Trabajo Práctico Profesional

Decentraveller - Una aplicación basada en tecnología blockchain para intercambiar información de puntos turísticos

Alumno	**Padrón**
Gianmarco Cafferata	99423
Matías Scakosky	99627
Uriel Kelman	99616

Tutor: Ing. Diego Essaya
Co-tutor: Ing. Agustín Rojas

December, 2023

https://github.com/urielkelman/decentraveller

Contents

1. Introducción

El presente documento tiene como objetivo detallar el desarrollo de Decentraveller, una aplicación social que permite a sus usuarios crear contenido sobre puntos turísticos y participar activa y democráticamente en la moderación del mismo. Para cumplir dicha finalidad, esta aplicación se basa en tecnología blockchain, a través de la cual se permite la participación, el almacenamiento y el acceso de datos en forma descentralizada, asegurando la transparencia y seguridad de la información.

En la actualidad, la mayoría de las aplicaciones informáticas se encuentran gobernadas a través de una entidad centralizada, como puede ser una compañía, una ONG, o cualquier tipo de corporación. Dichas entidades son las encargadas de confeccionar, en forma centralizada y generalmente sin la participación activa de sus usuarios, las reglas que rigen las aplicaciones de las cuales son propietarias. La centralización en las aplicaciones contemporáneas es un fenómeno común que, desafortunadamente, puede conllevar a arbitrariedades por parte de quien pone las reglas. Esta concentración de control no solo abre la puerta a la censura, sino que también puede dar lugar a decisiones motivadas por intereses ajenos a los de los usuarios finales de una aplicación.

Decentraveller se distingue de las aplicaciones tradicionales al empoderar a sus usuarios con herramientas que les permiten participar activamente en la definición de las normativas que rigen la plataforma. Esta iniciativa no solo involucra a los usuarios en la creación de reglas, sino que también los integra de manera directa en el proceso de moderación, asignándoles roles específicos para asegurar su cumplimiento. Además, Decentraveller implementa un sistema de resolución de disputas centrados en la toma de decisiones democráticas por parte de la comunidad de usuarios, garantizando así una gestión transparente y equitativa de los conflictos que puedan surgir en torno a la moderación. Esta visión colaborativa y descentralizada es el pilar fundamental sobre el cual se construye una experiencia única y democrática para todos los participantes de la plataforma.

Para llevar adelante su visión, las interacciones de los usuarios están regidas por una DAO, acrónimo de *Decentralized Autonomous Organization*, u Organización Autónoma Descentralizada. Una DAO es una forma de organizar y hacer funcionar organizaciones en la blockchain, haciendo uso de *smart contracts* para brindar transparencia, inmutabilidad, autonomía y seguridad a las mismas. Una de sus más notables capacidades es que una DAO permite a sus usuarios realizar votaciones para implementar propuestas nuevas y votar en ellas. En su formato más típico, la gobernanza de una DAO, entendida por el conjunto de usuarios que son partícipes de la DAO y se encuentran habilitados para realizar acciones en la misma, está determinada por aquellos que poseen un *token* que funciona como un cuantificador de la partipación. Con el objetivo de que aquellos usuarios que más vinculados se encuentran a la aplicación sean los que mayor poder de participación tengan en la DAO, estos tokens son otorgados a los usuarios al crear contenido en Decentraveller.

Además de innovar utilizando tecnología blockchain para democratizar el acceso y la participación de sus usuarios a través de una DAO, Decentraveller busca brindar una experiencia de usuario agradable mezclando el mundo descentralizado con el mundo centralizado. La tecnología blockchain aún se encuentra en un estadío sumamente temprano por lo que presenta ciertas falencias a la hora de tratar a los datos. Estas falencias se traducen en que resulta complejo procesar grandes volúmenes de datos almacenados en una blockchain en forma eficiente. Debido a esto, la construcción de Decentraveller implicó no solamente el almacenamiento de los datos y transacciones en la blockchain, sino que también fueron introducidos componentes centralizados tales como servidores, que también almacenan y sirven datos a la aplicación y además cumplen un rol fundamental en brindar a los usuarios la mejor experiencia posible. Estos servidores facilitan la agregación y el acceso eficiente a los datos y permiten, también, la implementación de *features* adicionales no dependientes de la blockchain. Un ejemplo destacado son los sistemas de recomendación personalizados, que enriquecen significativamente la experiencia del usuario al ofrecer contenidos en base a sus intereses y preferencias.

Finalmente, la interfaz de usuario de Decentraveller se ha diseñado específicamente para dispositivos móviles, decisión fundamentada en la creciente prevalencia de los *smartphones* en la vida cotidiana. En los últimos años, el uso de teléfonos inteligentes se ha masificado, convirtiéndose en una herramienta indispensable para la mayoría de las personas, con el complementp de que numerosas empresas de tecnología están priorizando el desarrollo de aplicaciones móviles sobre las versiones web, enfocándose en mejorar la experiencia del usuario en estos dispositivos. Además de intentar subirse a esta tendencia, la naturaleza propia de la aplicación, centrada en el intercambio de opiniones y críticas sobre lugares turísticos, se beneficia de la inmediatez que ofrece una plataforma móvil permitiendo a los usuarios compartir sus experiencias y críticas en tiempo real o justo después de visitar un punto turístico.

2. Escenarios

2.1 Casos de uso

Se comienza la descipción de la plataforma construida con un diagrama de casos de uso. El mismo ofrece una visión panorámica de los distintos escenarios y funcionalidades a las cuales puede acceder un usuario de la aplicación. Desde la creación de contenido hasta la participación en la moderación y toma de decisiones de la DAO, este diagrama es una herramienta clave para comprender la amplitud y profundidad de la experiencia de usuario que ofrece nuestra plataforma.

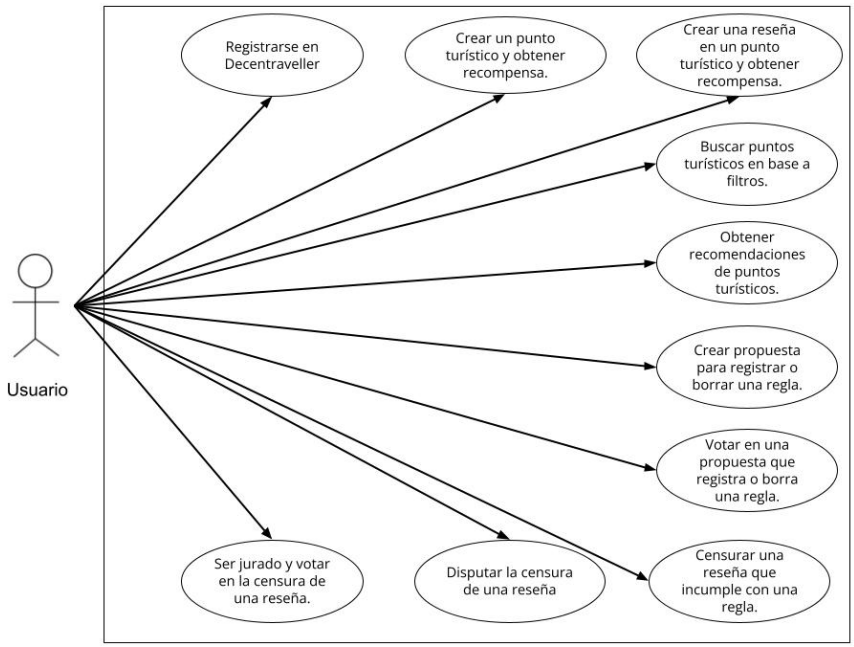


Figura 1: Diagrama de casos de uso de Decentraveller.

2.2 Descripción de casos de uso

Registrarse en Decentraveller

Descripción: Consiste en que un nuevo usuario de Decentraveller pueda crear su perfil para hacer uso de la plataforma.

Actores: Usuario.

Precondiciones:

1. El usuario ha conectado su billetera a la aplicación.
2. El usuario no se encuentra registrado previamente.
3. El usuario posee suficiente *gas* para pagar por la transacción de registro.

Flujo básico:

1. El caso de uso comienza cuando el Usuario quiere ingresar en la aplicación y no posee un perfil.

2. El sistema muestra un formulario a llenar con campos correspondientes a un pseudónimo, a un país, y al interés principal de quien se registra.

3. El usuario ingresa los datos solicitados y crea la transacción de registro en su *wallet*.

4. El sistema registra al usuario y este puede acceder a las pantallas principales de la aplicación.

Flujos de excepción:

1. La dirección de la *wallet* del usuario se ha registrado previamente. El sistema indica que hubo un problema en la registración.

2. Existe otro usuario previamente registrado con el mismo pseudónimo. El sistema indica que hubo un problema en la registración.

Crear un punto turístico y obtener recompensa

Descripción: Consiste en que un usuario de la aplicación pueda dar de alta un punto turístico de interés para que tanto él como el resto de usuarios pueda escribir reseñas para el mismo.

Actores: Usuario.

Precondiciones:

1. El Usuario ejecutó previamente el caso de uso *Registrarse en Decentraveller*.

2. El usuario ha conectado su billetera a la aplicación.

3. El usuario posee suficiente *gas* para pagar por la transacción de creación del punto turístico.

Flujo básico:

1. El caso de uso comienza cuando el Usuario quiere registrar un nuevo punto turístico en la plataforma.

2. El sistema muestra un botón que permite ser redirigido a una pantalla específica para la creación del punto turístico.

3. El usuario presiona dicho botón y el sistema despliega un formulario con los datos que debe ingresar el usuario para registrar el punto turístico.

4. El Usuario ingresa el nombre, la dirección y la categoría del punto que quiere registrar, y crea la transacción en su *wallet*.

5. El sistema registra el punto turístico, el cual empezará a mostrarse en los listados de las pantallas pertinentes para que los usuarios registrados puedan escribir reseñas.

6. El sistema otorga una cantidad de *tokens* en formato de recompensa por haber generado contenido en la aplicación.

Flujos de excepción:

1. Existe registrado un punto turístico con la misma geolocalización.

Crear una reseña en un punto turístico y obtener recompensa

Descripción: Consiste en que un Usuario seleccione un punto turístico existente y cree una reseña puntuando al mismo.

Actores: Usuario.

Precondiciones:

1. El Usuario que escribirá la reseña ejecutó previamente el caso de uso *Registrarse en Decentraveller*.

2. El Usuario ha conectado su billetera a la aplicación.

3. Algún usuario de la plataforma ha ejecutado satisfactoriamente el caso de uso *Crear un punto turístico y obtener recompensa*, registrando el punto turístico donde será dejada la reseña.

4. El usuario posee suficiente *gas* para pagar por la transacción de creación de la reseña.

Flujo básico:

1. El caso de uso comienza cuando un Usuario desea escribir una reseña con puntaje de algún punto turístico existente.

2. A partir de alguna de las pantallas que muestran listados de puntos turísticos, el Usuario selecciona uno de ellos y el sistema lo lleva al detalle de dicho punto.

3. El sistema muestra un botón para dejar una reseña en el punto turístico.

4. El Usuario selecciona dicho botón y el sistema despliega un formulario con los datos para dar de alta la reseña.

5. El Usuario ingresa un texto, imágenes y un puntaje entre 1 y 5 que conformarán los datos de la reseña, y crea la transacción en su *wallet*.

6. El sistema registra la reseña, la cual comenzará a mostrarse en el detalle del punto turístico y estará sujeta a moderación por otros usuarios.

7. El sistema otorga una cantidad de tokens en formato de recompensa por haber generado contenido en la aplicación.

Flujos de excepción: N/A.

Buscar puntos turísticos en base a filtros

Descripción: Consiste en que un usuario pueda buscar puntos turísticos en base a ciertos filtros que permitan enriquecer su experiencia en la plafatorma.
Actores: Usuario

Precondiciones:

1. El Usuario ejecutó previamente el caso de uso *Registrarse en Decentraveller*.
2. El usuario ha conectado su billetera a la aplicación.

Flujo básico:

1. El Usuario se dirige a la pantalla de exploración de puntos turísticos.
2. El Usuario selecciona la ubicación en dónde le gustaría encontrar lugares.
3. El sistema despliega opciones que permiten al usuario añadir filtros adicionales tales como la distancia de la ubicación seleccionada, la puntuación de los puntos, el orden en que quiere que se muestren los resultados, etc.
4. El usuario ingresa el valor de los filtros a su gusto.
5. El sistema muestra los puntos turísticos resultantes de la búsqueda.

Flujos de excepción:

1. El sistema no encuentra ningún punto turístico en base a los filtros seleccionados por el usuario.

Crear propuesta para registrar o borrar una regla

Descripción: Consiste en que un usuario puede registrar una propuesta para registrar una nueva regla o borrar una regla existente con la intención de modificar el listado de reglas de Decentraveller. Dicha propuesta será votada por otros usuarios de la aplicación.

Actores: Usuario.

Precondiciones:

1. El Usuario ejecutó previamente el caso de uso *Registrarse en Decentraveller*.

2. El Usuario ha conectado su billetera a la aplicación.

3. El usuario posee una cantidad mínima de *tokens* de Decentraveller, indicando que previamente ha generado contenido en la plataforma, es decir que ejecutó el caso de uso *Crear un punto turístico y obtener recompensa* y/o *Crear una reseña en un punto turístico y obtener recompensa*.

4. El usuario posee suficiente *gas* para pagar por la transacción de creación de propuesta.

Flujo básico:

1. El Usuario se dirige a la pantalla que contiene toda la información de las reglas comunitarias.

2. El sistema muestra un listado de todas las reglas actuales, además de las votaciones en curso, y dos botones: un botón que permite registrar una nueva regla o, seleccionando una regla existente, un botón que permite proponer su eliminación.

3. El Usuario presiona el botón deseado, según quiera proponer un alta o una eliminación en el listado de reglas, y crea la transacción en su *wallet* para tal fin.

4. El sistema registra la propuesta y da comienzo a un período de tiempo en el cual los usuarios pueden ver la propuesta pero no votar. Dicho período tiene como finalidad anunciar que comenzará una votación.

5. El sistema da inicio a la votación y durante cierto período de tiempo todos los usuarios partícipes de la DAO (aquellos que tenían *tokens* al momento de registrar la propuesta) podrán votar a favor o en contra de la misma proporcionalmente a su posesión de *tokens*.

6. Si la propuesta es perdedora terminada la votación, finaliza el flujo. En caso de que la propuesta resulte ganadora pasado el tiempo de votación, el Usuario genera una última transacción utilizando su billetera para terminar de dar de alta o baja la propuesta en el listado.

7. El sistema incorpora o elimina la propuesta al listado de reglas según corresponda.

Flujos de excepción:

1. El usuario no posee la cantidad suficiente de *tokens* para registrar una propuesta de alta o baja de una regla.

Votar en una propuesta que registra o borra una regla

Descripción: Consiste en que un usuario pueda votar a favor o en contra en una propuesta creada por otro usuario para dar de alta o de baja una regla en el listado de reglas comunitarias.

Actores: Usuario.

Precondiciones:

1. El Usuario que quiere votar debe tener poder de voto; esto significa que debe haber generado contenido para tener *tokens*. El poder de voto está definido por la cantidad de *tokens* en el momento en el que se crea la propuesta.
2. Otro usuario debe haber ejecutado el caso de uso *Crear propuesta para registrar o borrar una regla*, hasta el punto en el que se abre la votación para la comunidad.
3. El usuario posee suficiente *gas* para pagar por la transacción de votación.

Flujo básico:

1. El caso de uso comienza cuando el Usuario quiere participar activamente en la votación de propuestas de reglas comunitarias.
2. El sistema muestra un listado de las votaciones en curso.
3. El usuario elige la votación en la que le desea votar, y el sistema muestra el detalle de la votación.
4. El usuario indica si desea votar a favor o en contra de la propuesta, y crea una transacción en su *wallet* para tal fin.

Flujos de excepción:

1. El usuario no tenía poder de voto (*tokens*) al momento de creación de la propuesta.
2. El tiempo de votación de la propuesta expiró.

Censurar una reseña que incumple con una regla

Descripción: Consiste en que un usuario moderador puede censurar una reseña la cual considera que está incumpliendo con alguna regla vigente de la comunidad. Dicha reseña dejará de estar visible para los usuarios de la comunidad con la excepción de quien la escribió.

Actores: Usuario.

Precondiciones:

1. El Usuario ejecutó previamente el caso de uso *Registrarse en Decentraveller*.
2. El Usuario posee el rol de **moderador**.
3. Otro usuario ejecutó el caso de uso *Crear una reseña en un punto turístico y obtener recompensa*.
4. La reseña a censurar no fue censurada previamente, es decir que no debe haber existido una disputa ya resuelta para esa reseña.
5. El usuario posee suficiente *gas* para pagar por la transacción de censura.

Flujo básico:

1. El Usuario con rol de moderador se encuentra situado en la pantalla de detalle de un punto turístico.
2. El sistema muestra al Usuario un listado de reseñas creadas para este punto turístico.
3. El Usuario puede presionar en cualquier de estas reseñas para acceder al detalle de las mismas.
4. El sistema muestra al Usuario el detalle de la reseña, y un botón que le permite censurarla.
5. El Usuario presiona dicho botón.
6. El sistema le muestra al usuario un listado de las reglas vigentes, para que seleccione cuál es aquella que está incumpliendo la reseña a censurar.
7. El Usuario selecciona cuál es la regla incumplida y crea una transacción en su *wallet* que censura la reseña.
8. El sistema cambia el estado de la reseña, y esta deja de mostrarse al resto de los usuario de Decentraveller.

Flujos de excepción:

1. El Usuario que intenta censurar no posee el rol de moderador.
2. La reseña ha sido censurada previamente.
3. La reseña posee una disputa vigente, cuya resolución aún no fue votada.
4. La regla incumplida fue eliminada por una propuesta anteriormente.

Disputar la censura a una reseña

Descripción: Consiste en que un usuario que haya escrito una reseña para un punto turístico y esta reseña sea censurada, pueda disputar la censura sometiéndola a la votación de un jurado conformado por miembros de la comunidad.

Actores: Usuario.

Precondiciones:

1. El Usuario ejecutó previamente el caso de uso *Registrarse en Decentraveller*.
2. El Usuario ejecutó el caso de uso *Crear una reseña en un punto turístico y obtener recompensa*.
3. Otro usuario con rol de moderador ejecutó el caso de uso *Censurar una reseña que incumple con una regla* censurando la reseña dada de alta por el Usuario.
4. El usuario posee suficiente *gas* para pagar por la transacción que origina la disputa.

Flujo básico:

1. Al ingresar en el detalle la reseña, el sistema indica al Usuario que esta ha sido censurada por un moderador, en conjunto con un botón para iniciar una disputa por esta censura.
2. El Usuario presiona este botón, creando una transacción en su *wallet* para iniciar la disputa.
3. El sistema registra la disputa y elige una serie de jurados al azar que deben tener en su poder *tokens* de la comunidad. Cada uno de estos usuarios tiene un voto para apoyar la continuidad o la remoción de la censura. Además, la reseña vuelve a ser visible para el resto de los usuarios, con la aclaración de que se encuentra en proceso de disputa.
4. Los jurados proceden a votar a favor o en contra de la censura.
5. En el caso en el que gane la disputa el usuario que busca remover la censura, el sistema muestra al usuario, en el detalle de la reseña, un botón para ejecutar la transacción que cambia el estado de la censura.
6. El Usuario pulsa este botón y crea una transacción en su *wallet* para ejecutar la remoción de la censura.
7. El sistema registra el cambio de estado y la reseña vuelve a mostrarse como si la censura jamás hubiese ocurrido.

Flujos de excepción:

1. La reseña no se encuentra censurada.
2. El Usuario no es el dueño de la reseña censurada.

Ser jurado y votar en la censura de una reseña

Descripción: Consiste en que un usuario perteneciente a la comunidad sea seleccionado como jurado en la disputa de la censura de una reseña, para que se exima votando para mantener o remover la censura disputada.

Actores: Usuario.

Precondiciones:

1. El Usuario ejecutó previamente el caso de uso *Registrarse en Decentraveller*.
2. El Usuario posee *tokens* de Decentraveller, por lo que previamente debe haber ejecutado el caso de uso *Crear un punto turístico y obtener recompensa* y/o *Crear una reseña en un punto turístico y obtener recompensa*.
3. Un primer usuario ejecutó previamente el caso de uso *Crear una reseña en un punto turístico y obtener recompensa*.
4. Un segundo usuario con rol de moderador ejecutó el caso de uso *Censurar una reseña que incumple con una regla*, censurando la reseña creada por el primero.
5. El primer usuario ejecutó el caso de uso *Disputar la censura a una reseña* hasta el punto donde se seleccionan los usuarios que oficiarán de jurados y se abre la votación.
6. El usuario posee suficiente *gas* para pagar por la transacción de votación.

Flujo básico:

1. El sistema muestra al Usuario, en el detalle de la reseña, que puede votar a favor o en contra de la censura.
2. El Usuario crea una transacción en su *wallet* para votar mantener o remover la censura, según considere.
3. El sistema registra el voto del usuario y lo contabiliza para la resolución de la disputa.

Flujos de excepción:

1. La reseña no se encuentra con una disputa en curso.
2. El Usuario no ha sido seleccionado como jurado en la disputa.
3. La votación ha finalizado.

3. Arquitectura - Vista física

3.1 Componentes del sistema

La arquitectura general del sistema está compuesta por múltiples componentes diseñados para operar en conjunto de manera colaborativa, con el fin de ofrecer una experiencia de usuario óptima. El siguiente *Diagrama de robustez* detalla cómo estos componentes están organizados y cómo interactúan entre sí para formar la estructura integral del sistema.

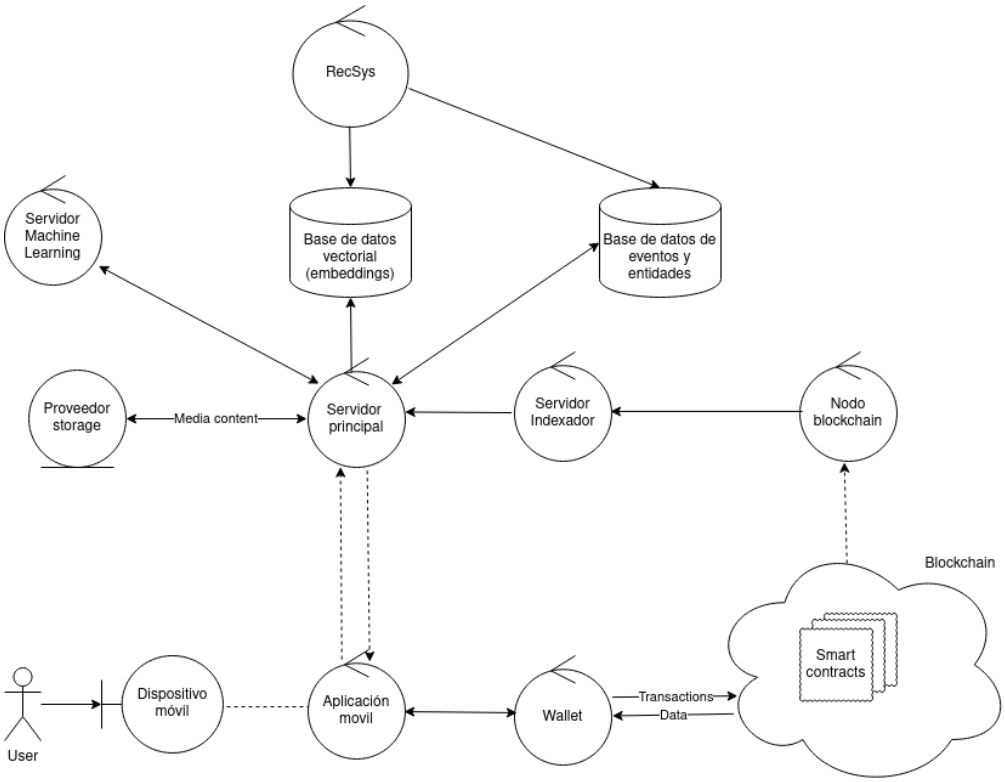


Figura 2: Arquitectura general del sistema con sus componentes.

El instrumento que actúa como frontera para que el usuario pueda comenzar a interactuar con los distintos aspecto del sistema es su **dispositivo móvil**. Para poder acceder al mismo, debe tener instalada la aplicación de Decentraveller, la cual se encuentra diseñada para correr en teléfonos celulares compatibles con sistema operativo *Android*.

La primera de las funcionalidades que cumple la **aplicación móvil** dentro del sistema es la de permitir a los usuarios enviar transacciones a la *blockchain*. Para esto, para cada uno de los flujos que implica generar y en-

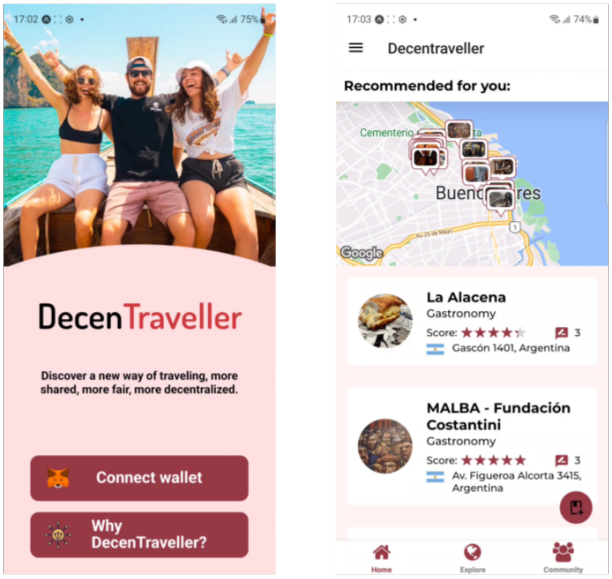


Figura 3: Pantallas de la aplicación móvil

viar una transacción, la aplicación construye la transacción con todos datos y metadatos necesarios y, utilizando una ***wallet*** (con la que se comunica utilizando la herramienta *WalletConnect*), envía esta transacción hacia la ***blockchain***. Este proceso implica una colaboración efectiva entre la **aplicación móvil** y la **wallet**, siendo esta última un componente externo crucial que actúa como un intermediario, o 'traductor', entre la **aplicación** y la infraestructura de la **blockchain**.

Una vez que una transacción llega a la ***blockchain***, será ejecutada alguna función del ecosistema de *smart contracts* de Decentraveller que se encuentran desplegados en ella. Dichos contratos poseen la lógica del sistema para generar y almacenar contenido en forma descentralizada, permitir la existencia de la DAO, remunerar a los usuarios por su participación, etc. A su vez, para cada una de las transacciones relevantes, las funciones invocadas en los *smart contracts* están programadas para lanzar *eventos* que luego pueden ser consumidos por el **servidor indexador**.

La responsabilidad principal del **servidor indexador** consiste en, a partir de suscripciones definidas a ciertos eventos específicos lanzados por ciertos contratos desplegados en la **blockchain**, escuchar por estos eventos para procesarlos y luego enviarlos al servidor principal utilizando su API Http. La suscripción a estos eventos se realiza a través de la comunicación establecida contra el **nodo blockchain**, el cual almacena datos de la blockchain y envía los eventos almacenados hacia sus suscriptores a través de una abstracción conocida

como *WebSocket*.

El **servidor principal** es uno de los componentes más importantes del sistema ya que concentra varias funcionalidades críticas. En primer lugar, recibe los eventos indexados por el **nodo indexador** y los almacena en una base de datos relacional, componente denominado **base de datos de eventos y entidades**. En línea con la responsabilidad de almacenar los datos indexados de la blockchain de forma de que estos puedan ser consumidos en forma más eficiente que leyéndolos de la misma en forma directa, ofrece una API rica en *endpoints* que permite a la **aplicación móvil** acceder a los datos indexados. Un gran cantidad de los *endpoints* disponibilizados realizan agregaciones complejas sobre los datos; agregaciones que realizadas a través de una lectura cruda sobre la **blockchain** serían sumamente ineficientes de realizar.

Alrededor del **servidor principal** nos encontramos con una serie de componentes satélites que contribuyen al objetivo de enriquecer la experiencia de usuario. Observando el diagrama es posible ver que son tres los componentes de almacenamiento: en primer lugar la **base de datos de eventos y entidades**, la cual ya se ha nombrado; en segundo lugar, la **base de datos de embeddings**; y en tercer lugar el **proveedor de Storage**.

La **base de datos de embeddings** almacena el resultado de correr el proceso llevado a cabo por el componente **RecSys**, abreviatura de sistema de recomendación. Este proceso consume datos almacenados por el **servidor principal** en la **base de eventos y entidades**, y a partir de ello corre un algoritmo que genera una serie de *embeddings* que representan a los distintos puntos turísticos en formato vectorial. Estos *embeddings* son almacenados en la **base de datos vectorial**, y luego son leídos por el **servidor principal** para que de esta forma se pueda realizar la recomendación de contenido a los usuarios de la aplicación (para comprender el mecanimos de generación de *embeddings*ver sección **Algoritmo de recomendación de contenido**).

Por último, tenemos al **proveedor de Storage**. Este componente es un nodo conectado a la red IPFS, y su función principal es la de almacenar contenido multimedia. Cuando los usuarios desean crear una nueva reseña, se les ofrece la posibilidad de adjuntar imágenes en la misma. Estas imágenes no pueden ser almacenadas en la **blockchain** ya que esta no es compatible con archivos multimedia; por lo tanto, las imágenes son guardadas en el nodo que participa del protocolo de almacenamiento descentralizado IPFS, y en la **blockchain** únicamente se almacena un **hash** que referencia unívocamente a cada una de las imágenes. Cuando los usuarios desean ver alguna de estas imágenes, el **servidor principal** puede servirlas accediendo al **nodo IPFS** utilizando la referencia de la imagen buscada.

4. Vista de procesos

En esta sección, se presentan una serie de diagramas de secuencia que permiten distinguir con mayor detalle la interacción de los componentes del sistema para determinados flujos. Estas visualizaciones no solo buscan aclarar la funcionali-

dad y el flujo de trabajo del sistema, sino que también destacan las relaciones y dependencias claves entre sus distintas partes, proporcionando así una comprensión integral de la arquitectura operativa del sistema.

4.1 Creación de información (punto turístico, reseña o reglas) en la plataforma

El siguiente diagrama busca representar en forma genérica la generación de contenido e información de los usuarios en la plataforma. No hace alusión a un flujo en particular, sino que busca ser representativo de todos aquellos flujos que terminen creando algún tipo de información en el sistema para que luego sea consumido por el resto de los usuarios.

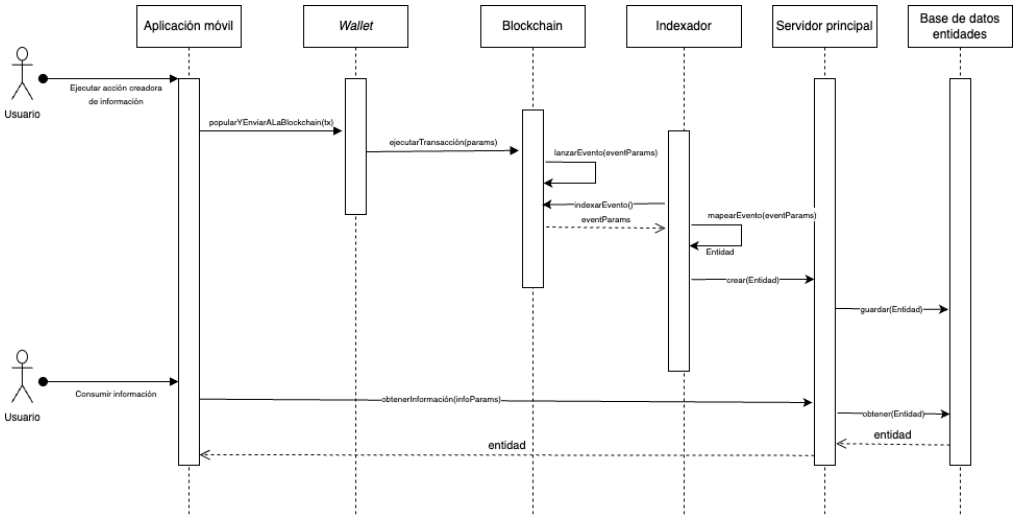


Figura 4: Diagrama de secuencia de creación y almacenamiento de información genérica.

El flujo comienza cuando un usuario desea generar contenido en Decentraveller. Para esto, la **aplicación móvil** desplegará algún elemento, tal como pueden ser un formulario o un botón que dispara una acción particular, y a partir del accionar del usuario creará la transacción con todos los datos pertinentes. Utilizando una **wallet** como intemediaria, que también debe estar instalada en su dispositivo móvil (como por ejemplo puede ser *Metamawk*), la transacción es ejecutada en la *blockchain* con los parámetros definidos por el accionar del usuario. La ejecución de la transacción implica a su vez la ejecución de una función en alguno de los *smart contracts* de Decentraveller. Las funciones están programadas para que, cuando se guarda algún tipo de información, sea lanzado un evento predefinido con los datos que interesa guardar *off-chain*. Este evento es indexado por el **servidor indexador**, el cual aplica una transformación sobre los datos recibidos adaptándolos a los esperados por la interfaz del **servidor**

principal, y se los envía mediante una solicitud HTTP. El **servidor principal** recibe los datos y a partir de ellos crea una entidad que es almacenada en la **base de datos de entidades**. Esto permite que luego, ya sea el mismo usuario o uno distinto, pueda recuperar la información de la entidad guardada invocando algún endpoint HTTP del **servidor principal**, el cuál realizará una consulta a la **base de datos de entidades** para obtener la información deseada y terminará retornándola al usuario solicitante.

4.2 Generación y consumo de recomendaciones

El siguiente diagrama busca ilustrar las interacciones entre componentes del flujo mediante el cual un usuario puede obtener recomendaciones de puntos turísticos.

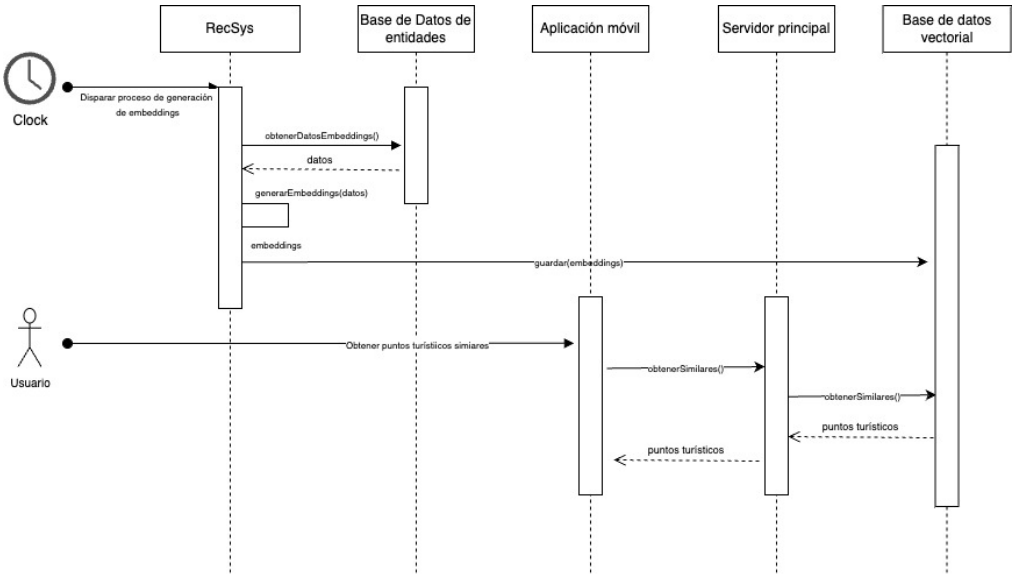


Figura 5: Diagrama de secuencia de creación, almacenamiento y consumo de *embeddings* para recomendaciones

El flujo comienza cuando algún *trigger* programado para correr cada una determinada cantidad de tiempo (como por ejemplo podría ser un *crontab*) inicia la ejecución del proceso **RecSys**. Este proceso realiza una serie de consultas para obtener ciertos datos de la **base de datos de entidades**, y sobre ellos corre un algoritmo de procesamiento cuya salida final es una serie de *embeddings* que son almacenados en la **base de datos vectorial**. Cuando un usuario invoca un endpoint de recomendaciones a través de la **aplicación móvil**, como por ejemplo puede ser el endpoint para obtener los puntos turísticos similares a otro, el **servidor principal** recupera los embeddings almacenados en la **base de datos vectorial** y realiza una evaluación para obtener los similares y retornarlos

al usuario solicitante.

5. Vista lógica

En esta sección se presentan un conjunto de diagramas que intentan mostrar la organización de las distintas abstracciones y entidades que forman parte de Decentraveller y la forma en la que se relacionan entre ellas.

5.1 Diagrama de clases servidor principal

El siguiente diagrama presenta las distintas clases que podemos encontrar en el **servidor principal**. Debe notarse que el mismo no busca mostrar las clases que son propias de las decisiones de arquitectura interna del servidor tales como controladores, adaptadores, conectores, etc. sino que su objetivo es mostrar las entidades que terminan por ser almacenadas en la base de datos de entidades para su posterior consumo por parte de los usuarios. Estas entidades representadas en clases son guardadas utilizando un ORM(*Object relational mapping*), por lo que presentan una relación uno a uno con la información que se termina guardando en las distintas tablas.

Un **Profile** es creado en el servidor principal cada vez que un usuario se registra en Decentraveller. Al crear su perfil el servidor le asignará un *avatar* en formato de imagen, cuyos datos estáran presentes en la entidad **Image**, que contiene la información para encontrar la imagen en IPFS. Al haberse registrado, se le permitirá al usuario interactuar con el sistema creando contenido de distintas maneras. El contenido que puede crear un usuario puede ser en forma de un **Place**, es decir agregando un lugar nuevo, o en forma de una **Review**, agregando una reseña a un **Place** existente. Adicionalmente, pueden optar por agregar imágenes a su reseña lo que creará una **ReviewImage** por cada imagen agregada, la cual estará asociada a su **Review** correspondiente. Por otro lado, los usuarios pueden participar en la comunidad y para ello proponer una **Rule** nueva, generando una propuesta para tal fin, que de ser aprobada agregará una nueva regla las listado de reglas vigentes de la comunidad. A partir de este listado, una **Review** puede ser censurada si un usuario considera que existe una **Rule** incumplida, lo que llevará a que la censura sea sometida a una disputa en la que votarán distintos jurados a favor de mantener o remover la censura.

5.2 Diagrama de contratos

Dado que se puede establecer un paralelismo entre la programación de *smart contracts* con la programación orientada a objetos, en donde un contrato deployado en la blockchain con sus funciones y variables son similares a una instancia de un objeto con sus métodos y atributos, se ha elegido utilizar un diagrama de clase para representar al ecosistema de contratos que existen en Decentraveller.

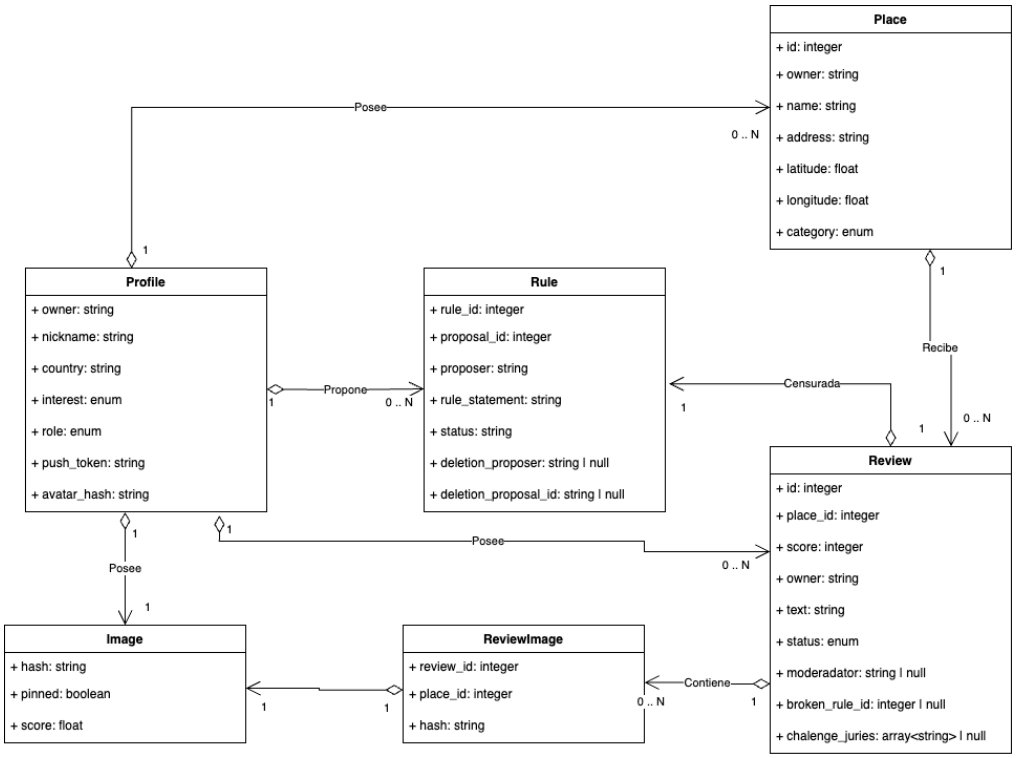
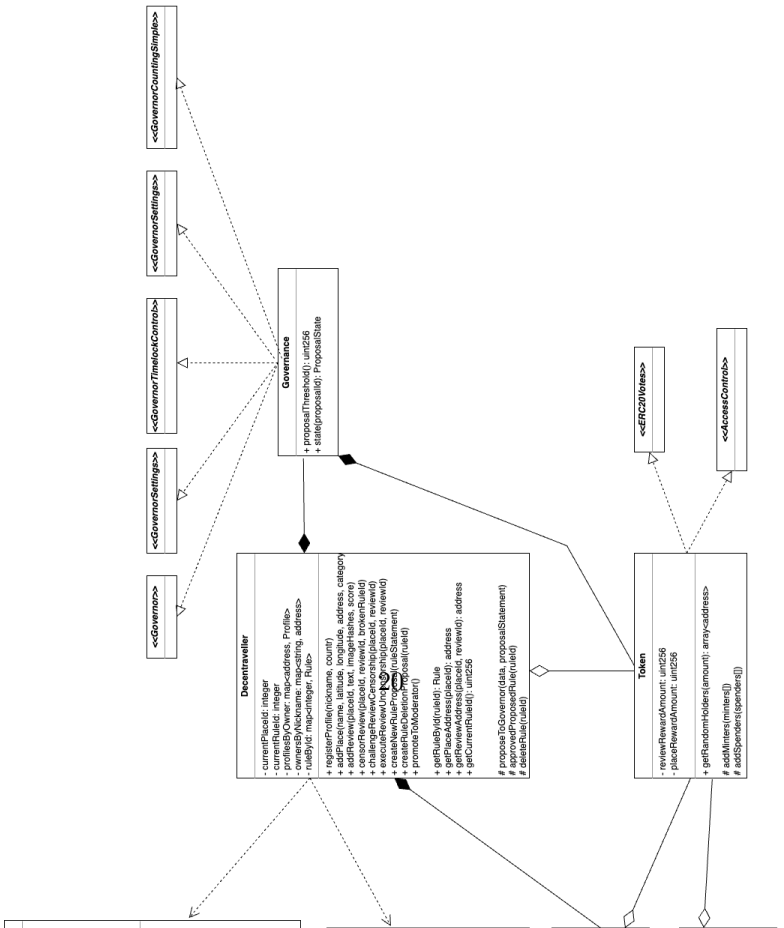


Figura 6: Diagrama de clases de entidades almacenadas en el servidor principal.



El contrato principal que contiene la mayoría de las operaciones que podrán realizar los usuarios es **Decentraveller**. En él, podemos ver una gran cantidad de funciones que tienen una correspondencia uno a uno con los casos de uso descritos previamente. Se encuentra en el centro del diagrama y se relaciona con todo el resto de contratos desplegados.

Cada vez que se quiere crear un nuevo punto turístico o una nueva reseña en un punto turístico existente se despliega un nuevo contrato **Place** o **Review** respectivamente. Para esto, se utiliza el patrón *factory*, donde ambos tipos de contratos tienen sus correspondientes *factories* llamadas **PlaceCloneFactory** y **ReviewCloneFactory**. A su vez, debe notarse que los contratos **Review** y **Place** implementan la interfaz **Initializable** y poseen la función *initialize*. Esta interfaz forma parte del conjunto de interfaces de la librería *OpenZeppelin*, la cual contiene una basta cantidad de contratos, interfaces, librerías e implementaciones de contratos inteligentes con un uso sumamente extendido que sirven como estándar y garantía de seguridad. En este caso particular, la interfaz *Initializable* ayuda a que el patrón *factory* se combine con la utilización de clones, dando lugar a un patrón más complejo conocido como *clone factory*. La utilización de *clones*, donde cada uno de las entidades desplegadas en forma de contrato individual es interpretada como un clon, es un patrón que al implementarlo permite hacer un uso más eficiente del gas disminuyendo el costo de realizar transacciones que invocan métodos pertenecientes al *clone*.

Si miramos hacia la derecha del diagrama, nos encontramos con el contrato que administra la gobernanza de la plataforma, **Governance**. Este contrato hace un extensivo uso de las librerías para crear gobernanzas *on-chain* provistas por *OpenZeppelin*, implementando las interfaces **Governor**, **GovernorSettings**, **GovernorTimelockControl**, **GovernorVotes** y **GovernorCountingSimple**. La combinación de la implementación de todos estos contratos base con sus respectivos parámetros nos brinda el comportamiento que deseamos para las acciones relacionadas a la DAO tales como dar de alta propuestas relacionadas a las reglas, votar en la mismas, contabilizar los votos según cierta lógica, etc. En el caso particular de nuestra gobernanza, la misma se encuentra regida por los propietarios del token de Decentravaller.

El contrato que administra la emisión de los *tokens* de Decentraveller es el llamado **Token**. Este contrato es utilizado por la gobernanza fundamentalmente para obtener información de los balances de los propietarios del *token*. Para esto, **Token** implementa la interfaz provista por *OpenZeppelin* llamada **ERC20Votes**, que resulta ser una extensión de la conocida interfaz **ERC20** a la cual se le añadieron algunas funcionalidades para que el *token* pueda ser utilizado como mecanismo del poder de votación en la gobernanza. A su vez, el contrato **Token** implementa otra interfaz de *OpenZeppelin* conocida como **AccessControl**, que permite la existencia roles y también de métodos que pueden ser invocados únicamente por direcciones que tengan un rol específico asignado. Este es por el ejemplo el caso de **PlaceCloneFactory** y **PlaceReviewFactory**: una vez que despliegan los nuevos clones, invocan al contrato **Token** para asignar nuevos *tokens* al balance de los usuarios que crearon dichos clones en forma de recompensa. Para realizar esta operación, se les asigna un rol par-

ticular que les permite emitir *tokens* en el momento en el que se realiza el *deploy* inicial de los contratos.

6. Vista de desarrollo

En esta sección son presentados una serie de diagramas de paquete que desglosan la organización del código en los diversos componentes que conforman la arquitectura de nuestra plataforma. Estos diagramas son fundamentales para entender la estructura modular del sistema, mostrando claramente cómo se agrupan distintos módulos en paquetes cohesivos y cómo estos paquetes interactúan entre sí. Además, ilustran como algunos de estos paquetes utilizan dependencias externas a fines de llevar a cabo el objetivo para el que fueron diseñados.

6.1 Diagrama de paquetes del servidor

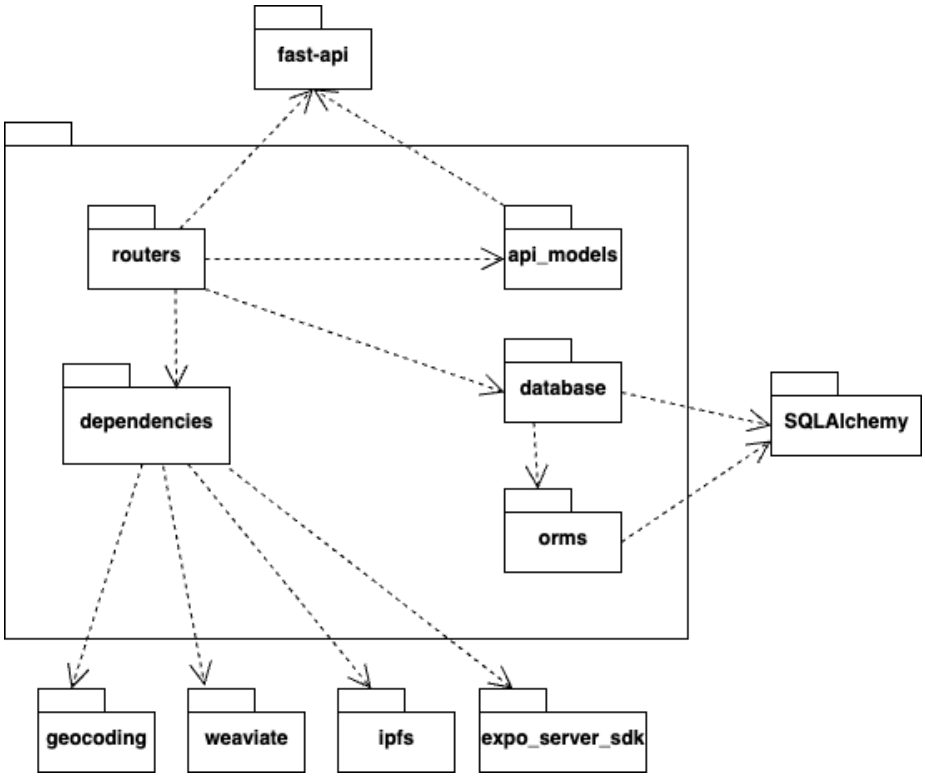


Figura 8: Diagrama de paquetes correspondiente al servidor principal.

El paquete **routers** engloba todos los controladores de la aplicación, constituyendo las clases esenciales que definen los endpoints HTTP, los cuales fa-

cilitan la comunicación entre componentes externos y el servidor principal. Por otro lado, la API está caracterizada por los objetos definidos en el paquete **api_models**. Tanto **routers** como **api_models** hacen uso del *framework* **FastAPI**, especialmente diseñado para el desarrollo ágil y sencillo de servidores web en Python. Por otro lado, una gran cantidad de endpoints HTTP son utilizados para guardar y retornar entidades. Para esto, los controladores se comunican con los objetos que pertenecen al paquete **database**, los cuales contienen la lógica para comunicarse con la base de datos de entidades. Las entidades almacenadas a la base de datos en conjunto con sus relaciones se encuentran mapeadas en objetos presentes en el paquete **orms**, y para promover la comunicación transparente entre el servidor y la base datos, **database** y **orms** basan sus objetos en la utilización de herramientas provistas por la librería **SQLAlchemy**.

Luego encontramos el paquete **dependencies**. Este paquete contiene las clases que tienen alguna lógica para conectarse a algún tipo de componente externo en el cual se quiere realizar cierta acción o consumir alguna información. Tal es así que este paquete utiliza dependencias tales como **geocoding**, para obtener información de geolocalización; **weaviate**, para conectarse a la base de datos que almacena *embeddings* que serán utilizados para realizar recomendaciones; *ipfs*, para comunicarse con el nodo ipfs que aloja contenido multimedia; y **expo_server_sdk**, el cual permite enviar notificiaciones push a los usuarios de la aplicación.

6.2 Diagrama de paquetes del servidor indexador

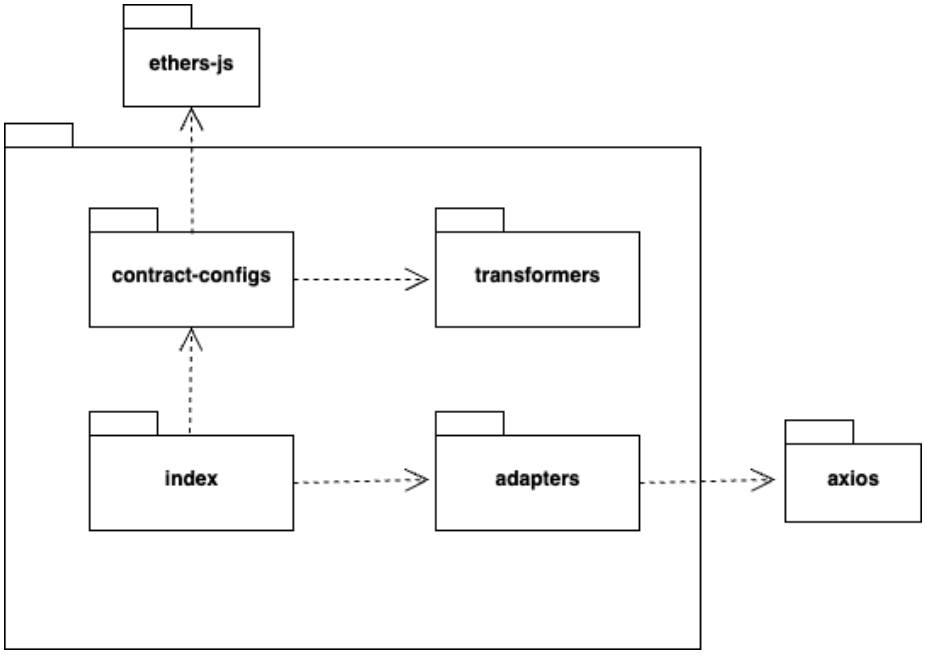


Figura 9: Diagrama de paquetes correspondiente al servidor indexador.

El paquete **index** contiene un único archivo en donde se encuentra la lógica que corre al inicializarse el servidor indexador. Esta lógica consiste en tomar la configuración de las suscripciones a eventos de *smart contracts* especificados en el paquete **contract-configs** y efectuar estas suscripciones. Para armar la configuración, **contract-configs** hace uso de la dependencia externa **ethers-js**, una librería especialmente diseñada para comunicarse con nodos conectados a la blockchain que permite enviar y recibir información a través de estos. Asimismo, **contract-configs** define en la configuración de la suscripción la transformación que debe aplicarse en un evento recibido con el fin de generar una entidad a enviar al servidor principal, y para esto se vale de los objetos presentes en el paquete **transformers**. Una vez recibido el evento y aplicada la transformación, el archivo presente en **index** utlizará un objeto de **adapters** para enviar la entidad resultante del mapeo hacia el servidor principal. El *adapter* realizará un request HTTP que contiene dicha entidad, y para esto utilizará la dependencia externa **axios**, la cual consiste en una librería que provee abstracciones para realizar comunicaciones mediante el protocolo HTTP.

6.3 Diagrama de paquetes de la aplicación móvil

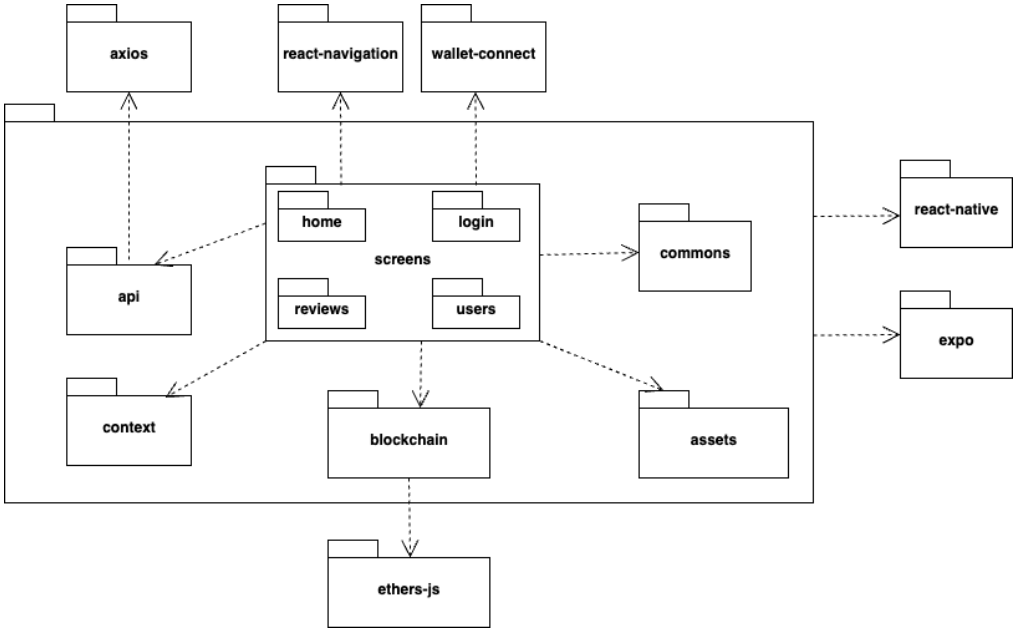


Figura 10: Diagrama de paquetes correspondiente a la aplicación móvil.

El paquete central que contiene a las pantallas de la aplicación es **screens**. Nótese que este se encuentra sub-dividido en cuatro sub-paquetes agrupados según pantallas que están relacionadas entre sí. El sub-paquete **home** contiene las pantallas correspondientes a la navegación principal de la aplicación; el sub-paquete **login** a las pantallas de ingreso y conexión con la *wallet* del usuario (notar que posee como dependencia externa a **wallet-connect**); el sub-paquete **reviews** posee todas las pantallas relacionadas a las reseñas y las correspondientes disputas que pueden surgir en las mismas; y por último en el sub-paquete **users** encontramos todas las pantallas correspondientes al perfil del usuario y al flujo de registración del mismo.

El paquete principal hace uso de un conjunto de paquetes satélite. El paquete **api** contiene las entidades encargadas de comunicarse con el servidor principal así como sus modelos. Dentro del paquete **context** se define cómo se administrará el estado compartido por todos los componentes de la aplicación. En **blockchain**, encontramos todas las abstracciones y configuraciones que permiten a la aplicación enviar transacciones a la *blockchain* y leer datos de la misma. Por su parte, **assets** contiene distintos elementos visuales utilizados para conformar las pantallas de la aplicación. Por último, en **commons** se incluyen una serie de componentes y funciones comunes que son utilizados a lo largo de todas las pantallas y componentes.

Finalmente, debe destacarse que la aplicación está diseñada a partir de las dependencias externas **react-native** y **expo**. Estas dependencias definieron el *framework* en el cual fue estructurada la aplicación, por lo que distintas funcionalidades otorgadas por las mismas aparecen uniformemente desparramadas por todos los paquetes de la aplicación.

7. Modelos de inteligencia artificial y utilización de datos

7.1 Algoritmo de recomendación de contenido

Se construyó un sistema de recomendación por **filtrado colaborativo** que permite encontrar puntos turísticos similares. El sistema consiste de una **base de datos orientada a vectores** donde se guardan los vectores (o **embeddings**) de los lugares, y un script responsable de actualizar esos vectores. Cuanto menor resulta ser la distancia coseno entre dos puntos, más símilares son entre sí.

El script crea a partir de una consulta en la **base de datos de eventos y entidades** una matriz binaria con una fila por cada lugar y una columna por cada usuario que hiciera más de tres reviews. La matriz contiene un 1 en las posiciones donde el usuario de la respectiva columna hiciera una reseña con puntaje mayor o igual a tres al lugar de la respectiva fila y 0 en otro caso. De esta forma, cada fila resulta ser un **embedding** para cada lugar. Como la dimensión de cada **embedding** resulta tan alta como la cantidad de usuarios, el script reduce la matriz aplicando **Singular Value Decomposition** a cien dimensiones. Finalmente, se borran todos los vectores de la base de datos y se sobreescriben con los nuevos.

La ventaja de tener una **base de datos vectorial** es que está optimizada para consultas por distancia, pudiendo consultar eficientemente una cantidad fija de lugares más similares a otro. Esto es aprovechado por el **servidor principal** para dos casos de uso:

1. Lugares similares a otro: La **aplicación movil** muestra en la pantalla dedicada de cada lugar algunas recomendaciones de otro lugares. La implementación para el **servidor principal** resulta muy sencilla ya que solo tiene que consultar los **embeddings** más similares al del lugar inicial.

2. Recomendaciones personalizadas de lugares: La **aplicación movil** muestra una cantidad limitada de lugares que le recomienda al usuario de forma personalizada en la pantalla de **home**. Si el usuario ha escrito reseñas en ciertos lugares, las recomendaciones, además de incluir lugares basados en la ubicación, incluyen los lugares que son similares a los últimos que el usuario reseñó.

7.2 Modelo de *scoring* de calidad y belleza de imágenes

Uno de los problemas más novedosos del área de **computer vision** es el intento de predecir qué tanta calidad tiene y qué tan bella es una imagen. Los factores que influyen (luminación, resolución, definición, encuadre, propósito de la imagen, color, etc.) tienen una dimensión objetiva pero tambien subjetiva.

El sistema solo recibe imágenes en las reseñas. Naturalmente, puede surgir la pregunta de cómo es elegida la imagen de portada que aparece en la pantalla dedicada correspondiente a un punto turístico. Lo deseable es elegir la mejor imagen entre todas las subidas.

Con este objetivo, fue reproducido parcialmente un paper [1] que utiliza **transfer learning** sobre un dataset [2] extraído de un blog de fotografía con puntajes de usuarios del 1 al 10. El objetivo de la red neuronal es predecir si el promedio del puntaje de la imagen es mayor o menor a 5, tratándolo como un problema de clasificación binaria.

Fueron aplicados cambios para que la predicción pueda ser más rápida utilizando una red neuronal de base para el **transfer learning** más pequeña [3] y se eligió una arquitectura propia para el clasificador. Los resultados obtenidos en el dataset de test son muy cercanos a los del paper original con 0.77 de **accuracy**.

8. Limitaciones

La siguiente sección presenta algunas de las limitaciones que fueron encontradas durante el desarrollo de la plataforma.

8.1 Entornos productivos y escenarios de prueba

Una característica distintiva del sistema desarrollado es la necesidad de incluir IN volumen considerable de datos para crear escenarios que demuestren eficazmente su calidad en lo que respecta a experiencia de usuario. Esta exigencia implica generar una cantidad significativa de transacciones en la *blockchain*. Por esta razón, los escenarios de prueba más complejos solo se pudieron llevar a cabo en una *blockchain* local, es decir, una versión emulada de la *blockchain* ejecutándose en un entorno de desarrollo. No fue posible implementarlos en entornos de producción o incluso en *blockchains* de prueba, ya que su configuración podría requerir días o meses, y el contexto de desarrollo de Decentraveller implicaron tanto tiempo como recursos limitados. Por ejemplo, ciertas

[1] Talebi, H., & Milanfar, P. (2018). NIMA: Neural image assessment. IEEE transactions on image processing, 27(8), 3998-4011.

[2] Murray, N., Marchesotti, L., & Perronnin, F. (2012, June). AVA: A large-scale database for aesthetic visual analysis. In 2012 IEEE conference on computer vision and pattern recognition (pp. 2408-2415). IEEE.

[3] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

funcionalidades, como la votación de propuestas en la DAO, que incluyen un componente temporal (por ejemplo, votaciones que duran un día entero), hacen su ejecución en *blockchains* reales inviable dentro del marco de tiempo del desarrollo del proyecto.

La imposibilidad de testear todos los *features* en un entorno de producción conlleva ciertas desventajas. Las *blockchains* locales suelen ser más permisivas y menos propensas a presentar errores durante la ejecución de transacciones o la consulta de datos, en comparación con sus contrapartes en producción. Al no someter la aplicación a un entorno real, es probable que se pierdan oportunidades de identificar y realizar mejoras importantes en el sistema. A su vez, utilizar un entorno local y implicó que se realizarán algunas suposiciones fuertes sobre la disponibilidad y consistencia del sistema que resulta pertinente enumerar.

En primer lugar, no se incluyeron mecanismos de monitoreo y observabilidad de ningún tipo de la salud sobre los componentes incluídos en la infraestructura, ya que se asume que estos se encontrarán corriendo todo el tiempo y no estarán sujetos a escenarios de falla. En segundo lugar, se asume que no existirán fallos en la comunicación entre los componentes del *backend*, es decir que no habrán intermitencias en la red relacionadas a un despliegue de los componentes en zonas geográficas distintas. En tercer lugar, no se tuvo en cuenta el retraso que puede surgir de enviar una transacción a la *blockchain* en momentos en el que el precio del gas resulta inestable (el costo computacional de ejecutar la transacción), ya que el desarrollo fue llevado a cabo en un ambiente controlado con una blockchain de prueba. Todos estos puntos deberían ser tenidos en cuenta en caso de tener la intención de desplegar el sistema en un ambiente productivo real.

8.2 Acoplamiento a *walletconnect*

Una de las únicas tecnologías para que una aplicación móvil pueda comunicarse con la *blockchain* es *walletconnect*. Esta tecnología actúa de intermediaria entre las aplicaciones descentralizadas y las *wallets*, estableciendo un protocolo de comunicación en donde las *wallets* deben implementar la comprensión de ciertos mensajes tipicados, y las aplicaciones deben comunicarse con las *wallets* respetando el formato de estos mensajes. Este protocolo de comunicación implica la presencia de un servidor intermedio mantenido por la empresa que desarrolla *walletconnect*. En el caso de que este servidor presente alguna falla o dejara de correr, la aplicación móvil queda inutilizable para sus usuarios.

Durante el desarrollo del proyecto, *walletconnect* decidió migrar de la versión 1 a la versión 2 de su protocolo para introducir mejoras en el protocolo, deprecando totalmente la versión 1. Esto generó un atraso inesperado en el proyecto, ya que debido al acoplamiento existente con esta tecnología, se debió frenar todo lo que se estaba desarrollando en ese momento para realizar la migración. Dicha migracion no estaba dentro del *scope* original, resultando en un claro ejemplo de como el acoplamiento con una tecnología que está en constante evolución y sobre la que no se tiene control puede potencialmente atrasar el desarrollo de un proyecto. Más adelante, en la sección de *Trabajo Futuro*, serán exploradas

algunas alternativas que eliminan a *walletconnect* como dependencia.

A pesar de que no se tuvieron en cuenta

8.3 Nodo de hardhat e indexado

El entorno de desarrollo utilizado para emular una blockchain fue Hardhat. Hardhat provee un nodo conectado a una blockchain simulada que cumple con todas las funcionalidades de un nodo para una blockchain EVM compatible. Este tipo de nodos, y el de Hardhat no es la excepción, limitan los servicios que se suscriben a eventos indexados por el nodo en su tiempo de conexión con hasta cinco minutos. Esto implica que o bien se debe realizar una reconexión frecuentemente con la posibilidad de perder eventos, o bien modificar un nodo EVM a fines de eliminar esa limitación; y, para la industria hay una tercera opción que consiste en contratar servicios pagos como Infura que resuelven el problema ofreciendo sus propios nodos customizados. En el caso de Decentraveller, fue realizado un *fork* del código de Hardhat para eliminar esa restricción, permitiendo que la conexión entre un agente externo que quiere consumir datos de la blockchain y el nodo no posea límites en el tiempo que puede permanecer viva.

9. Trabajo futuro

En esta sección serán descritas algunas ideas en pos de continuar el trabajo realizado y mejorar ciertos aspectos de la plataforma. Por un lado, algunas de ellas están relacionadas a aspectos de ingeniería del sistema, como por ejemplo la tolerancia a fallos, mientras que otras se encuentran enfocadas a evolucionar ciertos aspectos del sistema orientados a la experiencia de usuario.

9.1 Reconciliaciones

Uno de los aspectos considerados al construir el sistema es que todos los procesos de los componentes se encuentran corriendo absolutamente todo el tiempo y no existe ninguna falla que pueda hacer que estos procesos se detengan. Esto es válido para algunos componentes como por ejemplo la *blockchain*, debido a su naturaleza descentralizada, pero en un entorno productivo real no es válido para otros componentes tales como el servidor indexador o el servidor principal. No tener en cuenta estas consideraciones al intentar desplegar el sistema en un entorno productivo podría llevar a que existan errores de consistencia en los datos, afectando la experiencia del usuario en la aplicación.

Una posible solución para mitigar estas inconsistencias sería la de introducir procesos de reconciliación. Estos procesos correrían en escenarios tales como la inicialización de un servidor, la detección de alguna falla de un proceso corriendo en un servidor, y/o con alguna periodicidad parametrizable, y tendrían como principal objetivo asegurarse de que los datos almacenados en la base de

datos, que representan la información que el usuario ve en la aplicación, sean consistentes con los datos almacenados en la blockchain.

Un posible ejemplo de la aplicación de esta idea podría ser el siguiente. Supóngase que durante un período de tiempo breve, el servidor indexador sufrió una caída y por lo tanto algunos eventos fueron perdidos sin ser indexados, lo que deriva en que no sean almacenados generando la inconsistencia. Al iniciar el servidor indexador, este podría utilizar el último bloque que indexó antes de sufrir la caída, y recuperar todos los eventos lanzados hasta el bloque actual. De esta forma, estos eventos podrían ser almacenados en la base de datos, y el usuario volvería a ver en su aplicación la información en forma consistente respecto a lo guardado en la blockchain.

9.2 *Wallet* propia con *account abstraction*

No quedan dudas que uno de los aspectos más molestos a la hora de utilizar la aplicación es que cada vez que se desea generar una transacción para realizar alguna modificación de estados o crear información en la blockchain, se debe pasar por una wallet externa al sistema como por ejemplo *Metamask*. Este pasaje utilizando la *wallet* como intermediaria para enviar transacciones a la blockchain peca de ser sumamente lento y repetitivo.

Una forma de mitigar este problema consistiría en que cada usuario tenga una *wallet* propia que no sea como las tradicionales. En el último tiempo y particularmente en el último año, el ecosistema de Ethereum y desarrollo en blockchains EVM (*Ethereum Virtual Machine* compatibles comenzó a incorporar fuertemente a sus narrativas el concepto de *account abstraction*. Este concepto hace referencia a la utilización de *smart contract wallets* que implementen el estándar definido en la EIP-4337, el cual permite crear billeteras representadas en contratos inteligentes con lógicas arbritrarias en la validación y el firmado de transacciones. Dentro de está lógica arbtiraria podría incluirse la posibilidad de que el dueño de una *smart contract wallet* genere *session keys*, las cuales emulen el comportamiento típico de las *sessions keys* que existe hoy en día en las aplicaciones web modernas, pero firmando transacciones en vez de autenticando solicitudes HTTP.

En el caso particular de Decentraveller, cada usuario podría tener una *smart contract wallet* desplegada al registrarse que le permita generar *session keys* por un determinado período de tiempo. Al comenzar a utilizar la aplicación, el usuario firmaría una primera transacción de generación de *session keys*, y luego estas claves podrían ser guardadas en el almacenamiento local del teléfono móvil y utilizarse como firma para las transacciones subsiguientes. La *smart contract wallet* con la que se generaron las claves contendría la lógica para determinar la validez y expiración de las mismas, y una vez que estas claves expiren se podrían regenerar nuevamente. De esta forma, el usuario no debería estar abriendo un *wallet* externa y firmando transacciones todo el tiempo (solamente cuando haya que generar claves), mejorando drásticamente su experiencia en la aplicación. Obviamente, el desarrollo de una *smart contract wallet* es un tópico complejo que requiere de un profundo entendimiento de desarrollo blockchain,

de la EIP-4337 y de cuestiones específicas de criptografía, pero sería un proyecto innovador de llevar a cabo y podría estandarizarse y utilizarse no solamente para Decentraveller sino también para otras aplicaciones descentralizadas.

9.3 Extensión de funcionalidades de la DAO

La DAO de Decentraveller cuenta con un conjunto de funcionalidades que en la práctica de una DAO real, en la que sus participantes se encuentren realmente comprometidos con el desarrollo de la misma, pueden resultar insuficientes. Una posible extensión de estas funcionalidades podría incluir la posibilidad de que los participantes de la DAO no solamente se involucren en la creación de propuestas para agregar o eliminar reglas, sino también en el agregado de nueva lógica en los contratos de Decentraveller.

Si bien es cierto que una de las caraterísticas fundamentales de la *blockchain* es la inmutabilidad, pero haciendo uso del conocido patrón *proxy*, *OpenZeppelin* ha desarrollado un conjunto de interfaces y librerías de contratos que, al implementarlos en los contratos de Decentraveller, permitirían que mediante una transacción especial estos puedan someterse a un *upgrade*, cambiando así su implementación. La idea, entonces, consiste en que los usuarios de la DAO puedan crear propuestas que ejecuten posibles *upgrades* hacia implementaciones que ellos mismos realicen. Estos *upgrades* serían sometidos a la votación de la comunidad, y de salir esta victoriosa, el *upgrade* podría ser aplicado cambiando la lógica de los contratos. Algunas de las cosas que esto permitiría incluyen, por ejemplo, cambiar los mecanismos de recompensas al generar contenido en la aplicación, cambiar parámetros relacionados a las votaciones de la DAO, agregar lógica para la elección de moderadores, y cualquier otra lógica que los usuarios desearan agregar o modificar. La inclusión de este *feature* dinamizaría la aplicación y sus cambios, abriendo un mundo enorme de posibilidades para el destino de la misma, destino que estaría pura y exclusivamente ligado a la creatividad y las decisiones de los usuarios.

9.4 Cambios en el *key management*

Un aspecto clave de Decentraveller es que una plataforma descentralizada. Esto, en parte, significa que los usuarios son dueños de sus propios datos y a su vez son responsables de los cambios de estados que generen en la aplicación (de hecho, son ellos mismos los que tienen que pagar con *gas* cada vez que quieren realizar una transacción). Si bien existen muchas ventajas de que sea una plataforma descentralizada, la utilización de una *wallet* puede llevar a que existan fricciones para aquellos usuarios que estén habituados a plataformas tradicionales centralizadas donde el concepto de *wallet* ni siquiera se encuentra presente. Una forma posible de mitigar estas fricciones es abstraer el manejo de claves y *wallets* de los usuarios y acercarles una solución que les resulte más familiar.

La propuesta consiste en que las transacciones contra la blockchain pasen a ser creadas en el servidor principal en vez de en la aplicación móvil. El reg-

istro pasaría a ser de tipo tradicional con usuario y contraseña y el servidor crearía una billetera asociada al usuario cuando este completa la registración. La clave privada de esta billetera sería almacenada encriptada mediante algún algoritmo de clave simétrica, cuya clave de encriptación y desencriptación sería la contraseña elegida por el usuario. Migrar a una solución de este estilo implicaría que el usuario deba colocar su contraseña cada vez que desee realizar una transacción, ya que el servidor debería desencriptar su clave privada, a diferencia de crear esta transacción a través de su *wallet*. Por supuesto este approach tiene sendas desventajas: en primer lugar, de alguna forma los usuarios (o mismo la DAO de Decentraveller si tuviera un tesoro) deberían fondear estas *wallets* con *gas* una vez creadas, ya que de otra forma no podrían crear transacciones; y en segundo lugar, al estar la responsabilidad del almacenamiento de claves del lado del servidor, una caída de este implicaría que los usuarios dejarían de poder crear transacciones. Esto, claramente, implica sacrificar ciertos aspectos de la descentralización en pos de eliminar una fricción natural en la experiencia de usuario que surge en las plataformas descentralizadas.

9.5 Perfiles y contenido en *Lens Protocol*

Lens Protocol es una infraestructura de red social descentralizada y abierta diseñada para correr sobre la blockchain, que permite a los desarrolladores construir aplicaciones sociales interoperables y donde los usuarios mantienen la propiedad y el control de su contenido y datos. Con su enfoque en la descentralización, Lens Protocol busca que las redes sociales construídas sobre el protocolon cambien la dinámica de las redes sociales actuales, dando a los usuarios una mayor autonomía y poder sobre sus interacciones y contenidos en línea.

Lens Protocol utiliza tokens no fungibles (NFTs) para representar de manera única y descentralizada tanto perfiles de usuario como publicaciones realizadas en una red social. Cada perfil y cada publicación son creados como un NFT individual, lo que garantiza la propiedad y el control exclusivo por parte de su identidad digital del contenido que generado a través de esta. Esta implementación permite que los usuarios posean realmente sus perfiles y publicaciones, a diferencia de las redes sociales tradicionales centralizadas, donde son las plataformas las que controlan los datos pudiendo explotarlos en su propio beneficio. Como se encuentra basado en NFTs, el protocolo facilita la interacción, transferencia y portabilidad de los perfiles y publicaciones a través de diferentes aplicaciones y plataformas, promoviendo un ecosistema de redes sociales interoperable.

La idea para aplicar en Decentraveller podría consistir en lo siguiente. Cuando un usuario complete su registración, se le crearía un perfil en formato de NFT dentro del ecosistema de Lens (también podría agregarse lógica para importar perfiles). Luego, cada vez que el usuario desee agregar contenido en Decentraveller, dando de alta un punto turístico o escribiendo una reseña acerca de uno existente, el contenido sería generado en forma de un NFT de Lens. Esto permitiría otorgarles a los usuarios el *ownership* sobre el contenido creado, a partir de lo cual podrían derivarse un montón de *features* interesantes, como la comercialización del contenido, la construcción de sistemas de reputación mon-

etizables, la creación de sistemas de recompensación basados en la reputación, etc. Por último, un punto interesante a destacar es que, como consecuencia directa de descentralizar el contenido, podrían surgir más *frontends* de Decentraveller, o incluso una competencia directa de la plataforma que utilice el contenido originalmente agregado a Decentraveller. Esto teminaría promoviendo la competencia entre productos, en donde muy posiblemente los usuarios optarían por elegir a aquel cuya experiencia de usuario les resulte más atractiva.

9.6 Mejoras en el algoritmo de elección de moderadores y selección de jurados

La forma en la que se eligen los moderadores de Decentraveller es bastante elemental. En primer lugar, los usuarios que primero se registren en la plataforma serán seleccionados como moderadores, y luego también podrán existir usuarios que quemando una suma fija del token de gobernanza podrán adquirir el rol. No fueron incorporados mecanismos para remover el rol, por ejemplo en caso de censuras incorrectas o inactividad en la plataforma. A su vez, en el caso en el que muchos usuarios se registraran en la plataforma, la cantidad de moderadores debería incoporar alguna relación de proporción con los usuarios totales, así como también se podrían introducir componentes de aleatoriedad en la elección de los mismos.

Por otro lado, la selección de jurados en donde se obtiene una cantidad fija de *holders* del token de gobernanza introduce componentes pseudo-aleatorios para intentar ser lo más justo posible. Estos componentes son heredados del bloque en el que se ejecuta la transacción que termina seleccionado los jurados, y son el *timestamp*, la dificultad, la dirección del validador del bloque y el límite de gas a pagar. Dada la naturaleza de la *blockchain* en la que no hay componentes que agreguen entropía, condición necesaria para obtener un número aleatorio, una posible mejora podría consistir en introducir la utilización de un oráculo, cuya función es conectar a una *blockchain* con componentes externos, a fines de utilizarlo como vía para obtener un número aleatorio real. El oráculo a utilizar podría ser *Chainlink*, apoyándose en la funcionalidad otorgada por el contrato *AggregatorV3Interface*.

10. Conclusiones

El desarrollo de la plataforma constituyó un desafío técnico por demás interesante y desafiante, que implicó la construcción de una arquitectura sofisticada en donde se relacionan muchos componentes con una considerable cantidad de tecnologías utilizadas. Si bien Decentraveller es un sistema sin dudas complejo, se ha podido seguir durante todo el desarrollo la premisa de mantener la experiencia de usuario lo más simple posible, con las dificultades que el desarrollo de las aplicaciones descentralizadas conlleva.

Una de las singularidades técnicas del trabajo radica en la combinación de tecnologías descentralizadas, en donde por ejemplo encontramos a *smart*

contracts diseñados para correr en cualquier *blockchain* **EVM** compatible o a la integración con IPFS, con las tecnologías centralizadas, donde encontramos *backends* típicos y bases de datos relacionales. Esta combinación permite a la aplicación tener *features* que serían imposibles de conseguir sin la presencia de las entidades centralizadas, sin resignar las virtudes de las aplicaciones descentralizadas inherentes a que se encuentran desplegadas en una *blockchain*.

En términos prácticos, la presencia del sistema de recomendación de contenido basado en filtrado colaborativo y del sistema de elección de imágenes para los puntos turísticos utilizando machine learning demuestra que es posible incorporar la utilización de algoritmos que utilizan datos aplicándolos, valga la redundancia, a un set de datos generados en una tecnología descentralizada como es la *blockchain*. De esta manera, la incorporación de nuevos componentes donde están alojados los procesos que corren estos algoritmos permitió construir una aplicación descentralizada con una experiencia sumamente enriquecida.

La integración de una DAO en el núcleo de la plataforma demuestra un serio compromiso con la participación comunitaria y la toma de decisiones democráticas. Los usuarios de Decentraveller, en lugar de ser usuarios comunes, son considerados actores principales brindándoles la capacidad de proponer y votar sobre reglas fundamentales destinadas a guiar el desarrollo de la comunidad alrededor de la aplicación. Este enfoque, respaldado por la gobernanza que brindan las organizaciones autónomas descentralizadas, no solo involucra la aplicación de valores fundamentales del desarrollo *blockchain*, tales como la transparencia, la seguridad y la descentralización (que fue la motivación principal para la elección del desarrollo de esta idea) sino que también implica la adaptación a un nuevo paradigma de desarrollo cuyo principal objetivo es el empoderamiento de los usuarios.

Finalmente, cabe destacar una de las filosofías principales en el desarrollo de la plataforma. Todos los componentes intentaron ser diseñados y codificados siguiendo buenas prácticas de ingeniería con el objetivo de que la aplicación sea extensible y puedan ser agregados más *features* a la misma. Haber seguido esta forma de desarrollo hace que la aplicación pueda ser utilizada como base para la construcción de aplicaciones más complejas apalancándose en la arquitectura construída, la utilización de tecnologías modernas y las buenas prácticas de programación utilizadas.