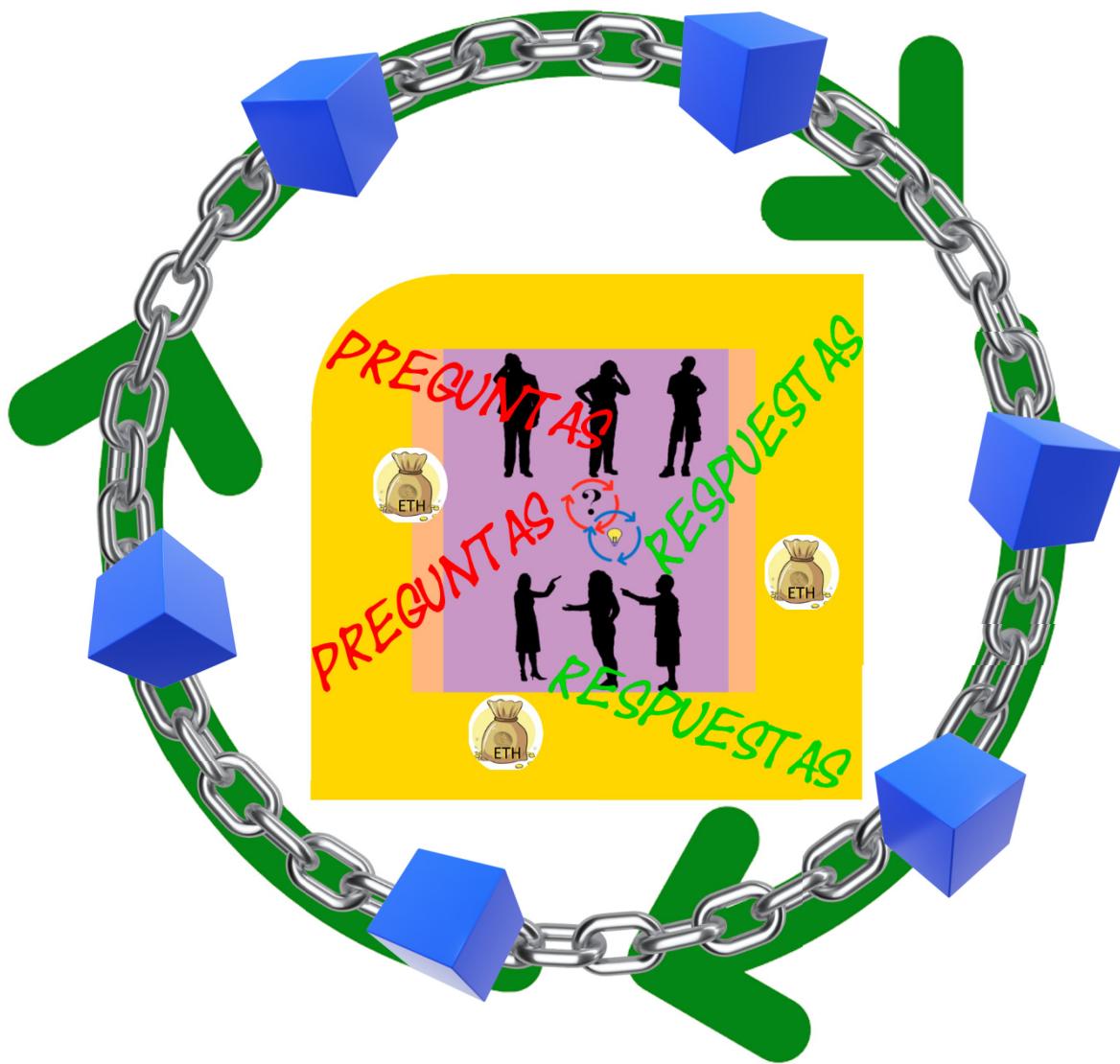


Proyecto Ethereum

Sistema de conocimiento colaborativo

Esbrinachain



Autor: **Joaquim de Dalmases Juanet**



Esta obra está sujeta a una licencia de
[Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

Índice de Contenido

Contenido	
01. INTRODUCCIÓN	6
02. GITHUB	6
03. PROPUESTA DE PROYECTO.....	8
04. ESTRUCTURA DEL PROYECTO	11
4.1 Smarts Contracts	12
4.1.1 Test unitarios	26
4.2 Angular Framework.....	32
4.2.1 Estructura del proyecto	32
4.2.2 Descripción de los componentes.....	32
4.2.3 Particularidades en la funcionalidad de Esbrinachain.....	40
4.2.4 Complementos destacables del proyecto Esbrinachain	46
4.3 Cloud Firestore – Firebase.....	49
4.3.1 Colecciones de datos en Firestore Cloud	50
05. Casos de uso identificables en Esbrinachain	51
5.1 Gestión de alta de usuarios	51
5.2 Crear una pregunta.	54
5.3 Crear una respuesta.	57
5.4 Iniciar una votación y finalizar una votación por parte del administrador. ..	59
5.5 Votar un respuesta.....	60
5.6 Resolución de la votación.....	61
5.7 Cálculo de la reputación del usuario en el sistema.....	62
5.8 Incremento de la recompensa de una pregunta (3 tentativas)	62
5.9 Detalles de visualización en el interfaz.....	63
5.10 Soporte de Idioma.....	63
5.11 Línea de estado para información del usuario.	64
5.12 Consulta mediante pago de la respuesta más votada.....	64
5.13 Configuración de un backend de datos en Firestore	64
06. CONCLUSIONES	65
07. AGRADECIMIENTOS.....	65
08. Anejo I : Instalación de Metamask Wallet.....	66

ÍNDICE DE FIGURAS

<i>Figura 1:</i> Publicación en GitHub del proyecto Esbrinachain .	7
<i>Figura 2:</i> Documento INICIAL de especificación técnica del proyecto Esbrinachain .	8
<i>Figura 3:</i> Documento FINAL de especificación técnica del proyecto Esbrinachain .	10
<i>Figura 4:</i> Estructura de directorios completa del proyecto Esbrinachain .	11
<i>Figura 5:</i> Código fuente del contrato inteligente ‘preguntar’ (<i>esb_preguntar.sol</i>).	13
<i>Figura 6:</i> Estructura de una pregunta, una respuesta y de un usuario.	14
<i>Figura 7:</i> Variables globales declaradas en el contrato inteligente preguntar .	14
<i>Figura 8:</i> Declaración de las funciones y parámetros del contrato preguntar .	15
<i>Figura 9:</i> Declaración de los eventos del contrato inteligente preguntar .	15
<i>Figura 10:</i> Código fuente del contrato inteligente responder .	17
<i>Figura 11:</i> Variables definidas en el contrato inteligente responder .	18
<i>Figura 12:</i> Eventos definidos en el contrato inteligente responder .	18
<i>Figura 13:</i> Funciones definidas en el contrato inteligente responder .	19
<i>Figura 14:</i> Variables declaradas en el contrato inteligente esbrinachain .	20
<i>Figura 15:</i> Funciones declaradas en el contrato inteligente esbrinachain .	20
<i>Figura 16:</i> Código fuente del contrato inteligente esbrinachain .	26
<i>Figura 17:</i> Salida por consola de la carga del contrato esbrinachain para los test unitarios Mocha.	27
<i>Figura 18:</i> Código fuente de los test unitarios del proyecto Esbrinachain .	29
<i>Figura 19:</i> Resultado obtenido en la ejecución de los test unitarios.	31
<i>Figura 20:</i> Componente get-preg .	33
<i>Figura 21:</i> Componente get-resp .	34
<i>Figura 22:</i> Visualización en el aplicativo web del componente pregunta .	35
<i>Figura 23:</i> Visualización de los componentes pregunta y respuesta .	35
<i>Figura 24:</i> Visualización del componente respuesta .	36
<i>Figura 25:</i> Visualización de la reputación del sistema en esbrinachain .	37
<i>Figura 26:</i> Pantalla para el alta de nuevos usuarios en Esbrinachain .	37
<i>Figura 27:</i> Pantalla de generación de wallet para un usuario recién registrado.	38
<i>Figura 28:</i> Pantalla de acceso a utilizar el wallet Metamask.	38
<i>Figura 29:</i> Librerías para la generación del wallet de usuario.	39
<i>Figura 30:</i> Variables locales de sesión identificativas del usuario.	40
<i>Figura 31:</i> Código implementado para la creación de una pregunta.	41
<i>Figura 32:</i> Código implementado para la creación de una respuesta.	43
<i>Figura 33:</i> Código HTML del template del componente respuesta incrustado en el de pregunta .	44
<i>Figura 34:</i> Detección automática del cambio al estado ‘ votando ’ en una pregunta.	44
<i>Figura 35:</i> Código implementado para ejecutar un voto por una respuesta.	45
<i>Figura 36:</i> Fichero de configuración Firestore1.ts .	46
<i>Figura 37:</i> Fichero de configuración para el soporte en varios idiomas (<i>idioma.ts</i>).	47
<i>Figura 38:</i> Recursos para la adquisición de faucets y estado de la red Sepolia.	48
<i>Figura 39:</i> Código de actualización de estado para una pregunta en Cloud Firestore.	49
<i>Figura 40:</i> Gráfica representativa de uso y límites de Firestore Cloud.	49
<i>Figura 41:</i> Cuadro de las colecciones y documentos utilizados en el Backend.	50
<i>Figura 42:</i> Relación de índices compuestos en Firestore.	50
<i>Figura 43:</i> Fase1: Alta de un nuevo usuario: no existe en el backend ni en la aplicación.	51
<i>Figura 44:</i> Fase 2: El usuario está registrado pero no dispone de wallet.	52
<i>Figura 45:</i> Fase 3: El usuario está registrado y dispone de wallet.	52
<i>Figura 46:</i> Fase 4: El usuario está registrado y dispone de wallet y utiliza el wallet integrado.	53
<i>Figura 47:</i> Fase 3: El usuario está registrado y dispone de wallet y utiliza el wallet de Metamask.	53
<i>Figura 48:</i> Fase 4: Cualquier usuario registrado puede darse de baja del sistema.	53
<i>Figura 49:</i> Interfaz de usuario de la aplicación web del sistema Esbrinachain .	54
<i>Figura 50:</i> Proceso de crear una pregunta en el sistema Esbrinachain .	55
<i>Figura 51:</i> Visualización de la pregunta creada, sus metadatos y el resto de datos actualizados.	56

<i>Figura 52:</i> Detalle de la transacción para crear una pregunta.....	56
<i>Figura 53:</i> Interfaz de la aplicación web al crear una respuesta.....	57
<i>Figura 54:</i> Línea de notificación del Interfaz web de Esbrinachain	57
<i>Figura 55:</i> Respuesta creada con índice 1 en el interfaz de Esbrinachain	58
<i>Figura 56:</i> Detalle la transacción para crear una nueva respuesta.....	58
<i>Figura 57:</i> Situación inicial antes del inicio de la votación.....	59
<i>Figura 58:</i> Cambio de estado de una pregunta de ‘ abierta ’ a ‘ votando ’.....	59
<i>Figura 59:</i> Proceso de Votación de una pregunta en el sistema Esbrinachain	60
<i>Figura 60:</i> Receipt de la transacción de voto.....	60
<i>Figura 61:</i> Variable ‘ preguntas ’ en el contrato inteligente Esbrinachain para la primera pregunta	61
<i>Figura 62:</i> Pregunta en estado de consulta en el sistema Esbrinachain	61
<i>Figura 63:</i> Ranking por reputación en el sistema Esbrinachain	62
<i>Figura 64:</i> Primera tentativa por incrementar la recompensa.....	62
<i>Figura 65:</i> Resultado de la primera tentativa de incremento de recompensa.....	63
<i>Figura 66:</i> Puntos de información de interés para el usuario de Esbrinachain	63
<i>Figura 67:</i> Disponibilidad de tres idiomas para el interfaz de la aplicación web.....	63
<i>Figura 68:</i> Visualización del resultado de la votación de una pregunta en Esbrinachain	64
<i>Figura 69:</i> Cuadro resumen del proceso de instalación de Metamask.....	66

01. INTRODUCCIÓN

El **objetivo principal** de este proyecto consiste en **poner en práctica** los conocimientos adquiridos durante la primera parte del [Máster en Blockchain Engineering](#), centrado en el desarrollo de **aplicaciones web o móviles sobre blockchain pública**. Para el proyecto **Esbrinachain** se ha procedido a definir tres contratos inteligentes utilizando el **lenguaje Solidity** y una aplicación web desarrollada en **Angular Framework v17**. Para la conexión entre **Angular** y la **blockchain** se han utilizado las librerías **web3**, **bitcore-mnemonic**, **ethereumjs-wallet**, **bip39** y **ethereumjs-util**. Lo más cercano a practicar un caso real con la red Ethereum es desarrollar aplicaciones sobre la **Testnet de Sepolia**, y eso, es lo que se ha realizado en el proyecto **Esbrinachain**. Para el usuario interesado en conocer el funcionamiento de una base de datos documental en la nube puede ser interesante ver los desarrollos realizados sobre **Firestore (Google Cloud)**.

En los siguientes apartados se procederá a describir cada una de las partes en las que podemos dividir el desarrollo del proyecto. Una de las **ventajas** de usar una red blockchain pública, consiste en evitar inversiones en crear una **estructura hardware** para el proyecto. La red Testnet de Sepolia al ser **pública**, concede la conexión a cualquier usuario potencial de la aplicación, utilizando un navegador de Internet (como Google Chrome). Así pues, el lector puede probar el proyecto sin costes, pero si deberá tener en cuenta que para realizar transacciones en Sepolia debe disponer de ‘faucets’, tokens ETH propios de Sepolia que no tienen coste, pero que resulta cada vez más complicado adquirirlos sin haber realizado operaciones previas en la Mainnet de Ethereum. También es necesario disponer de alojamiento web para la aplicación, y para **Esbrinachain** se ha utilizado la opción de alojar la aplicación en **Github** de forma gratuita, aprovechando la capacidad que ofrece para publicar una aplicación sin coste.

Este documento contiene la descripción técnica de todo el desarrollo del proyecto **Esbrinachain**, e incluye la documentación relacionada con:

- Definir e implementar el proyecto tipo Ethereum para una aplicación Web.
- Metodología para la puesta en marcha y uso de este proyecto.

02. GITHUB

Para tener acceso al proyecto completo y disponer de una primera toma de contacto de **Esbrinachain**, podemos acceder a Github utilizando el siguiente enlace: <https://github.com/EsbrinaChain/esbrina>

En este enlace encontraremos el **desarrollo de la aplicación web** en Angular Framework, el **documento técnico** y el **video presentación** del autor.

La aplicación web está localizada en la carpeta **docs**. El conjunto de carpetas y ficheros es básicamente fiel reflejo del desarrollo web realizado en Angular Framework. En el fichero de GitHub **README.CMD** podemos leer una primera definición de lo que es el proyecto **ESBRINACHAIN**:

“Un sistema colaborativo de conocimiento sobre red blockchain pública”

Esbrinachain proporciona una espacio en la nube para poder **preguntar lo que nos interese saber y poder contestar lo que a los demás les interesa**. Esbrinachain se diferencia del resto de proyectos, en que basa su funcionamiento en **contratos inteligentes** almacenados en una red blockchain pública. Su propósito es la gestión del conocimiento y la gestión de **activos de valor** para cada usuario. Proporciona también las características propias de transparencia e independencia de la necesidad de una tercera parte que ejerza como ente oficial reputado o certificador en la **transmisión de activos de valor**. Incorpora para cada usuario el uso de billeteras y autonomía para los usuarios para que ofrezcan recompensas económicas por sus preguntas y reciban beneficios por las respuestas que resulten ser las más votadas por el resto de los usuarios. En la figura 1, se puede ver el proyecto publicado en Github.

EsbrinaChain

Sistema colaborativo de conocimiento en blockchain

Blockchain Sepolia Testnet: [EsbrinaChain]



Interfaz de la aplicación - Website

Login Registro
Idioma: ES

ESBRINA CHAIN - Sistema de Conocimiento Colaborativo.

Compartiendo el conocimiento de todos usando tecnología Blockchain.

Bote acumulado **100 Wei**

Tiempo de Respuesta: 7 días
Período de Votación: 7 días
Cupo de Respuestas: 15

[Realizar una Nueva Pregunta](#)

Nº 1 ¿De qué color tienen los ojos los delfines?

Autor: user6@gmail.com	Fecha: 07/09/2024	Estado: votando	Recompensa: 50 + (wei)(intento:2)	Fecha Votación 08/09/2024	Bote acumulado 100 Wei
			Usuario user5@gmail.com Balance del Usuario 0.361340275884959435 (wei) 0.0000000000000001 ETH		
Tiempo de Respuesta: 7 días Período de Votación: 7 días Cupo de Respuestas: 15					

1- Verdes y azules
2- Grises
3- El color de los ojos de los delfines, va cambiando según su edad. En la edad joven es azul claro, cuando llega a ser adulto tiene los ojos negros y cuando son viejos se les ponen los ojos de un gris oscuro
4- Negros

Votar

Nº 2 ¿En que año acabó la II Guerra Mundial?

Autor: user1@gmail.com	Fecha: 07/09/2024	Estado: consulta	Recompensa: 50 + (wei)(intento:2)	Fecha Votación 08/09/2024	Bote acumulado 100 Wei
			Usuario user1@gmail.com votos 0 user2@gmail.com votos 0 user3@gmail.com votos 0 user4@gmail.com votos 0		
1- 1945 2- 1944 3- 1943					

Ver Solución
Coste 50%

Figura 1: Publicación en GitHub del proyecto Esbrinachain.

03. PROPUESTA DE PROYECTO

ESBRINACHAIN se pensó como un proyecto que utiliza la tecnología blockchain de la red compatible **ETHEREUM** para implementar **UN SISTEMA DE CONOCIMIENTO COLABORATIVO**. En este apartado se describe la propuesta inicial (figura 2) que se presentó a **EBIS Techschool**, y se subrayan los cambios realizados durante la realización del proyecto, quedando definida la propuesta final (figura 3).

Sistema colaborativo de conocimiento en blockchain.

Blockchain: EsbrinaChain token: SAP

El sistema es público y de libre acceso sobre la testnet de Ethereum (Sepolia).

Cada participante puede formular la pregunta que deseé y fijar un valor en tokens, por el que está dispuesto a pagar una respuesta a modo de recompensa al usuario que la facilite. Opcionalmente se puede fijar un número de respuestas para cerrar la pregunta, sino por defecto serán 100. El valor de coste por la respuesta quedará custodiado por el depósito de capital del sistema.

Si el propietario de una pregunta no ve ninguna respuesta a su pregunta podrá realizar 3 incrementos de valor de recompensa, o eliminar la pregunta formulada. Si elimina la pregunta recibirá el valor del 50% de lo propuesto por una respuesta y el otro 50% permanecerá en el depósito del sistema.

Una pregunta que permanezca sin respuesta durante 1 semana será eliminada por el sistema y el usuario autor no recibirá el aporte realizado por una respuesta pasando al bote del sistema.

Los usuarios que estén dispuestos a ganar la recompensa de una pregunta formulada proporcionaran una única respuesta.

Una vez que se hayan recibido el cupo de respuestas para una pregunta, se inicia un proceso de votación en dónde los usuarios que no sean autores de alguna de las posibles respuestas indicarán que respuesta consideran más adecuada. En la valoración de dos respuestas similares se decantará por la primera o de mejor detalle. El sistema después de 1 semana asignará a la pregunta la respuesta más votada.

El ganador del resultado de la votación recibirá en su wallet la cantidad estipulada por su respuesta. A partir de entonces la pregunta pasará de 'Activa' a 'Consultable' y para conocer su respuesta cada usuario deberá aportar el 50% del coste de la respuesta y la podrá ver durante el periodo de 1h.

Cada usuario dispondrá de una estadística que registrará el porcentaje sobre el número de respuestas correctas e incorrectas realizadas y el número de preguntas formuladas sobre el que se calculará su reputación en el sistema.

Si algún usuario quiere incrementar del valor de recompensa de una respuesta a una pregunta lo podrá hacer y el sistema recibirá el importe en custodia.

El sistema será capaz de realizar una clasificación de la temática de las preguntas finalizadas y su índice de participación por parte de los usuarios o alguna estadística de interés sobre los datos de funcionamiento del sistema.

Cada mes el sistema premiará de manera pseudoaleatoria a uno de entre los 5 usuarios con mayor reputación con el valor del 80% de los beneficios actuales del sistema.

La idea de marketing/venta de este sistema es que es mucho más preciso en la respuesta y devuelve la información que quiere el usuario, que no recibe un aluvión de respuestas sino un conjunto menor, casi sin errores porque están ajustadas a lo que pregunta el usuario, y además, la respuesta final es consensuada por la mayoría, y que por lo tanto, no solo puede decidir con cuál se queda, sino que obtiene cual es el parecer o criterio de la mayoría. Además se pueden generar nuevas preguntas afinando las respuestas con el resto de los usuarios.

Figura 2: Documento INICIAL de especificación técnica del proyecto Esbrinachain.

Como se puede observar en la propuesta de proyecto inicial se añaden muchos complementos de aplicación, y finalmente se decidió concentrar los esfuerzos de desarrollo en las operaciones básicas de:

- Gestión de **Usuarios**: alta y asignación de wallet (personalizado si se facilita una semilla y contraseña) o mediante la billetera digital de Metamask.
- Crear una **pregunta** en el sistema, con su correspondiente recompensa.
- Crear una **respuesta** en el sistema.
- Implementar un sistema de **votación** que determine la respuesta idónea a una pregunta.
- Calcular la reputación de cada usuario o usuarios que sean autor/autores de la respuesta más votada.
- Control de los **estados** de cada pregunta: **abierta** (a respuestas), **votando** (cuando se ha iniciado el periodo de votación), en **consulta** (cuando la pregunta ha sido respondida y votada y los usuarios del sistema quieran conocer la respuesta) y finalmente **anulada** (casos en que no ha recibido respuestas o no ha recibido votos).
- Control de la **duración de los períodos** definidos para cada estado de una pregunta.
- Cálculo del vencedor (uno o más) en la votación de una pregunta
- Cálculo del **Pago**, a cada autor de la respuesta más votada.
- Cálculo de la **reputación** de cada usuario.
- Los incrementos de recompensa a una pregunta para incentivar la obtención de respuestas.
- Concentrar todos los complementos tipo de estadísticas, en un solo caso de uso en que cada usuario puede ver la su clasificación de reputación en comparación con el resto de los usuarios.

Aunque pueda parecer que la propuesta de proyecto se ha reducido, queda compensada con aspectos introducidos durante el desarrollo de la aplicación, que inicialmente no se había apercibido que podían ser necesarios:

- Disponer de un backend a modo de base de datos que de soporte al refresco de datos, que por no ser críticos, pueden no almacenarse en la cadena de bloques de Sepolia, o bien que controle la gestión de los usuarios del sistema. En nuestro caso se ha usado Firestore como base de datos cloud facilitada por Google.
- Sincronización correcta del backend con el estado vigente del sistema en la cadena de bloques. Los posibles errores al intentar transacciones pueden ocasionar discrepancias puesto que hay que considerar la interacción asíncrona con la cadena de bloques.
- Diseño del interfaz web para la aplicación.
- Desarrollo de test unitarios para el aplicativo.
- Permitir la modificación de parámetros de configuración, como el **tiempo de respuesta**, el **tiempo de votación** y el **cupo de respuestas**.

Han quedado para versiones futuras, las siguientes propuestas originales:

- Los premios mensuales a final por mejor reputación en el sistema.
- No se hacen valoraciones sobre las respuestas, sino que se reparte equitativamente a los ganadores de una votación.
- El **periodo de consulta** no se define en 1h, se visualiza la respuesta más votada 15 segundos, con un coste 50% de la recompensa, excepto para el autor de la pregunta que es gratuito.
- La única estadística considerada es la clasificación global de reputación de los usuarios del sistema. Así pues no se han implementado clasificaciones de las temáticas de las preguntas, que bien podrían valorarse en etapas dónde el diseño ya estuviera avanzado y consolidado.
- Los usuarios no pueden modificar el estado de una pregunta a anulada. Solo el sistema define qué preguntas pasan al estado de **anuladas**.

Finalmente la propuesta de proyecto más realista que combina rendimiento final de la aplicación y práctica real de los conocimientos adquiridos durante el curso es la mostrada en la figura 3.

Sistema colaborativo de conocimiento en blockchain.

EsbrinaChain

El sistema es público y de libre acceso sobre la testnet de Ethereum (Sepolia).

Cada participante puede formular la pregunta que desee y fijar un valor en tokens, por el que está dispuesto a pagar una respuesta a modo de recompensa al usuario que la facilite. Opcionalmente se puede fijar un número de respuestas para cerrar la pregunta, sino por defecto serán 15. El valor de la recompensa por la respuesta más votada a una pregunta quedará custodiado en el bote de capital del sistema.

Una pregunta que permanezca sin respuesta durante un periodo superior al periodo de respuesta será establecida como anulada por el sistema. El autor perderá el aporte realizado por una respuesta pasando al bote del sistema.

Los usuarios que estén dispuestos a ganar la recompensa de una pregunta formulada solo podrán proporcionar una única respuesta.

Una vez que se hayan recibido el cupo de respuestas para una pregunta, o bien se haya agotado el periodo de respuesta, se inicia un proceso de votación en dónde los usuarios que no sean autores de la pregunta ni de sus respuestas votarán la respuesta que consideran más adecuada. El sistema después de agotarse el periodo de votación asignará a la pregunta la respuesta más votada. Existe la posibilidad de que el administrador del sistema pueda iniciar o finalizar una votación.

El ganador/es del resultado de la votación recibirá en su 'dirección de cuenta' la cantidad estipulada por su respuesta. A partir de entonces la pregunta pasará de 'votando' a 'consulta' y para conocer su respuesta cada usuario deberá aportar el 50% del coste de la respuesta y la podrá ver durante el periodo de 15 segundos como máximo, excepto el autor de la pregunta que lo puede hacer sin restricciones.

Cada usuario podrá consultar su reputación en el sistema y compararla con la del resto de usuarios.

Todos los usuarios tienen la capacidad para incrementar el valor de la recompensa de una pregunta, siempre que esta no esté en estado de consulta. Cada pregunta permitirá únicamente tres tentativas para incrementar la recompensa.

La innovación de esta propuesta de sistema colaborativo basado en cadena de bloques, (aplicable en su marketing de venta), reside en reforzar la idea de que es mucho más preciso en la respuesta a una pregunta por ser resultado de una valoración consensuada con los usuarios. Además el sistema facilita la información que solicita el usuario de manera más precisa, sin devolver un aluvión de respuestas. El autor de una pregunta puede decidir con cuál se queda independientemente del resultado de la votación, pero alternativamente también recibe cuál es el parecer de la mayoría. Los usuarios están incentivados a dar respuestas con criterio por recibir una recompensa y por mantener su reputación. Además se pueden generar nuevas preguntas considerando las respuestas previas de los usuarios. La innovación técnica consiste en ser compatible con la tecnología blockchain de la red pública Ethereum.

Figura 3: Documento FINAL de especificación técnica del proyecto Esbrinachain.

04. ESTRUCTURA DEL PROYECTO

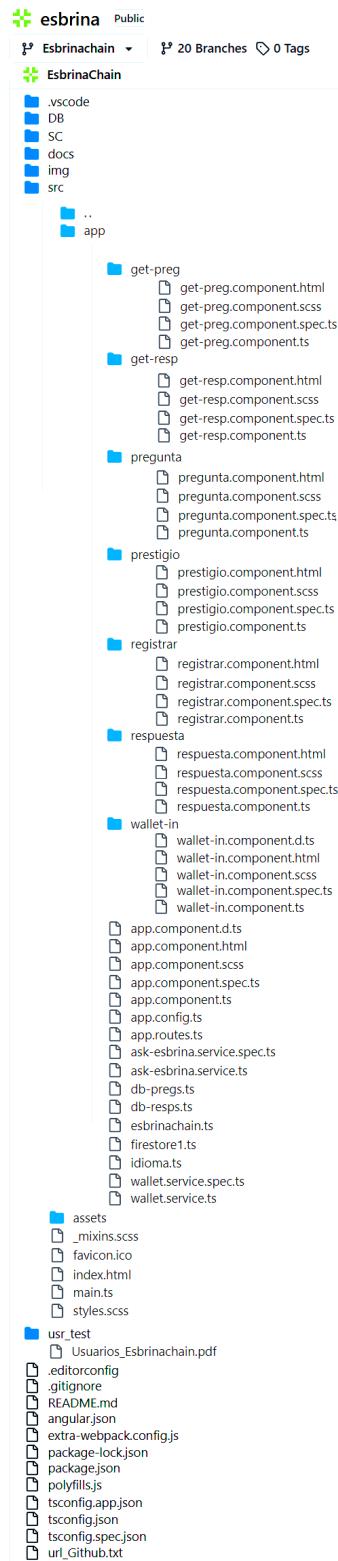
El proyecto **Esbrinachain** está publicado en Github en la dirección <https://github.com/EsbrinaChain/esbrina>. En las siguientes secciones se hará referencia a partes de la estructura de directorios descrita en este apartado y que se puede descargar cuando se necesite poner en marcha el proyecto clonandolo.

En la carpeta **DB** se dispone de una copia de los ficheros de datos usados en el desarrollo y en la configuración de Firestore. En **SC**, una copia de los códigos de los tres Smart Contracts creados para la blockchain Sepolia. En **docs**, se almacena la compilación de la aplicación web publicada en Github usando Angular Framework. En **img**, imágenes usadas en la publicación web o en el fichero README.CMD de Github.

La carpeta **src**, contiene los componentes principales desarrollados en Angular Framework para implementar el sistema Esbrina:

- **get-preg**: formulario de una ventana de diálogo para la formulación de preguntas de los usuarios.
- **get-resp**: formulario de una ventana de diálogo para la generación de respuestas a preguntas.
- **pregunta**: contiene el código relacionado a la gestión de preguntas y desencadenar las posibles respuestas. Es el componente principal del interfaz del sistema Esbrina.
- **respuesta**: componente que define el aspecto y disposición de las respuestas de una pregunta, y se usa también para todo el código de implementación de la votación de respuestas.
- **wallet-in**: implementa todo el código relacionado con la generación de un wallet personalizado para un usuario. Utiliza un servicio **wallet-service.ts**.
- **registrar**: componente que implementa la variante de wallet de **Metamask**.
- **prestigio**: implementa la lista de reputación o prestigio de los usuarios de **Esbrina**.
- **app**: componente principal de la aplicación que contiene al resto de componentes y da forma al interfaz de la aplicación web de **Esbrinachain**.

usr_test: contiene los datos de los perfiles de los usuarios del sistema utilizados en simulaciones, junto con sus wallets y semillas.



Cada componente Angular se divide en 4 ficheros diferenciables por su extensión:

.html: diseño html del componente.
.scss: especificación CSS de estilo.
-spec.ts: configuración adicional.
.ts: código typescript en Angular.

A modo de ejemplo, en el caso del componente **pregunta**:

html:



SCSS:

```

  .tdIndex {
    margin-left: auto;
    margin-right: auto;
    border: 2px solid #darkcyan;
    border-radius: 35px;
    background-color: #darkcyan;
    color: #white;
    text-align: center;
  }

  .tdHidden {
    border: none;
  }

  .tdHiddenBtnResp {
    border: none;
    width: 25px;
    text-align: center;
  }
  
```

.ts:

```

import { PrestigioComponent } from './prestigio/prestigio.component';
//import {fireBaseConfig, providerETH, contract_address} from './firebase';
import {firebaseConfig, providerETH, contract_address} from './firebase';

@Component({
  selector: 'app-pregunta',
  standalone: true,
  imports: [MatButtonModule, ReactiveFormsModule,
    MatSelectModule, MatTableModule, CommonModule,
    GetPregComponent, NotSelectModule, MatFormFieldModule,
    MatListModule, FormModule, ReactiveFormModule,
    templateUrl: './pregunta.component.html',
    styleUrls: ['./pregunta.component.scss']
])
export class PreguntaComponent {
  
```

Figura 4: Estructura de directorios completa del proyecto **Esbrinachain**.

El fichero **Firestore1.ts** almacena la configuración de acceso a Firestore Cloud. Se han utilizado 2 porque en el caso de tener agotarse los límites operacionales permitidos en los perfiles gratuitos, disponer de recambio.

Los ficheros **dbs-preg.ts** y **db-resp.ts** contienen ejemplos de preguntas y respuestas usados durante el desarrollo, simulando los procesos de crear preguntas y respuestas en el sistema.

Finalmente el fichero **idioma.ts** implementa toda la mensajería usada en la aplicación en tres idiomas, castellano, catalán e inglés como una prestación más del sistema de conocimiento colaborativo.

4.1 Smarts Contracts

Una de las características más importantes de la red blockchain es la posibilidad de programar ‘Smarts Contracts’ o contratos inteligentes. En Esbrinachain, nos permiten **formalizar las relaciones contractuales del funcionamiento del sistema colaborativo de conocimiento**, gestionando **quien pregunta, quien responde, quién vota** y como lo hacen, es decir, establecen las reglas de negocio y funcionamiento bajo las cuáles el sistema funciona de manera transparente y de forma automatizada. Ejemplos de reglas son: ‘hay que formular una pregunta siempre ofreciendo una recompensa’, ‘hay un periodo de respuesta’, ‘hay un periodo de votación’, ‘existe un administrador del sistema’, ‘solo se puede responder una vez’ etc. Es el contrato inteligente quién resolverá la votación y asignará beneficios a los autores de la respuesta más votada. El control lo establece de forma automatizada, y al ser immutable su código no permite la manipulación y da seguridad al funcionamiento del sistema.

Utilizar contratos inteligentes proporciona al sistema **Esbrinachain**, seguridad, inmediatez, precisión y transparencia. El lenguaje utilizado para su definición ha sido **Solidity**, y se han desarrollado utilizando el compilador online de REMIX <https://remix.ethereum.org/>.

En **Esbrinachain** existen tres contratos, contenidos en los ficheros:

1. **esb_pregunta.sol**: se ocupa de las reglas de negocio relacionadas con la acción de ‘Preguntar’. Contiene el código del contrato **esbAdmin** (modificador para operaciones reservadas solo al administrador).
2. **esb_respuesta.sol**: se ocupa de las reglas de negocio relacionadas con la acción de ‘Responder’.
3. **esbrinachain.sol**: hereda de los dos anteriores el código y se ocupa de dar soporte al proceso de votación y en general a todos los aspectos del sistema Esbrinachain.

En este apartado se podrá visualizar el código implementado para cada uno. Existe una EIP (Ethereum Improvement Proposal) o propuesta de mejora, la EIP-170 que establece que el límite de tamaño de los Smart Contracts se ha definido en los 24575 bytes. El desarrollo de Esbrinachain está cercano a este límite y por ello cualquier modificación implicaría una reducción de código. Por esto, es importante al desarrollar contratos inteligentes de Solidity, tener en cuenta los límites de tamaño de los contratos cuando se diseñan y no una vez desarrollados.

En el sistema Esbrinachain se ha dividido el código en 3 contratos, y se ha tenido en cuenta alguna de las siguientes recomendaciones:

1. Uso de bibliotecas y proxies.
2. Acortar los mensajes de error.
3. No usar las variables evitables.
4. No pasar estructuras de datos ‘struct’ como parámetros, sino los propios parámetros.
5. Utilizar errores personalizados.
6. Declarar la visibilidad correcta para funciones y variables.
7. Reducir el uso de modificadores.

En la figura 5, se muestra el código fuente del contrato inteligente **preguntar**.

```

1 // SPDX-License-Identifier:GPL-3.0
2 pragma solidity < 0.9;
3
4 contract esbAdmin {
5     address public _admin;
6     constructor() { 73765 gas 49400 gas
7         _admin = msg.sender;
8     }
9     modifier soloAdmin() {
10         require(msg.sender == _admin, unicode"Debe ser ADMIN para realizar esta operación.");
11         _;
12     }
13 }
14
15 contract preguntar is esbAdmin {
16
17     enum estado_preg {abierta, votando, consulta, anulada}
18
19     uint constant num_incr_recompensa = 3;
20     uint public total_preg;
21
22     event PreguntaCreada(uint indexed _id_preg, address indexed _autor);
23     event SubidaRecompensaPreg(uint indexed id_preg);
24     event GanadorPregunta(uint indexed id_preg, address[] indexed autor_s);
25
26     struct Usuario {
27         string nom;
28         string app;
29         string aliase;
30         bool vetado;
31         bool existe;
32         int prestigio;
33     }
34     struct Pregunta {
35         estado_preg estado;
36         uint recompensa;
37         uint creada;
38         uint fecha_votacion;
39         address autor;
40         string enunciado;
41         bool votada;
42     }
43
44     uint public total_usuarios;
45     mapping (uint=>Pregunta) public preguntas;
46     mapping(address => Usuario) public usuarios;
47
48     function creaPregunta(string memory texto, 8 infinite gas
49                           string memory nombre, string memory apellidos,
50                           string memory alia) public payable {
51         require(bytes(texto).length>0,"Enunciado inexistente");
52         require(!usuarios[msg.sender].vetado, "El usuario esta vetado.");
53         require(msg.value > 0,"No se ha enviado recompensa para resolver la pregunta.");
54         if (!usuarios[msg.sender].existe){
55             require(bytes(nombre).length>0 ||
56                     bytes(apellidos).length>0 ||
57                     bytes(alia).length>0," No se ha facilitado ni nombre, ni apellidos, ni alias.");
58             total_usuarios++;
59             usuarios[msg.sender] = Usuario(nombre, apellidos, alia, false, true, 0);
56         }
50         total_preg++;
52         preguntas[total_preg] = Pregunta(estado_preg.abierta, msg.value,
53                                         block.timestamp, 0, msg.sender, texto, false);
54         emit PreguntaCreada(total_preg, msg.sender);
55     }
56
57     // Esta función permite a cualquier usuario incrementar el valor de la recompensa.
58     function subirRecompensaPreg(uint idx_preg) public payable { 8 infinite gas
59         require(total_preg > 0, "No existen preguntas en el sistema.");
60         require(preguntas[idx_preg].estado == estado_preg.abierta, "La pregunta no esta abierta a respuestas.");
61         if (msg.value > 0){
62             preguntas[idx_preg].recompensa += msg.value;
63             emit SubidaRecompensaPreg(idx_preg);
64         }
65     }
66
67     // Retorna el estado de una pregunta: abierta, votándose, consulta o anulada.
68     function estadoPreg(uint idx_preg) public view returns(string memory){ 8 infinite gas
69         string memory estado;
70         if(preguntas[idx_preg].estado == estado_preg.abierta)  estado = "Abierta.";
71         if(preguntas[idx_preg].estado == estado_preg.votando)  estado = "Votando.";
72         if(preguntas[idx_preg].estado == estado_preg.consulta)  estado = "Consulta.";
73         if(preguntas[idx_preg].estado == estado_preg.anulada)  estado = "Anulada.";
74
75         return estado;
76     }
77
78 }
79
80
81
82
83
84
85
86
87
88
89
90

```

Figura 5: Código fuente del contrato inteligente ‘preguntar’ (esb_preguntar.sol).

El contrato **preguntar**, incluye un contrato sencillo llamado **esbAdmin**, definido para establecer el administrador del sistema, (dirección que despliega el contrato), y contiene el modificador **soloAdmin**, que se define para restringir el acceso a determinadas funciones, solo al administrador.

En el contrato **preguntar** se define el conjunto de estados posibles para las preguntas: **abierta**, **votando**, **consulta** y **anulada**. También las estructuras funcionales de una pregunta y de los usuarios del sistema. En la figura 6 se incluye también la estructura Respuesta definida en el contrato ‘responder’:

```
struct Pregunta {
    estado_preg estado;
    uint recompensa;
    uint creada;
    uint fecha_votacion;
    address autor;
    string enunciado;
    bool votada;
}

struct Respuesta {
    uint id_resp;
    uint id_preg;
    uint votos;
    address autor;
    string enunciado;
    bool ganadora;
}

struct Usuario {
    string nom;
    string app;
    string alias;
    bool vetado;
    bool existe;
    int prestigio;
}
```

Figura 6: Estructura de una pregunta, una respuesta y de un usuario

Cada pregunta almacena su **estado**, la **recompensa** asociada por contestarla de manera óptima, la **fecha en que se crea**, la **fecha en que se inicia su votación**, su **autor**, su **enunciado** y si ha sido **votada** o no.

Cada respuesta almacena su **identificador único**, el **identificador único de su pregunta**, el **número de votos** recibidos en la votación de la pregunta, su **autor**, su **enunciado** y si ha resultado **ganadora** en la votación.

Cada usuario está definido por un **nombre**, **apellido**, **alias**, si ha sido **vetado**, y su **prestigio**.

Hay que tener en cuenta que el compilador de Solidity considera un espacio para guardar cada variable, pero en el caso de las estructuras de tipo ‘struct’ **se debe tener en cuenta el orden y agrupar los campos del mismo tipo**. Como se puede observar los tipos ‘uint’, ‘string’ y ‘bool’ están juntos, para optimizar el espacio ocupado. Solidity empaqueta las subvariables del mismo tipo para llenar espacios de 256bits. Por eso, una buena práctica para ahorrar su coste en gas consiste en ajustar los campos del mismo tipo al mínimo posible, y colocarlos juntos en la estructura. Hay que tener en cuenta que toda la funcionalidad del contrato la hereda el contrato **esbrinachain** que es el principal.

En el contrato inteligente ‘**preguntar**’, se definen las siguientes variables:

```
enum estado_preg {abierta, votando, consulta, anulada}
uint constant num_incr_recompensa = 3;
uint public total_preg;
uint public total_usuarios;
mapping (uint=>Pregunta) public preguntas;
mapping(address => Usuario) public usuarios;
```

Figura 7: Variables globales declaradas en el contrato inteligente **preguntar**.

En el contrato **preguntar** se han definido dos mappings:

- preguntas**: su función es almacenar todas la preguntas ‘struct’ incluidas en Esbrinachain por orden de llegada. Para acceder a una pregunta utilizamos su índice **preguntas[i]**. Para acceder a la información de la pregunta, **preguntas[i].campo**. Es de tipo **public** por tanto la podremos consultar fácilmente desde el aplicativo web sin coste.
- usuarios**: igual funcionamiento pero utilizamos como índice la ‘address’ o **dirección pública del usuario**. También es **public** y por ello consultable sin coste.

La variable **total_preg**, mantiene un registro del total de preguntas existentes en el sistema de conocimiento y un comportamiento parecido con los usuarios lo realiza la variable **total_usuarios**. El contrato dispone de una variable para definir cuántos **incrementos de recompensa** están permitidos en una pregunta.

En la figura 8, podemos observar las tres funciones definidas en el contrato **preguntar**:

```
function creaPregunta(string memory texto,    payable
                      string memory nombre, string memory apellidos,
                      string memory alias) public payable
function subirRecompensaPreg(uint idx_preg) public payable
function estadoPreg(uint idx_preg) public view returns(string memory)
```

Figura 8: Declaración de las funciones y parámetros del contrato **preguntar**.

Para incluir una pregunta en el contrato, sólo debemos usar la función **creaPregunta**, y pasarle como parámetro el enunciado y tres campos de tipo string, nombre, apellido y alias del autor la pregunta. Podemos dejar alguno vacío, pero no los tres. Es importante identificar que esta función es **public** y **payable**, lo cual significa, que la podemos ejecutar externamente y que al hacerlo, se debe pasar una cantidad o valor en la transacción que servirá como **recompensa**. La recompensa se almacenará en el balance del contrato **Esbrinachain**, que en el sistema representa el bote acumulado de los activos generados.

Para subir la recompensa de una pregunta, podemos ejecutar la función **subirRecompensa**, que también es de tipo **payable**. El incremento de recompensa solo se puede realizar 3 veces.

La última función es de tipo '**view**', lo que significa que no consume gas y puede ejecutarse de forma gratuita simplemente con una llamada '**call**'. Con solo indicar el índice de la pregunta nos retorna el **estado actual** de la misma en el sistema.

La función **estadoPreg**, nos será muy útil en la programación de la aplicación web, ya que los cambios de estado modifican las pantallas visualizadas. Es muy importante poder consultar el estado de las variables y saber que acciones realiza un contrato internamente para que la aplicación de interfaz web pueda estar sincronizada. Para este cometido en los contratos inteligentes se definen los eventos. Cuando el contrato **preguntar** inserta una pregunta en el mapping de preguntas, emite un evento **PreguntaCreada**, cuando se hace efectivo el incremento de una recompensa se emite el evento **SubidaRecompensaPreg**, y cuando una pregunta tiene un vencedor, por ser el autor de la respuesta más votada, se emite el evento **GanadorPregunta**. En la figura 9 podemos observar las declaraciones de cada evento y la información que registran para que el aplicativo web disponga de esa información. Es la red blockchain la que se encarga de mantener asociada al contrato y accesible toda esta información de manera indexada, si así, lo especificamos en las declaraciones.

```
event PreguntaCreada(uint indexed _id_preg, address indexed _autor);
event SubidaRecompensaPreg(uint indexed id_preg);
event GanadorPregunta(uint indexed id_preg, address[] indexed autor_s);
```

Figura 9: Declaración de los eventos del contrato inteligente **preguntar**.

El segundo contrato inteligente heredado por el contrato principal esbrinachain, es el contrato inteligente **responder**.

En el contrato **responder** se gestiona el proceso de responder a una pregunta. Existe un **tiempo de respuesta** predefinido a **1 semana**, que puede ser configurado por el administrador del sistema. Otro valor predefinido es el **cupo de respuestas** posibles fijado en **15**. El tiempo de respuesta finaliza cuando se responde a una pregunta y el contrato inteligente comprueba que el periodo de respuesta se ha agotado. Entonces **el contrato, no tiene en cuenta la respuesta** y cambia el estado de la pregunta de estar '**abierta**' a respuestas, a '**votando**'. En la figura 10, se puede observar el código fuente del contrato **responder** completo.

```

1 // SPDX-License-Identifier:GPL-3.0
2 pragma solidity <0.9;
3
4 import "./esb_preguntas.sol";
5
6 contract responder is preguntar {
7     uint public tiempo_respuesta = 1 weeks;
8     uint public cupo_respuestas = 15;
9     uint public total_resp;
10
11    struct Respuesta {
12        uint id_resp;
13        uint id_preg;
14        uint votos;
15        address autor;
16        string enunciado;
17        bool ganadora;
18    }
19
20    mapping(uint => mapping(uint => Respuesta)) public preg_resp;
21    mapping(uint => uint) public pregNumResp;
22
23    event InicioVotacion(uint indexed id_preg);
24    event RespuestaCreada(
25        uint indexed id_preg,
26        uint indexed id_resp,
27        address indexed autor
28    );
29    event RespuestaFueraDeTiempo(
30        uint indexed idx_preg,
31        uint indexed id_resp,
32        address indexed autor
33    );
34    event RespuestaFueraDeCupo(
35        uint indexed idx_preg,
36        uint indexed cupo_max,
37        address indexed autor
38    );
39    event PreguntaAnulada(uint indexed id_preg, string motivo);
40
41 // Mira si un usuario a respondido ya a una pregunta
42 function calcAdrRespuestasAPregunta( ) payable {
43     uint id_preg,
44     address _adr
45 } public view returns (bool _existe) {
46     _existe = false;
47     for (uint i = 1; i <= pregNumResp[id_preg]; i++) {
48         if (preg_resp[id_preg][i].autor == _adr) _existe = true;
49     }
50
51 // devuelve el conjunto de respuestas a una pregunta
52 function calcRespAPreg(uint id_preg) public view returns (uint[] memory) {
53     uint[] memory _resp = new uint[](pregNumResp[id_preg]);
54     for (uint i = 1; i <= pregNumResp[id_preg]; i++) {
55         _resp[i - 1] = preg_resp[id_preg][i].id_resp;
56     }
57     return _resp;
58 }
59
60 function adrRespondeVence(uint idx_preg, address _adr) internal view returns (bool, bool) {
61     if (preguntas[idx_preg].estado == estado_preg.consulta) {
62         uint[] memory rsp = calcRespAPreg(idx_preg);
63         for (uint i = 1; i < rsp.length + 1; i++) {
64             if (
65                 preg_resp[idx_preg][i].ganadora &&
66                 preg_resp[idx_preg][i].autor == _adr
67             ) {
68                 return (true, true);
69             }
70             if (
71                 !preg_resp[idx_preg][i].ganadora &&
72                 preg_resp[idx_preg][i].autor == _adr
73             ) {
74                 return (true, false);
75             }
76         }
77     }
78     return (false, false);
79 }
80 // Para consultar la respuesta mas votada de una pregunta se debe enviar el 50% de su recompensa
81 // excepto que sea el autor de la pregunta que podrá consultarla sin coste.
82 function consultaRespuestaVotada(uint idx_preg) public payable returns(uint) {
83     string memory estado = estadoPreg(idx_preg);
84     require(preguntas[idx_preg].estado == estado_preg.consulta,
85     string(abi.encodePacked("La pregunta debe ser consultable, pero su estado es ",estado)));
86     uint precio = preguntas[idx_preg].recompensa / 2;
87     if (msg.sender != preguntas[idx_preg].autor){
88         require(msg.value >= precio,"Se debe enviar el coste de la consulta para ver la respuesta votada");
89     }
90     return RespMasVotada(idx_preg);
91 }
92
93 // Devuelve un array con la opción más votada o opciones en caso de empate.
94 function RespMasVotada(uint idx_preg) internal view returns(uint _votada) {
95     if(preguntas[idx_preg].estado == estado_preg.consulta){
96         uint[] memory rsp = calcRespAPreg(idx_preg);
97         for (uint i=1; i < rsp.length+1; i++){
98             if(preg_resp[idx_preg][i].ganadora){
99                 _votada = i;
100            }
101        }
102    }
103    return _votada;
104 }

```

```

106     function creaRespuesta(    ) infinite gas
107         uint id_preg,
108         string memory texto,
109         string memory nombre,
110         string memory apellidos,
111         string memory alia
112     } public {
113         require(total_preg > 0, "No existen preguntas en el sistema.");
114         require(
115             preguntas[id_preg].creada > 0,
116             "No existe una pregunta con Fecha de creacion 0."
117         );
118         require(
119             preguntas[id_preg].estado == estado_preg.abierta,
120             "La pregunta ya no puede recibir respuestas porque NO esta abierta."
121         );
122         uint num_resp_act = calcRespAPreg(id_preg).length;
123         // Pasado el periodo de respuestas, si no se han proporcionado se anula la pregunta.
124         if (
125             num_resp_act == 0 &&
126             (block.timestamp - preguntas[id_preg].creada) >= tiempo_respuesta
127         ) {
128             preguntas[id_preg].estado = estado_preg.anulada;
129             emit PreguntaAnulada(id_preg, "Anulada por falta de respuestas.");
130             return;
131         }
132         if (block.timestamp - preguntas[id_preg].creada >= tiempo_respuesta) {
133             preguntas[id_preg].estado = estado_preg.votando;
134             preguntas[id_preg].fecha_votacion = block.timestamp;
135             emit RespuestaFueraDeTiempo(
136                 id_preg,
137                 (block.timestamp - preguntas[id_preg].creada) % 86400,
138                 preguntas[id_preg].autor
139             );
140             emit InicioVotacion(id_preg);
141         } else if (num_resp_act > cupo_resuestas) {
142             preguntas[id_preg].estado = estado_preg.votando;
143             preguntas[id_preg].fecha_votacion = block.timestamp;
144             emit RespuestaFueraDeCupo(
145                 id_preg,
146                 cupo_resuestas,
147                 preguntas[id_preg].autor
148             );
149             emit InicioVotacion(id_preg);
150         } else {
151             // El usuario no debe estar vetado
152             require(!usuarios[msg.sender].vetado, "El usuario esta vetado.");
153             // El autor de la respuesta no puede ser el autor de la pregunta
154             require(
155                 preguntas[id_preg].autor != msg.sender,
156                 "El usuario autor no puede responder a sus preguntas."
157             );
158             // El usuario solo puede responder 1 vez a una pregunta
159             require(
160                 !calcAadrRespuestasAPregunta(id_preg, msg.sender),
161                 "No se puede responder mas de 1 vez."
162             );
163             // Si no es usuario tenemos que recibir un nombre o un apellido o un alias
164             if (!usuarios[msg.sender].existe) {
165                 require(
166                     keccak256(abi.encodePacked(nombre)) !=
167                     keccak256(abi.encodePacked("")) ||
168                     keccak256(abi.encodePacked(apellidos)) !=
169                     keccak256(abi.encodePacked("")) ||
170                     keccak256(abi.encodePacked(alia)) !=
171                     keccak256(abi.encodePacked("")),
172                     "Hace falta indicar nombre o apellido o alias."
173                 );
174                 total_usuarios++;
175                 usuarios[msg.sender] = Usuario(
176                     nombre,
177                     apellidos,
178                     alia,
179                     false,
180                     true,
181                     0
182                 );
183             }
184             total_resp++;
185             pregNumResp[id_preg]++;
186             preg_resp[id_preg][pregNumResp[id_preg]] = Respuesta(
187                 pregNumResp[id_preg],
188                 id_preg,
189                 0,
190                 msg.sender,
191                 texto,
192                 false
193             );
194             emit RespuestaCreada(id_preg, pregNumResp[id_preg], msg.sender);
195         }
196     }
197 }
```

Figura 10: Código fuente del contrato inteligente responder.

El contrato **responder** incluye las siguientes variables mostradas en la figura 11:

```
uint public tiempo_respuesta = 1 weeks;
uint public cupo_resuestas = 15;
uint public total_resp;
mapping(uint=>mapping(uint=>Respuesta)) public preg_resp;
mapping(uint=>uint) public pregNumResp;
```

Figura 11: Variables definidas en el contrato inteligente **responder**.

El **tiempo de respuesta** se define a 1 semana, el **cupo_resuestas** a 15 y existe una variable que registra el número de respuesta incluidas en el contrato. Todas las variables son **public**, lo que permitirá consultarlas desde el aplicativo web sin coste de gas.

En este contrato se han definido dos mappings:

1. **pregNumResp**: registra el número de respuestas que tiene cada pregunta.
2. **preg_resp**: mapping que anida un segundo mapping. El primero indexa las preguntas por un índice numérico y éste da acceso a la estructura de sus respuestas indexada también numéricamente por el orden de llegada. Así pues, **preg_resp[i]** da acceso al mapping de respuestas de la pregunta i, y **preg_resp[i][j]** a la respuesta jésima de la pregunta **preg_resp[i]**. Si decidimos acceder al contenido de campos de la estructura ‘struct’ que define una respuesta (figura 6), utilizaremos la notación **preg_resp[i][j].campo**.

Los eventos definidos en el contrato **responder** se muestran en la figura 12:

```
event InicioVotacion(uint indexed id_preg);
event RespuestaCreada(uint indexed id_preg, uint indexed id_resp, address indexed autor);
event RespuestaFueraDeTiempo(uint indexed idx_preg, uint indexed id_resp, address indexed autor);
event RespuestaFueraDeCupo(uint indexed idx_preg, uint indexed cupo_max, address indexed autor);
event PreguntaAnulada(uint indexed id_preg, string motivo);
```

Figura 12: Eventos definidos en el contrato inteligente **responder**.

El contrato emite un evento **RespuestaCreada** cuando una respuesta es incluida en el contrato, o lo que es lo mismo registrada en el mapping **preg_resp**, indicando un índice para la pregunta que responde, un índice de llegada de la respuesta dentro del conjunto de respuestas de la pregunta y una dirección del autor de la respuesta.

Cuando una respuesta está **fueras de tiempo**, el contrato debe pasar la pregunta a estado ‘**votando**’ y emite un evento **RespuestaFueraDeTiempo**. Si la respuesta excede el límite de respuestas definido en el momento actual del contrato, entonces se emite un evento **RespuestaFueraDeCupo**.

En el contrato se define un evento **InicioVotacion**. Solo se usa si el administrador decide iniciar el proceso de votación para una pregunta. Esencialmente esta acción no es necesaria pero a efectos de realizar la demostración del proyecto permite agilizar la presentación cambiando el estado de la pregunta y finalizando el proceso de respuestas. El evento **PreguntaAnulada** lo emite el contrato cuando o bien se agota el periodo de respuestas y no se ha respondido a la pregunta, o existen respuestas pero durante el periodo de votación nadie las ha votado.

Es importante observar que el contrato **responder** hereda la funcionalidad del contrato **preguntar** y toda su funcionalidad está disponible mediante la instrucción **import "./esb_preguntas.sol"**. La herencia se consigue con la instrucción de definición del contrato: ‘**contract responder is preguntar**’.

A continuación mostramos en la figura 13, las declaraciones de las funciones definidas en el contrato:

```

function calcArdRespuestasAPreguntas(uint id_preg, address _adr) public view returns (bool _existe)
function calcRespAPreg(uint id_preg) public view returns (uint[] memory)
function adrRespondeVence(uint idx_preg, address _adr) internal view returns (bool, bool)
function creaRespuesta(uint id_preg, string memory texto, string memory nombre,
                      string memory apellidos, string memory alias) public
function consultaRespuestaVotada(uint idx_preg) public payable returns(uint)
function RespMasVotada(uint idx_preg) internal view returns(uint _votada)

```

Figura 13: Funciones definidas en el contrato inteligente **responder**.

La función **calcArdRespuestasAPreguntas** recibe como parámetros el **id** de la pregunta a considerar y la **dirección** de un usuario y devuelve **true**, si el usuario ha respondido a esa pregunta. Como se puede observar es **internal** lo que significa que no es accesible desde fuera del contrato, y es de uso interno. Cuando se calcula el prestigio de un usuario, se realiza un conteo de las preguntas a las que un usuario ha respondido o no y si lo ha hecho con acierto, para lo cual se usa en parte esta función, y la función **adrRespondeVence** que realiza el conteo y cálculo del prestigio del usuario puntuándolo entre 0 y 10. La complejidad de la fórmula del prestigio queda modificable y para futuras versiones hacerla más precisa. En esta primera versión, simplemente se califica teniendo en cuenta el porcentaje de aciertos.

La función **calcRespAPreg** es **public** y nos retorna el número de respuestas de una pregunta, y es muy útil al desarrollar el aplicativo porque en Solidity se pueden devolver ‘arrays’ enteros pero no si contienen estructuras ‘**struct**’ como elementos. Por ello saber el número de respuestas nos permitirá recuperarlas, una a una, conociendo el límite del ‘**for**’ a implementar.

La función **creaRespuesta** nos permite registrar respuestas indicando a qué pregunta ‘**id**’ corresponden, el texto o enunciado de la respuesta y los datos string (nombre, apellido, alias) asociados a su autor.

La función **creaRespuesta** la analizamos en detalle, línea a línea, por ser de las más importantes:

El contrato inteligente mantiene el número de respuesta en cada caso de cada pregunta, y solo necesita recibir el ‘**id**’ de la pregunta a la que responde, el **enunciado texto** de la respuesta y los **datos del autor**, que puede ser uno de tres posibles (nombre, apellido, alias). Para poder responder deben cumplirse una serie de condiciones. si no se cumplen todas las instrucciones ‘require’, se anula la acción de responder:

1. Línea 87: Si no existen preguntas : **total_preg >0**
2. Línea 92: La pregunta no está en estado 0 o ‘**abierta**’ a respuestas

Línea 96: Se calcula cuántas respuestas tiene actualmente la pregunta.

Líneas 98-105: Si el tiempo de respuesta para la pregunta ha vencido, si no ha recibido respuestas se anulará, pasa a estado=3 o ‘**anulada**’, emitiéndose un evento **PreguntaAnulada**.

Líneas 106-114: Si el tiempo de respuesta ha vencido pero la pregunta ha recibido respuestas, entonces se debe ignorar esa respuesta y pasar la pregunta ha estado ‘**votando**’ (línea 107), emitiendo un evento **RespuestaFueraDeTiempo** (línea 109) y un evento **InicioVotacion** (línea 114). Es en este momento cuándo una pregunta actualiza su fecha de votación al tiempo actual (línea 108) y empieza a contar el tiempo de votación.

Líneas 115-123: En el caso de que el cupo de respuestas se haya sobrepasado, entonces igualmente se pasa al estado de ‘**votando**’ (línea 116), se actualiza la fecha de votación (línea 117) y se emite un evento **RespuestaFueraDeCupo** (línea 118-122), y un evento **InicioVotacion** (línea 123).

Líneas 124-157: Se controla que el usuario no haya sido vetado por el administrador. Que es una opción que no está programada pero si preparada para futuras versiones. Se controla que el usuario que responde no sea el autor de la pregunta (línea 129), y que no intente responder más de una vez (línea 134).

El contrato requiere que se proporcione alguno de los campos de identificación de usuario (nombre, apellidos, alias), pues en caso de no existir en el sistema se le da de alta incrementando la variable **total_usuarios**.

Línea 158: Se incrementa la variable que controla el número de respuestas.

Línea 159: Se incrementa el número de respuestas de la pregunta respondida.

Líneas 160-167: Se actualiza el mapping de preguntas-respuestas insertando la estructura completa de la respuesta en el índice de la pregunta.

Línea 168: Se emite el evento **RespuestaCreada** indicando la pregunta, el índice de orden de la respuesta y su autor.

El contrato **responder** hereda el contrato **preguntar**. Mientras que el contrato principal **esbrinachain**, hereda el contrato **responder** e indirectamente el contrato **preguntar**, por lo que dispone de toda la funcionalidad. La división en tres contratos nos proporciona una ordenación por temática de la funcionalidad del proyecto.

El contrato **esbrinachain** enlaza los procesos de crear preguntas y asignar respuestas a las preguntas con el proceso de votación de cada pregunta, la resolución de la votación y la determinación de uno o más ganadores (en caso de mismo número máximo de votos de varias respuestas a la pregunta). Este desenlace genera dos procesos finales:

- Pago de la recompensa por realizar la respuesta más votada a una pregunta.
- Cálculo y asignación de la reputación o prestigio del usuario en el sistema **Esbrinachain**.

En la figura 16, se puede revisar el código fuente completo del contrato inteligente **esbrinachain**.

El contrato **esbrinachain** incluye las siguientes variables mostradas en la figura 14:

```
uint public tiempo_votacion = 1 weeks;
// address usuario, mapeo id_preg_votada, id_resp_votada
mapping(address => mapping(uint => uint)) public votaciones;
address[] private usr_premio_mensual;
```

Figura 14: Variables declaradas en el contrato inteligente **esbrinachain**.

El tiempo de votación, por defecto se fija a 1 semana, aunque el administrador lo puede modificar.

Se define un ‘mapping’ para almacenar las votaciones. Dada una dirección de un usuario podemos disponer de todas la preguntas a las que ha votado: **votaciones[address][i]**, y para saber qué respuesta ha votado en cada pregunta **votaciones[address][i][j]**, en dónde ‘j’ indica el índice de la respuesta votada. Como solo se puede votar por una respuesta, no deberá existir nunca dos valores iguales en **votaciones[address][i]**.

La variable **usr_premio_mensual**, queda definida para futuras versiones.

Las funciones definidas en el contrato, las podemos ver listadas en la figura 15:

```
function valoresVariables() public view returns (uint, uint, uint, uint, uint, uint)
function votarRespuesta(uint idx_preg, uint idx_resp) public
function resolucionVotacion(uint idx_preg) internal
function resolucionPagosBeneficios(uint idx_preg,uint num_resp,uint conPremio) internal
function iniciaVotacion(uint idx_preg) public soloAdmin
function finalizaVotacion(uint idx_preg) public soloAdmin
function admCfgTiempoRespuesta(uint num,uint t,string memory units) external soloAdmin
function calcPrestigio(address _usr) internal returns (int)
function calcPrestigioPreg(int venceUsr,int noVenceUsr) internal pure returns (int)
function consUsrPrestigio(address _usr) external payable returns (int)

// 1 Hour 3600 Sec 1 Day 86400 Sec
// 1 Week 604800 Sec 1 Month 2629743 Sec
```

Figura 15: Funciones declaradas en el contrato inteligente **esbrinachain**.

La función **valoresVariables** se utiliza para obtener los valores de variables predefinidas, como el **tiempo_respuesta**, **tiempo_votacion**, **cupo_resuestas**, **total_pregs**, **total_resps** y **num_incr_recompensa**. La función **votarRespuesta**, la describimos en detalle por ser de las más importantes:

Acepta como parámetros el índice de la pregunta que se está votando **idx_preg**, y el índice de la respuesta que se quiere votar **idx_resp**, es public y no retorna valores.

Líneas 32-39: Para poder votar una respuesta el usuario no debe ser autor de la pregunta, ni haber proporcionado ninguna respuesta. Para hacer efectivo el voto se debe hacer dentro del periodo de votación.

Líneas 36-40: Si el periodo de votación ha vencido y la pregunta no está en estado **1 (votando)**, entonces se informa al usuario de que '*No se puede votar*', y se genera error, para abandonar la función. En el caso de estar en estado votando, como el periodo de votación ha vencido se actualiza el estado de la pregunta a estado **2 (consulta)**.

Líneas 41-44: Si se ha cambiado el estado a 2 y la pregunta tiene la propiedad '**votada**' a **true**, significa que se ha estado votando y el periodo ha finalizado y se debe **resolver la votación** además de emitir el evento **FinalVotacion**.

Líneas 45-51: Si la propiedad '**votada**' no está a **true**, significa que durante todo el periodo de votación nadie ha votado, y por ello se actualiza la pregunta a estado **3 (anulada)**, y se emite un evento **PreguntaAnulada**.

Línea 52: Implementa el caso en que el periodo de votación no ha vencido.

Líneas 53-57: Podemos votar siempre que el estado de la pregunta sea **1 (votando)** y en caso de no ser así se informa al usuario y se genera error para abandonar la función.

Líneas 58-59: Se considera el caso de que el usuario no esté vetado, si así fuera también se informa y finaliza la ejecución. En la actual versión los usuarios no se pueden vetar y es una estructura para futuras versiones.

Líneas 60-64: El usuario autor de una pregunta no puede votarla, si se da este caso se informa y se finaliza la ejecución de la función.

Líneas 65-69: Se considera el caso de que **solo se vote 1 sola vez**. Se comprueba que no existe una entrada en el 'mapping' de **votaciones**, y si existe se informa y se finaliza la ejecución de la función.

Líneas 70-74: Se valida que el **idx_preg** facilitado se corresponda con el índice de una pregunta existente en el sistema.

Líneas 75-78: Se comprueba que el **idx_resp** facilitado se corresponda con uno de los posibles índices de las respuestas a la pregunta que se está votando.

Líneas 79-81: Una vez realizadas todas las comprobaciones se materializa el voto incrementando en una unidad la propiedad '**votos**' de la respuesta con índice **idx_resp** de la pregunta con índice **idx_preg**. Se reafirma el estado true de la propiedad '**votada**' de la pregunta y se registra la votación en el 'mapping' de **votaciones** para el usuario **sender** (de la transacción que ejecuta la función **votarRespuesta**), asignando al elemento **votaciones[sender][idx_preg]** el valor **idx_resp**).

Hemos visto que cuando el contrato inteligente determina que el periodo de votación ha finalizado se resuelve la votación. Esta acción la realiza la función **resolucionVotación**, que acepta como parámetro el índice de la pregunta. Veamos el detalle de esta función:

Línea 86: Se calcula un vector (**array**) con los índices de las respuestas de la pregunta con índice **idx_preg** identificando como '**rsp**'.

Líneas 87-88: Si la pregunta no tiene respuestas se actualiza su estado a **3 (anulada)**, y se finaliza la ejecución de la función.

Líneas 89-105: Se procede a resolver la votación. Para resolver la votación se utiliza un bucle '**for**' para encontrar el **valor del máximo** de la propiedad '**votos**' de entre todas las respuesta, y luego

de nuevo otro bucle ‘**for**’ para actualizar todas aquellas respuestas con número de votos igual al valor máximo calculado. La propiedad que nos interesa actualizar es **ganadora** a true, y cada vez que lo hagamos lo anotamos, incrementando el valor de la variable ‘**j**’ a modo de contador.

La manera de reseguir la lista y actualizar cada respuesta es utilizando el ‘mapping’ **preg_resp** que enlaza preguntas y respuestas: **preg_resp[idx_preg]** identifica a la pregunta y para obtener los datos de cada una de sus respuestas utilizamos **preg_resp[idx_preg][i]** dándole valores a **i** para cada uno de los posibles índices de respuestas calculados y disponibles en el vector **rsp**. Otro objetivo de esta función consiste en facilitar la dimensión del vector de índice de respuestas ‘**rsp**’ (cuántas respuestas tiene la pregunta) y el número de respuestas ganadoras (almacenado en la variable ‘**j**’) a la función **resolucionPagosBeneficios**.

Describimos con detalle esta función a continuación, por ser de las importantes:

La función **resolucionPagosBeneficios** recibe tres parámetros, los comentados en el párrafo anterior y el índice de la pregunta para la cual se realiza el pago de su recompensa, al autor de la respuesta más votada.

- Línea 113: Se realiza el cálculo de la cantidad a pagar. Si hay más de un ganador se divide la recompensa por el número de ganadores.
- Líneas 114-119 Se recorre el mapping **preg_resp**, y para cada respuesta ganadora se realiza el pago al autor de la respuesta mediante la función de pago **transfer** de **payable**.
- Línea 117: Se calcula la reputación del autor de la respuesta más votada utilizando la función **calcPrestigio**.

La función **calcPrestigio** recibe como parámetro, la dirección del usuario que queremos definir su reputación y funciona juntamente con la función **calcPrestigioPreg**. La intención en esta primera versión del sistema no ha sido implementar una función de reputación compleja. Pero si definir una función **calcPrestigioPreg** en dónde hacer la implementación y llamarla des de **calcPrestigio**. En esta primera versión se le asigna al usuario una calificación entre 0 y 10 según su promedio de aciertos a preguntas respondidas. Sirva de ejemplo que si un **usuario solo contesta 1 pregunta y es la más votada**, su prestigio será de **10**, y si por ejemplo responde tres preguntas y solo resulta ser el más votado en una, recibe un prestigio o reputación con valor 3. El sistema valora que el usuario no responda porque sí a las preguntas, siendo penalizado su prestigio, cuando comete errores.

En el contrato inteligente **esbrinachain** se han definido **dos funciones** ‘sólo ejecutables por el administrador’ y que por ello llevan el modificador **soloAdmin**. Son **iniciaVotacion** y **finalizaVotacion**. Han sido utilizadas durante el desarrollo del aplicativo y pensado para la demo presentación del proyecto, para poder cambiar el estado de las preguntas a **1 (votando)** sin esperar a agotar los tiempos de respuesta. Como podrían ser de utilidad para el administrador en el mantenimiento del sistema, finalmente se han mantenido.

En el caso de la función **iniciaVotacion**, se cambia el **estado de la pregunta** con el índice pasado como parámetro y se **modifica la fecha de votación** de la pregunta a un minuto antes del tiempo de la transacción. En el caso de la función **finalizaVotacion**, se cambia el **estado** de la pregunta y se **resuelve la votación**.

Otra función que se mantiene derivada del desarrollo y que está en el mismo caso de las dos anteriores es **admCfgTiempoRespuesta**. Inicialmente se creó para modificar el tiempo de respuesta de una pregunta, y finalmente se amplió a tres variables: **tiempo_respuesta**, **tiempo_votacion** y **cupo_respuestas**.

Solo hay que tener en cuenta que el **tiempo_respuesta** va asociado a la fecha de creación de una pregunta (propiedad **creada** de una pregunta), y el **tiempo_votacion** al periodo definido cuando se finaliza el tiempo de respuesta. En la figura 16, se muestra el código completo del contrato inteligente **Esbrinachain**.

```

1 // SPDX-License-Identifier:GPL-3.0
2 pragma solidity <0.9;
3
4 import "./esb_respuestas.sol";
5
6 contract esbrinachain is responder {
7     uint public tiempo_votacion = 1 weeks;
8
9     event FinalVotacion(uint indexed id_preg, address indexed autor, uint indexed respGanadora);
10
11    // address usuario, mapeo id_preg_votada, id_resp_votada
12    mapping(address => mapping(uint => uint)) public votaciones;
13    address[] private usr_premio_mensual;
14
15    // Devuelve valores variables globales
16    function valoresVariables()   payable infinite gas
17    public
18    view
19    returns (uint, uint, uint, uint, uint, uint)
20    {
21        return (
22            tiempo_respuesta,
23            tiempo_votacion,
24            cupo_respuestas,
25            total_preg,
26            total_resp,
27            num_incr_recompensa
28        );
29    }
30
31    function votarRespuesta(uint idx_preg, uint idx_resp) public {   payable infinite gas
32        if (
33            (block.timestamp - preguntas[idx_preg].fecha_votacion) >=
34            tiempo_votacion
35        ) {
36            require(
37                preguntas[idx_preg].estado == estado_preg.votando,
38                "La pregunta no se puede VOTAR."
39            );
40            preguntas[idx_preg].estado = estado_preg.consulta;
41            if (preguntas[idx_preg].votada) {
42                // Resolucion votación
43                resolucionVotacion(idx_preg);
44                emit FinalVotacion(idx_preg, preguntas[idx_preg].autor, RespMasVotada(idx_preg));
45            } else {
46                preguntas[idx_preg].estado = estado_preg.anulada;
47                emit PreguntaAnulada(
48                    idx_preg,
49                    "Pregunta anulada por falta de votaciones"
50                );
51            }
52        } else {
53            // El estado de la pregunta tiene que estar en 'Votando'
54            require(
55                preguntas[idx_preg].estado == estado_preg.votando,
56                "La pregunta no se puede VOTAR."
57            );
58            // El usuario no debe estar vetado
59            require(!usuarios[msg.sender].vetado, "El usuario esta vetado.");
60            // El autor de la pregunta no puede votarla
61            require(
62                preguntas[idx_preg].autor != msg.sender,
63                "El autor no puede VOTAR sus preguntas."
64            );
65            // como solo puede votar 1 vez no debe existir un elemento dado una address y un id_preg
66            require(
67                votaciones[msg.sender][idx_preg] == 0,
68                "Un usuario solo puede votar 1 vez una pregunta."
69            );
70            uint[] memory rsp = calcRespAPreg(idx_preg);
71            require(
72                preguntas[idx_preg].creada != 0,
73                "No existe una pregunta con este ID."
74            );
75            require(
76                idx_resp > 0 && idx_resp <= rsp[rsp.length - 1],
77                "No existe ese ID de respuesta para esa pregunta."
78            );
79            preg_resp[idx_preg][idx_resp].votos++;
80            preguntas[idx_preg].votada = true;
81            votaciones[msg.sender][idx_preg] = idx_resp;
82        }
83    }

```

```

84
85     function resolucionVotacion(uint idx_preg) internal {    █ infinite gas
86         uint[] memory rsp = calcRespAPreg(idx_preg);
87         if (rsp.length == 0) {
88             preguntas[idx_preg].estado = estado_preg.anulada;
89         } else {
90             uint max=0;
91             for (uint i = 1; i <= rsp.length; i++) {
92                 if (preg_resp[idx_preg][i].votos > max) {
93                     max = preg_resp[idx_preg][i].votos;
94                 }
95             }
96             uint j=0;
97             for (uint i = 1; i <= rsp.length; i++) {
98                 if(preg_resp[idx_preg][i].votos == max){
99                     preg_resp[idx_preg][i].ganadora=true;
100                j++;
101            }
102        }
103        resolucionPagosBeneficios(idx_preg, rsp.length, j);
104    }
105}
106 // Pago de la recompensa al ganador/ganadores con la respuesta más votada.
107 // En caso de varios ganadores el premio se divide entre todos a partes iguales.
108 function resolucionPagosBeneficios(    █ infinite gas
109     uint idx_preg,
110     uint num_resp,
111     uint conPremio
112 ) internal {
113     uint pago = preguntas[idx_preg].recompensa / conPremio;
114     for (uint i = 1; i <= num_resp; i++) {
115         if (preg_resp[idx_preg][i].ganadora) {
116             payable(preg_resp[idx_preg][i].autor).transfer(pago);
117             calcPrestigio(preg_resp[idx_preg][i].autor);
118         }
119     }
120 }
121 // El administrador puede iniciar la votación de cualquier pregunta.
122 function iniciaVotacion(uint idx_preg) public soloAdmin {    █ infinite gas
123     require(
124         preguntas[idx_preg].estado == estado_preg.abierta,
125         "La pregunta no esta abierta."
126     );
127     require(
128         calcRespAPreg(idx_preg).length > 0,
129         "No se han dado respuestas a esta pregunta."
130     );
131     preguntas[idx_preg].estado = estado_preg.votando;
132     preguntas[idx_preg].fecha_votacion = block.timestamp - 1 minutes;
133 }
134
135 // Función utilizada por el administrador para forzar la finalización de la votación
136 // Pero pasado el intervalo de votación el contrato finalizará la votación de una pregunta,
137 // cuando se intente votar fuera del periodo de votación.
138 function finalizaVotacion(uint idx_preg) public soloAdmin {    █ infinite gas
139     require(
140         preguntas[idx_preg].estado == estado_preg.votando,
141         "La pregunta no se esta votando."
142     );
143     preguntas[idx_preg].estado = estado_preg.consulta;
144     //preguntas[idx_preg].fecha_votacion = block.timestamp - 2 weeks;
145     resolucionVotacion(idx_preg);
146 }
147

```

```

148     // Función que permite al admin configurar los parámetros del contrato
149     function admCfgTiempoRespuesta() payable infinite gas
150     {
151         uint num,
152         uint t,
153         string memory units
154     } external soloAdmin {
155         // 1 Hour    3600 Seconds
156         // 1 Day     86400 Seconds
157         // 1 Week    604800 Seconds
158         // 1 Month   (30.44 days) 2629743 Seconds
159         // 1 Year    (365.24 days) 31556926 Seconds
160         uint valor;
161         if (
162             keccak256(abi.encodePacked(string(units))) ==
163             keccak256(abi.encodePacked("m"))
164         ) valor = t * 60;
165         if (
166             keccak256(abi.encodePacked(string(units))) ==
167             keccak256(abi.encodePacked("h"))
168         ) valor = t * 3600;
169         if (
170             keccak256(abi.encodePacked(string(units))) ==
171             keccak256(abi.encodePacked("d"))
172         ) valor = t * 86400;
173
174         if (num == 1) tiempo_respuesta = valor;
175         else if (num == 2) tiempo_votacion = valor;
176         else {
177             if (num == 3) cupo_respuestas = t;
178         }
179     }
180
181     // Fórmula para el cálculo de prestigio de un usuario en el sistema EsbrinaChain:
182     // Prestigio = ((RespuestasAcertadas)*10) / (totalRespuestas)
183     // El prestigio califica a un usuario entre 0-10 puntos.
184     // en el mejor de los casos.
185     // El prestigio solo se incrementa haciendo preguntas o respondiendo. En el primer caso
186     // (responder)puntua un 0.1 respecto del total y el segundo un 0.9. Responder mal penaliza bastante.
187
188     function calcPrestigio(address _usr) internal returns (int) { payable infinite gas
189         require(
190             preguntas[1].creada != 0,
191             "No existen preguntas en el sistema."
192         );
193         require(
194             preg_resp[1][1].autor != address(0),
195             "No existen respuestas en el sistema."
196         );
197         int venceUsr = 0;
198         int noVenceUsr = 0;
199         int prestigio = 0;
200         for (uint i = 1; i <= total_preg; i++) {
201             if (
202                 preguntas[i].estado == estado_preg.consulta &&
203                 calcAdrRespuestasAPregunta(i, _usr)
204             ) {
205                 (bool resp, bool vence) = adrRespondeVence(i, _usr);
206                 if (resp) {
207                     if (vence) {
208                         venceUsr++;
209                     } else noVenceUsr++;
210                 }
211             }
212             prestigio += calcPrestigioPreg(venceUsr, noVenceUsr);
213             usuarios[_usr].prestigio = prestigio;
214             return prestigio;
215         }
216     }

```

```

216     function calcPrestigioPreg(    █ infinite gas
217         int venceUsr,
218         int noVenceUsr
219     ) internal pure returns (int) {
220         if (venceUsr + noVenceUsr == 0) return 0;
221         else return ((venceUsr) * 10) / (venceUsr + noVenceUsr);
222     }
223     // 0.0005 ETH aprox. 1.5$
224     function consUsrPrestigio(address _usr) external payable returns (int) {    █ infinite gas
225         require(
226             preguntas[1].creada != 0,
227             "No existen preguntas en el sistema."
228         );
229         require(
230             preg_resp[1][1].autor != address(0),
231             "No existen respuestas en el sistema."
232         );
233         require(
234             msg.value >= 0.0005 ether,
235             unicode"Para conocer su prestigio debe enviar 0.0005 ETH."
236         );
237         int venceUsr = 0;
238         int noVenceUsr = 0;
239         int prestigio = 0;
240         for (uint i = 1; i <= total_preg; i++) {
241             if (
242                 preguntas[i].estado == estado_preg.consulta &&
243                 calcAdrRespuestasAPregunta(i, _usr)
244             ) {
245                 (bool resp, bool vence) = adrRespondeVence(i, _usr);
246                 if (resp) {
247                     if (vence) {
248                         venceUsr++;
249                     } else noVenceUsr++;
250                 }
251             }
252         }
253         prestigio += calcPrestigioPreg(venceUsr, noVenceUsr);
254         usuarios[_usr].prestigio = prestigio;
255         return prestigio;
256     }
257
258     // Recupera fondos del Bote
259     function BoteGotGod() public soloAdmin {    █ infinite gas
260         payable(_admin).transfer(address(this).balance);
261     }
262
263 }
```

Figura 16: Código fuente del contrato inteligente **esbrinachain**.

4.1.1 Test unitarios

Parte importante en el desarrollo de la funcionalidad de los contratos inteligentes (Smart Contracts) es disponer de un conjunto de prueba unitarios que valide las funciones. Para el proyecto **Esbrinachain** se han definido utilizando el software **Mocha**.

En la figura 17, se puede observar la salida por pantalla, para **validar la carga del contrato** Esbrinachain, como parte del código implementado posteriormente en las pruebas unitarias de Mocha. También se puede observar al final, la estructura de directorios dispuesta a tal efecto cumpliendo las reglas de Mocha.

```

Esbrinachain tests
eth_accounts
[
  '0xC97BEFA0D937E9b03500d65be0D7450A80e6C385',
  '0x8fbFa19c0eF4B22DFA9210562d342E6d5416480f',
  '0x789e510ba9EdD71df97C6cbE69EFF36B0b7d1Fa5',
  '0xD5215a0ce9243D191f7DE70535919fCdB8AeE5e9',
  '0x91C172eF550a8c1e18aa1C4701Aa2014601bEc11',
  '0x8575fB7d904ebcf2d10E9267512e5DE6BdC0D3d1',
  '0x48FCd5a155D025Ea9613ced79485DaDf3896BD03',
  '0x3A45aFD2070e63ad1Ed9e65DF3bB051E521B2B75',
  '0x0a9fD422Da2644bAF06a7874E8be55d2a15183e6',
  '0x4b312F25683B4a7d890FbFDe223249124c40de0E'
]
eth_getBlockByNumber
eth_gasPrice
eth_sendTransaction

Transaction: 0xec05bf17ceb00de5f25fdf8e245be80abad3dfcc091440f0bab8d99339ad5371
Contract created: 0xc6a5e18d026c7f41f351fd200c6f6f184b143dd7
Gas usage: 5217448
Block number: 1
Block time: Thu Aug 29 2024 19:37:07 GMT+0200 (hora de verano de Europa central)

eth_getTransactionReceipt
eth_getCode
Dirección del contrato: 0xc6a5e18d026c7f41f351fd200c6f6f184b143dd7
Dirección del contrato: 0xc6a5e18d026c7f41f351fd200c6f6f184b143dd7
  ↴ Desplegar contrato

1 passing (94ms)

c:\atest> dir
El volumen de la unidad C es OS
El número de serie del volumen es: 0467-1005

Directorio de C:\atest

29/08/2024  19:16    <DIR>          .
29/08/2024  18:40            1.541 compile.js
29/08/2024  18:52    <DIR>          contracts
29/08/2024  19:58    <DIR>          node_modules
29/08/2024  19:58            329.043 package-lock.json
29/08/2024  19:58            281 package.json
29/08/2024  13:05    <DIR>          test
              3 archivos         330.865 bytes
              4 dirs   514.022.830.080 bytes libres

C:\atest>

```

Figura 17: Salida por consola de la carga del contrato esbrinachain para los test unitarios Mocha.

En la figura 18, se muestran **los test unitarios** implementados para el proyecto **Esbrinachain**.

En la publicación del proyecto de GitHub se dispone del código en la carpeta **test_unitarios**, en el fichero de **esbrina_test.js**.

```

JS esbrina_test.js X JS compile.js S esbrinachain.sol
test > JS esbrina_test.js > ⚙ describe("Esbrinachain tests") callback > ⚙ it('Resolucion del ganador/res de la votacion') callback
  1 const assert = require('assert');
  2 const ganache = require('ganache');
  3 const Web3 = require('web3');
  4 const web3 = new Web3(ganache.provider());
  5 const { abi, bytecode } = require("../compile");
  6
  7
  8 console.log(abi);
  9 console.log(bytecode);
 10
 11 let accounts;
 12 let esbrinachain;
 13

Run | Debug | Show in Test Explorer
 14 describe("Esbrinachain tests", () => {
 15
 16   before("Obtener cuentas en la red: ", async () => {
 17     accounts = await web3.eth.getAccounts();
 18     console.log(accounts);
 19     esbrinachain = await new web3.eth.Contract(abi),
 20       deploy({ data: bytecode, arguments: [] });
 21       send({ from: accounts[0], gas: 10000000 });
 22     console.log('Dirección del contrato: ', esbrinachain.options.address);
 23   });
  Run | Debug | Show in Test Explorer
 24   it("Desplegar contrato", () => {
 25     console.log('Dirección del contrato: ', esbrinachain.options.address);
 26     assert.ok(esbrinachain.options.address);
 27   });
 28

Run | Debug | Show in Test Explorer
 29 it('Crear pregunta', async () => {
 30   const enunciado = "De qué color tienen los ojos los delfines?";
 31   const autor = "user1@gmail.com";
 32   await esbrinachain.methods.creaPregunta(enunciado, autor,"").send({ from: accounts[1], value:10, gas:7000000 });
 33     on('transactionHash', (transactionHash) => { console.log('TxId:', transactionHash); });
 34     then((receipt) => { console.log("Receipt: ", receipt); });
 35   const pregunta = await esbrinachain.methods.preguntas(1).call();
 36   assert.equal(pregunta.enunciado,enunciado,"Enunciado de la pregunta 1 no se ha registrado!");
 37   assert.equal(pregunta.autor, accounts[1], "autor de la pregunta 1 no se ha registrado!");
 38
 39 });
 40

Run | Debug | Show in Test Explorer
 41 it('Crear respuesta', async () => {
 42   const enunciado = "Verdes y azules?";
 43   const autor = "user2@gmail.com";
 44   const haRespondidoAPreg = await esbrinachain.methods.calcAdrRespuestasAPregunta(1, accounts[2]).call();
 45   assert.notEqual(haRespondidoAPreg,true,"No se puede responder más de una vez una pregunta.");
 46   await esbrinachain.methods.creaRespuesta(1,enunciado, autor,"").send({ from: accounts[2], gas:7000000 });
 47     on('transactionHash', (transactionHash) => { console.log('TxId:', transactionHash); });
 48     then((receipt) => { console.log("Receipt: ", receipt); });
 49   const pregunta = await esbrinachain.methods.preguntas(1).call();
 50   assert.notEqual(pregunta.autor,accounts[2],"El autor de la pregunta es el autor de la respuesta.");
 51   const resp1 = await esbrinachain.methods.preg_resp(1, 1).call();
 52   assert.equal(resp1.autor, accounts[2], "El autor de la respuesta no ha quedado registrado.");
 53
 54

Run | Debug | Show in Test Explorer
 55 it('Inicio del periodo de Votación (caso admin)', async () => {
 56   const pregunta = await esbrinachain.methods.preguntas(1).call();
 57   assert.equal(pregunta.estado, 0, "La pregunta ha de estar 'abierta' para poder pasar a 'votando'.");
 58   try {
 59     await esbrinachain.methods.iniciaVotacion(1).send({ from: accounts[0], gas: 7000000 });
 60       on('transactionHash', (transactionHash) => { console.log('TxId:', transactionHash); });
 61       then((receipt) => { console.log("Receipt: ", receipt); });
 62
 63   } catch(error) {
 64     console.log(error.message);
 65   }
 66   const admin = await esbrinachain.methods._admin().call();
 67   assert.equal(accounts[0],admin,"Solo el administrador del sistema puede iniciar la votación manualmente.")
 68   const tieneRespuestas = await esbrinachain.methods.calcRespAPreg(1).call();
 69   assert.notEqual(tieneRespuestas.length, 0, "La pregunta no tiene respuestas que votar.");
 70
 71   const resp1 = await esbrinachain.methods.preg_resp(1, 1).call();
 72   assert.equal(resp1.autor, accounts[2], "El autor de la respuesta no ha quedado registrado.");
 73
 74   assert.equal(pregunta.estado, 0, "La pregunta NO estaba en estado 0 (abierta)");
 75   const pregunta1 = await esbrinachain.methods.preguntas(1).call();
 76   assert.equal(pregunta1.estado, 1, "La pregunta ahora no está en estado 1 (votando)");
 77
 78 });
 79

```

```

    Run | Debug | Show in Test Explorer
  80  it("Votar una respuesta de una pregunta en periodo de Votación", async () => {
  81    const pregunta = await esbrinachain.methods.preguntas(1).call();
  82    assert.equal(pregunta.estado, 1, "La pregunta ha de estar en estado 'votando'.");
  83    assert.ok(pregunta.votada == false, "Para poder votar una pregunta no puede haberse resuelto la votación (votada=true)");
  84    const usuario = await esbrinachain.methods.usuarios(accounts[2]).call();
  85    assert.ok(usuario.vetado == false, "Para poder votar una pregunta el usuario no debe estar vetado.");
  86    const haRespondidoAPreg = await esbrinachain.methods.calcAdrRespuestasAPregunta(1, accounts[3]).call();
  87    assert.notEqual(haRespondidoAPreg, true, "No se puede VOTAR una pregunta si se es autor de una respuesta.");
  88    assert.notEqual(pregunta.autor, accounts[3], "El creador de la pregunta no la puede votar.");
  89    try {
  90      await esbrinachain.methods.votarRespuesta(1,1).send({ from: accounts[3], gas: 7000000 });
  91      on('transactionHash', (transactionHash) => { console.log('TxId:', transactionHash); }).
  92      then((receipt) => { console.log("Receipt: ", receipt); });
  93    } catch (error) {
  94      console.log(error.message);
  95    }
  96  });
  97
  98
  99  Run | Debug | Show in Test Explorer
100  it('Final del periodo de Votación (caso admin)', async () => {
101    const pregunta = await esbrinachain.methods.preguntas(1).call();
102    assert.equal(pregunta.estado, 1, "La pregunta ha de estar en estado 'votando' para poder pasar a 'consulta'.");
103    try {
104      await esbrinachain.methods.finalizaVotacion(1).send({ from: accounts[0], gas: 7000000 });
105      on('transactionHash', (transactionHash) => { console.log('TxId:', transactionHash); }).
106      then((receipt) => { console.log("Receipt: ", receipt); });
107    } catch(error) {
108      console.log(error);
109    }
110    const admin = await esbrinachain.methods._admin().call();
111    assert.equal(accounts[0],admin,"Solo el administrador del sistema puede FINALIZAR la votación manualmente.");
112
113    assert.equal(pregunta.estado, 1, "La pregunta NO estaba en estado 1 (votando)");
114    const pregunta1 = await esbrinachain.methods.preguntas(1).call();
115    assert.equal(pregunta1.estado, 2, "La pregunta ahora no está en estado 2 (consulta)");
116
117  });
118
  Run | Debug | Show in Test Explorer
119  it('Resolucion del ganador/res de la votacion', async () => {
120    let pregunta = await esbrinachain.methods.preguntas(1).call();
121    assert.notEqual(pregunta.estado, 3, "La pregunta está anulada.");
122    assert.ok(pregunta.votada==true,"Para resolver la votación la pregunta debe estar votada (campo votada=true)");
123    const respAntesVoto = await esbrinachain.methods.preg_resp(1, 1).call();
124    assert.equal(respAntesVoto.autor, accounts[2], "El autor de la respuesta no ha quedado registrado.");
125    const votos = respAntesVoto.votos;
126    const usuarioAntes = await esbrinachain.methods.usuarios(accounts[2]).call();
127
128    try {
129      await esbrinachain.methods.votarRespuesta(1,1).send({ from: accounts[4], gas: 7000000 }).
130      on('transactionHash', (transactionHash) => { console.log('TxId:', transactionHash); }).
131      then((receipt) => { console.log("Receipt: ", receipt); });
132    } catch (error) {
133      console.log(error.message);
134    }
135
136    const respDespuesVoto = await esbrinachain.methods.preg_resp(1, 1).call();
137    assert.ok(respAntesVoto.votos == respDespuesVoto.votos, "La resolución de la votación no se ha producido, el voto se ha contabilizado.");
138    const usuarioDesp = await esbrinachain.methods.usuarios(accounts[2]).call();
139    assert.notEqual(usuarioAntes.prestigio, usuarioDesp.prestigio, "El usuario ganador no ha incrementado su reputación.");
140
141  );
142
143
144
145
146
147
148  });

```

Figura 18: Código fuente de los test unitarios del proyecto **Esbrinachain**.

En la figura 19, se puede observar la salida de los test unitarios superados satisfactoriamente, y mostrando al final un caso de error.

Figura 19: Resultado obtenido en la ejecución de los test unitarios.

Aunque el lector pueda alarmarse por el error, está corregido, solo es una muestra de funcionamiento de los test con Mocha, y de la importancia de que no solo sirven para validar nuestro código a posteriori, sino también para **desarrollarlo**.

La buena definición de los contratos inteligentes es una parte crucial en el desarrollo de una aplicación web sobre una red blockchain. No solo porque el código debe ajustarse a un tamaño limitado, sino porque es fácil incurrir en dobles desarrollos de las mismas comprobaciones. En el proyecto Esbrinachain se ha considerado que la independencia del código de los Smarts Contracts debía ser autosuficiente y no depender del interfaz del cliente web implementado, y por eso se ha mantenido algún código que también está implementado en el aplicativo web.

4.2 Angular Framework

Para el desarrollo del interfaz de la aplicación web, se ha utilizado el entorno de desarrollo de Angular Framework en su versión 17. Se puede consultar en el siguiente enlace:

<https://github.com/EsbrinaChain/esbrina>

y ejecutar la publicación del proyecto en:

<https://esbrinachain.github.io/esbrina>

Angular Framework es un entorno de desarrollo web de una sola página que permite el desarrollo de frontend y backend para un aplicativo web. Es gratuito y de código abierto y está basado en **TypeScript**, **HTML** y **CSS**. Utiliza un modelo basado en la **Vista ('View')**, compuesta de plantillas HTML y CSS para el diseño del interfaz de la aplicación web y de código **TypeScript** para la comunicación entre el interfaz y el modelo de datos **'Model'**. Permite actualizar de forma instantánea la vista a partir de los cambios en el modelo de datos de manera bidireccional (**'binding'**). Mediante esta característica se puede incorporar en una sola página diferentes diseños de página web llamados componentes que pueden ser actualizados de manera independiente. Para la edición y desarrollo de la programación del proyecto se ha utilizado el editor **Visual Code**. Como componente de innovación una variante de base de datos de Firebase llamada **Firebase Cloud Firestore**, que nos ha permitido crear un backend basado en cache, almacenado en la nube. El resultado ha sido una mejora de rendimiento en el acceso y visualizado de datos, que es más rápido y ágil que el obtenido directamente con la testnet blockchain de Sepolia.

4.2.1 Estructura del proyecto

El proyecto **Esbrinachain** incluye para definir la aplicación web, un **componente principal** para la ejecución del proyecto, siete **componentes adicionales**, y un **servicio** (que implementa la gestión del wallet digital). La estructura del proyecto es la misma que está replicada en Github en la ‘branch’ **Esbrinachain**, por tanto es la misma que se puede consultar en la figura 4.

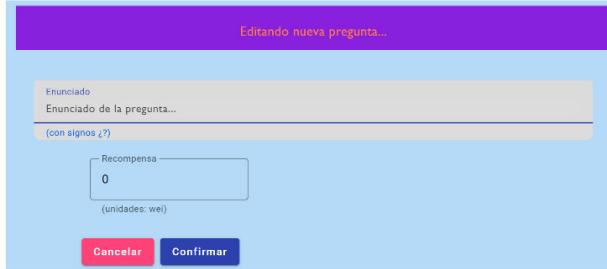
4.2.2 Descripción de los componentes

4.2.2.1. Componente ‘get-preg’

Para la implementación de la adquisición de los datos de las preguntas del sistema se ha utilizado un componente de Angular llamado **‘matDialog’**, que permite abrir una ventana emergente sobre el diseño de la página web, para la introducción del **enunciado de la pregunta** y su **recompensa**. Este componente necesita que se le pase como parámetro la plantilla HTML+CSS (para definir el aspecto/estilo de la ventana y para gobernar todo lo que sucede en la ventana), y un código TypeScript asociado que lo implemente. Para todo ello, se ha creado el componente **get-preg**.

Analizando como se ha implementado este componente también se puede comprender de manera sencilla como se ha implementado la introducción de las respuestas a cada pregunta.

En la figura 20, se puede observar la plantilla ('template') del componente y visualización.



```

get-preg
  get-preg.component.html
  get-preg.component.css
  get-preg.component.spec.ts
  get-preg.component.ts

```

```


Editando nueva pregunta...



Enunciado  
Enunciado de la pregunta...  
(con signos ⓘ?)



Recompensa  
0  
(unidades: wei)



Cancelar
Confirmar



1 <div class="fondo">
2   <div class="centrado globals">
3     <div class="titulo" mat-dialog-title>Editando nueva pregunta...</div><br>
4     <mat-dialog-content>
5       <div class="enunciado">
6         <mat-form-field appearance="fill" class="textarea">
7           <mat-label>Enunciado</mat-label>
8           <textarea matInput type="text" cdkTextareaAutosize [(ngModel)]=>dataPreg.enunciado class="textarea" placeholder="Enunciado de la pregunta..."></textarea>
9           <mat-hint>(con signos ⓘ?)</mat-hint>
10        </mat-form-field>
11      </div><br>
12      <mat-form-field appearance="outline" width="30" class="recompensa_globals">
13        <mat-label>Recompensa</mat-label>
14        <input matInput type="text" [(ngModel)]=>dataPreg.recompensa class="recompensa" placeholder="Valor de la recompensa..."></input>
15        <mat-hint>(unidades: wei)</mat-hint>
16      </mat-form-field>
17    </mat-dialog-content>
18    <mat-dialog-actions class="globals">
19      <button mat-raised-button color="accent" (click)=>cancelado()>Cancelar</button>
20      <button mat-raised-button [mat-dialog-close]=>dataPreg color="primary" (click)=>confirmado()>Confirmar</button>
21    </mat-dialog-actions>
22  </div>
23 </div>
24
25 .globals{
26   margin-left: 10%;
27   margin-right: 10%;
28 }
29
30 .textarea {
31   border: 1px solid #blueviolet;
32   border-radius: 10px;
33   border-color: #blueviolet;
34   width: 100%;
35   background-color: #gainsboro;
36   font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;
37   font-size: 18px;
38   color: #rgb(20, 113, 236);
39 }
40
41 .titulo{
42   text-align: center;
43   font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;
44   font-size: 18px;
45   color: #coral;
46   background-color: #blueviolet;
47 }
48
49 .fondo{
50   background-color: #rgb(189, 219, 244);
51 }
52
53 .button{
54   background-color: #rgb(38, 88, 188);
55   border-radius: 8px;
56   color: #white;
57   padding: 15px;
58   text-shadow: 1px;
59   color: #rgb(20, 50, 220);
60 }
61
62 .enunciado {
63   font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif;
64   font-size: 18px;
65   color: #rgb(20, 50, 220);
66 }
67
68 @Component({
69   selector: 'app-get-preg',
70   standalone: true,
71   imports: [FormsModule, ReactiveFormsModule, MatDialogModule, MatDialogContent, MatDialogActions, MatDialogClose, MAT_DIALOG_DATA ] from '@angular/material/dialog',
72   exports: [MatButtonModule, MatInputModule, MatDialogModule, MatDialogRef, MatDialogContent, MatDialogActions, MatDialogClose, MAT_DIALOG_DATA ],
73   templateUrl: './get-preg.component.html',
74   styleUrls: ['./get-preg.component.css'
75 })
76 export class GetPregComponent {
77
78   dataPreg: any;
79
80   constructor(
81     private dialog: MatDialogRef<GetPregComponent>,
82     @Inject(MAT_DIALOG_DATA) public data: any ) {
83     this.dataPreg = data;
84   }
85
86   cancelado() {
87     this.dialog.close();
88   }
89
90   confirmado() {
91     this.dialog.close(this.dataPreg);
92   }
93
94 }
95


```

Figura 20: Componente get-preg.

4.2.2.2. Componente 'get-resp'

En la figura 21, mostramos el componente **get-resp**, que se utiliza para recoger los parámetros de una respuesta.

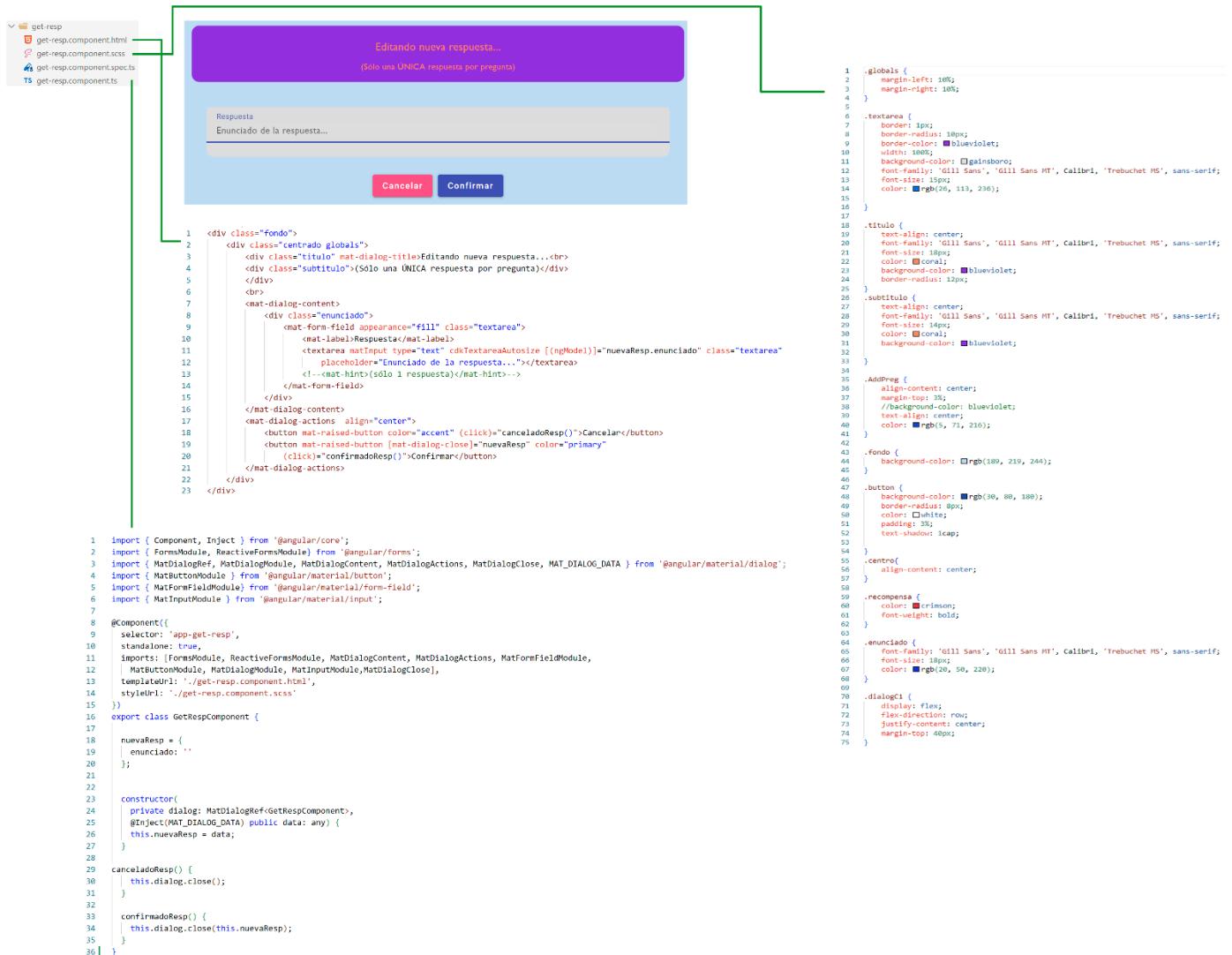


Figura 21: Componente **get-resp**.

4.2.2.3. Componente 'pregunta'

El componente **pregunta** se ha diseñado para representar cada pregunta registrada en **Esbrinachain**.

Cada pregunta está registrada tanto en el contrato inteligente (blockchain) como en el backend (Firestore Cloud), pero no tienen estructuras iguales. En el contrato inteligente se restringen los datos a los mínimamente necesarios, mientras que en el backend se puede tener más información si es de utilidad al aplicativo web. Aunque el proyecto Firestore, es gratuito, tampoco es ilimitado, pues se contabilizan el número de lecturas, escrituras y borrados, disponiendo de un máximo de 15.000 Kb diarias.

Los datos utilizados para describir cada pregunta son: **índice** por orden de llegada al sistema, **enunciado** de la pregunta, **autor** (address y email), **fecha de creación**, **estado** (abierta, votando, consulta, anulada), **recompensa**, **fecha de votación** y **número de incrementos de recompensa**.

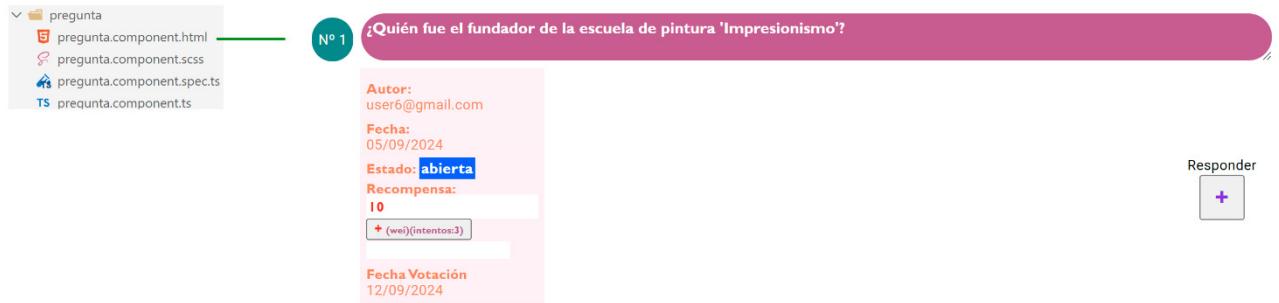


Figura 22: Visualización en el aplicativo web del componente **pregunta**.

En el componente **pregunta**, se ha realizado la mayor parte de toda la programación del aplicativo e implementa la programación de todo lo que sucede en el espacio de la pregunta, incluso incluir el componente **respuesta**:

1. Controla la recepción de datos de preguntas y respuestas.
2. Controla que los datos de una pregunta se guarden en el contrato inteligente **Esbrinachain** mediante la ejecución de funciones para la construcción, envío y gestión de la recepción de transacciones, utilizando funciones asíncronas.
3. Controla que los datos de una respuesta se almacenen también en el contrato **Esbrinachain**.
4. Se ocupa de mantener la recepción de parámetros de entrada de otros componentes. Ejemplos de estos parámetros son el **wallet del usuario** (modalidades: interno de la aplicación y Metamask), y el objeto contenedor del **cliente web3** para la conexión a la red blockchain de Sepolia.
5. Gestiona una lista de todas las preguntas incluidas en el sistema y refresca sus datos cada 30 seg.
6. Se ocupa de sincronizar los datos existentes en la red blockchain y en el backend de la nube.
7. Según el estado de la pregunta, ajusta los controles necesarios para cada acción como, responder a una pregunta, votar una respuesta o consultar la respuesta más votada, ver figura 23.

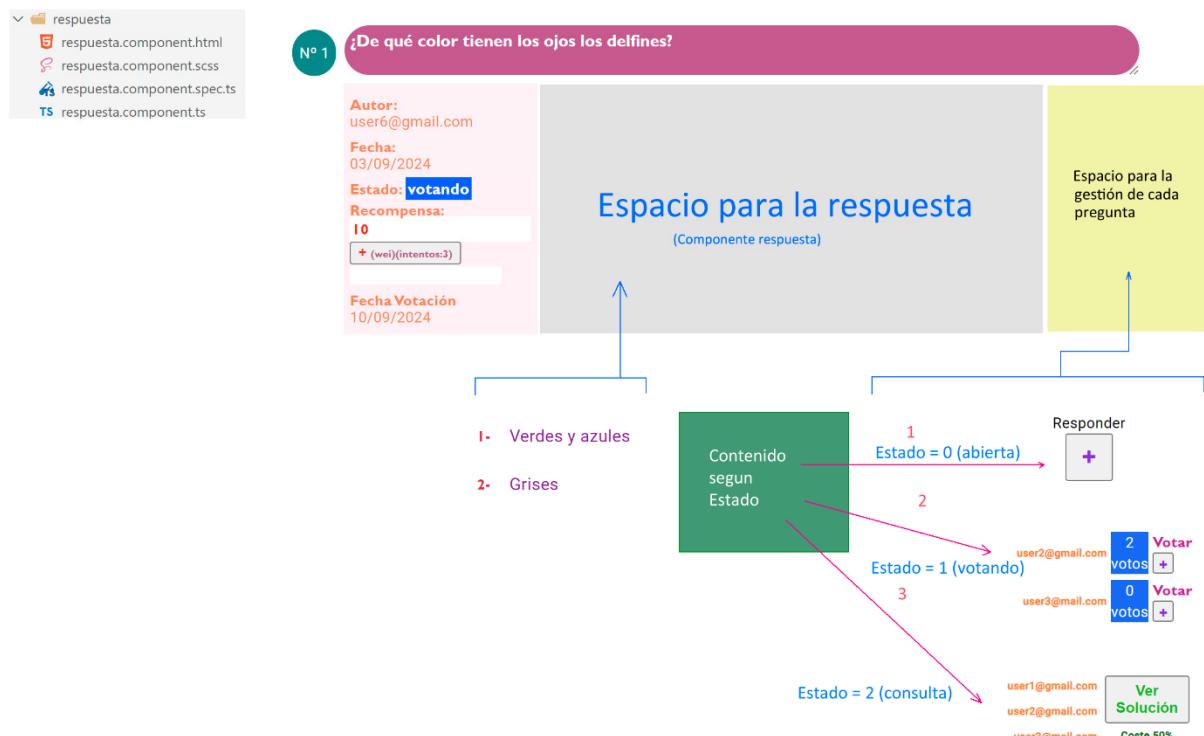


Figura 23: Visualización de los componente **pregunta** y **respuesta**.

4.2.2.4. Componente ‘respuesta’

Como se puede observar en la figura 24, el espacio de respuesta a una pregunta contiene los datos representativos de una respuesta como: **índice de la respuesta** por orden de llegada, **enunciado** de la respuesta, el **autor** de la respuesta (email), el **número de votos** recibidos en la votación, **si está anulada** o no y **si es la más votada** en el proceso de votación.



Figura 24: Visualización del componente **respuesta**.

El componente **respuesta** recibe como ‘input’ el **número de pregunta** al que va asociado y gestiona la visualización de todas las respuestas a una pregunta, porque esta incrustado dentro de la plantilla del componente **pregunta**.

Como en el backend de Firestore disponemos de las preguntas y las respuestas en una colección de documentos, se pueden visualizar con mayor velocidad que si se tuviera que realizar una llamada de lectura a la red blockchain.

El criterio seguido para **enlazar de manera inequívoca preguntas y respuestas** es almacenar el **blockNumber** y el **transactIndex** de la pregunta en la respuesta. Esta estrategia nos permite liberarnos de indexar numéricamente los elementos de una colección en el backend (tanto preguntas como respuestas), aspecto que **no es menor** porque la respuesta de la red blockchain es **asíncrona** y puede **no conservar el orden inicial** de nuestras peticiones.

4.2.2.5. Componente ‘prestigio’

El componente prestigio se ha generado para mostrar a modo de ventana emergente, la lista de reputación del sistema **Esbrinachain**. De esta forma, el usuario puede comparar en qué **posición se encuentra** respecto el resto de los usuarios, porque aparece resaltado en un color diferente, ver figura 25.

Solo mencionar, que por la sencillez del caso de uso, se ha probado de acceder directamente al contrato y **aplicar un bucle sobre una llamada ‘call’ (lectura)**. No se necesita leer nada del backend ni actualizar datos y por eso es un buen caso de uso para ver cómo responde el aplicativo en su rendimiento si actuamos de esta manera. Queda para el futuro, comprobar el rendimiento cuando el número de usuarios sea elevado, aunque previsiblemente será bajo, lo que confirma la necesidad de un backend para la aplicación.

En la figura 25, se puede observar cómo visualiza el aplicativo web, la **reputación del usuario en el sistema** resaltando al usuario activo de la lista completa de reputaciones.



Figura 25: Visualización de la reputación del sistema en esbrinachain.

4.2.2.6. Componentes 'registrar' y 'wallet-in'

Los componentes **registrar** y **wallet-in** son los encargados al iniciar la aplicación, de asignar el wallet al usuario. En primer lugar, se da de alta al usuario en el Backend, y sólo se requiere una dirección de correo electrónico, caso de la figura 26:

The registration page has a blue header bar with the text 'ESBRINA CHAIN - Sistema de Conocimiento Colaborativo.' and a language selection dropdown set to 'ES'. The main content area features a yellow background with a central graphic showing four stylized human figures in a circle, with the words 'PREGUNTAS' and 'RESPUESTAS' repeated in red and green respectively around them. Below the graphic, the text 'Nº de usuarios actuales: 0' and a 'Registrar nuevo usuario' button are visible. A registration form follows, containing fields for 'Email', 'Contraseña', and 'Alias', and a 'Registro' button at the bottom.

Figura 26: Pantalla para el alta de nuevos usuarios en Esbrinachain.

La aplicación comprueba que los datos de la dirección de correo estén en un formato correcto, y acepta una contraseña para el usuario en la aplicación, además de un alias.

En la figura 27, se solicita la **frase de recuperación** de alguna dirección que disponga el usuario, y se **integra el wallet en el sistema**, de manera que **se podrá operar sin usar la billetera de Metamask**. La contraseña que se debe proporcionar no es la misma que la indicada en la figura 26, sino la que utilizó el usuario para generar su dirección de red blockchain y que calza con su frase de recuperación.



Figura 27: Pantalla de generación de wallet para un usuario recién registrado.

Si el usuario se siente más confortable utilizando la billetera Metamask, puede pulsar en la opción **Login**, de la barra del **MENÚ horizontal** situada en la parte superior de la pantalla y acceder mediante la opción '**Login con Metamask**', figura 28:



Figura 28: Pantalla de acceso a utilizar el wallet Metamask.

Resumiendo, se han configurado dos posibilidades para asignar el wallet (billetera digital) a un usuario:

1. El usuario ya tiene wallet y dispone de una frase de recuperación y quiere integrar su billetera en la aplicación.
2. El usuario está acostumbrado a usar **Metamask**, (debe estar integrado ya en su dispositivo PC).

Tal como se nos indicó durante las clases del curso, se ha desarrollado el wallet integrado, utilizando las mismas librerías mostradas en la figura 29:

```
// Librerías de Ethereum y DApp
import * as Mnemonic from 'bitcore-mnemonic';
import { hdkey } from 'ethereumjs-wallet';
import * as bip39 from 'bip39';
import * as util from '@ethereumjs/util';
import Web3 from 'web3';
import * as CryptoJS from 'crypto-js';
```

Figura 29: Librerías para la generación del wallet de usuario.

El código se puede consultar fácilmente, porque se ha implementado un servicio en Angular Framework para poderlo servir a futuros componentes. En la práctica, se facilita el servicio solo al componente **wallet-in**, mientras que el resto de los componentes lo reciben utilizando un decorador tipo `@Input`.

Para el lector que le interese profundizar en cómo se generan las claves públicas y privadas sin depender de una billetera estándar como Metamask, le resultará interesante consultar el código contenido en el fichero **wallet-in.component.ts**.

El componente **registrar** se utiliza para asignar el wallet de Metamask a los usuarios que elijan esta opción. En este caso la aplicación utiliza el cliente web integrado en la billetera de Metamask ‘**window.ethereum**’.

Una vez obtenida la dirección Ethereum contenida en el wallet de Metamask, la aplicación no diferencia sus flujos operacionales y trabaja con el objeto wallet y con las direcciones pública y privada contenidas en su interior. Para los envíos de transacciones, como se verá más adelante en el código, si estamos usando una billetera Metamask la variable ‘**metamask**’ estará a true y los envíos de transacciones no hará falta firmarlos, simplemente usar la función **sendTransaction**. En caso de usar un wallet integrado, se firmará primero el contenido de la transacción con la función **signTransaction**, y luego se enviará con la función **sendSignedTransaction**.

4.2.2.7. Componente ‘app’

El componente **app**, se utiliza para presentar el contenido global de toda la aplicación web del proyecto **Esbrinachain**. Integra el resto de los componentes, mostrándolos por pantalla según el valor de variables lógicas que controlan cada caso de uso:

1. El usuario **no ha utilizado nunca la aplicación**, se ha de **registrar** y definir **variables locales** y su **wallet**, ver figura 26.
2. El usuario **ya está dado de alta**, pero **no ha definido sus variables locales ni su wallet**, ver figura 27.
3. El usuario **se ha dado de alta y tiene variables locales y wallet**. En ese momento, el usuario gobierna si inicia la aplicación usando un wallet integrado o el wallet de Metamask, pulsando en la palabra **Login** del menú superior, ver figura 28. Si en cambio accede a la opción **Registro**, se le permite eliminar su cuenta si lo cree conveniente. De esta manera, la aplicación cumplirá con la ley de protección de datos, en el aspecto de poder eliminar sus datos, cuando lo crea oportuno.

Es importante aclarar, que en el párrafo anterior, por variables locales se refiere a las variables locales definidas en el contexto de sesión del navegador, por la aplicación. En la figura 30, se muestran las definidas en el caso del aplicativo **Esbrinachain**.

The screenshot shows the Chrome DevTools Network tab with the 'Filtrar' (Filter) button set to 'https://esbrinachain.github.io'. Under the 'Almacenamiento' (Storage) section, the 'Almacenamiento local' (Local Storage) for the domain is expanded, showing the following table:

Clave	Valor
esbrinaUser	U2FsdGVkX19SDA6NvCQYH9eAbvX2oV9...
esbrinaUserMail	user6@gmail.com
seeds	U2FsdGVkX1/IHRITf+uTRSkTxWRPnd859...

Figura 30: Variables locales de sesión identificativas del usuario

Si el aplicativo no detecta, la variable **esbrinaUser**, considera que **se ha de dar de alta el usuario**. Y si no encuentra la variable **seeds**, entonces considera que **se debe asignar un wallet al usuario**. La variable **esbrinaUserMail** también es necesaria y no debe borrarse. El aplicativo web está diseñado para sincronizar la existencia de estas variables con los datos de usuario almacenados en el backend. La gestión de usuarios se realiza básicamente en el backend y sólo se aplica el registro de usuario en los contratos inteligentes guardando la dirección de red blockchain pública del usuario que está operando y realizando las transacciones.

4.2.3 Particularidades en la funcionalidad de Esbrinachain

El concepto más importante aprendido durante la programación del proyecto **Esbrinachain**, es quizás el hecho de que al programar para una cadena de bloques el orden de las peticiones no tiene porqué conservarse en la respuesta de la red. Que existe un **tiempo de latencia** en la consecución de la transacción y que no siempre la respuesta va a ser inmediata. Es por ello por lo que hay que programar creando **funciones asíncronas** que esperen a que la respuesta llegue por parte de la red blockchain. También es importante conocer cómo funciona la programación con '**Promesas**', como las que utiliza la **librería web3**. La promesa es una estrategia mediante la cual la respuesta nos llega de manera asíncrona y que se adapta a las necesidades que se requieren.

Especial cuidado, hay que ponerlo, en aspectos cruciales, como construir una petición a una transacción con los campos adecuados, y utilizar las funcionalidades facilitadas por el **cliente web3**, para calcular el precio de la **unidad de gas** y disponer del **gas estimado** para la realización de la transacción. El correcto uso de las **unidades y cantidades** usadas en el envío de la transacción también es esencial. Y finalmente saber **captar la respuesta de la cadena de bloques a nuestra transacción** (lo que se conoce en la jerga blockchain como el **receipt**). En el receipt, el objeto retornado por una transacción que ejecuta una función de escritura, podemos **obtener los datos de los eventos desencadenados en la ejecución de la función**. Las llamadas a funciones de modificación o escritura del contrato inteligente no reportan parámetros de salida, es por ello necesario **utilizar los eventos de manera eficiente**, para recuperar información de los cambios sucedidos en nuestro contrato desplegado en la cadena de bloques.

En el proyecto **Esbrinachain**, se ha adoptado un criterio restrictivo que quizás no llegue a ser muy popular en un caso real. Consiste en cambiar el estado de las preguntas a medida que hay un acceso para realizar una acción y se detecta la finalización de un periodo para esa acción (aplicado a los tiempos de respuesta y votación). Eso implica:

1. Querer responder, consumir gas para responder y no poder hacerlo, sino que se recibe una notificación de la finalización del tiempo de respuesta.
2. Querer votar, consumir gas para votar y no poder hacerlo, sino que se recibe una notificación de la finalización del tiempo de votación.

En futuras versiones es posible que se deba habilitar soluciones diferentes, si afecta al funcionamiento del sistema por generar un disgusto, rechazo o falta de motivación en el usuario.

A continuación, se proporciona un detalle del flujo en las operaciones y las transacciones enviadas en el aplicativo web a la hora de **crear una pregunta, responder una pregunta, votar una pregunta, resolver una votación y calcular la reputación del autor de la respuesta más votada**.

En los tres casos, cuando se crea una pregunta, o una respuesta, o se hace efectivo el voto por una respuesta, primero se realiza una transacción al contrato y si esta funciona bien, se reflejan esos datos en el backend, **siempre esperando la resolución de la transacción para no gravar datos equívocos en el backend**.

El sufijo 'SC' en la notación del nombre de las funciones indica que gestiona el acceso al Smart Contract. En la figura 31, se muestra el primero de los tres casos:

Caso crear Pregunta

```

526 showDialog(){
527   const dialogConfig = new MatDialogConfig();
528   dialogConfig.width = '70%';
529   dialogConfig.autoFocus = true;
530   dialogConfig.data = { enunciado: '', recompensa: '0' };
531   this.dialogRef = this.matDialog.open(GetPregComponent, dialogConfig);
532   this._data = this.dialogRef.afterClosed().subscribe((result: any) => {
533     if (result != undefined){
534       | this.creaPreguntaSC(result.enunciado, result.recompensa);
535     }
536   });
537 }

La función ShowDialog obtiene los datos,
la función creaPreguntaSC hace la petición al contrato para
ejecutar la función 'creaPregunta',
y la función insertaPregunta registra los datos en el Backend

298 async insertaPregunta(idp: any, enunciado: any, recompensa: any, blockNumber: any, transactionIndex: any) {
299   this.balanceWalletAddress = await this.getBalanceAddress(this.wallet.address);
300   // Usando Ganache retorna el valor del balance de la cuenta en ETH cuando deberían ser wei.
301   const recompensaETH = this.web3Obj.utils.fromWei(recompensa, "ether");
302   //console.log(this.balanceWalletAddress);
303   //console.log(`Recompensa: ${recompensa}`);
304   //console.log(`RecompensaETH: ${recompensaETH}`);
305   if(recompensaETH < this.balanceWalletAddress){
306     this.totalPregs++;
307     const prg = {
308       blockNumber: Number(blockNumber),
309       transactionIndex: Number(transactionIndex),
310       idp: Number(idp),
311       anulada: false,
312       autor: window.localStorage.getItem('esbrinaUserMail'),
313       autorAddress: this.wallet.address,
314       creado: this.createDate(new Date(new Date().getTime())),
315       enunciado: enunciado,
316       estado: "abierta",
317       fecha_votacion: this.createDate(new Date().getTime() + 7 * 24 * 60 * 60 * 1000),
318       idioma: "es",
319       recompensa: Number(recompensa),
320       incRecompensa:this._num_incr_recompensa_sc,
321       email: window.localStorage.getItem('esbrinaUserMail'),
322     };
323     console.log("Pregunta: ",prg,"Total Preguntas: ",this.totalPregs);
324     await addDoc(collection(this.db, "Pregs"), prg);
325     this.conPregsQuery();
326   }
327 }

254 async creaPreguntaSC(enunciado: any, recompensa: any) {
255   const email = window.localStorage.getItem('esbrinaUserMail');
256   const gasPrice = await this.web3Obj.eth.getGasPrice();
257   console.log("Gas Price: ", gasPrice);
258   const gasEstimated = await this.contract.methods.creaPregunta(enunciado, email, email, email).estimateGas({
259     from: this.wallet.address,
260     value: recompensa
261   });
262   console.log("Gas Estimated", gasEstimated);
263   var rawData = {
264     from: this.wallet.address, // admin (address generada con la semilla facilitada).
265     to: contract.address,
266     value: Number(recompensa),
267     //gasPrice: this.web3Obj.utils.toHex(10000000000),
268     //gasLimit: this.web3Obj.utils.toHex(1000000),
269     gasPrice: gasPrice,
270     gasLimit: gasEstimated,
271     nonce: await this.web3Obj.eth.getTransactionCount(this.wallet.address),
272     data: this.contract.methods.creaPregunta(enunciado, email, email, email).encodeABI()
273   }
274   console.log(rawData);
275   var signed: any;
276   this.lineEstadoPreg(this.idiomaSel.m57, 3000, 'visible');
277   let receipt;
278   try {
279     if (this.metamask) {
280       receipt = await this.web3Obj.eth.sendTransaction(rawData);
281     } else {
282       signed = await this.web3Obj.eth.accounts.signTransaction(rawData, this.wallet.privateKey.toString('hex'));
283       receipt = await this.web3Obj.eth.sendSignedTransaction(signed.rawTransaction);
284     }
285     console.log("Receipt: ", receipt);
286     console.log(`Logs[0].topics[1]`, receipt.logs[0].topics[1]);
287     const idp = this.web3Obj.utils.hexToNumber(receipt.logs[0].topics[1]);
288     this.lineEstadoPreg(this.idiomaSel.m49, 3000, 'visible');
289     this.insertaPregunta(idp, enunciado, recompensa, receipt.blockNumber, receipt.transactionIndex);
290   } catch(error) {
291     console.log(error);
292     this.lineEstadoPreg(this.idiomaSel.m50, 3000, 'visible');
293   }
294 }
```

Figura 31: Código implementado para la creación de una pregunta.

Líneas 526 Ejecutaremos la función '**showDialog**' para abrir el diálogo de crear una pregunta, obtendremos los datos y si no se ha pulsado el botón '**Cancelar**' utilizaremos estos datos (**enunciado, recompensa**) para ejecutar la función '**creaPreguntaSC**'.

Línea 534: Ejecutamos '**creaPreguntaSC**' pasando como parámetros **enunciado, recompensa**.

Línea 255: Obtenemos el email del usuario de las variables locales.

Línea 256-263: Obtenemos de la cadena de bloques el **precio de gas** y el **gas límite estimado** usando ejecuciones asíncronas con la palabra **await**. Esto significa, que se esperará al resultado de la promesa para disponer de los valores más actuales.

Líneas 264-274: Construimos un objeto JSON, que represente una transacción.

Línea 273: Se rellena el campo **data** del objeto JSON, con el **encodeABI** del método a ejecutar que en nuestro caso es '**creaPregunta(enunciado, recompensa, email, email, email)**'. Se ha tomado

esta estrategia para evitar al usuario estar entrando datos personales. A efectos reales siempre se gestiona todo con la dirección del emisor de la transacción más su correo electrónico.

- Líneas 280-290: Si se usa **Metamask** no hace falta firmar la transacción porque lo hace el wallet, y utilizamos el método **sendTransaction**, mientras que si gestionamos internamente el wallet, firmamos el objeto transacción con el método **signTransaction** y enviamos la transacción firmada, con el método **sendSignedTransaction**. Tanto en un caso como en otro siempre de forma asíncrona y recibimos el resultado en la varia **receipt**. Si no salta un error por el **try-catch**, significa que la transacción ha ejecutado la función en el contrato y se ha generado un evento ‘**PreguntaCreada**’. Si se produce error puede ser **por falta de fondos en el wallet**.
- Línea 289: Del objeto de respuesta **receipt** tenemos acceso a la información **logs** que contiene cada evento emitido durante la transacción. En el caso de evento ‘**PreguntaCreada**’ se indica el identificador de orden de la pregunta en el contrato (**idp**), y éste es el valor que nos interesa para actualizar en el Backend con una pregunta nueva.
- Línea 291: Ejecutamos la función ‘**insertaPregunta**’ diseñada para actualizar el Backend y dar de alta la pregunta.
- Líneas 298-327: Se actualiza el Backend en Firestore Cloud y se refresca la lista de preguntas con la función **conPregsQuery()** que ejecuta la consulta de todas las preguntas incluyendo la nueva pregunta y por ser la lista diferente de la anterior, Angular Framework actualiza el modelo de datos en la ‘View’ y los datos son actualizados en la pantalla.

Caso crear Respuesta

El caso de crear una respuesta tiene un flujo de operaciones muy parecido. Solo cambian las condiciones que deben cumplirse para que un usuario pueda responder a una pregunta. Estas condiciones ya las mostramos al analizar la función del contrato inteligente **responder**:

1. No ser autor de la pregunta.
2. Solo responder 1 sola vez en cada pregunta.
3. Que la pregunta esté en estado ‘**abierta**’ para recibir respuestas.

En el caso de crear una respuesta, se añade una dificultad. Cuando el periodo de respuesta ha finalizado, el contrato no crea la respuesta y se informa al usuario del error en la transacción y automáticamente se muestra el nuevo estado en el aplicativo web. Observando el código de la figura 32, el lector puede comprobar que en este caso es necesario recuperar el índice de respuesta del evento ‘**RespuestaCreada**’ para actualizar la respuesta en el backend. Pero la operativa es similar a la de crear una pregunta.

Variaciones que el lector puede detectar son:

1. Se utiliza el componente **get-resp** para la recogida de datos de la respuesta.
2. Se arrastran los parámetros de **blockNumber** y **transactionIndex** de la pregunta, para después actualizar el backend.
3. Se mira una vez hecha la transacción, si ha variado el estado de la pregunta. Un cambio de estado implica que la respuesta no se ha registrado en el contrato y, o bien el tiempo de respuesta ha finalizado pasando la pregunta a estado “votando” o bien si no ha recibido respuestas se ha anulado.

También se **ha implementado una línea de notificaciones** en cada pregunta para que el usuario conozca en cada momento lo que va sucediendo, funciones ‘**LinEstadoPreg**’ y ‘**LinEstadoResp**’. La primera está destinada al momento en que se crean preguntas y la pantalla utiliza el botón “**Realizar una nueva pregunta**” y localiza

las notificaciones en el espacio ocupado por la primera pregunta cuando esta existe. La segunda se posiciona en el mismo espacio indicado pero en la pregunta en que estemos operando.

```

485 |     async dialogRespuesta(blockNumber:any,transactionIndex:any,idPreg: any, email: any) {
486 |       this.actualizaDatosPregSC(blockNumber, transactionIndex, idPreg);
487 |       const usarDialog = await this.haRespondido(idPreg);
488 |       let noAutorPreg = false;
489 |       const pregunta = await this.contract.methods.preguntas(idPreg).call();
490 |       console.log("Autor pregunta en contrato (address): ", pregunta.autor.toLowerCase());
491 |       if ([pregunta.autor.toLowerCase()] != this.wallet.address.toLowerCase()) {
492 |         noAutorPreg = true;
493 |       }
494 |       // console.log("pregunta.autor.toLowerCase()", usarDialog, "noAutor: ", noAutorPreg);
495 |       const estado_actual = await this.contract.methods.estadoPreg(idPreg).call();
496 |       //console.log("Estado actual: ", estado_actual);
497 |       if (!usarDialog && noAutorPreg && estado_actual=="Abierta.") {
498 |         const dialogConfig = new MatDialogConfig();
499 |         dialogConfig.width = '70%';
500 |         dialogConfig.autoFocus = true;
501 |         dialogConfig.data = { enunciado: '' };
502 |         this.dialogRefResp = this.matDialog.open(GetRespComponent, dialogConfig);
503 |         this.datos = this.dialogRefResp.afterClosed().subscribe(result => {
504 |           if (result != undefined) {
505 |             |   this.creaRespuestaSC(idPreg, result.enunciado, blockNumber, transactionIndex); |
506 |           }
507 |         });
508 |       } else {
509 |         if (usarDialog) {
510 |           |   this.linEstadoResp(idPreg, this.idiomaSel.m51, 3000,'visible');
511 |         }
512 |         if (!noAutorPreg) this.linEstadoResp(idPreg, this.idiomaSel.m52, 3000,'visible');
513 |         if (estado_actual!="Abierta.") this.linEstadoResp(idPreg, this.idiomaSel.m53, 3000,'visible');
514 |         console.log("Ya se ha respondido la pregunta: ", usarDialog,
515 |         "\nEs autor de la pregunta: ", !noAutorPreg,
516 |         "\nSolo la combinación 'false', 'false' permite responder."
517 |         "\nEstado pregunta:", estado_actual);
518 |       }
519 |     }

```

la función 'dialogRespuesta' obtiene los datos de la respuesta,
la función 'creaRespuestaSC' realiza la transacción de llamada
a la función del contrato 'creaRespuesta'
la función 'insertaRespuesta' almacena los datos en el Backend

```

399 |     async insertaRespuesta(id_resp: any, idPreg: any, enunciado_resp: any,
400 |                             blockNumber_preg: any, transactionIndex_preg: any) {
401 |       const rsp = {
402 |         email: window.localStorage.getItem('esbrinachainUserMail'),
403 |         id_resp: Number(id_resp),
404 |         blockNumber: blockNumber_preg,
405 |         transactionIndex: transactionIndex_preg,
406 |         enunciado: enunciado_resp,
407 |         ganadora: false,
408 |         votos: 0,
409 |         anulada: false,
410 |         id_preg: Number(idPreg),
411 |       };
412 |       //console.log("Respuesta introducida: ",rsp);
413 |       const r = await addDoc(collection(this.db, "Resps"), rsp);
414 |       this.conPregQuery();
415 |
416 |     this.conPregQuery();
417 |   }

```

```

347 |     async creaRespuestaSC(id_preg: any, enunciado_resp: any,blockNumber:any,transactionIndex:any) {
348 |       const email = window.localStorage.getItem('esbrinachainUserMail');
349 |       const gasPrice = await this.web3Obj.eth.getGasPrice();
350 |       const gasEstimated = await this.contract.methods.creaRespuesta(id_preg, enunciado_resp, email, email, email, email);
351 |       console.log("Gas Price: ",gasPrice);
352 |       console.log("Gas Estimate: ",gasEstimated);
353 |       var rawData = {
354 |         from: this.wallet.address, // admin (address generada con la semilla facilitada).
355 |         to: contract_address,
356 |         value: 0,
357 |         //gasPrice: this.web3Obj.utils.toHex(gasPrice * BigInt(2)),
358 |         //gasLimit: this.web3Obj.utils.toHex(gasEstimated),
359 |         gasPrice: this.web3Obj.utils.toHex(BigInt(800000000000)),//50070176532n 46790342006n
360 |         gasLimit: this.web3Obj.utils.toHex(1000000),
361 |         nonce: await this.web3Obj.eth.getTransactionCount(this.wallet.address),
362 |         data: this.contract.methods.creaRespuesta(id_preg,enunciado_resp, email, email, email).encodeABI()
363 |       }
364 |       //console.log(rawData);
365 |       var signed: any;
366 |       let receipt;
367 |       this.linEstadoResp(id_preg, this.idiomaSel.m48, 2000, 'visible');
368 |       if (this.metamask) {
369 |         |   receipt = await this.web3Obj.eth.sendTransaction(rawData);
370 |       }
371 |       else {
372 |         signed = await this.web3Obj.eth.accounts.signTransaction(rawData, this.wallet.privateKey.toString('hex'));
373 |         receipt = await this.web3Obj.eth.sendSignedTransaction(signed.rawTransaction);
374 |       }
375 |       console.log("Receipt: ", receipt);
376 |
377 |       const preg_estado = await this.contract.methods.estadoPreg(id_preg).call();
378 |       console.log("preg_estado: ",preg_estado);
379 |       if (preg_estado == "Abierta.") {
380 |         const idResp = this.web3Obj.utils.hexToNumber(receipt.logs[0].topics[2]);
381 |         console.log("receipt.logs[0].topics[2]: ", receipt.logs[0].topics[2]);
382 |         console.log("idResp: ", idResp);
383 |         this.insertaRespuesta(idResp, id_preg, enunciado_resp, blockNumber, transactionIndex);
384 |         this.linEstadoResp(id_preg, this.idiomaSel.m49, 5000, 'visible');
385 |       }
386 |       else if (preg_estado=="Votando.") {
387 |         await this.updEstadoPregBackend(id_preg, "votando");
388 |         this.linEstadoResp(id_preg, this.idiomaSel.m64, 5000, 'visible');
389 |         console.log("Datos de pregunta actualizados en el backend: votando");
390 |       }
391 |       else if (preg_estado == "Anulada.") {
392 |         await this.updEstadoPregBackend(id_preg, "anulada");
393 |         this.linEstadoResp(id_preg, this.idiomaSel.m65, 5000, 'visible');
394 |       }
395 |       console.log("Datos de pregunta actualizados en el backend: anulada");
396 |     }

```

Figura 32: Código implementado para la creación de una respuesta.

Caso realizar el voto por una respuesta

En cada caso de uso que hemos revisado, podemos observar que el flujo de operaciones contempla la existencia de un wallet. A este nivel de la aplicación, el usuario ya ha sido creado y ya se le ha asignado un wallet o billetera digital. A estas alturas de operativa, solo nos hemos de preocupar por si el wallet es **Metamask** o uno **interno creado con la aplicación**. En realidad, si disponemos del software de Metamask integrado, utilizaremos el cliente web3 que lleva integrado (**window.ethereum**), sino utilizaremos el objeto web3 extraído de la librería web3 instalada en el proyecto angular framework.

En la figura 34, se muestra el código de implementación del voto a una respuesta. La operación de votar la puede hacer cualquier usuario registrado y con billetera digital que no sea autor de la pregunta, ni autor de sus respuestas. Al votar por una respuesta enviará una transacción que incrementará el valor del campo 'votos' de la estructura que representa a una respuesta en el contrato inteligente. Si la transacción funciona bien y no tiene errores, entonces también actualizaremos el mismo campo pero de la estructura utilizada en el backend, para poder actualizar los datos mostrados por pantalla.

Para sincronizar los datos del backend y del contrato se trabaja también con funciones asíncronas que esperan el resultado de la respuesta hasta que este se produce. En este caso es más fácil, porque las estructuras de datos ya existen y solo se trata de actualizar los valores. Utilizamos la misma técnica que en los casos anteriores, pero es importante conocer que todas las acciones se desencadenan desde el

componente **respuesta**, que recibe mediante parámetros @Input, las mismas variables globales definidas en el componente **pregunta**:

1. La variable **this.DB** que contiene acceso al entorno de Firestore o backend.
2. La variable **this.webObj** que contiene el wallet.
3. El **id_preg** y **estado** (mostrada en el componente pregunta dónde está incrustado el componente respuesta).
4. El **tipo de wallet** (Metamask=true o no).
5. Los campos que enlazan la lista de preguntas con las respuestas: **blockNumber** y **transactionIndex**.
6. El idioma activo.
7. Y finalmente dos variables @Output que utilizan eventos de Angular F. para comunicar un componente ‘hijo’(**respuesta**) con uno ‘padre’ (**pregunta**). El parámetro de salida (@Output) ‘refresh’ emite un evento para refrescar la lista de preguntas y el parámetro de salida (**notifica**) emite un evento para mostrar un mensaje informativo de lo que va pasando en la línea de notificaciones del componente **pregunta**.

En la figura 33, se muestra el tag o marca HTML del componente **respuesta**, incrustado en el template del componente **pregunta** con toda esta información:

```
<app-respuesta (refresh)="actualizaListaPregs()"
(notifica)="notificacio(item.idp,$event)" [id_preg]="item.idp"
[web3obj]="web3obj" [wallet]="wallet" [estado_actual]="item.estado"
[metamask]="metamask" [idiomaSel]="idiomaSel"
[blockNumber]="item.blockNumber" [transactionIndex]="item.transactionIndex"></app-respuesta>
```

Figura 33: Código HTML del template del componente **respuesta** incrustado en el de **pregunta**.

El componente **respuesta**, detecta cuando un periodo de respuesta se ha agotado cuando un usuario intenta responder, y en ese momento el diseño del ‘template’ o plantilla HTML cambia para mostrar el componente respuesta cuando la pregunta está en estado ‘votando’. Se incluyen unos contadores de voto y un botón para votar para cada respuesta asociada a la pregunta visitada.



Figura 34: Detección automática del cambio al estado ‘votando’ en una pregunta.

A continuación se describe que el conjunto de acciones que se desencadenan cuando pulsamos el botón ‘+’ (‘votar’) de una respuesta, hasta que finalmente el voto queda contabilizado (ver la dirección de las flechas en la figura 35).

```

178 |    async selectVoto(id_preg: any, id_resp: any) {
179 |        let noAutorPreg = false;
180 |        const pregunta = await this.contract.methods.preguntas(id_preg).call();
181 |        if (!pregunta.autor.toLowerCase() != this.wallet.address.toLowerCase())
182 |            noAutorPreg = true;
183 |        console.log(`Pregunta a Votar: ${pregunta}`);
184 |        // No es autor de ninguna respuesta en la pregunta
185 |        const EsAutorResp = await this.haRespondido(id_preg);
186 |        console.log(`EsAutorResp: ${EsAutorResp}`);
187 |        // No ha votado otras respuestas => solo puede votar 1 vez
188 |        const noHaVotado = await this.sinVoto(id_preg);
189 |        console.log(`noHaVotado: ${noHaVotado}`);
190 |        const estadoActualPreg = await this.contract.methods.estadoPreg(id_preg).call();
191 |        console.log(`estadoActual: ${pregunta.estado}, ${estadoActualPreg}`);
192 |        if (!EsAutorResp && !noHaVotado && !noAutorPreg && pregunta.estado == 1) {
193 |            console.log(`Votando por la resp ${id_resp} de la pregunta ${id_preg}`);
194 |            await this.votarRespuestaSC(id_preg, id_resp);
195 |        }
196 |    }
197 |    else if (pregunta.estado == 1 && (!EsAutorResp || !noHaVotado || !noAutorPreg)) {
198 |        console.log(`La pregunta ${id_preg}, ${noHaVotado}, pero este usuario no puede votarla.`);
199 |        if (!noAutorPreg) this.notifica.emit(this.idiomaSel.m66);
200 |        if (!noHaVotado) this.notifica.emit(this.idiomaSel.m67);
201 |    } else if (estadoActualPreg == "Consulta") {
202 |        console.log(`La pregunta ${id_preg}, ${noHaVotado}, ${noAutorPreg}, ${noHaVotado} no esté en estado 'votando' sino en estado de ${estadoActualPreg}`);
203 |        this.estadoActualPreg = "consulta";
204 |        this.updEstadoPregBackend(id_preg, "consulta");
205 |    }
206 |
207 |}
208 |
209 |}

161 |    async updVotoBackend(id_preg: any, id_resp: any) {
162 |        console.log(`updVotoBackend: ${id_preg}, ${id_resp}`);
163 |        const queryResps = query(collection(this.db, '/Resps'),
164 |            where('id_preg', '=', Number(id_preg)),
165 |            where('id_resp', '=', Number(id_resp)));
166 |        const usSnapshot = await getDocs(queryResps);
167 |        const id = usSnapshot.docs.map(doc => doc.ref.id);
168 |        const item = doc(this.db, 'Resps', id[0]);
169 |        let docSnap = await getDoc(item);
170 |        let updData: any;
171 |        if (docSnap.exists()) {
172 |            updData = docSnap.data();
173 |            updData.votos += 1;
174 |            console.log(`updData: ${updData}`);
175 |            await setDoc(item, updData);
176 |        }
177 |    }
178 }

92 |    async votarRespuestaSC(id_preg: any, id_resp: any) {
93 |        //console.log(`Parametros votacion: ${id_preg}, ${id_resp}`);
94 |        const gasPriceResp = await this.web3obj.eth.getGasPrice();
95 |        this.notifica.emit(this.idiomaSel.m58 + gasPriceResp.toString());
96 |        console.log(`Gas Price: ${gasPriceResp}`);
97 |        const gasEstimatedResp = await this.contract.methods.
98 |            votarRespuesta(id_preg, id_resp).estimateGas({ from: this.wallet.address });
99 |        console.log(`Gas Estimated: ${gasEstimatedResp}`);
100 |        this.notifica.emit(this.idiomaSel.m58 + gasEstimatedResp.toString());
101 |        var rawData = {
102 |            from: this.wallet.address,
103 |            to: contract_address,
104 |            value: 0,
105 |            gasPrice: this.web3obj.utils.toHex(BigInt(800000000000)),
106 |            gaslimit: this.web3obj.utils.toHex(1000000),
107 |            nonce: await this.web3obj.eth.getTransactionCount(this.wallet.address),
108 |            data: this.contract.methods.votarRespuesta(id_preg, id_resp).encodeABI()
109 |        }
110 |        //console.log(rawData);
111 |        let receipt;
112 |        var signed: any;
113 |        try{
114 |            this.notifica.emit(this.idiomaSel.m60);
115 |            if (this.metamask) {
116 |                receipt = await this.web3obj.eth.sendTransaction(rawData);
117 |            }
118 |            else {
119 |                signed = await this.web3obj.eth.accounts.signTransaction(rawData,
120 |                    this.wallet.privateKey);
121 |                receipt = await this.web3obj.eth.sendSignedTransaction(signed.rawTransaction);
122 |            }
123 |            console.log(`Receipt: ${receipt}`);
124 |            if (receipt.logs.length == 0) {
125 |                await this.updEstadoPregBackend(id_preg, id_resp);
126 |                this.notifica.emit(this.idiomaSel.m63);
127 |            }
128 |            else if (receipt.logs.length == 1) {
129 |                console.log(`Se ha intentado votar y la pregunta ha cambiado a estado 'consulta'`);
130 |                await this.updEstadoPregBackend(id_preg, "consulta");
131 |                this.notifica.emit(this.idiomaSel.m61);
132 |            }
133 |            else if (receipt.logs.length == 2) {
134 |                await this.updEstadoPregBackend(id_preg, "anulada");
135 |                this.notifica.emit(this.idiomaSel.m62);
136 |            }
137 |        }catch (error) {
138 |            console.log(error);
139 |            this.notifica.emit(this.idiomaSel.m50);
140 |        }
141 |        this.refresh.emit();
142 |    }
143 |}

```

Figura 35: Código implementado para ejecutar un voto por una respuesta.

Entender la operativa una vez explicados los dos casos anteriores es sencillo. Cuando se pulsa el botón ‘votar’ se ejecuta la función **selectVoto()**, se obtiene del contrato los datos más actuales de la pregunta actual y se comprueba que el usuario que quiere votar no incumpla:

- No ser el propietario de la pregunta.
- No haber votado la pregunta anteriormente.
- La pregunta esté en estado ‘votando’.
- No ser el autor de alguna de las respuestas.

Si se incumple alguno de los condicionantes para poder votar, se muestra una notificación por pantalla y no se deja votar. En caso contrario, se llama a la función **votarRespuestaSC**, que se encarga de enviar una

transacción para ejecutar la función del contrato ‘**votarRespuesta**’ pasando como parámetro el índice de la pregunta y el índice de la respuesta.

Como en los anteriores casos de uso, los eventos desencadenados en el contrato nos determinarán las acciones que debe ejecutar el aplicativo:

- a. Si al votar el periodo de votación se ha agotado pero existen respuestas realizadas, la pregunta cambiará a estado ‘**consulta**’, y no se contabilizará el voto, generándose dos eventos en el contrato que significa un registro de **tres** entradas en el vector de **logs** del recibo (**receipt**) devuelto por la transacción. De todo ello se notifica al usuario en la línea de notificación.
- b. Si al votar el periodo de votación se ha agotado y no existen respuestas realizadas, la pregunta cambiará a estado (‘**anulada**’), y no se contabilizará el voto, generándose un evento ‘**PreguntaAnulada**’ en el contrato que significa un registro de **dos** entradas en el vector de **logs** del recibo (**receipt**) devuelto por la transacción. De todo ello se notifica al usuario en la línea de notificación.
- c. Si el vector **logs** del recibo (**receipt**) devuelto por la transacción no tiene ninguna entrada (length=0), entonces significa que se ha podido votar correctamente y se actualiza el backend en Firestore, ejecutando la función **updVotoBackend**. Una actualización del backend siempre se acompaña en la aplicación web con una ejecución del evento **refresh**, que actualiza en pantalla los datos de la lista de preguntas.

4.2.4 Complementos destacables del proyecto Esbrinachain

En este apartado recogemos algunos complementos importantes ya sea para el funcionamiento de la aplicación web del proyecto **Esbrinachain**, o para tener en cuenta en otros desarrollos en la cadena de bloques:

1. Se ha creado un fichero de configuración de nombre **Firestore1.ts** para:
 - a. Definir el objeto **firebaseConfig**, para la conexión a Firestore.
 - b. Almacenar la URL al proveedor usado para acceder a la red Sepolia
 - c. Almacenar en la variable ‘**contract_address**’ (dirección Ethereum del contrato inteligente) desplegado en a la red Sepolia, ver figura 36.

```

7  export const firebaseConfig = {
8    apiKey: "AIzaSyAHz9zSUK258f3CyoMA2cvE8Kf2BnF442c",
9    authDomain: "esbrinachain-777.firebaseio.com",
10   projectId: "esbrinachain-777",
11   storageBucket: "esbrinachain-777.appspot.com",
12   messagingSenderId: "825098637790",
13   appId: "1:825098637790:web:1c3930b7e4033004c70d4f",
14   measurementId: "G-Y0VFSVPTBC"
15 };
16
17 //export const providerETH = 'https://sepolia.drpc.org';
18 export const providerETH = 'https://lb.drpc.org/ogrpc?network=sepolia&dkey=AvUhdnE01EOFnfamA-Vzcgm0fMHgbcsR773QUh7cII5S';
19 export const contract_address: any = "0x26748Da13d70D2FBa73ff4a7BEe84bFce94bad0";

```

Figura 36: Fichero de configuración **Firestore1.ts**

2. El proyecto Esbrinachain ofrece un soporte en tres idiomas (castellano, inglés y catalán) para el interfaz web. Existe el fichero **idioma.ts**, que contiene los mensajes mostrados por pantalla. Los mensajes existentes se pueden modificar dinámicamente, o bien se pueden añadir nuevos idiomas. Si bien es cierto que el interfaz tiene soporte en tres idiomas, los autores de las preguntas, son los que deciden el idioma que utilizan y en ese idioma se almacenan las preguntas. Para ver una muestra del fichero **idioma.ts**, ver figura 37.

```

export const es = {
  m0: "Idioma:",
  m1: "ESBRINA CHAIN - Sistema de Conocimiento Colaborativo.",
  m2: "Compartiendo el conocimiento de todos usando tecnología Blockchain.",
  m3: "Preguntar",
  m4: "Responder",
  m5: "Votar",
  m6: "Opinar",
  m7: "Estadísticas",
  m8: "Configuración",
  m9: "Login",
  m10: "Login con Metamask",
  :
  m67: "Si ha respondido a la pregunta, no puede votarla",
  m68: "Solo se puede votar 1 respuesta.",
  m69: "12 Palabras reservadas...",
  m70: "Enviando Tx - Obteniendo las respuestas más votadas...",
  m71: "Coste = ",
  m72: " Usuario ",
  m73: " Balance del Usuario",
  m74: "",
  m75: "",
  m76: "",
  m77: "",
};

export const en = {
  m0: "Language:",
  m1: "ESBRINA CHAIN - Collaborative Knowledge System",
  m2: "Sharing everyone's knowledge with blockchain technology.",
  m3: "Asking",
  m4: "Answering",
  m5: "Voting",
  m6: "Opinions",
  m7: "Statistics",
  m8: "Configuration",
  m9: "Login",
  m10: "Login with Metamask",
  :
  m68: "One vote per question is permitted.",
  m69: "12 reserved words...",
  m70: "Sending Tx - getting the most voted answers...",
  m71: "Cost = ",
  m72: " User ",
  m73: " User Balance ",
  m74: "",
  m75: "",
  m76: "",
  m77: "",
};

export const cat = [
  m0: "Idioma:",
  m1: "ESBRINA CHAIN - Sistema de Coneixement Col.laboratiu.",
  m2: "Compartint el coneixement de tots amb tecnologia Blockchain.",
  m3: "Preguntar",
  m4: "Respondre",
  m5: "Votar",
  m6: "Opinar",
  m7: "Estadístiques",
  m8: "Configuració",
  m9: "Login",
  m10: "Login amb Metamask",
  :
];

```

Figura 37: Fichero de configuración para el soporte en varios idiomas (**idioma.ts**).

3. Se ha implementado un **Backend** en la nube, utilizando la **tecnología Firestore** de Firebase, que se describirá en el siguiente apartado. Google cloud ofrece varias opciones para este cometido. Para el proyecto Esbrinachain se ha elegido Firestore cloud. Esta opción es la más completa, pero siempre se puede optar por **Realtime Database**, la base de datos clásica de Firebase JSON que es adecuada para aplicaciones con modelos de datos sencillos con búsquedas simples y escalabilidad limitada.
4. Se dispone a disposición, un **juego de usuarios y billeteras digitales** con los que probar el aplicativo web. Está en la carpeta de proyecto **usr_test**, aunque se debe recibir autorización para conocer su contraseña.
5. Gestión del acceso a la red Sepolia y adquisición de Faucets, ver figura 38:
Para la obtención de faucets (ETH útiles en la testnet Sepolia) se han utilizado las 2 primeras URL, mientras que la tercera es muy útil para conocer el estado del proveedor de la red Sepolia. Si el proveedor está caído, las transacciones darán error aunque todo esté bien programado.
Otro link interesante para el lector podría ser <https://drpc.org/> para disponer de proveedor Sepolia con más prestaciones de velocidad mediante suscripción temporal gratuita.

- ⌚ Ethereum Sepolia Faucet
<https://cloud.google.com/application/web3/faucet/ethereum/sepolia>
- 🌐 Chainlink Faucets - Get Testnet Tokens
<https://faucets.chain.link>
- 🌐 Sepolia RPC and Chain settings | ChainList
<https://chainlist.org/chain/11155111>

RPC Server Address	Height	Latency	Score	Privacy	Connect Wallet
wss://ethereum-sepolia-rpc.publicnode.com	6651489	0.055s	✓	✓	Connect Wallet
https://ethereum-sepolia.rpc.subquery.net...	6651489	0.091s	✓	○	Connect Wallet
https://sepolia.gateway.tenderly.co	6651489	0.106s	✓	✗	Connect Wallet
https://gateway.tenderly.co/public/sepolia	6651489	0.106s	✓	✗	Connect Wallet
https://eth-sepolia.public.blastapi.io	6651489	0.115s	✓	⚠	Connect Wallet
https://ethereum-sepolia.blockpi.network/v...	6651489	0.123s	✓	⚠	Connect Wallet
https://sepolia.drpc.org	6651489	0.123s	✓	○	Connect Wallet
https://ethereum-sepolia-rpc.publicnode.c...	6651489	0.124s	✓	✓	Connect Wallet

Figura 38: Recursos para la adquisición de faucets y estado de la red Sepolia.

Es importante identificar los enlaces de mayor rendimiento a la red blockchain. Si una red no está disponible y coincide con una petición realizada en ese momento por la aplicación web, esta fallará, y no siempre se devolverá una descripción clara del motivo. Para estos casos, en el sistema **Esbrinachain**, se ha optado por mostrar el error aunque sea indeterminado y permitir que el usuario vuelva a intentar la transacción en otro momento. Los errores puntuales no suelen ser un problema, pero cuando el proveedor usado es el causante de múltiples errores, es una buena opción disponer de alternativas de servicio que permitan salvar el rendimiento de la aplicación web cuando esté en producción. Por esta razón ficheros de configuración dinámicos al margen del código de la aplicación pueden ayudar a solventar su mantenimiento. Se podrían efectuar los cambios de configuración de forma dinámica, y compilar y publicar el proyecto en pocos minutos.

4.3 Cloud Firestore – Firebase

Firebase dispone de un servicio de base de datos documental en la nube llamado Firestore. Para el desarrollo de aplicaciones es gratuito y útil, si no se hace un uso intensivo. En el proyecto **Esbrinachain** se ha incluido este servicio para la implementar la base de datos del Backend, por no tener coste, y por servir eficientemente en el soporte de datos.

En la figura 39, se muestra un ejemplo de uso: la **actualización de los datos de una pregunta**. Se da este ejemplo, ya que el resto de las entidades como las respuestas y los usuarios se gobiernan de manera muy parecida. Siempre se accede al elemento de la colección mediante una consulta y se actualiza el elemento.

```

330  async updEstadoPregBackend(id_preg: any, estado_actual: any) {
331    const queryPreg = query(collection(this.db, '/Pregs'), where("idp", "==", Number(id_preg)));
332    const usSnapshot = await getDocs(queryPreg);
333
334    const id = usSnapshot.docs.map(doc => doc.ref.id);
335    const item = doc(this.db, "Pregs", id[0]);
336    let docSnap = await getDoc(item);
337    let updData: any;
338    if (docSnap.exists()) {
339      updData = docSnap.data();
340      if (estado_actual == 'anulada') updData.anulada = true;
341      updData.estado = estado_actual;
342      await setDoc(item, updData);
343      console.log("La pregunta ", id_preg, " ha cambiado a estado", updData.estado);
344      this.conPregsQuery();
345    }
346  }

```

Figura 39: Código de actualización de estado para una pregunta en Cloud Firestore.

En la figura 40, se muestra el uso de un día normal de programación, para que el lector tenga más información del margen de utilización de este servicio respecto a sus limitaciones.



Figura 40: Gráfica representativa de uso y límites de Firestore Cloud.

4.3.1 Colecciones de datos en Firestore Cloud

En Firestore se han almacenado tres colecciones de datos: **Usuarios**, **Pregs** y **Resps**. Las estructura de las colecciones en el Backend no tienen por qué coincidir con las de los contratos inteligentes, aunque si comparten lógicamente los datos más importantes. En la figura 41, se puede identificar un ejemplo de documento de cada una de las tres colecciones creadas:

(default)	Pregs	OWTLxuxlDx3Jlm73fujd
+ Iniciar colección	+ Agregar documento	+ Iniciar colección
Pregs >	OWTLxuxlDx3Jlm73fujd >	+ Agregar campo
	XrcZKt7vSGWa6Tp19YNy	anulada: false
	v8HI3DyUQHsvqaSPXswB	autor: "user6@gmail.com"
		autor_address: "0xf562c02033df4b174885d8c7678dc1489340f6d9"
		blockNumber: 6645458
		creada: "06/09/2024"
		email: "user6@gmail.com"
		enunciado: "¿Quién fue el fundador de la escuela de pintura 'Impresionismo'?"
		estado: "votando"
		fecha_votacion: "06/09/2024"
		idioma: "es"
		idp: 3
		incRecompensa: 3
		recompensa: 50
		transactionIndex: 59
	+ Agregar documento	+ Iniciar colección
	0Pfd3hKCwAw0ZKehy0sg >	+ Agregar campo
Resps >	SU2f3khognExeV6pFEhi	anulada: false
	j5tKsd9rHbMe3gxVp9wJ	blockNumber: 6643927
	nZZZYVvaKTgm16BjIUuA	email: "user6@gmail.com"
	q2CetpjuKXQQadDCy4H0	enunciado: "verdes y azules"
		ganadora: false
		id_preg: 1
		id_resp: 1
		transactionIndex: 54
		votos: 1
+ Iniciar colección	+ Agregar documento	+ Iniciar colección
	BuxR5bLBH70XXEiMW71M >	+ Agregar campo
Usuarios >	L4D0VQB19tw3jTmrTkSe	alias: "user5"
	LUqphhLqJah86gC91LFn	creado: 5 de septiembre de 2024, 9:24:49 p.m. UTC+2
	YZgfIKoM0us6HIAwNtg3	email: "user5@gmail.com"
	dAcijJlryponDhFucZrI	existe: true
	szxUKbMLaLdtg0Z0TVzX	id: 6
		psw: "xxx"
		reputacion: 0
		vetado: false
		wallet: "0xf985206f62eee19dc6eb26b669600a4d6a2e1e78"

Figura 41: Cuadro de las colecciones y documentos utilizados en el Backend.

Como se ha comentado anteriormente, las claves generadas de manera automática son de mucha utilidad en la programación asíncrona. El lector que quiera más prestaciones como disponer de campos autonuméricos tiene que buscar usos en una base de datos más completa como Firebase.

Para las consultas de los datos se han tenido que generar índices compuestos que permiten aumentar la velocidad en la recuperación de datos, ver figura 42. Los índices simples por defecto están ya implementados.

Resp	blockNumber Ascendente transactionIndex Ascendente id_resp Ascendente __name__ Colección Habilitado
Ascendente	
Pregs	blockNumber Ascendente transactionIndex Ascendente __name__ Ascendente Colección Habilitado
Resps	id_preg Ascendente id_resp Ascendente __name__ Ascendente Colección Habilitado

Figura 42: Relación de índices compuestos en Firestore.

05. Casos de uso identificables en Esbrinachain

5.1 Gestión de alta de usuarios

La gestión de los usuarios requiere de varias operaciones:

- Alta de un usuario en el backend
- Asignación de wallet cuando se dispone de frase de recuperación. El aplicativo ofrece dos posibilidades integrado y Metamask.
- Baja de un usuario, y eliminación de sus datos en el proyecto.

En el proceso de alta, como se puede observar en la figura 43, el nuevo usuario no existe en el backend, ni en la aplicación. En nuestro caso de uso, existen 4 usuarios y vamos a crear el quinto. El aplicativo detecta que no tiene registradas en el ‘Almacenamiento local’ del navegador ninguna de las variables identificativas de un usuario.

The figure consists of two main parts. On the left is a screenshot of the Esbrina Chain web application. It features a sidebar with a 'Users' section. In the main area, there's a 'Agregar documento' (Add document) button and a list of user documents. One document is highlighted with a timestamp: '5 de septiembre de 2024, 8:14:24 p.m...' and a ID: 'YZgflKoM0us6HIAwNtg3'. Below this, there's a registration form with fields for Email, Contraseña (Password), and Alias, and a 'Registro' (Register) button. At the bottom, it says 'Nº de usuarios actuales: 4' (Current number of users: 4). On the right is a screenshot of the browser's developer tools Network tab. The 'Aplicación' (Application) section shows a single request to 'http://localhost:4200'. The 'Almacenamiento' (Storage) section shows the 'LocalStorage' tab with several items listed, none of which correspond to the user just registered. The 'Servicios en segundo plano' (Background services) section also lists various browser services.

Figura 43: Fase1: Alta de un nuevo usuario: no existe en el backend ni en la aplicación.

Si rellenamos los campos de datos del formulario inicial, email, contraseña y alias, la aplicación web comprobará que el ‘email’ proporcionado tiene un formato correcto y aceptará la contraseña y el alias a efectos de **registrar al usuario** en la aplicación cuando pulsemos el botón ‘Registro’, ver figura 44.

Figura 44: Fase 2: El usuario está registrado pero no dispone de wallet.

En estos momentos, el usuario no dispone de wallet pero si está dado de alta en el backend y la aplicación dispone de las variables '**esbrinaUser**' y '**esbrinaUserMail**' registradas en el almacenamiento local del navegador. Para disponer de wallet, deberá introducir la frase de recuperación para generar su par de claves pública y privada. El aplicativo web puede generar el wallet del usuario e integrarlo en el aplicativo como se realizaría en el caso de darse de alta en Metamask. Y si el usuario ya se hubiera dado de alta en Metamask, dispone también de esos datos. Se solicita la **frase de recuperación** (debe pertenecer a un conjunto controlado de palabras), y la **contraseña** la que se asoció al wallet. Todo este proceso se consigue usando la librería '**bitcore-mnemonic**'. Al pulsar el botón '**Wallet sin Metamask**' se genera el wallet y se cifrará su frase de recuperación en la variable '**seeds**'. De esta manera, el usuario, cada vez que entre a la aplicación, solo facilitando su contraseña permitirá al aplicativo generar su par de claves pública y privada que se utilizarán en el envío de transacciones (sin usar el wallet de Metamask), aunque tenga la extensión instalada en el navegador. En la figura 45, se puede comprobar el estado de la aplicación web cuando tiene **el usuario registrado y con wallet** potencialmente disponible. De esta manera el usuario nunca expone sus claves pública y privada sino que se generan en tiempo de ejecución.

Figura 45: Fase 3: El usuario está registrado y dispone de wallet.

En este momento, si el usuario desea realizar la operación de **Login**, dispone en el menú horizontal situado en la parte superior de la pantalla, las dos opciones de wallet con tan solo hacer clic en la palabra '**Login**'.

En la figura 46, se visualiza la pantalla para usar el **wallet integrado** en la aplicación y en la figura 47 la que permite usar el **wallet de Metamask**, siempre que ya esté disponible su extensión en el navegador de Internet.

The screenshot shows the ESRINA CHAIN application interface. At the top, there is a navigation bar with 'Login' and 'Registro' buttons. A language selector dropdown is set to 'ES'. On the right, a developer tools window titled 'Aplicación' is open, showing network traffic for 'localhost:4200'. It displays session storage items: 'esrinaUser' (value: 'U2FsdGvkX1+R9OTKKF/Dv9QES00rXlbeA...'), 'esrinaUserMail' (value: 'user6@gmail.com'), and 'seeds' (value: 'U2FsdGvkX1/OZX+dPBCmjYhF=syOda9...'). Below the navigation bar, the main content area has a title 'ESBRINA CHAIN - Sistema de Conocimiento Colaborativo.' and a subtitle 'Compartiendo el conocimiento de todos usando tecnología Blockchain.' There is a yellow graphic with silhouettes of people and text 'PREGUNTAS' and 'RESPUESTAS'. A 'Contraseña' input field and a 'Wallet sin Metamask' button are present.

Figura 46: Fase 4: El usuario esta registrado dispone de wallet y utiliza el wallet integrado.

The screenshot shows the ESRINA CHAIN application interface with the Metamask extension open. The main content area has the same title and subtitle as Figure 46. Below them is a purple graphic with silhouettes of people and text 'PREGUNTAS' and 'RESPUESTAS'. A 'Login con Metamask' button is visible. To the right, the Metamask extension window is shown, displaying a welcome message '¡Bienvenido de nuevo!' and a note 'La Web descentralizada espera'. It includes fields for 'Contraseña' and 'Desbloquear', and links for '¿Olvidó su contraseña?' and '¿Necesita ayuda? Comuníquese con Soporte de Metamask.'

Figura 47: Fase 3: El usuario esta registrado dispone de wallet y utiliza el wallet de Metamask.

Si el usuario, utiliza la opción ‘**Registro**’ del menú horizontal situado en la parte superior de la pantalla, ver figura 48, dispondrá de la opción de eliminar su cuenta y dejar de estar registrado en la aplicación web y el backend.



Figura 48: Fase 4: Cualquier usuario registrado puede darse de baja del sistema.

5.2 Crear una pregunta.

Cuando el usuario está registrado y dispone de wallet está en condiciones para utilizar el sistema colaborativo de conocimiento **Esbrinachain**. En la figura 49, se muestra la pantalla el interfaz de usuario de la aplicación web de **Esbrinachain**. Se ha mantenido la consola de programación a efectos de mostrar detalles del proceso de **Login** en este manual técnico, pero en ningún caso es necesario para un usuario de la aplicación.

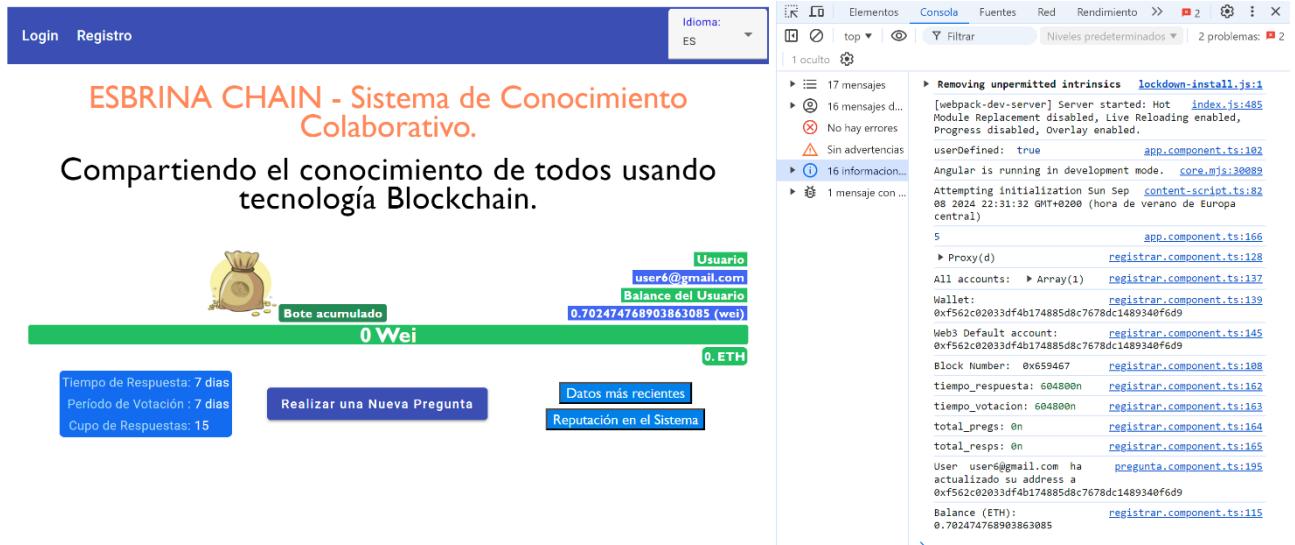


Figura 49: Interfaz de usuario de la aplicación web del sistema **Esbrinachain**.

Como se ha podido comprobar en la figura 49, el sistema **no tiene preguntas registradas, ni efectivo en el Bote** (balance del contrato inteligente). El usuario actual es user6@gmail.com, que dispone de un balance efectivo de **0.70 wei**. Si efectúa una pregunta, el **tiempo de respuesta** fijado por el sistema para responder es de **7 días**, cuando disponga de respuestas, **pasada una semana**, entrará en **periodo de votación** y el **cupo de respuestas** actual está fijado en **15 respuestas** como máximo.

Para crear una pregunta, el usuario sólo necesita 3 requisitos:

1. Efectivo suficiente en su billetera digital para **efectuar la transacción** más el **coste de la recompensa** ofrecida por la pregunta, que se enviará como valor en la transacción.
2. Pulsar el botón '**Realizar una Nueva Pregunta**'
3. Facilitar el **enunciado** y la **recompensa** ofrecida por una respuesta.

En la figura 50, se muestra la ventana emergente que permite entrar el enunciado de la pregunta y la recompensa ofrecida por el autor de la pregunta. En este caso estamos utilizando el wallet de Metamask y emerge para enviar la transacción. Una vez enviada la transacción, el usuario debe esperar un tiempo breve (o en algún caso, no tan breve) a que la red blockchain la procese y la registre en un bloque de la cadena de bloques. En cuanto este proceso finaliza, quedará registrada en el sistema. Si la transacción falla, se deberá probar de nuevo.

El interfaz de la aplicación muestra todos los metadatos de la pregunta:

1. Autor
2. Fecha de creación (la fecha de creación marca el inicio del tiempo de respuesta).
3. Estado de la pregunta. Inicialmente está **abierta** a respuestas.
4. Recompensa ofrecida.
5. El botón para incrementar la recompensa inicial. Se dispone de tres tentativas.
6. Fecha de votación, calculada sumando fecha de creación y periodo de respuesta fijado en el sistema.

The screenshot shows the Esbrinachain application interface. At the top, there is a navigation bar with 'Login' and 'Registro' buttons, and a language selection dropdown set to 'ES'. Below the header, the title 'ESBRINA CHAIN - Sistema de Conocimiento Colaborativo.' is displayed. A central modal window is open, titled 'Editando nueva pregunta...'. Inside the modal, the question text is '¿De qué color tienen los ojos los delfines?' and the reward amount is '50' (wei). The modal has 'Cancelar' and 'Confirmar' buttons. To the right of the modal, a sidebar displays the user's information: 'Usuario' (user6@gmail.com), 'Balance del Usuario' (0.702474768903863085 (wei)), and '0. ETH'. At the bottom of the screen, a MetaMask wallet interface is visible, showing an account balance of 0.000001 SepoliaETH and a transaction history.

Figura 50: Proceso de crear una pregunta en el sistema **Esbrinachain**.

En la figura 51, se puede visualizar el interfaz de usuario con la pregunta creada y la recompensa incluida en el bote y el balance del usuario actualizado.

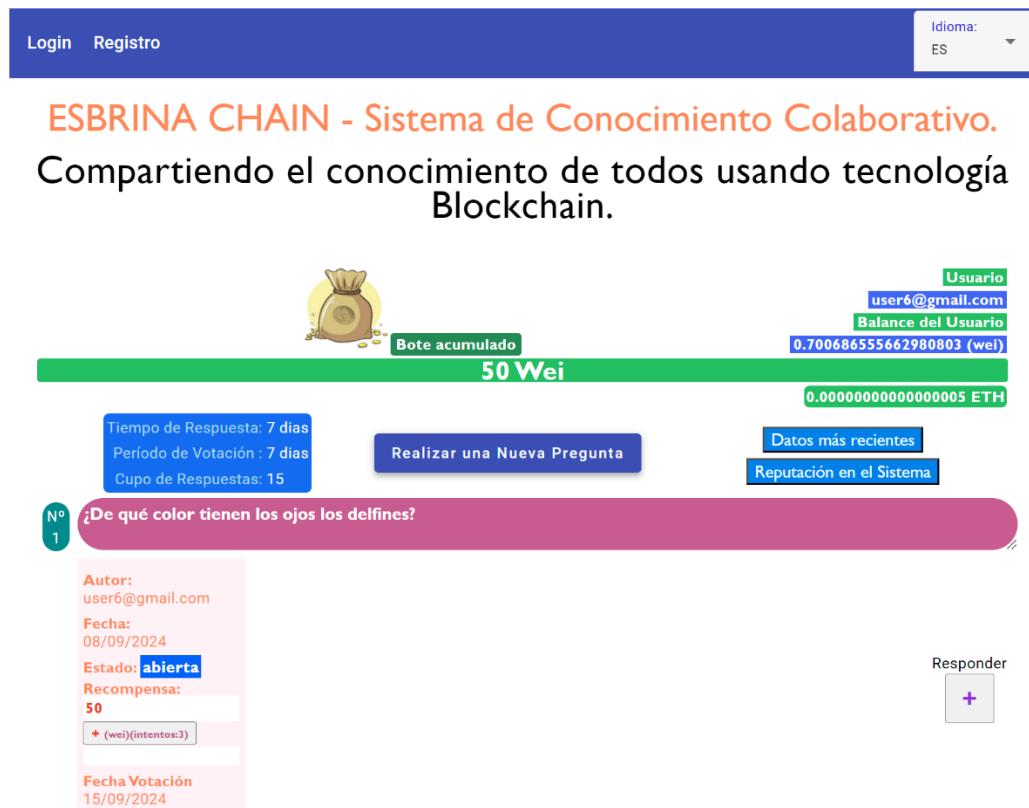


Figura 51: Visualización de la pregunta creada, sus metadatos y el resto de datos actualizados.

Internamente, el aplicativo genera una transacción, la firma y la envía a la blockchain, y tan solo cuando obtiene el ‘**receipt**’ de la misma la registra en el backend y la muestra por pantalla, ver figura 52:

El usuario dispone de **0.7024 wei en su wallet** y de la figura 50, sabemos que aproximadamente gastará **0.0018**. Se calcula el **Gas Price**, en un valor de **5810754596 wei**
Se calcula el **Gas estimado** en un valor de **308279 wei**
Emisor de la transacción:

Emisor de la transacción:

Receptor: contrate intelecto;

Receptor: contrato intelectual.

Receipt:
Devuelve `blockNumber` y `transactionIndex` que usaremos

Se retorna un vector **logs** con un campo **topic** que retorna los valores del evento ‘**PreguntaCreada**’ y recogemos el valor que nos interesa, el **índice de la pregunta**:
Logs[0].topics[1] cuyo valor es **1**.

`Logs[0].topics[1]` cuyo valor es `1`. Con este dato podemos registrar

Con este dato podemos registrar la pregunta en el backend, en el mismo orden en que se registra en la blockchain, utilizando la estructura Pregunta de Firestore Cloud. Como varios usuarios pueden hacer peticiones a la vez, nos permite que haya sincronización.

Figura 52: Detalle de la transacción para crear una pregunta.

5.3 Crear una respuesta.

Una vez que el sistema muestra la lista de preguntas, si están en estado ‘**abierta**’ los usuarios del sistema pueden responderlas. Como ya vimos en anteriores secciones, se ha de cumplir:

1. Que el usuario tenga suficiente efectivo para enviar la transacción.
2. Que no sea el autor de la pregunta.
3. Que no haya respondido antes a la pregunta.

Pulsando el botón ‘+’ con etiqueta ‘**Responder**’, se nos muestra la ventana emergente que permite dar el **enunciado de la respuesta** y **enviar la transacción de la respuesta**. Si resulta exitosa, se mostrará por pantalla después de ser registrada en el backend e incluida en la lista de respuestas. Vemos el proceso en la figura 53, y el detalle de la línea de notificación que informa al usuario del proceso de la transacción en la figura 54.

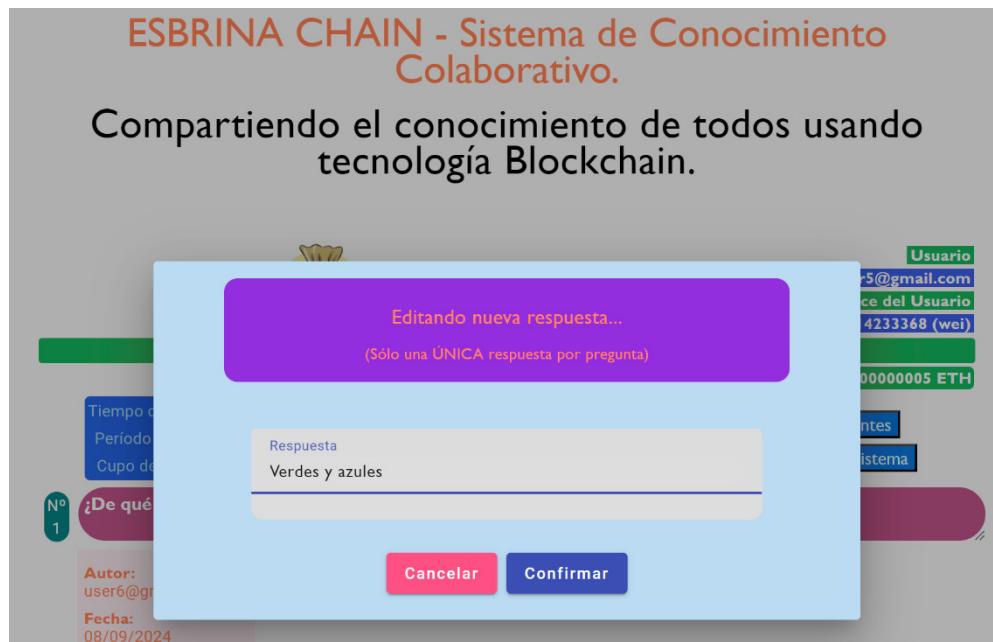


Figura 53: Interfaz de la aplicación web al crear una respuesta.



Figura 54: Línea de notificación del Interfaz web de Esbrinachain.

Si la transacción tiene éxito y se graba en un bloque de la cadena de bloques, como en el caso de crear una pregunta, recibiremos el ‘**receipt**’ del cual podremos extraer el índice de respuesta indicado en el evento ‘**RespuestaCreada**’. Este es el id de respuesta usado al actualizar la respuesta y que juntamente con el `blockNumber` y `transactionIndex` de la pregunta nos permitirá enlazar en el backend las preguntas y las respuestas. En la figura 55, se observa que la respuesta se ha creado y queda asociada a su pregunta.

ESBRINA CHAIN - Sistema de Conocimiento Colaborativo.

Compartiendo el conocimiento de todos usando tecnología Blockchain.

The screenshot shows a user interface for a blockchain-based poll application. At the top right, there's a green bar with the text "Usuario" (User), "user6@gmail.com", "Balance del Usuario" (User Balance), and the value "0.700686555662980803 (wei)". Below this, a large green banner displays "Bote acumulado" (Accumulated Reward) and "50 Wei". To the left of the banner is a yellow illustration of a money bag with coins. On the right side of the banner, it says "0.000000000000000005 ETH". In the center, there's a button labeled "Realizar una Nueva Pregunta" (Create a New Question). To the left of the banner, there are three boxes: "Tiempo de Respuesta: 7 días" (Response Time: 7 days), "Período de Votación : 7 días" (Voting Period: 7 days), and "Cupo de Respuestas: 15" (Response Capacity: 15). To the right of the banner, there are two more boxes: "Datos más recientes" (Latest Data) and "Reputación en el Sistema" (Reputation in the System). The main content area features a question: "¿De qué color tienen los ojos los delfines?" (What color are the eyes of dolphins?). Below the question, there's a box containing author information: "Autor: user6@gmail.com", "Fecha: 08/09/2024", "Estado: abierta" (open), "Recompensa: 50", and a button "+ (wei)(intentos:3)". At the bottom, there's a section for voting with the text "Fecha Votación 15/09/2024", a "Responder" (Answer) button, and a box showing "0 votos" (0 votes) with a "+" button.

Figura 55: Respuesta creada con índice 1 en el interfaz de Esbrinachain.

El detalle de la transacción efectuada lo podemos observar en la figura 56:

Receipt:

Devuelve **blockNumber** y **transactionIndex** que usaremos para registrar la respuesta en el backend.

Se retorna un vector **logs** con un campo **topic** que retorna los valores del evento '**RespuestaCreada**' y obtenemos el **segundo valor** que es el que nos interesa, el **índice de la respuesta**:

`Logs[0].topics[2]` cuyo valor es **1**.

Con este dato podemos registrar la respuesta en el backend en el mismo orden en que se registra en la blockchain, utilizando la estructura Respuesta de Firestore Cloud.

Figura 56: Detalle de la transacción para crear una nueva respuesta.

5.4 Iniciar una votación y finalizar una votación por parte del administrador.

El periodo de respuesta de una pregunta está predefinido y es el que se utiliza en el contrato inteligente, pero a efectos de presentación del proyecto y de este documento técnico, se ha dispuesto que el administrador del contrato tenga la **posibilidad de iniciar la votación** de una respuesta y que también pueda finalizarla.

Iniciar una votación, implica entre otras operaciones cambiar el estado de la pregunta a votar a estado ‘**votando**’. El lector, ya ha revisado el código y conoce como está diseñado el código de interfaz. Cuando un usuario intente responder se le notificará que la pregunta ha cambiado de estado, y el diseño del interfaz cambiará para facilitar la votación.

Suponemos la situación inicial de la figura 57, varios usuarios han creado una respuesta, y o bien se acaba el periodo de respuesta o bien el administrador inicia la votación (sólo el administrador puede hacerlo tal como indica el código del contrato inteligente), el interfaz cambiará al aspecto de la figura 58:

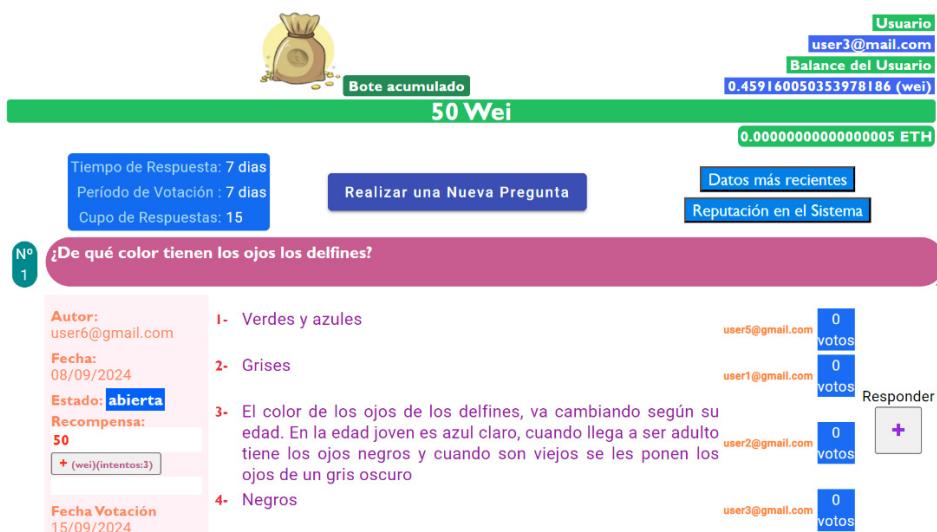


Figura 57: Situación inicial antes del inicio de la votación

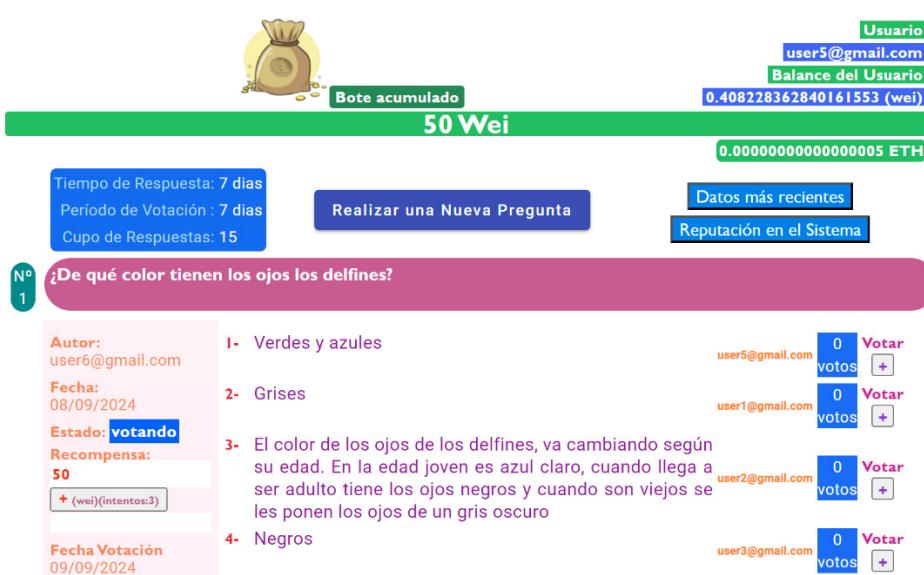


Figura 58: Cambio de estado de una pregunta de ‘abierta’ a ‘votando’.

5.5 Votar un respuesta.

En estado ‘**votando**’, cualquier usuario con wallet y acceso al sistema **Esbrinachain**, puede votar en cualquiera de las preguntas por una respuesta, siempre que disponga de saldo suficiente para enviar la transacción y:

1. No sea autor de la pregunta.
 2. No haya votado antes la respuesta.
 3. No sea autor de una respuesta.

En la figura 59, se muestra la acción de voto del usuario user4@gmail.com que cumple todos los criterios para poder votar por la pregunta Nº1. Se puede observar, que en la línea de notificación el usuario de la aplicación está informado de la evolución de todo el proceso.



Figura 59: Proceso de Votación de una pregunta en el sistema Esbrinachain.

Internamente, la aplicación envía una transacción para ejecutar la función del contrato inteligente ‘votarRespuesta’. En la figura 60, vemos el recibo (**receipt**) de la transacción:

Receipt:

En este caso nos interesa evaluar el vector de **logs** : Si retorna un vector **logs** sin elementos `Logs[:]`, entonces la votación ha ido bien, y se incrementa su contador en un voto.

Si el vector de **logs** tiene 1 elemento, indica que o bien se ha generado un evento '**FinalVotacion**' porque el periodo de votación ha finalizado, o bien uno '**PreguntaAnulada**' porque se intenta votar fuera de tiempo una pregunta con respuestas pero sin votos.

Cuando el voto es efectivo, podemos registrar el voto en el backend.

Figura 60: Receipt de la transacción de voto.

5.6 Resolución de la votación.

Cuando el tiempo de votación se ha agotado (por defecto 7 días), cualquier intento de voto desencadena un cambio de estado de la pregunta de ‘**votando**’ a ‘**consulta**’. En este proceso de cambio el contrato inteligente efectúa una **resolución de votación**. En la resolución de votación, se contabilizan los votos y se determina que usuario o usuarios han recibido el máximo número de votos por sus respuestas. Se contabiliza el pago **equitativo de la recompensa**, y el contrato inteligente envía del ‘**bote**’ o depósito del contrato la cantidad a cada uno, al mismo tiempo que se le **recalcula su reputación en el sistema**.

En la figura 61, se observa cómo queda registrada la pregunta en el contrato, con el campo ‘**votada**’ a **true**:

preguntas	1
0:	uint8: estado 2
1:	uint256: recompensa 50
2:	uint256: creada 1725829260
3:	uint256: fecha_votacion 1725836448
4:	address: autor 0xF562C02033DF4b174 885D8c7678dC1489340F6d9
5:	string: enunciado ¿De qué color tienen los ojos los delfines?
6:	bool: votada true

Figura 61: Variable ‘**preguntas**’ en el contrato inteligente **Esbrinachain** para la primera pregunta

Para el usuario de la aplicación todo este proceso es transparente y sólo visualiza la **eliminación de los contadores de voto** y el **cambio de estado de la pregunta**. En estado de consulta, las preguntas disponen de un botón que permite consultar la pregunta con un **coste del 50% de la recompensa**, excepto para el autor de la pregunta que puede consultarla sin coste.

En la figura 62, se observa cómo queda en el interfaz de **Esbrinachain** la pregunta en estado de consulta:

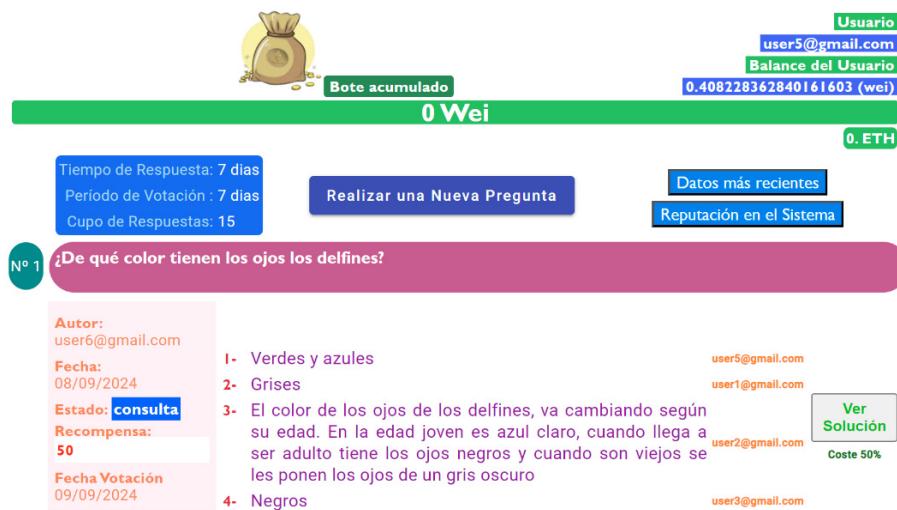


Figura 62: Pregunta en estado de consulta en el sistema **Esbrinachain**.

En la figura 62, hay tres detalles para tener en cuenta:

1. La resolución de la votación se ha realizado y el **bote ha pasado de 50 a 0**.
2. Las preguntas en estado de ‘**consulta**’, **no permiten incrementar su recompensa**.
3. El usuario actual, es el **autor de la respuesta más votada**, y en su **balance a recibido los 50 wei**.

5.7 Cálculo de la reputación del usuario en el sistema.

En el apartado 4.1, ya se expuso como se calculaba la reputación del usuario cuando resulta ser el autor de la respuesta más votada de una pregunta. En cuanto al caso de uso, cada usuario puede consultar su reputación y **compararla con la del resto de usuarios**, pulsando el botón con la etiqueta ‘Reputación en el Sistema’, vemos en la figura 63, la ventana emergente que se muestra con el usuario resaltado.



Figura 63: Ranking por reputación en el sistema **Esbrinachain**.

En el estado de este caso de uso, sólo existe una pregunta, y el usuario [user5@gmail.com](#) es el autor de la respuesta más votada. Por eso aparece el primero destacado con una reputación de **10**, todo lo que ha contestado ha sido lo más votado.

5.8 Incremento de la recompensa de una pregunta (3 tentativas).

En este caso de uso, se trata de hacer efectivo un incremento en el campo ‘recompensa’ de la pregunta Nº2. Tan solo se permiten 3 tentativas y cuando se agotan el botón desaparece. En la figura 64, vemos la tentativa y en la figura 65, su resultado.

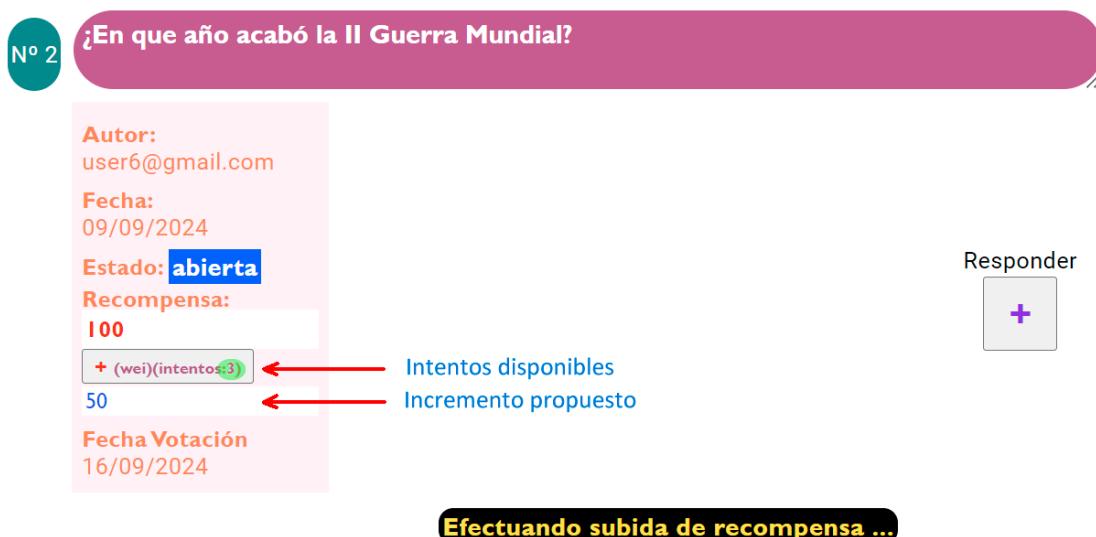


Figura 64: Primera tentativa por incrementar la recompensa.

Nº 2	¿En qué año acabó la II Guerra Mundial?
Autor: user6@gmail.com	
Fecha: 09/09/2024	
Estado: abierta	
Recompensa: 150	
+ (wei)(intentos:2)	
Fecha Votación 16/09/2024	
Responder 	

Figura 65: Resultado de la primera tentativa de incremento de recompensa.

Cuando se resuelve la transacción, la recompensa queda incrementada y el contador resta una tentativa.

5.9 Detalles de visualización en el interfaz.

En la aplicación web del sistema **Esbrinachain** se tiene informado al usuario con aspectos relacionados al sistema. Este es el objetivo primordial de este caso de uso. Información como el **tiempo de respuesta** a una pregunta, la **duración** de una votación, el **número máximo de respuestas** posible para una pregunta, la **cantidad existente en el 'Bote'** del sistema, el **balance del usuario** conectado, son datos que hay que mantener actualizados para una alta calidad de uso del sistema de conocimiento (ver figura 66).



Figura 66: Puntos de información de interés para el usuario de Esbrinachain.

5.10 Soporte de Idioma.

También se ofrece al usuario la posibilidad de disponer el interfaz en varios idiomas. En futuras versiones se podría mantener un sistema de traducción de las preguntas y almacenarlas en varios idiomas o ver una solución óptima en rendimiento.



Figura 67: Disponibilidad de tres idiomas para el interfaz de la aplicación web.

5.11 Línea de estado para información del usuario.

Existe una línea de notificaciones al usuario, originada por la potencial latencia de las transacciones. También se ha utilizado para informar de otras informaciones como el **Precio del Gas** o el **Gas estimado**. La línea de notificaciones se puede usar desde el componente preguntas y desde el componente respuestas.

Queda para futuras ediciones, una mejor solución para evitar que el **refresco de pantalla** para actualizar la lista de preguntas **no borre la línea de notificación**. Quizás una ventana emergente podría ser una mejor solución. En un principio se configuro una **actualización manual de la información en pantalla** (botón ‘**Datos más recientes**’), para evitar tener que automatizarlo, pero para el usuario quizás es mejor automático, por esa razón se han conservado las dos opciones en la versión final.

5.12 Consulta mediante pago de la respuesta más votada.

Tal como se ha mencionado en el apartado 5.6, en estado de ‘**consulta**’, las preguntas disponen de un botón que permite consultar la respuesta más votada de una pregunta con un **coste del 50% de la recompensa** (excepto para el autor de la pregunta que puede consultar sin coste). Cuando se ejerce esta opción se dispone de **15 segundos** en los que la respuesta o respuestas más votadas son **resaltadas en fondo verde**. Luego el sistema las oculta.



Figura 68: Visualización del resultado de la votación de una pregunta en Esbrinachain.

5.13 Configuración de un backend de datos en Firestore.

Este caso de uso ha quedado descrito ya en el apartado 4.3, pero no está de más comentar que es una buena prestación, que permite no almacenar en la blockchain datos que no son prioritarios o críticos.

Finalmente recordar, que se puede ver la demostración del proyecto visualizando el video de presentación del proyecto que está publicado en Github.

06. CONCLUSIONES

En este apartado de conclusiones finales, no se va a realizar ningún comentario técnico más, tan solo destacar lo que ha sido necesario y fundamental para realizar un proyecto como este: pasión, tenacidad y paciencia. Pasión por la cadena de bloques, por conocer tecnológicamente algo nuevo y que desconocía. Tenacidad porque en algún momento era bastante probable no acabarlo. Y paciencia porque este sistema de aprendizaje que gobierna la informática por ‘prueba y error’, la necesita.

La realización del proyecto **Esbrinachain** ha cumplido con creces las expectativas iniciales. La sensación personal es de satisfacción por haber acabado el proyecto y haber tenido la oportunidad de hacerlo como quería, salvando problemas iniciales que lo hacían complicado en sus inicios. Se ha podido aplicar todo el conocimiento adquirido en el Master e incluso ampliarlo, con mucha dedicación y obteniéndolo de la experiencia práctica. El autor de este proyecto, aunque dispone de una sólida base en varios ámbitos de la informática, había programado en Javascript, HTML, CSS y scripts de servidor en un periodo incipiente de Internet. Desconocía Angular Framework, no se había conectado a ninguna red Blockchain, no tenía experiencia en web3 y menos en la programación de contratos inteligentes con Solidity. Tampoco había utilizado servicios de la nube como Firebase/Firestore, y a pesar de todo este ‘cocktel’, se ha podido finalizar el proyecto gracias a las facilidades dadas por **EBIS Techschool**. Se ha tenido que recorrer una curva de aprendizaje nada sencilla por la cantidad de lenguajes y entornos modulares que forman parte del proyecto. La conclusión, es que ha valido mucho la pena, y ánimo a cualquier lector que le haya interesado, a probarlo.

07. AGRADECIMIENTOS

Dar las gracias a los profesores de este módulo de máster que han hecho posible la elaboración de este proyecto con su dedicación, conocimiento y esfuerzo en todas las clases.

08. Anejo I : Instalación de Metamask Wallet

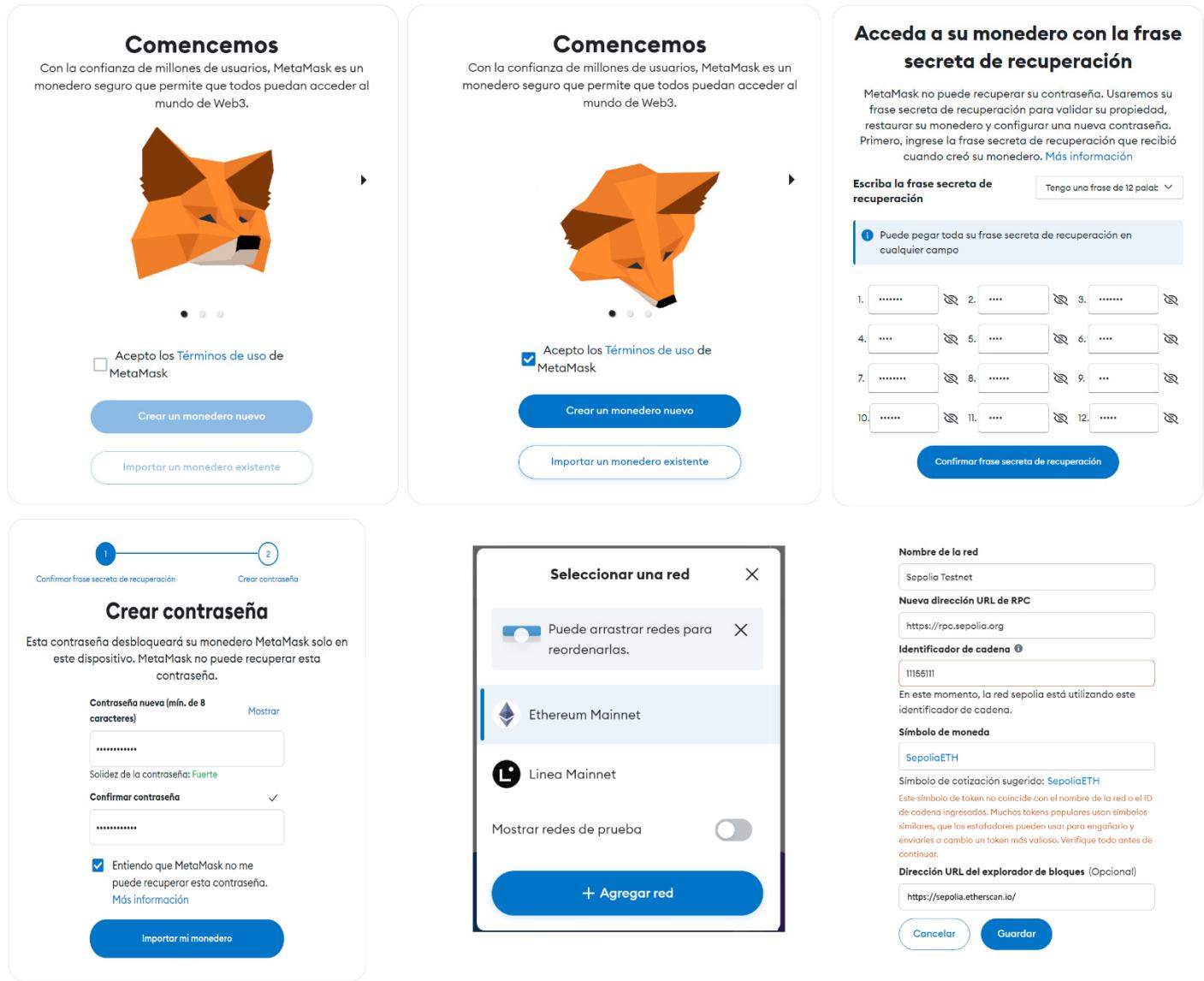


Figura 69: Cuadro resumen del proceso de instalación de Metamask

Para el uso del wallet de Metamask en la aplicación **Esbrinachain**, este ya debe de estar instalado en su navegador de Internet. Proceda a su descarga y compruebe que no tiene complicación alguna.

En la figura 69, hemos añadido el conjunto de pantallas de configuración del wallet de Metamask en la versión disponible al editar este documento, y la configuración de la testnet Sepolia sobre la que funciona **Esbrinachain**.