Smart Exam

Aplicación descentralizada para la gestión de exámeness en la Blockchain.

Equipo:

Luis Corbacho Flores
Javier Español Alfonso
Pedro Otero García

December 24, 2023

Índice

1 Introducción

1.1 Motivación

En la sociedad actual son frecuentes las pruebas y exámenes oficiales para evaluar los conocimientos y habilidades de las personas en un campo concreto. Son diversos y abundantes los campos en los que se requiere una certificación oficial. Por ejemplo: para trabajar en la empresa pública como funcionario, acceder a la universidad (tanto pública como privada), exámenes teórico y práctico de conducción, de manipulador de alimentos, prueba de nivel de Inglés, etc.

En una situación ideal, el candidato debería tener la oportunidad de revisar la prueba una vez haya sido corregida, pero no en todos los casos sucede. Sin ir más lejos, en Galicia los alumnos no pueden ver su examen corregido hasta solicitar que el examen se revise (esto implica que su nota pueda disminuir en la revisión[1]). Aunque Galicia no es la única comunidad autónoma y en otros países como en Italia y su prueba análoga (Esame di Maturità) tienen el mismo problema. Esta no es una particularidad del sector público, puesto que en exámenes de entidades certificadoras privadas como la Universidad de Cambridge y sus prestigiosos certificados de nivel de Inglés no se permite a los candidatos revisar sus exámenes tras la corrección. Un individuo tiene el derecho a saber si su prueba fue entregada, inalterada y corregida pertinentemente, especialmente en los casos en los que se pone en juego un puesto de trabajo, el dinero de la prueba o su futuro académico.

Tener bien localizadas e identificadas las pruebas y sus respectivas correcciones no es solo importante para los candidatos, sino que también lo es para los organizadores de la prueba. Se pueden dar casos de falsificación de títulos como el reciente caso de una mujer que trabajó siete meses en el hospital de Berga (Barcelona) como médica sin tener título[2].

Teniendo en cuenta estas problemáticas se ha planteado la creación de una DApp (*Decentralized Application*[1]) que implementa un contrato inteligente[1] llamado **Smart Exam**[3]. Este contrato inteligente será el encargado de gestionar los datos de los exámenes para almacenarlos usando tecnologías como la *blockchain* de Ethereum [2] y mecanismos de almacenamiento descentralizado. El objetivo final es que todas las interacciones, tanto creación de los exámenes, inscripción, entrega de los candidatos, corrección y revisión de los resultados sean transparentes, rastreables e inmutables para todos los usuarios del servicio, tanto examinadores como examinados.

[1] https://ciug.gal/gal/faq-abau, preguntas 23 y 24.
[2] Noticia del periódico El País
[3] **Nota:** Cuando se use el termino *Smart Exam* se entiende que no solo se hace referencia a la aplicación descentralizada sino que también a todo el conjunto de datos y acciones que giran alrededor del examen y que se quiere gestionar mediante la DApp.

1.2 Definición del escenario

Una entidad certificadora (EC) que quiera realizar una prueba obtiene la propiedad de un contrato inteligente *Smart Exam*, ya sea porque ha pagado por la creación del contrato en la *blockchain* o porque el propietario anterior le cede su propiedad. Una vez EC figura como propietaria del contrato podrá asignar profesores al *Smart Exam* que serán los encargados de corregir las entregas de los estudiantes/candidatos[4]. La EC también podrá editar tanto el enunciado del examen como sus datos de creación (fecha de inicio de examen, precio de inscripción, duración, entre otros) que serán añadidos al *Smart Exam* de tal modo que el archivo con el enunciado se carga en IPFS [3] y se guarda su referencia CID (*Content IDentifier*) en el *smart-contract*. Los datos del examen solo se podrán editar antes de que este se inicie

Un estudiante que quiera realizar el examen primero tendrá que inscribirse en su *Smart Exam* antes de la fecha de realización del examen (con un margen de tiempo en caso de ser necesario) y abonando la tasa o precio de inscripción especificado en el *Smart Exam*. Cuando llegue la fecha de inicio del examen, si el estudiante está inscrito podrá añadir su archivo con las respuestas al *Smart Exam* (mismo procedimiento que para el enunciado) siempre y cuando el examen no haya finalizado.

Cuando el examen esté acabado los profesores podrán descargar los exámenes de los alumnos para corregirlos. Las correcciones se añadirán a *Smart Exam*, como el resto de archivos, para que puedan ser descargadas por los respectivos estudiantes. Cada estudiante solo podrá obtener la corrección de su examen.

1.3 Objetivos

La finalidad de la DApp *Smart Exam* es cumplir con los siguientes objetivos:

- Permitir seguir la trazabilidad de un examen desde que el enunciado se añade al *Smart Exam* hasta que un alumno recibe su corrección.
- Usar cripto-divisa como forma de pago.
- Almacenar los datos y los archivos relacionados con el *Smart Exam* de manera descentralizada.
- Garantizar la inmutabilidad de los datos almacenados.
- Encriptar los datos sensibles relacionados con el *smart-contract*.
- Para identificar a los actores que interactúan con el *smart-contract* solo se utilizará su dirección en la *blockchain* como medida de anonimización.

[4] A partir de este punto se usarán los términos estudiante y candidato con indistinción.

2 Estudio del estado del arte

En este estudio se examina el estado actual de las DApps basadas en *blockchain* (BC) para esta gestión, explorando sus ventajas, desafíos y aplicaciones prácticas. Se enfoca en la seguridad de datos, implicaciones éticas y posibles limitaciones tecnológicas, proporcionando una visión integral para el desarrollo futuro de soluciones más efectivas en este campo.

2.1 Análisis de soluciones parecidas

De entre las opciones existentes para la gestión de *smart-contracts* en DApps orientadas a la educación destaca **ODEM** [4] (*On-Demand Education Marketplace*). ODEM es una DApp que usa la *blockchain* para conectar estudiantes y docentes eliminando intermediarios en la educación. Esta DApp permite a los estudiantes encontrar cursos y profesores que se adapten a sus necesidades. Los contratos inteligentes en la *blockchain* facilitan acuerdos educativos personalizados, y la plataforma garantiza la autenticidad de certificados y credenciales. Su objetivo es hacer que la educación superior sea más accesible, económica y transparente. ODEM utiliza su propio *token* ERC-20 [5] para realizar pagos dentro de la DApp y los estándar ERC-900 [6] para que estudiantes y educadores apuesten con tokens ODE, evaluando así el interés en programas y cursos. Si alguien se retira, pierde los tokens apostados. Los ODE eliminan problemas de moneda fiduciaria. También emplea ERC-780 [7] para crear certificados digitales (ODEM-Credits) en la blockchain, dados a quienes completan cursos. Estos certificados incluyen datos del estudiante, programa, fecha y enlace al certificado físico en IPFS con contraseña.

Otra solución interesante es **Blockcerts** [8] (*Blockchain Certificates*) Inicialmente desarrollado por *Mit Media Lab*, *Blockcerts* provee un sistema descentralizado de credenciales. La red *blockchain* funciona como un proveedor de confianza, verificando credenciales de forma segura. *Blockcerts* puede ser usado en contextos académicos y profesionales para verificar la legalidad de un título o certificado y solicitar trabajo o continuar los estudios en otras instituciones.

Por último, una plataforma que llama la atención a la hora de gestionar contratos es **OpenLaw**, una plataforma legal basada en *blockchain* que simplifica la redacción y ejecución de contratos mediante contratos inteligentes en Ethereum. Permite redactar acuerdos legales en lenguaje natural, convertirlos en contratos inteligentes y desplegarlos en la *blockchain* para garantizar su integridad y ejecución automática. Ofrece una biblioteca de plantillas legales personalizables, facilita la colaboración en tiempo real entre partes interesadas y promueve la seguridad en la negociación, firma y ejecución de contratos. Su enfoque digitalizado elimina intermediarios, reduce errores humanos y mejora la eficiencia en procesos legales.

2.2 Comparativa con la solución propuesta

En esta sección se explicarán las similitudes y diferencias en cuanto a funcionalidades y ciberseguridad entre la solución propuesta y las mencionadas en el punto anterior.

2.2.1 Funcionalidades

OpenLaw permite crear contratos legales desde 0 en base a una plantilla en un programa *off-chain*[5] y posteriormente se da la opción de subirlo a la BC de Ethereum. En contraste con *Smart Exam* que usa el mismo contrato para todos los exámenes (creando una nueva instancia para cada examen) y que una vez el examen se ha creado todas las operaciones serán *on-chain*.

Por otro lado, *ODEM* puede ser vista como una red social que conecta alumnos con profesores y que emite certificados digitales de cursos propios, una versión en Web3 de plataformas didácticas como Coursera o edx, mientras que *Smart Exam* está pensado para gestionar pruebas de carácter oficial que requieren burocracia y un alto nivel de rigurosidad. Además con *Smart Exam* los estudiantes solo tendrían que pagar por la inscripción a un examen y no por interactuar con los profesores a modo de tutorías o lecciones particulares como en *ODEM*.

Por último, *Blockcerts* es un mecanismo de certificación como aquellos que se usan *off-chain* (p.e. el certificado de DNI electrónico). *Smart Exam* por su parte es una DApp cuyo propósito final no es la certificación pero que permite comprobar si una cierta persona ha participado en un examen y obtener su corrección de ser necesario.

2.2.2 Ciberseguridad

Hablando en cuanto a la ciberseguridad de las anteriores aplicaciones dentro de BC, cabe mencionar que todos los datos que se publiquen en ella son públicos. Esta condición se ha de tener en cuenta para respetar la privacidad de los usuarios. Por ejemplo los contratos que crea *OpenLaw* son visibles en la BC. Aunque los datos de los participantes en el contrato puedan estar anonimizados o encriptados, las condiciones y comportamiento del mismo son públicos[6].

Mientras *Blockcerts* es considerada tan segura como las versiones *off-chain*, usando *árboles de Merkle* [9] junto con ECDSA [10](*Elliptic Curve Digital Signature Algorithm*), es necesario que el usuario genere y almacene de forma segura el certificado. Este mecanismo es susceptible a ataques o típicos fallos *off-chain* como robos o pérdidas con el coste tangible y económico que eso conlleva (interactuar con la BC para expedir y verificar certificados conlleva un precio en GAS[7]). Por otro lado, *ODEM* usa *ERC-780* para la certificación que aún no es considerado un estándar Ethereum Improvement Proposal.

[5] La application no está implementada en la *blockchain*.

[6] Se recuerda que *OpenLaw* utiliza Ethereum para implementar los contratos que es una **blockchain pública de permisos abiertos (*permissionless*)**.

[7] El GAS se refiere a la unidad de medida utilizada para calcular y cobrar el costo de las operaciones y la ejecución de contratos inteligentes en la red, asegurando así su correcto funcionamiento y evitando abusos.

3 Metodología

La metodología propuesta para desarrollar la aplicación descentralizada *Smart Exam* consta de los siguientes pasos:

- **Definir actores e interacciones**: Indicar los tipos de usuario que existen en el escenario aclarando su *rol* y las interacciones entre ellos.
- **Diseño del contrato inteligente**: Una vez se sabe como son las interacciones entre los actores se diseñará un contrato inteligente para el lenguaje de programación Solidity. En el diseño se incluirán las funciones para las interacciones, visibilidad de las mismas, restricciones a funciones según tipo de usuario y retornos de ejecución.
- **Implementación y pruebas del contrato inteligente**: El contrato se implementará y compilará en un entorno controlado con una blockchain simulada, por ejemplo con Ganache de Truffle Suite o con Remix IDE.
- **Desplegar el contrato en una *tesnet*[8] y crear *middleware***: Despleagar el contrato en una *tesnet* como la de Sepolia para que el contraro pueda ser instanciado a partir de su ABI[9] y su dirección en la BC. Además se creará un *middleware* capaz de crear un objeto del contrato real en la BC para poder interactuar con él.
- **Integración con IPFS**: Dentro del *middleware* se creará una conexión con un nodo en IPFS para subir y descargar archivos con IPFS. Cuando se quiera almacenar un archivo en el contrato, este primero será almacenado en IPFS y el CID (hash) obtenido se almacenará en el contrato inteligente. De este modo, cuando se necesite descargar un archivo primero se leerá el CID del contrato inteligente y después se buscará en el nodo conectado para descargar.
- **Desarrollo de la Interfaz de Usuario (UI)**: Creación de una interfaz de usuario amigable que permita a la EC adquirir el contrato, asignar profesores, establecer los detalles del examen y administrar la inscripción de estudiantes. También se hará la implementación de secciones específicas para profesores y estudiantes, permitiendo la carga de archivos y la visualización de los exámenes y sus correcciones respectivas.
- **Seguridad y Pruebas**: Auditar el código, garantizando que no pueda ser usado para implementar otras funcionalidades contrarias a aquellas para las que ha sido creado. Además se realizarán una serie de pruebas exhaustivas para validar el funcionamiento correcto del contrato inteligente y la aplicación en diferentes situaciones.
- **Despliegue y mantenimiento**: El contrato se desplegará en la *blockchain* de Ethereum y se dará mantenimiento necesario al código para arreglar errores y añadir los parches de seguridad necesarios.

[8] Una tesnet es una BC de pruebas cuyo comportamiento es idéntico a una BC convencional con la diferencia que el GAS/cripto-divisa utilizada no tiene valor en el mundo real.

[9] ABI (*Application Binary Interface*), contiene la información del contrato inteligente

4 Análisis

4.1 Actores

Los actores que interactúan entre ellos dentro de la DApp *Smart Exam* son los siguientes:

- **Contrato inteligente**: Será el encargado de almacenar los datos del examen así como de gestionar las interacciones relacionadas con el examen y comprobar los permisos de cada actor antes de realizar las interacciones (p.e. un alumno no puede añadir su examen corregido al contrato).
- **IPFS**: La DApp se conectará a esta para almacenar y descargar archivos.
- **Entidad Certificadora (EC)**: Propietaria del contrato. Es la única autorizada a añadir y eliminar profesores al examen. Es la encargada de especificar los parámetros del examen, así como de iniciar la prueba añadiendo el examen al contrato inteligente.
- **Profesores**: Son los encargados de corregir los exámenes de los estudiantes una vez finalice la prueba.
- **Usuarios sin identificar**: Se les permite entrar a la DApp para inscribirse al examen, pagando la tasa correspondiente.
- **Estudiante**: Se tratará como estudiante a aquellos usuarios que estén inscritos en el examen. Los estudiantes podrán descargar el examen a la hora de inicio fijada, añadir sus respuestas al examen y descargar su corrección cuando esté corregido.

4.2 Requisitos

En esta sección se detallan los requisitos funcionales y no funcionales que la aplicación descentralizada debe cumplir:

Requisitos Funcionales

1. La EC debe ser la propietaria del contrato inteligente aunque no haya implementado el contrato en la BC.
2. La EC debe tener capacidad de añadir y eliminar profesores al contrato a partir de la dirección de estos en la BC.
3. La EC podrá enviar a su dirección en la BC la recaudación del examen.
4. La EC podrá, siempre y cuando el examen no haya empezado, editar los parámetros asociados al examen:
   - Fecha
   - Duración
   - Fecha de inscripción
5. La EC iniciará el examen cuando añada el enunciado a la DApp.

6. Los profesores podrán ver las direcciones de los alumnos inscritos.
7. Los profesores podrán añadir y/o modificar el archivo con la corrección de la entrega de un alumno particular.
8. Los profesores podrán ver los CID de las correcciones que han añadido al *Smart Exam*.
9. Cualquier usuario con una dirección en la BC utilizada, podrá inscribirse en el examen si no se ha inscrito previamente.
10. Cualquier usuario con una dirección en la BC utilizada, podrá consultar los parámetros asociados al examen.
11. Un estudiante podrá entregar el archivo con sus respuestas al examen tras la fecha de inicio y antes de que acabe la duración de este.
12. Un estudiante/profesor/EC podrá ver y descargar el enunciado del examen una vez este haya empezado.
13. Un estudiante podrá obtener la corrección de su entrega y su nota una vez un profesor las añada al *Smart Exam*.
14. Cualquier usuario con una dirección en la BC utilizada, podrá comprobar si una dirección dicha BC tiene el certificado.

Requisitos NO Funcionales

1. **Transparencia**: Se debe poder conseguir una trazabilidad de todas las operaciones relacionadas con el *Smart Exam*.
2. **Privacidad**: Los datos que puedan ser vinculados con una persona física no podrán ser accedidos por el resto de usuarios.
3. **Escalabilidad**: La aplicación descentralizada debe estar diseñada de tal modo que se puedan generar tantos *Smart Exam* como se necesiten.
4. **Experencia de Usuario (UX)**: La interfaz debe ser sencilla y eficaz para que el mayor número de usuarios se sientan cómodos y sepan como manejar las interacciones con la aplicación.
5. **Explicabilidad y comprensibilidad**: La DApp debe ser autoexplicativa y en caso de que ocurra algún error el usuario debe ser capaz de interpretarlo y corregirlo/manejarlo.
6. **Disponibilidad**: Se debe garantizar que los estudiantes puedan descargar los examenes a la hora de inicio con un número de peticiones simultáneo equivalente al número de inscritos para un examen independientemente de la posición geográfica de estos.

4.3 Justificación del uso de Ethereum

El *token* de *Ethereum* (*Ether*, ETH) es una de las tres principales cripto-divisas junto con el de *Bitcoin* y *Litecoin* [11], lo que garantiza que tenga un gran soporte por la comunidad y actualizaciones más continuas. En una tecnología como *blockchain* que está en continua evolución y desarrollo. Esto es crucial ya que depende de sus usuarios para no desaparecer y mantener la mayor seguridad posible. Un problema de seguridad en una BC significa perder dinero de sus usuarios, algo indeseable en nuestra aplicación. Que el *Ether* sea una cripo-divisa "fuerte" en el sector también es de gran importancia porque baja la volatilidad de la misma, que afecta directamente al precio del GAS que los usuarios pagan en cada interacción con los contratos.

Desde el punto de vista del desarrollo resulta interesante que Ethereum sea una BC **sin permisos** (*permissionless*), ya que permite al creador del contrato implementarlo directamente en la BC. Además este tipo de BC también permiten que cualquiera pueda tener una dirección de la misma, facilitando su uso por los usuarios de la aplicación. Otra gran característica de Ethereum para *Smart Exam* es que es **pública** implicando que quien quiera podrá ver todas las transacciones que se han realizado dentro de la BC de Ethereum dando las características de transparencia y trazabilidad que se piden en los requisitos.

Por último, la gran diferencia entre Ethereum y el resto de *blockchain públicas sin permisos* es el número de nodos que participan en su BC, lo que está directamente relacionado con la disponibilidad y el tiempo de respuesta de la BC para una petición (como descargar el examen). Aunque no se puede saber con certeza el número de nodos operativos de una BC, La web Publish0x hizo una publicación en el año 2022 en donde estimaba que Ethereum era la segunda BC con más nodos (solo por detrás de Bitcoin) y que tenía más del doble que la segunda *blockchain pública sin permisos* (Cardano).

La suma de todas estas características son las que han llevado a elegir a Ethereum como la BC para desplegar el contrato inteligente *Smart Exam*.

5 Diseño

En este capítulo se aborda el diseño integral de *Smart Exam*. Se exploran casos de uso, diagramas de secuencia, la arquitectura de comunicaciones, el modelo de datos y las tecnologías empleadas. Proporciona una visión detallada y estructurada del diseño conceptual y técnico que respalda la funcionalidad y desarrollo de la plataforma.

5.1 Casos de uso

A continuación se plantean los casos de uso relacionados con los actores pricipales: **Entidad Certificadora (EC), Profesores, Estudiantes** y **Usuario Externos** como se puede ver en la figura 1.

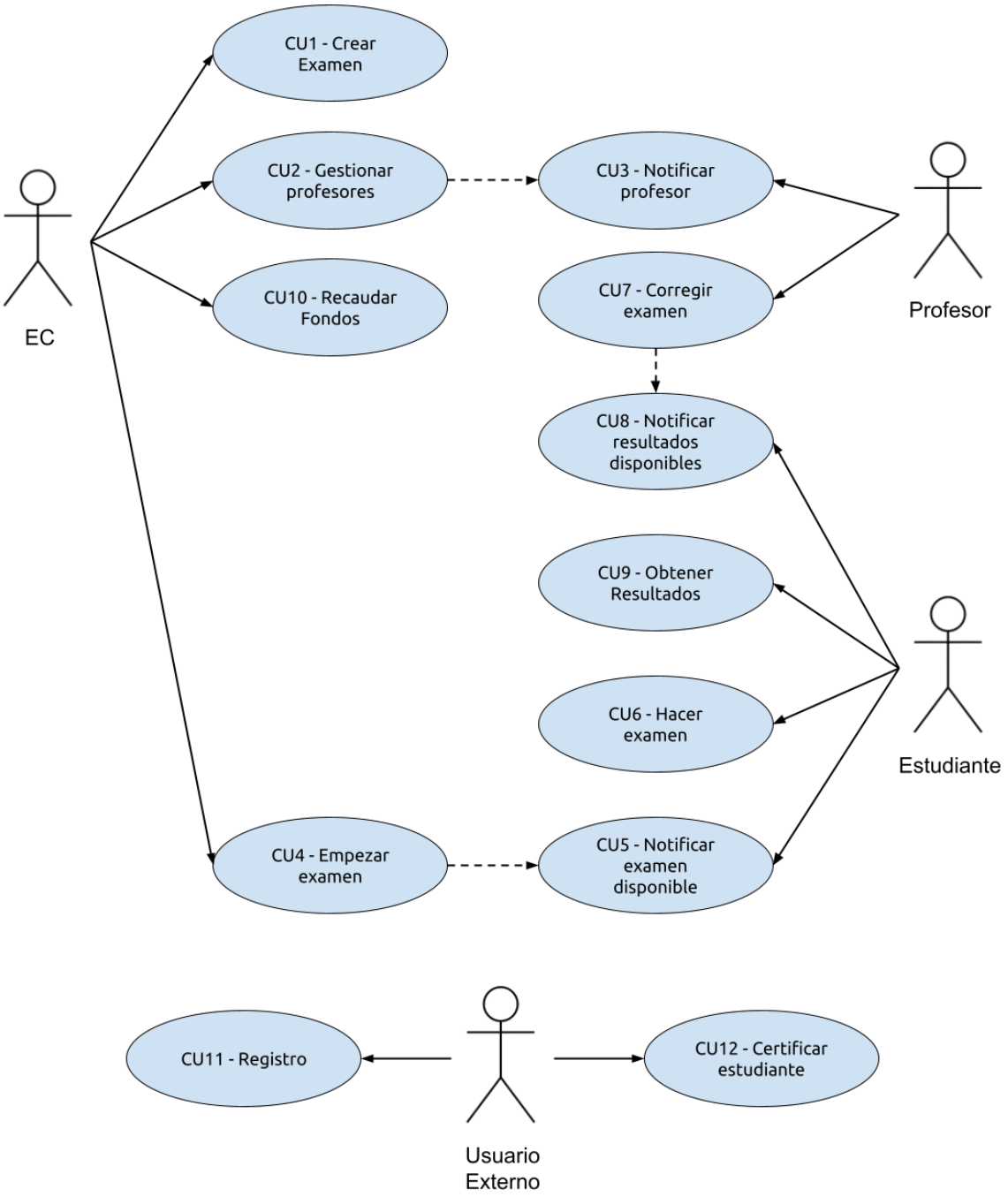


Figure 1: Diagrama casos de uso.

Caso de uso	CU1 - Crear Examen
Actores	• *Actor Principal:* Entidad Certificadora (EC)
Condiciones	La EC tiene que estar registrada en la plataforma
Flujo básico	• Fijar hora de la prueba. • Fijar duración de la prueba. • Fijar precio de inscripción.
Consecuencia	Se creará un nuevo contrato inteligente *Smart Exam* con los parámetros fijados del examen.
Información adicional	Si la EC cambia los parámetros del examen estos también se cambiarán en el *Smart Exam* correspondiente (no se creará un nuevo contrato inteligente, se actualizará el correspondiente contrato inteligente).

Caso de uso	CU2 - Gestionar profesores
Actores	• *Actor Principal:* Entidad Certificadora (EC)
Condiciones	El examen tiene que estar creado.
Flujo básico	• La EC añade un profesor al examen a partir de la dirección de un profesor particular en la BC.
Flujos alternativos	Una vez el profesor se ha añadido a un examen este se podrá eliminar si aún no ha corregido ningún examen.
Consecuencia	Los profesores añadidos podrán acceder a los servicios de corrección.

Caso de uso	CU3 - Notificar profesor
Actores	• *Actor Principal:* Entidad Certificadora (EC) • *Otros actores:* Profesor
Condiciones	Se tiene que haber inscrito o eliminado el profesor en el contrato inteligente.
Flujo básico	Al profesor en cuestión le llegará una notificación con el estado actual de su estatus.

Caso de uso	CU-4 Empezar examen
Actores	• *Actor Principal:* Entidad certificadora
Condiciones	Solo se podrá empezar un examen después de que llegue la fecha de inicio.
Flujo básico	• Selecciona el archivo con el enunciado. • Se añade el archivo a la base de datos (IPFS a través de OrbitDB). • El CID del enunciado se añade al contrato inteligente.
Consecuencia	El examen está disponible para ser descargado por la propia EC, los profesores asociados y los estudiantes inscritos.

Caso de uso	CU5 - Notificar examen disponible.
Actores	• *Actor Principal:* Entidad Certificadora (EC) • *Otros actores:* Estudiante
Condiciones	El examen tiene que haber sido empezado.
Flujo básico	• Se notifica a los estudiantes inscritos de que el examen está disponible.

Caso de uso	CU6 - Hacer examen
Actores	• *Actor Principal:* Estudiante
Condiciones	El examen tiene que haber sido empezado.
Flujo básico	• El estudiante descarga el archivo con el enunciado. (Se lee el CID del enunciado en el contrato inteligente y se obtiene el archivo de IPFS). • Cuando el estudiante tenga su documento de respuesta lo añadirá a la DApp (como para el enunciado, archivo en IPFS a través de OrbitDB y CID en el contrato inteligente asociado a la dirección del estudiante).
Consecuencia	El archivo con las respuestas del estudiante puede ser descargado por cualquier profesor asociado al exámen.

Caso de uso	CU7 - Corregir examen
Actores	• *Actor Principal:* Profesor
Condiciones	El examen tiene que haber finalizado (Hora de corrección > Fecha inicio + duración)
Flujo básico	• El profesor descarga el archivo enunciado (como en el caso del estudiante). • El profesor lista las direcciones de los estudiantes inscritos (este dato sale del contrato inteligente). • El profesor obtiene el archivo con las respuestas de un estudiante a partir de la dirección del estudiante en la BC. La dirección se usa para buscar en el contrato el CID correspondiente a sus respuestas y así poder descargarlo con OrbitDB. • Una vez el profesor tenga el documento de respuestas, este se añadirá al *Smart Exam*. Igual que la respuesta del estudiante, el archivo se guardará en IPFS y el CID obtenido en el contrato asociado a la dirección del estudiante. • Junto con el fichero de corrección el profesor también deberá asociar la nota del examen.
Flujos alternativos	• Si al buscar el examen de un estudiante no se encuentra el CID de sus respuestas, significa que el estudiante no ha entregado el examen y por tanto no se añadirá ninguna corrección

Consecuencia	El estudiante podrá descargar y visualizar tanto las correcciones como el examen. Además un tercer actor podrá comprobar que la dirección del estudiante tiene asociado el certificado en caso de que el estudiante haya aprobado.

Caso de uso	CU8 - Notificar resultados disponibles
Actores	• *Actor Principal:* Profesor • *Otros actores:* Estudiante
Condiciones	La corrección y la nota de las respuestas del estudiante tienen que haber sido añadidas al *Smart Exam*
Flujo básico	• Se notifica al estudiante que la corrección y la nota de su entrega ya están disponibles.

Caso de uso	CU9 - Obtener resultados
Actores	• *Actor Principal:* Estudiante
Condiciones	Las respuestas tienen que haber sido corregidas por un profesor.
Flujo básico	• El estudiante descarga la corrección de su entrega. El CID de su corrección está asociado a su dirección dentro del contrato inteligente. Ya con el CID se obtiene el contrato de IPFS. • El estudiante puede visualizar la nota de su entrega.

Caso de uso	CU10 - Recaudar fondos
Actores	• *Actor Principal:* Entidad Certificadora (EC)
Flujo básico	• En cualquier momento la EC podrá transferir las cripto-divisas almacenadas en la dirección del contrato a la suya propia.

Caso de uso	CU11 - Registro
Actores	• *Actor Principal:* Usuario externo
Condiciones	No estar registrado previamente como el tipo de usuario del que se quiere hacer el registro. P.e. Una EC puede registrarse para tener una cuenta de alumno pero no otra vez como EC.
Flujo EC	• La EC interesada podrá consultar el funcionamiento de la aplicación descentralizada *Smart Exam*. • La EC tendrá que pagar por los servicios de la aplicación. • La EC se registrará en la aplicación.
Flujos Profesor	• Una vez una EC añade la dirección de un profesor a un *Smart Exam* este ya tiene acceso a la interfaz de profesor. El único dato que se guarda de los profesores es su dirección.

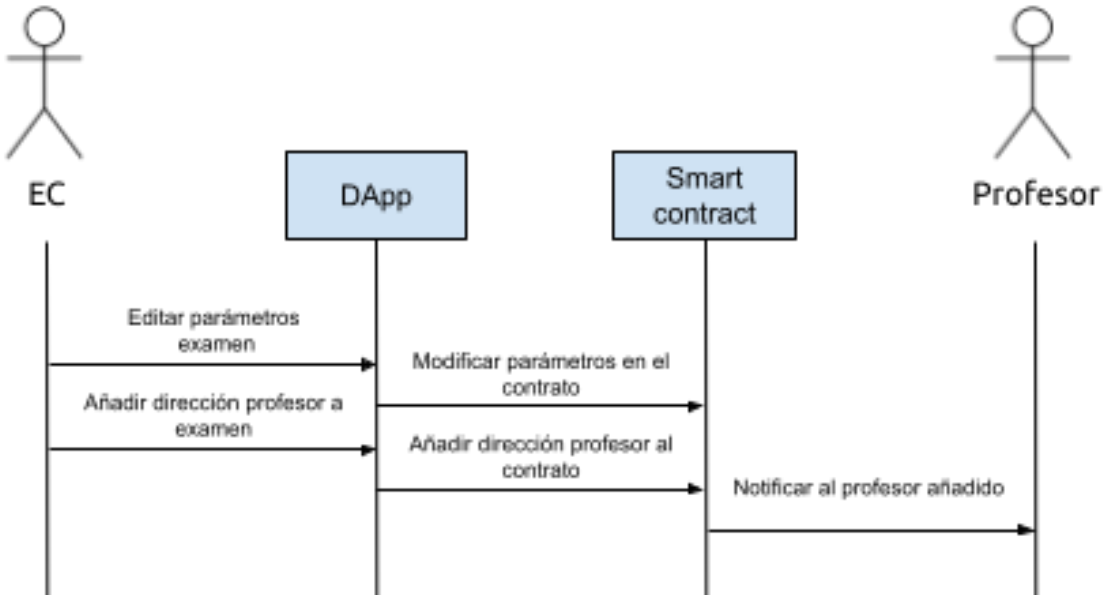
Flujo Estudiante	• Un estudiante podrá buscar las pruebas que ofrece cada EC. • Una vez se seleccione una prueba, el estudiante podrá inscribirse pagando la cantidad de inscripción o tasa correspondiente.
Flujos adicionales:	• Si un estudiante intenta inscribirse a un examen que ya ha comenzado, la petición se denegará y este recibirá un mensaje de error.
Consecuencias:	El nuevo usuario registrado podrá interactuar con la DApp visualizando una interfaz con ciertas funcionalidades dependiendo de su rol.

Caso de uso	CU12 - Certificar examen
Actores	• *Actor Principal:* Usuario externo
Flujo básico	Un usuario externo puede comprobar si una dirección en la BC utilizada tiene el certificado de un cierto examen.

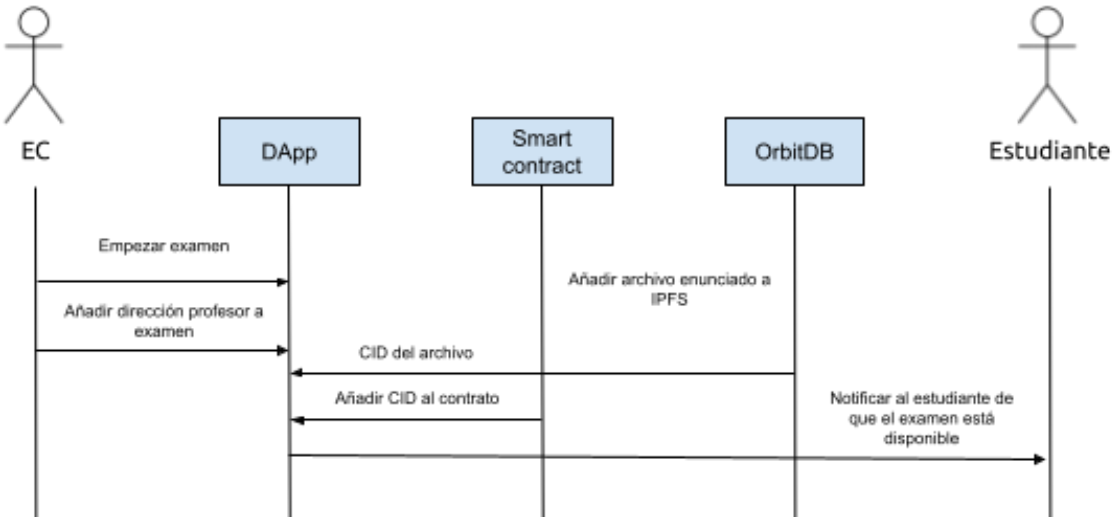
5.2 Diagramas de secuencia

A continuación se presentas los diagramas de secuencia de las principales operaciones que se llevan a cabo en la aplicación descentralizada:

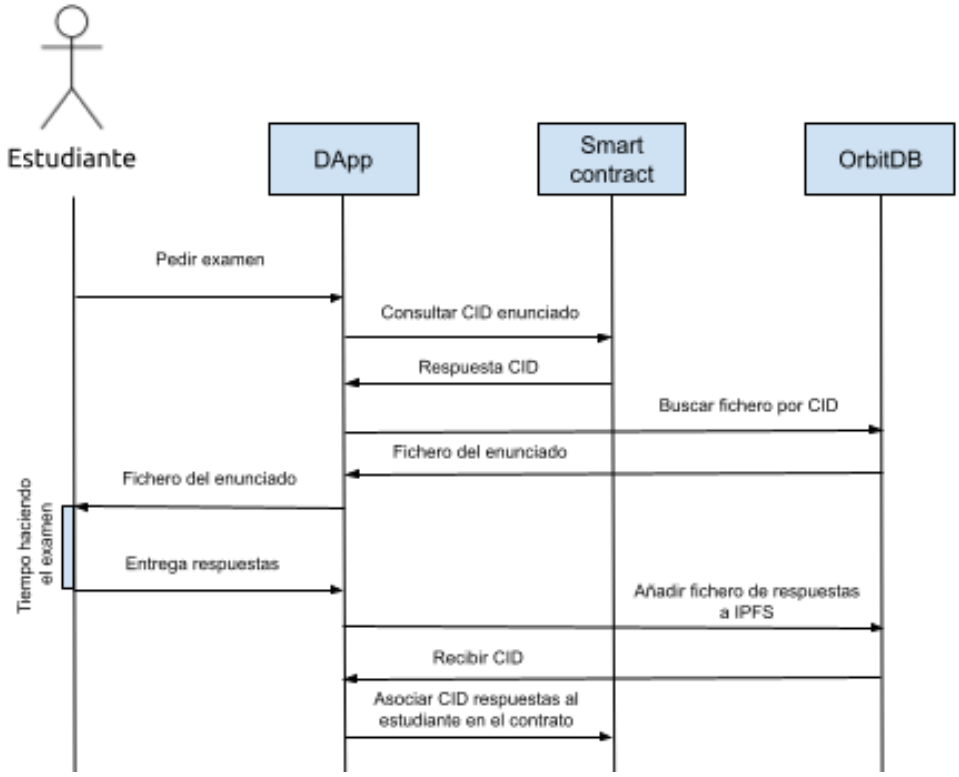
- **Creación de un examen:**



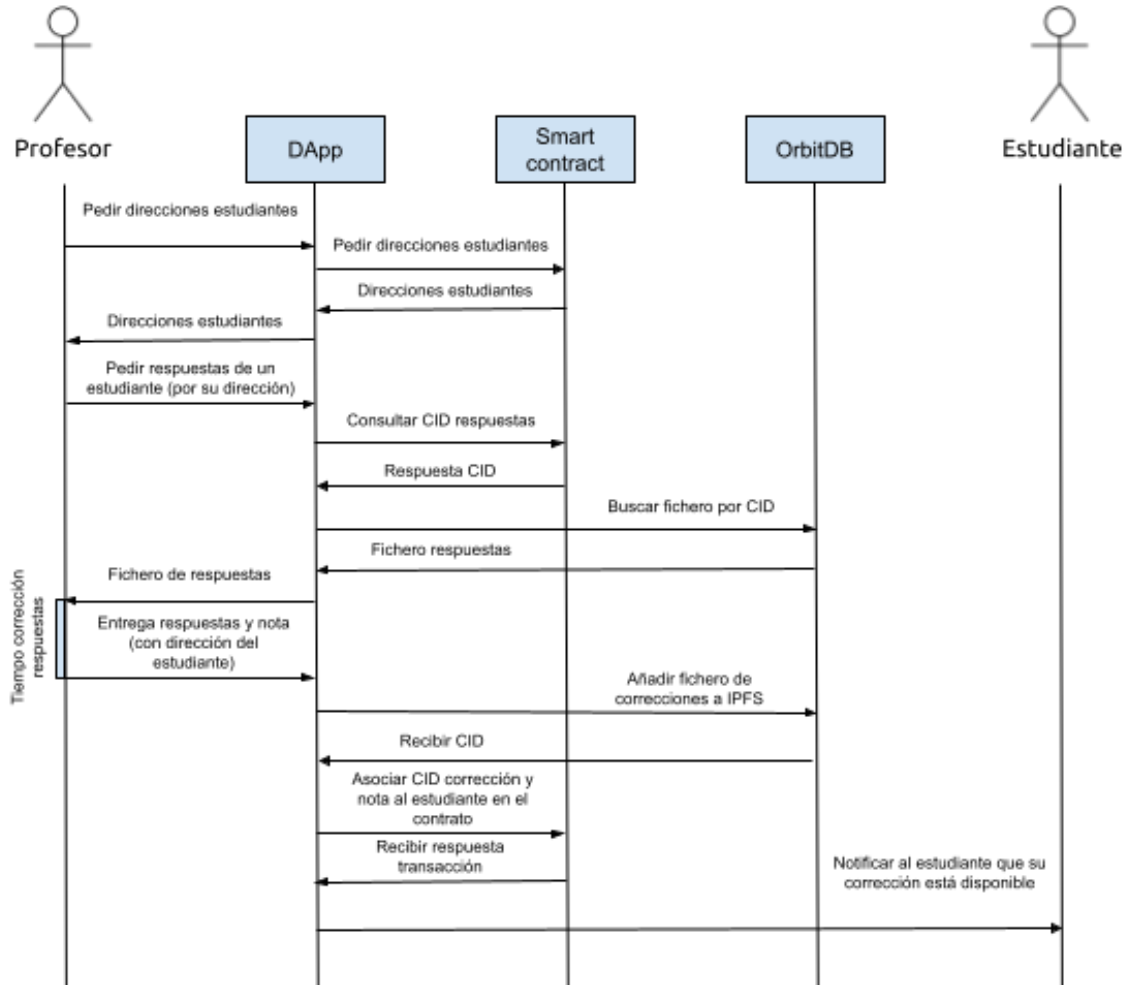
- **Comienzo del examen:**



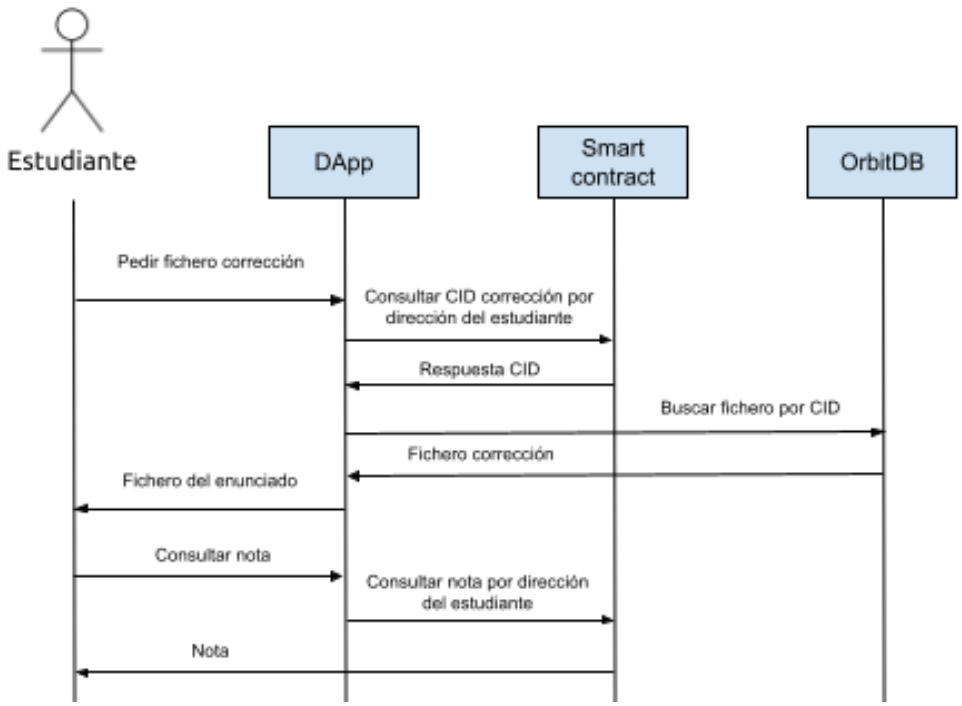
- **Entrega de las respuestas de un estudiante:**



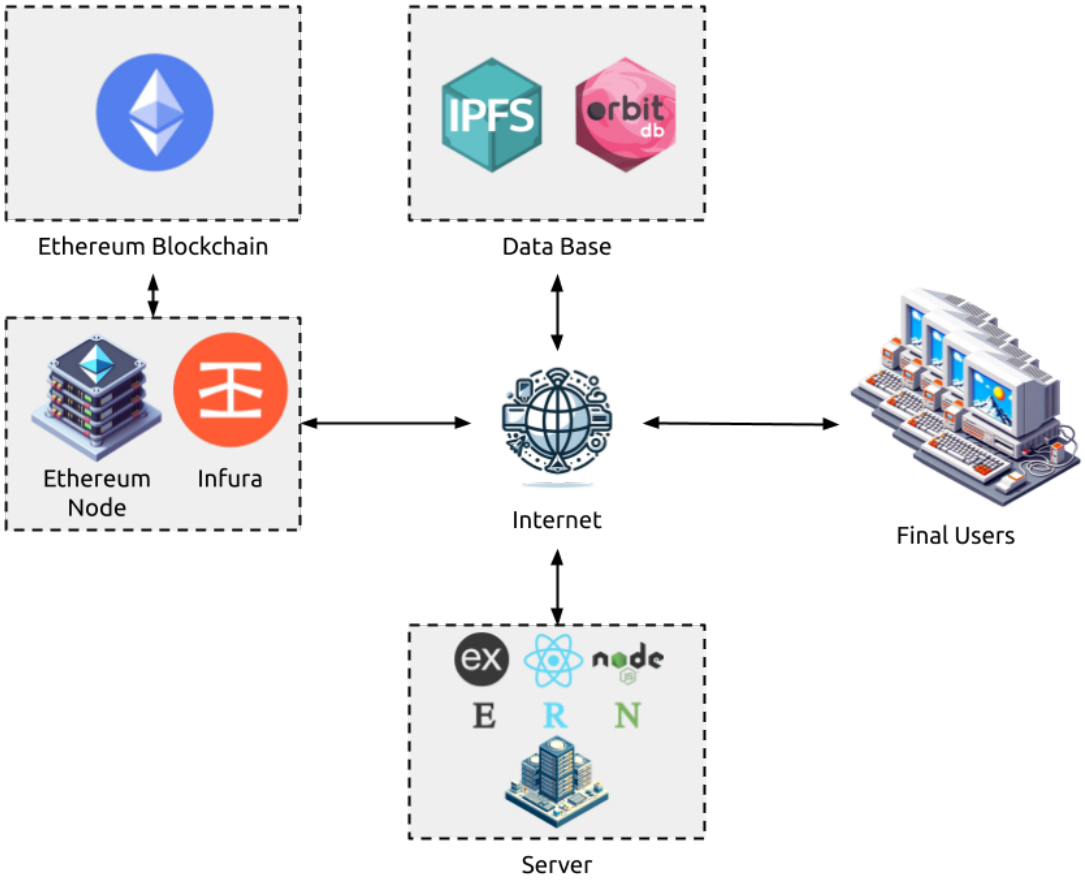
- **Corrección de un examen:**



- **Comprobar los resultados (corrección y nota) por parte de un estudiante:**



5.3 Arquitectura de comunicaciones



5.4 Modelo de datos

Dentro de los datos nos encontramos dos tipos:

- Ficheros: Enunciados, entregas y correcciones.
- No ficheros: Cadenas de texto, números y direcciones.

Los datos de tipo *fichero* serán guardados dentro de IPFS haciendo uso de OrbitDB para facilitar la organización en el almacenamiento. Los datos en OrbitDB se guardaran en estructuras de clave valor en las que cada EC tendrá el siguiente modelo de datos:

Estructura EC (tipo *keyvalue*):

- *Identificador*: Derivado por la dirección de la EC en la BC.
- *Enunciados*: Subestructura clave valor del tipo *docstore* utilizada para recuperar documentos a partir de un ID. En este caso el ID viene derivado por la dirección del contrato inteligente utilizado y el documento guardado sería el enunciado de un examen.

- *Entregas*: Igual al anterior pero el ID viene determinado por la dirección del contrato y la del estudiante que ha hecho la entrega. El documento es la entrega del estudiante.
- *Correcciones*: Mismo tipo de estructuras que las anteriores pero al calculo del ID se le añade la dirección del profesor que ha corregido el examen. El documento es la corrección de la entrega del estudiante.

Por otro lado, los datos que son ficheros se guardan en el contrato ya que son necesarios para comprobar que usuario puede acceder a cada función y dentro de las funciones en sí para cumplir con sus propósitos. Se organizan en variables globales, mapas y vectores. Estas decisiones se comentarán en la sección de implementación 6.

5.5 Tecnologías utilizadas

- **Web3**: Es fundamental para las DApps, ya que permite la interacción con contratos inteligentes y nodos de **blockchain** desde el lado del cliente. Facilita la implementación de funcionalidades descentralizadas y la gestión de transacciones.
- **Solidity**: Es esencial para crear contratos inteligentes en Ethereum, ofreciendo transparencia, seguridad y la capacidad de ejecutar lógicas programables de forma confiable en una red descentralizada.
- **Stack OERN**: *Smart Exam* esta pensado para utilizarse bajo plataforma Web y en la actualidad está bastante extendido implementar APIs [12] (Applicaction Programming Interface) para agilizar las comunicaciones. En particular, como el servidor de *Smart Exam* puede perder mucho tiempo en la busqueda de datos descentralizada es útil usar una filosofía RESTfull [13] en la que el servidor no tiene que guardar el estado de las sesiones de los usuarios. No se han encontrado otras referencias del stack propuesto pero resulta ideal para el desarrollo de la aplicación descentralizada *Smart Exam*. Básicamente es el uso del stack MERN [14] pero cambiando la base de datos MongoDB por OrbitDB:
  - **OrbitDB**: Almacena datos de manera organizada como las bases de datos convencionales pero de manera descentralizada haciendo uso de IPFS.
  - **Express.js**: simplifica el manejo de rutas, middleware y operaciones HTTP.
  - **React**: Framework para generar y personalizar interfaces dinámicas.
  - **Node.js** para ejecutar el entorno de desarrollo y agilizar las respuestas por parte del servidor.

  Esta combinación garantiza una experiencia de usuario ágil, escalabilidad, y datos distribuidos de forma segura y eficiente en una red descentralizada.

6 Implementación

6.1 Smart Contract

A continuación se especifican los pasos que se han seguido para crear la aplicación descentralizada empezando por el contrato inteligente[10]. El contrato es la piedra angular de la aplicación, se puede ver su implementación en el apéndice A. En el se definen todas las funcionalidades que pueden hacer cada tipo de usuario. En concreto se trata de las siguientes funciones:

- **Relacionadas con la EC (propietario del contrato):**
  - **registerProfessor(address _profAdd):** Añade una dirección de profesor al contrato.
  - **deleteProfessor(address _profAdd):** Elimina una dirección de profesor del contrato.
  - **editExamParameters(uint _dateExam, uint8 _duration, uint24 _enrollingPrice):** Edita los parámetros del examen, como fecha, duración y precio de inscripción.
  - **startExam(string memory _statement):** Inicia el examen estableciendo el enunciado y la fecha de inicio.
  - **withdraw():** Permite al propietario retirar los fondos almacenados en el contrato.
- **Relacionadas con el profesor:**
  - **getStudents():** Permite a un profesor obtener las direcciones de los estudiantes inscritos.
  - **getStudentSubmission(address _studAdd):** Permite a un profesor obtener el CID de la entrega de un estudiante desde su dirección.
  - **setCorrection(address _studAdd, string memory _correction, uint8 _score):** Permite a un profesor agregar una corrección para un examen de un estudiante.
  - **getCorrections():** Permite a un profesor obtener todas las correcciones.
- **Relacionadas con el estudiante:**
  - **setSubmission(string memory _submission):** Permite a un estudiante almacenar el CID de su entrega.
  - **getMyExam():** Getter para obtener el CID de la entrega del estudiante que llama a la función.
  - **getMyCorrection():** Getter para obtener el CID de la corrección del estudiante que llama a la función.

[10]Todo el código relacionado con la DApp *Smart Exam* está disponible en el repositorio: https://github.com/pedrotega/munics/tree/main/BC/Lab3

  - **getMyScore():** Getter para obtener la puntuación del estudiante que llama a la función.

- **Otros:**

  - **enroll():** Permite a una dirección inscribirse en un examen pagando la tarifa de inscripción.
  - **certificateStudent(address _studAdd):** Getter para confirmar si un estudiante aprobó o no el examen.
  - **isOwner(), isProfessor(), isStudentEnrolled():** Funciones de consulta para verificar el rol del remitente.
  - **getStatement():** Permite obtener el CID del enunciado del examen a los usuarios relacionados con el examen: *Propietario*, *Profesor* o *Extudiante*.

Dentro de las funciones se usan los siguientes modificadores para revertir la transacción que se esté llevando a cabo a su estado inicial en caso de que se produzca un comportamiento no autorizado. P.e. un alumno que quiera cambiar su nota:

- **onlyProfessor:** Comprueba si la dirección que llama a la función pertenece a un profesor. Si no es un profesor, la función no se ejecuta y muestra un mensaje de error.
- **checkStudent:** Verifica si un estudiante está registrado según la dirección proporcionada. Si la dirección no corresponde a un estudiante, se muestra un mensaje de error.
- **checkSubmission:** Exige que exista una entrega de un estudiante correspondiente a la dirección proporcionada. Si no hay una entrega registrada, la función no se ejecuta y muestra un mensaje de error.
- **checkNOTSubmission:** Exige que NO exista una entrega de un estudiante correspondiente a la dirección proporcionada. Si ya hay una entrega registrada, la función no se ejecuta y muestra un mensaje de error.

Ahora se procederán a citar las variables globales y sus características:

- **Parámetros del examen:**

  - `statement (string)`: Hash del archivo real del examen (para evitar alteraciones).
  - `dateLastUpload (uint256)`: Fecha del último momento en que se hizo una modificación relacionada con los parámetros del examen.
  - `dateExam (uint256)`: Fecha exacta en la que debería tener lugar el examen.
  - `dateStartExam (uint256)`: Fecha real en que comenzó el examen (se añadio el enunciado).
  - `duration (uint8)`: Duración del examen en minutos.
  - `enrollingPrice (uint24)`: Precio para inscribirse en el examen en wei.

  - **Nota**: Para optimizar el almacenamiento se decidió que `duration` solo ocupe `8b` ya que no se contemplan exámenes de más de 4 horas. El precio de inscripción puede parecer alto, sin embargo la siguiente opción disponible es `uint16` que no llega a un ether. Teniendo en cuenta que existen certificaciones de miles de euros y que el precio del 1 ETH equivale a unos 2k € a fecha de este trabajo, se ha decidido elegir entero sin signo de `24b`. Por último, las variables relacionadas con las fechas son de 256b porque representan el valor en *Unix TimeStamps* de la fecha (los segundos que han pasado del 1 de Enero de 1970) y por tanto son necesarios dichos bits.

- **Struct Student:** Recoge los datos relacionados con un estudiante.
  - `submission (string)`: Almacena el CID de la entrega del estudiante.
  - `correction (string)`: Almacena el CID de la corrección del estudiante.
  - `score (uint8)`: Puntuación del estudiante, entre 0 y 10.

- `_professors, mapa(dirección, bool)`: Almacena las direcciones de los profesores añadidos por el propietario de tal manera que la clave es la dirección del profesor y el valor `verdadero`, porque está añadido. Si se desea prescindir de un profesor, bastaría con fijar el valor de su dirección en el mapa a `falso`

- `_students, mapa(dirección, estructura Student)`: Asocia las direcciones de los estudiantes con la estructura que contiene su información.

- `_studAdds, vector de direcciones`: Vector que almacena las direcciones de los estudiantes.

- `_correctionCIDs, vector de strings`: Almacena los CIDs de las correcciones.

- **Nota:** Se utilizan los mapas en las variables de comprobación o busqueda ya que requieren menos operaciones para una búsqueda aleatoria, por lo tanto, se ahorra gas. Mientras que los vectores son más eficientes guardando datos en memoria, útil para cuando no se requiere buscar elementos concretos.

Respecto al código del *Smart Contract* también es importante aclarar que se divide en dos contratos por razones de comprensibilidad: `SmartExam.sol` y `SmartExamBase.sol` (el primero hereda del segundo). En el base se definen todas las funcionalidades relacionadas con el propietario del contrato, los modificadores y las variables globales. En cambio, en el otro están definidas las funciones generales y las relacionadas con los usuario *profesor* y *estudiante*. Además, el contrato base hereda a su vez de un contrato de OpenZeppelin llamado *Ownable.sol* que implementa funciones relacionadas con el propietarlo, como una función para obtener su dirección (`owner()`) u otra para ceder la propiedad a otro usuario (`transferProperty()`). Se puede consultar el diagrama UML del código en el anexo B

6.2 JavaScript

En la parte de implementación relacionada con *JavaScript* se utilizan varios elementos para construir una aplicación web interactiva que interactúa con la red Ethereum y contratos inteligentes. Aquí está la descripción de los elementos y cómo interactúan entre sí:

1. **React**: Es el framework de *JavaScript* utilizado para construir la interfaz de usuario de la aplicación. Divide la interfaz en componentes reutilizables y maneja la lógica de representación visual.

2. **ethers.js**: Es una biblioteca de *JavaScript* utilizada para interactuar con la red Ethereum y los contratos inteligentes. Permite la conexión a la red, la firma de transacciones y la llamada a métodos en los contratos.

3. **crypto-js**[11]: Esta biblioteca es la que aporta la protección de los datos sensibles en la aplicación (respuestas de exámenes y sus respectivas correcciones). Se ha utilizado de la siguiente manera:

    - *Respuestas de exámenes*: Se utilizará el algoritmo de intercambio de claves de Diffie-Hellman [15] (DH) para crear claves simétricas AES-256 [16, 17] con las que se cifraran las respuestas que se suben a la BC. El funcionamiento es el siguiente:
        - La EC tiene un par de claves DH para un examen específico.
        - Cada estudiante tiene su correspondiente par de claves DH.
        - El estudiante usa la clave pública del profesor con su clave privada para generar la clave AES-256 y cifrar su examen.
        - La EC usa su clave privada con la de un alumno en especifico para crear la misma clave AES-256 que se ha usado para encriptar para poder desencriptar el examen del estudiante.
        - Una vez en IPFS todo el mundo puede acceder al archivo encriptado del examen pero solo el estudiante que ha hecho el examen y la EC pueden desencriptarlo.
    - *Correcciones de respuesta*: Se utiliza el algoritmo de clave asimétrica RSA. Se encriptaría la corrección de un alumno con su clave pública haciendo que solo esta pueda desencriptarlo con su clave secreta

4. **kubo-rpc-client**: Se utiliza para comunicarse con IPFS y almacenar/descargar archivos descentralizados, como los enunciados de exámenes, las correcciones de los estudiantes, etc.

5. **HTML y CSS**: Crean la estructura y el estilo de la interfaz de usuario.

La interacción entre estos elementos se realiza a través de funciones, formularios y eventos en React:

[11] Las funciones de encriptación no se han llegado a implementar en el prototipo

- **Formularios y Eventos**: Se utilizan para capturar la entrada del usuario, como la carga de archivos (enunciados, correcciones, etc.), acciones como el registro de profesores, la inscripción en exámenes, la descarga de archivos, etc.
- **Funciones de Manejo de Eventos**: Son funciones que se activan en respuesta a eventos de usuario, como enviar formularios, cargar archivos, hacer clic en botones, etc. Estas funciones manejan la lógica de la aplicación, como interactuar con la red Ethereum, subir archivos a IPFS, llamar a métodos en contratos inteligentes, etc.
- **Interacción con Contratos Inteligentes**: Se realiza a través de la biblioteca *ethers.js*. Se conecta a la red Ethereum, obtiene una instancia del contrato inteligente y llama a sus funciones para realizar acciones específicas, como registrar o eliminar profesores, editar parámetros del examen, obtener información sobre estudiantes, obtener correcciones, etc.
- **Actualización de la Interfaz de Usuario**: Después de realizar alguna de las acciones anteriores se actualiza la interfaz de usuario para mostrar los resultados, los mensajes de éxito o error, la información obtenida, etc.

6.3 Comunicación IPFS y *blockchain* de Ethereum

Nota. Este apartado no se ha llegado a implementar en la práctica por falta de recursos y tiempo.

Como se puede ver en el apartado de Arquitectura de comunicación 5.3 para almacenar los datos en IPFS se utiliza OrbitDB para poder almacenar los datos de forma más ordenada. Además, OrbitDB se encarga de reproducir la información entre nodos de IPFS aumentando la disponibilidad de los recursos y reduciendo los tiempos de espera, haciendola una mejor solución que tener un nodo propio. La comunicación con servidor se hace de manera sencilla a través de su API de *JavaScript* lo que encaja con el desarrollo seguido facilitando el mantenimiento de la DApp.

Por otro lado, otra parte crucial con la aplicación es la comunicación con la BC de Ethereum. En un principio se puede pensar en una solución basada en la herramienta Infura que facilita el desarrollo de la aplicación y se encarga de la redundancia de la información entre varios nodes de la red de Ethereum. Esta podría parecer una solución ideal como OrbitDB pero, aunque también usa un API de *JavaScript*, cobra por el número de peticiones mensuales que se hacen a la API. Por este motivo es una buena decisión cuando el volumen de usuario sea grande, se cambiará de mecanismo de comunicación con la BC por mantener un nodo, o varios, de Ethereum. Esta solución reducirá al mínimo los tiempos de espera si se mide adecuadamente la carga de peticiones por minuto que se pueden tener. Como complemento a esta solución también se desarrollará un método de compartición de los ficheros propios con otros nodos para que los ficheros estén disponibles para usuarios que no quieran usar directamente la aplicación *Smart Exam*. Se recuerda que el objetivo final es garantizar la transpariecia y la trazabilidad de los exámenes.

Importante: Existe una **demostración** de la aplicación en el repositorio de *github* del proyecto en dónde se explica paso a paso que sucede en una serie de vídeos de muestra.

Repositorio: https://github.com/pedrotega/munics/tree/main/BC/Lab3

7 Análisis de los riesgos de seguridad

En esta sección se van a revisar las posibles amenazas que pueden comprometer la seguridad dentro de la aplicación descentralizada:

Riesgos de Seguridad

- **Vulnerabilidades en Contratos Inteligentes**: Fallas en la lógica del contrato inteligente podrían permitir a un usuario malicioso obtener acceso a la recaudación. Fallos en la definición de las funciones pueden causar grandes consumos de GAS, especialmente cuando se implementan bucles.
- **Accesos a funciones de usuarios no autorizados**: Usuario sin permisos específicos pueden acceder a funciones a las que no tienen acceso (estudiantes pueden intentar subir sus propias correcciones, aunque en este caso el programa dará error.).
- **Problemas de autenticación**: El único dato que identifica a los usuarios es la cuenta en BC, lo que no es suficiente para garantizar la identidad de un persona física si a esta le roban la cuenta.

Riesgos Tecnológicos:

- **Problemas de IPFS:** Posible falta de disponibilidad o integridad de archivos almacenados en IPFS debido a fallos en la red o pérdida de datos.
- **Errores en el Desarrollo:** Bugs o problemas de implementación podrían conducir a comportamientos inesperados o fallos en la plataforma. En este caso del *frontend* y del *middleware* de la DApp.

8 Plan de pruebas

Para probar el correcto funcionamiento de la aplicación descentralizada se han realizado las siguientes pruebas:

Pruebas especificas del contrato inteligente: Estas pruebas son de gran importancia porque una vez desplegado el contrato no puede ser modificado y cada vez que se despliega lleva asociado una tasa que se debe pagar. Además, fallos de control de acceso o bucles pueden resultar en pérdidas de dinero tanto de las EC como de los demás usuarios que interactúan con el contrato.

- Revisar el código y auditarlo, es decir, una tercera parte no relacionada con el desarrollo intenterá encontrar irregularidades de este.

- Hacer test manuales en entornos de prueba. En este caso se ha utilizado *Remix IDE* para este proceso.

- Realizar test automáticos unitarios. Consiste en una batería de test que comprueban una funcionalidad específica cada vez. P.e. cambiar la duración del examen y leer el valor del contrato para ver que se ha cambiado. El *framework* utilizado para esta prueba ha sido ha sido foundry (los ficheros utilizados se encuentran en la capeta testing del repositorio). Los resultados obtenidos al ejecutar el comando `test` son:

```
(base) pedro@pedro-ubuntu:~/BC/testing$ forge test
[ ⠊] Compiling...
No files changed, compilation skipped

Running 6 tests for test/SmartExam_professor_student.t.sol:SmartExamTestProfessor
[PASS] test_get_my_exam() (gas: 15411)
[PASS] test_get_student_submission() (gas: 22626)
[PASS] test_get_students() (gas: 13258)
[PASS] test_is_professor() (gas: 14518)
[PASS] test_is_student() (gas: 19871)
[PASS] test_set_and_get_correction() (gas: 97942)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 3.33ms

Running 7 tests for test/SmartExam_owner.t.sol:SmartExamTestOwner
[PASS] test_add_delete_professor() (gas: 29352)
[PASS] test_edit_exam_parameters() (gas: 39946)
[PASS] test_initial_owner() (gas: 9837)
[PASS] test_is_owner() (gas: 14338)
[PASS] test_start_exam() (gas: 47769)
[PASS] test_transfer_ownership() (gas: 22030)
[PASS] test_withdraw() (gas: 139863)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 3.33ms

Ran 2 test suites: 13 tests passed, 0 failed, 0 skipped (13 total tests)
```

- Una vez se han realizado los anteriores test y no se han encontrado errores, se realiza el despliegue en una *tesnet*. La *tesnet* utilizada ha sido *Sepolia*. Una vez desplegado, se realizan los dos test anteriores y se itera el despligue del contrato en la BC de pruebas hasta no encontrar errores.

Nota. Las siguientes pruebas no se han realizado por la falta de recursos y tiempo.

Pruebas de IPFS:

- Se comprueba periódicamente la conexión con al menos un nodo de IPFS para garantizar que se puede tener servicio para un número razonable de usuarios.

- Se harán pruebas de carga y descarga de archivos en el nodo local.

- Para comprobar el mecanismo de redundancia en IPFS, se buscarán los archivos cargados en el nodo local en el resto de IPFS para obtener un número mínimo de coincidencias deseado.

- En las pruebas anteriores también se medirá el tiempo en realizar las operaciones con IPFS para garantizar unos mínimos de calidad de usuario.

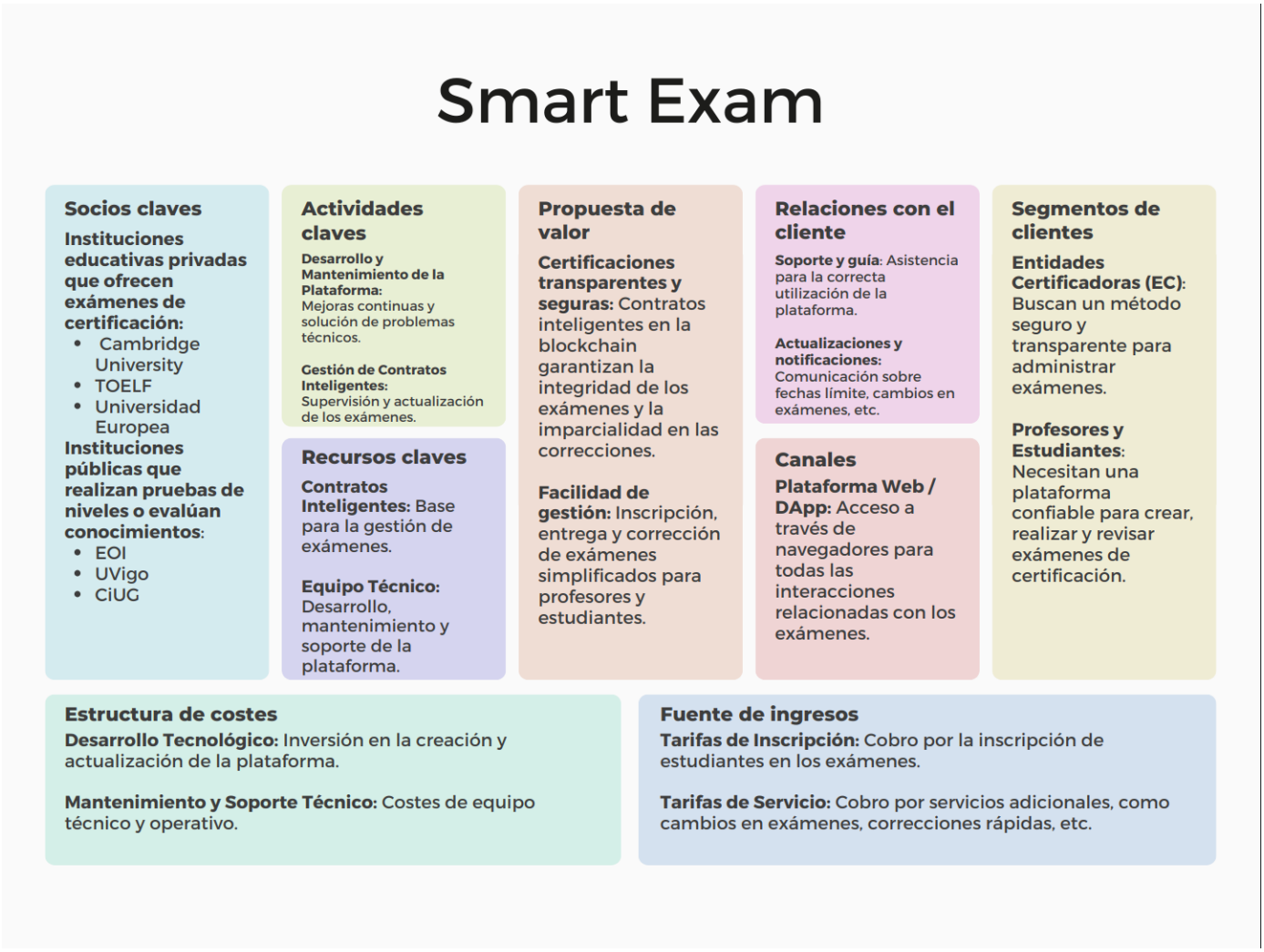
Pruebas de las DApp:

- Probar la DApp con usuarios reales para comprobar que es intuitiva y con una curva de aprendizaje suave para el público objetivo de la misma.
- Se harán pruebas de carga comprobando:
  - Número máximo de peticiones al servidor.
  - Número máximo de peticiones que involucren llamadas al contratro inteligente.
  - Número máximo de peticiones que involucren cualquier tipo de interacción con IPFS.

  Esta prueba permite calcular cuantos usuarios podemos manejar en la aplicación sin perder UX. Este conocimiento es, además, crucial para saber cuantos estudiantes pueden realizar un mismo examen sin que pierdan minutos de la prueba entre tiempos de espera.
- Toda la aplicación también será auditada por un profesional externo.

Una vez la DApp esté operativa se realizarán estas pruebas periódicamente no solo cuando se haga cualquier tipo de cambio (que en esos casos será obligatorio). La *blockcain* e IPFS no son infraestructuras estáticas, están vivas y dependen de sus usuarios por lo que es importante garantizar que con el paso del tiempo la aplicación siga dando un servicio de calidad.

9 Lean canvas



10 Manual de usuario

En esta sección se describirán los pasos a seguir para desplegar la aplicación así como las pantallas que verán los distintos usuarios:

- Clonar el repositorio de *github*.
- Desplegar el contrato en una blockchain compatible con Ethereum (también vale una *tesnet* compatible como Sepolia). Se puede usar para ello un entorno como el de *Remix IDE*.
- Es necesario tener una cuenta en una *wallet* con la divisa suficiente que se vaya a usar para desplegar el contrato (se puede tener por ejemplo el *wallet* de Metamask con *Sepolia*).
- Una vez desplegado copiar el código ABI resultante en la carpeta $smartexam\_dapp/src/contracts/src/abis/SmartExam.json$ y la dirección en donde está el contrato en la BC en $smartexam\_dapp/src/contracts/src/addresses.js$.
- Iniciar el proyecto de node: `node init`.
- Instalar las bibliotecas necesarias con el comando `npm install`
- Ejecutar el proyecto: `npm start`
- Acto seguido se abrirá una página con la interfaz de la aplicación.
- **Nota**: En esta implementación se ha utilizado un nodo local de IPFS pero se puede añadir la dirección de uno o varios nodos públicos disponibles en IPFS.

A continuación se muestran las pantallas que vería cada usuario al entrar en la web de la DApp:

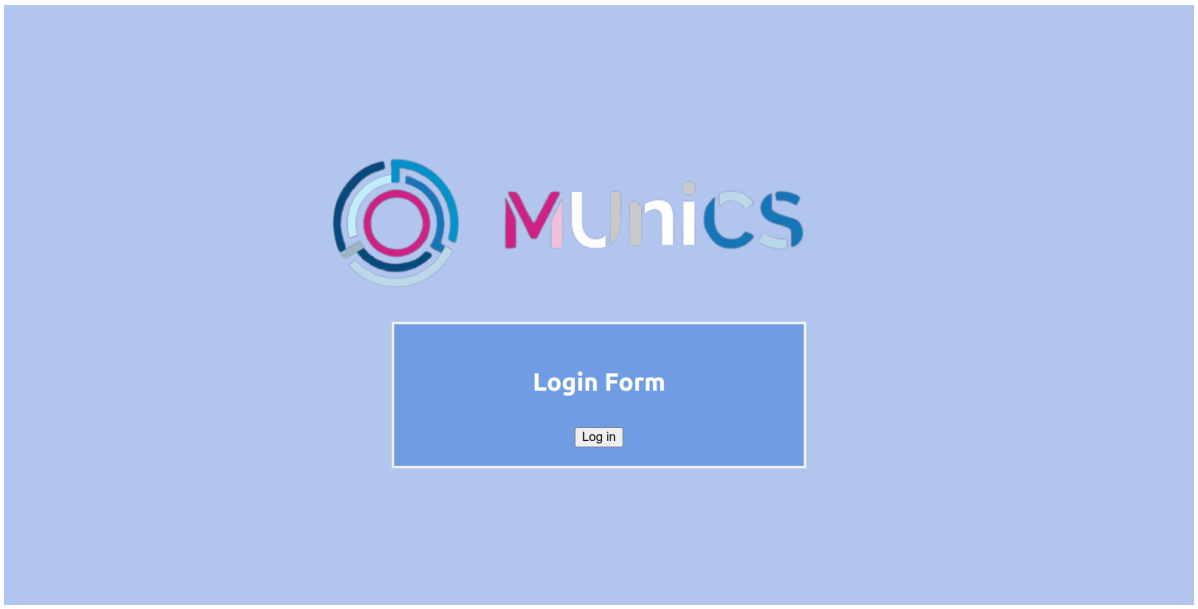


Figure 2: Página de log in.

Figure 3: Página del propietario 1 - Zona de información

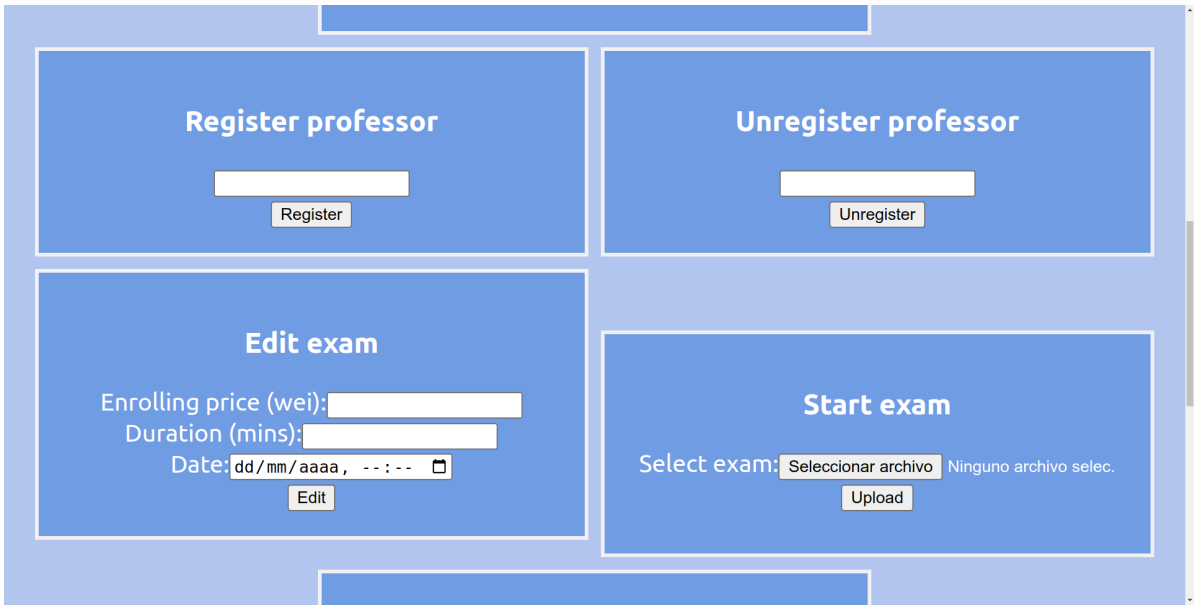


Figure 4: Página del propietario 2 - Zona de funcionalidades 1

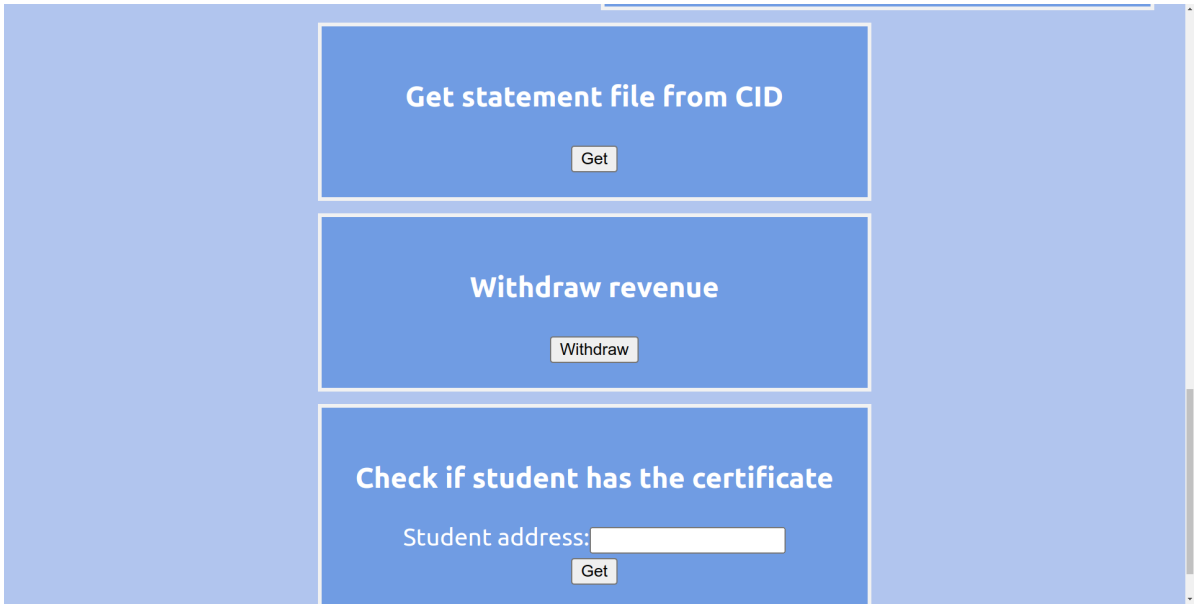


Figure 5: Página del propietario 3 - Zona de funcionalidades 2

Figure 6: Página del profesor 1 - Zona de información

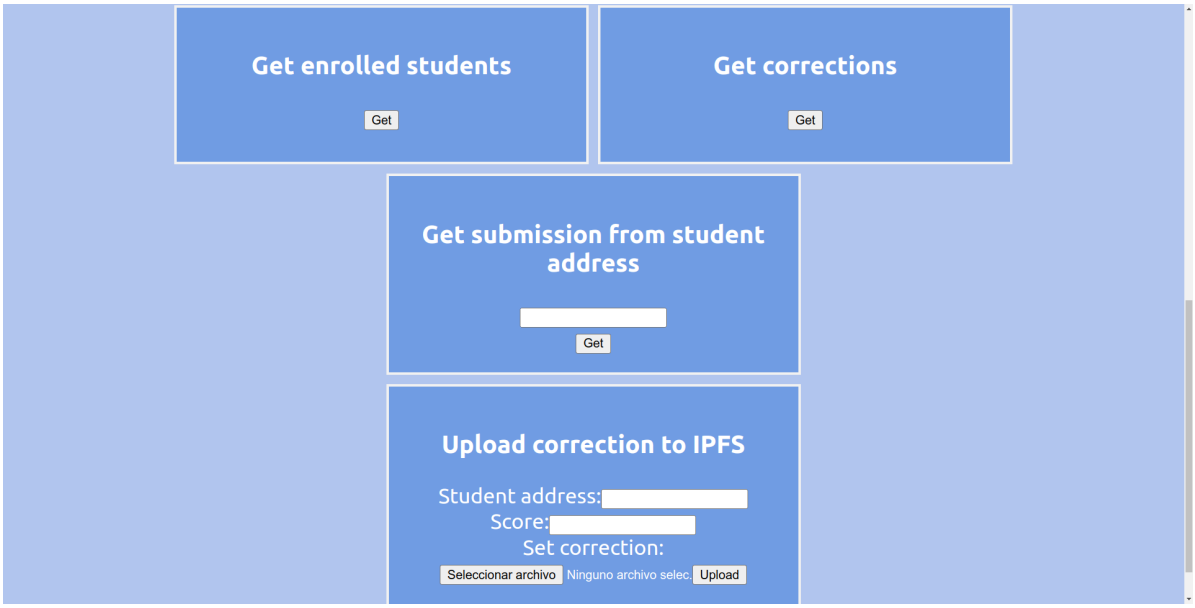


Figure 7: Página del profesor 2 - Zona de funcionalidades

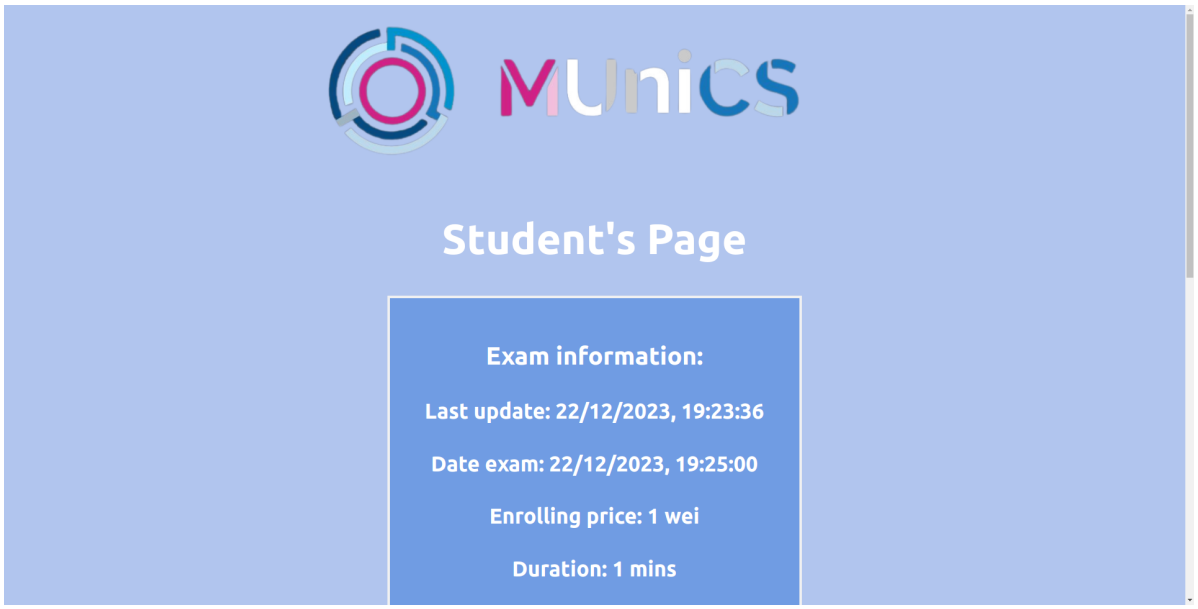


Figure 8: Página del estudiante 1 - Zona de información

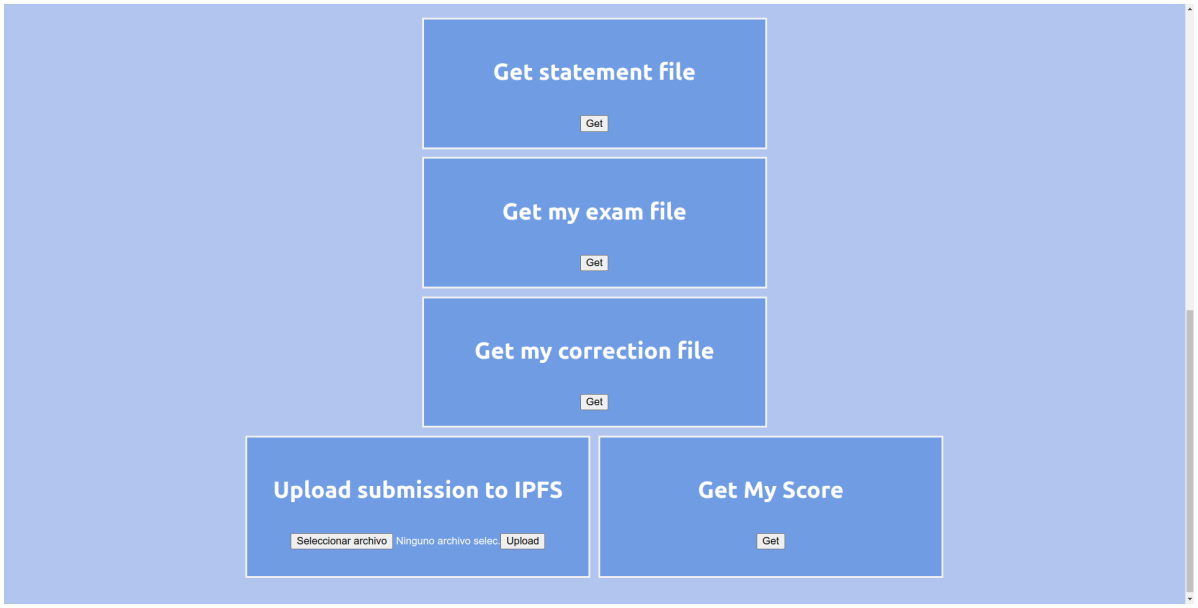


Figure 9: Página del estudiante 2 - Zona de funcionalidades

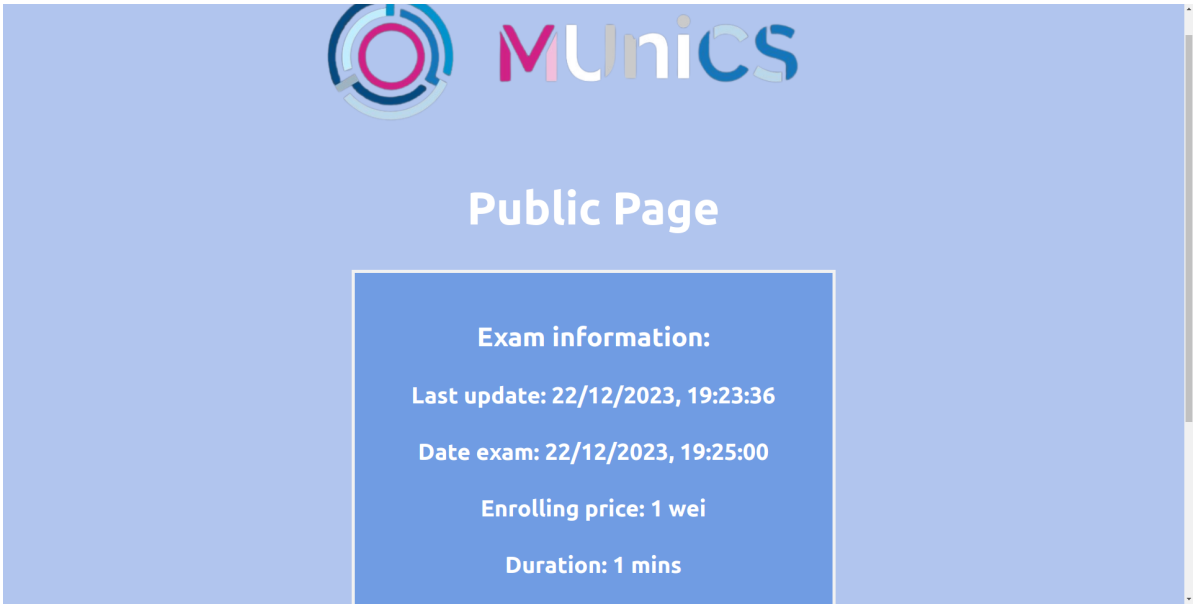


Figure 10: Página pública para usuarios no registrados - Zona información

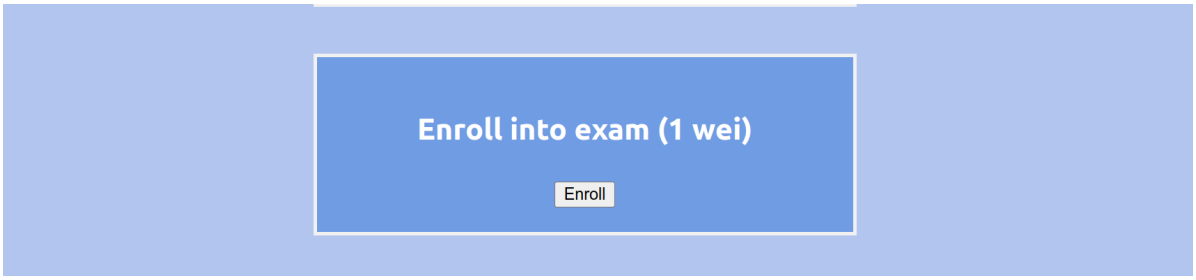


Figure 11: Página pública para usuarios no registrados - Zona funcionalidades

11 Conclusiones

Se ha diseñado e implementado una aplicación para la gestión de exámenes basada en tecnologías descentralizadas, cumpliendo las premisas por las que se había empezado el proyecto: Crear un sistema justo, transparente y trazable. A continuación, se revisan los objetivos que se pretendían alcanzar y como se han resuelto:

- La **trazabilidad** de los documentos que están guardados en el examen está garantizada por las propiedades del contrato inteligente. Cualquier persona, incluidos los estudiantes, pueden ver quien ha corregido sus respuestas y si ha habido modificaciones durante la realización del examen que le hayan podido afectar. Todo los estados del contrato inteligente están guardados en los bloques de la *blockchain*. Por lo que se pueden seguir de manera transparente las modificaciones del contrato inteligente en su historia y quien las ha realizado.

- En el estado actual solo se aceptan cripto-divisas de la red de Ethereum. El uso de estas divisas hace más igualitario el pago de tasas para certificaciones globales, como por ejemplo las certificaciones de la Universidad de Cambridge. Una certificación tiene, por tanto, el mismo precio en todo el mundo.

- Relacionado con el primer punto se encuentra la **inmutabilidad** del contrato. Definiendo una serie de restricciones, tanto de tiempo como de permisos, se ha conseguido que la información relacionada con el examen no se pueda cambiar una vez es oficial. Por ejemplo, una vez comienza el examen, el enunciado no se podrá modificar. Siguiendo con este ejemplo, aunque se encuentre un problema en el código que permita hacer estas modificaciones, como cambiar el examen cuando este haya acabado, el estado del enunciado durante el examen estará guardado en los bloques anteriores. Esta propiedad, de nuevo, hace que se sepa la verdad sobre el estado del examen y que dirección en la BC ha realizado dichos cambios.

- Los archivos relacionados con el examen son **encriptados** y **almacenados de manera descentralizada** en IPFS. Haciendo que los datos sean accedidos por cualquier persona, pero solo aquellos que estén relacionados con el archivo puedan descifrarlo.

- Finalmente, solo se ha utilizado la dirección de los implicados en la BC para su identificación. Esta máxima se ha utilizado para garantizar la máxima anonimidad, partiendo de la base que en la BC de Ethereum no puedes saber el origen de una de las direcciones (solamente sus transacciones).

12 Líneas futuras

El objetivo principal a futuro es poder crecer y aumentar el número de las funcionalidades de la aplicación descentralizada *Smart Exam*. Se tiene especial interés en poder realizar los exámenes dentro de la aplicación y que todos los pasos consten en la BC. Exámenes como la parte teórica de los permisos de la DGT[12] o el tan socorrido ejemplo de las cerificaciones de la Universidad de Cambridge ya se hacen en ordenadores. Por tanto, su adaptación a la aplicación desarrollada resultaría viable.

Para mejorar la implementación del producto y los avances en el mismo, se ha decido que en el futuro se utilizará **hardhat** [13] como herramienta de desarrollo. Hardhat dispone de una suite de pruebas integrada, que incluye soporte para TypeScript, garantiza una codificación más segura y reduce errores durante el desarrollo. Además para realizar pruebas locales dispone en un entorno de red simulado que facilita y agiliza el proceso de desarrollo antes del despliegue en la red principal de Ethereum. Esto también evita el problema de conseguir cripo-divisa de *tesnets* como la de *Sepolia*.

La verificación de la autoridad de los exámenes actualmente depende de la dirección en la BC de los estudiantes. Sin embargo, cada vez es más común el uso de certificados de estilo convencional en los campos en donde se usan aplicaciones descentralizadas. Por este motivo, como se pudo ver en la sección 2 resulta coherente usar certificados basados en BC como **Blockcerts**. La integración de *Smart Exams* con *Blockcerts* es viable y permitiría a los usuarios identificarse directamente con otros servicios sin la necesidad de acudir a la DApp de *Smart Exam*.

Finalmente, otra vía que se pretende explotar es la de migrar a otras *blockchain públicas sin permisos* como *Cardano* [18] o *NEO* [19]. El uso de las diferentes BCs sería transparente para los usuarios. Además, esta solución aumentaría la disponibilidad de los contratos inteligentes y prevendría errores críticos como el fallo de servicio o desaparición de una de las *blockchain*. Con esta solución también convendría aceptar distintos tipos de cripto-divisa dentro de la aplicación descentralizada, al menos de los *tokens* de las BCs que se utilicen.

[12] https://www.dgt.es/inicio/
[13] https://hardhat.org/

13 Lecciones aprendidas

El desarrollo de *Smart Exam* y el estudio de su entorno ha representado un proyecto innovador y desafiante. A lo largo de este proceso, se han encontrado diferentes desafíos e incidencias, que serán comentadas en la siguiente sección, pero también se han adquirido valiosas lecciones y perspectivas sobre las tecnologías utilizadas, destacando las ventajas que estas presentan.

El motivo de la descentralización: Las aplicaciones descentralizadas son más lentas y no ofrecen servicios distintos a los convencionales, entonces ¿Qué valor tienen las aplicaciones descentralizadas?
Su fuerza reside en la transparencia y trazabilidad de las operaciones. Todo el mundo puede ver que sucede dentro de una BC pública, se puede detectar si alguien ha manipulado un contrato inteligente y cuando. Hoy en día las soluciones en la BC que tienen éxito son aquellas en la que se quiere garantizar la ética y transparencia: Sistemas antifraude fiscal, trazabilidad del producto desde su origen hasta el punto de venta, sistemas de votación, etc.

Importancia de la Seguridad en *Blockchain*: La seguridad desempeña un papel crucial en cualquier aplicación, y en el contexto de *blockchain*, su atención adquiere una importancia aún mayor. Hemos aprendido que la implementación de medidas de seguridad robustas, como el la gestión segura de claves, es esencial para salvaguardar la integridad de los datos de los usuarios y garantizar la confidencialidad durante las transacciones en la *blockchain*. Esto repercute directamente en la gestión de la privacidad, ya que también se ha aprendido a equilibrar la transparencia de la *blockchain* con la necesidad de preservar la privacidad de los usuarios. El uso de técnicas de anonimización han sido cruciales para abordar estos desafíos.

Usabilidad y Experiencia del Usuario: La usabilidad es un elemento clave para el éxito de cualquier aplicación. Se han adquirido conocimientos para conseguir que la interfaz de usuario sea intuitiva y de fácil uso. Esto es importante, sobretodo, cuando nos encontramos ante una tecnología tan nueva para el público que no está familiarizado con las DApps. La retroalimentación constante de los usuarios y las pruebas de usabilidad han sido fundamentales para refinar la interfaz y mejorar la experiencia global del usuario.

Optimización de código: Se ha llegado a una solución de compromiso en la implementación del contrato. La eficiencia de los *smart contracts* en la *blockchain* es esencial para asegurar la escalabilidad de la aplicación. Se han adquirido conocimientos sobre la optimización del código de los *smart contracts* para reducir los costes de gas y mejorar la velocidad de las transacciones, asegurándonos de que la experiencia del usuario no se vea comprometida. Se han llegado a soluciones buscando equilibrio entre el coste de gas y la eficiencia operativa.

13.1 Incidencias

- **Legislación y cumplimiento normativo:** Desde la ventaja que todo en la BC es público también nace el problema de la anonimidad. Es ampliamente conocido que un usuario tiene derecho a ser olvidado, pero hay un matiz. Este matiz es que es suficiente que nadie pueda acceder a los datos de un usuario para ser olvidado. Por ello cuando se suben a IPFS, los datos se encuentran encriptados, de tal manera que si un usuario quiere ser olvidado de un *Smart Exam*, simplemente se desechará la clave AES-256 generada para encriptar sus respuestas ya que no está almacenada en el contrato.

- Conseguir *Sepolia* para hacer las pruebas en la *tesnet* también ha sido un problema. Se podía conseguir 0.5 *Sepolia* al día y haciendo varias implementaciones al día se podía acabar con facilidad. Esto implicaba estar atento y no olvidarse de recaudar algo todos los días.

- La variabilidad del GAS en la *tesnet*. Como el precio de implementación e interacción de los contratos varía con lo saturada que estaba la *tesnet* en los nodos cercanos se tenían que evitar horas de máxima concurrencia. Entre las 16:00-17:00h se han llegado a ver precios de despliegue del contrato inteligente de 0.44 Sepolia. Algo impensable pensando en hacer varios despliegues para resolver problemas.

- Probar el contrato en modo manual cuando aumentan las funcionalidades resulta tedioso. Por eso el uso de test automáticos resulta de gran ayuda. Sin embargo, en la mayoría de test unitarios que se han probado no se podía cambiar la dirección del emisor de la llamada al contrato (`msg.sender`), algo que resulta crucial para diferenciar entre propietario, profesor, estudiante y usuario no registrado. Finalmente con la herramienta *Foundry* se ha podido cambiar el remitente del mensaje. Sin embargo, si dentro de la función que el usuario llama se llama a otra función, el `msg.sender` vuelve a cambiar.

14 Planificación

Fase de planificación (13 de Diciembre - 15 de Diciembre)

- **Definición del propósito:** Especificar claramente el propósito y los objetivos de la plataforma. Identificar los problemas que la tecnología *blockchain* resolverá en el contexto de la plataforma de exámenes.
- **Análisis de mercado y usuarios:** Realiza un análisis y revisión de mercado para comprender la demanda y la competencia así como las propuestas ya publicadas. Perfilar a los usuarios finales y comprender sus necesidades y expectativas para poder realizar un enfoque correcto sobre la dirección que debe tomar el proyecto.

Fase de investigación y diseño (16 de Diciembre - 18 de Diciembre)

- **Diseño de la arquitectura:** Definir la arquitectura técnica de la DApp. Diseñar los contratos inteligentes necesarios. Justificación de la elección en función de los requisitos técnicos y de costos.

Fase de desarrollo (19 de Diciembre - 23 de Diciembre)

- **Desarrollo de contratos inteligentes:** Implementación de los contratos inteligentes que gestionarán la lógica de la plataforma.
- **Desarrollo de la interfaz de usuario:** Diseñar una interfaz intuitiva y fácil de usar para todos los usuarios.
- **Integración:** Integrar ambas partes para completar el desarrollo y obtener un resultado final funcional.

Fase de pruebas (futuro)

- **Realización de todo tipo de comprobaciones:** Realizar auditorías de seguridad para garantizar la robustez de los contratos inteligentes. Identificar y corregir posibles vulnerabilidades. Conducir pruebas exhaustivas con usuarios reales o simulados. Recopilar comentarios y realizar ajustes según sea necesario.

Marketing y lanzamiento (futuro)

- **Estrategia de lanzamiento:** Desarrollar una estrategia de lanzamiento para dar a conocer la plataforma. Considerar campañas de marketing digital y relaciones públicas.
- **Obtención de retroalimentación:** Facilitar canales para la retroalimentación de los usuarios después del lanzamiento.

Referencias

[1] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[2] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[3] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[4] ODEM. Odem: On-demand education marketplace, 2018.

[5] Fabian Vogelsteller and Vitalik Buterin. Eip 20: Erc-20 token standard. *Ethereum Improvement Proposals*, 20, 2015.

[6] Ethereum. Erc-900: Simple staking interface. https://eips.ethereum.org/EIPS/eip-900, 2018.

[7] Ethereum. Erc: Ethereum claims registry 780. https://github.com/ethereum/EIPs/issues/780, 2017.

[8] Natalie Smolenski and Kim Hamilton Duffy. Blockcerts: los bloques fundamentales de las blockchain para el aprendizaje a lo largo de la vida y el empleo en una economía global. In *Blockchain en Educación: Cadenas rompiendo moldes*, pages 119–130. Dykinson, 2018.

[9] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded merkle tree: Solving data availability attacks in blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 114–134. Springer, 2020.

[10] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.

[11] Fan Fang, Carmine Ventre, Michail Basios, Leslie Kanthan, David Martinez-Rego, Fan Wu, and Lingbo Li. Cryptocurrency trading: a comprehensive survey. *Financial Innovation*, 8(1):1–59, 2022.

[12] Brajesh De and Brajesh De. *API management*. Springer, 2017.

[13] Mark Masse. *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.", 2011.

[14] Monika Mehra, Manish Kumar, Anjali Maurya, Charu Sharma, et al. Mern stack web development. *Annals of the Romanian Society for Cell Biology*, 25(6):11756–11761, 2021.

[15] Ueli M Maurer and Stefan Wolf. The diffie–hellman protocol. *Designs, Codes and Cryptography*, 19(2-3):147–171, 2000.

[16] Vincent Rijmen and Joan Daemen. Advanced encryption standard. *Proceedings of federal information processing standards publications, national institute of standards and technology*, 19:22, 2001.

[17] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. Dhaes: An encryption scheme based on the diffie-hellman problem. *IACR Cryptol. ePrint Arch.*, 1999:7, 1999.

[18] Jeremy Wood. Cardano. https://cardano.org/, 2018.

[19] Da HongFei and Erik Zhang. Neo whitepaper. https://www.allcryptowhitepapers.com/neo-whitepaper/, 2017.

A Código contrato inteligente: *Smart Exam*

A.1 SmartExamBase.sol

```
// SPDX-License-Identifier: MIT.
pragma solidity 0.8.20;

import "@openzeppelin/contracts/access/Ownable.sol";


contract SmartExamBase is Ownable {

    // The owner is the one who creates the contract.
    constructor() Ownable(msg.sender){}

    /********************* - Parameters Exam - **************************/
    // Hash of the real exam file (to avoid tamper).
    string internal statement = "null";
    // Date of the latest the statement was uploaded.
    uint public dateLastUpload = block.timestamp;
    // Exact date when the exam should take place.
    uint public dateExam = block.timestamp;
    // Real date when the exam started.
    uint public dateStartExam = block.timestamp;
    // Duration of the exam in minutes.
    uint8 public duration = 0;
    // Price to enroll into the exam.
    uint24 public enrollingPrice = 0 wei;

    struct Student {
        /**
        * exams_enrolled: Mapping to know the exams that the student have been enrolled.
        * exams_done: Mapping to get the exams done by the student.
        */
        string submission;
        string correction;
        uint8 score;
    }

    // '_professors' stores the addresses of the professors added by the owner.
    mapping(address => bool) private _professors;
    // '_students' maps the addresses of the students with his/her struct information.
    mapping(address => Student) internal _students;
    // '_studAdds' array that store the addresses of the students.
    address[] internal _studAdds;
    // '_correctionCIDs' stores the CIDs of the corrections.
    string[] internal _correctionCIDs;

    // Check if an address matchs with a professor address.
    modifier onlyProfessor() {
        require(_professors[msg.sender] == true, "Only professors can access to this function.");
```

```
48          _;
49      }
50
51      // Check if the student exists.
52      modifier checkStudent(address _studAdd) {
53          bytes32 sub_bytes = keccak256(bytes(_students[_studAdd].
    submission));
54          require(sub_bytes != keccak256(bytes("")), "Student is not
    enrolled.");
55          _;
56      }
57
58      // Requires a submission to exist.
59      modifier checkSubmission(address _studAdd) {
60          bytes32 sub_bytes = keccak256(bytes(_students[_studAdd].
    submission));
61          require(sub_bytes != keccak256(bytes("null")), "Submission does
     not exist.");
62          _;
63      }
64
65      // Requires a submission to NOT exist.
66      modifier checkNOTSubmission(address _studAdd) {
67          bytes32 sub_bytes = keccak256(bytes(_students[_studAdd].
    submission));
68          require(sub_bytes == keccak256(bytes("null")), "Submission
    already exists.");
69          _;
70      }
71
72      // 'isOwner' check if sender is the owner.
73      function isOwner() public view returns(bool){
74          return msg.sender == owner();
75      }
76
77      // 'isProfessor' check if sender is a professor.
78      function isProfessor() public view returns(bool){
79          return _professors[msg.sender];
80      }
81
82      // 'isStudentEnrolled' check if sender is a student enrolled.
83      function isStudentEnrolled() public view returns(bool){
84          bytes32 sub_bytes = keccak256(bytes(_students[msg.sender].
    submission));
85          return (sub_bytes != keccak256(bytes("")));
86      }
87
88      /*************************************************************
89      /********************** - OWNER (EC) - ***********************
90      /*************************************************************/
91
92      // 'registerProfessor' function used by the owner to add professors
     addresses.
93      function registerProfessor(address _profAdd) external onlyOwner {
94          //We use the revert instead of require because it rollup the
    state of the contract and it does not use gas.
95          require(_professors[_profAdd] == false, "Professor already
```

```
    added.");
96          _professors[_profAdd] = true;
97      }
98
99      // 'deleteProfessor' function used by the owner to add professor
    addresses.
100     function deleteProfessor(address _profAdd) external onlyOwner {
101         require(_professors[_profAdd] == true, "Professor does not
    exist.");
102         _professors[_profAdd] = false;
103     }
104
105     // 'editExamParameters' function used by owner to set and edit
    exams.
106     function editExamParameters(
107         uint _dateExam,
108         uint8 _duration,
109         uint24 _enrollingPrice
110     ) external onlyOwner {
111         dateExam = _dateExam;
112         dateStartExam = _dateExam;
113         duration = _duration;
114         enrollingPrice = _enrollingPrice*1 wei;
115         dateLastUpload = block.timestamp;
116     }
117
118     // 'startExam' unction used by the owner to start the exam adding
    the CID of the exam.
119     function startExam(
120         string memory _statement
121     ) external onlyOwner {
122         bytes32 sub_bytes = keccak256(bytes(statement));
123         require(sub_bytes == keccak256(bytes("null")), "Exam already
    started.");
124         require(block.timestamp >= dateExam, "Cannot start a exam
    before the dateExam.");
125         statement = _statement;
126         dateStartExam = block.timestamp;
127     }
128
129     // 'withdraw' function used by the owner to get the ether stored in
     the contract address
130     // from the students enrolling payments.
131     function withdraw() external onlyOwner {
132         payable(owner()).transfer(address(this).balance);
133     }
134
135     // 'getStatement' allows owner, professors and students to get
    statement CID.
136     function getStatement() external view returns(string memory){
137         require(isOwner() || isProfessor() || isStudentEnrolled() ==
    true, "You cannot access to the statement");
138         return statement;
139     }
140 }
```

Listing 1: SmartExam.sol

A.2 SmarExam.sol

```
// SPDX-License-Identifier: MIT.
pragma solidity 0.8.20;

import "./SmartExamBase.sol";

contract SmartExam is SmartExamBase{

    /*************************************************************
    ********************** - PROFESSOR - **********************
    *************************************************************/


    // 'getStudents' let a professor to obtain the address of the enrolled students.
    function getStudents() external view onlyProfessor returns (address[] memory) {
        return _studAdds;
    }

    // 'getStudentSubmission' let a professor to obtain the submission CID of a student
    // from his/her address.
    function getStudentSubmission(
        address _studAdd
    ) external view onlyProfessor checkStudent(_studAdd) checkSubmission(_studAdd) returns (string memory) {
        return _students[_studAdd].submission;
    }

    // 'setCorrection' let a professor add a correction for an exam.
    function setCorrection(
        address _studAdd,
        string memory _correction,
        uint8 _score
    ) external onlyProfessor checkStudent(_studAdd) checkSubmission(_studAdd) {
        require((_score >= 0) && (_score <= 10), "The score have to be between 0 and 10");
        _students[_studAdd].correction = _correction;
        _students[_studAdd].score = _score;
        _correctionCIDs.push(_correction);
    }

    // 'getCorrections' let a professor to obtain all the corrections.
    function getCorrections() external view onlyProfessor returns (string[] memory) {
        return _correctionCIDs;
    }

    /*************************************************************
    ********************** - STUDENT - **********************
    *************************************************************/

    // 'enroll' allows an address to enroll into an exam.
```

```
48      function enroll() external payable {
49          require(msg.value == enrollingPrice*1 wei, "Pay the exact
    amount of money.");
50          require(block.timestamp <= dateStartExam, "Exam is not
    available.");
51          Student memory s = Student("null", "null", 0);
52          _students[msg.sender] = s;
53          _studAdds.push(msg.sender);
54      }
55
56
57      // 'setSubmission' let a student stores its submission CID.
58      function setSubmission(
59          string memory _submission
60      ) external checkStudent(msg.sender) checkNOTSubmission(msg.sender)
    {
61          bytes32 sub_bytes = keccak256(bytes(statement));
62          require(sub_bytes != keccak256(bytes("null")), "Exam is not
    available yet.");
63          require(block.timestamp <= dateStartExam + duration*60, "ERROR:
     Submission out of date.");
64          _students[msg.sender].submission = _submission;
65      }
66
67      // 'getMySubmission' getter to reach the submission CID of the
    student that
68      // is calling the funciton.
69      function getMyExam() external view returns(string memory) {
70          return _students[msg.sender].submission;
71      }
72
73      // 'getMyCorrection' getter to reach the correction CID of the
    student that
74      // is calling the funciton.
75      function getMyCorrection() external view returns(string memory) {
76          return _students[msg.sender].correction;
77      }
78
79      // 'getMyScore' getter to reach the score of the student that
80      // is calling the funciton.
81      function getMyScore() external view returns(uint8) {
82          return _students[msg.sender].score;
83      }
84
85      // 'certificateStudent' getter to confirm that a student pass or
    not.
86      function certificateStudent(address _studAdd) external view
    checkStudent(_studAdd) returns(bool) {
87          return (_students[_studAdd].score >= 5);
88      }
89  }
```

Listing 2: SmartExam.sol

B Diagrama UML