WP4-A5. Erstellung eines interaktiven RockChain-Tools.

Inhalt

1. EINLEITUNG

Dieses Dokument präsentiert die Ergebnisse der Aktivität WP4.A5 – Erstellung des interaktiven RockChain-Tools. Während sich die bisherigen Aufgaben in WP4 auf die Datenschicht (WP4.A1), die Verfeinerung des E-Learning-Tools (WP4.A2) und die funktionalen Spezifikationen (WP4.A3) konzentrierten, beschreibt WP4.A5, wie die endgültige interaktive Version von RockChain erstellt, verpackt und für den Vertrieb und die Verwendung in realen Schulungskontexten vorbereitet wurde.

Das Ziel von WP4.A5 ist zweigeteilt. Einerseits bietet es einen konsolidierten Überblick über die technische Architektur des RockChain-Tools und zeigt, wie der mobile Client, die Backend-Dienste und die Echtzeit-Orchestrierung zusammenarbeiten, um Multiplayer-Schulungssitzungen zu unterstützen. Andererseits dokumentiert es den Produktions- und Bereitstellungs-Workflow, der zu einsatzbereiten Android- und iOS-Builds, einer stabilen Backend-Bereitstellung und grundlegenden Verfahren für Hosting, Zugriff und Wartung führt.

Das Ergebnis ist eine Beschreibung der Entstehungsgeschichte, die von technischen Mitarbeitern, die das Tool warten oder erweitern müssen, sowie von Projektpartnern genutzt werden kann, die einen klaren Überblick darüber benötigen, wie das interaktive RockChain-Tool zu einer produktionsreifen Ressource für die berufliche Bildung und Erwachsenenbildung geworden ist.

2. SYSTEMARCHITEKTUR DES INTERAKTIVEN ROCKCHAIN-TOOLS

Aus technischer Sicht ist das interaktive RockChain-Tool als dreischichtiges System aufgebaut:

- Ein mit Expo und React Native entwickelter **mobiler Client**.
- Ein **Firebase-Backend, das** Authentifizierung, persistenten Speicher und serverlose Funktionen bereitstellt.
- Ein **autoritativer Node.js + Socket.IO-Server,** der Echtzeitrunden koordiniert und sicherstellt, dass alle Spieler einen konsistenten Spielstatus sehen.

Diese Komponenten werden durch eine Internationalisierungsschicht, Dienstprogramme zur Konfiguration der Umgebung und grundlegende Beobachtungsmechanismen ergänzt.

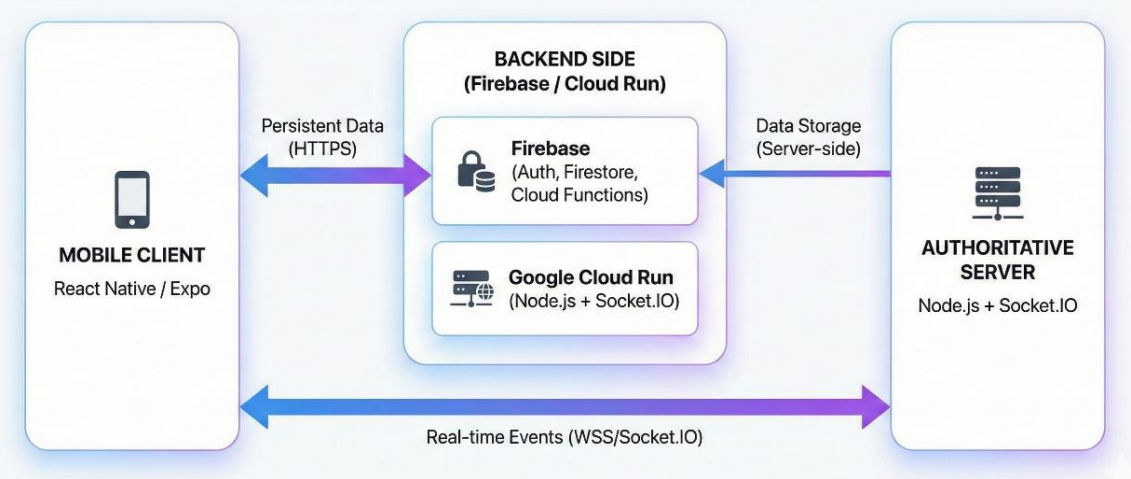

Abbildung1 : Architektur auf hoher Ebene.

2.1. Mobiler Client: Frameworks, Einstiegspunkt und Navigation

Der mobile Client von RockChain ist als React-Anwendung aufgebaut, die auf React Native läuft und von Expo gepackt und orchestriert wird. In der Praxis bedeutet dies:

- Alle Bildschirme und UI-Elemente sind als React-Funktionskomponenten unter Verwendung von JSX geschrieben.
- React 19 stellt das Komponentenmodell, Hooks (*useState, useEffect, useContext*, benutzerdefinierte Hooks) und die Context-API bereit, die die App verwendet, um den Spielstatus über Bildschirme hinweg auszutauschen.

- React Native 0.79 rendert diese Komponenten als native Ansichten auf Android und iOS, sodass sich die App wie eine native App anfühlt und verhält, während sie eine einzige JavaScript-Codebasis gemeinsam nutzt.
- Expo fungiert als Wrapper und Toolchain: Es stellt den Entwicklungsserver, das Bundling, das EAS-Build-System und den Zugriff auf native Dienste (Netzwerk, Speicher, Geräteinformationen) bereit, ohne dass separate Xcode-/Android Studio-Projekte gepflegt werden müssen.

Zur Laufzeit beginnt alles in *App.js*, dem Einstiegspunkt des Clients. Diese Datei verbindet die zentralen, übergreifenden Dienste, die jeder Bildschirm benötigt:

- Es sorgt für Internationalisierung, indem es den gesamten Komponentenbaum in *I18nextProvider i18n={i18n} ein.* Dadurch kann jeder Bildschirm Übersetzungsschlüssel anstelle von fest codierten Zeichenfolgen verwenden und automatisch die Sprache des Benutzers auswählen.
- Es umschließt die Benutzeroberfläche in einem *NavigationContainer* (aus React Navigation), der einen einzigen Navigationsbaum für die gesamte App definiert (Stapel, Registerkarten und verschachtelte Abläufe).
- Es umschließt die Navigation innerhalb von zwei globalen Providern, *GameProviderHybridRobust* und *SimpleMiningProvider,* und rendert ein *NetworkStatusBanner* auf der obersten Ebene:
  - *GameProviderHybridRobust* ist eine React Context + Hook-Schicht, die spielbezogene Zustände (aktuelles Spiel und Runde, Spieler, Inventare, Timer, Ranglisten, Navigationsschutz) offenlegt.
  - *SimpleMiningProvider* bietet einen dedizierten Mini-Kontext für Mining-Probleme, wodurch es einfacher wird, Proof-of-Work-Herausforderungen zu pushen und zu lösen, ohne sie eng mit dem Rest der Benutzeroberfläche zu koppeln.
  - *NetworkStatusBanner* überwacht Verbindungsänderungen und zeigt Warnungen an, wenn das Gerät offline ist oder eine schlechte Verbindung hat.

Da diese Anbieter über dem Navigationscontainer angesiedelt sind, kann jeder Bildschirm, egal wie tief er in der Stapelstruktur liegt, über Hooks auf den Spielstatus, den Mining-Status und den Netzwerkstatus zugreifen, anstatt diese Logik lokal neu zu implementieren.

Auf dieser technischen Grundlage folgt die App einem einfachen, linearen Navigationsablauf, der den Lebenszyklus eines Spiels widerspiegelt. React Navigation wird mit einem Hauptstapel verwendet, der die Benutzer durch folgende Phasen führt:

- *Anmeldung*: Hier registriert sich ein Spieler oder meldet sich mit Firebase Auth an.

- *Spiel:* Ein Wrapper-Bildschirm, der den Kontext für die ausgewählte *game-ID* und *user-ID* einrichtet.
- *WaitingRoom:* Hier versammeln sich die Spieler und sehen, wer vor Beginn der Runden am Spiel teilnimmt.
- *GameTabs:* Die Hauptumgebung im Spiel, die während der Runden verwendet wird.
- *Bildschirme für das Ende der Runde und die Ergebnisse*: Hier werden Zusammenfassungen, Ranglisten und Endergebnisse angezeigt.

Wenn der Benutzer *GameTabs* aufruft, wechselt das Layout zu einer tabbasierten Struktur. Die untere Tab-Leiste ist Teil des Navigationsbaums (ein Standard-React-Navigation-Tab-Navigator), kann jedoch visuell ausgeblendet werden, sodass das Erlebnis eher einem zusammenhängenden Spiel ähnelt als einer typischen App mit sichtbaren Tabs. Intern entscheiden Listener, die an die Socket.IO-Ereignisse und *GameContextHybridRobust* angehängt sind, wann zwischen den Modulen gewechselt wird – beispielsweise wird Mining automatisch geöffnet, wenn ein neues Problem eintrifft, oder es wird zur Zusammenfassung am Ende der Runde navigiert, wenn der Timer Null erreicht –, sodass die Lernenden sich nicht selbst um komplexe Übergänge kümmern müssen.

Während einer aktiven Runde stehen in *GameTabs* sechs Hauptmodule zur Verfügung:

- *Markt*: Das zentrale Modul, in dem die Spieler Steinblöcke, Abfall und recycelte Produkte kaufen und verkaufen. Es abonniert Produkt- und Preisaktualisierungsereignisse und rendert dynamische Listen basierend auf der aktuellen Marktlage.
- *Bergbau*: Eine fokussierte Ansicht, die erscheint, wenn eine Proof-of-Work-Herausforderung ausgelöst wird. Sie zeigt zeitgesteuerte Rechenaufgaben an, sammelt die Antworten lokal und sendet sie zur Validierung an den autoritativen Server.
- *Recycling*: Das Modul, in dem Teile des Inventars des Spielers in RockCoins oder neue Ressourcen umgewandelt werden können, wodurch die Logik der Kreislaufwirtschaft in konkreten Aktionen und Transaktionen umgesetzt wird.
- *Statistiken*: Ein Dashboard, das mit React Native-Komponenten und react-native-chart-kit implementiert wurde und Leistungsindikatoren und Ranglisten pro Runde und für das gesamte Spiel anzeigt.
- *Profil*: Ein persönlicher Bereich, der den Avatar des Spielers, den aktuellen Fortschritt und die Spracheinstellungen anzeigt und in zukünftigen Erweiterungen möglicherweise weitere grundlegende Einstellungen offenlegt.
- *Bestände*: Die Ansicht fasst Mengen und Preise für wichtige Produkte zusammen und dient als schneller „Marktüberblick". Spieler können sie öffnen, um sich einen kurzen Überblick darüber zu verschaffen, welche Ressourcen reichlich

vorhanden oder knapp sind, bevor sie sich für Kauf, Verkauf oder Recycling entscheiden.

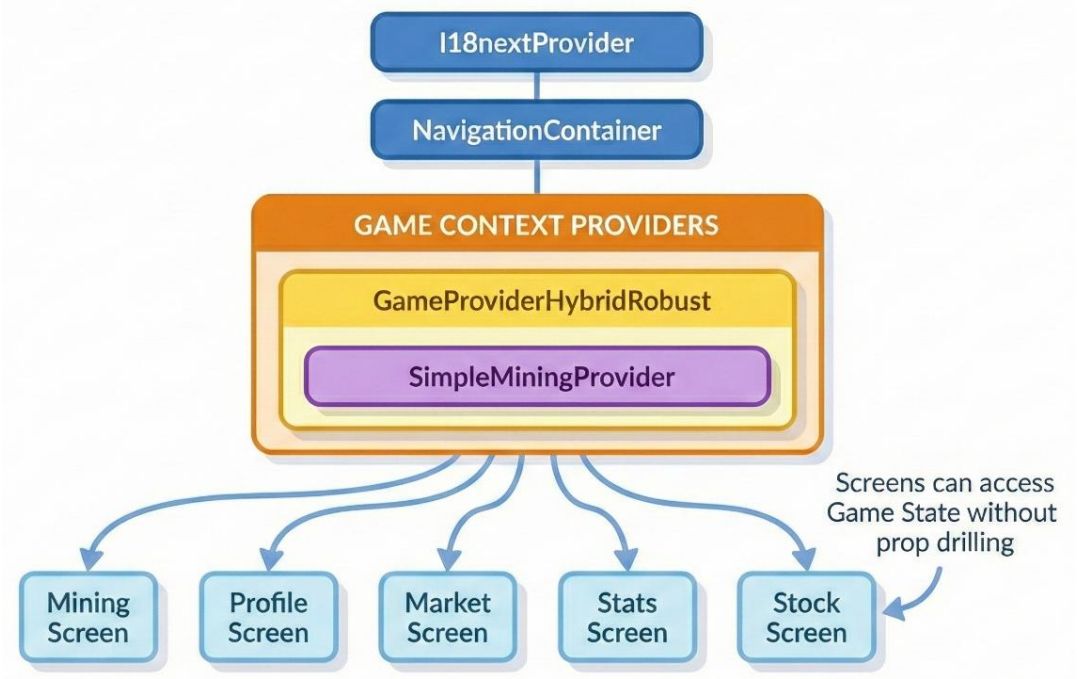

Abbildung2 : Komponenten- und Providerbaum.

Alle diese Bildschirme sind Standard-React-Komponenten, die ihre Daten in erster Linie von *GameProviderHybridRobust* und dem Socket.IO-Ereignisstrom erhalten und nicht aus dem lokalen Ad-hoc-Status. Diese Kombination aus React-Komponenten und Hooks, React Native-Rendering, Expo-Build/Runtime, React Navigation für die Flusskontrolle und Kontextanbietern für den gemeinsamen Status verwandelt den mobilen Client in ein zusammenhängendes, deklaratives Frontend, das relativ einfach zu verstehen und zu erweitern ist und sich dennoch wie ein natives Multiplayer-Spiel auf den Geräten der Lernenden anfühlt.

2.2. Clientseitige Zustandsverwaltung

Auf dem Client stützt sich RockChain stark auf die React Context API und benutzerdefinierte Hooks, um den gemeinsamen Status zu verwalten. Anstatt dass jeder Bildschirm seine eigene Kopie der Spieler, Timer oder Mining-Daten behält, zentralisiert die App den Großteil der Logik in einigen wenigen, genau definierten Anbietern. Bildschirme werden dann meist zu „Ansichten”: Sie abonnieren diesen Status und

rendern ihn, entscheiden aber nicht selbst über die Regeln. Dadurch bleibt die App auch dann vorhersehbar, wenn viele Ereignisse in Echtzeit eintreffen.

Das Herzstück dieser Ebene ist *GameContextHybridRobust*. Technisch gesehen handelt es sich um einen React Context, der von einer Provider-Komponente und einer Reihe von Hooks unterstützt wird, die den aktuellen Spielstatus für jeden Bildschirm sichtbar machen. Konzeptionell ist es die wichtigste Informationsquelle auf dem Client. Es enthält:

- Die Liste der Spieler im aktuellen Spiel und ihre Bestände (Produkte, Abfall, RockCoins).
- Die Countdown-Timer und Kennungen der aktuellen Runde, damit alle Bildschirme wissen, wie viel Zeit noch verbleibt und welche Runde aktiv ist.
- Den Mining-Status und die Ranglisten, sodass die Mining-Ergebnisse und Ranglisten in den Ansichten „Markt“, „Statistiken“ und anderen Ansichten konsistent sind.
- Eine synchronisierte Ansicht der Serveruhr, die aus Socket.IO-Nachrichten erstellt wird und es dem Client ermöglicht, die verbleibende Zeit abzuleiten, ohne zu weit vom maßgeblichen Server abzuweichen.
- Mehrere „sichere Navigations“-Schutzmechanismen, die Race Conditions verhindern, wenn die Navigation gleichzeitig mit der Verarbeitung neuer Ereignisse geändert wird (z. B. um zu vermeiden, dass ein Bildschirm gerade zum Ende der Runde veraltete Daten anzeigt).

GameContextHybridRobust speichert nicht nur Daten, sondern integriert auch eine Reihe von Idempotenz- und Resynchronisierungsmechanismen rund um die WebSocket-Verbindung. Jedes relevante Ereignis trägt Identifikatoren wie *round-ID* und version, und der Kontext verfolgt, was bereits angewendet wurde. Wenn die Verbindung unterbrochen und später wiederhergestellt wird, kann der Kontext den autoritativen Status vom Server erneut anfordern oder abgleichen, anstatt blindlings dem letzten gerenderten Zustand zu vertrauen. Dadurch kann ein Spieler sein Telefon für einen Moment sperren oder kurzzeitig die WLAN-Verbindung verlieren, ohne dass das Spiel komplett unterbrochen wird.

Die Mining-Logik ist in einem dedizierten Kontext gekapselt, der oft als *SimpleMiningContext* bezeichnet und über *SimpleMiningProvider* verfügbar gemacht wird. Die Idee dabei ist, eine leichtgewichtige Warteschlange mit Mining-Problemen und deren UI-Behandlung vom Rest des Spielzustands getrennt zu halten. Wenn der autoritative Server neue Mining-Herausforderungen ausgibt, werden diese in diese Warteschlange gestellt; wenn der Benutzer antwortet oder der Timer abläuft, werden sie verarbeitet und entfernt. Da das Mining in diesem isolierten Kontext abgewickelt wird, kann der Rest der Benutzeroberfläche auch dann reaktionsfähig bleiben, wenn mehrere Probleme schnell hintereinander erstellt und gelöst werden, und die Mining-

bezogene Benutzeroberfläche (wie Modals oder Countdowns) muss nicht manuell in jeden Bildschirm eingebunden werden.

Um diese beiden Hauptkontexte herum verwendet der Client eine Reihe von Hilfs-Hooks, um bestimmte Geschäftsregeln zu gruppieren:

- *useLearningProgress* überwacht Spielereignisse und zeichnet Bildungserfolge (z. B. abgebautes Gestein, reduzierter Abfall) in Firestore auf und kann visuelles Feedback auslösen, wenn bestimmte Meilensteine erreicht sind.
- *useNetworkStatus* umschließt die NetInfo-API und speist das *NetworkStatusBanner*, sodass jeder vorübergehende Verbindungsverlust für den Spieler sofort sichtbar ist.
- *useTutorial* steuert die Onboarding-Abläufe für Erstnutzer und entscheidet, wann Erklärungen oder Hinweise angezeigt werden und wann der Spieler frei interagieren kann.

Zusammen implementieren diese Kontexte und Hooks ein klares Designprinzip: Bildschirme sollten so einfach wie möglich sein, während übergreifende Funktionen (Timer, Spieler, Bergbau, Lernfortschritt, Konnektivität, Tutorials) in gemeinsamen, testbaren Modulen untergebracht sind. Dadurch ist die App leichter zu verstehen, einfacher zu erweitern (neue Bildschirme können dieselben Hooks wiederverwenden) und unter Echtzeitbedingungen robuster, da es einen einzigen Ort gibt, an dem der Spielstatus aus Backend-Ereignissen abgeleitet wird.

2.3. Firebase-Backend: Authentifizierung und persistente Daten

Im Backend setzt RockChain auf Firebase als verwaltete Plattform, die drei wichtige Dienste integriert: Authentifizierung, Cloud Firestore und Cloud Functions. Zusammen ermöglichen diese Dienste die sichere Verwaltung der Identität jedes Benutzers, die persistente Speicherung von Spieldaten und Lernfortschritten sowie die kontrollierte Ausführung sensibler Vorgänge direkt auf dem Server. Obwohl alle technischen Details zum Datenmodell, zu den Abläufen und zu den Sicherheitsregeln im Lieferumfang WP4-A1 dokumentiert sind, finden Sie hier einen Überblick darüber, wie diese Infrastruktur innerhalb des interaktiven Tools verwendet wird.

Das Herzstück des Systems ist Cloud Firestore, das als „Langzeitgedächtnis“ von RockChain fungiert. Anstelle von Tabellen wie in einer herkömmlichen Datenbank organisiert Firestore Informationen in Sammlungen und Dokumenten. In der Produktionsumgebung sind die wichtigsten Sammlungen:

- *Benutzer*: Enthält ein Dokument pro Spieler mit dessen Basisprofil und Leistung in jedem Spiel (rockCoins, Abfall, Produkte), gruppiert nach Spiel-ID.
- *Spiele*: ein Dokument pro Spielsitzung, in dem der Zugangscode, der Host, die Spielerliste, der aktuelle Status, die Runde und bestimmte Schlüsselindikatoren gespeichert sind, die sowohl vom Client als auch vom Server in Echtzeit verwendet werden.
- *Markt*: Produktkataloge und dynamische Preise pro Spiel, die vom Hauptdokument getrennt gehalten werden, um es nicht mit häufigen Aktualisierungen zu überladen.
- *Blockchain*: eine vereinfachte Historie der in jedem Spiel geminten Blöcke, in der aufgezeichnet wird, wer was wann gemint hat, sodass sich Blockchain-inspirierte Mechanismen in den Daten widerspiegeln.

Bei kritischen Vorgängen schreiben Clients nicht direkt in diese Sammlungen. Stattdessen rufen sie Serverfunktionen (Cloud Functions) auf, die mit besonderen Berechtigungen ausgeführt werden, Validierungen und Geschäftsregeln anwenden und erst dann die Daten in Firestore ändern. Diese Funktionen werden im Bericht WP4-A1 detailliert beschrieben, aber innerhalb der App haben sie drei Hauptaufgaben:

- Verwaltung des Spiellebenszyklus: Funktionen wie *createGame*, *joinGame* und *deleteGame* erstellen und löschen Spieldokumente, überprüfen die Eindeutigkeit des Codes und verhindern veraltete Sitzungen vom selben Host.
- Markt- und Mining-Logik: Funktionen wie *assignProductsToGame*, *updateProductPrices*, *generateMiningProblem* und *submitMiningSolution* steuern die Entwicklung von Produkten, Preisen und Mining-Herausforderungen und gewährleisten einheitliche Regeln für alle Spiele.
- Belohnungen und Profile: Am Ende jeder Runde oder jedes Spiels konsolidieren Funktionen wie *assignRoundRewards* und *updateUserProfile* die Ergebnisse und aktualisieren Benutzerprofile und Snapshot-Dokumente, wodurch die Konsistenz zwischen dem, was auf dem Bildschirm zu sehen war, und dem, was dauerhaft gespeichert wird, gewährleistet wird.

Darüber hinaus umfasst die Integration mit Firebase das Authentifizierungssystem und eine kleine Ebene lokaler Speicherung auf dem Gerät (*AsyncStorage*). Die Authentifizierung stellt sicher, dass jede Anfrage an das Backend mit einer eindeutigen Benutzer-ID verknüpft ist, die in Firestore und auf dem WebSocket-Server konsistent verwendet wird. Der lokale Speicher ermöglicht es der App, aktive Sitzungen zu speichern und wichtige Daten zwischen den Starts zu sichern, wodurch Unterbrechungen bei kurzen Verbindungsausfällen oder versehentlichem Schließen der App vermieden werden.

Einzelheiten zur vollständigen Implementierung, einschließlich Sicherheitsregeln, Datenflüssen und Aspekten im Zusammenhang mit der DSGVO, finden Partner im technischen Bericht WP4-A1.

2.4. Autoritativer Server und Echtzeitdienste

Obwohl Firebase die Speicher- und Serverlogik sicher und dauerhaft verwaltet, benötigt RockChain auch eine Ebene, die fast sofort auf Spieleraktionen reagiert, Timer verwaltet und in Echtzeit entscheidet, wer Mining-Herausforderungen gewinnt. Um diesen Anforderungen gerecht zu werden, verwendet das Projekt einen Echtzeitserver, der mit Node.js und Socket.IO entwickelt, in einem Docker-Image gepackt und auf Google Cloud Run bereitgestellt wurde. WP4-A1 beschreibt diese Architektur im Detail, aber hier ist eine Zusammenfassung, wie sie das interaktive Tool unterstützt.

Einfach ausgedrückt fungiert dieser Server als Schiedsrichter für jedes laufende Spiel. Für jedes Spiel speichert er im Speicher:

- Welche Spieler sind verbunden und in welchem Raum befinden sie sich?
- Ihre aktuellen Bestände und ihre Positionen in der Rangliste.
- Die aktive Mining-Herausforderung (falls vorhanden).
- Den offiziellen Status der Runde, einschließlich der genauen Zeit, zu der sie enden sollte.

Basierend auf diesem Status generiert der Server Identifikatoren wie *round-ID*, *version* und *roundEndsAtMs*, die an die gesendeten Ereignisse angehängt werden. Auf diese Weise wissen sowohl der Client als auch das Backend jederzeit, auf welche Runde sie sich beziehen, und es kommt nicht zu Verwechslungen durch doppelte oder verspätete Nachrichten.

Das Verhalten des Spiels ist als einfache Zustandsmaschine mit vier Hauptphasen strukturiert:

- PLAYING: Die Runde ist aktiv; die Spieler können kaufen, recyceln oder Mining-Probleme lösen, während der Timer läuft.
- ROUND_CALCULATING: Aktionen werden angehalten und der Server berechnet die Ergebnisse.
- ROUND_END: Die endgültigen Werte sind nun verfügbar; die Spieler können ihre Belohnungen und Positionen einsehen.
- WAITING_FOR_NEXT_ROUND: Das Spiel wird pausiert, bis der Host beschließt, eine neue Runde zu starten oder das Spiel zu beenden.

Während das Spiel diese Phasen durchläuft, sendet der Server verschiedene Ereignisse wie *start_round, ROUND_CALCULATING, round_ended, game_completed,*

inventory_data, player_rankings_data oder *economy:state*. Die mobile App empfängt diese Ereignisse und aktualisiert Bildschirme, Timer und Zusammenfassungen in Echtzeit, sodass alle Spieler einen einheitlichen Überblick über das Geschehen haben.

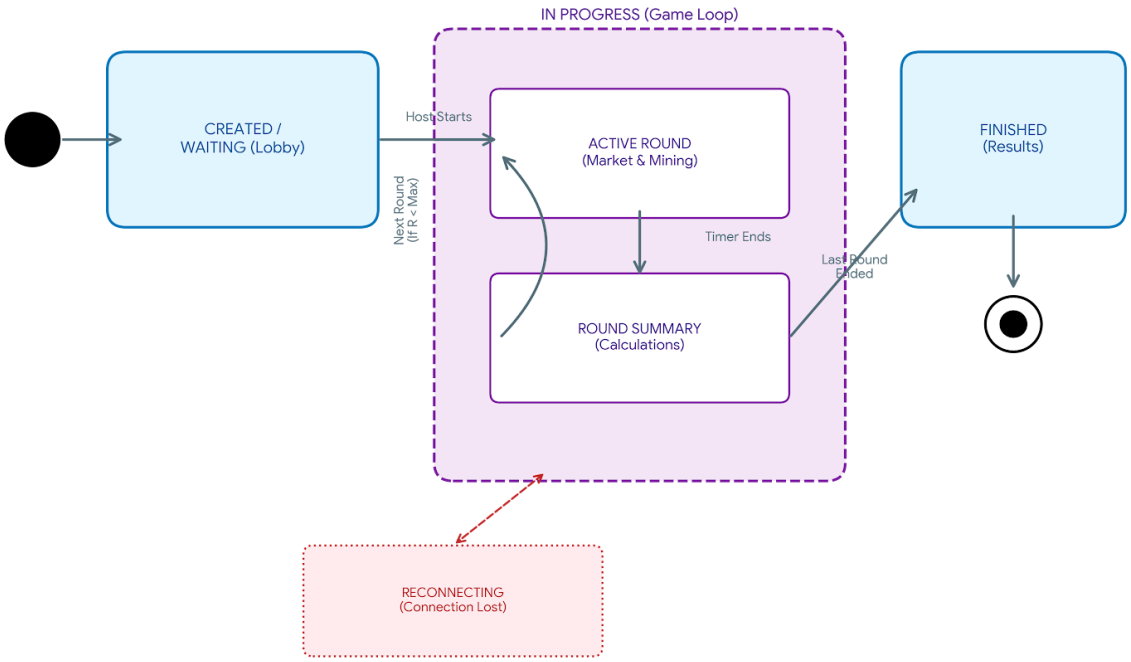

Abbildung „3 “: Rockchain-Zustandsmaschine.

Da viele Aktionen fast gleichzeitig eintreffen können (insbesondere am Ende einer Runde), verwendet der Server Sperrmechanismen und idempotente Aktualisierungen, um Konflikte zu vermeiden. Kritische Vorgänge, wie die Vergabe von Belohnungen oder die Aktualisierung von Beständen, werden in geordneter Weise ohne Überschneidungen ausgeführt und mit Rundenkennungen versehen, um Wiederholungen zu vermeiden. Wenn der Server auf Firebase zugreifen kann, speichert er auch die Ergebnisse jeder Runde (Bestände, Belohnungen, wirtschaftlicher Status), um sicherzustellen, dass der In-Memory-Snapshot und die dauerhaft gespeicherten Daten übereinstimmen.

Auf der Client-Seite ist diese gesamte Komplexität in einem Socket.IO-Adapter gekapselt. Dieser Adapter wählt automatisch den richtigen Server (lokal oder Cloud Run) aus, öffnet und pflegt die WebSocket-Verbindung, leitet Authentifizierungs- und Spielinformationen nach einer erneuten Verbindung weiter und übersetzt Backend-Ereignisse in das von der mit React entwickelten App erwartete Format. Dank dieser mittleren Schicht kann sich der Server im Laufe der Zeit weiterentwickeln, ohne die Benutzererfahrung oder die pädagogische Funktionsweise des Tools zu beeinträchtigen.

2.5. Übergreifende Aspekte: Internationalisierung, Konfiguration und Beobachtbarkeit

Über die Hauptkomponenten für Client und Backend hinaus umfasst RockChain mehrere übergreifende Mechanismen, die das Tool länderübergreifend nutzbar machen, die Bereitstellung in verschiedenen Umgebungen vereinfachen und die Fehlerbehebung während Pilotprojekten erleichtern. Drei davon sind besonders relevant: Internationalisierung, Umgebungskonfiguration und Beobachtbarkeit.

2.5.1. Internationalisierung und Barrierefreiheit

Von Anfang an wurde RockChain als Tool für mehrere Länder konzipiert. Das bedeutete, dass die Sprachunterstützung nicht nachträglich hinzugefügt werden konnte. Stattdessen wurde in das Projekt eine auf i18next basierende Internationalisierungsschicht integriert.

Die Kernkonfiguration befindet sich in *src/i18n/index.js*, die:

- Lädt Sprachpakete für Englisch (EN), Spanisch (ES), Deutsch (DE), Kroatisch (HR) und Rumänisch (RO).
- Die Sprache des Geräts erkennt oder anhand der expliziten Auswahl des Spielers entscheidet, welches Paket aktiviert werden soll.
- Eine i18n-Instanz bereitstellt, die der Rest der App über I18nextProvider verwendet.

Alle benutzerseitigen Texte werden als Übersetzungsschlüssel in *src/i18n/locales/*.json* gespeichert. Für jede unterstützte Sprache gibt es eine entsprechende JSON-Datei, in der diese Schlüssel realen Zeichenfolgen zugeordnet sind. Komponenten fordern dann Texte anhand des Schlüssels an, anstatt Englisch oder eine andere Sprache fest zu codieren.

Um die Erfahrung dauerhaft zu machen, wird die ausgewählte Sprache in AsyncStorage gespeichert. Das bedeutet:

- Bei der ersten Ausführung der App kann die Sprache des Geräts als Standard verwendet werden.
- Wenn der Benutzer die Sprache über die Benutzeroberfläche ändert, wird diese Einstellung lokal gespeichert.
- Bei späteren Starts stellt die App dieselbe Sprache wieder her, ohne erneut danach zu fragen.

Das Hinzufügen einer neuen Sprache ist unkompliziert und erfordert keine Änderungen am Kerncode:

1. Erstellen Sie eine neue JSON-Datei unter *src/i18n/locales/* mit denselben Schlüsseln wie die vorhandenen Sprachen.

2. Registrieren Sie den neuen Sprachcode in *supportedLngs* in *src/i18n/index.js*.
3. Stellen Sie Übersetzungen für alle relevanten Schlüssel bereit.

Dieses Design unterstützt eines der Kernziele von RockChain: Das Tool kann auf weitere Länder und Kontexte übertragen werden, indem an den Übersetzungsdateien gearbeitet wird, anstatt die Logik der App zu ändern.

2.5.2. Umgebungskonfiguration und Netzwerkerkennung

Da RockChain in mehreren Ausführungskontexten (lokale Entwicklung, Staging und Produktion) laufen muss, vermeidet das Projekt fest codierte URLs oder Annahmen zur Umgebung. Stattdessen verwendet es eine kleine, aber explizite Konfigurationsebene.

Zwei Module sind hier von zentraler Bedeutung:

src/config/environment.js

- Berechnet Werte wie SOCKET_URL, NODE_ENV, Timeouts und Debug-Flags.
- Liest aus den Einstellungen zur Erstellungszeit und den Umgebungsvariablen, um beispielsweise zu entscheiden, ob die App mit einem lokalen Socket.IO-Server, einer Staging-Instanz oder dem Produktionsdienst Cloud Run kommunizieren soll.

src/utils/networkUtils.js

- Erkennt, ob die App in einem Simulator/Emulator oder auf einem physischen Gerät ausgeführt wird.
- Wählt auf Grundlage dieser Informationen die geeignete Serveradresse aus:
    - Localhost oder eine bestimmte LAN-IP während der Entwicklung.
    - Die Cloud Run-URL in Staging- oder Produktions-Builds.

Darüber hinaus ermöglichen Expo-Umgebungsvariablen (wie EXPO_PUBLIC_WEBSOCKET_URL) dem Projekt, Endpunkte in bestimmten Build-Profilen zu überschreiben. Beispielsweise kann ein Produktions-EAS-Build den WebSocket-Endpunkt fest mit der offiziellen Cloud Run-URL verbinden, während ein Preview-Build auf eine Testinstanz verweisen kann.

Das Ergebnis ist, dass dieselbe Codebasis für verschiedene Umgebungen neu erstellt werden kann, indem einfach das richtige Build-Profil und die richtige Konfiguration ausgewählt werden, ohne dass Quelldateien bearbeitet werden müssen. Dies unterstützt direkt die Reproduzierbarkeit und erleichtert es Partnern, die Bereitstellung zu klonen oder zu aktualisieren.

2.5.3. Beobachtbarkeit und Ausfallsicherheit

Schließlich umfasst die Architektur einen Mindestsatz an Tools, um in Echtzeit zu verstehen, was gerade passiert, und um vorübergehende Netzwerkprobleme zu überstehen. Dies ist besonders wichtig im Unterrichtskontext, wo die WLAN-Qualität variieren kann.

Auf der Client-Seite:

- *GameContextHybridRobust* protokolliert Informationen über Taktabweichungen, doppelte Ereignisse und Resynchronisierungsversuche in der Konsole. Während der Entwicklung und Pilotphase helfen diese Protokolle dabei, Situationen zu identifizieren, in denen der Client und der autoritative Server vorübergehend voneinander abweichen, und festzustellen, wie oft eine Neusynchronisierung erforderlich ist.
- Die Kombination aus automatischer Socket-Wiederverbindung und React Context stellt sicher, dass der Client bei Wiederherstellung der Verbindung den aktuellen autoritativen Status erneut anfordern kann, anstatt sich auf veraltete Daten zu verlassen.

Auf der Serverseite:

- Ein *logMetric*-Dienstprogramm gibt strukturierte JSON-Ereignisse für wichtige Momente wie Verbindung, *round_start*, *round_end*, resync und error aus. Wenn der Server auf Cloud Run läuft, können diese Ereignisse von Google Cloud Logging oder ähnlichen Tools erfasst werden, wodurch eine Zeitleiste der Ereignisse in jeder Spielsitzung bereitgestellt wird.
- Der Server erzwingt eine idempotente Ereignisbehandlung unter Verwendung von Identifikatoren wie *roundId*, version und, falls relevant, *operationId*. Dies verhindert doppelte Belohnungen oder inkonsistente Zustände, wenn Nachrichten erneut gesendet werden oder verspätet eintreffen.
- Rundenbezogene Sperren (*roundLocks*, *gameStates*) stellen sicher, dass kritische Abschnitte – wie Berechnungen am Ende einer Runde – nicht gleichzeitig für dasselbe Spiel und dieselbe Runde ausgeführt werden.

Zusammengenommen bedeuten diese Mechanismen, dass die Architektur nicht nur für Piloten funktional, sondern auch nachvollziehbar und robust ist. Entwickler und technisch versierte Partner können Protokolle einsehen, um zu verstehen, wie sich Sitzungen entwickelt haben, während Spieler ein Spiel erleben, das auch bei kurzzeitigen Verbindungsproblemen weiterhin funktioniert.

3. WICHTIGE LAUFZEITABLÄUFE IM INTERAKTIVEN ROCKCHAIN-TOOL

Obwohl im vorigen Abschnitt erklärt wurde, wie die Architektur von RockChain aufgebaut ist, konzentrieren wir uns hier darauf, was eine Person bei der Verwendung des Tools erlebt: wie sie sich anmeldet, wie sie einem Spiel beitritt oder ein Spiel erstellt, wie die Runden verlaufen und welche Reaktionen das System bietet. Kurz gesagt, wir betrachten den schrittweisen Prozess aus der Sicht des Benutzers.

3.1. Authentifizierung und Onboarding

Alles beginnt auf dem Anmeldebildschirm (*LoginScreen*), wo Benutzer zwei Dinge tun können:

- Ein neues Konto erstellen, indem sie die erforderlichen Mindestangaben (wie E-Mail-Adresse, Passwort und einen Namen, der im Spiel angezeigt wird) eingeben.
- Sich mit einem bestehenden Konto anmelden und dabei ihre gespeicherten Anmeldedaten wiederverwenden.

Dieser Bildschirm ist direkt mit dem Firebase-Authentifizierungssystem verbunden. Wenn jemand seine Daten eingibt und auf „Enter“ klickt, führt die App Folgendes aus:

1. Sie sendet die Anmeldedaten zur Überprüfung an Firebase.
2. Wenn alles korrekt ist, gibt Firebase eine authentifizierte Benutzer-ID und einen Zugriffstoken zurück.
3. Wenn ein Fehler auftritt (z. B. falsches Passwort oder E-Mail-Adresse bereits registriert), zeigt die App eine eindeutige Meldung an und fordert zur Korrektur der Daten auf.

Was passiert nach der erfolgreichen Anmeldung?

Drei Dinge geschehen automatisch und fast gleichzeitig:

Das Profil wird in der Datenbank erstellt (oder abgerufen).

- Wenn es sich um ein neues Konto handelt, erstellt die App oder eine Funktion auf dem Server ein Dokument in der Datenbank mit grundlegenden Informationen: sichtbarer Name, E-Mail-Adresse, Erstellungsdatum, Standardsprache usw.
- Wenn es bereits existierte, wird das Profil zusammen mit dem Verlauf früherer Spiele, falls vorhanden, wiederverwendet.

Es wird ein sicheres Token für Spielaktionen abgerufen.

- Mit diesem Token kann die App mit dem Server kommunizieren und auf geschützte Funktionen zugreifen, z. B. das Erstellen oder Beitreten von Spielen.
- Es wird auch verwendet, um in Echtzeit eine Verbindung zum Server herzustellen, damit das System weiß, wer hinter jeder Aktion steht.

Zugriff auf den Hauptspielbereich.

- Die App zeigt den Anmeldebildschirm nicht mehr an und wechselt direkt zum Spielbereich, wo die Person folgende Möglichkeiten hat:
- Ein neues Spiel starten.
- Durch Eingabe eines Codes an einem bestehenden Spiel teilnehmen.
- Grundlegende Informationen über RockChain anzeigen.
- Oder ihre bisherigen Erfolge anzeigen.

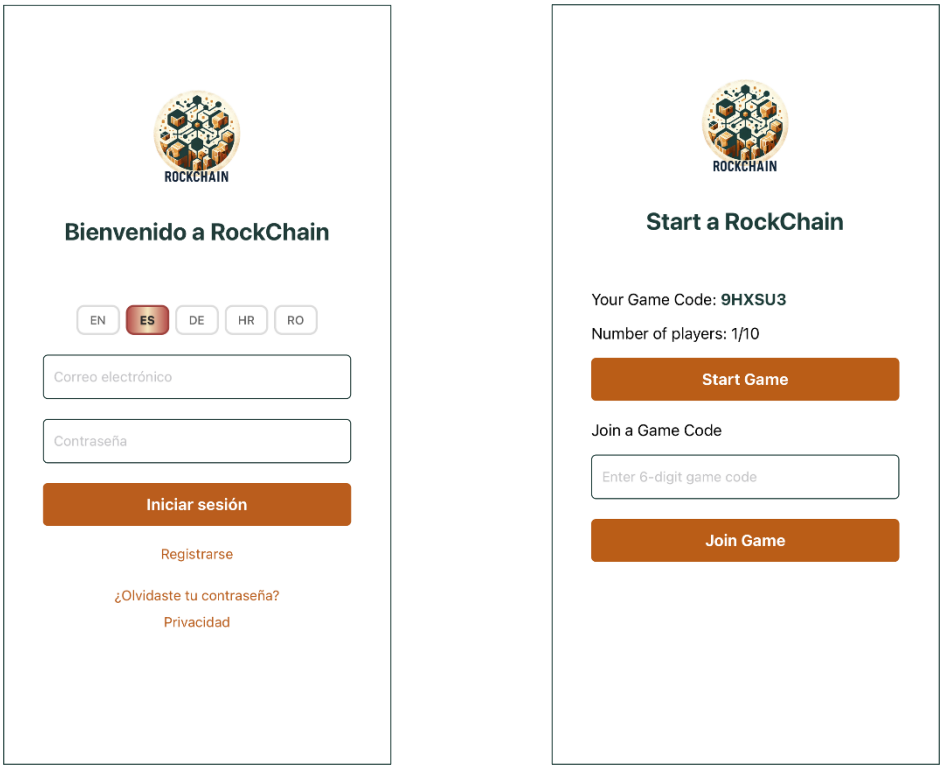

Abbildung4 : Anmelde- und Spielbildschirme.

RockChain unterscheidet nicht zwischen „normalen Konten" und „Host-Konten". Jeder, der angemeldet ist, kann ein Spiel erstellen und wird zum Host, wenn er seinen Code teilt und andere Spieler seiner Sitzung beitreten. Dadurch ist der Ablauf für alle gleich: Jede Person greift auf ihren Spielbereich zu, und dann entscheiden die Gruppen ganz natürlich, welchen sie als gemeinsames Spiel verwenden möchten.

3.2. Erstellen von Spielen und Beitreten durch Spieler

Der Lebenszyklus eines Spiels auf RockChain ist so einfach und nahtlos wie möglich gestaltet. Sobald sich eine Person anmeldet, behandelt die App sie wie alle anderen: Sie erhält automatisch ein eigenes Spiel, das sie hostet, und kann von dort aus entscheiden, ob sie es als Sitzung für ihre Gruppe verwenden oder mit einem Code an der Sitzung einer anderen Person teilnehmen möchte.

Es muss nicht zwischen „Erstellen" und „Hosten" entschieden werden: Wenn Sie Ihren Code teilen und das Spiel starten, sind Sie der Host. So einfach ist das.

3.2.1. Automatisches Host-Spiel auf *GameScreen*

Unmittelbar nach der erfolgreichen Anmeldung leitet die App die Person zum Hauptbildschirm des Spiels weiter. In diesem Moment erstellt die Anwendung automatisch ein persönliches Spiel im Hintergrund: Es muss kein „Erstellen"-Button gedrückt werden.

Im Backend übernimmt dieser Prozess folgende Aufgaben:

- Es wird überprüft, ob diese Person bereits aktive Spiele hatte, und wenn ja, werden diese geschlossen, um Duplikate zu vermeiden.
- Es wird sichergestellt, dass der globale Produktkatalog und grundlegende Marktdaten verfügbar sind.
- Erstellen eines neuen Spieldokuments mit:
    - Einem eindeutigen Code zum Teilen (z. B. *ABC123*)
    - Der Benutzer-ID als Host
    - Anfangsstatus „Warten"
    - Rundenzähler auf Null gesetzt
    - Und allen erforderlichen Standardeinstellungen

Nach Abschluss der Erstellung verfügt die App über alles, was sie zum Starten benötigt. Auf dem Bildschirm sieht die Person etwa Folgendes:

- *„Ihr Spielcode:* ABC123" oben.
- *„Verbundene Spieler: 1*".
- Eine Schaltfläche „Spiel *starten*".
- Ein Feld zur Eingabe eines Codes, wenn sie an einer anderen Sitzung teilnehmen möchte.

Wenn Sie sich entscheiden, Ihr Spiel als Gruppensitzung zu nutzen, gehen Sie einfach wie folgt vor:

- Teilen Sie Ihren Code mit Ihren Teamkollegen (laut, auf dem Bildschirm, per Chat usw.).
- Warten Sie, bis alle verbunden sind (dies wird auf dem Spielerzähler angezeigt).

- Drücken Sie auf „Spiel starten“.
- Es sind keine weiteren Maßnahmen erforderlich: Das persönliche Spiel wird zum offiziellen Gruppenspiel.

3.2.2. An einem Spiel anderer Spieler teilnehmen

Das Gegenteil kann auch passieren, wenn Sie diesen Bildschirm erreichen: Möglicherweise haben Sie bereits Ihr eigenes Spiel erstellt, möchten aber einem anderen Spiel beitreten, das ein Freund bereits gestartet hat.

Dafür ist der Bereich „Einem Spiel beitreten” vorgesehen. Sie müssen lediglich Folgendes tun:

- Geben Sie den vom Gastgeber freigegebenen Sitzungscode ein.
- Drücken Sie die Taste *„An Spiel teilnehmen*“.

Die App wird dann:

- Nach diesem Spiel anhand des Codes suchen.
- Überprüfen, ob es noch für neue Spieler offen ist.
- Fügt den Benutzer zur Teilnehmerliste hinzu.
- Erstellt oder aktualisiert seine persönlichen Spieldaten (*Münzen, Inventar, Abfall usw*.).
- Ändert den Kontext seines Spiels: Von diesem Moment an wird er Teil der Sitzung des Gastgebers.

Von diesem Moment an sieht die Person den Code des Gastgebers und die korrekte Anzahl der verbundenen Spieler auf dem Bildschirm. Wenn die Gruppe vollständig ist, kann der Gastgeber auf *„Spiel* starten“ klicken, wodurch alle in den Warteraum gelangen und das Spiel beginnt.

3.3. Warteraum und Synchronisation

Sobald eine Person beschließt, ihr eigenes Spiel als Gastgeber zu nutzen oder dem Spiel einer anderen Person beizutreten, werden alle Teilnehmer vom Hauptbildschirm des Spiels in den Warteraum gebracht. Dieser Bildschirm hat eine ganz bestimmte Aufgabe: die Gruppe an einem stabilen Startpunkt zu versammeln und sicherzustellen, dass alle Geräte perfekt synchronisiert sind, wenn der Countdown beginnt.

Die Lobby verwaltet ihren Status nicht separat. Stattdessen überwacht sie zwei wichtige Informationsquellen:

- *GameContextHybridRobust*, das Folgendes sammelt:
  - Aktualisierungen des Spieldokuments (games/{*game-ID*}), wie z. B. neue Spieler oder Statusänderungen

- Die aktuelle Liste der Spieler und den Gesamtstatus des Spiels (Warten, im Spiel, beendet usw.)
- Echtzeit-Serverereignisse (*Socket.IO*), die Folgendes anzeigen:
  - den Beginn des Countdowns,
  - Änderungen der Spielphase (z. B. Wechsel von „Warten" zu „Spielen")

Mit diesen Informationen zeigt die Lobby nur das Wesentliche an:

- Die Liste der mit diesem Spiel verbundenen Spieler,
- den aktuellen Spielstatus (Meldungen wie *„Warten auf* Spieler", *„Bereit zum* Start", *„Start in 3...2...1...*"),
- Optionale Anzeigen, dass die Spieler bereit sind, sofern diese Funktion aktiviert wurde

Was passiert, wenn der Gastgeber auf *„Spiel starten"* drückt?

Wenn der Gastgeber der Meinung ist, dass die Gruppe bereit ist, und auf *„Spiel starten"* drückt:

1. Ihr Gerät ruft die Funktion *„startGame"* im Backend auf und sendet die Spiel-ID.
2. Der Server überprüft Folgendes:
   - Das Spiel befindet sich in einem gültigen Zustand (z. B. es läuft noch nicht und es sind genügend Spieler vorhanden).
   - Er koordiniert sich mit dem Echtzeitserver, um den Start der ersten Runde vorzubereiten:
     - Eine eindeutige Rundenkennung (*round-ID*) wird generiert
     - Der genaue Zeitpunkt, zu dem die Runde endet (*roundEndsAtMs*), wird berechnet
     - Das Spiel wird als bereit markiert, um nach dem Countdown zur Spielphase überzugehen

Der Server gibt dann über *Socket.IO* eine Reihe von Ereignissen aus:

- Ein Countdown-Start-Ereignis (*„Die Runde beginnt in wenigen* Sekunden")
- Optionale Ereignisse, die den Fortschritt des Countdowns markieren (*„3... 2... 1...*")
- Ein Ereignis, das anzeigt, dass die Runde offiziell begonnen hat

Jeder Warteraum empfängt diese Ereignisse, und *GameContextHybridRobust* verwendet den Zeitstempel (*roundEndsAtMs*), um die verbleibende Zeit auf jedem Gerät zu berechnen. Selbst wenn es geringfügige Unterschiede bei der WLAN-Verbindung gibt, verlassen alle Spieler den Warteraum und treten mit derselben Zeitbegrenzung in die Runde ein.

Der Warteraum fungiert als Synchronisationskammer. Er stellt sicher, dass:

- Alle sind im Spiel miteinander verbunden und sichtbar.
- Der Gastgeber kann den genauen Startzeitpunkt festlegen.
- Dank einer Kombination aus Cloud-Funktionen und Echtzeitkommunikation beginnt die gesamte Gruppe die Runde gleichzeitig.

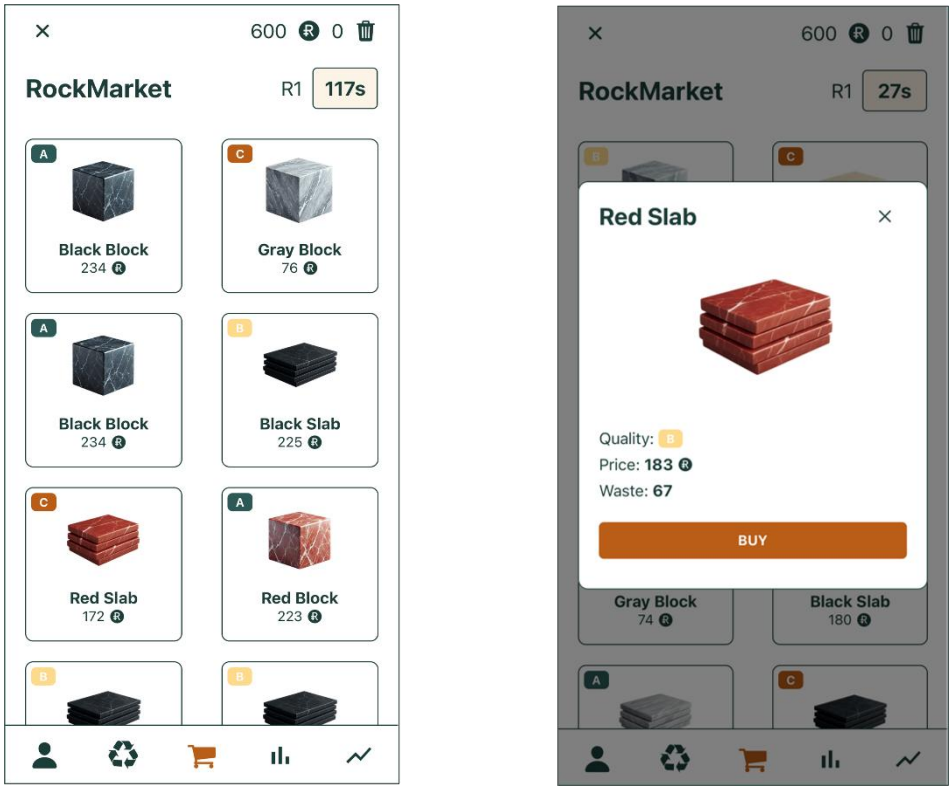

Abbildung5 : MARKET-Bildschirm.

3.4. Spielablauf während der Runde: Navigation zwischen den Bildschirmen

Wenn eine Runde beginnt, verschiebt die App automatisch alle Spieler aus der Lobby zur Hauptspieloberfläche, die als ***GameTabs*** bezeichnet wird. Von diesem Zeitpunkt an kann sich jede Person während der gesamten Runde frei zwischen mehreren wichtigen Bildschirmen bewegen.

Die Spielzeit wird durch einen zentralen Wert namens *roundEndsAtMs* gesteuert, der genau definiert, wann die Runde enden soll. Dieser Wert wird vom Server in Echtzeit bereitgestellt und vom Spielkontext (*GameContextHybridRobust*) in einen lokalen Countdown umgewandelt, sodass alle Geräte unabhängig von geringfügigen Verbindungsverzögerungen die gleiche Zeitbegrenzung haben.

Während einer typischen 120-Sekunden-Runde können die Spieler mit den folgenden Modulen interagieren:

- *MarketScreen*: Hier können Spieler Produkte (*Blöcke, Platten, recycelte Materialien*) kaufen und verkaufen. Die Produktliste wird automatisch aktualisiert, wenn der Server Preise oder Angebote ändert.
- *RecycleScreen*: Hier können Sie einen Teil Ihres Inventars in RockCoins oder neue Materialien umwandeln. Jede Aktion wird im Backend validiert, das vor der Ausführung überprüft, ob die Regeln eingehalten werden.
- *MiningScreen*: Bietet „Proof-of-Work"-Herausforderungen. Die App zeigt mathematische Aufgaben mit einem Zeitlimit an, der Spieler gibt seine Antwort ein und der Server entscheidet, wer richtig und wer am schnellsten war.
- *StatsScreen*: Zeigt Leistungsindikatoren und Ranglisten mit Grafiken an, die entsprechend dem Fortschritt aktualisiert werden.
- *ProfileScreen*: Bietet eine Zusammenfassung des Benutzerprofils, der aktuellen Ressourcen und der RockCoins. Hier können Sie auch die Sprache ändern. Die Informationen stammen direkt aus Firestore.

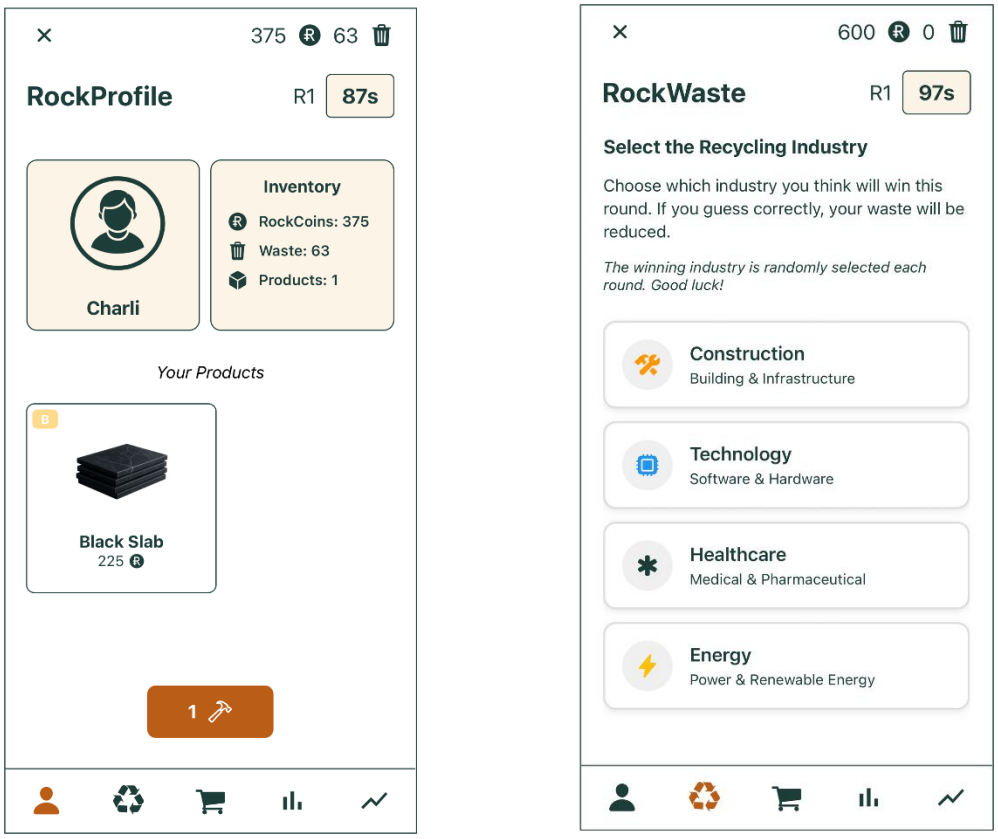

Abbildung6 : PROFIL- und RECYCLE-Bildschirme.

- *StockScreen*: Bietet einen Überblick über den Markt: Welche Produkte sind verfügbar, wie variieren die Preise und welche Ressourcen sind knapp oder reichlich vorhanden? Dies ist nützlich für die Planung vor dem Kauf oder Recycling.

Ein Grundprinzip bleibt auf allen diesen Bildschirmen gleich: Die App trifft niemals selbst endgültige Entscheidungen. Jede vom Spieler ausgeführte Aktion wird als Absicht interpretiert, die an den Server gesendet wird. Zum Beispiel:

- *„Ich möchte Produkt X zum aktuellen Preis kaufen."*
- *„Ich möchte diesen Abfall in dieser Branche recyceln."*
- *„Dies ist meine Antwort auf die Bergbau-Herausforderung"*

Der Echtzeit-Server empfängt jede Aktion, prüft, ob sie gemäß dem aktuellen Spielstatus (Inventar, Preise, Timer usw.) gültig ist, und:

- Wendet sie an, wenn alles in Ordnung ist, und aktualisiert die Daten im Speicher (und gegebenenfalls in Firestore).
- Er lehnt sie ab, wenn sie nicht den Regeln entspricht oder zu spät eingeht.

Der Server sendet dann die Aktualisierungen an alle Spieler: neue Inventare, Ranglisten, Preisänderungen oder Bergbauergebnisse.

Dank dieser Struktur ist jede Runde:

- Interaktiv: Die Spieler können zwischen verschiedenen Bildschirmen wechseln und frei Entscheidungen treffen.
- Fair und konsistent: Für alle gelten die gleichen Regeln, und der Spielstatus wird für die gesamte Gruppe synchronisiert.
- Zeitgesteuert: Der Wert *„roundEndsAtMs"* stellt sicher, dass alle Spieler die Runde zur gleichen Zeit beenden, unabhängig von geringfügigen Unterschieden bei ihren Geräten oder Netzwerken.

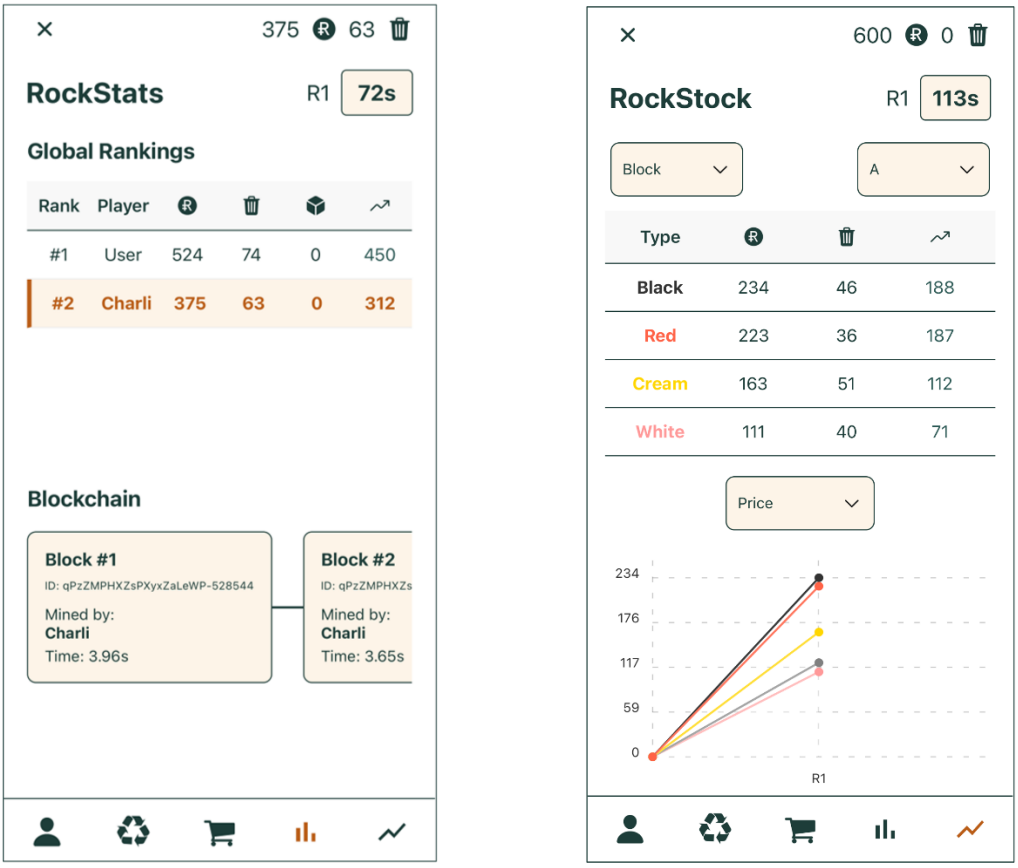

Abbildung7 : Bildschirme STATS und STOCK

3.5. Berechnungen am Ende der Runde und Endergebnisse

Wenn die Rundenzeit abgelaufen ist (*roundEndsAtMs*), akzeptiert das System keine neuen Aktionen von Spielern mehr. Von diesem Moment an geht die Kontrolle vollständig an das Backend über, das für das Einfrieren des Spielzustands, die Berechnung der Ergebnisse und die übersichtliche und konsistente Anzeige der Informationen verantwortlich ist.

Auf dem Echtzeitserver (Socket.IO) umfasst der Prozess zum Beenden einer Runde folgende Schritte:

1. **Aktionen einfrieren:**
   Der Server ändert den Spielstatus auf *ROUND_CALCULATING* und akzeptiert keine neuen Züge mehr. Alle Nachrichten, die zu spät eintreffen, werden ignoriert oder als ungültig markiert.
2. **Sichere Berechnung der Belohnungen:**
   Mithilfe von Sperren und Überprüfungen auf Rundenebene, um Wiederholungen zu verhindern, führt der Server folgende Schritte durch:
   - Verarbeitet jede Aktion nur einmal

- Verhindert doppelte Belohnungen oder die doppelte Zählung desselben Ereignisses
- Stellt sicher, dass gleichzeitige Aktionen keine Fehler oder inkonsistente Daten verursachen

3. **Auswahl der Gewinnerbranche:**
   Die konfigurierte Logik wird angewendet, um zu bestimmen, welche Branche die Runde gewonnen hat – eine wichtige Information für das Feedback, das die Spieler erhalten.
4. **Speichern der Ergebnisse in Firestore**
   Der Server aktualisiert:
   - Das Spieldokument in games/{*game-ID*} (Status, Rundennummer, Punktestände)
   - Die individuellen Statistiken in users/{*user-ID*}.*games*[*game-ID*]

   Sofern verfügbar, wickelt Firebase Admin diese Vorgänge sicher und effizient ab.

5. **Geräte benachrichtigen**
   Sobald die Berechnungen durchgeführt und die Daten gespeichert wurden, sendet der Server eine Reihe von Ereignissen, wie z. B.:
   - *ROUND_CALCULATING*
   - *ROUND_ENDED*
   - *REWARDS_ASSIGNED*

   Diese Meldungen ermöglichen es den Clients, die aktive Phase des Spiels zu beenden, anzuzeigen, dass die Runde vorbei ist, und anschließend die detaillierten Ergebnisse anzuzeigen.

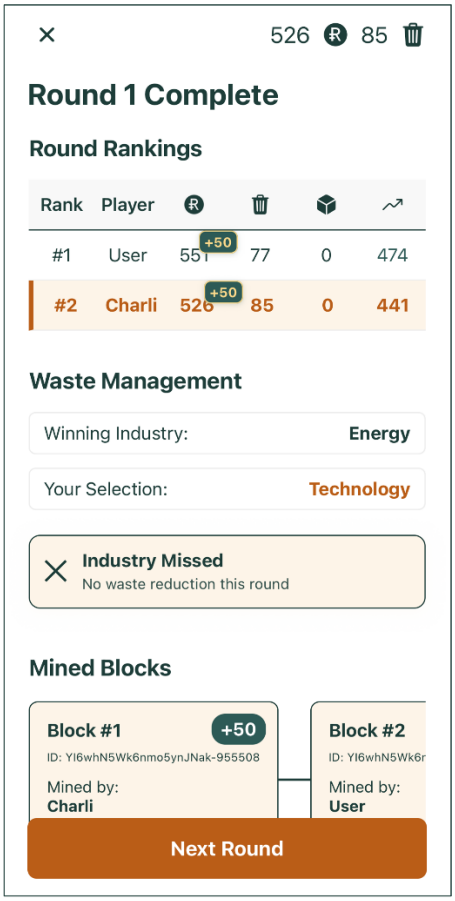

Abbildung8 : Bildschirm „Ende der Runde".

In der App überwacht der Spielkontext (*GameContextHybridRobust*) diese Ereignisse und aktualisiert die Benutzeroberfläche. Er wechselt von der aktiven Spielansicht zu Übersichtsbildschirmen, wie z. B.:

- *EndOfRoundScreen* zeigt die Gewinnerbranche, wer die Mining-Herausforderung gelöst hat (falls es eine gab) und wie sich das Inventar und die RockCoins der einzelnen Spieler verändert haben.
- *GameResultsScreen* (oder ein ähnlicher Abschlussbildschirm) zeigt die Gesamtwertung der Spieler, die gesammelten Statistiken aus den gespielten Runden und andere nützliche Indikatoren für Diskussionen oder Reflexionen.

Damit ist die Runde für die gesamte Gruppe beendet. Da alle wichtigen Daten in Firestore gespeichert sind, können die Spieler:

- zum Hauptbildschirm zurückkehren und als Gastgeber ein neues Spiel mit derselben oder einer anderen Gruppe starten oder
- das Spiel beenden und mit anderen Kursaktivitäten fortfahren.

Abgeschlossene Spiele und ihre Daten bleiben für die spätere Verwendung in WP5 (Bewertungen, Analysen oder Überprüfungen in zukünftigen Sitzungen) verfügbar, unabhängig davon, was die Spieler als Nächstes tun.

Zusammen schließt dieser Ablauf den Zyklus, der bei der Anmeldung begann: Er reicht von der Authentifizierung über die Spieleinrichtung bis hin zur aktiven Erfahrung und schließlich zu einem Abschluss mit klaren Ergebnissen. Dank der Kombination aus einem Echtzeit-Server, einer soliden Struktur in Firestore und gut gestalteten Übersichtsbildschirmen wird die technische Architektur zu einer praktischen, wiederholbaren und nützlichen Erfahrung in verschiedenen Bildungskontexten.

4. HOSTING, ZUGRIFF UND VERTRIEB

Aus betrieblicher Sicht wird das interaktive RockChain-Tool als gehosteter Cloud-Dienst plus mobiler App bereitgestellt. Schulungszentren müssen keine lokalen Server installieren oder warten, sondern benötigen lediglich einen Internetzugang und mindestens eine der wichtigsten Plattformen.

Dieser Abschnitt bietet einen prägnanten technischen Überblick darüber, wie das Tool gehostet wird und wie darauf zugegriffen werden kann. Schritt-für-Schritt-Anleitungen für Trainer zur Vorbereitung und Durchführung einer Sitzung finden Leser in den „Guideline Notes" (Leitfaden-Hinweisen) von WP4-A4, damit Benutzer das Tool problemlos verwenden können.

4.1. Backend-Hosting

Das Backend wird im Rahmen des RockChain-Projekts auf ***Google Cloud*** gehostet und besteht aus den in Abschnitt 2 beschriebenen Elementen:

- *Cloud Firestore*: speichert Spiele, Benutzer, Marktdaten und lernbezogene Datensätze.
- *Firebase Cloud Functions*: Implementierung der authentifizierten Vorgänge zum Erstellen und Verwalten von Spielen, Generieren von Mining-Problemen, Zuweisen von Belohnungen und Aktualisieren von Benutzerprofilen.
- *Node.js + Socket.IO-Server auf Cloud Run*: koordiniert Echtzeitrunden und fungiert als maßgeblicher Zeitnehmer und Schiedsrichter.

Alle Kerndaten werden in einer europäischen Region gespeichert, wodurch die Bereitstellung den Datenschutzanforderungen der EU entspricht und die Einhaltung der Vorschriften für Partner, die in verschiedenen Ländern tätig sind, vereinfacht wird. Da diese Dienste vollständig verwaltet werden, sind keine lokalen Datenbankserver oder eine benutzerdefinierte Infrastruktur in den Bildungszentren erforderlich: Wartung und Skalierung werden auf Cloud-Ebene durchgeführt.

4.2. Verteilung der mobilen App

Der RockChain-Client wird als **mobile Anwendung** für mindestens eine der beiden Hauptplattformen verteilt:

- Eine Android-Version (APK/AAB), die für die Installation über den Store oder die direkte Verteilung auf verwaltete Geräte geeignet ist.
- Eine iOS-Version (IPA über TestFlight oder App Store), abhängig von der mit jedem Partner vereinbarten Vertriebsstrategie.

Die Links und/oder QR-Codes finden Sie auch im RockChain-Webbereich unter: https://rockchain.eu/tool/

In der Praxis können Trainer, die einen Kurs vorbereiten, den RockChain-Webbereich besuchen, den QR-Code scannen oder dem Link für ihre Plattform folgen und die aktuelle Produktionsversion der App auf ihren Geräten oder auf Tablets der Einrichtung installieren.

4.3. Zugriffsablauf für Trainer und Lernende

Für Endnutzer erfolgt der Zugriff auf RockChain in einer einfachen, wiederholbaren Reihenfolge:

1. **Installieren Sie die App**
    - Lernende und Trainer installieren die RockChain-App auf ihren Android- oder iOS-Geräten, entweder auf persönlichen Smartphones/Tablets oder auf Geräten im Besitz der Einrichtung, auf denen die App möglicherweise vorinstalliert ist.
2. **Erstellen oder verwenden Sie ein Konto**
    - Bei der ersten Verwendung registrieren sie sich oder melden sich über Firebase Auth vom *LoginScreen* aus an, wie in Abschnitt 3.1 beschrieben.
    - Konten können über mehrere Sitzungen und Kurse hinweg wiederverwendet werden, sodass sich die Benutzer nicht jedes Mal neu registrieren müssen.
3. **An Spielsitzungen teilnehmen oder diese veranstalten**
    - Nach der Anmeldung gelangt jeder Spieler zum *GameScreen*, wo im Hintergrund ein persönlicher Spielcode für ihn vorbereitet wird.
    - Kleine Gruppen entscheiden, wessen Code verwendet werden soll: Ein Spieler fungiert als Gastgeber, indem er seinen Code teilt und das Spiel startet, und die anderen nehmen über die Eingabe „Join game“ (Spiel beitreten) teil.
    - Trainer können entweder die Lernenden ihre eigenen Spiele hosten lassen (eines pro Gruppe) oder selbst als Gastgeber fungieren, wenn sie eine Demo durchführen oder das Timing genauer kontrollieren möchten.

Aus Sicht des Trainers bedeutet dies, dass die Durchführung einer RockChain-Sitzung hauptsächlich aus folgenden Schritten besteht:

- Sicherstellen, dass alle Teilnehmer die App installiert haben und sich anmelden können.
- Die Lernenden in kleine Gruppen einteilen (jede mit einem Gastgeber, der einen Spielcode teilt).
- Überwachung des Fortschritts der Gruppen durch die Runden und Nutzung der Ergebnisbildschirme für die Nachbesprechung.

4.4. Anforderungen an Schulungszentren

Da RockChain auf Cloud-Hosting und mobiler Verteilung basiert, sind die Anforderungen an die Infrastruktur von Schulungszentren minimal:

- Netzwerk: ein WLAN-Netzwerk im Klassenzimmer mit Internetzugang, das gleichzeitige Verbindungen von der Anzahl der in einer Sitzung verwendeten Geräte unterstützen kann.
- Geräte: Android- oder iOS-Smartphones/Tablets, die die Mindestanforderungen an das Betriebssystem für die eingesetzte Version erfüllen (wie in der RockChain-Dokumentation angegeben).
- Keine lokale Serverinstallation: Die Zentren müssen keine zusätzlichen Server bereitstellen; die gesamte Spielkoordination, Speicherung und Authentifizierung wird in der Cloud abgewickelt.

Diese Kombination aus Google Cloud-Backend, mobilen Apps für Android und iOS und Zugriff über den RockChain-Webbereich ermöglicht die Nutzung des interaktiven RockChain-Tools in verschiedenen Ländern und Institutionen ohne komplexe lokale Einrichtung, während eine einzige, zentralisierte Bereitstellung beibehalten wird, die von den Projektpartnern gewartet und aktualisiert werden kann.

5. ÜBERWACHUNG, RESILIENZ UND WARTUNG

Ein letzter Aspekt dieses Dokuments ist die Sicherstellung, dass das interaktive RockChain-Tool im Laufe der Zeit überwacht, debuggt und gewartet werden kann, insbesondere während Pilotprojekten in realen Klassenzimmern. Das Ziel ist nicht der Aufbau eines umfangreichen Observability-Stacks, sondern die Bereitstellung ausreichender Transparenz und Robustheit, damit die Partner verstehen können, was geschieht, und das System stabil halten können.

5.1. Beobachtbarkeit und Protokollierung

Sowohl der Client als auch der Server verfügen über leichtgewichtige Beobachtungsmechanismen, die bei der Entwicklung, der Pilotbereitstellung und der Analyse von Vorfällen helfen.

Auf der Client-Seite protokolliert *GameContextHybridRobust* wichtige technische Signale an die Konsole, wie z. B.:

- Taktabweichung zwischen dem lokalen Gerät und dem autoritativen Server.
- Resynchronisierungsversuche (z. B. nach einer erneuten Verbindung).
- Erkennung doppelter oder fehlerhafter Ereignisse.

Diese Protokolle werden hauptsächlich während der Entwicklung und Pilotphase verwendet, wenn Entwickler oder technische Partner die App mit angeschlossenen Debugging-Tools ausführen. Sie erleichtern die Reproduktion und Diagnose von Problemen im Zusammenhang mit Timing, Navigation und Zustandskonsistenz.

Auf der Serverseite schreibt ein *logMetric*-Dienstprogramm strukturierte JSON-Ereignisse in Cloud Logging. Typische Ereignisse sind:

- Neue Verbindungen und Trennungen.
- Rundenbeginn und Rundenende.
- Resynchronisierungsvorgänge.
- Fehler oder unerwartete Bedingungen.

Mit diesen Protokollen können technische Mitarbeiter:

- Sehen, wie viele Spiele in jedem Zeitraum gespielt wurden.
- Fehlermuster oder Leistungsengpässe identifizieren (z. B. wiederholte Neusynchronisierungen für bestimmte Spiele).
- die Fehlerbehebung unterstützen, wenn Partner Vorfälle melden, indem sie Benutzerberichte mit konkreten Ereignissen in den Protokollen korrelieren.

Der allgemeine Ansatz ist bewusst schlank gehalten: Es gibt kein komplexes Überwachungs-Dashboard, aber es stehen genügend strukturierte Informationen zur Verfügung, um grundlegende Wartungs-, Optimierungs- und Fehlerbehebungsmaßnahmen zu unterstützen.

5.2. Ausfallsicherheit und Fehlerbehandlung

Da RockChain ein Echtzeit-Multiplayer-Tool ist, das über das WLAN im Klassenzimmer genutzt wird, muss es mit zeitweiligen Verbindungsunterbrechungen und vorübergehenden Ausfällen zurechtkommen, ohne das Spielerlebnis zu beeinträchtigen. Daher sind mehrere Ausfallsicherheitsstrategien in die Architektur integriert:

- Automatische Wiederherstellung der Socket-Verbindung: Der Socket.IO-Adapter des Clients versucht, die Verbindung automatisch wiederherzustellen, wenn das Netzwerk vorübergehend ausfällt. Nach einer erfolgreichen Wiederherstellung der Verbindung sendet er das Authentifizierungstoken und die join_game-Informationen erneut, damit der Server den Socket wieder mit dem richtigen Spiel und Benutzer verbinden kann.
- Idempotente Operationen auf dem Server: Serverseitige Operationen basieren auf Identifikatoren wie *round-ID*, Versionsnummern und, falls erforderlich, Operations-IDs. Dadurch ist es möglich, wiederholte oder verspätete Nachrichten zu erkennen und zu ignorieren, anstatt sie zweimal anzuwenden, wodurch doppelte Belohnungen oder inkonsistente Zustände bei der Wiederholung von Nachrichten vermieden werden.
- Rundenbezogene Sperren für kritische Phasen: In sensiblen Phasen wie Berechnungen am Ende einer Runde verwendet der Server rundenbezogene Sperren, sodass für jedes Spiel und jede Runde jeweils nur ein Berechnungsablauf ausgeführt wird. Dies verhindert Race Conditions, wenn gegen Ende des Countdowns viele Aktionen eintreffen.
- Reibungsloser Umgang mit Desynchronisation: Wenn ein Client offline war, in den Hintergrund verschoben wurde oder eine kurze Unterbrechung der Verbindung hatte, ermöglicht die Kombination aus *GameContextHybridRobust* und dem autoritativen Server dem Gerät, sich bei seiner Rückkehr mit dem aktuellen Status neu zu synchronisieren. Anstatt sich auf das zuvor gerenderte Bild zu verlassen, kann der Client seine Ansicht der aktiven Runde, der Bestände und der Ranglisten aktualisieren.

In der Praxis bedeuten diese Mechanismen, dass kurzfristige Netzwerkprobleme oder vorübergehende Fehler eine Spielsitzung nicht ungültig machen sollten. Trainer können in der Regel ohne tiefgreifende technische Eingriffe mit der Aktivität fortfahren, und

Lernende, die kurzzeitig die Verbindung verlieren, können wieder beitreten und in einem konsistenten Zustand weiterspielen.

5.3. Wartung und zukünftige Entwicklung

Aus Sicht der Wartung legt WP4.A5 einige einfache, aber wichtige Grundsätze fest, die als Leitlinien für die zukünftige Weiterentwicklung des Tools dienen sollen:

- Behalten Sie die Echtzeitlogik auf dem autoritativen Server zentralisiert bei: Alle zeitkritischen Entscheidungen (wer zuerst abgebaut hat, wann die Runde endet, wie Belohnungen vergeben werden) verbleiben auf dem Server. Der Client nutzt *authoritativeState*, anstatt zu versuchen, eigene alternative Regeln zu implementieren. Dies verringert das Risiko von Abweichungen zwischen verschiedenen Client-Versionen.
- Behandeln Sie Cloud-Funktionen als einzigen Einstiegspunkt für Backend-Operationen: Jede neue Operation, die persistente Daten ändert, sollte als Cloud-Funktion implementiert, in functions/index.js registriert und immer über authentifizierte Wrapper auf dem Client aufgerufen werden. Dadurch bleiben Sicherheitsprüfungen und Geschäftslogik auf dem Server und es ist einfacher, Datenflüsse zu verstehen.
- Änderungen am Dokumentenschema und Migrationen: Wenn sich die Dokumentenschemata von Firestore weiterentwickeln (z. B. durch Hinzufügen neuer Felder zu *games/{game-ID}* oder *users/{user-ID}.games[game-ID]*), sollten diese Änderungen ausdrücklich dokumentiert werden. Wenn vorhandene Daten angepasst werden müssen, können Migrationsskripte oder Dienstprogramme unter *functions/src* abgelegt werden, damit Partner einen klaren Weg zur Aktualisierung der Produktionsdaten haben.

Durch Befolgen dieser Vorgehensweisen können technische Partner neue Funktionen hinzufügen – wie zusätzliche Bildschirme, weitere Indikatoren, erweiterte Protokollierung oder neue Spielmodi –, ohne die Stabilität des bestehenden Spiels zu beeinträchtigen. RockChain bleibt über die ursprüngliche Projektlaufzeit hinaus wartbar und erweiterbar, während das während der Pilotprojekte in WP5 validierte Kernverhalten beibehalten wird.

6. SCHLUSSFOLGERUNGEN

WP4.A5 hat eine produktionsreife Version des interaktiven RockChain-Tools geliefert und damit die in früheren Aufgaben von WP4 durchgeführten Arbeiten in den Bereichen Design, Implementierung und Bereitstellung konsolidiert. Das resultierende System vereint einen modernen, mehrsprachigen mobilen Client, ein Firebase-basiertes Backend und einen autoritativen Echtzeit-Server auf Cloud Run, die alle auf einer skalierbaren Cloud-Infrastruktur bereitgestellt und in Android- und iOS-Builds gepackt sind, die auf Standardgeräten installiert werden können, die in der beruflichen Bildung und Erwachsenenbildung verwendet werden.

Durch die explizite Dokumentation der Systemarchitektur, der Laufzeitabläufe, des Produktions- und Bereitstellungsworkflows sowie der Hosting-, Zugriffs- und Wartungsverfahren stellt diese Aktivität sicher, dass RockChain kein einmaliger Prototyp ist, sondern eine reproduzierbare und wartbare Ressource. Technische Mitarbeiter haben eine klare Referenz darüber, wie das Tool strukturiert ist und wie es aktualisiert werden kann, während Ausbilder über eine stabile, installierbare Anwendung verfügen, die mit minimalem lokalen Aufwand in reale Kurse integriert werden kann.

Auf diese Weise wird das interaktive RockChain-Tool zum praktischen Kernstück des E-Learning-Angebots des Projekts: Es setzt die pädagogische und curriculare Arbeit aus früheren Arbeitspaketen in eine konkrete, spielbare Erfahrung um, die in Klassenzimmern in allen Partnerländern eingesetzt werden kann. Das Tool ist nun bereit, Pilotaktivitäten in WP5 zu unterstützen, und bietet eine solide Grundlage für eine mögliche zukünftige Nutzung und Erweiterung über die Laufzeit des Projekts hinaus.