Trabajo práctico evaluativo.

Cierre de conceptos de Node JS

Consigna:

Crear una aplicación de tipo servidor, usando ***Node JS***, que satisfaga el siguiente requerimiento.

Escenario:

1. Un cliente necesita crear un ***sistema de gestión***, donde va a registrar pagos de los usuarios. Para ello, el servidor debe:
    a. Registrar pagos
    b. Registrar los usuarios
    c. Poder subir, solo en formato PDF los recibos.

Funcionalidades:

1. El servidor consta de 2 tipos de usuarios, los administradores, capaces de cargar nuevos usuarios, y subir archivos, como tambien, registrar pagos.
2. Los usuarios comunes ***SOLO PUEDEN VER SUS REGISTROS DE PAGOS*** y ***DESCARGAR LOS RECIBOS***.
3. Debe existir ***UN SOLO SUPERUSUARIO***, capaz de crear usuarios administradores, y los usuarios administradores pueden crear ***USUARIOS COMUNES***.

Socket:

1. Es opcional el uso de sockets, pero requerido una vez que el sistema comience a crecer. El mismo debe ser capaz de comunicar los usuarios ***ADMINISTRADORES*** conectados, compartiendo la información cargada. En otras palabras, los administradores están siempre conectados al sistema, por lo que cada registro cargado, debe ser visualizado en tiempo real, actualizando, del lado del cliente, la tabla que contenga los registros.

Paginador:

1. Como la información va a crecer en medida, para alivianar la carga de los registros, se debe trabajar con paginadores.
    a. Recuerden que existen los ***query params***, donde pueden enviar parámetros de busqueda.
    b. El paginador debe tener:
        i. Página
        ii. Límite
        iii. Ordenar por

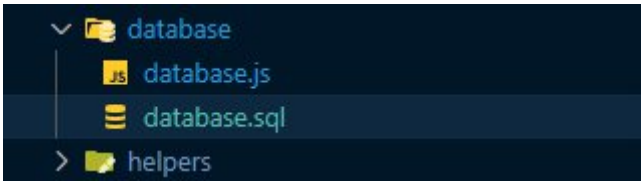
Este concepto es nuevo, por lo que no es obligatorio, pero se tendrá en cuenta el esfuerzo adicional.

Requisitos:

1. **Buenas prácticas**:
    a. Implementar principios de ***responsabilidad unica*** y ***clean code***.
    b. ***Comentarios*** de tipo descriptivos:

```
/**
 * @description
 * This function do something
 * @param { string } someParam
 * @returns { string } return something as string
 */
const someFunction = (someParam) => {
  return someParam;
};
```

2. **Manejo de errores**:
    a. Incluir un manejo adecuado de ***errores***, asegurando que se devuelvan respuestas ***claras*** y ***apropiadas*** (por ejemplo, códigos de error HTTP y mensajes detallados):
        i. Asegurar que la aplicación devuelva códigos de error HTTP correctos (ej. 404 cuando una parcela está ocupada, 400 para datos inválidos, etc.).
        ii. Proporcionar mensajes claros al cliente cuando ocurran errores.
3. **Base de datos**:
    a. Pueden usar ***Sequilize***, o no, es decisión de cada estudiante. Pero incluyan, en la carpeta de database, el ***DUMP*** de la base de datos



```
CREATE TABLE `some_table` .........
```

4. Incluir el ***MER*** de la base de datos. Pueden usar herremientas tales como DRAW.IO o EXCALIDRAW.
5. Usar pruebas unitarias de cada función creada. La idea es que el servidor se pruebe por ese medio, y no usando Thunder Client, Postman o Insomnia.
6. **Opcional**: Integrar ***Socket.IO*** para la comunicación en tiempo real (se valorará el esfuerzo adicional)