WP4-A1. Producția bazei de date pentru instrumentul de e-Learning.

Conținut

1. INTRODUCERE

Acest raport rezumă concluziile finale ale WP4. A1: Producția unei baze de date pentru instrumentul de e-Learning. În cadrul general al proiectului RockChain, această activitate se concentrează pe proiectarea și realizarea stratului de date back-end care susține jocul serios, astfel încât sesiunile multiplayer, dinamica pieței și evenimentele de minerit să poată rula în timp real pe o infrastructură stabilă și pregătită pentru producție.

Sarcina crucială a WP4. A1 este să definească și să dezvolte o bază de date și o arhitectură back-end care să poată susține gestionarea consecventă a profilurilor utilizatorilor, sesiunilor de joc, rundelor, recompenselor și deciziilor din joc, conform scenariilor de învățare definite în proiect. Implementarea actuală a unei astfel de arhitecturi se bazează pe *Firebase Firestore* ca bază de date centrală, un set de funcții Firebase care încapsulează operații sensibile de scriere și logica jocului, precum și pe un *server dedicat* WebSocket implementat pe *Google Cloud Run*, care gestionează comunicarea în timp real, cu latență scăzută, între jucători.

Acest raport are un focus deliberat îngust și tehnic: documentează modul în care stratul de date RockChain este structurat și implementat în producție. Acesta descrie principalele componente ale arhitecturii și găzduirii acestora, structura colecțiilor și documentelor cheie Firestore și cele mai relevante fluxuri de date care creează și actualizează aceste informații în timpul unui joc. De asemenea, rezumă considerațiile esențiale de producție legate de standardele de securitate, performanță și întreținerea de bază a bazelor de date.

Cu această descriere concisă a stratului de date, WP4. A1 oferă o referință practică pe care dezvoltatorii și partenerii tehnici o pot folosi pentru a implementa, opera sau extinde infrastructura RockChain. Arhitectura rezultată va forma baza pe care atât instrumentul de e-learning front-end, cât și activitățile pilot ale pachetelor de lucru ulterioare pot fi construite în mod fiabil.

2. PREZENTARE GENERALĂ ARHITECTURA DATELOR

2.1. Concepte generale

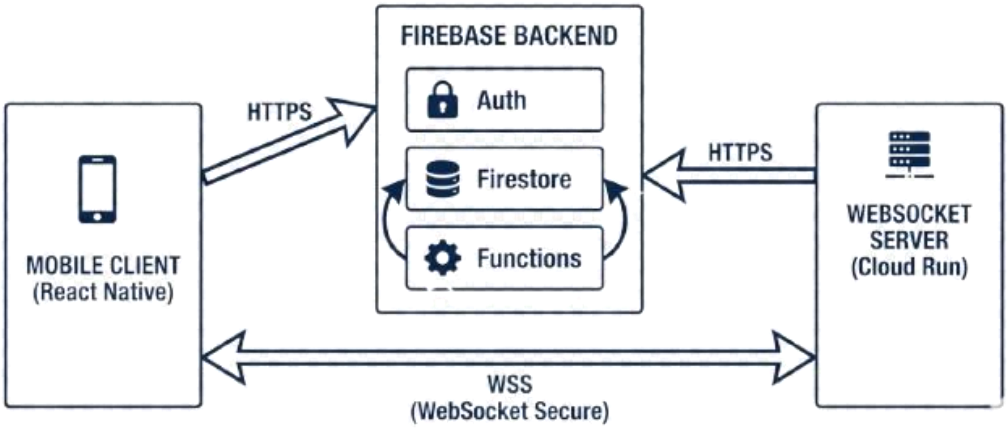
Instrumentul de e-learning RockChain rulează în producție pe o arhitectură în trei straturi, centrată pe un singur proiect Firebase și un serviciu dedicat în timp real:

- Clientul **mobil React Native** (Android/iOS) oferă interfața cu utilizatorul și leagă cursanții de joc.
- **Firebase backend** cu Firestore, Funcții, Autentificare - stochează toate datele persistente și încapsulează operațiunile sensibile.
- Serverul **WebSocket** implementat pe Google Cloud Run gestionează comunicarea în timp real între jucători în timpul unui joc, cu latență scăzută.

La nivel general, arhitectura urmează un model hibrid:

- Firestore servește ca o bază de date autoritară care stochează jocuri, utilizatori, blocuri, produse, piețe și recompense.
- Firebase Functions acționează ca o poartă controlată pentru scrieri critice, cum ar fi crearea de jocuri, unirea la jocuri și actualizarea informațiilor despre runde și recompense, aplicând reguli de validare și business pe partea de server.
- Serverul WebSocket păstrează starea în memorie a fiecărui joc activ și transmite evenimente în timp real precum startul rundei, date de piață, probleme de minerit și rezultate către toți clienții conectați, stocând rezultatele relevante doar în puncte bine definite din Firestore.

Deoarece toate componentele împărtășesc aceeași identitate Firebase și același context de securitate, autentificarea, autorizarea și accesul la date sunt consistente pe întregul stack. Această arhitectură poate fi replicată pentru dezvoltare și testare cu un alt proiect Firebase și implementare a WebSockets, păstrând structura exactă, dar folosind acreditări și URL-uri non-producție.



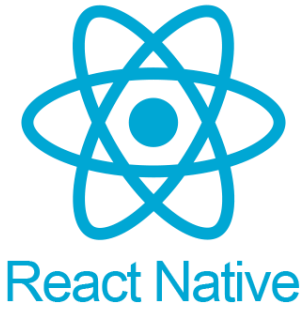
Figură 1: Diagramă arhitecturală

Această separare a responsabilităților permite RockChain să combine o stocare durabilă și bine structurată a ceea ce s-a întâmplat în fiecare sesiune cu actualizări rapide, bazate pe evenimente, ale evenimentelor care se întâmplă în acel moment al jocului.

2.2. Componente principale

2.2.1. Client mobil React Native

Aplicația mobilă React Native este **singurul punct de intrare pentru utilizatori**. Scopul este să prezinte interfața jocului și să răspundă la starea backend-ului, în timp ce aplicația în sine nu gestionează persistența.



Figură 2: React Native

În producție, clientul mobil:

- Se autentifică cu autentificarea Firebase și primește o identitate de utilizator (*userId*), care este folosită constant pe tot parcursul backend-ului.
- Se alătură jocurilor prin apeluri dedicate de funcții Firebase, cum ar fi crearea sau alăturarea unui joc, care servesc pentru a valida cererea și apoi a actualiza Firestore.
- Deschide o conexiune WebSocket către punctul final Cloud Run odată ce utilizatorul este într-un joc și trimite *gameId-ul*, userId-ul autentificat și informațiile de bază ale jucătorului, astfel încât să poată plasa socket-ul în camera de joc corespunzătoare.
- Abonează-te la evenimentele backend (fie de la ascultători Fistore, fie de la mesaje WebSocket) și mapează-le pe starea aplicației locale, care este apoi redată pe ecran.

Clientul nu scrie direct pe Firestore pentru operațiuni critice. În schimb:

- Firebase Functions ar trebui invocate ori de câte ori este necesară o modificare durabilă a modelului de date, exemple includ actualizarea stării jocului și scrierea recompenselor.

- Ele emit evenimente WebSocket, cum ar fi *"gata de joc", "cumpără produs", "trimite răspuns de minerit",* care sunt procesate pe server și persistate acolo unde este cazul.

Acest design păstrează aplicația mobilă ca un strat subțire și reactiv, axat pe interacțiunea utilizatorului, toate deciziile autoritare fiind luate în backend despre starea jocului.

2.2.2. Backend Firebase: Firestore și funcții

Backend-ul Firebase constă din Cloud Firestore cu funcții Firebase, care împreună oferă un strat de date securizat și scalabil.

Figură 3: Baza de foc

Cloud Firestore (baze de date și stocare autoritară)

Firestore stochează orice informație care trebuie să supraviețuiască unei conexiuni WebSocket și să fie disponibilă pentru analiză sau recuperare. Următoarele se numără printre cele stocate în RockChain:

- **Starea** jocului în colecția *de jocuri* : Fiecare document reprezintă un meci. Include codul jocului, lista jucătorilor, runda curentă și statusul, industria câștigătoare pe rundă, cronometrele și recompensele atribuite.
- **Starea** utilizatorului în *colecția utilizatorului: fiecare document stochează profilul utilizatorului - numele afișat, avatarul - și datele pe joc: rockCoins, deșeuri acumulate, industrie selectată, inventar*
- **Minerit și date "inspirate de blockchain"** în *blockchain/{gameId}/blocuri: pentru fiecare joc, această subcolecție stochează problemele de minerit, răspunsurile primite și câștigătorul final al fiecărui bloc.*
- **Catalogul de produse** din *colecția de produse: șabloane pentru produsele de marmură disponibile (tip, tip de marmură și descriere), folosite pentru a construi ofertele de piață pe rundă.*
- *Piață*: configurația pieței pe meci. Output-ul este o viziune organizată a parametrilor pieței, pe runde și tip de produs. Acest lucru este util pentru reconstruirea sau afișarea pieței în contexte diferite de listele plate în timp real.

În acest caz, Firestore este optimizat pentru citiri scurte și frecvente de către un client mobil – starea curentă a jocului, inventarul actualizat etc. – și pentru scrieri controlate din codul backend care asigură integritatea și consistența modelului jocului.

Firebase Functions (logică de business și scrieri controlate)

Firebase Functions centralizează principalele reguli de afaceri și modificările sensibile ale Firestore. Clientul nu scrie direct în documente critice precum jocuri sau date de joc ale utilizatorului, ci numește în schimb Funcții care:

- **Creează joc**, *creeazăJoc*: creează un document în jocuri cu codul jocului, gazda, status, numărul maxim de jucători și alte câmpuri standard
- Permite unui utilizator **să se alăture unui joc**, de exemplu: *joinGame* verifică dacă un joc există și poate fi alăturat; adaugă utilizatorul pe lista jucătorilor; actualizează *playerCount* și creează/actualizează valorile lor inițiale: *rockCoins*, *deșeuri*, *status*, *timestamps*....
- **Actualizează** câmpurile sensibile din joc: de exemplu, informații despre rundă, rezultate agregate sau recompense declanșate de evenimentele clientului sau ca rezultat al proceselor backend la finalul unei runde.

```
exports.createGame = onCall({
  memory: '256MiB',
  timeoutSeconds: 60,
  region: 'us-central1',
  minInstances: 0,
  maxInstances: 10,
  concurrency: 80,
  retry: false,
  ingressSettings: 'ALLOW_ALL'
}, async (request) => {
  const userId = request.auth?.uid;
  const gameCode = generateGameCode();

  if (!userId) {…
  }

  const gameData = {…
  };

  try {…
  } catch (error) {
    console.error('❌ Error al crear el juego:', error);
    throw new Error(`Failed to create game: ${error.message}`);
  }
});

function generateGameCode() {
  const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
  let code = '';
  for (let i = 0; i < 6; i++) {
    code += characters.charAt(Math.floor(Math.random() * characters.length));
  }
  return code;
}
```

Figură 4: funcție createGame

Prin direcționarea acestor operațiuni prin funcții, proiectul:

- Asigură validarea și regulile consecvente în fiecare sesiune.
- Menține regulile de securitate Firestore mai simple, deoarece deciziile complexe rămân în cod backend în loc de seturi mari de reguli.
- Reduce riscul de scrieri malițioase sau inconsistente din partea clienților neîncrezători.

În practică, calea autoritară este întotdeauna: *Client → Firebase Function / logică backend → Firestore*.

2.2.3. Serverul WebSocket pe Google Cloud Run (strat în timp real)

Serverul WebSocket este un serviciu construit cu Node.js și rulează ca un container pe Google Cloud Run. Sarcina sa principală este să mențină **toți jucătorii conectați** pe parcursul jocului, cu o comunicare rapidă, bidirecțională, fără întreruperi.

Ce face exact? Poftim:

- Asigură că **comunicarea** în fiecare joc curge **în timp real**. Aceasta include notificări precum cine se alătură, cine este pregătit, când începe o rundă, cum merge piața, noile probleme legate de minerit, rezultate și când se termină fiecare rundă sau întregul joc.
- Ține evidența fiecărui joc activ în memorie. Acest lucru îi permite să coordoneze ceea ce se întâmplă fără a suprasolicita Firestore. De exemplu:
  - Cine participă și dacă sunt pregătiți.
  - În ce rundă se află și cât timp durează până se termină.
  - Cum evoluează stocurile și piața în acel moment.
  - Ce probleme de minerit sunt active și ce răspunsuri au fost trimise.
- De asemenea, salvează rezultatele în Firestore, dar doar în momente cheie: când cineva câștigă un bloc, la finalul unei runde sau când sunt atribuite recompense. Uneori face asta direct, alteori o transmite funcțiilor Firebase.

În plus, deoarece rulează pe Cloud Run, serviciul se scalează singur în funcție de câte jocuri sunt active. Și totul este conectat printr-un endpoint WebSocket securizat (WSS), în cadrul aceluiași proiect Firebase unde se află Firestore și Functions.

În rezumat:

- Serverul **WebSocket îți spune ce se întâmplă în timp** real în joc.
- **Firestore și funcțiile stochează istoricul oficial**: ce s-a întâmplat, cum decurge jocul și în ce stare se află jucătorii.

Datorită acestei divizii, RockChain poate oferi jocuri multiplayer care se simt agile și fluide, fără a sacrifica controlul și înregistrarea a ceea ce s-a întâmplat.

3. MODELUL DE DATE ȘI DESIGNUL BAZEI DE DATE

În producție, stratul de date al RockChain se învârte în jurul câtorva colecții cheie din Firestore. Fiecare are o funcție bine definită: sunt proiectate să fie citite constant de aplicația mobilă, în timp ce scrierile sunt efectuate controlat prin funcții Firebase și procese backend.

Împreună, aceste colecții stochează:

- Statutul general al fiecărui joc.
- Statutul individual al fiecărui jucător în cadrul unui joc.
- Istoria evenimentelor miniere.
- Configurația pieței folosită în fiecare rundă.

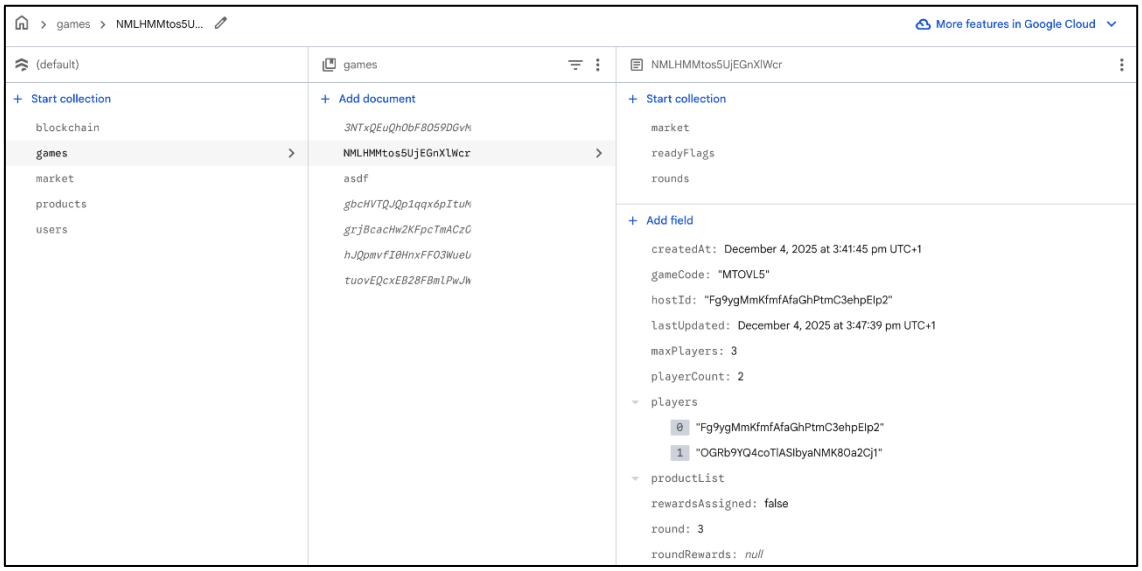
3.1. Colecții și documente Firestore

3.1.1. Jocuri – Stare globală a jocului

Colecția de jocuri stochează câte un document pentru fiecare joc. Fiecare *game*/{*gameId*} conține date care descriu ce se întâmplă în acel joc în general.

Acestea sunt câteva dintre domeniile cheie:

- *gameCode*: codul public pe care jucătorii îl folosesc pentru a se alătura.
- *player*: lista *utilizatorId-urilor* din joc, împreună cu date precum câți jucători sunt (*playerCount*) și maximul permis (*maxPlayers*).
- status: în ce etapă se află jocul (poate fi *așteptare*, *început*, in_progress, *rundăSfârșit*, *așteptândpenRunda următoare* sau terminat).
- runda: numărul rundei curente.
- *câștigătorIndustrie*: industrie care a câștigat ultima rundă completă.
- *roundTime*: marcaj temporal care indică când se termină runda curentă; folosit ca bază pentru afișarea contoarelor și efectuarea tranzițiilor.
- *RewardsAssigned*: un boolean care indică dacă recompensele pentru ultima rundă au fost deja scrise.
- *RoundRewards*: rezumat al recompenselor fiecărui jucător din runda anterioară (pe industrie, eliminarea deșeurilor, minerit etc.).



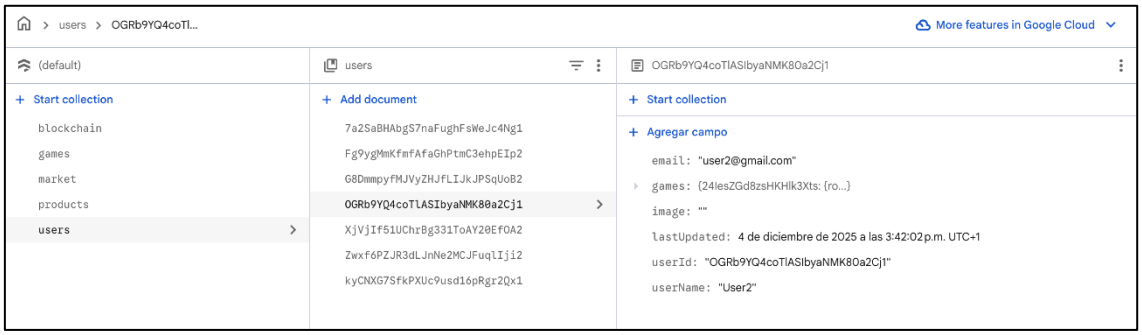
Figură 5: colectarea datelor despre jocuri

Acest document este principala poartă atunci când se încarcă sau se reia un joc. Prin simpla citire *a jocurilor*/{*gameId*}, sistemul poate ști cine joacă, în ce rundă se află și dacă recompensele au fost deja atribuite.

3.1.2. Utilizatori – Date de profil și pe meci

Colecția utilizatorului stochează un document per utilizator, identificat prin userId-ul lor Firebase. Fiecare document *utilizator*/{*userId*} are două straturi de informații:

- **Profilul** utilizatorului, cu câmpuri precum *userName*, un avatar opțional (*imageUrl*) și *lastUpdated*, care servește ca timestamp pentru auditurile de bază.
- **Date de joc**, organizate într-o hartă de joc, unde fiecare intrare este asociată cu un *gameId*. Pentru fiecare joc la care participă utilizatorul, sunt stocate lucruri precum următoarele:
    - Monede de joc: *rockCoins* și deșeuri acumulate.
    - Industria pe care au ales-o (*a selectat Industria*) pentru a câștiga recompense în cadrul modelului economiei circulare.
    - Inventarul lor de produse (*produse*), care arată ce au cumpărat pe piață.
    - Date suplimentare precum *gameCode*, status, când s-au alăturat (*joinedAt*) și coada de minerit (*miningQueue*).
    - Informații despre ultima problemă de minerit cu care au interacționat (*lastMiningProblem*), inclusiv *blockId-ul*, detaliile problemei și un timestamp.



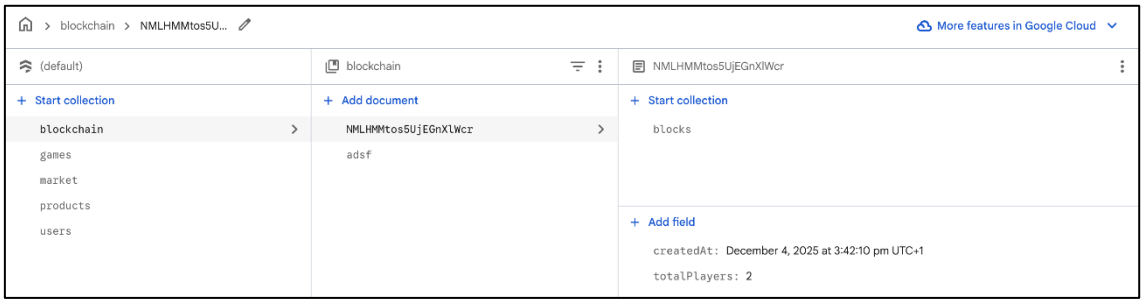
Figură 6: colectarea datelor utilizatorilor

Concentrarea tuturor acestor informații într-un singur document pentru fiecare utilizator face lucrurile mult mai ușoare: de exemplu, poți afla instantaneu situația actuală a jucătorului într-un anumit joc, fără a fi nevoie să faci mai multe interogări sau combinații complicate.

3.1.3. blockchain/{gameId}/blocuri – probleme de minerit și blocuri câștigătoare

Pentru fiecare joc, subcolecția *blockchain/{gameId}/blocuri* ține evidența evenimentelor de minerit, folosind o structură inspirată de blockchain. Fiecare *document blocks/{blockId}* reprezintă un bloc cu următoarele informații:

- Problema de minerit activ (*activeProblem*), care include:
  - *problemId*, care corespunde *blocId-ului*.
  - O problemă simplă de matematică (*mathProblem*) împreună cu soluția sa corectă.
  - Data creării (*createdAt*) și detaliile tranzacției asociate (cum ar fi produsul, tipul operațiunii și utilizatorul implicat).
- Răspunsuri primite (răspunsuri) în timp ce problema este activă.
- Numărul rotund (*rundă*) și un *createdAt* care indică când a fost creat blocul.
- Un obiect câștigător care se umple odată ce câștigătorul cursei miniere este decis. Include: *userId*, *username*, răspunsul dat, dacă a fost corect, timpul necesar pentru a răspunde (*elapsedTime*) și când a răspuns (*responseTime*), ceea ce este util pentru auditare.
- Un indicator *isActive* care semnalează dacă blocul este încă deschis sau dacă există deja un câștigător.



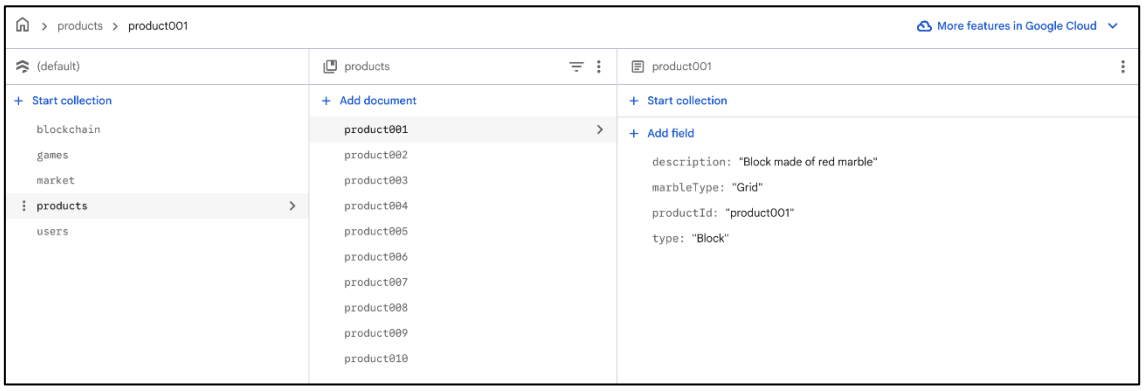
Figură 7: colectarea datelor blockchain

Această structură permite RockChain să păstreze o evidență clară și verificabilă a problemelor generate, a modului în care jucătorii au reacționat și a modului în care au fost atribuite recompensele legate de minerit.

4.1.4. Produse – Catalogul Global de Produse

Colecția de produse definește catalogul global al produselor din bilă disponibile în joc. Fiecare document funcționează ca un șablon stabil, cu câmpuri precum:

- *productId* (de exemplu: "product001")
- *tip* ("Bloc" sau "Placă")
- *marbleType* ("Roșu", "Alb", "Gri", "Negru", "Crem")
- *calitate* ("A", "B", "C")
- Descriere și alte date statice



Figură 8: colectarea datelor de produs

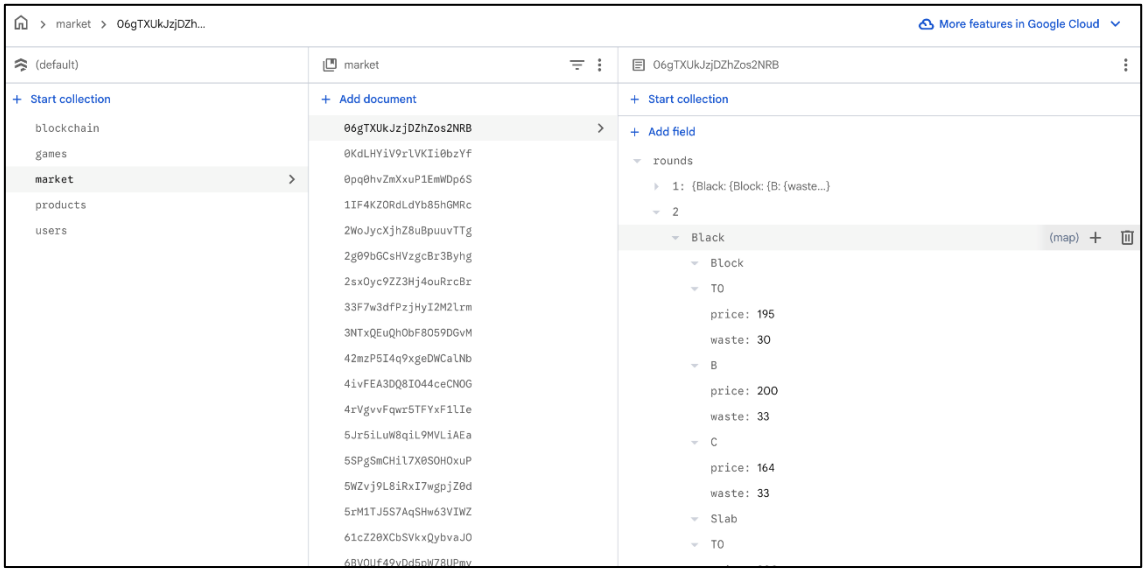
Aceste șabloane sunt folosite de logica backend pentru a asambla lista specifică de produse afișate pe piață pentru fiecare rundă. Ele servesc ca bază pentru generarea prețurilor și definirea parametrilor legați de deșeuri.

4.1.5. Piața – Structura pieței pe runde

Pentru fiecare joc, colecția *market*/{*gameId*} stochează o vizualizare structurată a pieței. Această perspectivă este folosită în principal pentru a reconstrui modul în care piața a fost configurată în diferite runde sau pentru analize ulterioare.

Câmpul principal este runde: o hartă imbricată cu structura *rounds[round][marbleType][type][quality] = { preț, risipă }*, unde:

- *Rotund* este numărul rotund,
- *marbleType* este unul dintre tipurile de marmură configurate,
- *tipul* poate fi "Block" sau "Slab",
- *calitatea* este "A", "B" sau "C", fiecare cu valorile sale de preț și nivel de deșeuri.



Figură 9: colectarea datelor de piață

Această structură funcționează ca un fel de rezumat sau o viziune istorică a pieței. Completează listele de produse în timp real generate și trimise prin WebSocket în timpul jocului. Este deosebit de utilă pentru a analiza cum a fost piața în fiecare rundă, fără a fi nevoie să treci prin fiecare produs individual.

3.2. Relațiile dintre entități

Deși Firestore este o bază de date NoSQL orientată pe documente și nu gestionează relațiile așa cum ar face-o o bază de date relațională, modelul RockChain definește relații logice clare între colecțiile sale.

3.2.1. Joc – Jucători

Fiecare *document games/{gameId}* include jucătorii săi în array-ul unui jucător, care conține *userId-urile*.

În același timp, fiecare document *utilizator/{userId}* stochează, în harta *jocului*[*gameId*], starea specifică a acelui utilizator în acel joc. În practică, aceasta reprezintă o relație mulți-la-mulți între jocuri și utilizatori, implementată prin referințe încrucișate.

3.2.2. Joc – Blocuri de minerit

Fiecare joc are propria subcolecție: *blockchain*/{*gameId*}/*blocuri*.

Toate blocurile sunt legate de un singur *gameId*, iar fiecare include numărul rotund căruia îi aparține. Este o relație unu-la-mulți: un joc poate avea mai multe blocuri, dar fiecare bloc aparține unui singur joc.

3.2.3. Joc – Piață

Fiecare joc are un document *unic de piață*/{*gameId*}, unde piața este organizată pe runde. Deși Firestore nu impune relații, aceasta este logic o relație unu-la-unu între *jocuri*/{*gameId*} și *piață*/{*gameId*}.

3.2.4. Produse – Piață / Stocuri

Colecția globală de produse definește șabloanele de bază pentru produse.

Atât câmpul de runde din *market*/{*gameId*}, cât și array-urile de produse din *games*/{*gameId*} sunt construite din valorile și parametrii definiți în *acele productId-uri*. Este o relație unu-la-mulți: un singur produs din catalog poate apărea pe piețe diferite și în mai multe inventaruri ale jucătorilor.

3.2.5. Cum sunt reconstruite starea unui joc și a unui jucător?

Backend-ul urmează de obicei acest tipar:

- Citește *jocuri*/{*gameId*} pentru a afla starea generală a jocului.
- Citește *users*/{*userId*} și verifică *game*[*gameId*] pentru a vedea cum se descurcă acel jucător.
- Citește *market*/{*gameId*} dacă ai nevoie de o imagine completă a prețurilor și risipei pe rundă.
- Citește *blockchain*/{*gameId*}/*blocuri* (și filtrează după rundă dacă este necesar) pentru a analiza istoricul mineritului.

Modelul aplică o ușoară denormalizare pentru a reduce apelurile și a simplifica interogările clientului, fără a pierde trasabilitatea de care partenerii de proiect au nevoie pentru a analiza datele ulterior.

4. FLUXURI CHEIE DE DATE ÎN PRODUCȚIE

Dincolo de modelul static de date, RockChain funcționează datorită unei serii de fluxuri recurente de date care creează și actualizează documente în Firestore în timpul unui joc. Aceste fluxuri conectează clientul mobil, funcțiile Firebase, serverul WebSocket și colecțiile pe care le-am descris anterior. Împreună, asigură că toți jucătorii văd aceeași stare a jocului, în timp real și cu trasabilitate.

4.1. Crearea și aderarea la joc

Crearea jocurilor și integrarea jucătorilor depind de funcțiile Firebase care fac primele scrieri în colecțiile jocurilor și ale utilizatorilor. Acest lucru asigură că fiecare joc începe cu o stare consecventă și că jucătorii sunt înregistrați corect atât în documentul global al jocului, cât și în profilul lor de utilizator.

4.1.1. Crearea jocului

Când un utilizator decide să găzduiască un joc nou:

- După autentificare, clientul apelează o funcție Firebase (*createGame*).
- Funcția creează un document nou în *games*/{*gameId*}.
- În *users*/{*userId*}, funcția creează sau actualizează intrarea *games*[*gameId*] cu valorile de start.

Acest proces asigură că atât starea generală a jocului, cât și starea individuală a jucătorului gazdă sunt bine definite încă de la început.

4.1.2. Alăturarea unui joc existent

Când un jucător se alătură unui meci existent:

- Jucătorul furnizează *gameCode-ul* în client.
- Clientul apelează o funcție Firebase (*joinGame*), care rezolvă *gameCode* într-un *gameId*.
- Funcția verifică dacă jocul este ușor de alăturat și apoi:
    - Adaugă jucătorul la *games*/{*gameId*}/*players* și crește *playerCount*.
    - *Creează sau actualizează* utilizatorii/{userId}/games[*gameId*] cu datele lor inițiale din joc (monede, deșeuri, status, timestamp-uri).

```
exports.joinGame = onCall({
  maxInstances: 10,
  memory: '256MiB',
}, async (request) => {
  console.log('🎮 joinGame iniciado');
  console.log('📦 Datos recibidos:', JSON.stringify(request.data, null, 2));

  // Extraer datos de la petición
  const { gameId } = request.data || {};
  const userId = request.auth?.uid;

  console.log(`🎯 Intentando unir usuario ${userId} al juego ${gameId}`);

  // Validaciones básicas
  if (!userId) {
  }

  if (!gameId) {
  }

  try {
  } catch (error) {
    console.error('❌ Error en joinGame:', error);

    // Mensajes de error específicos
    if (error.message.includes('Game not found')) {
      throw new Error('Game not found. Please check the code and try again.');
    } else if (error.message.includes('Game is full') || error.message.includes('No caben más jugadores')) {
      throw new Error('No caben más jugadores en esta sala. La sala está llena.');
    } else if (error.message.includes('not accepting new players')) {
      throw new Error('Game has already started. Please try another game.');
    } else if (error.message.includes('User not found')) {
      throw new Error('User not found. Please try logging in again.');
    } else {
      console.error('❌ Error no manejado:', error);
      throw new Error('Failed to join the game. Please try again.');
    }
  }
});
```

Figură 10: joinFuncție de joc

Acest lucru evită scrierile directe ale clienților în colecția de jocuri și asigură că toți jucătorii sunt înregistrați sub aceleași reguli.

4.1.3. Conectarea la stratul în timp real

Odată ce documentele Firestore există:

- Fiecare client deschide o conexiune WebSocket către punctul final Cloud Run.
- Clientul trimite *gameId*, *userId* și date de profil de bază.
- Serverul WebSocket înregistrează socket-ul în intrarea *corespunzătoare gameRooms* și începe să urmărească utilizatorul în starea sa în memorie.

În acest moment, atât starea persistentă (Firestore), cât și starea în timp real (hărțile în memorie WebSocket) sunt aliniate, iar jocul este gata să treacă de la așteptare la prima rundă.

4.2. Actualizări și runde de piață

Pentru fiecare rundă, piața este generată și gestionată în principal pe serverul WebSocket, care creează lista de produse în memorie și o trimite clienților prin evenimentul *economy:state*. Opțional, o vedere agregată a acestei configurații este

păstrată în market/{gameId} astfel încât prețurile și valorile reziduale să poată fi reconstruite ulterior, fără a depinde de starea din memorie.

4.2.1. Generarea pieței pe serverul WebSocket

Pentru gameID-ul activ și *runda*, serverul:

- Folosește șabloanele globale de produse (tipuri, tipuri de bile, calități, intervale de preț) pentru a genera setul complet de combinații posibile.
- Aplică logica prețului și a risipei (calitate A/B/C, Block vs Slab, diferențe între bile și alte reguli de joc).
- Selectează aleatoriu un subset de produse pentru acea rundă și le stochează în memorie ca lista actuală de piețe pentru acel joc.

```
const assignProductsToGame = async ({ gameId, round }) => {
  const productsRef = admin.firestore().collection('products');
  const gameRef = admin.firestore().collection('games').doc(gameId);
  const marketRef = admin.firestore().collection('market').doc(gameId);

  const marbleTypes = ['Red', 'White', 'Gray', 'Black', 'Cream'];
  const productTypes = ['Block', 'Slab'];
  const qualities = ['A', 'B', 'C'];

  const qualityMultipliers = {…
  };

  const calculateProductPrice = (basePrice, productType) => {…
  };

  const calculateProductWaste = (baseWaste, productType) => {…
  };

  try {…
  } catch (error) {
    console.error('Error assigning products to game:', error);
    throw new Error('Failed to assign products to game');
  }
};

module.exports = { assignProductsToGame };
```

Figură 11: atribuie funcția ProductsToGame

Aceasta produce o piață specifică pentru runde, care reflectă atât catalogul static, cât și parametrii dinamici ai jocului curent.

4.2.2. Emisia în timp real și progresia rundelor

Pentru fiecare *rundă*:

- Serverul WebSocket trimite un *eveniment* economy:*state* tuturor jucătorilor din joc, cu o încărcătură utilă care conține lista produselor disponibile în acea rundă și orice metadate relevante.
- Pe client, *GameContext-ul* (*GameContextHybridRobust*) stochează aceste informații în starea locală și le expune diferitelor ecrane (*piață*, statistici, antet).
- Jucătorii interacționează cu această piață (cumpărând produse, schimbând stocurile) prin evenimente WebSocket, în timp ce serverul păstrează copia funcțională a pieței și stocurile în memorie.

Această împărțire clară înseamnă că Firestore păstrează o viziune durabilă și agregată asupra pieței pe rundă, în timp ce stratul WebSocket livrează lista de produse din beton și gestionează interacțiunile rapide în fiecare rundă.

4.3. Minerit, validare și recompense

Mineritul în RockChain combină interacțiunea în timp real cu stocarea persistentă. Serverul WebSocket generează și distribuie probleme de minerit, în timp ce subcolecția *blockchain*/{*gameId*}/*blocuri* păstrează o evidență permanentă a fiecărei probleme, a răspunsurilor primite și a câștigătorului.

La sfârșitul fiecărei runde, toate aceste informații sunt consolidate într-un set de recompense per jucător, care sunt apoi scrise înapoi în Firestore.

4.3.1. Crearea blocurilor și a problemelor

Când un eveniment minerit este declanșat pentru un *anumit joc* și *rundă*:

- Serverul WebSocket creează o nouă problemă de mining și o asociază cu un *blockId*.
- Creează sau actualizează un document în *blockchain*/{*gameId*}/*blocks*/{*blockId*}.

Această scriere inițială asigură că fiecare eveniment minier are o ancoră persistentă în Firestore din momentul în care este creat.

4.3.2. Distribuție în timp real și trimiterea răspunsurilor

Odată ce blocul este creat:

- Serverul emite un eveniment *mining*:*problem* către toți jucătorii din joc, care conține problema și contextul relevant.
- Clienții afișează problema și permit utilizatorilor să trimită răspunsuri într-o perioadă limitată de timp.
- Fiecare jucător își trimite răspunsul printr-un *eveniment mining*:*submit* WebSocket, pe care serverul îl înregistrează în memorie și îl poate adăuga la array-ul de răspunsuri din documentul de bloc corespunzător.

Această interacțiune modelează o **"cursă de minerit"** în care jucătorii concurează pentru a oferi soluția corectă cât mai rapid posibil.

4.3.3. Validarea și persistența rezultatelor de exploatare

Pe măsură ce răspunsurile sosesc:

- Serverul evaluează fiecare încercare în raport cu soluția corectă.
- Când un câștigător valid este detectat conform regulilor jocului (primul **răspuns corect**), serverul:

- Determină jucătorul câștigător.
- Marcează blocul ca fiind închis (*isActive* = false).
- Actualizează *blockchain*/{*gameId*}/*blocks*/{*blockId*} cu un obiect câștigător care conține: userId, *userName*, răspunsul lor și dacă a fost corect, *elapsedTime* și *responseTime* pentru audit.

- Serverul emite apoi un eveniment *mining*:*result* către toți clienții, inclusiv informații despre câștigător și orice feedback imediat.

```
// Función para manejar respuestas de minado con tiempo del cliente
const handleMiningSubmit = (blockId, userId, response, userName, clientResponseTime) => {
  console.log(`[MINING][SUBMIT] blockId=${blockId} user=${userId} response=${response} clientResponseTime=${clientResponseTime}`);

  // Definir timestamp actual al inicio de la función
  const now = Date.now();

  // Usar el tiempo del cliente si está disponible, sino calcular del servidor
  let elapsed;
  if (clientResponseTime !== undefined && clientResponseTime > 0) {···
  } else {···
  }

  // Obtener el problema para verificar la respuesta correcta
  const problemData = miningProblems.get(blockId);
  if (!problemData) {···
  }

  const correctAnswer = parseFloat(problemData.activeProblem.solution);
  const userAnswer = parseFloat(response);
  const isCorrect = userAnswer === correctAnswer;

  console.log(`[MINING][SUBMIT] blockId=${blockId} user=${userId} isCorrect=${isCorrect} elapsed=${elapsed}ms`);

  // Inicializar o actualizar respuestas del bloque
  if (!miningResponses.has(blockId)) {···
  }

  const blockData = miningResponses.get(blockId);

  // Verificar si el usuario ya respondió
  const existingResponseIndex = blockData.responses.findIndex(r => r.userId === userId);
  if (existingResponseIndex !== -1) {···
  } else {
    // Añadir nueva respuesta
  const newResponse = {···
  };
  blockData.responses.push(newResponse);
  }

  // Determinar ganador y estado de completado
  const correctResponses = blockData.responses.filter(r => r.isCorrect);
  let winner = null;
  let isCompleted = false;

  if (correctResponses.length > 0) {···
  }
```

Figură 12: funcția handleMiningSubmit

Acest flux garantează că fiecare eveniment de minerit are o înregistrare urmăribilă, auditabilă, în Firestore în timp ce este gestionat în timp real de serverul WebSocket.

4.3.4. Consolidarea recompenselor la sfârșitul rundei

La finalul fiecărei runde, RockChain adună tot ce s-a întâmplat în faza de piață și minerit pentru a calcula un set unic de recompense pentru fiecare jucător. Aceste recompense sunt apoi scrise în Firestore, actualizând atât documentul global al jocului, cât și statusul individual al fiecărui jucător.

Declanșarea finalului rundei

Când timpul oficial al rundei expiră (conform câmpului *roundTime* și al timestamp-urilor din *gameAuthorities*) sau când toate condițiile necesare sunt îndeplinite, serverul WebSocket schimbă statusul jocului în faza de închidere a rundei.

În acel moment, adună datele pe care le are în memorie, cum ar fi:

- Blocuri extrase și câștigătoarele lor.
- Inventarele jucătorilor.
- Industrii selectate.
- Nivelurile de deșeuri și alți indicatori.

Calcularea recompenselor per jucător

Backend-ul calculează, pentru fiecare *userId*:

- **Recompense pentru minerit**: de exemplu, o cantitate fixă de RockCoins pentru fiecare bloc minat corect.
- **Reducerea deșeurilor**: bazată pe eficiența industriei alese și pe regulile jocului (de exemplu, câtă deșeuri este eliminată prin adoptarea unor strategii circulare).

Scrierea rezultatelor către Firestore

În *games*/{*gameId*}, următoarele sunt actualizate:

- *câștigândIndustrie pentru runda respectivă.*
- *roundRewards*[*userId*] = { *industryReward*, *wasteRemoved*, *miningReward*, ... } pentru toți jucătorii.
- *recompenseAtribuite* = adevărat.
- *status*, care este setat pe roundEnd sau waitingForNextRound, în funcție de ce urmează.

În *users*/{*userId*}, în *games*[*gameId*], următorul lucru este actualizat:

- *RockCoins*, adăugând recompensele câștigate în rundă.
- *deșeuri*, scăzând ce a fost eliminat (fără să scadă sub zero, desigur).
- Și asigură că starea persistentă a jucătorului reflectă cu acuratețe rezultatul rundei.

Notificarea clienților

- După ce datele sunt finalizate de scris, serverul WebSocket trimite un *eveniment round_end* cu un rezumat al recompenselor și al industriei câștigătoare.
- Clienții își actualizează statul local (recompense, inventar, industrie câștigătoare) și afișează rezultatele înainte de a trece la runda următoare.

Datorită acestui flux combinat, Firestore acționează ca o înregistrare definitivă a ceea ce a câștigat fiecare jucător și cum s-au schimbat resursele lor, în timp ce WebSocket asigură că totul se simte imediat și sincronizat pe toate dispozitivele.

5. SECURITATE, CONFIDENȚIALITATE ȘI PREGĂTIRE PENTRU PRODUCȚIE

Backend-ul RockChain a fost proiectat astfel încât doar componentele de încredere să poată modifica datele oficiale ale sistemului, astfel încât informațiile despre utilizatori și drivere să rămână limitate, protejate și ușor de gestionat în timp. Această secțiune rezumă modul în care securitatea, confidențialitatea și operațiunile de bază sunt gestionate la nivelul de date în producție.

5.1. Reguli de securitate Firestore și autentificare

Mediul de producție RockChain combină autentificarea Firebase, regulile de securitate Firestore și funcțiile Firebase pentru a asigura că doar utilizatorii și serviciile autorizate pot accesa sau modifica datele.

5.1.1. Autentificarea ca poartă de intrare

- Orice acces la backend necesită autentificare prealabilă prin autentificare Firebase (fie prin email/parolă, fie printr-un alt furnizor compatibil).
- Fiecărui utilizator i se atribuie un userID stabil, care este folosit în Firestore și pe serverul WebSocket.

5.1.2. Acces limitat la datele de profil și meci

- Regulile de securitate Firestore asigură că fiecare utilizator autentificat poate doar:
  - Citește și actualizează propriul *document de utilizatori*/{*userId*}.
  - Citește datele de la jocurile în care sunt înregistrați.
  - Nu pot scrie direct în documente critice precum *jocuri*/{*gameId*} sau *blockchain*/{*gameId*}/*blocuri*/{*blockId*} din client.
- Accesul la datele altor jucători este restricționat la minimul necesar pentru funcționarea jocului (de exemplu, nume sau clasamente publice).

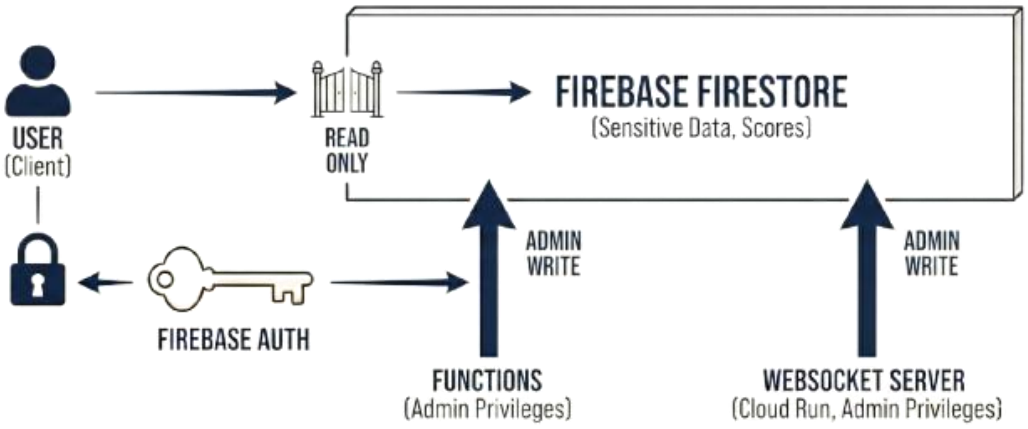
5.1.3. Scrieri controlate de funcții Firebase

- Majoritatea operațiunilor care afectează gameplay-ul (crearea sau alăturarea la jocuri, actualizarea statusului jocului, salvarea recompenselor etc.) pot fi executate doar din funcțiile Firebase cu privilegii ridicate.
- Regulile Firestore sunt păstrate simple și practic verifică că:
  - Clienții autentificați pot citi anumite documente dacă au permisiune.
  - Doar funcțiile backend sau conturile de servicii pot scrie pe căi protejate (*jocuri*, *blockchain*, anumite câmpuri din *utilizatori* etc.).

5.1.4. Serverul WebSocket ca serviciu de încredere

- Serverul WebSocket rulează pe Cloud Run și folosește un cont de serviciu cu permisiuni limitate pentru:
  - Citește și scrie documente necesare pentru a gestiona runde, minerit și recompense.
  - Cheamă funcțiile backend atunci când este necesară validare suplimentară.
- Toată comunicarea dintre client și serverul WebSocket se face prin conexiuni securizate (WSS), iar fiecare mesaj primit este validat în raport cu starea curentă din memorie și din Firestore înainte de aplicarea unor modificări permanente

Cu aceste măsuri, RockChain asigură că niciun utilizator neautentificat nu poate accesa datele și că chiar și clienții autentificați pot citi și trimite doar intenții. Modificările reale ale bazei de date sunt întotdeauna controlate de backend, asigurând integritatea jocului.



Figură 13: Cum funcționează securitatea în Rockchain.

5.2. GDPR și protecția datelor

Deoarece RockChain este folosit în activități pilot cu persoane reale, stratul său de date a fost conceput în conformitate cu principiile Regulamentului **General privind Protecția Datelor (GDPR) al Uniunii** Europene și cu legile naționale aplicabile.

5.2.1. Minimizarea datelor și limitarea scopului

- Sistemul stochează doar informațiile strict necesare pentru:
  - Conducerea sesiunilor de joc (ID-uri de utilizator, participare, resurse în joc).
  - Evaluarea și îmbunătățirea activităților de instruire (statistici agregate, jurnale anonimizate).

- Nu sunt colectate sau stocate categorii speciale de date personale (cum ar fi sănătatea, religia sau opiniile politice).

5.2.2. Pseudonimizarea jucătorilor

- La nivel tehnic, utilizatorii sunt identificați în principal prin ID-ul lor de utilizator Firebase. Opțional, pot folosi un nume sau un avatar vizibil în timpul experienței de învățare.
- Partenerii proiectului din fiecare pilot sunt responsabili pentru gestionarea, separat și local, a oricărei corespondențe între un *userID* și identitatea reală a participantului. Astfel, sistemul funcționează folosind identificatori pseudonime.

5.2.3. Baza legală și informațiile pentru participanți

- RockChain este folosit în contexte structurate de formare (educație pentru adulți, formare profesională, dezvoltare profesională continuă).
- Fiecare sit pilot oferă participanților următoarele:
    - O notificare de confidențialitate care explică ce date sunt colectate în RockChain, în ce scopuri și pentru cât timp.
    - Detalii clare de contact pentru exercitarea drepturilor lor ca persoane interesate (acces, corectare, ștergere etc.).
- În funcție de cadrul legal al țării, baza legală poate fi consimțământul sau interesul legitim de a oferi și evalua instruirea. Acest lucru este documentat la nivelul fiecărui pilot de către partenerul relevant.

5.2.4. Locație de depozitare și măsuri de securitate

- Toată comunicarea cu backend-ul (Firestore, funcții, WebSocket) este criptată în timpul tranzitului folosind HTTPS sau WSS.
- Datele sunt, de asemenea, criptate în repaus, folosind mecanismele standard Firebase și Google Cloud.
- Accesul la consola Firebase și mediul de producție este limitat la un grup restrâns de administratori ai consorțiului, care folosesc autentificare robustă și permisiuni bazate pe roluri.

5.2.5. Păstrare, anonimizare și ștergere

- Datele de joc sunt păstrate doar atât timp cât este necesar pentru:
    - Verifică piloții.
    - Generează rezultatele proiectului convenite (cum ar fi rapoartele și statisticile generale privind utilizarea și învățarea).
- După ce această perioadă s-a încheiat, partenerii pot:
    - Anonimizați suplimentar datele, eliminând orice legătură directă între *userId* și listele locale de identitate.
    - Șterge jocurile, utilizatorii sau seturile întregi de date din instanța Firestore de producție.

- Dacă un participant solicită acest lucru, informațiile sale pot fi șterse sau pseudonimizate prin modificarea sau ștergerea *documentului* utilizatorilor/{*userId*} și a datelor conexe, așa cum este explicat în notificarea de confidențialitate a pilotului.

În concluzie, stratul de date RockChain a fost proiectat să funcționeze într-un mod transparent, limitat și reversibil, susținând obiectivele educaționale ale proiectului fără a pierde controlul asupra datelor personale, care rămân întotdeauna în mâinile partenerilor locali pilot.

5.3. Backup-uri, curățenie și întreținere de bază

Pentru a fi pregătit pentru producție, backend-ul RockChain trebuie să fie nu doar sigur, ci și ușor de operat și întreținut. Configurația actuală include proceduri ușoare pentru backup-uri, curățarea datelor învechite și întreținere zilnică.

5.3.1. Backup-uri și opțiuni de recuperare

- Firestore oferă deja redundanță și replicare încorporate la nivel de stocare.
- În plus, administratorii pot exporta periodic colecții de chei (precum *jocuri*, *utilizatori*, *blockchain*, *produse*, *piață*) către Cloud Storage, fie prin scripturi programate, fie prin funcții cloud.
- Aceste backup-uri servesc atât pentru:
    - Recuperează datele în caz de ștergeri accidentale.
    - Păstrați instantaneele înghețate ale anumitor faze pilot pentru documentare sau analiză, chiar dacă ulterior sunt șterse din baza de date activă.

5.3.2. Curățarea jocurilor vechi și a datelor piloților

- Pentru a preveni creșterea necontrolată a datelor și pentru a susține minimizarea:
    - Jocurile vechi pot fi marcate ca arhivate și șterse, inclusiv *subcolecțiile* blockchain/{gameId} și *documentele de piață*/{*gameId*}.
    - Intrări conexe în *users*/{*userId*}.*games*[*gameId*] poate fi, de asemenea, șters sau anonimizat.
- Această curățare poate fi realizată prin:
    - Sarcini programate (funcții cloud sau scripturi externe).
    - Acțiuni manuale coordonate de partenerul tehnic.

5.3.3. Monitorizare și verificări operaționale de bază

- Dashboard-urile Firebase și Cloud Run oferă:
    - Metrici de utilizare (citiri, scrieri, lățime de bandă).
    - Jurnale de erori pentru funcții și serverul WebSocket.

  - Indicatori de performanță care ajută la identificarea interogărilor care ar putea necesita indici noi.
- În timpul piloților, se efectuează evaluări regulate pentru:
  - Confirmați că limitele sau cotele nu au fost depășite.
  - Detectează configurații incorecte (cum ar fi reguli de securitate slab definite sau indici lipsă).
  - Ajustați resursele sau strategiile de indexare dacă numărul utilizatorilor simultani crește.

Datorită acestor măsuri, backend-ul RockChain rămâne ușor și ușor de gestionat, permițând partenerilor să folosească instrumentul atât în timpul, cât și după proiect, fără a fi nevoie de o echipă DevOps dedicată, asigurând în același timp integritatea și disponibilitatea datelor.

6. PAȘI URMĂTORI ȘI CONCLUZII

Lucrările desfășurate în WP4. A1 a dus la un strat de date concret și funcțional pentru instrumentul de învățare RockChain. Aceasta se bazează pe o arhitectură hibridă care combină funcțiile Firestore, Firebase și un server dedicat WebSocket care rulează pe Google Cloud Run, consumat dintr-o aplicație mobilă integrată în React Native. Acest stack tehnologic oferă deja o bază solidă pentru gestionarea jocurilor, utilizatorilor, pieței și evenimentelor de minerit în timp real, cu responsabilități clare pentru fiecare componentă și un model Firestore compact care reflectă elementele cheie ale jocului și sistemul de recompense.

De aici înainte, pașii următori nu mai sunt atât despre realizarea unor schimbări structurale majore, cât despre consolidarea și consolidarea a ceea ce a fost deja construit. Pe de o parte, regulile de securitate Firestore pentru colecțiile principale (jocuri, utilizatori, blockchain) sunt ajustate fin și testate în scenarii realiste pentru a asigura un control bine controlat al accesului. De asemenea, verificăm dacă separarea dintre mediile de dezvoltare și cea de producție este implementată corect în toate părțile sistemului: de la compilații mobile la funcții și serverul WebSocket. Ca parte a acestei pregătiri, se desfășoară sesiuni pilot la scară mică cu utilizatori interni și parteneri de proiect pentru a confirma că fluxurile de date funcționează corect în condiții reale de joc.

În plus, sunt integrate mecanisme ușoare de monitorizare și jurnalizare pentru cele mai sensibile operațiuni, cum ar fi modificările rundelor, calculele recompenselor și validarea mineritului, care vor ajuta la depanarea potențialelor erori în timpul piloților. Indexurile celor mai active colecții sunt, de asemenea, revizuite și ajustate, pe baza tiparelor reale de interogare, căutând un echilibru bun între performanță și costul de scriere. Se definesc și rutine simple de mentenanță pentru a curăța sesiunile vechi și a exporta colecții de chei, cum ar fi jocuri, utilizatori și blockchain, atât pentru backup, cât și pentru analiză.

Privind spre viitor, plănuim să expunem puncte de intrare bine documentate, fie prin funcții, fie prin API-ul WebSocket, astfel încât alte activități din același WP4 să se poată conecta la acest strat de date fără a fi nevoie să duplicăm logica. De asemenea, căutăm să valorificăm informațiile deja stocate (cum ar fi istoricul mineritului, configurațiile pieței și recompensele) pentru a informa deciziile pedagogice în proiectarea scenariilor sau analiza învățării în următoarele pachete de lucru

În concluzie, WP4. A1 a transformat RockChain dintr-un design abstract despre "ce să stochezi" într-o infrastructură de date concretă, pregătită pentru producție și capabilă să susțină jocuri multiplayer pe dispozitive mobile. Arhitectura actuală este intenționat simplă și ușor de înțeles, dar suficient de robustă pentru a rula sesiuni reale, a impune reguli de securitate de bază și a înregistra urmăribil tot ce se întâmplă în fiecare joc. Concentrând pașii următori pe stabilizare, monitorizare și mici extensii incrementale, în

loc de reproiectări majore, partenerii de proiect pot trata acest strat de date ca pe coloana vertebrală fiabilă a instrumentului: ceva ce poate fi implementat, întreținut și adaptat după necesitate, atât în timpul piloților, cât și în posibile evoluții viitoare dincolo de durata de viață a Proiectului.