WP4-A5. Producción de la herramienta interactiva RockChain.

"Financiado por la Unión Europea. Las opiniones y puntos de vista expresados solo comprometen a su(s) autor(es) y no reflejan necesariamente los de la Unión Europea o los de la Agencia Ejecutiva Europea de Educación y Cultura (EACEA). Ni la Unión Europea ni la EACEA pueden ser considerados responsables de ellos."

Índice

1. INTRODUCCIÓN

Este documento presenta los resultados de la actividad WP4-A5 Producción de la herramienta interactiva RockChain. Mientras que las tareas anteriores en WP4 se centraban en la capa de datos (WP4-A1), perfeccionamiento de la herramienta de aprendizaje electrónico (WP4-A2) y las especificaciones funcionales (WP4-A3), WP4-A5 describe cómo se ha producido, empaquetado y preparado la versión interactiva final de RockChain para su distribución y uso en contextos reales de entrenamiento.

El objetivo del WP4-A5 es doble. Por un lado, ofrece una visión consolidada de la arquitectura técnica de la herramienta RockChain, mostrando cómo el cliente móvil, los servicios de backend y la orquestación en tiempo real trabajan juntos para soportar sesiones educativas multijugador. Por otro lado, documenta el flujo de trabajo de producción y despliegue que conduce a versiones listas para usar de Android o iOS, un despliegue backend estable y procedimientos básicos para alojamiento, acceso y mantenimiento.

El resultado es una descripción de "cómo se construyó" que puede ser utilizada por el personal técnico que necesita mantener o ampliar la herramienta, y por socios del proyecto que requieren una visión clara de cómo la herramienta interactiva RockChain se ha convertido en un recurso listo para producción y formación profesional y educación de adultos.

2. ARQUITECTURA DEL SISTEMA DE LA HERRAMIENTA INTERACTIVA ROCKCHAIN

Desde un punto de vista técnico, la herramienta interactiva RockChain está construida como un sistema de tres capas:

- Un cliente móvil desarrollado junto con Expo y React Native.
- Un backend de Firebase que proporciona autenticación, almacenamiento persistente y funciones serverless.
- Un servidor autoritativo de Node.js + Socket.IO que coordina las rondas en tiempo real y asegura que todos los jugadores vean un estado de juego consistente.

Estos componentes se complementan con una capa de internacionalización, utilidades de configuración del entorno y mecanismos básicos de observabilidad.

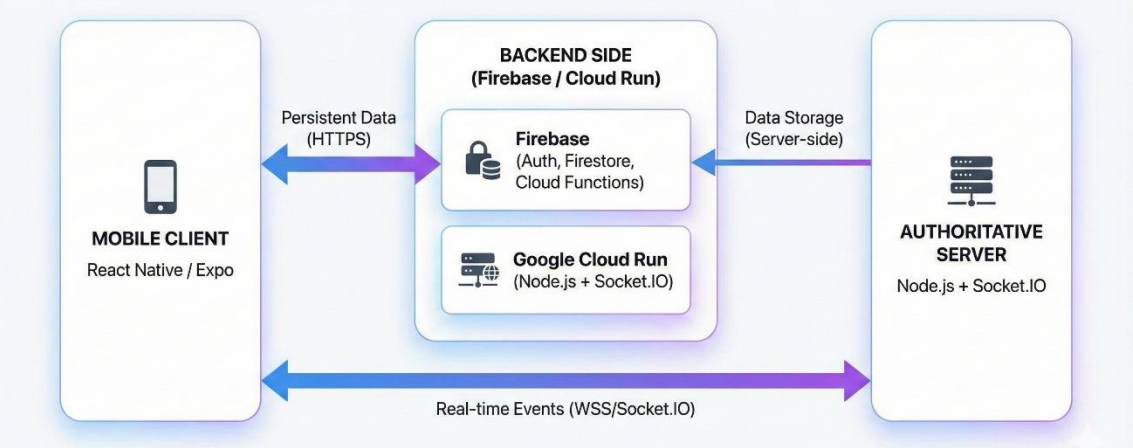

Figura 1: Arquitectura de alto nivel.

2.1. Cliente móvil: frameworks, punto de entrada y navegación

El cliente móvil RockChain está construido como una aplicación React que funciona en React Native, empaquetada y orquestada por Expo. En la práctica, esto significa:

- Todas las pantallas y elementos de la interfaz de usuario se escriben como componentes de funciones React usando JSX.
- React 19 proporciona el modelo de componentes, los hooks (*useState, useEffect, useContext*, hooks personalizados) y la API Context que la app utiliza para compartir el estado del juego entre pantallas.

- React Native 0.79 renderiza estos componentes como vistas nativas en Android e iOS, por lo que la app se siente y se comporta como una app nativa mientras comparte una única base de código JavaScript.
- Expo actúa como envoltorio y cadena de herramientas: proporciona el servidor de desarrollo, el empaquetado, el sistema de compilación EAS y acceso a servicios nativos (red, almacenamiento, información del dispositivo) sin mantener proyectos separados de Xcode/Android Studio.

En tiempo de ejecución, todo comienza en *App.js*, que es el punto de entrada del cliente. Este archivo conecta los servicios principales de corte cruzado que toda pantalla necesita:

- Inyecta la internacionalización envolviendo todo el árbol de componentes en *I18nextProvider i18n={i18n}.* Esto permite que cualquier pantalla use claves de traducción en lugar de cadenas codificadas y elija automáticamente el idioma del usuario.
- Envuelve la interfaz en un *NavigationContainer* (de React Navigation), que define un único árbol de navegación para toda la aplicación (pilas, pestañas y flujos anidados).
- Incluye la navegación dentro de dos proveedores globales, *GameProviderHybridRobust* y *SimpleMiningProvider,* y muestra un *NetworkStatusBanner* en el nivel superior:
    - *GameProviderHybridRobust* es una capa React Context + hook que expone el estado relacionado con el juego (partida actual y ronda, jugadores, inventarios, temporizadores, clasificaciones, guardias de navegación).
    - *SimpleMiningProvider* ofrece un mini-contexto dedicado para problemas de minería, facilitando la implementación y resolución de desafíos de prueba de trabajo sin acoplarlos estrechamente al resto de la interfaz.
    - *NetworkStatusBanner* escucha los cambios de conectividad y muestra advertencias cuando el dispositivo está desconectado o tiene mala conectividad.

Como estos proveedores están por encima del contenedor de navegación, cualquier pantalla, por muy profunda que esté en la pila, puede acceder al estado del juego, al estado de minería y al estado de la red mediante ganchos, en lugar de reimplementar esa lógica localmente.

Además de esta base técnica, la aplicación sigue un flujo de navegación sencillo y lineal que refleja el ciclo de vida de un juego. React Navigation se utiliza con una pila principal que guía a los usuarios a través de estas etapas:

- *Inicio de sesión*: donde un jugador se registra o inicia sesión usando la autenticación de Firebase.

- *Juego:* una pantalla de envoltorio que establece el contexto para el *gameId* y *userID* seleccionados.
- *WaitingRoom:* donde los jugadores se reúnen y ven quién se ha unido al juego antes de que comiencen las rondas.
- *GameTabs:* el entorno principal dentro del juego que se usa durante las rondas.
- *Pantallas de final de ronda y resultados*: donde se muestran resúmenes, clasificaciones y resultados finales.

Cuando el usuario introduce *GameTabs*, el diseño cambia a una estructura basada en pestañas. La barra de pestañas inferior forma parte del árbol de navegación (un navegador estándar de pestañas de navegación de React), pero puede ocultarse visualmente para que la experiencia se sienta más como un juego cohesivo que como una app típica con pestañas visibles. Internamente, los oyentes vinculados a los eventos de Socket.IO y *a GameContextHybridRobust* deciden cuándo cambiar entre módulos — por ejemplo, abriendo automáticamente Minería cuando surge un nuevo problema, o navegando al resumen de final de ronda cuando el temporizador llega a cero — para que los estudiantes no tengan que gestionar transiciones complejas por sí mismos.

Durante una ronda activa, hay seis módulos principales disponibles dentro *de GameTabs*:

- *Mercado*: el módulo central donde los jugadores compran y venden bloques de piedra, residuos y productos reciclados. Se suscribe a eventos de actualización de productos y precios y genera listas dinámicas basadas en el estado actual del mercado.
- *Minería*: una visión enfocada que aparece cuando se activa un desafío de prueba de trabajo. Muestra problemas de aritmética cronometrada, recopila respuestas localmente y las envía al servidor autorizado para su validación.
- *Reciclar*: el módulo donde partes del inventario del jugador pueden convertirse en RockCoins o nuevos recursos, activando la lógica de economía circular en acciones y transacciones concretas.
- *Estadísticas*: un panel de control implementado con componentes de React Native más un kit de gráficos react-native, que muestra indicadores de rendimiento y clasificaciones por ronda y a lo largo de todo el juego.
- *Perfil*: un área personal que muestra el avatar del jugador, su progreso actual y la configuración de idioma, y puede exponer otras preferencias básicas en futuras extensiones.
- *Acciones*: la vista resume cantidades y precios de productos clave, actuando como una rápida "instantánea del mercado". Los jugadores pueden abrirlo para entender brevemente qué recursos son abundantes o escasos antes de decidir si comprar, vender o reciclar.

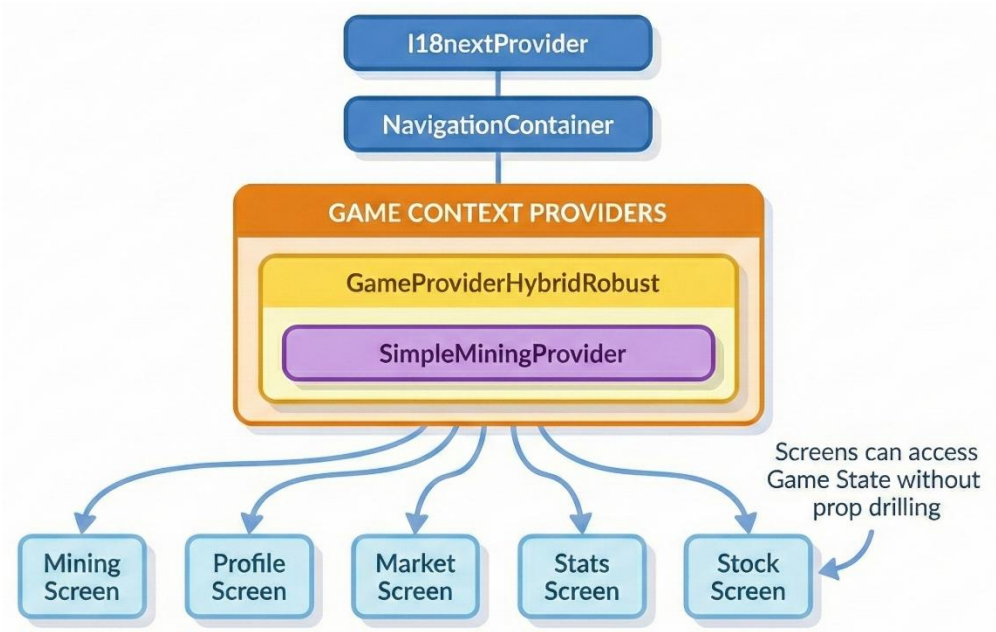

Figura 2: Árbol de componentes y proveedores.

Todas estas pantallas son componentes estándar de React que reciben sus datos principalmente de *GameProviderHybridRobust* y del flujo de eventos Socket.IO, en lugar de de estados locales ad hoc. Esta combinación, componentes y ganchos de React, renderizado React Native, build/runtime de Expo, React Navigation para control de flujo y proveedores de contexto para estado compartido, convierte al cliente móvil en una interfaz cohesiva y declarativa que es relativamente fácil de razonar y de ampliar, pero que sigue sintiéndose como un juego multijugador nativo en los dispositivos de los aprendices.

2.2. Gestión del estado en el lado del cliente

En el cliente, RockChain depende en gran medida de la API de contexto de React y ganchos personalizados para gestionar el estado compartido. En lugar de permitir que cada pantalla guarde su propia copia de reproductores, temporizadores o datos de minería, la app centraliza la mayor parte de la lógica en unos pocos proveedores bien definidos. Las pantallas entonces se convierten mayormente en "vistas": se suscriben a ese estado y lo renderizan, pero no deciden las reglas por sí mismas. Esto es lo que mantiene la app predecible incluso cuando muchos eventos llegan en tiempo real.

En el núcleo de esta capa está ***GameContextHybridRobust***. Técnicamente, es un Contexto React respaldado por un componente proveedor y un conjunto de ganchos

que exponen el estado actual del juego a cualquier pantalla. Conceptualmente, es la principal fuente de verdad sobre el cliente. Se mantiene unido:

- La lista de jugadores en el juego actual y sus inventarios (productos, residuos, RockCoins).
- Los temporizadores y los identificadores de la ronda actual, para que todas las pantallas sepan cuánto tiempo queda y qué ronda está activa.
- El estado y las clasificaciones de minería, para que los resultados y las tablas de clasificación sean consistentes en Mercado, Estadísticas y otras vistas.
- Una vista sincronizada del reloj del servidor, construida a partir de Socket.IO mensajes, que permite al cliente obtener el tiempo restante sin alejarse demasiado del servidor autoritario.
- Varios protectores de "navegación segura", que evitan las condiciones de la carrera cuando la navegación cambia al mismo tiempo que se procesan nuevos eventos (por ejemplo, evitando que una pantalla lea datos obsoletos justo al final de la ronda).

Más allá de almacenar solo datos, *GameContextHybridRobust* también integra un conjunto de mecanismos de idempotencia y resincronización alrededor de la conexión WebSocket. Cada evento relevante lleva identificadores como *roundId* y version, y el contexto lleva un registro de lo que ya se ha aplicado. Si la conexión se corta y luego se recupera, el contexto puede volver a solicitar o conciliar el estado autorizado del servidor en lugar de confiar ciegamente en lo que se había renderizado por última vez. Esto es lo que permite a un jugador bloquear su teléfono por un momento o perder brevemente el Wi-Fi sin romper completamente el juego.

La lógica de minería se encapsula además en un contexto dedicado, a menudo denominado *SimpleMiningContext* y expuesto a través de *SimpleMiningProvider*. La idea aquí es mantener una cola ligera de problemas de minería y su manejo de la interfaz separada del resto del estado del juego. Cuando el servidor autorizado emite nuevos desafíos de minería, se les empuja a esta cola; Cuando el usuario contesta o expira el temporizador, se procesan y eliminan. Como la minería se gestiona en este contexto aislado, el resto de la interfaz puede seguir respondiendo incluso si se crean y resuelven varios problemas en rápida sucesión, y la interfaz relacionada con la minería (como modales o cuentas atrás) no tiene que conectarse manualmente a cada pantalla.

En torno a estos dos contextos principales, el cliente utiliza un conjunto de ganchos auxiliares para agrupar reglas de negocio específicas:

- *useLearningProgress* escucha eventos del juego y registra logros educativos (por ejemplo, bloques minados, residuos reducidos) en Firestore y puede activar retroalimentación visual cuando se alcanzan ciertos hitos.

- *useNetworkStatus* envuelve la API de NetInfo y alimenta el *NetworkStatusBanner*, de modo que cualquier pérdida temporal de conectividad es visible de inmediato para el jugador.
- *useTutorial* controla los flujos de incorporación para usuarios primerizos, decidiendo cuándo mostrar explicaciones o pistas y cuándo dar un paso atrás para dejar que el jugador interactúe libremente.

En conjunto, estos contextos y ganchos implementan un principio de diseño claro: las pantallas deben ser lo más simples posible, mientras que el comportamiento de corte cruzado (temporizadores, jugadores, minería, progreso de aprendizaje, conectividad, tutoriales) reside en módulos compartidos y comprobables. Esto facilita la explicación de la aplicación, la ampliación (las nuevas pantallas pueden reutilizar los mismos ganchos) y es más robusta en condiciones de tiempo real, porque hay un único lugar donde el estado del juego se deriva de eventos de backend.

2.3. Backend de Firebase: autenticación y datos persistentes

En el backend, RockChain se apoya en Firebase como una plataforma gestionada que integra tres servicios clave: Autenticación, Cloud Firestore y Cloud Functions. En conjunto, estos servicios permiten la gestión segura de la identidad de cada usuario, el almacenamiento persistente de datos y progreso del aprendizaje del juego, y la ejecución controlada de operaciones sensibles directamente en el servidor. Aunque todos los detalles técnicos sobre el modelo de datos, los flujos y las reglas de seguridad están documentados en el entregable WP4-A1, aquí tienes una visión general de cómo se utiliza esta infraestructura dentro de la herramienta interactiva.

El corazón del sistema es Cloud Firestore, que actúa como la "memoria a largo plazo" de RockChain. En lugar de tablas como en una base de datos tradicional, Firestore organiza la información en colecciones y documentos. En el entorno de producción, las principales colecciones son:

- *usuarios*: contiene un documento por jugador, con su perfil básico y rendimiento en cada juego (rockCoins, residuos, productos), agrupados por identificador de juego.
- *Juegos*: un documento por sesión de juego, donde se almacenan el código de acceso, el anfitrión, la lista de jugadores, el estado actual, la ronda y ciertos indicadores clave usados tanto por el cliente como por el servidor en tiempo real.
- *Mercado*: catálogos de productos y precios dinámicos por juego, separados del documento principal para no saturarlo con actualizaciones frecuentes.
- *Blockchain*: Una historia simplificada de los bloques minados en cada partida, registrando quién minó qué y cuándo, permitiendo reflejar mecánicas inspiradas en blockchain en los datos.

Para operaciones críticas, los clientes no escriben directamente en estas colecciones. En su lugar, invocan funciones de servidor (Funciones en la Nube) que se ejecutan con privilegios especiales, aplican validaciones y reglas de negocio, y solo entonces modifican los datos en Firestore. Estas funciones se detallan en el informe WP4-A1, pero dentro de la aplicación tienen tres funciones principales:

- Gestión del ciclo de vida del juego: funciones como *createGame*, *joinGame* y *deleteGame* crear y eliminar documentos del juego, verificar la unicidad del código y evitar sesiones obsoletas del mismo host.
- Lógica de mercado y minería: Funciones como *asignarProductosToJuego*, *actualizarProductosPrecios*, *generarProblemaMiningProblema* y *SubmitMiningSolución* controlan cómo evolucionan los productos, precios y desafíos de minería, asegurando reglas coherentes en todos los juegos.
- Recompensas y perfiles: Al final de cada ronda o partida, funciones como *asignarRondas* y *actualizarPerfilde de Usuario* consolidan los resultados y actualizan los perfiles de usuario y los documentos de instantáneas, manteniendo la coherencia entre lo que se veía en pantalla y lo que se guardaba de forma permanente.

Además, la integración con Firebase incluye el sistema de autenticación y una pequeña capa de almacenamiento local en el dispositivo (*AsyncStorage*). La autenticación garantiza que cada solicitud al backend esté vinculada a un ID de usuario único, utilizado de forma consistente en Firestore y en el servidor WebSocket. El almacenamiento local permite a la aplicación recordar las sesiones activas y guardar datos esenciales entre lanzamientos, evitando interrupciones en caso de cortes breves de conexión o cierres accidentales de la app.

Para detalles sobre la implementación completa, incluyendo las normas de seguridad, los flujos de datos y los aspectos relacionados con el RGPD, los socios pueden consultar el informe técnico WP4-A1.

2.4. Servidor autorizado y servicios en tiempo real

Aunque Firebase gestiona el almacenamiento y la lógica del servidor de forma segura y persistente, RockChain también necesita una capa que responda casi instantáneamente a las acciones del jugador, gestione temporizadores y decida en tiempo real quién gana los desafíos de minería. Para satisfacer esta necesidad, el proyecto utiliza un servidor en tiempo real desarrollado con Node.js y Socket.IO, empaquetado en una imagen Docker y desplegado en Google Cloud Run. WP4-A1 detalla esta arquitectura, pero aquí tienes un resumen de cómo soporta la herramienta interactiva.

En términos sencillos, este servidor actúa como árbitro de cada partida en curso. Para cada partida, se guarda en la memoria:

- Qué jugadores están conectados y en qué sala.
- Sus inventarios actuales y sus posiciones en el ranking.
- El desafío de la minería activa (si es que lo hay).
- El estado oficial de la ronda, incluyendo la hora exacta en que debe terminar.

Basándose en este estado, el servidor genera identificadores como *roundId*, *version* y *roundEndsAtMs*, que se adjuntan a los eventos enviados. Esto permite que tanto el cliente como el backend sepan a qué ronda se refieren en todo momento y evita confusiones con mensajes duplicados o tardíos.

El comportamiento del juego está estructurado como una simple máquina de estados con cuatro fases principales:

- JUGANDO: la ronda está activa; los jugadores pueden comprar, reciclar o resolver problemas de minería mientras el temporizador corre.
- ROUND_CALCULATING: las acciones se pausan y el servidor calcula los resultados.
- ROUND_END: los valores finales ya están disponibles; los jugadores pueden ver sus recompensas y posiciones.
- WAITING_FOR_NEXT_ROUND: el juego se pausa hasta que el presentador decide empezar una nueva ronda o terminar el juego.

A medida que el juego avanza en estas fases, el servidor emite diferentes eventos como *start_round, ROUND_CALCULATING, round_ended, game_completed, inventory_data, player_rankings_data* o *economy:state*. La aplicación móvil escucha estos eventos y actualiza las pantallas, temporizadores y resúmenes en tiempo real, asegurando que todos los jugadores tengan una visión coherente de lo que está ocurriendo.

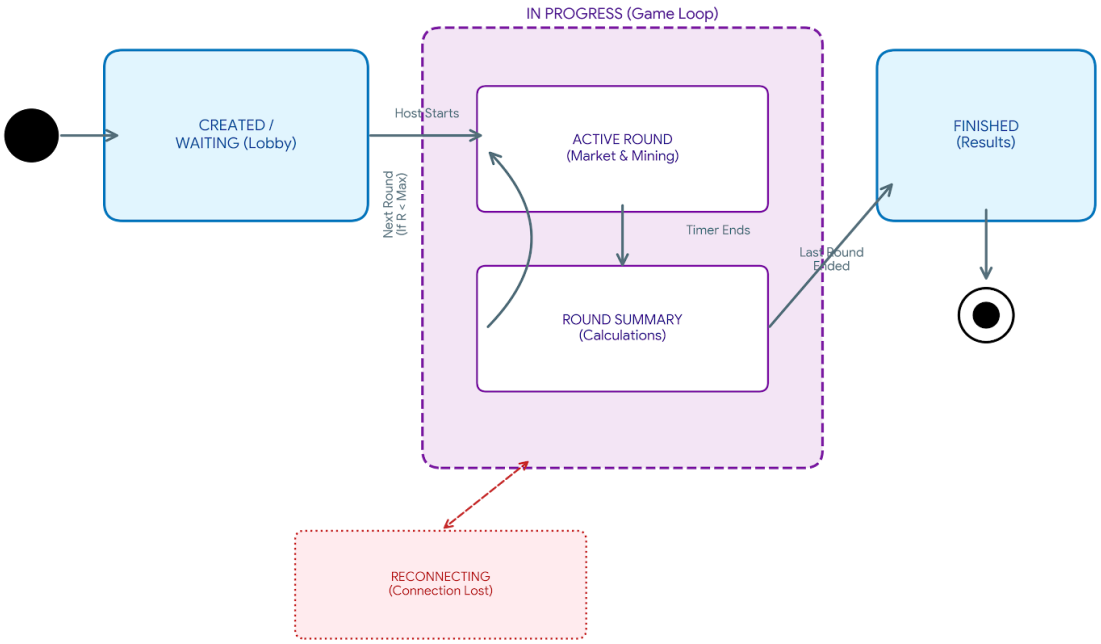

Figura 3: Máquina de estados de la cadena de roca.

Dado que muchas acciones pueden llegar casi al mismo tiempo (especialmente al final de una ronda), el servidor utiliza mecanismos de bloqueo y actualizaciones idempotentes para evitar conflictos. Las operaciones críticas, como asignar recompensas o actualizar inventarios, se ejecutan de forma ordenada, sin solapamientos, y se marcan con identificadores de ronda para ignorar repeticiones. Cuando el servidor puede acceder a Firebase, también guarda los resultados de cada ronda (inventarios, recompensas, estado económico), asegurando que la instantánea en memoria y los datos almacenados permanentemente estén alineados.

En el lado del cliente, toda esta complejidad está encapsulada en un adaptador Socket.IO. Este adaptador selecciona automáticamente el servidor correcto (local o Cloud Run), abre y mantiene la conexión WebSocket, reenvía la autenticación y la información del juego tras una reconexión, y traduce los eventos de backend al formato esperado por la aplicación desarrollada por React. Gracias a esta capa intermedia, el servidor puede evolucionar con el tiempo sin afectar la experiencia del usuario ni el funcionamiento pedagógico de la herramienta

2.5. Cuestiones transversales: internacionalización, configuración y observabilidad

Más allá de los componentes principales del cliente y backend, RockChain incluye varios mecanismos transversales que hacen que la herramienta sea utilizable en diferentes países, sea más fácil de desplegar en diferentes entornos y de depurar durante los

pilotos. Tres de ellas son especialmente relevantes: internacionalización, configuración del entorno y observabilidad.

2.5.1. Internacionalización y accesibilidad

Desde el principio, RockChain fue diseñado como una herramienta multinacional. Esto significaba que el apoyo lingüístico no podía ser una idea secundaria. En su lugar, el proyecto integró una capa de internacionalización basada en i18next.

La configuración principal está en *src/i18n/index.js*, que:

- Carga paquetes de idiomas para inglés (EN), español (ES), alemán (DE), croata (HR) y rumano (RO).
- Detecta el idioma del dispositivo o utiliza la elección explícita del jugador para decidir qué paquete activar.
- Expone una instancia i18n que el resto de la app usa a través de I18nextProvider.

Todos los textos orientados al usuario se almacenan como claves de traducción en *src/i18n/locales/*.json*. Para cada lenguaje soportado, existe un archivo JSON correspondiente donde esas claves se asignan a cadenas reales. Los componentes solicitan textos por clave en lugar de codificar directamente el inglés o cualquier otro idioma.

Para que la experiencia sea persistente, el idioma seleccionado se guarda en AsyncStorage. Esto significa que:

- La primera vez que la app se ejecuta, puede usar por defecto el idioma del dispositivo.
- Si el usuario cambia de idioma a través de la interfaz, esa preferencia se guarda localmente.
- En lanzamientos posteriores, la app restaura el mismo idioma sin necesidad de volver a preguntar.

Añadir un nuevo lenguaje es sencillo y no requiere tocar el código básico:

1. Crea un nuevo archivo JSON bajo *src/i18n/locales/* con las mismas claves que los idiomas existentes.
2. Registra el nuevo código del idioma en *los SupportedLngs* en *src/i18n/index.js*.
3. Proporciona traducciones para todas las claves relevantes.

Este diseño apoya uno de los objetivos principales de RockChain: la herramienta puede transferirse a otros países y contextos trabajando en archivos de traducción en lugar de modificar la lógica de la aplicación.

2.5.2. Configuración del entorno y detección de red

Dado que RockChain debe ejecutarse en varios contextos de ejecución (desarrollo local, staging y producción), el proyecto evita codificar directamente URLs o supuestos del entorno. En su lugar, utiliza una capa de configuración pequeña pero explícita.

Aquí hay dos módulos centrales:

src/config/environment.js

- Calcula valores como SOCKET_URL, NODE_ENV, tiempos de espera y flags de depuración.
- Lee desde la configuración de tiempo de compilación y las variables del entorno para decidir, por ejemplo, si la aplicación debe comunicarse con un servidor Socket.IO local, una instancia de staging o el servicio de producción Cloud Run.

SRC/Utils/networkUtils.js

- Detecta si la app está ejecutándose en un simulador/emulador o en un dispositivo físico.
- Basándose en esa información, elige la dirección del servidor adecuada:
    - Localhost o una IP LAN específica durante el desarrollo.
    - La URL de Cloud Run en compilaciones de staging o producción.

Además, las variables de entorno de Expo (como EXPO_PUBLIC_WEBSOCKET_URL) permiten al proyecto sobreescribir endpoints en perfiles de compilación específicos. Por ejemplo, una compilación EAS de producción puede cablear el endpoint WebSocket a la URL oficial de Cloud Run, mientras que una versión previa puede apuntar a una instancia de prueba.

El efecto neto es que la misma base de código puede reconstruirse para diferentes entornos simplemente eligiendo el perfil y configuración adecuados, sin editar archivos fuente. Esto apoya directamente la reproducibilidad y facilita que los socios clonen o actualicen el despliegue.

2.5.3. Observabilidad y resiliencia

Finalmente, la arquitectura incluye un conjunto mínimo de herramientas para entender lo que ocurre en tiempo real y sobrevivir a problemas temporales de red. Esto es especialmente importante en el aula, donde la calidad del Wi-Fi puede variar.

En el **lado del cliente**:

- *GameContextHybridRobust* registra información sobre el desplazamiento del reloj, eventos duplicados e intentos de resincronización en la consola. Durante el desarrollo y las pruebas piloto, estos registros ayudan a identificar situaciones

en las que el cliente y el servidor autorizado divergen temporalmente y con qué frecuencia se requieren resincronizaciones.
- La combinación de la reconexión automática del socket y el Contexto React garantiza que, cuando se restablece la conexión, el cliente pueda solicitar de nuevo el estado autoritativo actual en lugar de depender de datos obsoletos.

En el **lado del servidor**:

- Una *utilidad logMetric* emite eventos JSON estructurados para momentos clave como conexión, *round_start*, *round_end*, resincronización y error. Cuando el servidor se ejecuta en Cloud Run, estos eventos pueden ser capturados por Google Cloud Logging o herramientas similares, proporcionando una cronología de lo que ocurrió en cada sesión de juego.
- El servidor aplica el manejo de eventos idempotente usando identificadores como *roundId*, version y, cuando corresponde, *operationId*. Esto evita recompensas duplicadas o un estado inconsistente cuando los mensajes se vuelven a intentar o llegan tarde.
- Los bloqueos por ronda (*roundLocks*, *gameStates*) aseguran que las secciones críticas —como los cálculos de final de ronda— no se ejecuten simultáneamente para la misma partida y ronda.

En conjunto, estos mecanismos significan que la arquitectura no solo es funcional para pilotos, sino también rastreable y robusta. Los desarrolladores y socios con conocimientos técnicos pueden inspeccionar los registros para entender cómo evolucionaron las sesiones, mientras que los jugadores experimentan un juego que sigue funcionando incluso cuando surgen problemas de conectividad de corta duración.

3. FLUJOS CLAVE EN TIEMPO DE EJECUCIÓN EN LA HERRAMIENTA INTERACTIVA ROCKCHAIN

Aunque la sección anterior explica cómo se construye la arquitectura de RockChain, aquí nos centramos en lo que una persona experimenta al usar la herramienta: cómo inicia sesión, cómo se une o crea un juego, cómo progresan las rondas y qué tipo de respuestas ofrece el sistema. En resumen, analizamos el proceso paso a paso desde el punto de vista del usuario.

3.1. Autenticación e incorporación

Todo comienza en la pantalla de inicio de sesión (*Pantalla de Inicio de* sesión), donde los usuarios pueden hacer dos cosas:

- Crea una cuenta nueva introduciendo la información mínima requerida (como una dirección de correo electrónico, contraseña y un nombre visible en el juego).
- Inicia sesión con una cuenta existente, reutilizando sus credenciales guardadas.

Esta pantalla está conectada directamente al sistema de autenticación Firebase. Cuando alguien introduce sus datos y pulsa "Enter", la app hace lo siguiente:

1. Envía las credenciales a Firebase para su verificación.
2. Si todo está correcto, Firebase devuelve un ID de usuario autenticado y un token de acceso.
3. Si hay un error (por ejemplo, contraseña incorrecta o correo electrónico ya registrado), la app muestra un mensaje claro y pide corregir los datos.

¿Qué ocurre después de iniciar sesión con éxito?

Tres cosas ocurren automáticamente y casi simultáneamente:

El perfil se crea (o recupera) en la base de datos.

- Si es una cuenta nueva, la app o una función del servidor crea un documento en la base de datos con información básica: nombre visible, correo electrónico, fecha de creación, idioma por defecto, etc.
- Si ya existía, el perfil se reutiliza, junto con la historia de juegos anteriores, si es que existe.

Se obtiene un token seguro para las acciones del juego.

- Este token permite que la aplicación se comunique con el servidor y acceda a funciones protegidas, como crear o unirse a juegos.
- También se utiliza para conectarse al servidor en tiempo real, de modo que el sistema sepa quién está detrás de cada acción.

Accede a la zona principal de juego.

- La app deja de mostrar la pantalla de inicio de sesión y va directamente al área del juego, donde la persona puede:
- Empieza una partida nueva.
- Únete a una partida existente introduciendo un código.
- Consulta información básica sobre RockChain.
- O ver sus logros anteriores.

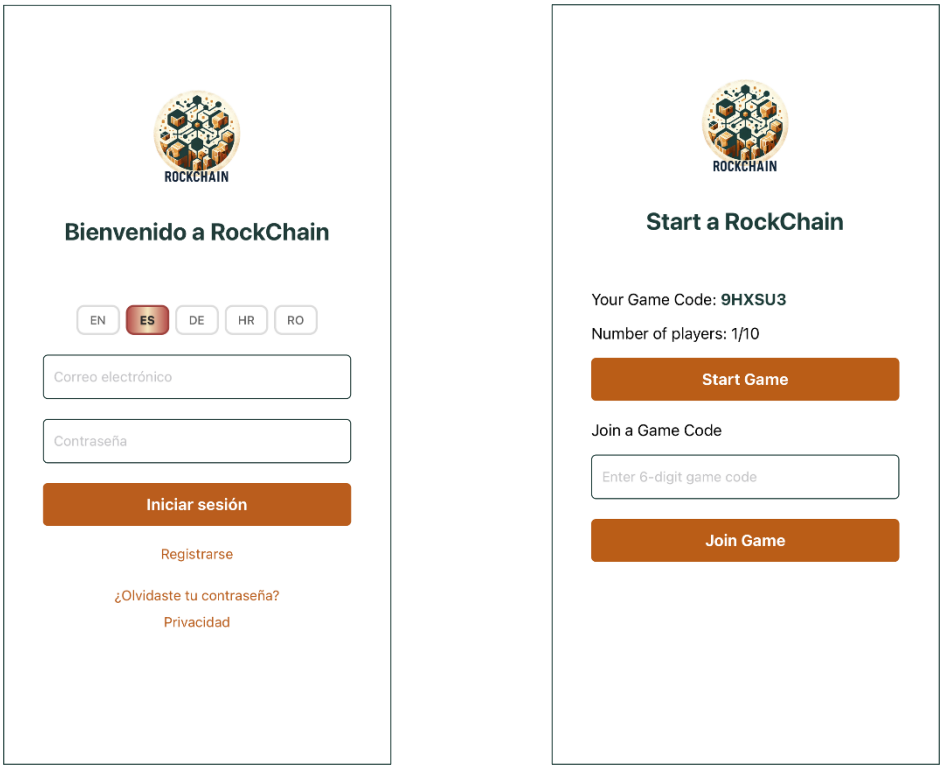

Figura 4: Pantallas de INICIO de sesión y JUEGO.

RockChain no diferencia entre "cuentas normales" y "cuentas de host". Cualquiera que esté conectado puede crear un juego y convertirse en anfitrión si comparte su código y otros jugadores se unen a su sesión. Esto hace que el proceso sea igual para todos: cada persona accede a su espacio de juego y luego los grupos deciden de forma natural cuál usar como juego compartido.

3.2. Creación de juegos y participación de jugadores

El ciclo de vida de un juego en RockChain está diseñado para ser lo más sencillo y fluido posible. Una vez que una persona inicia sesión, la app la trata como a cualquier otra persona: se le asigna automáticamente su propio juego, que aloja él, y desde ahí puede

decidir si lo usa como sesión para su grupo o si se une a la de otra persona usando un código.

No hay necesidad de decidir entre "crear" o "alojar": si compartes tu código y comienzas el juego, tú eres el anfitrión. Así de sencillo.

3.2.1. Juego anfitrión automático en *GameScreen*

Justo después de iniciar sesión con éxito, la app lleva a la persona a la pantalla principal del juego. En ese momento, la aplicación crea automáticamente un juego personal en segundo plano: no es necesario pulsar ningún botón de "crear".

En el backend, este proceso se encarga de:

- Comprobar si esa persona ya tenía partidas activas anteriores y, si es así, cerrarlas para evitar duplicados.
- Asegurando que el catálogo global de productos y los datos básicos de mercado estén disponibles.
- Crear un nuevo documento de juego con:
    - Un código único para compartir (como *ABC123*)
    - El ID de usuario como anfitrión
    - Estado inicial: "*en espera*"
    - El contador de rondas puesto a cero
    - Y cualquier configuración predeterminada que pueda ser necesaria

Cuando la creación está completa, la app tiene todo lo necesario para empezar. En pantalla, la persona verá algo como.

- "*Tu código de juego: ABC123*" arriba.
- "*Jugadores conectados: 1*".
- Un botón de "*Iniciar partida*".
- Un campo para introducir un código si quiere unirse a otra sesión.

Si decides usar tu partida como una sesión grupal, simplemente:

- Comparte tu código con tus compañeros (en voz alta, en pantalla, por chat, etc.).
- Espera a que todos se conecten (el contador de jugadores lo demuestra).
- Pulsa "Iniciar partida."
- No se requiere ninguna acción adicional: ese juego personal se convierte en el juego oficial de grupo.

3.2.2. Unirse al juego de otra persona

También puede ocurrir lo contrario: cuando llegas a esta pantalla, puede que ya tengas tu propio juego creado, pero quieres unirte a otro juego que un amigo ya haya empezado.

Para eso está la zona de "Unirse a una partida". Solo tienes que hacer eso:

- Introduce el código de sesión compartido por el anfitrión.
- Pulsa el botón "*Unirse al juego*".

La aplicación será:

- Busca ese juego usando el código.
- Comprueba que siga abierto a nuevos jugadores.
- Añadir al usuario a la lista de participantes.
- Crea o actualiza sus datos personales del juego (*monedas, inventario, residuos, etc*.).
- Cambia el contexto de su juego: a partir de ese momento, pasan a formar parte de la sesión del presentador.

A partir de ese momento, la persona ve el código del anfitrión y el número correcto de jugadores conectados en la pantalla. Cuando el grupo termine, el anfitrión puede pulsar "*Iniciar juego*", lo que llevará a todos a la sala de espera y comenzará el juego.

3.3. Sala de espera y sincronización

Una vez que una persona decide usar su propio juego como anfitrión o se une al de otra persona, todos los participantes son llevados desde la pantalla principal del juego a la sala de espera. Esta pantalla tiene una función muy específica: reunir al grupo en un punto de inicio estable y asegurarse de que, cuando comience la cuenta atrás, todos los dispositivos estén perfectamente sincronizados.

El lobby no gestiona su propio estado por separado. En cambio, escucha dos fuentes clave de información:

- ***GameContextHybridRobust***, que recopila:
  - Actualizaciones del documento del juego (*games*/{*gameId*}), como la llegada de nuevos jugadores o cambios de estado
  - La lista actual de jugadores y el estado general del juego (en espera, en el juego, terminado, etc.)
- **Eventos de servidor en tiempo real** (*Socket.IO*), que indican:
  - El inicio de la cuenta atrás,
  - Cambios de fase del juego (por ejemplo, pasar de "*esperar*" a "*jugar*")

Con esta información, el vestíbulo muestra solo lo esencial:

- La lista de jugadores conectados a ese juego,
- El estado actual del juego (mensajes como "*Esperando jugadores*", "*Listos para empezar*", "*Empezando en 3...2...1...*"),

- Indicadores opcionales de que los jugadores están listos, si esa función ha sido activada

¿Qué ocurre cuando el presentador pulsa "*Iniciar partida*"?

Cuando el presentador considera que el grupo está listo y pulsa "*Iniciar partida*":

1. Tu dispositivo llama a la *función startGame* en el backend, enviando el ID del juego.
2. El servidor comprueba que:
    - El juego está en un estado válido (por ejemplo, no está ya en progreso y hay suficientes jugadores)
    - Coordina con el servidor en tiempo real para prepararse para el inicio del primer turno:
        - Se genera *un identificador único de ronda (*roundId)
        - Se calcula la hora exacta en la que terminará la ronda (*roundEndsAtMs*)
        - El juego se marca como listo para pasar a la fase tras la cuenta atrás

El servidor entonces emite una serie de eventos a través de *Socket.IO*:

- Un evento de inicio de cuenta atrás ("*la ronda comenzará en unos segundos*")
- Eventos opcionales que marcan el progreso de la cuenta atrás ("*3... 2... 1...*")
- Un evento que indica que la ronda ha comenzado oficialmente

Cada sala de espera recibe estos eventos, y *GameContextHybridRobust* utiliza la marca de tiempo (*roundEndsAtMs*) para calcular el tiempo restante en cada dispositivo. Aunque haya pequeñas diferencias en la conexión Wi-Fi, todos los jugadores salen de la sala de espera y entran en la ronda con el mismo límite de tiempo

La sala de espera actúa como una **cámara de sincronización**. Garantiza que:

- Todos están conectados y son visibles en el juego.
- El presentador puede decidir el momento exacto para empezar.
- Todo el grupo comienza la ronda al mismo tiempo, gracias a una combinación de funciones en la nube y comunicación en tiempo real.

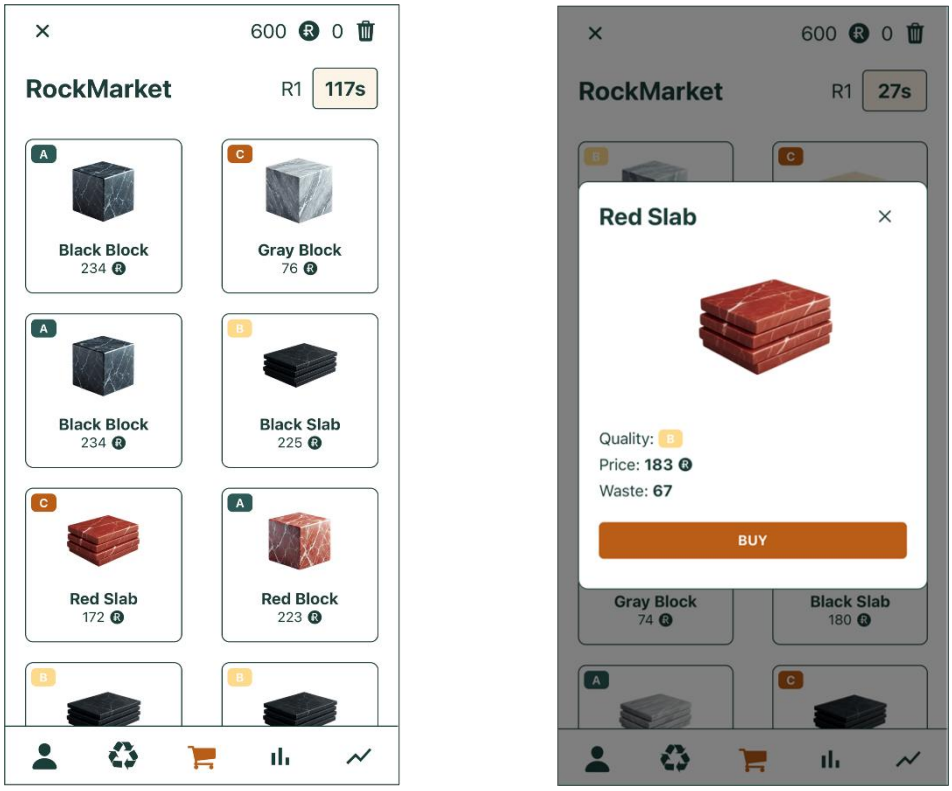

Figura 5: Pantalla MARKET.

3.4. Jugabilidad durante la ronda: navegar entre pantallas

Cuando comienza una ronda, la app mueve automáticamente a todos los jugadores desde el lobby a la interfaz principal del juego, llamada ***GameTabs***. A partir de ese momento, cada persona puede moverse libremente entre varias pantallas clave durante toda la ronda.

El tiempo de juego se controla mediante un valor central llamado *roundEndsAtMs*, que define exactamente cuándo debe terminar la ronda. Este valor lo proporciona el servidor en tiempo real, y el contexto del juego (*GameContextHybridRobust*) lo convierte en una cuenta atrás local para que todos los dispositivos tengan el mismo límite de tiempo, independientemente de pequeños retrasos en la conexión.

Durante una ronda típica de 120 segundos, los jugadores pueden interactuar con los siguientes módulos:

- *MarketScreen*: permite a los jugadores comprar y vender productos (*bloques, losas, materiales reciclados*). La lista de productos se actualiza automáticamente si el servidor cambia precios o suministro.

- *RecycleScreen*: aquí puedes transformar parte de tu inventario en RockCoins o nuevos materiales. Cada acción se valida en el backend, que comprueba si se siguen las reglas antes de aplicarla.
- *MiningScreen*: ofrece desafíos de "prueba de trabajo". La aplicación muestra problemas matemáticos con un límite de tiempo, el jugador envía su respuesta y el servidor decide quién acertó y quién fue el más rápido.
- *StatsScreen*: muestra indicadores de rendimiento y rankings con gráficos actualizados según el progreso.
- *PerfilPantalla*: ofrece un resumen del perfil del usuario, recursos actuales y RockCoins. También te permite cambiar el idioma. La información proviene directamente de Firestore.

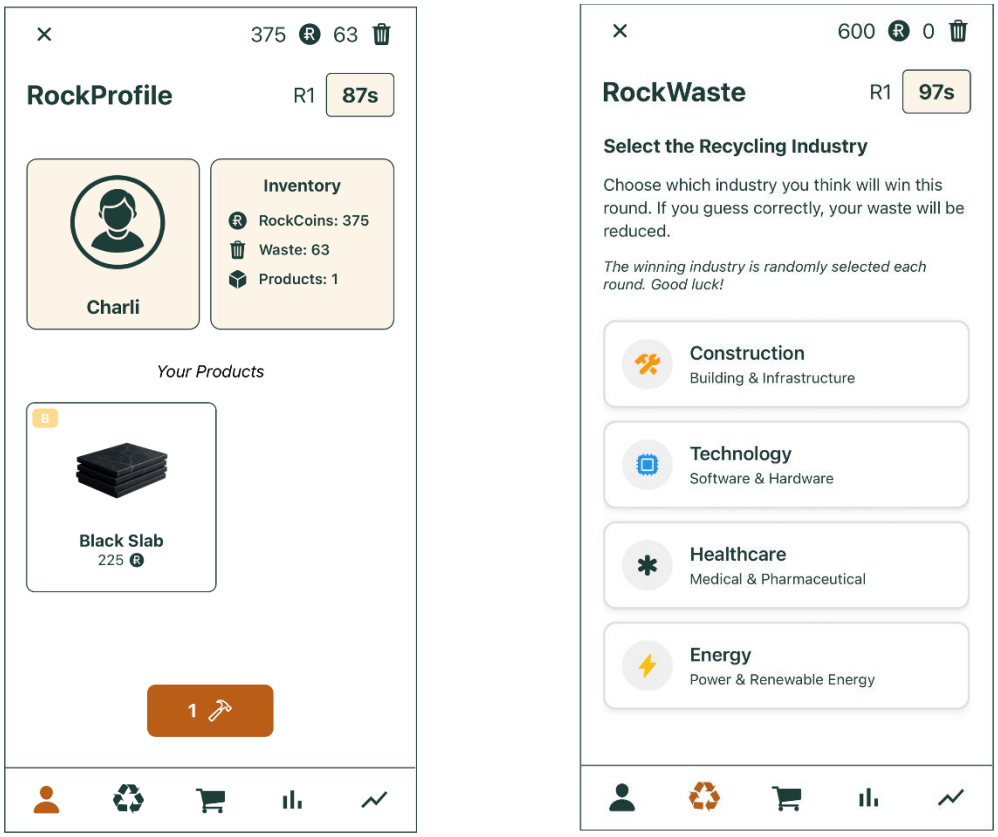

Figura 6: Pantallas PERFIL y RECICLAJE

- *StockScreen*: ofrece una visión general del mercado: qué productos están disponibles, cómo varían los precios y qué recursos son escasos o abundantes. Es útil para planificar antes de comprar o reciclar.

Un principio básico sigue siendo constante en todas estas pantallas: la app nunca toma decisiones finales por sí sola. Cada acción realizada por el jugador se interpreta como una intención enviada al servidor. Por ejemplo:

- *"Quiero comprar el producto X al precio actual"*
- *"Quiero reciclar estos residuos en esta industria"*
- *"Esta es mi respuesta al reto de la minería"*

El servidor en tiempo real recibe cada acción, comprueba si es válida según el estado actual del juego (inventario, precios, temporizador, etc.) y:

- Lo aplica si todo está en orden, actualizando los datos en memoria (y en Firestore si es necesario).
- Lo rechaza si no cumple con las normas o llega demasiado tarde.

El servidor envía entonces las actualizaciones a todos los jugadores: nuevos inventarios, clasificaciones, cambios de precio o resultados de minería.

Gracias a esta estructura, cada ronda es:

- Interactivo: los jugadores pueden moverse entre diferentes pantallas y tomar decisiones libremente.
- Justo y consistente: las mismas reglas se aplican a todos, y el estado del juego se mantiene sincronizado para todo el grupo.
- Controlado por el tiempo: el valor *roundEndsAtMs* garantiza que todos los jugadores terminen la ronda al mismo tiempo, independientemente de pequeñas diferencias en sus dispositivos o redes.

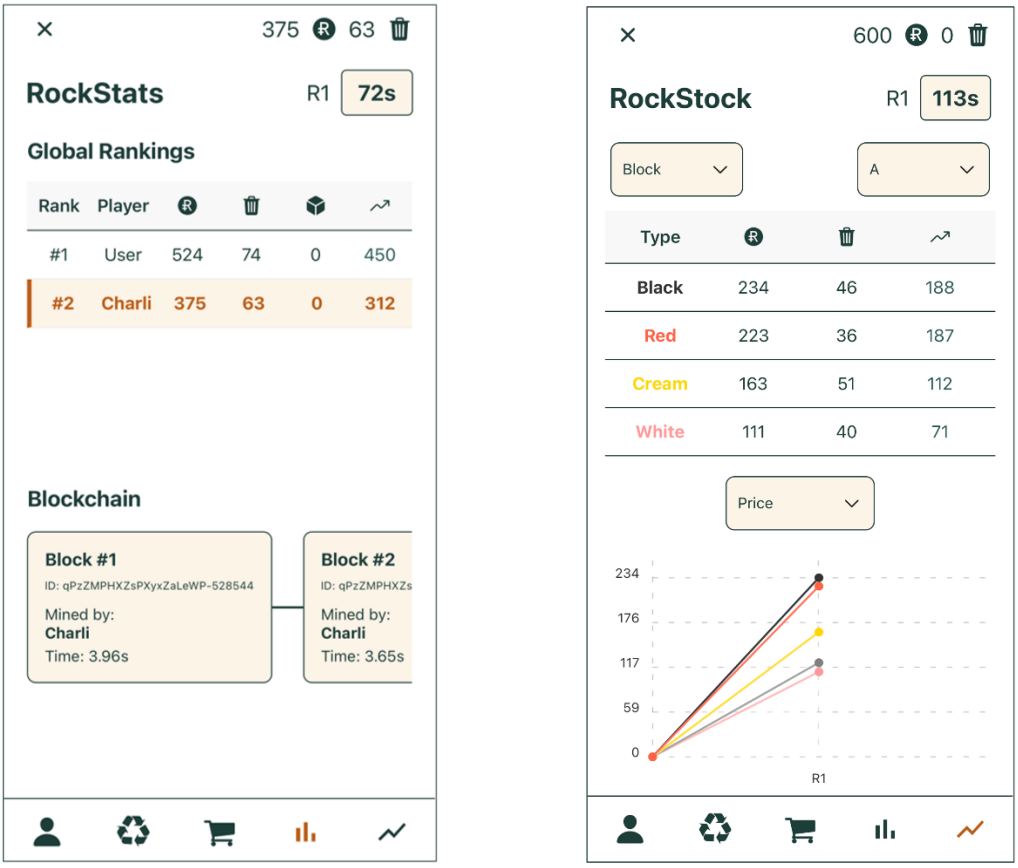

Figura 7: PANTALLAS ESTÁNDAR y STOCK

3.5. Cálculos de final de ronda y resultados finales

Cuando expira el tiempo de la ronda (*roundEndsAtMs*), el sistema deja de aceptar nuevas acciones de los jugadores. A partir de ese momento, el control pasa completamente al backend, que es responsable de congelar el estado del juego, calcular los resultados y mostrar la información de forma clara y consistente.

En el servidor en tiempo real (Socket.IO), el proceso de cierre de la ronda sigue estos pasos:

1. **Acciones de congelación:**
   El servidor cambia el estado del juego a *ROUND_CALCULATING* y deja de aceptar nuevos movimientos. Cualquier mensaje que llegue tarde se ignora o se marca como inválido.
2. **Calcula las recompensas de forma segura:**
   Utilizando bloqueos y comprobaciones a nivel de ronda para evitar repeticiones, el servidor:
   - Procesa cada acción solo una vez
   - Evita duplicar recompensas o contar el mismo evento dos veces

- Garantiza que las acciones simultáneas no generen errores ni datos inconsistentes

3. **Elegir la industria ganadora:**
   La lógica configurada se aplica para determinar qué industria ganó la ronda, una información clave para la retroalimentación que recibirán los jugadores.
4. **Guardando los resultados en Firestore**
   Las actualizaciones del servidor:
   - El documento del juego en *games*/{*gameId*} (estado, número de ronda, puntuaciones)
   - Las estadísticas individuales en *users*/{*userId*}.*juegos*[*gameId*]

Si está disponible, Firebase Admin gestiona estas operaciones de forma segura y eficiente.

5. **Notifique dispositivos**
   Una vez realizados los cálculos y los datos guardados, el servidor envía una serie de eventos como:
   - *ROUND_CALCULATING*
   - *ROUND_ENDED*
   - *REWARDS_ASSIGNED*

   Estos mensajes permiten a los clientes cerrar la fase activa del juego, mostrar que la ronda ha terminado y luego mostrar los resultados detallados.

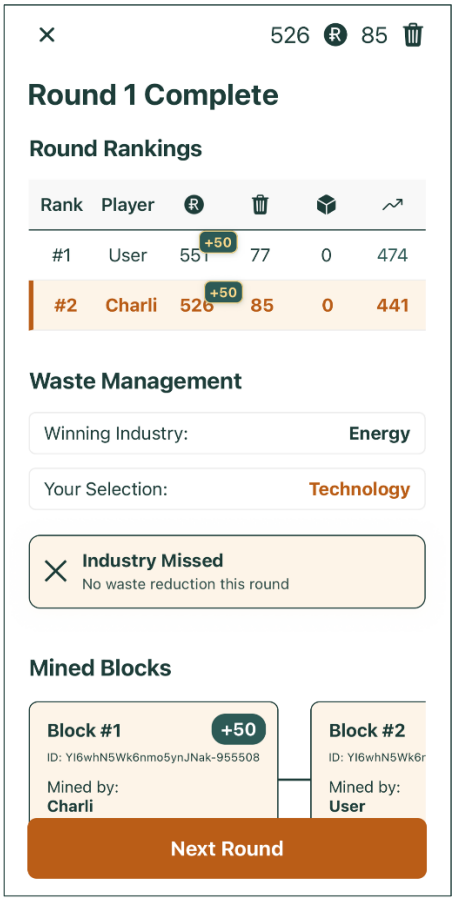

Figura 8: Fin de la pantalla redonda.

En la aplicación, el contexto del juego (*GameContextHybridRobust*) escucha estos eventos y actualiza la interfaz. Cambia de la vista activa del juego a pantallas de resumen, como:

- *EndOfRoundScreen* muestra la industria ganadora, quién resolvió el reto de minería (si es que existía), cómo cambiaron el inventario de cada jugador y las RockCoins.
- *GameResultsScreen* (o una pantalla final similar) muestra la clasificación general de los jugadores, estadísticas acumuladas de las rondas jugadas y otros indicadores útiles para discusión o reflexión.

Esto cierra la ronda para todo el grupo. Dado que todos los datos importantes se almacenan en Firestore, los jugadores pueden:

- volver a la pantalla principal y empezar una nueva partida como anfitriones con el mismo grupo u otro, o
- Sal del juego y continúa con otras actividades del curso.

Las partidas completadas y sus datos siguen disponibles para su uso posterior en WP5 (evaluaciones, análisis o revisiones en futuras sesiones), independientemente de lo que hagan los jugadores a continuación.

Juntos, este flujo completa el ciclo que comenzó al iniciar sesión: pasa de la autenticación a la configuración del juego, a la experiencia activa y finalmente a una conclusión con resultados claros. Gracias a la combinación de un servidor en tiempo real, una estructura sólida en Firestore y pantallas de resúmenes bien diseñadas, la arquitectura técnica se convierte en una experiencia práctica, repetible y útil en diferentes contextos educativos.

4. ALOJAMIENTO, ACCESO Y DISTRIBUCIÓN

Desde el punto de vista operativo, la herramienta interactiva RockChain se ofrece como un servicio en la nube alojado más una aplicación móvil. Los centros de formación no necesitan instalar ni mantener servidores locales; solo necesitan acceso a internet y dispositivos Android o iOS adecuados.

Esta sección ofrece una visión concisa y técnica de cómo se aloja y accede a la herramienta. Para instrucciones paso a paso orientadas al formador sobre cómo preparar y ejecutar una sesión, los lectores pueden consultar las "Notas de la Guía" de WP4-A4, para que los usuarios puedan usar la herramienta con facilidad.

4.1. Alojamiento en backend

El backend está alojado en ***Google Cloud*** bajo el proyecto RockChain y está compuesto por los elementos descritos en la Sección 2:

- *Cloud Firestore*: almacena juegos, usuarios, datos de mercado y registros relacionados con el aprendizaje.
- *Firebase Cloud Functions*: implementa las operaciones autenticadas que crean y gestionan juegos, generan problemas de minería, asignan recompensas y actualizan perfiles de usuario.
- *Node.js + Socket.IO servidor en Cloud Run*: coordina las rondas en tiempo real y actúa como el cronometrador y árbitro autoritario.

Todos los datos centrales se almacenan en una región europea, alineando el despliegue con los requisitos de protección de datos de la UE y simplificando el cumplimiento para socios que operan en diferentes países. Como estos servicios están completamente gestionados, no es necesario tener servidores de bases de datos locales ni infraestructura personalizada en los centros educativos: el mantenimiento y la escalabilidad se gestionan a nivel de nube.

4.2. Distribución de aplicaciones móviles

El cliente RockChain se distribuye como una **aplicación móvil** para una de las dos principales plataformas:

- Una versión de Android (APK/AAB), adecuada para su instalación mediante listado de tienda o distribución directa a dispositivos gestionados. ENLACE
- Una versión para iOS (IPA mediante TestFlight o la App Store), dependiendo de la estrategia de distribución acordada con cada socio. ENLACE

También puedes encontrar los enlaces y/o códigos QR publicados en la sección web de RockChain. https://rockchain.eu/tool/

En la práctica, los formadores que preparan un curso pueden visitar el espacio web de RockChain, escanear el código QR o seguir el enlace de su plataforma, e instalar la versión de producción actual de la app en sus dispositivos o en tabletas propiedad de la institución.

4.3. Flujo de trabajo de acceso para formadores y alumnos

Para los usuarios finales, el acceso a RockChain sigue una secuencia sencilla y repetible:

1. **Instala la app**
    - Los alumnos y formadores instalan la app RockChain en sus dispositivos Android o iOS, ya sea en teléfonos personales/tabletas o en dispositivos propiedad de la institución donde la app puede estar preinstalada.
2. **Crea o usa una cuenta**
    - En el primer uso, se registran o inician sesión mediante Firebase Auth desde *LoginScreen*, tal y como se describe en la Sección 3.1.
    - Las cuentas pueden reutilizarse a lo largo de las sesiones y cursos, por lo que los usuarios no necesitan registrarse de nuevo cada vez.
3. **Únete o organiza sesiones de juego**
    - Una vez iniciado sesión, cada jugador aparece en *la Pantalla de Juego*, donde se prepara un código personal de juego para él en segundo plano.
    - Pequeños grupos deciden qué código usar: un jugador actúa como anfitrión compartiendo su código y compartiendo la partida, y los demás se unen usando la entrada "Unirse a la partida".
    - Los formadores pueden permitir que los alumnos organicen sus propias partidas (una por grupo) o actuar como anfitriones si quieren hacer una demo o controlar el tiempo más estrictamente.

Desde la perspectiva de un entrenador, esto significa que dirigir una sesión de RockChain consiste principalmente en:

- Asegurarse de que todos los participantes tengan la app instalada y puedan iniciar sesión.
- Organizar a los alumnos en pequeños grupos (cada uno con un anfitrión que comparte un código de juego).

- Monitorizar cómo progresan los grupos durante las rondas y utilizar las pantallas de resultados para el informe.

4.4. Requisitos para los centros de formación

Dado que RockChain depende del alojamiento en la nube y la distribución móvil, los requisitos de infraestructura para los centros de formación son mínimos:

- **Red**: una red Wi-Fi para el aula con acceso a internet, capaz de soportar conexiones simultáneas desde el número de dispositivos utilizados en una sesión.
- **Dispositivos**: Smartphones/tablets Android o iOS que cumplan los requisitos mínimos del sistema operativo para la versión desplegada (según lo especificado en la documentación de RockChain).
- **No hay instalación de servidores locales**: los centros no necesitan desplegar servidores adicionales; toda la coordinación, almacenamiento y autenticación del juego se gestionan en la nube.

Esta combinación, el backend de Google Cloud, aplicaciones móviles para Android e iOS, y el acceso a través del área web de RockChain, permite que la herramienta interactiva RockChain se utilice en diferentes países e instituciones sin una configuración local compleja, manteniendo al mismo tiempo un despliegue centralizado que puede ser mantenido y actualizado por los socios del proyecto.

5. MONITORIZACIÓN, RESILIENCIA Y MANTENIMIENTO

Un aspecto final de este documento es asegurar que la herramienta interactiva RockChain pueda ser monitorizada, depurada y mantenida a lo largo del tiempo, especialmente durante pilotos en aulas reales. El objetivo no es construir una pila pesada de observabilidad, sino proporcionar suficiente visibilidad y robustez para que los socios puedan entender lo que está ocurriendo y mantener el sistema estable.

5.1. Observabilidad y registro

Tanto el cliente como el servidor incluyen mecanismos ligeros de observabilidad que ayudan durante el desarrollo, el despliegue piloto y el análisis de incidentes.

En el lado del cliente, *GameContextHybridRobust* registra señales técnicas clave para la consola, tales como:

- Desplazamiento de reloj entre el dispositivo local y el servidor autorizado.
- Intentos de resincronización (por ejemplo, tras la reconexión).
- Detección de eventos duplicados o fuera de orden.

Estos registros se utilizan principalmente durante el desarrollo y las pruebas piloto, cuando desarrolladores o socios técnicos ejecutan la aplicación con herramientas de depuración adjuntas. Facilitan la reproducción y el diagnóstico de problemas relacionados con el tiempo, la navegación y la consistencia del estado.

En el lado del servidor, una *utilidad logMetric* escribe eventos JSON estructurados en Cloud Logging. Los eventos típicos incluyen:

- Nuevas conexiones y desconexiones.
- La ronda comienza y la ronda termina.
- Resincroniza las operaciones.
- Errores o condiciones inesperadas.

Con estos registros, el personal técnico puede:

- Mira cuántos partidos se han jugado en cada periodo.
- Identifica patrones de error o cuellos de botella de rendimiento (por ejemplo, resincronizaciones repetidas para ciertos juegos).
- Apoyan la depuración cuando los socios reportan incidentes, correlacionando los informes de usuarios con eventos concretos en los registros.

El enfoque general es intencionadamente ligero: no hay un panel de control complejo, pero hay suficiente información estructurada para apoyar el mantenimiento básico, la optimización y la resolución de problemas.

5.2. Resiliencia y manejo de fallos

Como RockChain es una herramienta multijugador en tiempo real utilizada sobre Wi-Fi en el aula, debe hacer frente a la conectividad intermitente y fallos temporales sin romper la experiencia. Por ello, varias estrategias de resiliencia están integradas en la arquitectura:

- **Reconexión automática de la zócala:** El adaptador de Socket.IO del cliente intenta reconectarse automáticamente cuando la red se pierde temporalmente. Tras una reconexión exitosa, vuelve a enviar el token de autenticación y *la información de join_game* para que el servidor pueda volver a conectar el socket al juego y al usuario correctos.
- **Operaciones idempotentes en el servidor**: Las operaciones del lado del servidor dependen de identificadores como *roundId*, números de versión y, cuando sea necesario, IDs de operación. Esto permite reconocer mensajes repetidos o tardíos e ignorarlos en lugar de aplicarlos dos veces, evitando recompensas duplicadas o estados inconsistentes cuando se vuelven a intentar mensajes.
- **Bloqueos a nivel de ronda para fases críticas:** Durante fases sensibles como los cálculos de final de ronda, el servidor utiliza bloqueos a nivel de ronda para que solo se ejecute un flujo de cálculo a la vez para cada partida y ronda. Esto evita las condiciones de carrera cuando muchas acciones llegan cerca del final de la cuenta atrás.
- **Gestión elegante de la desincronización**: Si un cliente ha estado desconectado, en segundo plano o sufre una breve desconexión, la combinación de *GameContextHybridRobust* y el servidor autorizado permite que el dispositivo se resincronize con el estado actual cuando regresa. En lugar de depender de lo que se había renderizado antes, el cliente puede actualizar su vista de la ronda activa, inventarios y clasificaciones.

En la práctica, estos mecanismos significan que los problemas de red a corto plazo o errores transitorios no deberían invalidar una sesión de juego. Los entrenadores suelen poder continuar con la actividad sin necesidad de intervención técnica profunda, y los aprendices que pierden brevemente la conexión pueden reincorporarse y seguir jugando de forma constante.

5.3. Mantenimiento y evolución futura

Desde el punto de vista del mantenimiento, WP4. A5 establece algunos principios simples pero importantes para guiar la evolución futura de la herramienta:

- Mantén la lógica en tiempo real centralizada en el servidor autoritario: todas las decisiones críticas en el tiempo (quién minó primero, cuándo termina la ronda,

cómo se aplican las recompensas) permanecen en el servidor. El cliente consume *authoritativeState* en lugar de intentar implementar sus propias reglas alternativas. Esto reduce el riesgo de divergencia entre diferentes versiones del cliente.

- Trata las Funciones de la Nube como el punto de entrada único para las operaciones de backend: Cualquier nueva operación que cambie datos persistentes debe implementarse como una Función en la Nube, registrada en funciones/index.js y siempre llamada a través de envoltorios autenticados en el cliente. Esto mantiene las comprobaciones de seguridad y la lógica de negocio en el servidor y facilita razonar sobre los flujos de datos.
- Cambios y migraciones de esquemas de *documentos:* Cuando evolucionan los esquemas de documentos Firestore (por ejemplo, añadiendo nuevos campos a *games/{gameId}* o *users/{userId}.games[gameId]*), esos cambios deben documentarse explícitamente. Si es necesario ajustar los datos existentes, los scripts de migración o utilidades pueden colocarse bajo *funciones/src* para que los socios tengan un camino claro para actualizar los datos de producción.

Siguiendo estas prácticas, los socios técnicos pueden añadir nuevas funciones —como pantallas adicionales, indicadores adicionales, registro prolongado o nuevos modos de juego— sin comprometer la estabilidad del juego existente. RockChain sigue siendo mantenible y extensible más allá de la vida útil inicial del proyecto, preservando al mismo tiempo el comportamiento central que se ha validado durante los pilotos en WP5.

6. CONCLUSIONES

WP4-A5 ha entregado una versión lista para producción de la herramienta interactiva RockChain, consolidando el trabajo de diseño, implementación y despliegue realizado en tareas anteriores de WP4. El sistema resultante combina un cliente móvil moderno y multilingüe, un backend basado en Firebase y un servidor en tiempo real autoritativo en Cloud Run, todo desplegado en una infraestructura cloud escalable y empaquetado en versiones de Android o iOS que pueden instalarse en dispositivos estándar utilizados en formación profesional y educación para adultos.

Al documentar explícitamente la arquitectura del sistema, los flujos de ejecución, el flujo de trabajo de producción y despliegue, así como los procedimientos de alojamiento, acceso y mantenimiento, esta actividad asegura que RockChain no sea un prototipo puntual, sino un activo reproducible y mantenible. El personal técnico tiene una referencia clara sobre cómo está estructurada la herramienta y cómo actualizarla, mientras que los formadores disponen de una aplicación estable e instalable que puede integrarse en cursos reales con una configuración local mínima.

De este modo, la herramienta interactiva RockChain se convierte en el núcleo práctico de la oferta de e-learning del proyecto: operacionaliza el trabajo pedagógico y curricular de WPs anteriores en una experiencia concreta y jugable que puede utilizarse en entornos de aula de países socios. La herramienta está ahora lista para apoyar actividades piloto en WP5 y proporciona una base sólida para una posible explotación y extensión futura más allá de la vida útil del proyecto.