App To-Do's

Para qué sirve la app

Aplicación web que permite crear, eliminar, buscar y filtrar una lista de tareas.

Explicación del código:

1. Importación de dependencias: se importa la función `nanoid` de la biblioteca "nanoid", que se utiliza para generar identificadores únicos.

```
import { nanoid } from '../node_modules/nanoid/nanoid.js'
```

2. Selección de elementos del DOM: se utiliza el método `document.querySelector` y `document.getElementById` para seleccionar elementos HTML del DOM: el formulario para agregar tareas, el section de la lista de tareas, el select para filtrar por prioridad y el input-search (buscador).

3. Almacenamiento en Local Storage: se definen dos funciones para gestionar el almacenamiento en el Local Storage: `getTasksFromLocalStorage` para obtener las tareas almacenadas y `updateTaskToLocalStorage` para actualizar el almacenamiento después de agregar o eliminar una tarea.

4. Función `deleteTask`: elimina una tarea de la lista de tareas, y luego llama a `updateTaskToLocalStorage` para reflejar el cambio en el almacenamiento local.

5. Función `createTaskHTML`: se utiliza para crear el HTML de una tarea. Cada una se representa como `<article>`, e incluye el nombre de la tarea y un ícono para eliminarla. También se añade el método `.addEventListener()` al ícono de eliminar para que se pueda quitar la tarea cuando se hace click sobre aquél.

6. Función `colorTask`: asigna un color de fondo diferente a cada tarea según su prioridad (urgente, intermedia o normal).

7. Función `printTasks`: se encarga de mostrar las tareas en la lista de tareas. Borra el contenido del elemento `listTask`, recorre las tareas del `array` que se le pase como parámetro, y pinta éstas en pantalla. Por tanto, va servir para reflejar en el documento cualquier array que necesitemos.

8. Función `checkEmptyInputAndGetValue`: verifica si el campo donde se escribe una nueva tarea está vacío.

9. Función `createAlertBootstrap`: se utiliza para mostrar un mensaje de alerta en caso de no crear correctamente la tarea.

10. Función `createNewTaskBBDD`: crea una nueva tarea con un identificador único, un nombre y una prioridad, y la agrega al array `allTasks`.

11. Función `handleSubmit`: primero verifica si el campo de entrada de la tarea está vacío y muestra una alerta si es necesario. Luego, guarda la tarea y la prioridad, y llama a `createNewTaskBBDD` para agregarla al array. Finalmente, llama a `printTasks` para actualizar la lista de tareas en pantalla.

12. Función `filterTasksbyPriority`: filtra las tareas según la prioridad seleccionada en el elemento `selectFilter`. Puede mostrar todas las tareas o sólo las tareas de prioridad urgente, intermedia o normal.

13. Función `handleSearchFilter`: sirve para realizar la búsqueda de tareas según el texto introducido en el campo de búsqueda. Filtra las tareas en función del texto y la prioridad seleccionada y luego llama a `printTasks` para mostrar las tareas filtradas.

14. Event Listeners: se usa el método `.addEventListener` para los elementos del formulario, el select de prioridad y el campo de búsqueda, con el objetivo de responder a eventos como enviar el formulario (guardar nueva tarea), elegir un nuevo valor en el select (filtrar por prioridad) o escribir en el campo de búsqueda (input tipo search).

15. Inicialización: al final del código, se llama a `getTasksFromLocalStorage` para recuperar las tareas almacenadas en el Local Storage, y luego se llama a `printTasks` para mostrarlas en pantalla.