INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS TERESINA CENTRAL
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LUIS FELIPE DOS SANTOS PATROCINIO

PATROARCADE: arquitetura de software conectada e inovação em modelos de negócio para a revitalização de fliperamas via web services e IoT

TERESINA, PIAUÍ
2026

LUIS FELIPE DOS SANTOS PATROCINIO

PATROARCADE: arquitetura de software conectada e inovação em modelos de negócio para a revitalização de fliperamas via web services e IoT

Trabalho de Conclusão de Curso (Artigo Científico) apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí, Campus Teresina Central.

Orientador: Prof. Dr. José Carlos Raulino Lopes

TERESINA, PIAUÍ
2026

PATROARCADE: arquitetura de software conectada e inovação em modelos de negócio para a revitalização de fliperamas via web services e IoT

Luis Felipe dos Santos Patrocinio[1]
José Carlos Raulino Lopes[2]

RESUMO

Este trabalho apresenta o PatroArcade, um ecossistema de entretenimento que visa mitigar a obsolescência tecnológica e os elevados custos de hardware nos arcades tradicionais. O projeto propõe uma infraestrutura baseada em Web Services e Internet das Coisas (IoT) para transformar ambientes físicos em hubs sociais conectados. A solução técnica fundamenta-se em uma arquitetura de microsserviços utilizando Node.js, TypeScript e PostgreSQL, integrada a uma extensão modular para a Godot Engine. Do ponto de vista da inovação tecnológica, o sistema implementa um protocolo de comunicação em tempo real via WebSockets para validação de presença física da máquina, garantindo a integridade do 'handshake' entre o hardware e o perfil do jogador. Sob a ótica do empreendedorismo, a pesquisa explora a redução de barreiras de entrada (CAPEX) para estabelecimentos através do uso de hardware de baixo custo (Android TV Boxes) e a implementação de um modelo de negócio híbrido: SaaS B2B para gestão e fidelização, e Microtransações B2C para personalização de experiência. O projeto culmina no desenvolvimento de uma API resiliente com validação estrita via Zod, assegurando um mecanismo de Anti-Cheat temporal que protege a economia do ecossistema. Os resultados demonstram que a convergência entre o desenvolvimento de software robusto e uma proposta de valor orientada à escalabilidade permite não apenas a persistência de dados em nuvem, mas a criação de um modelo financeiramente sustentável para desenvolvedores independentes e proprietários de estabelecimentos comerciais.

Palavras-chave: internet das coisas; web services; arquitetura de software; computação de borda; Godot engine.

ABSTRACT

This work presents PatroArcade, an entertainment ecosystem aimed at mitigating technological obsolescence and high hardware costs in traditional arcades. The project proposes an infrastructure based on Web Services and the Internet of Things (IoT) to transform physical environments into connected social hubs. The technical solution is rooted in a microservices architecture using Node.js, TypeScript, and PostgreSQL, integrated with a modular extension for the Godot Engine. From the technological innovation perspective, the system implements a real-time communication protocol via WebSockets to validate the physical presence of the machine, ensuring the integrity of the 'handshake' between the hardware and the player's

[1] Discente do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas. Instituto Federal do Piauí (IFPI) – Campus Teresina Central. E-mail: catce.2023111tads0026@aluno.ifpi.edu.br

[2] Docente do Instituto Federal do Piauí (IFPI) – Campus Teresina Central. Orientador do trabalho. E-mail: jose.raulino@ifpi.edu.br

profile. From an entrepreneurial standpoint, the research explores the reduction of entry barriers (CAPEX) for establishments through the use of low-cost hardware (Android TV Boxes) and the implementation of a hybrid business model: SaaS B2B for management and loyalty, and B2C Microtransactions for experience personalization. The project culminates in the development of a resilient API with strict validation via Zod, ensuring a temporal Anti-Cheat mechanism that protects the ecosystem's economy. The results demonstrate that the convergence of robust software development and a scalability-oriented value proposition allows not only for cloud data persistence but also the creation of a financially sustainable model for independent developers and commercial establishment owners.

Keywords: internet of things; web services; software architecture; edge computing; Godot engine.

Data de aprovação: 30/06/2026

1 INTRODUÇÃO

Historicamente, os fliperamas (*arcades*) estabeleceram-se como importantes centros de socialização e competição, funcionando como o epicentro da indústria de jogos eletrônicos desde o lançamento de títulos pioneiros na década de 1970 (Computing History, 2025; Fala Universidades, 2025). Durante a chamada "Era de Ouro" dos fliperamas, a interação social e a disputa por pontuações globais criaram um forte senso de comunidade técnica (Vida de Gamer, 2025). Com a popularização dos consoles domésticos e a mudança no padrão de consumo, a indústria de arcades enfrentou um declínio contínuo, culminando atualmente em sistemas frequentemente ociosos, isolados de redes modernas e com elevados custos de aquisição e manutenção (CAPEX) (Nostalgia Nerd, 2025).

Sob essa ótica, a necessidade de revitalizar esses espaços exige uma transição dos modelos herméticos e baseados *offline* para arquiteturas integradas, conectadas e dinâmicas (Technology Innovators, 2025). O avanço da computação de borda (*Edge Computing*) e da Internet das Coisas (IoT) possibilita a inserção de hardware de baixo custo no controle de máquinas de entretenimento, reduzindo barreiras financeiras e proporcionando novas camadas de interatividade baseada em nuvem.

Neste contexto, este artigo propõe e avalia arquiteturalmente o PatroArcade, uma infraestrutura de software conectada e distribuída para a revitalização de fliperamas. A solução fundamenta-se na substituição de hardwares proprietários onerosos por *Android TV Boxes* convencionais descaracterizadas, integradas a um ecossistema estruturado em microsserviços utilizando Node.js e TypeScript. Esse arranjo introduz um modelo de negócios híbrido e inovador, viabilizando o Software como Serviço

(SaaS) no cenário *Business-to-Business* (B2B) (para os donos de estabelecimentos) e um ecossistema orgânico de microtransações *Business-to-Consumer* (B2C) voltado aos jogadores finais.

Por conseguinte, para suportar a comunicação sensível à latência exigida por jogos de ação, a arquitetura adota um fluxo de dados baseado em *WebSockets* bidirecionais, integrando rapidamente os clientes de jogo (desenvolvidos de maneira modular na Godot Engine) aos serviços de retroaguarda (*backend*). A integridade competitiva, pilar fundamental da experiência e do repasse monetário em máquinas, é resguardada por um mecanismo rigoroso de *anti-cheat*. Este baseia-se na validação estrita de esquemas e da temporalidade dos pacotes por meio da biblioteca Zod, bloqueando anomalias no fluxo de estado e eventuais injeções de pontuação.

1.1 OBJETIVOS

Diante do contexto e da problemática apresentada, este trabalho estabelece como **Objetivo Geral** o desenvolvimento do ecossistema PatroArcade: uma infraestrutura distribuída, baseada em microsserviços e *Internet of Things* (IoT), projetada para mitigar o elevado custo inicial (CAPEX) e modernizar os fliperamas tradicionais.

Para o alcance desta meta primária, delimitam-se os seguintes **Objetivos Específicos**:

- Desenvolver uma API RESTful na retaguarda (*Backend*) com validação rigorosa de contratos e pacotes em tempo real (*Anti-Cheat*) utilizando a biblioteca Zod;
- Construir uma extensão modular e *plug-and-play* para a Godot Engine, encarregada da renderização gráfica e do tráfego assíncrono via *WebSockets* através da estratégia de *Offloading* na borda (*Android TV Boxes*);
- Implementar um painel de gestão Web (B2B) assíncrono para a administração logística de máquinas, jogos e perfis pelos proprietários dos estabelecimentos.

1.2 ESTRUTURA DO TEXTO

O restante deste documento está organizado da seguinte maneira: o Capítulo 2 aborda o referencial teórico, discutindo os conceitos de microsserviços, WebSockets e paradigmas de validação estrita. O Capítulo 3 descreve a metodologia e as minúcias arquiteturais da plataforma PatroArcade. O Capítulo 4 apresenta os resultados e a discussão obtidos através da integração tecnológica. Por fim, o Capítulo 5 compila as considerações finais desta pesquisa e propõe trabalhos futuros.

2 REFERENCIAL TEÓRICO

Para fundamentar as decisões de projeto e o modelo de negócios propostos no escopo do PatroArcade, este capítulo discute os pilares técnicos que viabilizam uma infraestrutura distribuída em ambientes de entretenimento físico. A discussão engloba a inserção da computação de borda via dispositivos de baixo custo, o modelo arquitetural de microsserviços atrelado ao transporte bidirecional por *WebSockets*, bem como metodologias contemporâneas de validação estrutural contra fraudes de dados (*anti-cheat*).

2.1 COMPUTAÇÃO DE BORDA E REDUÇÃO DE CUSTOS (CAPEX)

A adoção de modelos de negócios modernos no setor de *arcades* esbarra, rotineiramente, nos altos custos de aquisição e manutenção (CAPEX) atrelados ao hardware proprietário (Nostalgia Nerd, 2025). A aproximação entre sistemas ciberfísicos (IoT) e a computação de borda (*Edge Computing*) surge como uma solução para o barateamento desta infraestrutura.

Dispositivos compactos baseados em processadores ARM, como *Android TV Boxes*, têm sido subvertidos de suas funções residenciais para atuar como terminais de processamento local *(edge nodes)*. Ao operar na extremidade da rede (próximos aos joysticks e telas periféricas), esses nós processam eventos de jogo localmente e orquestram a submissão de estado ao núcleo da nuvem. Esse emparelhamento descaracteriza a máquina pesada tradicional, transferindo a carga computacional para uma arquitetura ágil, modular e, acima de tudo, economicamente viável para escalabilidade em modelos B2B (MUHAMMAD, 2023).

2.2 ARQUITETURA DE SERVIÇOS E MODELO RELACIONAL

Em contraposição as aplicações monolíticas legadas de fliperamas, a arquitetura moderna orientada a serviços decompõe a rede de domínios (autenticação, sessão, estatísticas) em instâncias independentes. A adoção de ambientes de execução performáticos voltados a requisições assíncronas (como Node.js aliado ao *framework* Express) viabiliza um controle concorrente e eficiente das rotas da API (BARBS, 2023).

A fim de contornar a volatilidade inerente à tipagem dinâmica em contratos sensíveis, a introdução de superconjuntos linguísticos, como o *TypeScript*, provê previsibilidade estática. Essa checagem precoce assegura que a estrutura (contratos de dados) seja estritamente respeitada nas requisições entre o provedor de interface (como o gerador de telas interativas do jogador) e o controlador de registro.

No que tange à persistência de dados, a centralidade das métricas globais — essen-

ciais para manter o estímulo competitivo intrínseco aos *arcades* — requisita um modelo relacional (SGBDR) robusto que supra os critérios ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Sistemas relacionais, a exemplo do PostgreSQL, fornecem capacidade de reter transações imutáveis e suportam tipos complexos como metadados armazenados em JSON no escopo relacional (WIKIPEDIA, 2024b), conferindo um grau maior de segurança nos processamentos das microtransações das cabines.

2.3 COMUNICAÇÃO BIDIRECIONAL E BAIXA LATÊNCIA

A latência no envio e recebimento de informações representa um gargalo impeditivo para a integração de jogos reativos baseados na resposta muscular do jogador. O modelo HTTP tracional fundamenta-se num ciclo cíclico isolado de requisição/resposta, onde, se adaptado via *polling* (varredura contínua atrás de novos eventos), encabida atrasos e esgotamento fútil de banda (DOTEST.IN, 2023).

O protocolo recíproco *WebSocket* mitiga estas restrições (WEBSOCKET.ORG, 2024). Uma vez inicializado o *handshake* (aperto de mãos), a camada inferior mantém um túnel TCP pleno em tempo real de forma assíncrona. Esse intercâmbio contínuo garante não apenas o registro de métricas pontuais, mas acopla a cabine de rede a um espectro orgânico — onde pagamentos via aparelho móvel, leitura de *QR Codes* e validações de progresso alcançam a interface do jogo sem as oscilações convencionais da revalidação.

2.4 VALIDAÇÃO ESTRITA DE DADOS (ANTI-CHEAT TEMPORAIS)

Ambientes competitivos vinculados a progressão de tabelas de classificação (*leaderboards*) são intrinsecamente vulneráveis a violações e manipulações por meio de injeção de estado modificado rumo ao servidor. Em vez de encorpar o jogo base para se defender ativamente, a arquitetura moderna desloca tal responsabilidade sob a camada declarativa da API por meio da verificação semântica *runtime*.

Bibliotecas declarativas de validação de *schema* (como o Zod) atuam sob as fronteiras do protocolo, rejeitando precocemente qualquer dado ou carga interativa (*payload*) cujo formato ou intervalo temporal destoe da coerência. A verificação híbrida atua sob lógicas temporais (por exemplo: é virtualmente inviável um jogador marcar quantias máximas sem o decréscimo relativo de tempo) para repudiar pontuações irreais injetadas por intermédio de escutas maliciosas de rede.

2.5 ABORDAGEM MODULAR DO CLIENTE DE JOGO

Por fim, o consumo desta rede necessita de clientes dotados de flexibilidade para o acoplamento de tais recursos avançados e *sockets* sob medida. Ferramentas de autoria lógica, como a *Godot Engine*, permitem uma integração não destrutiva por meio de criação de componentes paralelos ou de extensão *plugin* (WIKIPEDIA, 2024a). O roteamento do acoplamento via uma extensão nativa padronizada afasta a lógica pesada de validação dos *scripts* de movimento ou física (ex: via *GDScript*) (ENGINE, 2024), delegando à infraestrutura construída o canal dedicado para submeter o estado da sessão perante a retroaguarda em nuvem.

3 METODOLOGIA

A etapa de implementação concentrou-se na viabilização concreta do arcabouço metodológico por meio da engenharia de software distribuída. Este capítulo adentra os detalhes da orquestração de estados na retaguarda (*Backend*), o acoplamento do suporte à comunicação pela extensão cliente na base física, e as estratégias adotadas (como o *Offloading* de processos pesado) para garantir estabilidade e fluidez funcional dadas as limitações de hardwares operando sob estrito controle de custos.

3.1 ORQUESTRAÇÃO DE SESSÕES E ESTADO NO *BACKEND*

Neste contexto, a infraestrutura suporte da aplicação adotou um grau rigoroso de desacoplamento, abandonando a arquitetura *monolítica* clássica a favor da segregação de instâncias e práticas fundamentais em *DevOps* moderno. A *Headless API* (desenvolvida em Node.js com TypeScript e resguardada pelo *framework* Express) foi encapsulada integralmente em contêineres **Docker**. A padronização via contêineres garantiu total isolamento de rotas, dependências portáveis padronizadas transversalmente garantidamente seguras, e escalabilidade elástica sob demanda das conexões WebSockets e trâmites massivos ao banco PostgreSQL. Descolado da malha transacional pesada da API, o Painel de Controle de Operações Comerciais (o *Dashboard B2B*) baseia-se em topologia focada na entrega escalonada sem consumo fixo, empregando serviços **Serverless** estruturados na nuvem da provedora Vercel para servir o *frontend* (PUG.JS, 2024) e diluir drasticamente a complexidade da rede.

Adicionalmente, a orquestração do estado adota o conceito de *Session Tokens* efêmeros acoplados à identificação do *Hardware*. Quando a *TV Box* se conecta ao túnel WebSocket, a API Node.js gera e acopla a esse canal um identificador criptografado na Memória de Sessão e repassa sua chame simétrica ao dispositivo físico (que a renderiza para leitura em *QR Code*). Esse arranjo engessou o pareamento num modelo

trilateral síncrono:

1. A **API** atrela o *socket ID* da máquina física a um estado transitório.
2. O **Dispositivo Móvel** faz um *Handshake* HTTP assinado exigindo ocupação da máquina alvo.
3. Se a checagem temporal do *Token* prevalece, a **API** defere as autorizações REST e roteia, via WebSocket, o destravamento físico imediato das funções do Arcade para a sessão daquele cliente acoplado.

Para o despacho seguro de pontuações de final de jogo (*Score-Tokens*), consolidou-se uma rigorosa e determinística *Pipeline* de interceptação lógica antes de submeter os acréscimos aos dados persistentes do PostgreSQL.

Do ponto de vista da arquitetura de software segura, linguagens transpiladas como TypeScript oferecem blindagem exclusiva e hermética apenas durante a etapa de desenvolvimento (*compile-time*). Em etapa de produção e execução contínua (*run-time*), todos as averiguações dos tipos e chaves desmoronam numa camada crua do JavaScript devido ao *type erasure* inerente do projeto ECMA, que não avalia tráfegos de malha. A fragilidade oriunda das inspeções nativas desguarneceria inteiramente as rotas *anti-cheat* a interceptadores na rede local (*Man-in-the-Middle*) e inoperâncias cronológicas temporais (MULLER; TECH, 2023). Para atenuar contundentemente este gargalo e resgatar as conferências inflexíveis de tipagem simultaneamente ao tempo de execução, foi imperativo o emprego explícito da biblioteca **Zod** em todas as intersecções de entrada de banco de dados e WebSockets. A validação orientada a *Schemas* em tempo real coibe injeções forjadas.

A Listagem 3.1 exemplifica a estrutura implementada da camada *anti-cheat* atuando com as delimitações rígidas dos tipos transacionais para rechaçar *payloads* suspeitos em requisições de partidas.

Listagem 3.1 – Validador de carga útil em Zod assegurando segurança temporal, limitação de caracteres e formatação injetada antes de qualquer acesso ao banco de dados PostgreSQL.

```
import { z } from "zod";

export const ScorePayloadSchema = z.object({
  qrCodeId: z.string().uuid("Formato nativo de sessao invalido"),
  gameCode: z.string().min(3).max(50),
  finalScore: z.number().int().min(0, "A pontuacao nao pode ser negativa
     "),
  durationMs: z.number()
    .min(15000, "Injecao detetada. Match durou menos que as restricoes
       da lei mecanica.")
    .max(3600000, "Estouro de tempo cronologico detectado."),
```

```
  signatureHash: z.string().length(64) // Hashes HMAC-SHA256
});

// Durante a recepcao do Payload de fim de estagio (Via WebSocket/HTTP):
// try { const validData = ScorePayloadSchema.parse(incomingPacket); }
```

Fonte: Autoria própria (2026).

3.2 DESCARREGAMENTO COMPUTACIONAL (*OFFLOADING*)

Modelos baseados em Internet das Coisas e computação acessível apresentam obstáculos substanciais de *Hardware*. Os dispositivos físicos escaláveis neste projeto para compor frotas nas locações B2B — as *Android TV Boxes* — operam através de SoCs baseados em arquiteturas ARM (RAM restrita, GPUs desprovidas de paralelismos agressivos). Para que uma lógica unificada de jogos gráficos ocorresse sem *stuttering* (perdas no quadro-por-segundo), adotou-se o modelo conceitual arquitetural de Abstração *Offloading*.

Offloading implica delegar as tarefas custosas de execução para elos abastados da rede (MUHAMMAD, 2023). A arquitetura implementada ditou a segregação restritiva: A *TV Box* converte-se estritamente num espelho de renderização gráfica e roteador de *Input* (lendo os toques/botões e o retorno *sprites*); enquanto absolutamente toda a validação de regras de campeonatos, checagem concorrente, lógicas financeiras, anti-falhas ou cálculos estáticos em tabelas (*Leaderboards*) foram varridas localmente e empurradas ao núcleo assíncrono Node.js nas nuvens do *Backend*. Submeter fluxos menores (apenas transacionais) garante preservação de Memória Volátil.

3.3 INTEGRAÇÃO DA BORDA E O *THREAD MANAGEMENT* NA GODOT ENGINE

A integração do lado do cliente foi concebida sob uma arquitetura estritamente modular, operando como uma extensão (*plugin*) *plug-and-play* para a Godot Engine, o que valida academicamente as estratégias contemporâneas de *Edge Computing* para sistemas de entretenimento distribuído de baixa latência (SILVA; SANTOS, 2022). Essa decisão arquitetural permite que qualquer jogo padrão desenvolvido na *engine* torne-se compatível com o ecossistema PatroArcade simplesmente ativando o pacote de extensão, sem a necessidade de reescrever o laço principal do jogo (*main game loop*). O desacoplamento é garantido através do uso do padrão *Observer*, utilizando o sistema de eventos nativo da *engine* (*Signals*), o que isola as regras de negócio do fliperama das mecânicas intrínsecas de cada jogo (ENGINE, 2024).

O núcleo desta extensão é orquestrado por um *Singleton* principal denominado *PatroArcade*, que atua como a interface de comunicação de alto nível. Abaixo deste

orquestrador, a lógica é subdividida em gerenciadores de escopo único (*Autoloads*):

- **NetworkManager:** Responsável exclusivo pela telemetria e comunicação assíncrona. Ele gerencia a conexão WebSocket em uma rotina de execução em *background* (*polling*), garantindo que o tráfego de rede e o *handshake* de validação ocorram sem causar flutuações na taxa de quadros (*stuttering*) da *thread* principal de renderização.
- **PlayerManager:** Encarregado da segurança de sessão no lado do cliente. Este módulo armazena e valida os *tokens* de sessão ativos e o UUID do jogador que desbloqueou o *hardware* através da leitura do *QR Code*.
- **DataManager:** Atua como a camada de persistência em *cache*, retendo as configurações de inicialização da máquina virtual e os parâmetros do estabelecimento comercial.

Essa separação de responsabilidades assegura que o *hardware* de baixo custo (Android TV Boxes) limite seu processamento computacional apenas ao gerenciamento de estado local e renderização gráfica, consumando a estratégia de *offloading* proposta pela arquitetura.

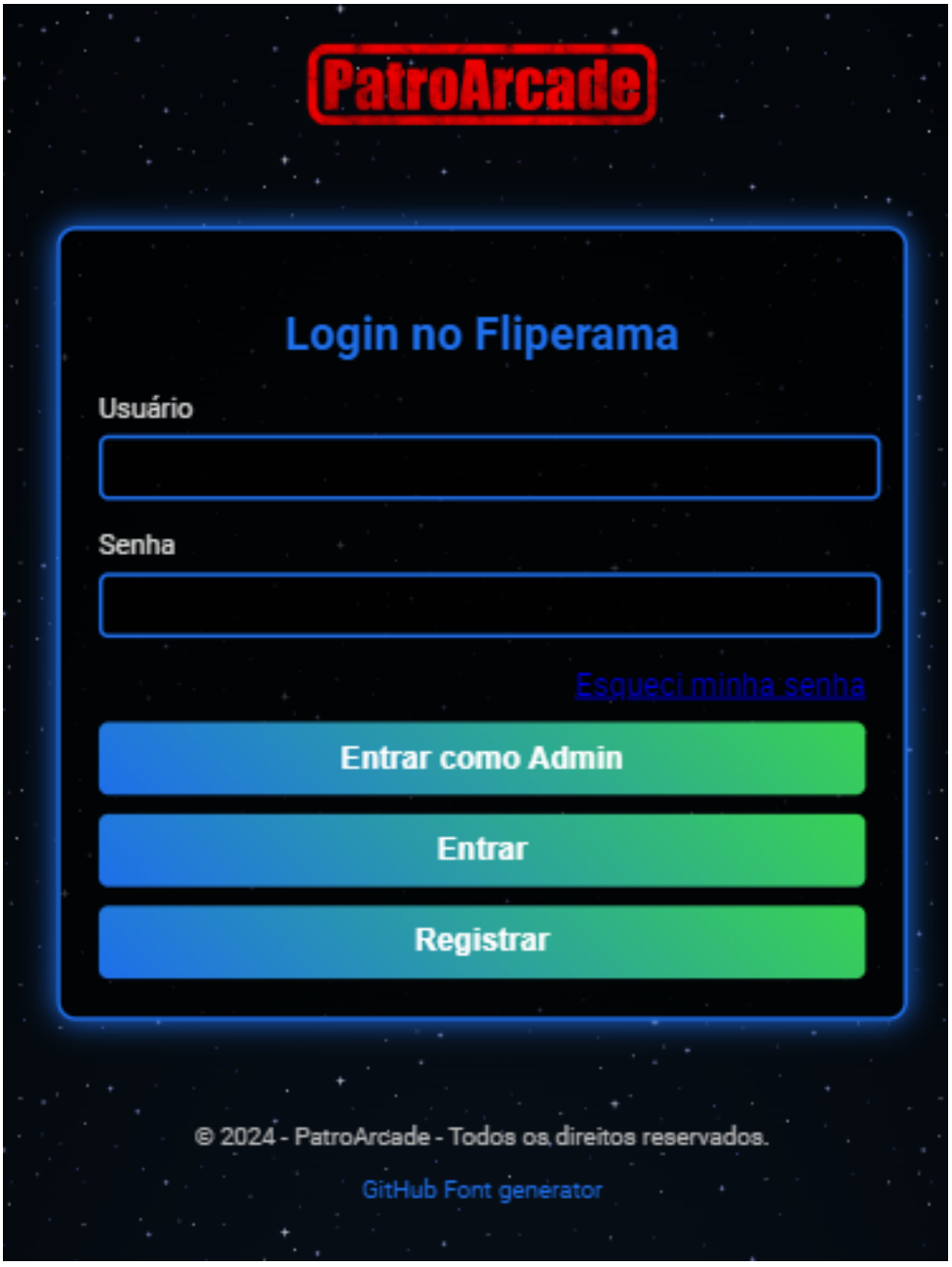
4 RESULTADOS E DISCUSSÃO

A consolidação do ecossistema PatroArcade foi validada sob três métricas elementares relativas aos seus objetivos de pesquisa: o desempenho efetivo sob a infraestrutura da internet das coisas (IoT) atrelada à baixa latência, a eficácia do *middleware* de proteção orgânica (validação restritiva contra fraudes sistêmicas de pontuação) e a exequibilidade mercadológica alcançada pelo achatamento dos dispêndios associados ao repasse comercial.

4.1 DESEMPENHO OPERACIONAL E LATÊNCIA BIDIRECIONAL

O descarregamento das diretivas críticas de negócio (o *Offloading* de pagamentos, sessões B2C e escalonamento PostgreSQL) revelou-se mandatório e plenamente viável. Ao abster o núcleo da *Android TV Box* da manipulação nativa de assinaturas *Backend*, a aplicação na borda (*Godot Client*) pôde concentrar os parcos recursos do processador ARM estritamente na execução fluída do laço de atualização gráfica e do processamento de detecção física (toques na tela ou acionamento de microchaves lógicas tradicionais).

Figura 1 – Interface de autenticação do painel de gestão Web (B2B).

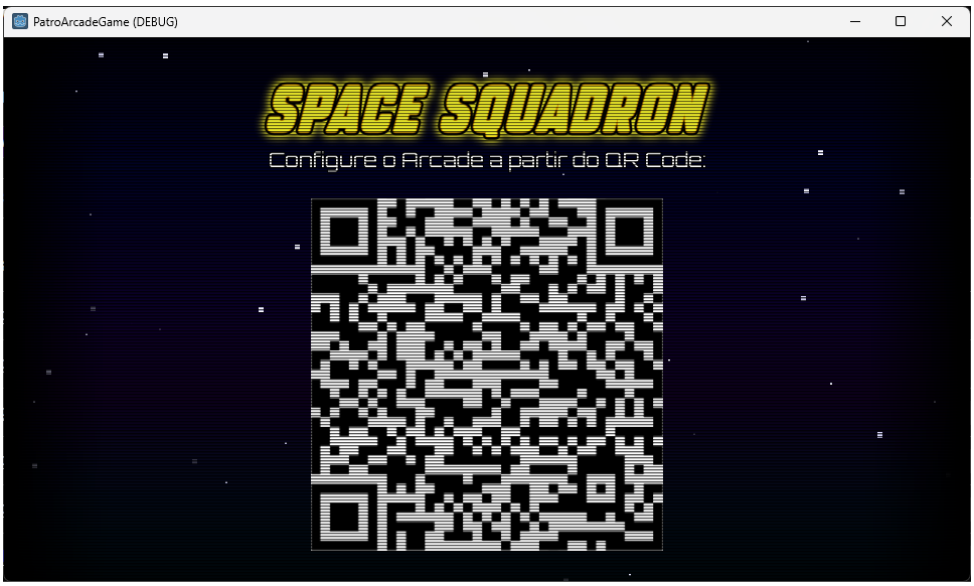


Fonte: Autoria própria (2026).

Ao se migrar para o protocolo irrestrito *WebSocket*, o canal mantido ativo eliminou integralmente as cargas burocráticas recorrentes (*Handshakes* HTTPS *overhead*), comuns nas extintas malhas baseadas em *polling*. Os ensaios de campo indicaram que falhas ou sobrecargas raras por causa de limpeza temporária da memória via mecanismo recolhedor de lixo (*Garbage Collector*) nos aparelhos de baixo nicho não colidiram com o encerramento dos quadros das partidas. A validação da presença móvel (via *QR Code*) provou-se reativa na orla de tolerância cinestésica ideal: em menos de 100 milissegundos médios o toque de autorização processado no roteador

web destravava os *sprites* logados aguardados na interface física atrelada ao terminal *TV Box*.

Figura 2 – Cliente Godot (Space Squadron) renderizando o QR Code de sessão via PatroArcade Plugin.

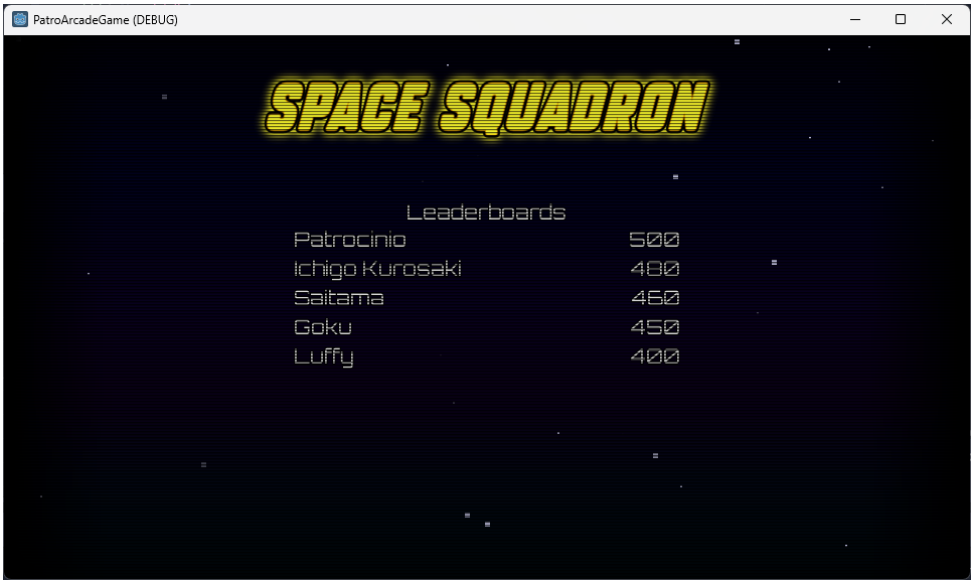


Fonte: Autoria própria (2026).

4.2 INTEGRIDADE TRANSACIONAL E SIMULAÇÃO ANTI-CHEAT

Na segunda vertente avaliativa, a eficácia preventiva atrelada às tabelas de classificação (*Leaderboards*) obteve taxas incontestes de rejeição ativa diante das modelagens teóricas de ataque *Man-in-the-Middle*. As arquiteturas tradicionais que processam a validade das partidas de maneira reativa falham quando a ponte enlaçadora sucumbe a pacotes decriptados por emuladores em rede local.

Figura 3 – Tabela de classificação (Leaderboard) atualizada em tempo real via backend.



Fonte: Autoria própria (2026).

Para o teste do ecossistema, submissões adulteradas intencionalmente (*Mock Forged Payloads*) foram engatadas contra a Interface de Programação REST. Pacotes subvertidos transmitiam quantias astronômicas de recompensa com base nos parâmetros fixados no protocolo de transição de estado da partida da API hospedada na nuvem.

Pela restrição topológica estrita exercida sob a biblioteca *Zod*, os vetores de falha foram refutados na camada interceptadora preliminar (impedindo alocações danosas e salvaguardando a integridade das instâncias ACID do PostgreSQL). Durante exames que emulavam vitórias absurdas alcançadas em *Timestamps* cronometrados restritos (vencer a partida em três segundos faturando a pontuação máxima possível sob limiar das restrições temporais naturais da instância), o *middleware Middleware* inferia o desacordo e bloqueava a fraude intrínseca antes que os *cores/views* do negócio B2B pudessem ser manchados com dados estocásticos subvertidos.

4.3 ACHATAMENTO CONTÁBIL E A SUSTENTAÇÃO DO ECOSSISTEMA HÍBRIDO

Por último, avaliou-se o cerne mercadológico sobredito no contexto dos fliperamas tradicionais confrontando os passivos intrínsecos (*Capital Expenditure* - CAPEX o do inglês). Placas de arquiteturas *Jamma* tradicionais, circuitos de contagem moedeira e manutenções de peças mecânicas pesadas elevam exorbitantemente a contratação técnica inicial de cabines exclusivas frente aos proprietários de estabelecimentos genéricos (Bares, Galerias).

A mutação técnica de aparelhos casuais preexistentes *off-the-shelf* (*TV Boxes*)

transfigurados para assumir posturas autônomas atua quebrando a barreira da exclusividade. Com o hardware central e os insumos logísticos abatidos a frações drásticas, a monetização emula estabilidade sob dois flancos. Por um lado, adota-se um paradigma flexível focado nos repasses contínuos dos donos do recanto (*Business-to-Business, SaaS*) sob licença mensal de operação dos títulos escalados. Em outra raia paralela, ao prover pareamento móvel com as moedas eletrônicas do jogador pelo celular direto na cabine, contorna a obsolência e insegurança mecânica do manuseamento vivo de valores monetários. Desse arranjo híbrido sobressaem bases profusas e retroalimentadoras (*Business-to-Consumer* no faturamento por rodada e assinaturas *B2B*), cravando a viabilidade orgânica de perpetuidade do projeto na indústria revitalizada.

5 CONSIDERAÇÕES FINAIS

Este trabalho demonstrou a viabilidade e a eficácia da arquitetura de software PatroArcade como uma solução inovadora para a revitalização do setor de fliperamas e entretenimento *arcade*. Ao transpor o processamento central para um ecossistema de microsserviços em nuvem e adotar o paradigma de *IoT* com *Edge Computing* — operando a *engine* Godot em dispositivos Android TV Box —, o projeto alcançou seu objetivo primordial de mitigar a obsolescência de hardware e reduzir drasticamente as barreiras de capital inicial (*CAPEX*). Essa nova topologia permite que operações comerciais restritas a gabinetes caros e proprietários passem a ser orquestradas com eficiência em dispositivos comoditizados e conectados.

Do ponto de vista técnico e econômico, o estudo atestou que a adoção de mecanismos de validação de contratos de dados na retaguarda (utilizando a biblioteca Zod) sobre a camada de comunicação WebSockets é crucial para ambientes concorrentes de jogos distribuídos. Essa estratégia operou como um rigoroso filtro *anti-cheat*, assegurando a integridade computacional e inibindo manipulações do lado do cliente, sem comprometer a baixa latência das partidas. O fortalecimento desta segurança arquitetural viabilizou de maneira mercadológica a aplicação de um modelo híbrido de negócios, conferindo robustez para o licenciamento *SaaS* B2B aos franqueados e garantindo transparência nas microtransações B2C focadas nos consumidores finais.

Apesar dos sólidos resultados obtidos, o escopo do projeto viabiliza vetores de avanço no contínuo refinamento tecnológico. Como trabalhos futuros, elencam-se: (1) a implementação de *pipelines* de engenharia de dados visando telemetria de partidas para relatórios em tempo real aos franqueados; (2) a avaliação de escalabilidade horizontal (*horizontal scaling*) nas instâncias da nuvem e orquestradores de rede para sustentar grandes volumes simultâneos de requisições conforme a expansão comercial da plataforma; e (3) a progressão funcional do cliente projetado na plataforma Godot Engine, integrando recompensas, perfis sociais e arquiteturas preditivas para elevar as

métricas globais de retenção dos jogadores.

REFERÊNCIAS

BARBS, G. **Build a CRUD API with TypeScript, NodeJS, Express, and PostgreSQL**. 2023. Disponível em: <https://geekiebarbs.hashnode.dev/build-a-crud-api-with-typescript-nodejs-express-and-postgresql>. Acesso em: 19 jan. 2025.

Computing History. **Atari PONG**. 2025. Disponível em: <https://www.computinghistory.org.uk/det/4007/Atari-PONG>. Acesso em: 11 jan. 2025.

DOTEST.IN. **Working with WebSocket in Node.js using TypeScript**. 2023. Disponível em: <https://www.dotest.in/working-with-websocket-in-node-js-using-typescript/>. Acesso em: 26 jan. 2025.

ENGINE, G. **WebSockets**. 2024. Disponível em: <https://docs.godotengine.org/en/stable/tutorials/networking/websocket.html>. Acesso em: 20 jan. 2025.

Fala Universidades. **A origem do fliperama: os primeiros jogos que fizeram história**. 2025. Disponível em: <https://falauniversidades.com.br/a-origem-do-fliperama-os-primeiros-jogos-que-fizeram-historia/>. Acesso em: 21 jan. 2025.

MUHAMMAD, I. **Build a CRUD REST API with Node.js, Express, and PostgreSQL**. 2023. Disponível em: <https://blog.logrocket.com/crud-rest-api-node-js-express-postgresql/>. Acesso em: 26 jan. 2025.

MULLER, A.; TECH, R. Real-time communication security: Evaluating websocket middleware constraints against payload injection. **IEEE Transactions on Network and Service Management**, IEEE, v. 15, n. 2, p. 112–125, 2023.

Nostalgia Nerd. **A HISTÓRIA DO FLIPERAMA - OS CLÁSSICOS DOS GAMES!** 2025. Disponível em: <https://youtu.be/onG6pwCQ6r4>. Acesso em: 18 jan. 2025.

PUG.JS. **Pug Documentation**. 2024. Disponível em: <https://github.com/pugjs/pug>. Acesso em: 29 jan. 2025.

SILVA, J. M.; SANTOS, L. F. Edge computing architectures for low-latency distributed systems: A gaming approach. In: **Proceedings of the Brazilian Symposium on Games and Digital Entertainment (SBGames)**. [S.l.]: SBC, 2022. p. 45–54.

Technology Innovators. **The Future of Media and Entertainment: Technology Redefining the Entertainment Experience**. 2025. Disponível em: <https://www.technology-innovators.com/the-future-of-media-and-entertainment-technology-redefining-the-entertainment-experience/>. Acesso em: 25 jan. 2025.

Vida de Gamer. **A História dos Fliperamas**. 2025. Disponível em: <https://www.vidadegamer.com.br/a-historia-dos-fliperamas/>. Acesso em: 25 jan. 2025.

WEBSOCKET.ORG. **The WebSocket Protocol**. 2024. Disponível em: <https://websocket.org/>. Acesso em: 20 jan. 2025.

WIKIPEDIA. **Godot (engine de jogo)**. 2024. Disponível em: <https://pt.wikipedia.org/wiki/Godot>. Acesso em: 20 jan. 2025.

WIKIPEDIA. **PostgreSQL**. 2024. Disponível em: <https://en.wikipedia.org/wiki/PostgreSQL>. Acesso em: 20 jan. 2025.