Introducción a la Programación de Smart Contracts en Solidity

Máster Inter-Universitario en Ciberseguridad

Memoria del Ejercicio 5: Contrato de *Ticketing para Conciertos*

Autores:
Aarón García Filgueira (aaron.gfilgueira@udc.es)
Sergio Luaces Martín (s.luaces@udc.es)

Universidade da Coruña

27 de octubre de 2025

Índice

1. Introducción

El presente trabajo forma parte de la *Práctica 1* de la asignatura *Tecnologías de registro distribuido y Blockchain*, y tiene como objetivo diseñar e implementar un primer contrato inteligente funcional. Con ello se busca entender, de forma práctica, cómo una blockchain puede automatizar procesos de negocio, eliminar intermediarios y asegurar la trazabilidad de las transacciones mediante reglas programadas.

Para este propósito se desarrolló el contrato ***Ticketing para Conciertos***, destinado a la emisión y gestión digital de entradas para eventos. En el sistema propuesto, cada ticket se registra en la blockchain con un identificador único y un propietario, convirtiéndose en un activo inmutable, verificable y no transferible.

El desarrollo se realizó en el lenguaje *Solidity* utilizando el entorno *Remix IDE* y la máquina virtual *Remix VM (Prague)*. Las pruebas funcionales confirman el correcto cumplimiento de los requisitos definidos: emisión, consulta y validación de tickets, preservando la integridad de la información y el control exclusivo del organizador.

En esta memoria se presenta el escenario de aplicación, el diseño y los elementos técnicos del contrato inteligente, junto con las pruebas realizadas, los resultados obtenidos y los diagramas de caso de uso que reflejan la interacción de los distintos actores con el sistema.

2. Análisis y definición del escenario

El escenario planteado corresponde a la **gestión digital de entradas para eventos**, un proceso que tradicionalmente presenta problemas de confianza y trazabilidad.

En los sistemas convencionales, la verificación de entradas depende de bases de datos centralizadas o intermediarios, lo que permite la duplicación, falsificación o reventa no autorizada. Además, el control de acceso suele requerir una entidad organizadora que valide manualmente la autenticidad de las entradas.

La solución propuesta aprovecha la funcionalidad de una **blockchain pública como Ethereum**, que proporciona un registro distribuido, inmutable y transparente. Mediante el uso de contratos inteligentes, las operaciones de emisión y validación se automatizan y se ejecutan sin intervención de terceros. Cada ticket se representa como una estructura de datos almacenada en la cadena, asociada a una dirección de usuario y a un estado de validez. De este modo, la red actúa como una autoridad de confianza compartida que garantiza la autenticidad de cada entrada.

Este tipo de aplicación requiere necesariamente una blockchain pública por varias razones:

- **Inmutabilidad y prueba de propiedad:** una vez emitido un ticket, su existencia y propietario quedan registrados de forma permanente y verificable por cualquier nodo de la red.
- **Transparencia y trazabilidad:** todos los cambios de estado (emisión, validación o invalidación) generan eventos públicos consultables en la cadena.
- **Descentralización y eliminación de intermediarios:** no se necesita un servidor central ni una base de datos administrada por el organizador, ya que la red Ethereum actúa como un soporte confiable.

En este contexto, el contrato *Ticketing* define una lógica de negocio simple pero completa: un único organizador emite entradas digitales, las asigna a usuarios y controla su validez en tiempo real, todo ello con las garantías de integridad y autenticidad que ofrece la infraestructura de Ethereum.

3. Diseño

3.1. Caso de uso y lógica de negocio

El caso de uso definido representa el proceso de emisión, gestión y control de entradas digitales para eventos. El contrato *Ticketing* asume el papel de una autoridad confiable que automatiza todas las operaciones relacionadas con el ciclo de vida de cada ticket. El organizador es el único actor autorizado para emitir nuevas entradas, asignarlas a los asistentes y validar su uso el día del evento.

Cada ticket se almacena en la blockchain como una estructura con cuatro atributos: un identificador único, la dirección del propietario, el nombre del evento y un estado de validez. Esta información queda registrada en la cadena de bloques, garantizando su inmutabilidad y su trazabilidad a lo largo del tiempo.

El flujo de negocio es sencillo y se resume en tres fases:

1. **Emisión:** el organizador crea un ticket y lo asigna a un usuario mediante la función `issueTicket`. El sistema genera un evento que deja constancia pública de la operación.
2. **Validación:** cuando el asistente accede al evento, el organizador o el personal designado ejecuta `verifyTicket` para marcar la entrada como usada. A partir de ese momento, el ticket deja de ser válido.
3. **Invalidación:** en caso de cancelación o error, el organizador puede anular manualmente una entrada con `invalidateTicket`, manteniendo registro del cambio de estado.

Las consultas disponibles (`getTicket`, `ownerOf`, `isValid`, `ticketsOf`) permiten a cualquier usuario verificar la autenticidad y el estado de sus entradas sin depender de intermediarios. Con ello se consigue un sistema transparente y resistente a manipulaciones, donde la lógica de negocio se ejecuta automáticamente a través del contrato inteligente.

3.2. Contenido del contrato inteligente

El contrato define la estructura de datos y las reglas que controlan todo el ciclo de vida de una entrada digital. La información principal se organiza en una estructura "`Ticket`", que contiene los campos `id`, `owner`, `eventName` e `isValid`. Estos registros se almacenan en un `mapping` indexado por el identificador del ticket, lo que permite acceder a la información de forma directa. Un segundo `mapping` mantiene la lista de identificadores asociados a cada usuario.

En la Tabla 1 se resumen los componentes y funciones principales del contrato, junto con sus parámetros de entrada y salida cuando corresponden.

La lógica se mantiene deliberadamente sencilla para asegurar una ejecución determinista y un consumo mínimo. Cada operación que modifica el estado del contrato genera un evento, lo que facilita la auditoría y la trazabilidad en la blockchain. La ausencia de funciones de transferencia impide la reventa o el intercambio de entradas, garantizando un control total del acceso por parte de la organización.

3.3. Usuarios del sistema y diagrama de casos de uso

El sistema define dos tipos principales de usuario: el **organizador** y el **asistente**. Ambos interactúan con el contrato inteligente, pero con permisos y objetivos muy distintos. Además, en un escenario real se contemplaría un actor secundario, el **personal de control de acceso**, que actúa en nombre del organizador durante el proceso de validación.

El **organizador** es el propietario del contrato y tiene control exclusivo sobre las operaciones críticas.

El **asistente**, por su parte, representa al usuario final que posee uno o varios tickets y puede consultar su estado o autenticidad sin necesidad de intermediarios.

Elemento	Entrada	Salida	Descripción
`organizer`	—	Dirección	Propietario del contrato; único con permisos de emisión y validación.
`struct Ticket`	—	—	`id`, `owner`, `eventName`, `isValid`.
`issueTicket()`	`address _to`, `string _eventName`	`uint256 id`	Emite nuevo ticket y lanza evento `TicketIssued`.
`verifyTicket()`	`uint256 id`	—	Marca ticket como usado; sólo organizador. Evento `TicketValidated`.
`invalidateTicket()`	`uint256 id`	—	Invalida ticket no usado (cancelación/error). Evento `TicketInvalidated`.
`getTicket()`	`uint256 id`	`(id, owner, eventName, isValid)`	Devuelve datos completos del ticket.
`isValid()`	`uint256 id`	`bool`	Indica si un ticket sigue siendo válido.
`ticketsOf()`	`address user`	`uint256[]`	Lista de identificadores asignados a un usuario.
`ownerOf()`	`uint256 id`	`address`	Propietario del ticket.
`isOwnerOf()`	`uint256 id`, `address user`	`bool`	Verifica titularidad del ticket.
`onlyOrganizer`	—	—	Modificador de acceso para funciones sensibles.
`TicketIssued`, `TicketValidated`, `TicketInvalidated`	—	Evento	Cambios de estado trazables en la cadena.
`NotOrganizer`, `TicketDoesNotExist`, `TicketAlreadyUsedOrInva`	—	Error	Errores personalizados para control y claridad.

Cuadro 1: Elementos, entradas y salidas principales del contrato *Ticketing*

El **personal de control de acceso** intervendría en el momento del evento para validar las entradas, empleando la misma lógica que el organizador pero con un propósito puramente operativo.

Los diagramas de caso de uso elaborados ilustran estas interacciones. En ellos se reflejan las funciones disponibles para cada actor y las dependencias entre operaciones, destacando las relaciones *include* entre las funciones de verificación o invalidación y la consulta previa del ticket.

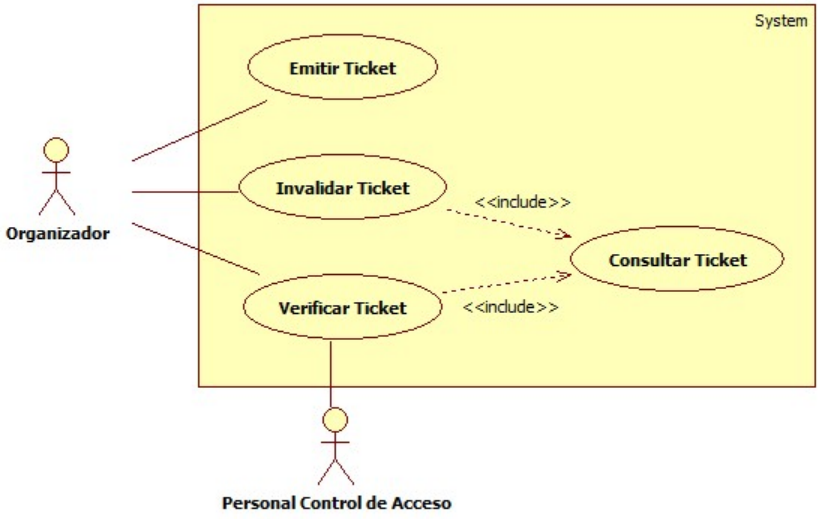

Figura 1: Casos de uso: Organizador y Personal Control de Acceso

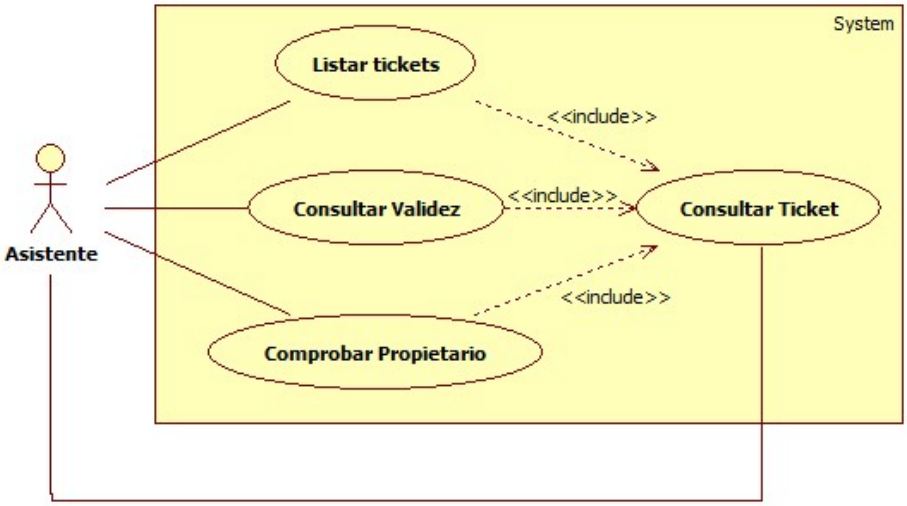

Figura 2: Casos de uso: Asistente

En el primer diagrama (Figura 1) se observa cómo el organizador gestiona el ciclo completo de las entradas, desde su creación hasta su invalidación, con control total sobre los estados de validez.

El segundo diagrama (Figura 2), por su parte, muestra las consultas accesibles para un asistente: listar tickets, comprobar propietario y validar su estado. Estas acciones dependen del caso base *Consultar Ticket*, que centraliza la obtención de información dentro del sistema.

Ambos diagramas evidencian una separación clara de responsabilidades y un flujo coherente con la lógica definida en el contrato inteligente, manteniendo la seguridad de las operaciones y la trazabilidad de todas las acciones.

4. Pruebas y resultados

Las pruebas se realizaron en el entorno *Remix IDE*, empleando el compilador *Solidity 0.8.30* y la máquina virtual *Remix VM (Prague)*.

El despliegue del contrato asigna automáticamente al creador la condición de *organizer*. A partir de ahí se verificó el correcto funcionamiento de las operaciones principales:

- **Emisión de tickets:** el organizador ejecuta `issueTicket()`, generando un identificador único y el evento `TicketIssued`.
- **Validación e invalidación:** al ejecutar `verifyTicket()` o `invalidateTicket()`, el estado del ticket pasa a `isValid = false`, emitiendo el evento correspondiente.
- **Consultas:** las funciones `getTicket()`, `isValid()` y `ticketsOf()` devuelven la información esperada para distintos usuarios.

En todos los casos se confirmó el comportamiento previsto: las restricciones de acceso se aplicaron correctamente y los eventos reflejaron con precisión cada cambio de estado.

Aunque las pruebas se realizaron de forma manual, el contrato sería fácilmente integrable en entornos de pruebas automatizadas mediante *Hardhat* o *Truffle*, permitiendo definir test unitarios sobre cada función y verificar su comportamiento en redes locales o testnets públicas.

5. Conclusiones

El desarrollo del contrato *Ticketing* permitió aplicar de forma práctica los conceptos fundamentales de la tecnología blockchain y de los contratos inteligentes en Ethereum. Su diseño demuestra cómo un proceso cotidiano, como la gestión de entradas, puede automatizarse de manera segura y transparente, eliminando la necesidad de intermediarios y reduciendo el riesgo de fraude o duplicación.

El resultado es un sistema funcional muy sencillo, que refleja las ventajas de la descentralización en la verificación de activos digitales. A futuro, podría ampliarse con nuevas funcionalidades, como la reventa controlada de tickets o la integración con estándares ERC-721, manteniendo los principios de autenticidad, control y confianza que se buscaban en el modelo actual.