CHAINGODZ

April 2025

# Public Report

Alexandre Fonseca de Souza
Software Engineer Intern
BTG Pactual

# EXECUTIVE SUMMARY

## ABOUT CHAINGODZ

ChainGodz is a next-generation blockchain-powered game designed to merge immersive gameplay with the transparency and ownership that only decentralized technology can provide. Built on the Solana blockchain and powered by a robust backend infrastructure, ChainGodz allows players to acquire, trade, and utilize digital assets with full on-chain verifiability. With its dual architecture — combining smart contracts and an off-chain PostgreSQL-backed system — the platform ensures seamless interactions, responsive gameplay, and real economic value through token-driven mechanics. ChainGodz aims to become a defining experience in Web3 gaming, where every item used, every transaction made, and every duel fought is recorded on the blockchain, creating a true player-owned universe.

## INTRODUCTION TO THE PRODUCT

ChainGodz is an on-chain battle and collection game where players interact with the Solana blockchain to buy and use digital items for gameplay. At its core, the product is designed to give players ownership over their in-game assets through smart contracts and token-based transactions. Players can collect consumables, utility items, and use them to influence battles or list them in a peer-to-peer marketplace.
Unlike traditional games, where items are centrally stored and can be revoked, ChainGodz ensures that all actions are verified by wallet signatures, and ownership is preserved on-chain. The front end is built in Next.js with Tailwind CSS, offering a sleek and reactive interface. The backend is written in Rust, ensuring high performance for handling game logic, while smart contracts developed using the Anchor framework on Solana handle the economic backbone.

# PRODUCT ARCHITECTURE

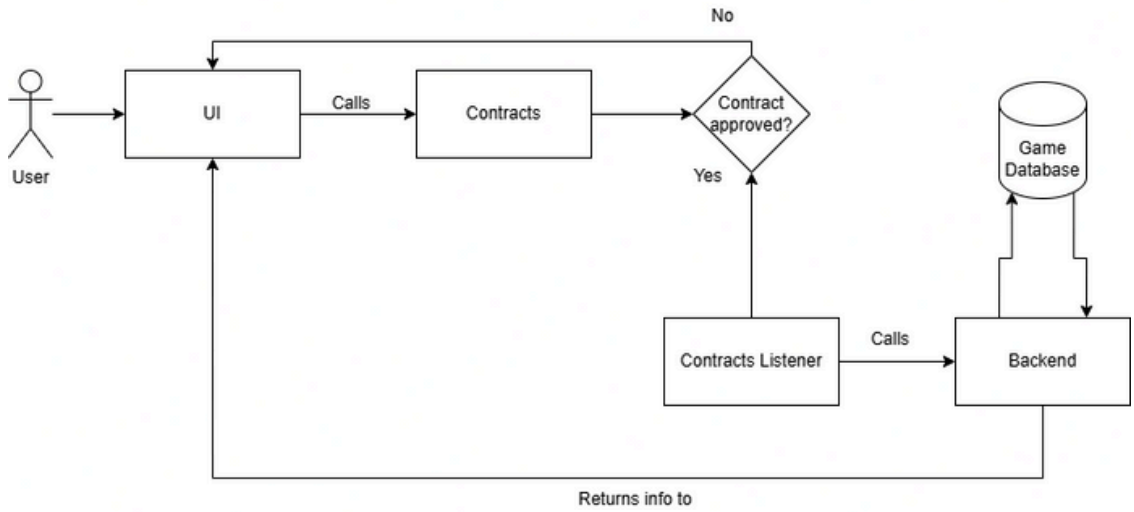
- **UI**: Developed using Next.js and Tailwind CSS, the UI integrates Phantom wallet authentication, item management, and trigger buttons for blockchain transactions.

- **Backend:** Written in Rust, the backend manages player profiles, inventory states, marketplace logic, and synchronizes with on-chain events via listeners.

- **Smart Contracts:** Built using the Anchor framework on Solana, the contracts handle core functionalities like transferring tokens, emitting events, and verifying game wallet ownership.

- **Database:** PostgreSQL stores all off-chain representations, including player XP, item metadata, match history, and marketplace transactions.

- **Architecture image:**



# FEATURE BREAKDOWN

- **Wallet Integration**: Players connect their Phantom (or other Solana-compatible) wallets to begin interacting with the game. All purchases and actions are verified using their public key.

- **Buy Item Flow:** Through a dedicated smart contract, users send a fixed amount of SOL to the game wallet. The backend then validates the transaction and associates an item to the player in the database.

- **Event Listeners:** Solana contracts emit events that confirm that the transaction was made. A backend listener running in a compute instance monitors these and updates the off-chain system in real-time.

- **Marketplace:** Players can list or purchase unique items via a decentralized interface. Ownership is validated via on-chain verification and synced off-chain for usability.

- **Duels and Battle System:** Each duel costs "Vigor", a player energy stat. Players may consume items for boosts. All changes are tracked and updated in both systems.

- **Game Inventory:**
  Items are categorized into:
  Consumables: Restore energy or XP.
  Boosters: Improve duel win probability.
  Marketable Assets: Tradeable and transferable items.

# COMMERCIAL STRATEGY

The commercial value of ChainGodz lies in its "play-to-own" model. Every item acquired is not just a virtual good, but a token-bound asset that can be used, traded, or sold. By integrating item scarcity, reward mechanisms, and a peer-to-peer economy, ChainGodz aims to capture:

- The growing interest in decentralized games.
- The market segment of NFT collectors and gamers.
- Future opportunities for DAO-based governance and staking.

With tokenomics planned for future development, players will eventually be able to stake assets, vote in governance, and earn rewards through game interaction.

# LEGAL AND ETHICAL COMPLIANCE

ChainGodz adheres to GDPR, LGPD, and CCPA principles by treating wallet addresses as pseudonymous identifiers. No PII is collected. Transactions are non-custodial, and no fiat operations are involved, limiting financial regulation requirements. Smart contracts are audited internally and all actions are transparent.

The system promotes responsible development with:

- Transparent logging of all contract activity.
- Clear user consent on wallet connect.
- Ethical design without dark patterns.

# MARKET OPPORTUNITY

With the blockchain gaming market projected to exceed $60B by 2027 (source: DappRadar), ChainGodz is poised to capture an early-mover advantage in the Solana-based gaming sector. The rapid rise of player-owned assets, combined with the performance of Solana and ease of development through Anchor, makes this the right time for launch.

ChainGodz offers:

- Real asset ownership.
- Seamless game and wallet integration.
- A roadmap to decentralized governance.

# SPRINT DOCUMENTATION ARCHIVE

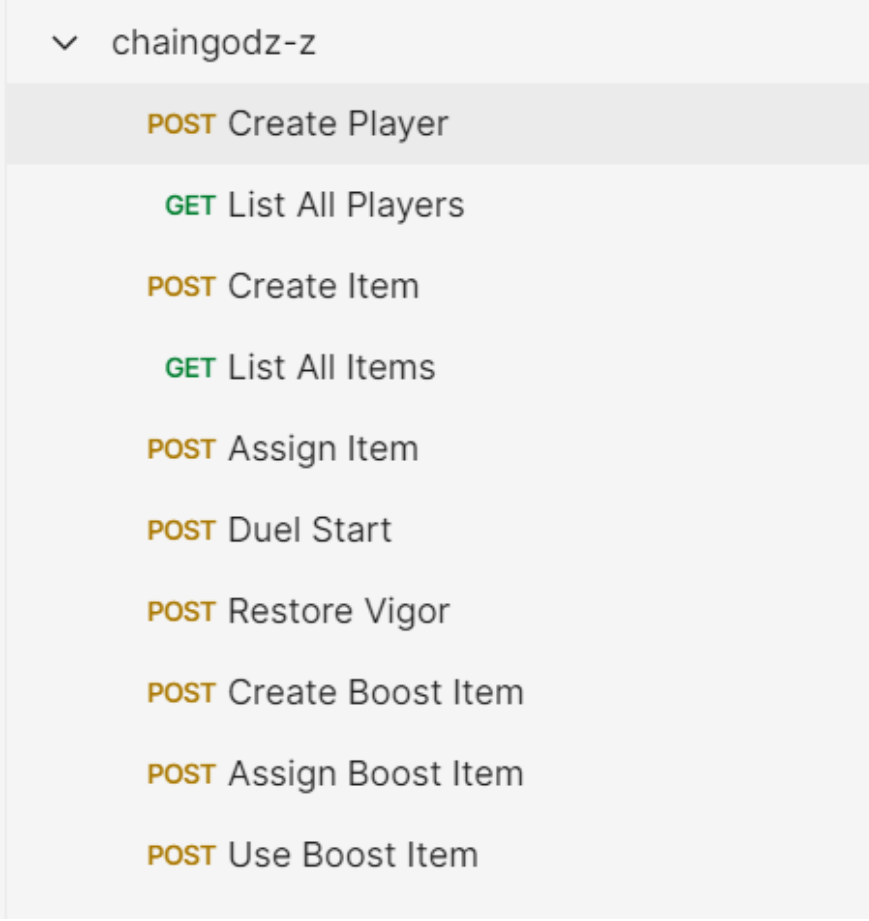
- **Sprint 1**: Key feature definition; Key Feature Architecture; Detailed Submission Plan.
- **Sprint 2**: Market Research Report; Persona Customer Journey.
- **Sprint 3:** Compliance Analysis; Product Development Plan.
- **Sprint 4:** Public Report.

# PROGRESS OVERVIEW

The ChainGodz platform has reached a significant point in its development lifecycle, with meaningful progress achieved across the three fundamental layers of the system: backend infrastructure, smart contract logic, and the frontend user interface. Each layer has evolved in parallel, ensuring the architecture remains cohesive, modular, and scalable for future feature expansion.

## BACKEND LAYER – RUST API & POSTGRESQL INTEGRATION

The backend, built entirely in Rust, serves as the off-chain processing layer and acts as the brain of the application, ensuring synchronization between blockchain actions and the internal game state. One of the key architectural accomplishments at this stage has been the setup of a robust RESTful API with clearly defined routes that encapsulate most of the game logic already.
The PostgreSQL schema is designed to handle multiple entities such as players, items, items_associated, and marketplace. Each of these tables supports indexed queries, foreign key relationships, and is designed with future scalability in mind. The use of JSON-capable fields and array columns in players (for storing item IDs, for example) enhances flexibility without compromising structure.



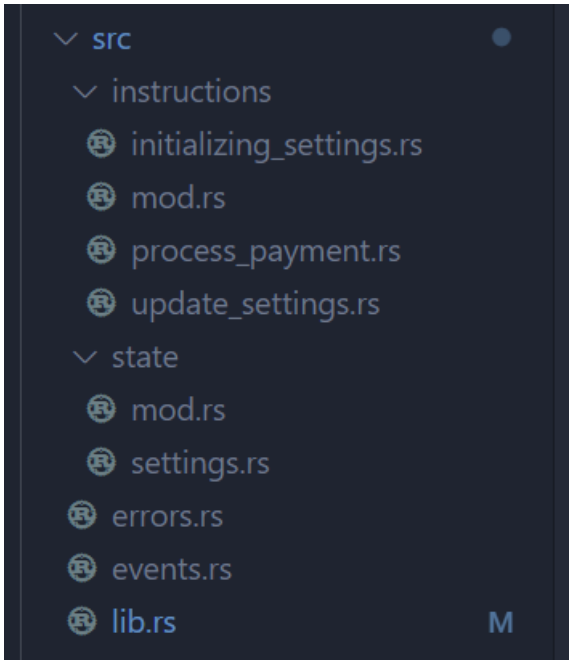

(IMAGEM 1: REPRESENTA AS CHAMADAS DAS ROTAS DA APLICAÇÃO NO POSTMAN)

## SMART CONTRACT LAYER – ANCHOR & SOLANA PROGRAMS

The smart contract component of ChainGodz has been built using the Anchor framework on Solana, ensuring fast transaction execution, robust auditing capabilities, and ease of integration with the frontend. Already have a buy function, that accepts a wallet address and performs a transfer of a pre-fixed amount of SOL to a predefined game wallet. This logic is securely bound to user-initiated wallet prompts to ensure user consent. The contracts emit structured events after successful transactions, which can be captured by off-chain services to synchronize backend state and provide frontend feedback. Also, the account ownership is validated using system instructions, and constraints such as #[account(mut)] and CHECK: annotations are used in Anchor to avoid arbitrary changes. The deployment of the contract to Solana localnet and subsequent testing through Phantom wallet transactions demonstrate the entire interaction loop functioning correctly for simple payment and event cases.

```rust
use std::str::FromStr;
use anchor_lang::system_program::{self, Transfer};
use crate::instructions::*;
use crate::state::*;

mod errors;
mod instructions;
mod state;
mod events;

declare_id!("FoTMV911CQibnBbyF4xaovetSkAh6qcaRJL26hvQpfGp");

const GAME_WALLET: &str = "4aPiggyRfzuRwZEL6CNnB6Emxf4Lme9gfpu14Z9eD4kd";

#[program]
pub mod game_contract {
    use super::*;

    pub fn process_payment(ctx: Context<BuyItem>, amount: u64) -> Result<()> {
        instructions::process_payment(ctx, amount)
    }

    pub fn initializing_settings(ctx: Context<InitializingSettings>) -> Result<()> {
        instructions::initializing_settings::initializing_settings(ctx)
    }

    pub fn update_settings(ctx: Context<UpdateSettings>, new_settings: Settings) -> Result<()> {
        instructions::update_settings::update_settings(ctx, new_settings)
    }
}
```

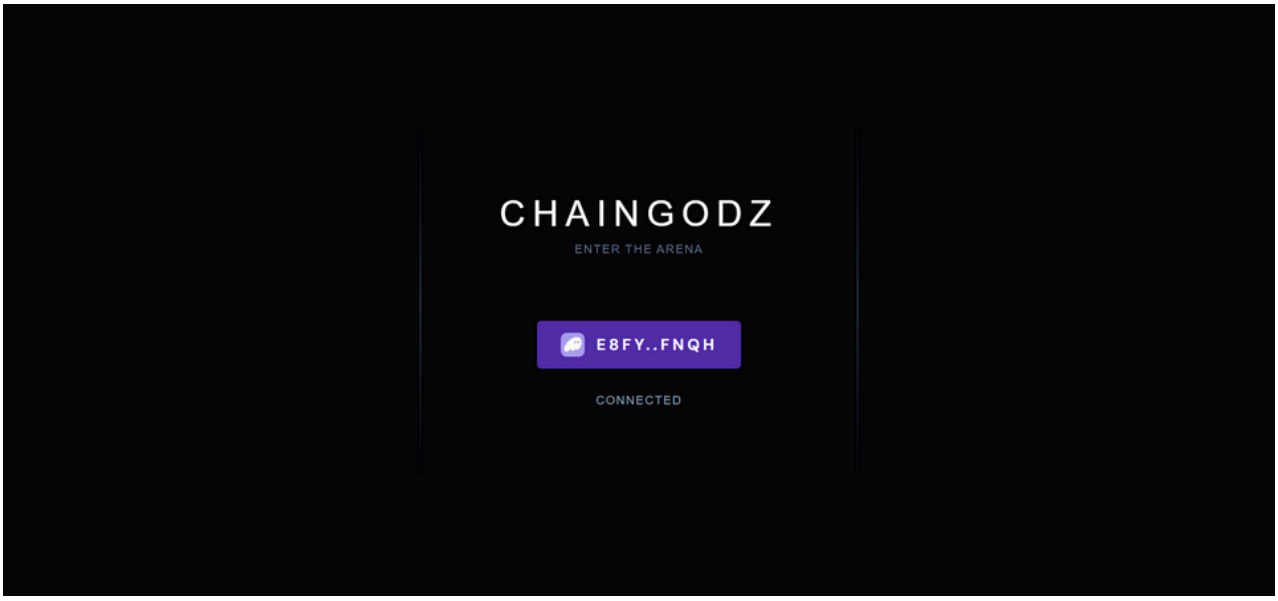
(IMAGEM 2: REPRESENTA O CÓDIGO REFERENTE AO ENTRYPOINT DO CONTRATO)



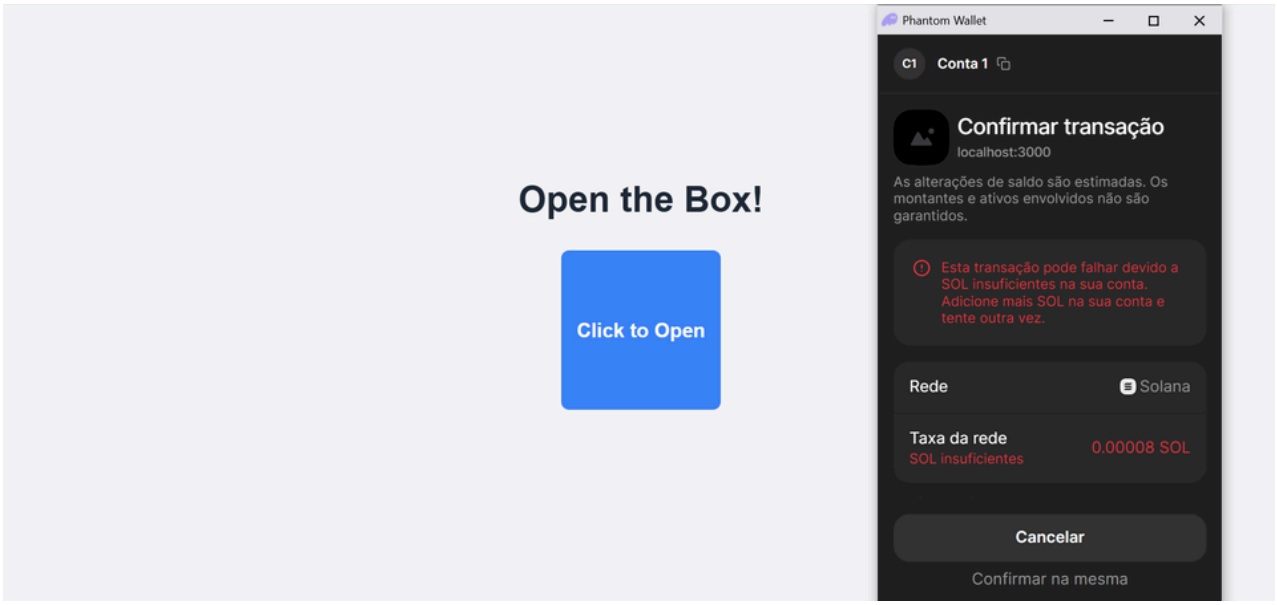

(IMAGEM 3: REPRESENTA A ESTRUTURA DO PROJETO ANCHOR)

## FRONTEND LAYER – NEXT.JS + TAILWIND + WALLET INTEGRATION

The frontend of ChainGodz is actively under development and already integrates core Web3 functionality that allows for meaningful end-to-end testing and visual feedback.The Phantom wallet is seamlessly integrated using @solana/wallet-adapter, allowing users to connect, trigger transactions, and receive responses without leaving the browser. A basic Buy Button is operational, capable of invoking the buy_item method on the smart contract. User interface elements dynamically retrieve the buyer's public key and display wallet connection status. Contract interactions are confirmed or denied via modal prompts, and failure messages are logged cleanly. The frontend is configured to interact with Solana's localnet for contract testing. Error handling mechanisms and logs assist in verifying logic correctness before mainnet deployment. While early-stage, the UI follows a modular and responsive design with utility-first CSS, ensuring future UI/UX scalability. This layer is essential for testing smart contract behavior, validating transaction results, and ensuring proper event generation—making.




(IMAGEM 4: REPRESENTA A TELA INICIAL APÓS CONEXÃO COM A CARTEIRA DO USUÁRIO)




(IMAGEM 5: REPRESENTA A REQUISIÇÃO DE UMA TRANSAÇÃO A PARTIR DA INTERFACE)