas pueden surgir en el momento y buscar distintas formas de solucionarlo, el uso de distintas herramientas (como la consola del navegador, la documentación de la API, etc). Las explicaciones fueron muy didácticas, y me pareció bueno el tiempo dedicado a solventar dudas.

Como feedback para mejora, hubo una o dos clases donde se comprimió mucha información en una sola hora, y resultó muy difícil de entender ya que eran todos temas nuevos, y luego las clases siguientes sobró tiempo; buscaría equilibrar un poco más el contenido y distribuirlo diferente para poder incorporar mejor los conocimientos en cada clase. Y esto lo complementaría con ejercicios más acotados en el Playground para poder practicar el tema introducido en la clase.

Como apreciación final, disfruté muchísimo esta materia, me encantó aprender estos temas y tengo muchas expectativas por seguir haciéndolo en el futuro y seguir descubriendo todo lo que implica ser un desarrollador FrontEnd. ¡Muchas gracias por toda la dedicación, el esfuerzo y el compromiso clase a clase para que aprendamos y logremos cumplir con los objetivos de la materia!