WP4-A1. Izrada baze podataka za e-Learning alat.

"Financirano sredstvima Europske unije. Izneseni stavovi i mišljenja su stavovi i mišljenja autora i ne moraju se podudarati sa stavovima i mišljenjima Europske unije ili Europske izvršne agencije za obrazovanje i kulturu (EACEA). Ni Europska unija ni EACEA ne mogu se smatrati odgovornima za njih."

Sadržaj

1. UVOD

Ovo izvješće sažima konačne nalaze WP4. O1: Izrada baze podataka za e-Learning alat. U cjelokupnom okviru RockChain projekta, ova aktivnost fokusira se na dizajn i realizaciju pozadinskog sloja podataka koji podržava ozbiljnu igru kako bi multiplayer sesije, tržišna dinamika i rudarski događaji mogli raditi u stvarnom vremenu na stabilnoj i produkcijski spremnoj infrastrukturi.

Ključni zadatak WP4. A1 je definirati i razviti bazu podataka i back-end arhitekturu koja će moći podržavati dosljedno rukovanje korisničkim profilima, sesijama igre, rundama, nagradama i odlukama unutar igre prema scenarijima učenja definiranim u projektu. Trenutna implementacija takve arhitekture temelji se na *Firebase Firestore* kao središnjoj bazi podataka, skupu Firebase funkcija koje obuhvaćaju osjetljive operacije pisanja i logiku igre, te *posvećenom WebSocket serveru* implementiranom na *Google Cloud Run*, koji upravlja komunikacijom u stvarnom vremenu i niskom latencijom između igrača.

Ovo izvješće ima namjerno uski i tehnički fokus: dokumentira kako je RockChain sloj podataka strukturiran i implementiran u produkciji. Opisuje glavne komponente arhitekture i njihov hosting, strukturu ključnih Firestore kolekcija i dokumenata te najrelevantnije tokove podataka koji stvaraju i ažuriraju te informacije tijekom igre. Također sažima ključne proizvodne aspekte vezane uz sigurnosne standarde, performanse i osnovno održavanje baze podataka.

S ovim sažetim opisom podatkovnog sloja, WP4. A1 pruža praktičnu referencu koju programeri i tehnički partneri mogu koristiti za implementaciju, upravljanje ili proširenje RockChain infrastrukture. Dobivena arhitektura bit će temelj na kojem se pouzdano mogu graditi i front-end alat za e-učenje i pilot aktivnosti kasnijih radnih paketa.

2. PREGLED ARHITEKTURE PODATAKA

2.1. Opći koncepti

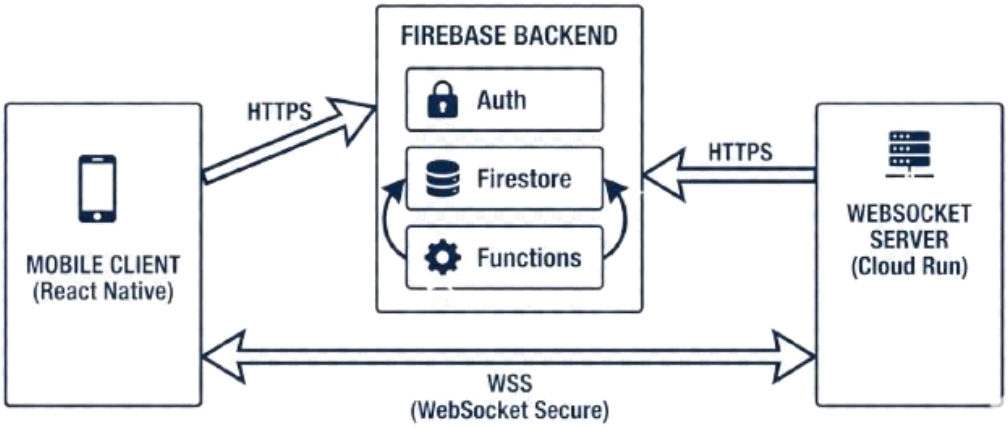
RockChain alat za e-učenje radi u produkciji na troslojnoj arhitekturi, usmjerenoj na jedan Firebase projekt i posvećenu uslugu u stvarnom vremenu:

- React **Native mobilni klijent** (Android/iOS) nudi korisničko sučelje i povezuje učenike s igrom.
- **Firebase backend** s Firestoreom, funkcijama, autentifikacijom - čuva sve trajne podatke i enkapsulira osjetljive operacije.
- **WebSocket poslužitelj** implementiran na Google Cloud Runu upravlja komunikacijom u stvarnom vremenu između igrača tijekom igre, s niskom latencijom.

Na visokoj razini, arhitektura slijedi hibridni model:

- Firestore služi kao autoritativna baza podataka koja pohranjuje igre, korisnike, blokove, proizvode, tržišta i nagrade.
- Firebase funkcije djeluju kao kontrolirani gateway za kritične zapise, poput kreiranja igara, pridruživanja igrama, ažuriranja informacija o rundama i nagradama, te primjenu validacijskih i poslovnih pravila na strani servera.
- WebSocket poslužitelj čuva stanje u memoriji svake aktivne igre i prenosi događaje u stvarnom vremenu poput početka runde, tržišnih podataka, problema s rudarenjem i rezultata svim povezanim klijentima, pohranjujući relevantne rezultate samo na jasno definiranim točkama u Firestoreu.

Budući da sve komponente dijele isti Firebase identitet i sigurnosni kontekst, autentifikacija, autorizacija i pristup podacima dosljedni su kroz cijeli sloj. Ova se arhitektura može replicirati za razvoj i testiranje s drugim Firebase projektom i implementacijom WebSocketsa, uz zadržavanje točne strukture, ali korištenje neprodukcijskih vjerodajnica i URL-ova.



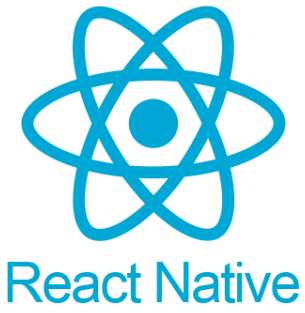
Slika 1: Dijagram arhitekture

Ova podjela odgovornosti omogućuje RockChainu da spoji izdržljivu, dobro strukturiranu pohranu onoga što se dogodilo u svakoj sesiji s brzim, događajima temeljenim ažuriranjima onoga što se događa u tom trenutku tijekom igranja.

2.2. Glavne komponente

2.2.1. React Native mobilni klijent

React Native mobilna aplikacija jedina **je ulazna točka za korisnike**. Svrha je prikazati sučelje igre i reagirati na stanje pozadine, dok sama aplikacija ne upravlja persistenceom.



Slika 2: React Native

U produkciji mobilni klijent:

- Autentificira se pomoću Firebase autentifikacije i dobiva korisnički identitet (*userId*) koji se dosljedno koristi kroz cijeli backend.
- Povezuje igre putem posebnih Firebase funkcijskih poziva, poput kreiranja ili pridruživanja igri, koji služe za validaciju zahtjeva i zatim ažuriranje Firestorea.
- Otvara WebSocket vezu s Cloud Run krajnjom točkom nakon što je korisnik u igri i šalje *gameId*, autentificirani *userId* i osnovne informacije o igraču kako bi mogao smjestiti socket u odgovarajuću igraću sobu.
- Pretplatite se na backend događaje (bilo od Firestore slušatelja ili WebSocket poruka) i mapirajte ih na lokalno stanje aplikacije, koje se zatim prikazuje na ekranu.

Klijent ne piše izravno u Firestore za kritične operacije. Umjesto toga:

- Firebase funkcije trebaju se pozivati kad god je potrebna trajna promjena podatkovnog modela, primjeri uključuju ažuriranje stanja igre i pisanje nagrada.
- Oni emitiraju WebSocket događaje, poput *'spremno za igranje'*, *'kupi proizvod'*, *'pošalji odgovor na rudarenje'*, koji se obrađuju na serveru i održavaju gdje je to prikladno.

Ovaj dizajn drži mobilnu aplikaciju tankim, reaktivnim slojem usmjerenim na interakciju s korisnikom, pri čemu se sve autoritativne odluke donose u pozadini o stanju igre.

2.2.2. Firebase backend: Firestore i funkcije

Firebase backend sastoji se od Cloud Firestore s Firebase funkcijama, koje zajedno pružaju siguran i skalabilan sloj podataka.

Slika 3: Firebase

Cloud Firestore (baza podataka i autoritativna pohrana)

Firestore pohranjuje sve informacije koje trebaju preživjeti WebSocket vezu i biti dostupne za analitiku ili oporavak. Sljedeće su među onima pohranjenim u RockChainu:

- **Stanje igre** u zbirci *igara* : Svaki dokument predstavlja meč. Uključuje kod igre, popis igrača, trenutnu rundu i status, pobjedničku industriju po rundi, tajmere i dodijeljene nagrade.
- **Stanje korisnika** u korisničkoj kolekciji: svaki dokument pohranjuje korisnički profil - prikazno ime, avatar - i podatke po igri: kamenNovčići, nagomilani otpad, odabrana industrija, inventar
- **Rudarenje i "blockchainom inspirirani" podaci** u *blockchainu/{gameId}/blokovima*: za svaku igru ova podkolekcija pohranjuje probleme rudarenja, primljene odgovore i konačnog pobjednika svakog bloka.
- **Katalog proizvoda** u kolekciji *proizvoda* : predlošci za dostupne mramorne proizvode (tip, tip mramora i opis), koji se koriste za izgradnju ponuda po rundi na tržištu.
- *Tržište*: konfiguracija tržišta po igri. Output je organiziran prikaz parametara tržišta, po rundama i vrsti proizvoda. To je korisno za obnovu ili prikaz tržišta u kontekstima koji nisu real-time flat liste.

U tom slučaju, Firestore je optimiziran za kratka, česta čitanja od strane mobilnog klijenta – trenutni status igre, ažurirani inventar itd. – i kontrolirane zapise iz backend koda koji osiguravaju integritet i dosljednost modela igre.

Firebase funkcije (poslovna logika i kontrolirano pisanje)

Firebase funkcije centraliziraju glavna poslovna pravila i osjetljive izmjene u Firestoreu. Klijent ne piše izravno u kritične dokumente poput igara ili podataka korisničkih igara, već poziva Funkcije koje:

- **Kreiraj igru**, *kreirajIgru*: stvara dokument u igrama s kodom igre, domaćinom, statusom, maksimalnim brojem igrača i drugim standardnim poljima
- Dopustite korisniku da se pridruži **igri,** na primjer: *joinGame* provjerava postoji li igra i može li se pridružiti; dodaje korisnika na popis igrača; ažurira *playerCount* i

stvara/ažurira njihove početne vrijednosti: *rockCoins*, *otpad, status*, *vremenske oznake*....

- **Ažurirajte** osjetljiva polja u igri: na primjer, informacije o rundama, agregirane rezultate ili nagrade koje se aktiviraju klijentskim događajima ili kao rezultat backend procesa na kraju 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;
}
```

Slika 4: createGame funkcija

Usmjeravanjem tih operacija kroz Funkcije, projekt:

- Osigurava dosljednu validaciju i pravila unutar svake sesije.
- Održava Firestore sigurnosna pravila jednostavnijima jer složene odluke ostaju u backend kodu, a ne u velikim skupovima pravila.
- Smanjuje rizik od zlonamjernih ili nedosljednih upisa od strane nepouzdanih klijenata.

U praksi je autoritativni put uvijek: *Klijent → Firebase funkcija / backend logika → Firestoreu***.**

2.2.3. WebSocket poslužitelj na Google Cloud Run (sloj u stvarnom vremenu)

WebSocket poslužitelj je usluga izgrađena na Node.js i radi kao kontejner na Google Cloud Runu. Njegov glavni zadatak je održavati **sve igrače povezanima** tijekom igre, uz brzu, dvosmjernu komunikaciju bez prekida.

Što točno radi? Evo:

- To osigurava da **komunikacija** u svakoj igri teče **u stvarnom vremenu**. To uključuje obavijesti poput tko se pridružuje, tko je spreman, kada runda počinje, kako tržište stoji, novi problemi s rudarstvom, rezultati i kada završava svaka runda ili cijela igra.
- Prati svaku aktivnu igru u memoriji. To mu omogućuje da koordinira što se događa bez preopterećenja Firestorea. Na primjer:
    - Tko sudjeluje i jesu li spremni.
    - U kojoj su rundi i koliko još traje do kraja.
    - Kako stoje zalihe i tržište u tom trenutku.
    - Koja su rudarska pitanja aktivna i kakvi su odgovori poslani.
- Također sprema rezultate u Firestoreu, ali samo u ključnim trenucima: kada netko osvoji blok, na kraju runde ili kada se dodjeljuju nagrade. Ponekad to radi izravno, a ponekad to prosljeđuje Firebase funkcijama.

Osim toga, budući da radi na Cloud Runu, usluga se skalira ovisno o broju aktivnih igara. Sve je povezano putem sigurnog WebSocket endpointa (WSS), unutar istog Firebase projekta gdje se nalaze Firestore i Functions.

Ukratko:

- **WebSocket** server **vam u stvarnom vremenu govori što se događa** unutar igre.
- **Firestore i Functions pohranjuju službenu povijest**: što se dogodilo, kako igra napreduje i u kakvom su stanju igrači.

Zahvaljujući ovom odjelu, RockChain može ponuditi multiplayer igre koje djeluju agilno i fluidno, bez žrtvovanja kontrole i snimanja onoga što se dogodilo.

3. DIZAJN MODELA PODATAKA I BAZE PODATAKA

U produkciji, podatkovni sloj RockChaina temelji se na nekoliko ključnih kolekcija u Firestoreu. Svaki ima jasno definiranu funkciju: dizajnirani su da ih mobilna aplikacija stalno čita, dok se zapisi obavljaju kontrolirano putem Firebase funkcija i backend procesa.

Zajedno, ove kolekcije pohranjuju:

- Opći status svake igre.
- Individualni status svakog igrača unutar igre.
- Povijest rudarskih događaja.
- Konfiguracija tržišta korištena u svakoj rundi.

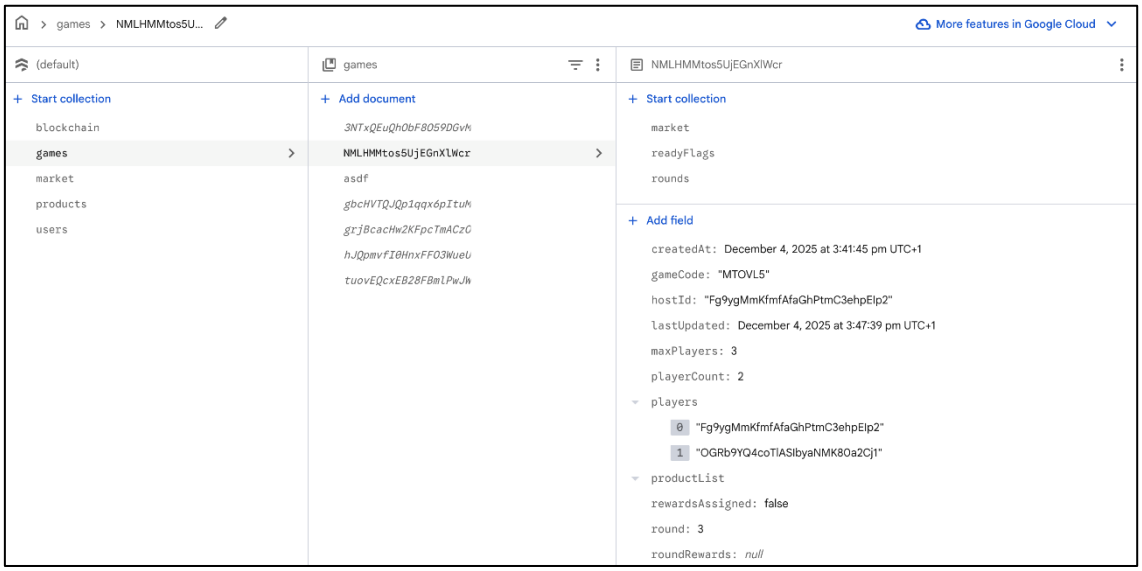
3.1. Firestore zbirke i dokumenti

3.1.1. Igre – Globalno stanje igre

Zbirka igara pohranjuje jedan dokument po igri. Svaka *igra/*{*gameId*} sadrži podatke koji opisuju što se općenito događa u toj igri.

Ovo su neka od ključnih područja:

- *gameCode*: javni kod koji igrači koriste za pridruživanje.
- *players*: popis *userID-ova* koji se nalaze u igri, zajedno s podacima poput broja igrača (*playerCount*) i maksimalnog dopuštenog broja (*maxPlayers*).
- status: u kojoj se fazi igra nalazi (može biti *čekanje*, *početak*, *in_progress*, *kraj runde*, *čekanje za sljedeću rundu* ili završeno).
- Runda: Broj tekuće runde.
- *pobjedničkaIndustrija*: Industrija koja je pobijedila u posljednjoj punoj rundi.
- *roundTime*: vremenska oznaka koja označava kraj trenutne runde; koristi se kao osnova za prikaz brojača i prijelaze.
- *nagradeDodijeljeno*: Boolean koji označava jesu li nagrade za posljednji krug već napisane.
- *rundaNagrade*: sažetak nagrada svakog igrača u prethodnoj rundi (po industriji, uklanjanju otpada, rudarstvu itd.).



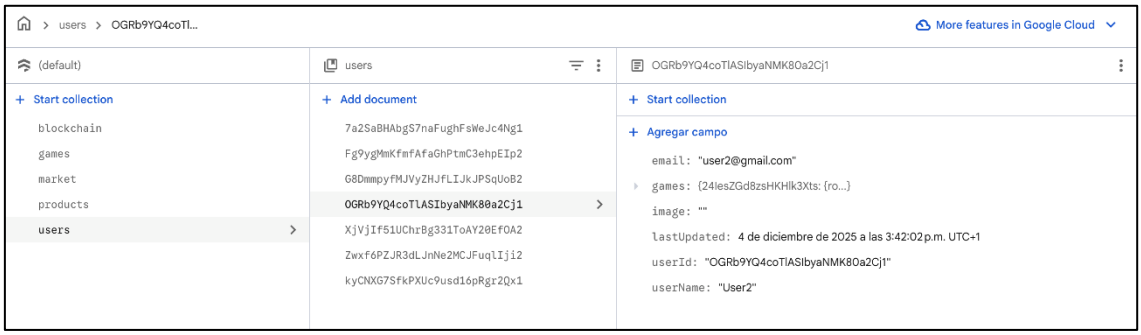
Slika 5: prikupljanje podataka o igrama

Ovaj dokument je glavni ulaz prilikom učitavanja ili ponovnog pokretanja igre. Jednostavnim čitanjem *games*/{*gameId*}, sustav može znati tko igra, u kojoj su rundi i jesu li nagrade već dodijeljene.

3.1.2. Korisnici – Profilni i podaci po utakmici

Kolekcija *korisnika* pohranjuje jedan dokument po korisniku, identificiran njihovim Firebase *userID-om*. Svaki *users*/{*userId*} dokument ima dva sloja informacija:

- **Korisnički profil** s poljima poput *userName*, opcionalni avatar (*imageUrl*) i *lastUpdated*, koji služi kao vremenska oznaka za osnovne revizije.
- **Podaci o igri,** organizirani unutar karte igre, gdje je svaki unos povezan s gameID-om. Za svaku igru u kojoj korisnik sudjeluje, pohranjuju se stvari poput sljedećeg:
    - Igraće kovanice: *kamenKovanice* i nagomilani otpad.
    - Industrija koju su odabrali (*odabrali su Industrija*) kako bi zaradili nagrade unutar modela kružne ekonomije.
    - Njihov inventar proizvoda (*proizvodi*), koji prikazuje što su kupili na tržištu.
    - Dodatni podaci poput *gameCodea*, statusa, kada su se pridružili (*joinedAt*) i rudarskog reda (*miningQueue*).
    - Informacije o posljednjem problemu rudarenja s kojim su se susreli (*lastMiningProblem*), uključujući *blockId,* detalje problema i vremensku oznaku.



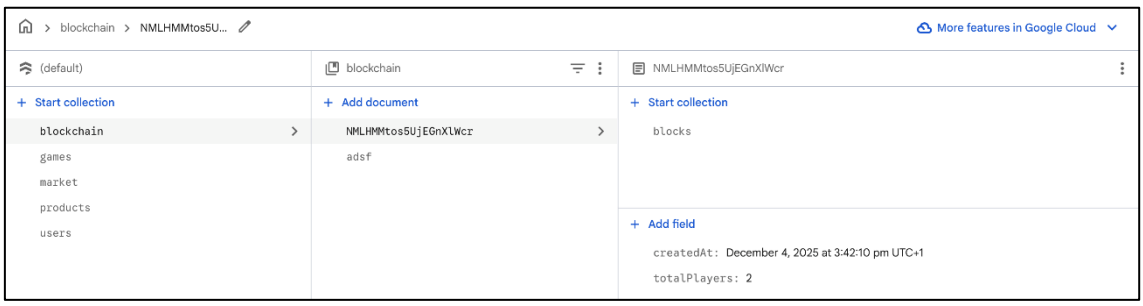
Slika 6: prikupljanje podataka korisnika

Koncentracija svih tih informacija u jedan dokument po korisniku znatno olakšava stvari: na primjer, možete odmah znati trenutnu situaciju igrača u određenoj igri, bez potrebe za višestrukim upitima ili kompliciranim kombinacijama.

3.1.3. blockchain/{gameId}/blokovi – problemi rudarenja i pobjednički blokovi

Za svaku igru, podkolekcija *blockchain/{gameId}/blocks* prati događaje rudarenja, koristeći strukturu inspiriranu blockchainom. Svaki dokument *blocks/{blockId}* predstavlja blok sa sljedećim informacijama:

- Problem aktivnog rudarenja (*activeProblem*), koji uključuje:
  - *problemId*, koji odgovara *blockId-u*.
  - Jednostavan matematički zadatak (*mathProblem*) zajedno s njegovim točnim rješenjem.
  - Datum stvaranja (*createdAt*) i detalji povezane transakcije (kao što su proizvod, vrsta operacije i uključeni korisnik).
- Odgovori primljeni (*odgovori*) dok je problem aktivan.
- Okrugli broj (*krug)* i kreirani *At* označavaju kada je blok stvoren.
- Pobjednički predmet koji se puni nakon što se odluči pobjednik rudarske utrke. Uključuje: *userID*, *korisničko ime*, dati odgovor, je li bio točan, vrijeme potrebno za odgovor (*elapsedTime*) i kada je odgovor (*responseTime*), što je korisno za reviziju.
- Indikator *isActive* koji signalizira je li blok još uvijek otvoren ili već postoji pobjednik.



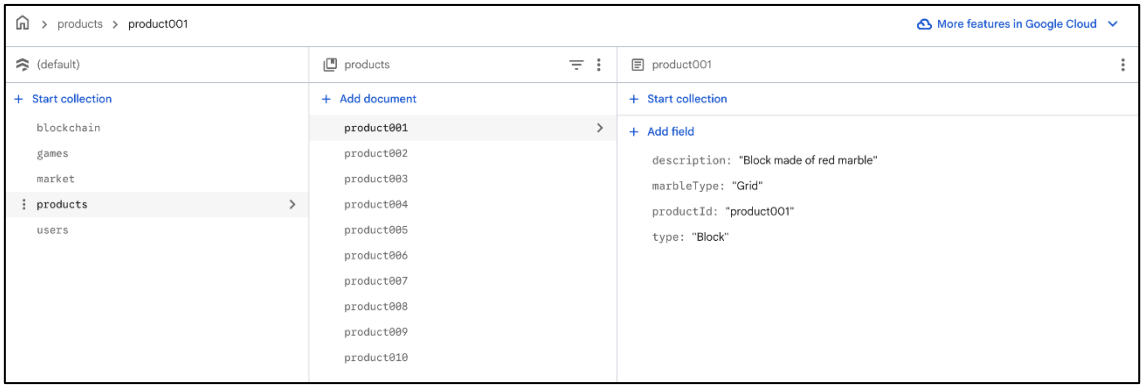
Slika 7: prikupljanje blockchain podataka

Ova struktura omogućuje RockChainu da vodi jasan i provjerljiv zapis o generiranim problemima, kako su igrači reagirali i kako su dodijeljene nagrade vezane uz rudarenje.

4.1.4. Proizvodi – Globalni katalog proizvoda

Kolekcija proizvoda definira globalni katalog mramornih proizvoda dostupnih u igri. Svaki dokument funkcionira kao stabilan predložak, s poljima kao što su:

- *productId* (na primjer: "product001")
- *tip* ("Block" ili "Slab")
- *mramorni tip* ("Crvena", "Bijela", "Siva", "Crna", "Krem")
- *kvaliteta* ("A," "B," "C")
- Opis i drugi statički podaci



Slika 8: prikupljanje podataka o proizvodu

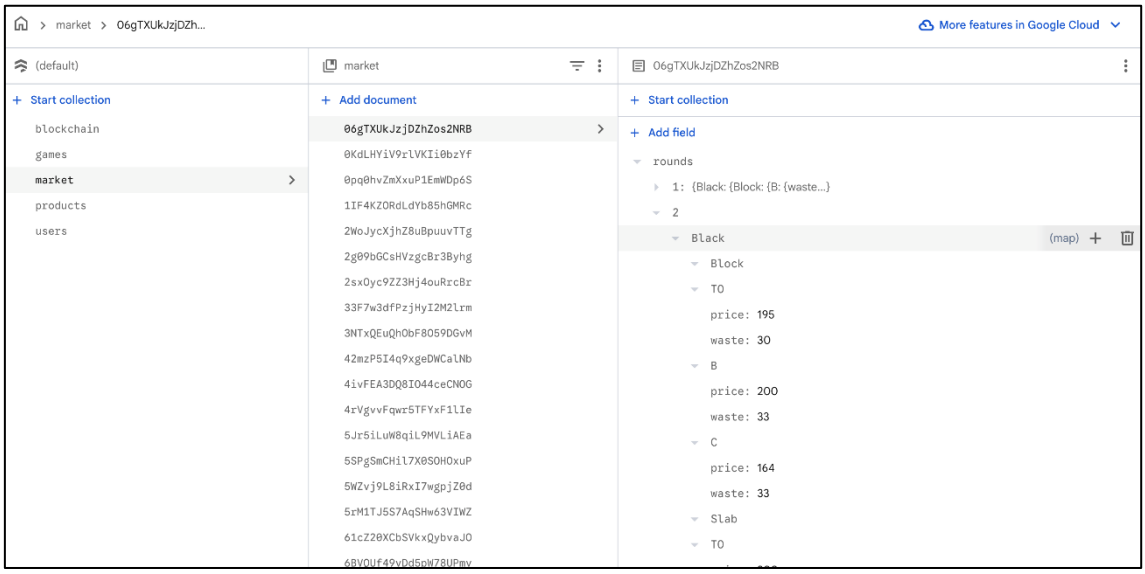
Ove predloške koristi backend logika za sastavljanje specifičnog popisa proizvoda prikazanih na tržištu za svaku rundu. One služe kao temelj za određivanje cijena i definiranje parametara vezanih uz otpad.

4.1.5. Tržište – struktura tržišta po rundama

Za svaku igru, kolekcija *tržišta*/{*gameId*} pohranjuje strukturirani prikaz tržišta. Ovaj se pogled uglavnom koristi za rekonstrukciju kako je tržište bilo konfigurirano u različitim rundama ili za naknadnu analizu.

Glavno polje je rounds: ugniježđena mapa sa strukturom *rounds[rounds][marbleType][type][quality] = { price, waste }*, gdje:

- *okrugli* je okrugli broj,
- *marbleType* je jedan od konfiguriranih tipova mramora,
- *tip* može biti "Block" ili "Slab",
- *kvaliteta* je "A", "B" ili "C", svaka sa svojom cijenom i razinom otpada.



Slika 9: prikupljanje tržišnih podataka

Ova struktura funkcionira kao svojevrsni sažetak ili povijesni prikaz tržišta. On nadopunjuje popise proizvoda u stvarnom vremenu koji se generiraju i šalju putem WebSocketa tijekom igre. Posebno je koristan za pregled kako je tržište bilo u svakoj rundi, bez potrebe da se svaki proizvod prolazi pojedinačno.

3.2. Odnosi između entiteta

Iako je Firestore dokumentno orijentirana NoSQL baza podataka i ne obrađuje odnose kao relacijska baza podataka, RockChain model definira jasne logičke odnose između svojih kolekcija.

3.2.1. Igra – Igrači

Svaki *dokument games*/{*gameId*} uključuje svoje igrače u nizu igrača, koji sadrži *userID-ove*.

Istovremeno, svaki *user*/{*userId*} dokument pohranjuje unutar game[*gameId*] mape specifičan status tog korisnika u toj igri. U praksi, to predstavlja odnos mnogo-na-više između igara i korisnika, implementiran pomoću unakrsnih referenci.

3.2.2. Igra – Rudarski blokovi

Svaka igra ima svoju podkolekciju: *blockchain*/{*gameId*}/*blokovi*.

Svi blokovi povezani su s jednim *gameID-om*, a svaki uključuje broj runde kojoj pripada. To je odnos jedan prema mnogima: igra može imati nekoliko blokova, ali svaki blok pripada samo jednoj igri.

3.2.3. Igra – Tržište

Svaka igra ima jedinstveni *dokument tržišta*/{*gameId*}, gdje je tržište organizirano po rundama. Iako Firestore ne nameće odnose, to je logično odnos jedan na jedan između *games*/{*gameId*} i *market*/{*gameId*}.

3.2.4. Proizvodi – Tržište / Zalihe

Globalna kolekcija proizvoda definira osnovne predloške proizvoda.

I polje rundi u *market*/{*gameId*} i nizovi proizvoda unutar *games*/{*gameId*} konstruirani su iz vrijednosti i parametara definiranih u tim *productId-ovima*. To je odnos jedan prema mnogima: jedan proizvod u katalogu može se pojaviti na različitim tržištima i u nekoliko inventara igrača.

3.2.5. Kako se rekonstruira status igre i igrača?

Backend obično slijedi ovaj obrazac:

- Pročitaj *games*/{*gameId*} da dobiješ opći status igre.
- Čitajte *users*/{*userId*} i provjerite *games*[*gameId*] da vidite kako taj igrač napreduje.
- Pročitaj *market*/{*gameId*} ako ti treba potpuni pregled cijena i otpada po rundi.
- Čitajte *blockchain*/{*gameId*}/*blokove* (i filtrirajte po rundama ako je potrebno) kako biste analizirali povijest rudarenja.

Model primjenjuje blagu denormalizaciju kako bi smanjio pozive i pojednostavio upite od klijenta, bez gubitka sljedivosti koju projektni partneri trebaju za analizu podataka kasnije.

4. KLJUČNI TOKOVI PODATAKA U PRODUKCIJI

Osim statičkog podatkovnog modela, RockChain radi zahvaljujući nizu ponavljajućih podatkovnih tokova koji stvaraju i ažuriraju dokumente u Firestore tijekom igre. Ti tokovi povezuju mobilnog klijenta, Firebase funkcije, WebSocket poslužitelj i kolekcije koje smo ranije opisali. Zajedno osiguravaju da svi igrači vide isto stanje igre, u stvarnom vremenu i s mogućnošću praćenja.

4.1. Stvaranje igre i pridruživanje

Kreiranje igara i uključivanje igrača ovise o Firebase funkcijama koje obavljaju prve zapise u kolekcije igara i korisnika. To osigurava da svaka igra započinje u dosljednom stanju i da su igrači ispravno registrirani i u globalnom dokumentu igre i u svom korisničkom profilu.

4.1.1. Kreiranje igara

Kada korisnik odluči hostati novu igru:

- Nakon prijave, klijent poziva Firebase funkciju (*createGame*).
- Funkcija stvara novi dokument u *games*/{*gameId*}.
- U users/{*userId*}, funkcija stvara ili ažurira unosne *igre*[*gameId*] s početnim vrijednostima.

Ovaj proces osigurava da su i opće stanje igre i individualno stanje domaćina jasno definirani od samog početka.

4.1.2. Pridruživanje postojećoj igri

Kada se igrač pridruži postojećem meču:

- Igrač unosi gameCode u klijent.
- Klijent poziva Firebase funkciju (*joinGame*), koja rješava gameCode na gameId.
- Funkcija provjerava da je igra moguća pridružiti se i zatim:
    - Dodaje igrača u *igre*/{*gameId*}/*players* i povećava *broj igrača*.
    - Kreira ili ažurira *korisnike*/{*userId*}/*games*[*gameId*] s njihovim početnim podacima u igri (valute, otpad, status, vremenske oznake).

```
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.');
    }
  }
});
```

Slika 10: funkcija pridruživanja igri

To izbjegava izravne klijentske zapise u kolekciju igara i osigurava da su svi igrači registrirani prema istim pravilima.

4.1.3. Povezivanje na sloj u stvarnom vremenu

Kada Firestore dokumenti postoje:

- Svaki klijent otvara WebSocket vezu prema Cloud Run endpointu.
- Klijent šalje *gameId*, *userId* i osnovne podatke profila.
- WebSocket poslužitelj registrira socket u odgovarajućem *gameRooms* unosu i počinje pratiti tog korisnika u njegovom stanju u memoriji.

U ovom trenutku, i persistentno stanje (Firestore) i real-time stanje (WebSocket in-memory mape) su usklađeni, te je igra spremna prijeći iz čekanja u prvu rundu.

4.2. Ažuriranja i runde tržišta

Za svaku rundu, tržište se generira i upravlja prvenstveno na WebSocket serveru, koji stvara popis proizvoda u memoriji i šalje ga kupcima putem događaja *economy:state*. Opcionalno, agregirani prikaz ove konfiguracije čuva se u market/{gameId} tako da se cijene i rezidualne vrijednosti mogu kasnije rekonstruirati bez oslanjanja na stanje u memoriji.

4.2.1. Generiranje tržišta na WebSocket poslužitelju

Za aktivni *gameId* i *rundu*, server:

- Koristi globalne predloške proizvoda (tipovi, tipovi kuglica, kvalitete, cjenovni rasponi) za generiranje punog skupa mogućih kombinacija.
- Primjenjuje logiku cijene i otpada (kvalitetni A/B/C, Block vs Slab, razlike u kuglicama i druga pravila igre).
- Nasumično odabire podskup proizvoda za tu rundu i pohranjuje ih u memoriju kao trenutni popis tržišta za tu igru.

```
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 };
```

Slika 11: assignProductsToGame funkciju

To proizvodi tržište specifično za rundu koje odražava i statički katalog i dinamičke parametre trenutne igre.

4.2.2. Emisija u stvarnom vremenu i progresija rundi

Za svaki *krug*:

- WebSocket server šalje događaj *economy*:*state* svim igračima u igri s payloadom koji sadrži popis proizvoda dostupnih u toj rundi i sve relevantne metapodatke.
- Na klijentu, *GameContext* (*GameContextHybridRobust*) pohranjuje te informacije u lokalnom stanju i izlaže ih različitim ekranima (*tržište*, *statistika*, *zaglavlje*).
- Igrači komuniciraju s ovim tržištem (kupuju proizvode, mijenjaju zalihe) putem WebSocket događaja, dok server čuva radnu kopiju tržišta i zalihe u memoriji.

Ova jasna podjela znači da Firestore ima trajan, agregirani pregled tržišta po rundi, dok WebSocket sloj isporučuje konkretan popis proizvoda i rješava brze interakcije tijekom svake runde.

4.3. Rudarenje, validacija i nagrade

Rudarenje u RockChainu kombinira interakciju u stvarnom vremenu s trajnom pohranom. WebSocket poslužitelj generira i distribuira probleme rudarenja, dok blockchain/{*gameId*}/*blocks* podkolekcija vodi trajni zapis o svakom problemu, primljenim odgovorima i tko je pobijedio.

Na kraju svake runde, sve te informacije konsolidiraju se u skup nagrada po igraču, koje se zatim zapisuju natrag u Firestore.

4.3.1. Stvaranje blokova i problema

Kada se pokrene rudarski događaj za određenu *igru* i *rundu*:

- WebSocket poslužitelj stvara novi problem rudarenja i povezuje ga s *blockId-om*.
- Stvara ili ažurira dokument u blockchainu/{*gameId*}/*blocks*/{*blockId*}.

Ovaj početni zapis osigurava da svaki rudarski događaj ima trajno sidro u Firestore od trenutka kada je stvoren.

4.3.2. Distribucija u stvarnom vremenu i slanje odgovora

Nakon što je blok stvoren:

- Server šalje događaj *rudarenja*:*problem* svim igračima u igri, koji sadrži problem i relevantan kontekst.
- Klijenti prikazuju problem i omogućuju korisnicima da pošalju odgovore unutar ograničenog vremenskog okvira.
- Svaki igrač šalje svoj odgovor putem događaja *mining*:*submit* WebSocket, koji server bilježi u memoriju i može dodati u niz odgovora u odgovarajućem blok dokumentu.

Ova interakcija modelira **"rudarsku utrku"** u kojoj se igrači natječu tko će što brže pronaći ispravno rješenje.

4.3.3. Validacija i trajnost rezultata rudarenja

Kako pristižu odgovori:

- Server procjenjuje svaki pokušaj prema ispravnom rješenju.
- Kada se prema pravilima igre otkrije valjani pobjednik (**prvi točan odgovor**), server:
    - Određuje pobjedničkog igrača.
    - Označava blok kao zatvoren (*isActive* = netočno).
    - Ažurira *blockchain*/{*gameId*}/*blocks*/{*blockId*} s winner objektom koji sadrži: *userId*, *userName*, njihov odgovor i je li bio ispravan, *elapsedTime* i *responseTime* za reviziju.

- Poslužitelj zatim šalje događaj *rudarenja*:*rezultat* svim klijentima, uključujući informacije o pobjedniku i eventualne trenutne povratne informacije.

```
// 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) {···
  }
```

Slika 12: handleMiningSubmit funkcija

Ovaj tijek jamči da svaki rudarski događaj ima prativi, auditabilni zapis u Firestoreu, dok ga WebSocket poslužitelj i dalje obrađuje brzinom u stvarnom vremenu.

4.3.4. Konsolidacija nagrada na kraju runde

Na kraju svake runde, RockChain prikuplja sve što se dogodilo u fazi tržišta i rudarenja kako bi izračunao jedinstven skup nagrada po igraču. Te se nagrade zatim zapisuju u Firestore, ažurirajući i globalni dokument igre i individualni status svakog igrača.

Pokretanje kraja runde

Kada službeno vrijeme runde istekne (prema polju *roundTime* i vremenskim oznakama u *gameAuthorities*), ili kada su ispunjeni svi potrebni uvjeti, WebSocket server mijenja status igre u fazu zatvaranja runde.

U tom trenutku prikuplja podatke koje ima u memoriji, kao što su:

- Iskopani blokovi i njihovi pobjednici.
- Inventari igrača.
- Odabrane industrije.
- Razine otpada i drugi pokazatelji.

Računalne nagrade po igraču

Backend izračunava, za svaki *userId*:

- **Nagrade za rudarenje**: na primjer, fiksni broj RockCoina za svaki ispravno iskopan blok.
- **Smanjenje otpada**: temeljeno na učinkovitosti odabrane industrije i pravilima igre (npr. koliko se otpada eliminira usvajanjem kružnih strategija).

Rezultati pisanja za Firestore

U games/{*gameId*}, sljedeće se ažuriraju:

- *osvajanjem Industrije u toj rundi.*
- *roundRewards*[*userId*] = { *industryReward*, *wasteRemoved*, *miningReward*, ... } za sve igrače.
- *rewardsAssigned* = točno.
- *status*, koji se postavlja na roundEnd ili waitingForNextRound, ovisno o tome što slijedi.

U users/{*userId*}, unutar *games*[*gameId*], sljedeće se ažurira:

- *rockCoins*, koji zbrajaju nagrade osvojene u rundi.
- *otpad,* oduzimajući ono što je uklonjeno (naravno, bez pada ispod nule).
- I osigurava da trajno stanje igrača točno odražava ishod runde.

Obavještavanje klijenata

- Nakon što su podaci gotovi, WebSocket poslužitelj šalje *round_end* događaj sa sažetkom nagrada i pobjedničke industrije.
- Klijenti ažuriraju svoju lokalnu državu (nagrade, zalihe, pobjednička industrija) i prikazuju rezultate prije nego što prijeđu u sljedeći krug.

Zahvaljujući ovom kombiniranom tijeku, Firestore djeluje kao konačni zapis o tome što je svaki igrač osvojio i kako su se njihovi resursi promijenili, dok WebSocket osigurava da sve djeluje odmah i sinkronizirano na svim uređajima.

5. SIGURNOST, PRIVATNOST I SPREMNOST ZA PROIZVODNJU

RockChain backend je dizajniran tako da samo pouzdane komponente mogu mijenjati službene sistemske podatke, te da informacije o korisnicima i driverima ostaju ograničene, zaštićene i jednostavne za upravljanje tijekom vremena. Ovaj odjeljak sažima kako se sigurnost, privatnost i osnovne operacije rješavaju na podatkovnom sloju u produkciji.

5.1. Firestore sigurnosna pravila i autentifikacija

RockChain produkcijsko okruženje kombinira Firebase autentifikaciju, Firestore sigurnosna pravila i Firebase funkcije kako bi se osiguralo da samo ovlašteni korisnici i usluge mogu pristupiti ili mijenjati podatke.

5.1.1. Autentifikacija kao ulazna vrata

- Sav pristup backendu zahtijeva prethodnu autentifikaciju putem Firebase autentifikacije (putem e-maila/lozinke ili drugog kompatibilnog pružatelja).
- Svaki korisnik dobiva stabilan *userId*, koji se koristi u Firestore i na WebSocket poslužitelju.

5.1.2. Ograničen pristup podacima o profilu i igri

- Firestore sigurnosna pravila osiguravaju da svaki autentificirani korisnik može samo:
  - Čitajte i ažurirajte *vlastite users*/{*userId*} dokumente.
  - Čitajte podatke iz igara u kojima su registrirani.
  - Ne mogu izravno pisati u kritične dokumente poput *games*/{*gameId*} ili *blockchain*/{*gameId*}/*blocks*/{*blockId*} iz klijenta.
- Pristup podacima drugih igrača ograničen je na minimum potreban za funkcioniranje igre (npr. javna imena ili rangiranje).

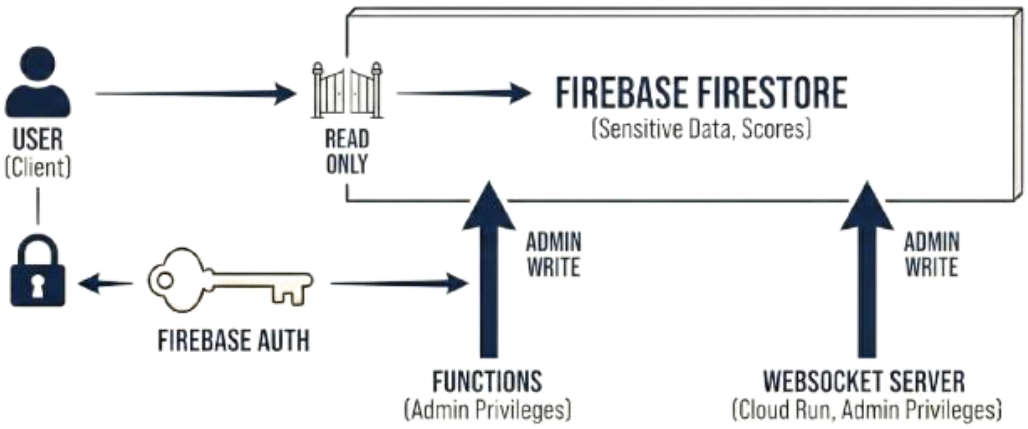
5.1.3. Upisi kontrolirani funkcijama Firebase

- Većina operacija koje utječu na gameplay (kreiranje ili pridruživanje igrama, ažuriranje statusa igre, spremanje nagrada itd.) može se izvoditi samo iz Firebase funkcija s povišenim privilegijama.
- Firestore pravila su jednostavna i u osnovi provjeravaju da:
  - Autentificirani klijenti mogu čitati određene dokumente ako imaju dopuštenje.
  - Samo backend funkcije ili servisni računi mogu pisati na zaštićene puteve (*igre*, *blockchain*, određena polja u *korisnicima* itd.).

5.1.4. WebSocket poslužitelj kao pouzdana usluga

- WebSocket poslužitelj radi na Cloud Runu i koristi servisni račun s ograničenim dozvolama za:
  - Čitajte i pišite dokumente potrebne za upravljanje rundama, rudarenjem i nagradama.
  - Pozovite backend funkcije kada je potrebna dodatna validacija.
- Sva komunikacija između klijenta i WebSocket poslužitelja odvija se preko sigurnih veza (WSS), a svaka primljena poruka provjerava se u odnosu na trenutno stanje u memoriji i u Firestore prije nego što se primijene trajne promjene

Ovim mjerama RockChain osigurava da nijedan neautentificirani korisnik ne može pristupiti podacima, te da čak i autentificirani klijenti mogu samo čitati i slati namjere. Stvarne izmjene baze podataka uvijek kontrolira backend, osiguravajući integritet igre.



Slika 13: Kako sigurnost funkcionira u Rockchainu.

5.2. GDPR i zaštita podataka

Budući da se RockChain koristi u pilot aktivnostima s stvarnim osobama, njegov sloj podataka dizajniran je u skladu s načelima **Opće uredbe o zaštiti podataka Europske unije (GDPR)** i važećim nacionalnim zakonima.

5.2.1. Minimizacija podataka i ograničenje svrhe

- Sustav pohranjuje samo informacije koje su strogo potrebne za:
  - Pokretanje sesija igre (korisnički ID-ovi, sudjelovanje, resursi unutar igre).
  - Evaluacija i poboljšanje aktivnosti obuke (agregirane statistike, anonimizirani zapisi).
- Ne prikupljaju se niti pohranjuju posebne kategorije osobnih podataka (poput zdravlja, religije ili političkih stavova).

5.2.2. Pseudonimizacija igrača

- Na tehničkoj razini, korisnici se prvenstveno identificiraju prema svom Firebase *userID-u*. Opcionalno, mogu koristiti vidljivo ime ili avatar tijekom procesa učenja.
- Projektni partneri u svakom pilotu odgovorni su za zasebno i lokalno upravljanje bilo kakvom korespondencijom između userID-a i stvarnog identiteta sudionika. Stoga sustav koristi pseudonimne identifikatore.

5.2.3. Pravna osnova i informacije sudionicima

- RockChain se koristi unutar strukturiranih obrazovnih konteksta (obrazovanje odraslih, strukovno usavršavanje, kontinuirani profesionalni razvoj).
- Svaka pilot lokacija sudionicima pruža:
    - Obavijest o privatnosti koja objašnjava koji se podaci prikupljaju u RockChainu, za koje svrhe i koliko dugo.
    - Jasni kontakt podaci za ostvarivanje njihovih prava kao subjekta podataka (pristup, ispravak, brisanje itd.).
- Ovisno o pravnom okviru zemlje, pravna osnova može biti pristanak ili legitimni interes za pružanje i evaluaciju obuke. To se dokumentira na razini svakog pilota od strane relevantnog partnera.

5.2.4. Lokacija skladištenja i sigurnosne mjere

- Sva komunikacija s backendom (Firestore, funkcije, WebSocket) šifrirana je tijekom prijenosa pomoću HTTPS-a ili WSS-a.
- Podaci se također šifriraju u mirovanju, koristeći standardne Firebase i Google Cloud mehanizme.
- Pristup Firebase konzoli i produkcijskom okruženju ograničen je na malu skupinu administratora konzorcija, koji koriste robusnu autentifikaciju i dozvole temeljene na ulogama.

5.2.5. Zadržavanje, anonimizacija i brisanje

- Podaci o igri čuvaju se samo onoliko dugo koliko je potrebno za:
    - Pokreni pilote.
    - Generirajte dogovorene rezultate projekata (kao što su izvještaji i opća upotreba i statistika učenja).
- Nakon završetka tog razdoblja, partneri mogu:
    - Dodatno anonimizirajte podatke, uklanjajući svaku izravnu poveznicu između *userId-a i lokalnih popisa identiteta.*
    - Izbrišite igre, korisnike ili cijele skupove podataka iz produkcijske Firestore instance.
- Ako sudionik to zatraži, njegovi podaci mogu biti izbrisani ili pseudonimizirani izmjenom ili brisanjem *njihovog user*/{*userId*} dokumenta i povezanih podataka, kako je objašnjeno u obavijesti o privatnosti pilota.

Ukratko, RockChain sloj podataka dizajniran je da radi transparentno, ograničeno i reverzibilno, podržavajući obrazovne ciljeve projekta bez gubitka kontrole nad osobnim podacima, koji uvijek ostaju u rukama lokalnih pilot partnera.

5.3. Backup, čišćenje i osnovno održavanje

Da bi bio spreman za produkciju, backend RockChaina mora biti ne samo siguran, već i jednostavan za korištenje i održavanje. Trenutna konfiguracija uključuje lagane procedure za sigurnosne kopije, čišćenje zastarjelih podataka i svakodnevno održavanje.

5.3.1. Sigurnosne kopije i opcije oporavka

- Firestore već nudi ugrađenu redundanciju i replikaciju na razini pohrane.
- Osim toga, administratori povremeno mogu izvoziti kolekcije ključeva (kao što su *igre*, *korisnici*, *blockchain*, *proizvodi*, *tržište*) u Cloud Storage, bilo putem zakazanih skripti ili cloud funkcija.
- Ove sigurnosne kopije služe i za:
    - Oporavite podatke u slučaju slučajnih brisanja.
    - Sačuvajte zamrznute snimke određenih pilot faza radi dokumentacije ili analize, čak i ako se kasnije očiste iz aktivne baze podataka.

5.3.2. Čišćenje starih igara i pilot podataka

- Kako bismo spriječili nekontrolirani rast podataka i podržali minimizaciju:
    - Stare igre mogu se označiti kao arhivirane i izbrisati, uključujući njihove *blockchain*/{*gameId*} podkolekcije i *market*/{*gameId*} dokumente.
    - Povezani unosi u *users*/{*userId*}.*games*[*gameId*] također se može izbrisati ili anonimizirati.
- Ovo čišćenje može se obaviti putem:
    - Zakazani zadaci (cloud funkcije ili vanjski skripti).
    - Ručne radnje koordinira tehnički partner.

5.3.3. Nadzor i osnovne operativne provjere

- Firebase i Cloud Run nadzorne ploče nude:
    - Metrike korištenja (čitanja, pisanja, propusnost).
    - Zapisi o greškama za funkcije i WebSocket poslužitelj.
    - Pokazatelji uspješnosti koji pomažu u prepoznavanju upita koji možda trebaju nove indekse.
- Tijekom pilotiranja redovito se provode pregledi kako bi:
    - Potvrdite da limiti ili kvote nisu prekoračeni.
    - Otkrivanje pogrešnih konfiguracija (kao što su loše definirana sigurnosna pravila ili nedostajući indeksi).

- Prilagodite resurse ili strategije indeksiranja ako se broj istovremenih korisnika poveća.

Zahvaljujući tim mjerama, RockChainov backend ostaje lagan i upravljiv, omogućujući partnerima da koriste alat i tijekom i nakon projekta bez potrebe za posvećenim DevOps timom, uz osiguranje integriteta i dostupnosti podataka.

6. SLJEDEĆI KORACI I ZAKLJUČCI

Rad proveden u WP4. A1 je rezultirao konkretnim i funkcionalnim slojem podataka za alat za učenje RockChain. Ovo se temelji na hibridnoj arhitekturi koja kombinira Firestore, Firebase funkcije i namjenski WebSocket poslužitelj koji radi na Google Cloud Runu, korišten iz mobilne aplikacije ugrađene u React Native. Ovaj tehnološki sloj već pruža čvrstu osnovu za upravljanje igrama, korisnicima, tržištima i rudarskim događajima u stvarnom vremenu, s jasnim odgovornostima za svaku komponentu i kompaktnim Firestore modelom koji odražava ključne elemente igre i sustava nagrađivanja.

Odavde, sljedeći koraci nisu toliko o velikim strukturnim promjenama, koliko o konsolidaciji i jačanju onoga što je već izgrađeno. S jedne strane, sigurnosna pravila Firestore za glavne kolekcije (igre, korisnici, blockchain) se usavršavaju i testiraju u realnim scenarijima kako bi se osiguralo da je pristup dobro kontroliran. Također provjeravamo je li razdvajanje razvojnog i produkcijskog okruženja ispravno implementirano u svim dijelovima sustava: od mobilnih verzija do funkcija i WebSocket servera. Kao dio ove pripreme, provode se male pilot sesije s internim korisnicima i projektnim partnerima kako bi se potvrdilo da protok podataka ispravno funkcionira u stvarnim uvjetima igre.

Osim toga, ugrađuju se lagani mehanizmi za nadzor i evidentiranje za najosjetljivije operacije, poput promjena rundi, izračuna nagrada i validacije rudarenja, što će pomoći u otklanjanju mogućih pogrešaka tijekom pilot-projekta. Indeksi najaktivnijih kolekcija također se pregledavaju i prilagođavaju na temelju stvarnih obrazaca upita, tražeći dobar balans između performansi i troškova pisanja. Također se definiraju jednostavne rutine održavanja za čišćenje starih sesija i izvoz kolekcija ključeva poput igara, korisnika i blockchaina, kako za sigurnosne kopije, tako i za analizu.

Gledajući unaprijed, planiramo izložiti dobro dokumentirane ulazne točke, bilo putem funkcija ili WebSocket API-ja, kako bi se druge aktivnosti unutar istog WP4 mogu povezati na ovaj podatkovni sloj bez potrebe za dupliciranjem logike. Također nastojimo iskoristiti informacije koje se već pohranjuju (poput rudarske povijesti, tržišnih konfiguracija i nagrada) za informiranje pedagoških odluka u dizajnu scenarija ili analizi učenja u sljedećim radnim paketima

Ukratko, WP4. A1 je RockChain odveo od apstraktnog dizajna o tome "što pohraniti" do konkretne podatkovne infrastrukture spremne za produkciju i sposobne podržavati multiplayer igre na mobilnim uređajima. Trenutna arhitektura namjerno je jednostavna i laka za razumjeti, ali dovoljno robusna za pokretanje pravih sesija, provođenje osnovnih sigurnosnih pravila i praćenje praćenja svega što se događa u svakoj igri. Usmjeravanjem sljedećih koraka na stabilizaciju, praćenje i male inkrementalne nadogradnje, umjesto na velike redizajne, partneri na projektu mogu ovaj sloj podataka tretirati kao pouzdanu okosnicu alata: nešto što se može implementirati, održavati i prilagođavati po potrebi, kako tijekom pilot-projekata tako i u mogućoj budućoj evoluciji nakon trajanja Projekta.