**MÓDULO 4 PRÁCTICA 1 APP MENSAJERÍA**

**\*El smart contract lo he desarrollado en inglés ya que he hecho más cursos en concreto sobre esto y bueno todos los contratos se hacían en inglés \***

**Principalmente un smart contract consta de una estructura y a esta se la dota de funcionalidades mediante eventos y funciones. A continuación explico paso a paso como he desarrollado el ejercicio.**

1. **Licencia:**  puse la que venia por defecto en el store.sol de Remix (GLP-3.0)
2. **Versión Solidity:** establecí un rango de las versiones del lenguaje Solidity mínima y máxima para poder compilar el Smart Contract
3. **Comentarios:** aquí se encuentra tanto el nombre del contrato como las principales funciones de este.
4. **Contrato:** empiezo definiendo con la palabra reservada *contract*el nombre del contrato. Dentro de este se encontrará tanto la estructura como las funciones.
   1. **Estructura (Struct):** mediante la palabra reservada *struct* defino las características necesarias (variables) que debe tener el mensaje. Una variable de tipo string, en la cual se almacenará el contenido del mensaje. Una variables de tipo address, en la cual se almacenará el remitente/autor del mensaje. Una variable de tipo booleano, por la cual sabremos si el mensaje ha sido leído o no.
   2. **Mapeo de los mensajes (Mapping):** El mapping recibe como parámetro que un numero de tipo uint256 que deberá ser mayor o igual a la estructura del mensaje (Struct Message), que actualmente es 0. Este mapa es público y almacenara en messages, creando así un identificador único (messageId) para cada mensaje llevando así a la vez un conteo automático de estos (messageCount).
   3. **Eventos:** nos servirán para depurar o hacer que las funciones queden más legibles y el código más limpio.
      1. **Evento Mensaje Escrito:** este evento será usado posteriormente para la funcion de escribir mensaje. Recibe los parámetros del contenido y el address del remitente, pero además recibe el parametro de un identificador de mensaje (messageId).
   4. **Funciones:** 
      1. **WriteMessage:** 
         1. **Estructura y Funcionamiento:** Esta función permite a un usuario escribir un mensaje. Toma como entrada un string \_content (variable local) que representa el contenido del mensaje. Antes de guardar el mensaje, se verifica que su longitud no exceda el límite permitido de 300 caracteres . Luego, se crea una nueva estructura Message con el contenido del mensaje, la dirección del autor (msg.sender) y el estado de lectura establecido en false. La estructura Message se guarda en el mapeo messages usando messageCount como clave. Se emite el evento MessageSent para así notificar/avisar la acción y finalmente se incrementa el contador de mensajes (messageCount) en uno.
      2. **ReadMessages:**
         1. **Estructura y Funcionamiento:** Esta función devuelve un array de strings que contiene el contenido de todos los mensajes escritos hasta el momento. Primero se crea un “array result” con el numero de mensajes que contiene y es igual a messageCount. Luego, se recorre el mapeo messages desde 0 hasta messageCount - 1 y se asigna el contenido de cada mensaje al elemento correspondiente del array result. Finalmente, se retorna el array result con todos los mensajes.
      3. **readUnreadMessages:**
         1. **Estructura y Funcionamiento :** Esta función devuelve un array de strings que contiene el contenido de los mensajes no leídos. Primero se inicializa unreadCount en 0. Luego, se recorre el mapeo messages y se incrementa unreadCount por cada mensaje cuyo estado de lectura (read) sea false. Después, se crea un nuevo array result de tamaño unreadCount y se recorre nuevamente el mapeo messages. Si el estado de lectura del mensaje es false, se asigna el contenido del mensaje al elemento correspondiente del array result y se incrementa el índice del array (currentIndex). Finalmente, se retorna el array result con los mensajes no leídos.
      4. **markMessageAsRead:**
         1. **Estructura y Funcionamiento:** permite al autor/remitente de un mensaje marcar este como leído. Solo el autor puede/tiene los permisos para realizar esta acción y solo se puede marcar como leído un mensaje que aún no haya sido leído. Así para saber esto el contrato busca en el mapeo del total de mensajes cuales han sido leídos y cuáles no. Una vez hecho esto “depura” los mensajes leídos, se queda con los mensajes no leídos y el autor puede poner el Id del mensaje no leído para marcarlo como leido.
      5. **getMessageAuthor:**
         1. **Estructura y Funcionamiento:** Esta función devuelve la dirección del autor de un mensaje dado su identificador \_messageId. Primero se verifica que el \_messageId sea válido (menor que messageCount). Luego se retorna la dirección del autor (messages[\_messageId].author).