## Preguntas

¿Qué son los streams en NodeJs?

Describa cómo es la creación de un stream en NodeJs

¿Qué método se usa para la escritura de streams y que parámetros emplea?

Describa cada uno de los eventos empleados para streams en NodeJs

¿Como se hace la lectura de streams en NodeJs?

Para qué sirve el método pipe() en los streams en nodeJS

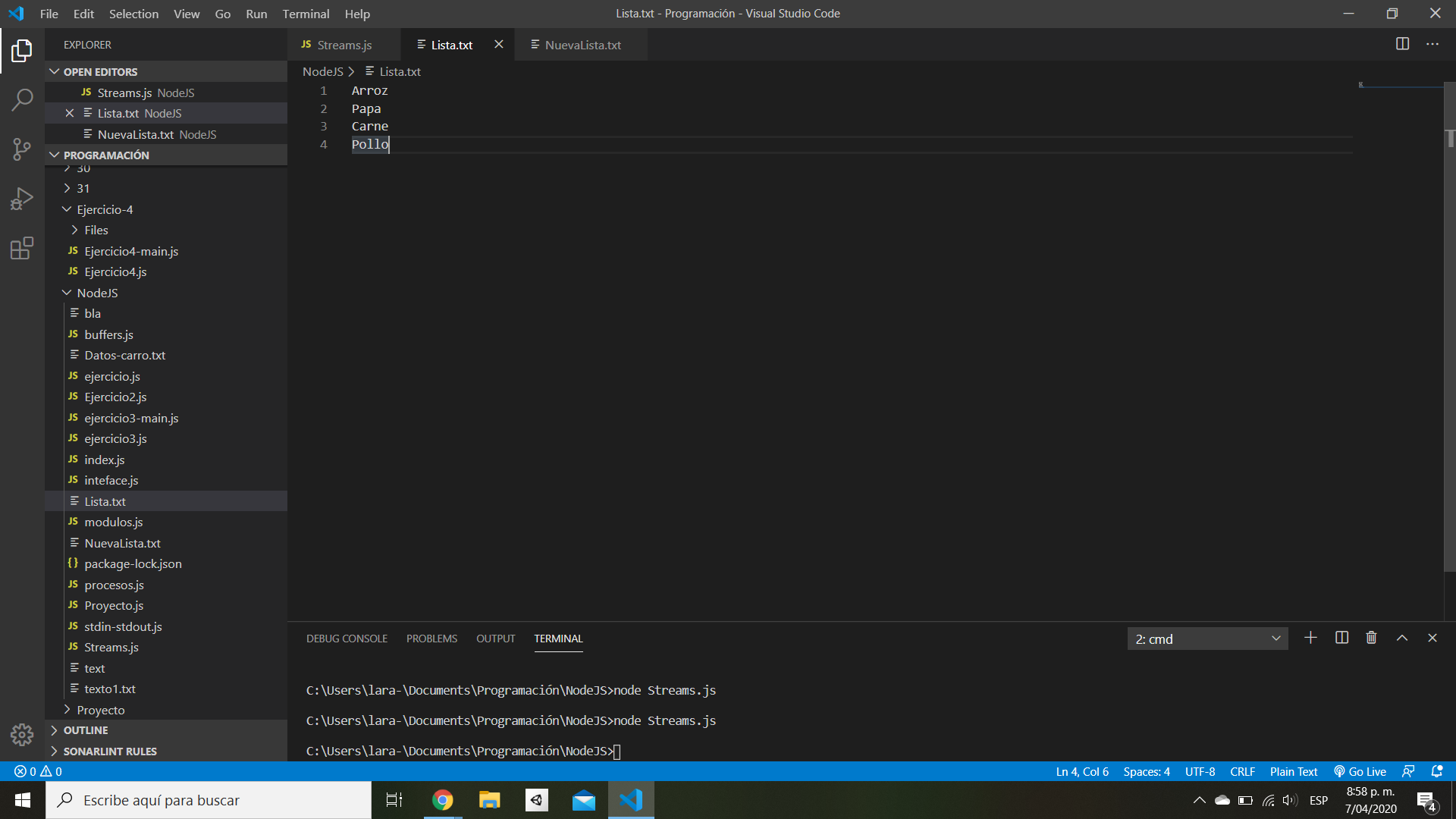
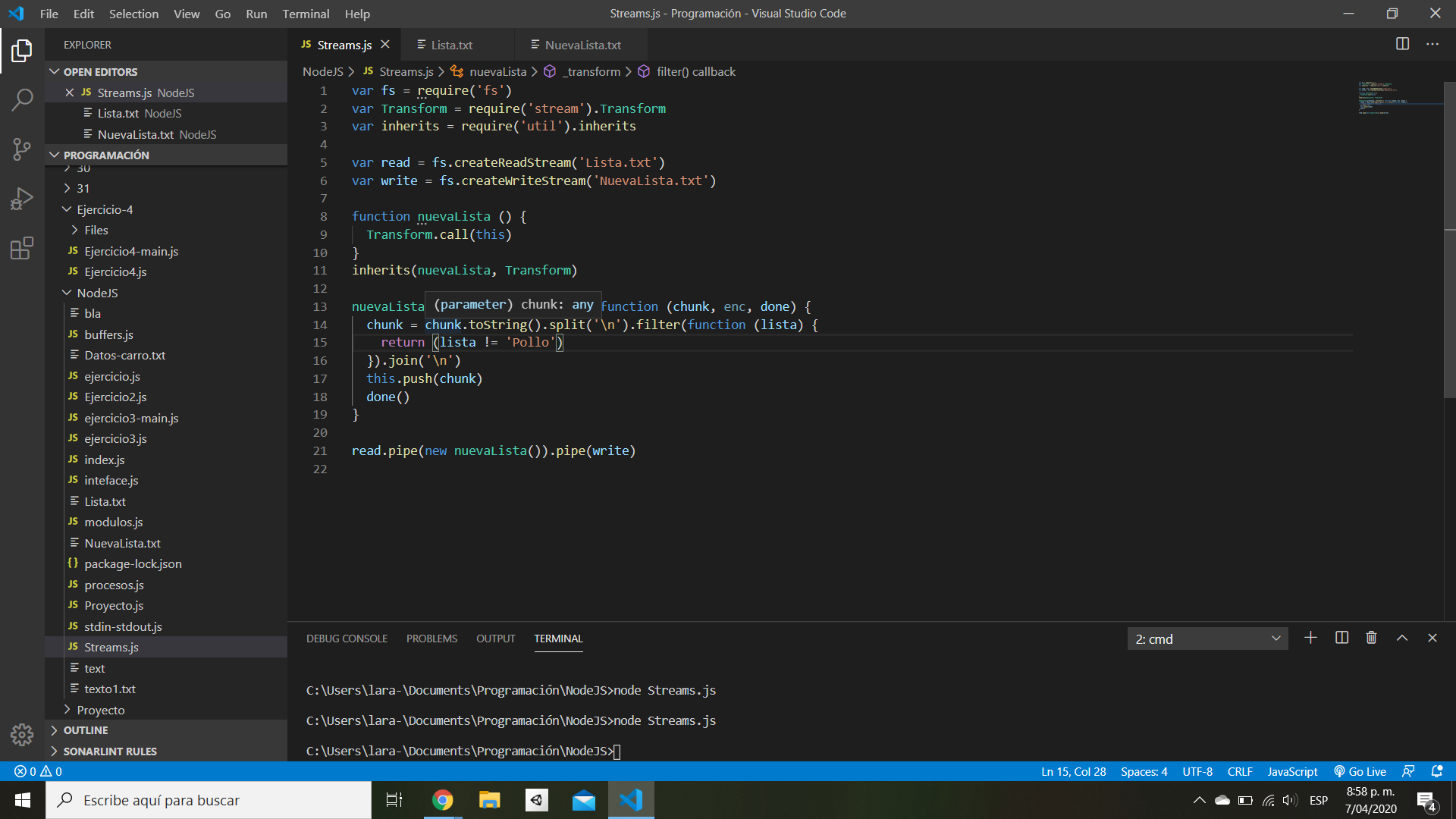
## Ejercicio

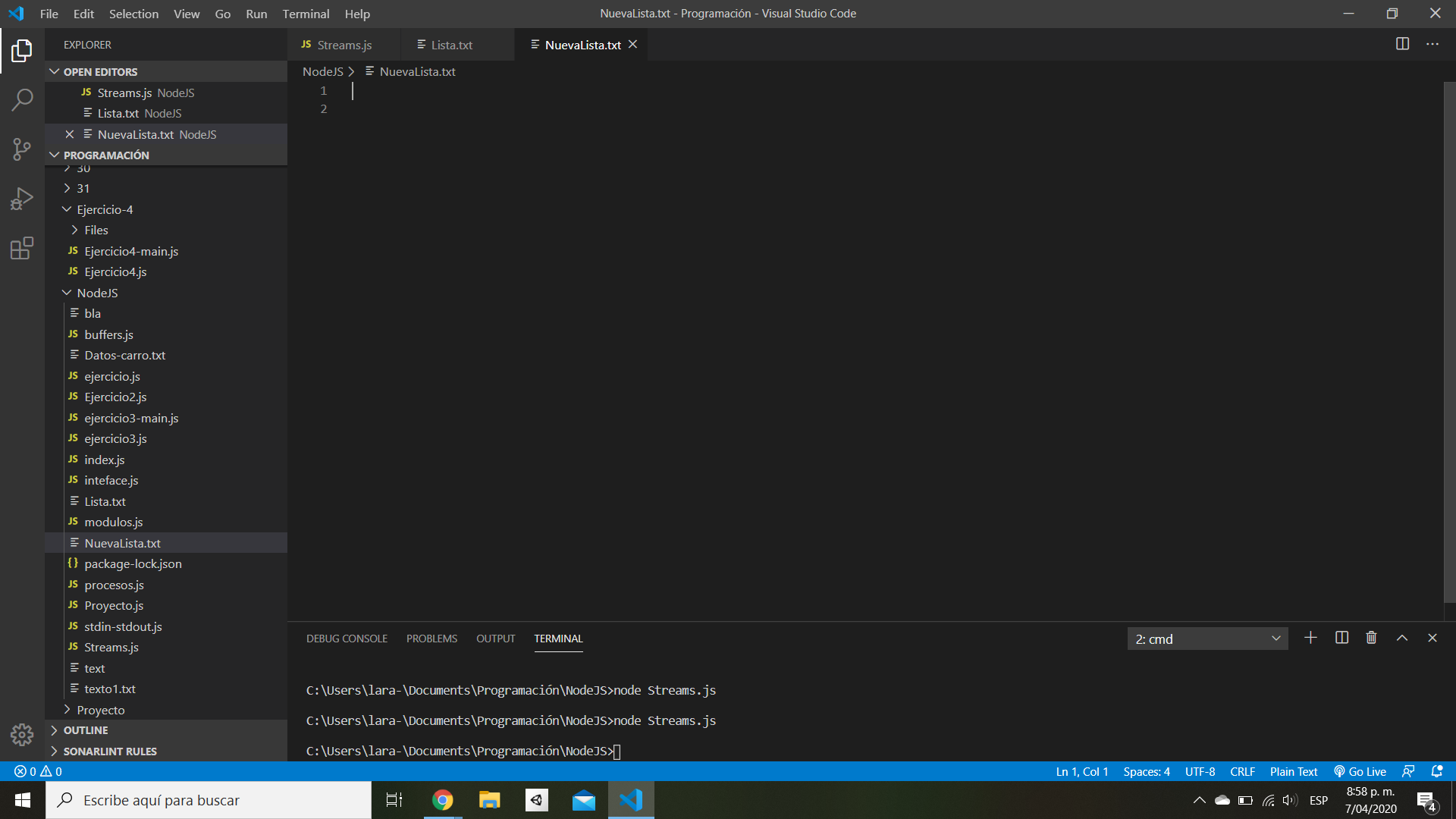
Desarrollar un programa en NodeJs que emplee el uso de Streams usando gran parte de las características, métodos y eventos creados para los streams en NodeJs.

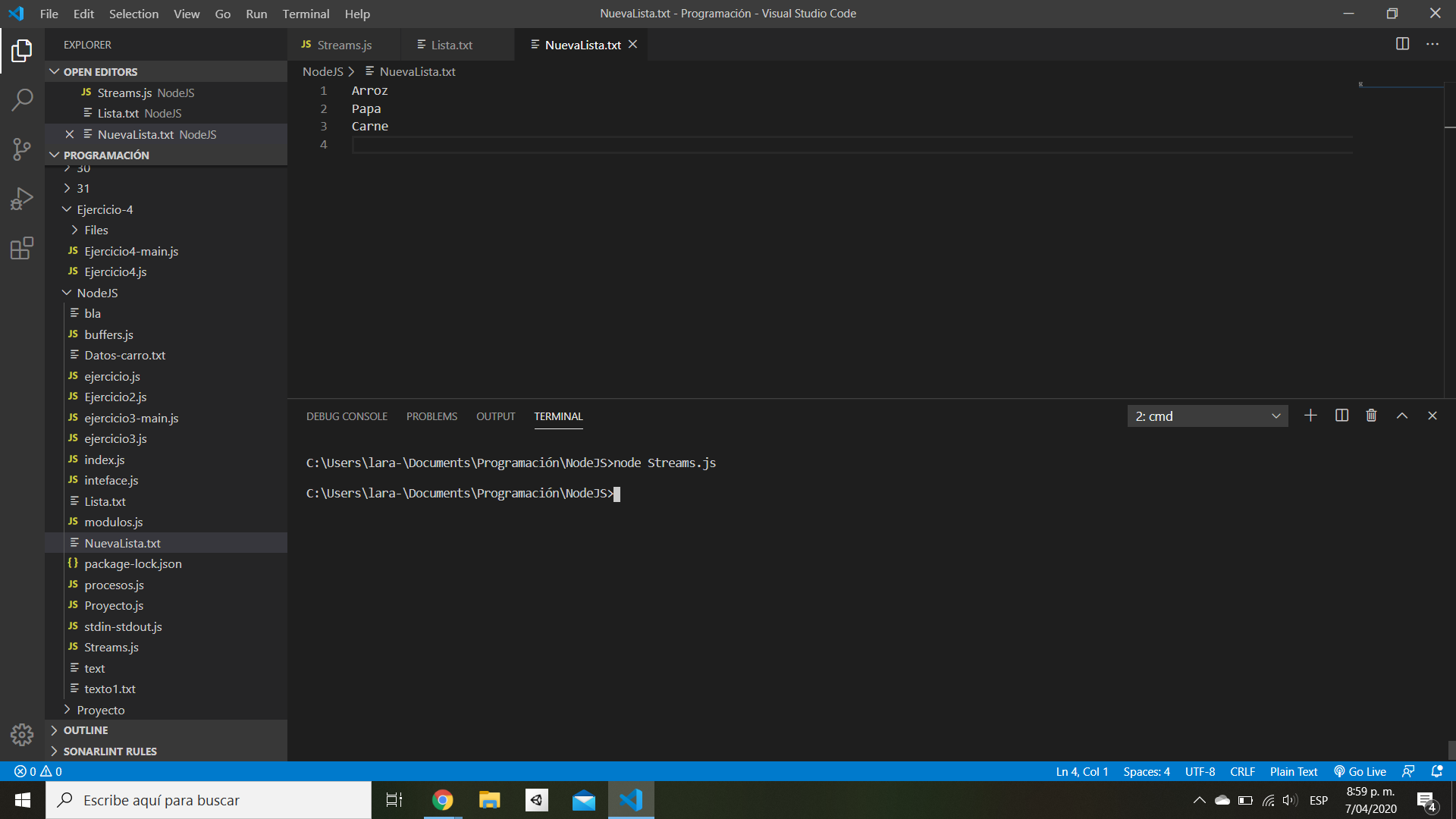
**Desarrollo**

* Los streams en node js son una alternativa para el manejo de datos con la particularidad de que no es necesario que estos se carguen en memoria inmediatamente para poder leerlos, sino que divide la información en porciones más pequeñas y la va suministrando en un flujo de datos constante, los cuales se procesan sin la necesidad de mantener todo en memoria. Esto hace que los streams sean muy eficientes para el manejo de grandes cantidades de datos, ya que optimiza considerablemente el uso de memoria y el tiempo que se requiere para cargar la información necesaria.
* Para la creación de un stream writable utilizamos la el método fs.createWriteStream() mediante el módulo de file system, y para la creación de un stream readable utilizamos fs.createReadableStream(). También podemos crear un stream importando el módulo “stream” he instanciando la clase readable o writable según el tipo que necesitemos.
* Para la escritura de streams se utiliza el método write() y emplea los parámetros:
  1. Chunk. Se le pasa un string, buffer o uint8Array con el contenido de lo que se quiere escribir.
  2. Encoding. Se le pasa un string especificando el tipo de codificación de texto del primer parámetro.
  3. Callback. Se le pasa una función callback para el manejo de errores.
* Eventos Stream writable:
  1. Event close: Se debe emitir cuando el stream y todos sus procesos subyacentes hayan terminado, este evento indica que no se va a emitir ningún otro evento después.
  2. Event drain : Este evento se utiliza para indicar que un stream writable ha sido vaciado y está listo para recibir más información.
  3. Event error : Se emite cuando ocurre un error durante la transmisión del stream.
  4. Event finish: Este evento se emite después del método stream.end(), el cual indica que no se va a escribir más en el stream y todos los datos han sido vaciados.
  5. Event pipe: Se emite cuando se llama al método stream.pipe() en un readable stream lo cual hace que la entrada de datos sea readable y la salida writable.
  6. Event unpipe: Se emite cuando se llama el método stream.unpipe(), y le quita la forma writable a la salida de los datos de un stream readable.
* Eventos de Stream Readable:
  1. Event close: Este evento se emite cuando el stream se ha cerrado.
  2. Event end: Se emite cuando ya no hay más datos que consumir.
  3. Event error: Se emite cuando ocurre un error durante la transmisión del stream.
  4. Event readable : Este evento se emite cuando hay datos disponibles para leer por el stream
  5. Event pause: Se emite después del llamado del método stream.pause() que indica que la transmisión de datos se ha detenido.
  6. Event resume: Se emite después del llamado del método stream.resume() el cual detiene la lectura de datos, se utiliza para terminar con él con el flujo del stream sin procesar los datos después del llamado de este método.
* Para hacer la lectura de streams en node js desde un stream writable tenemos que utilizar el método pipe()
* El método pipe() nos permite crear una conexión entre un stream de escritura y uno de lectura con el fin de poder leer o escribir los datos que reciben estos streams

**Ejercicio.**

****

****

****