Relato´rio Te´cnico: Dueling Protocol

## Lucas Damasceno

1Universidade Estadual de Feira de Santana (UEFS) Av. Transnordestina, s/n, Novo Horizonte

Feira de Santana – BA, Brasil – 44036-900

[lucas.damasceno.dev@gmail.com](mailto:lucas.damasceno.dev@gmail.com)

***Resumo.*** *O mercado de jogos multiplayer online tem crescido exponencial- mente, demandando arquiteturas de servidor cada vez mais robustas e de baixa lateˆncia para garantir uma experieˆncia de usua´rio competitiva e justa. Este re- lato´rio detalha o ”Dueling Protocol”, uma soluc¸a˜o de backend para um jogo de cartas 1v1, projetada para ser escala´vel, concorrente e confia´vel. A arquite- tura cliente-servidor foi implementada em Java, utilizando comunicac¸a˜o nativa via sockets TCP para ac¸o˜es cr´ıticas de jogo e UDP para um sistema de ping, visando a monitorac¸a˜o de lateˆncia. A persisteˆncia de dados dos jogadores e´ realizada de forma leve e leg´ıvel atrave´s de arquivos JSON. A lo´gica de nego´cio foi centralizada utilizando o padra˜o Facade, e a consisteˆncia em um ambiente de alta concorreˆncia foi garantida pelo uso de estruturas de dados thread-safe. Para validar a soluc¸a˜o, uma su´ıte de testes automatizados foi desenvolvida com Docker e Docker Compose, permitindo a simulac¸a˜o de mu´ltiplos clientes e a identificac¸a˜o de condic¸o˜es de corrida. Testes de estresse demonstraram a capa- cidade do servidor de gerenciar dezenas de conexo˜es simultaˆneas sem falhas, e os testes de cena´rio foram cruciais para identificar e corrigir uma condic¸a˜o de corrida no sistema de matchmaking, validando a robustez da implementac¸a˜o final.*

# Introduc¸a˜o

Com a ascensa˜o das plataformas de distribuic¸a˜o digital, o mercado de jogos independen- tes (\*indie\*) tornou-se um campo fe´rtil para a inovac¸a˜o. No entanto, para se destacar, especialmente no geˆnero multiplayer, e´ imperativo que a infraestrutura de backend seja performa´tica e confia´vel. Um servidor lento ou insta´vel pode arruinar a experieˆncia de jogo e afastar a base de jogadores. O desafio proposto neste projeto foi o desenvolvi- mento do backend para um jogo de cartas ta´tico 1v1, que exigia pareamento de joga- dores, comunicac¸a˜o bidirecional em tempo real e a gesta˜o de mu´ltiplas sesso˜es de jogo simultaˆneas.

Para resolver este problema, foi criado o ”Dueling Protocol”, um servidor de jogo robusto e concorrente. A arquitetura da soluc¸a˜o tem como ponto de entrada o GameServer, que aceita novas conexo˜es de clientes e instancia um ClientHandler dedicado para cada um, adotando o modelo ”uma thread por cliente”. Cada

ClientHandler e´ responsa´vel por toda a comunicac¸a˜o com um cliente espec´ıfico,

processando seus comandos. As operac¸o˜es e a lo´gica de jogo sa˜o centralizadas no GameFacade, que atua como um orquestrador, delegando tarefas a servic¸os especiali- zados, como o ConcurrentMatchmakingService para o pareamento de jogadores

e o StoreServiceImpl para a loja de pacotes de cartas. Essa abordagem desacopla a lo´gica de rede da lo´gica de nego´cio, facilitando a manutenc¸a˜o e a escalabilidade do sistema.

Este relato´rio esta´ organizado da seguinte forma: a Sec¸a˜o 2 detalha a

fundamentac¸a˜o teo´rica por tra´s das tecnologias e conceitos utilizados. A Sec¸a˜o 3 des- creve a metodologia de implementac¸a˜o, alinhada com os requisitos do projeto. A Sec¸a˜o 4 apresenta e analisa os resultados dos testes executados. Finalmente, a Sec¸a˜o 5 conclui o trabalho, recapitulando os resultados e sugerindo direc¸o˜es para trabalhos futuros.

# Fundamentac¸a˜o Teo´rica

Esta sec¸a˜o aborda os conceitos essenciais que serviram de alicerce para o desenvolvi- mento do Dueling Protocol.

## Arquitetura Cliente-Servidor

O modelo cliente-servidor e´ um paradigma de arquitetura de software que distribui as

responsabilidades entre dois processos independentes: o servidor, que proveˆ recursos ou servic¸os, e o cliente, que os solicita [T[anenbaum and Wetherall 2011].](#_bookmark8) Em jogos online, o servidor geralmente atua como a autoridade central, gerenciando o estado do jogo, va- lidando as ac¸o˜es dos jogadores e sincronizando os eventos entre eles. Esta centralizac¸a˜o e´ crucial para prevenir trapac¸as e garantir que todos os jogadores tenham uma visa˜o con- sistente do mundo do jogo. O Dueling Protocol adota este modelo, onde o servidor Java dete´m toda a lo´gica de nego´cio e o estado das partidas.

## Comunicac¸a˜o de Rede com Sockets

Sockets sa˜o a interface fundamental para a programac¸a˜o de rede, representando um ponto de extremidade em uma comunicac¸a˜o bidirecional entre dois programas na rede [[Kurose and Ross 2016].](#_bookmark6) A API de sockets permite que um programa se conecte a ou- tro, envie e receba dados. No Dueling Protocol, a biblioteca de sockets nativa do Java (java.net.Socket e java.net.ServerSocket) foi utilizada para estabelecer a comunicac¸a˜o entre o cliente e o servidor, fornecendo um controle de baixo n´ıvel sobre o fluxo de dados.

## Protocolos TCP e UDP

Os protocolos TCP (Transmission Control Protocol) e UDP (User Datagram Protocol) sa˜o os dois principais protocolos da camada de transporte da Internet. O TCP e´ orien-

tado a` conexa˜o e oferece entrega de dados confia´vel e ordenada. Em contrapartida, o

UDP na˜o possui garantia de entrega, o que o torna mais ra´pido e com menor sobrecarga [[Kurose and Ross 2016].](#_bookmark6) No projeto, o TCP foi escolhido para as ac¸o˜es cr´ıticas do jogo (como jogar uma carta), onde a perda de um comando e´ inaceita´vel. O UDP, por sua vez, foi utilizado para o PingServer, um servic¸o secunda´rio que permite medir a lateˆncia do cliente de forma ra´pida.

## Multithreading em Java

Multithreading e´ a capacidade de um processo de executar mu´ltiplas threads concorren- temente. Em um servidor, isso e´ essencial para atender a va´rios clientes simultaneamente

sem que um tenha que esperar o outro ser atendido [[Goetz et al. 2006].](#_bookmark5) O Dueling Pro- tocol implementa o modelo ”uma thread por cliente”, onde o GameServer cria uma nova thread para cada ClientHandler. Embora simples, este modelo e´ eficaz para um nu´mero moderado de conexo˜es e isola o trabalho de cada cliente.

## Serializac¸a˜o de Dados com JSON

JSON (JavaScript Object Notation) e´ um formato leve e leg´ıvel por humanos para in-

tercaˆmbio de dados. Sua simplicidade o tornou um padra˜o para a serializac¸a˜o de objetos e comunicac¸a˜o em APIs [[Bray 2017].](#_bookmark4) No projeto, o JSON foi utilizado para a persisteˆncia dos dados dos jogadores no PlayerRepositoryJson, permitindo que os dados se- jam salvos em disco de forma simples.

## Virtualizac¸a˜o com Conteˆineres (Docker)

Conteˆineres sa˜o uma forma de virtualizac¸a˜o que permite empacotar uma aplicac¸a˜o e suas dependeˆncias em uma unidade isolada e porta´til [[Merkel 2014].](#_bookmark7) Docker foi fundamental para criar um ambiente de teste consistente. Utilizando o Docker Compose, foi poss´ıvel orquestrar a execuc¸a˜o de mu´ltiplas instaˆncias de clientes e do servidor com um u´nico comando, automatizando a su´ıte de testes.

# Metodologia

Esta sec¸a˜o detalha a implementac¸a˜o te´cnica do Dueling Protocol.

## Arquitetura

A arquitetura do sistema foi projetada para ser modular e escala´vel, como ilustrado na Figura [1.](#_bookmark0)

* + - **GameServer**: Ponto de entrada que escuta por conexo˜es e instancia um

ClientHandler por cliente.

* + - **ClientHandler**: Gerencia a comunicac¸a˜o com um u´nico cliente, fazendo o parsing dos comandos e delegando para o GameFacade.
    - **GameFacade**: Orquestra a lo´gica de nego´cio, atuando como uma fachada para os

servic¸os de matchmaking, loja e persisteˆncia.

* + - **Servic¸os**: Componentes como ConcurrentMatchmakingService e

StoreServiceImpl implementam lo´gicas de nego´cio espec´ıficas.

* + - **Reposito´rios**: A camada de persisteˆncia, com PlayerRepositoryJson e

CardRepository, abstrai o acesso aos dados.

## Comunicac¸a˜o e API Remota

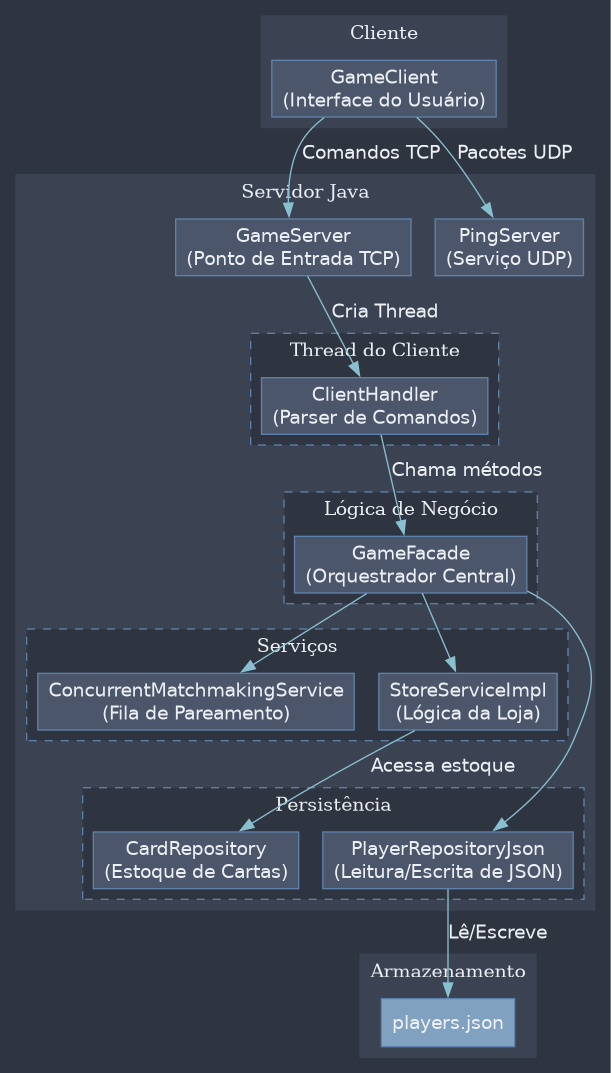
A comunicac¸a˜o utiliza uma API textual sobre TCP, com comandos no formato

COMANDO:ARG1:ARG2: A Tabela [1](#_bookmark1) detalha os principais comandos da API.

O diagrama de sequeˆncia na Figura [2](#_bookmark2) ilustra o fluxo de mensagens para a operac¸a˜o de matchmaking.

## Encapsulamento e Concorreˆncia

O ClientHandler valida e faz o parsing dos comandos, rejeitando mensagens mal- formadas. Para a concorreˆncia, o sistema utiliza estruturas de dados thread-safe, como ConcurrentLinkedQueue para a fila de matchmaking e ConcurrentHashMap para gerenciar clientes e jogos ativos, garantindo acesso seguro por mu´ltiplas threads.



**Figura 1. Diagrama de componentes da arquitetura do Dueling Protocol.**

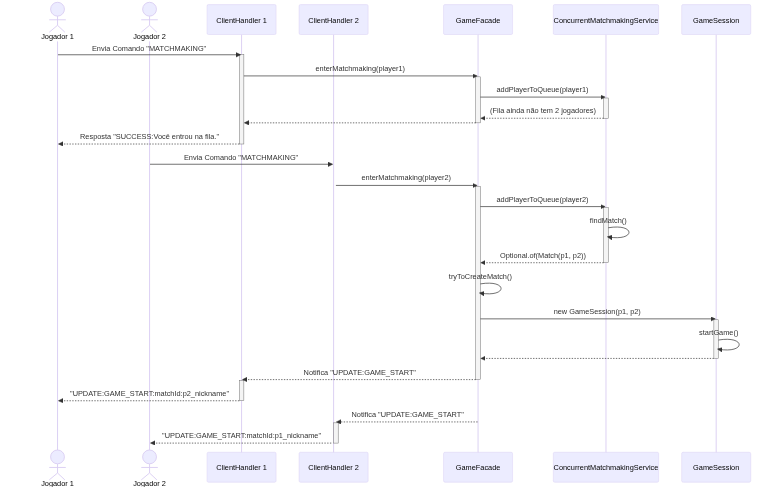
**Tabela 1. Principais comandos da API remota.**

## Comando Descric¸a˜o

CHARACTER SETUP Configura a rac¸a e classe do personagem. MATCHMAKING Adiciona o jogador a` fila de pareamento. STORE:BUY Compra um pacote de cartas.

GAME:PLAY CARD Realiza uma jogada de carta na partida.

UPGRADE Usa pontos para melhorar um atributo base.



**Figura 2. Diagrama de sequeˆncia para a criac¸ a˜o de uma partida.**

## Lateˆncia, Partidas e Pacotes

* + - **Lateˆncia**: Um PingServer UDP ecoa pacotes, permitindo que o cliente calcule o tempo de ida e volta (RTT).
    - **Partidas**: Uma condic¸a˜o de corrida no matchmaking foi corrigida: o GameFacade agora verifica se ambos os jogadores esta˜o conectados antes de criar a partida. Se um desconectou, o outro e´ devolvido a` fila.
    - **Pacotes**: A compra de pacotes e´ atoˆmica. O StoreServiceImpl usa um

ReentrantLock com pol´ıtica ”fair”para enfileirar as solicitac¸o˜es de compra, garantindo que o acesso ao estoque de cartas do CardRepository seja justo e sequencial.

## Testes e Emulac¸a˜o

Uma su´ıte de testes automatizada, orquestrada pelo script run all tests.sh, foi usada para validar a robustez do servidor. O Docker Compose cria um ambiente de emulac¸a˜o com um servidor e mu´ltiplos clientes. Os cena´rios testam desconexo˜es,

condic¸o˜es de corrida, comandos malformados e estresse com 10 clientes simultaˆneos por 30 segundos.

# Resultados

A execuc¸a˜o da su´ıte de testes validou a funcionalidade e a robustez do servidor. A Tabela [2](#_bookmark3) resume os resultados.

**Tabela 2. Resumo dos resultados dos testes automatizados.**

## Teste Objetivo Resultado

Desconexa˜o na Fila Lidar com desconexa˜o de cliente antes

da partida.

Jogada Simultaˆnea Testar o comportamento com ac¸o˜es concorrentes.

Malicious Bot Validar a robustez contra comandos malformados.

Teste de Estresse Avaliar desempenho com 10 clientes

simultaˆneos.

Sucesso Sucesso Sucesso Sucesso

A ana´lise dos logs foi fundamental. O teste de ”Desconexa˜o na Fila”foi crucial para identificar e corrigir a condic¸a˜o de corrida no matchmaking. Os logs confirmaram que, apo´s a correc¸a˜o, o servidor cancela a partida e devolve o jogador remanescente a` fila.

O teste de estresse validou a efica´cia do ReentrantLock no StoreServiceImpl, onde mu´ltiplos clientes compraram pacotes simultaneamente sem inconsisteˆncias.

# Conclusa˜o

Este trabalho alcanc¸ou o objetivo de implementar um servidor de jogo multiplayer robusto e concorrente. A arquitetura, baseada no padra˜o Facade e em estruturas de dados thread- safe, provou ser eficaz. A principal contribuic¸a˜o foi o desenvolvimento de uma su´ıte de testes automatizada com Docker, que foi indispensa´vel para identificar e corrigir bugs cr´ıticos de concorreˆncia.

Como trabalhos futuros, sugere-se a expansa˜o das mecaˆnicas de jogo, como um sistema de ”Energia”para ac¸o˜es em tempo real e a implementac¸a˜o de um sistema de mon- tagem de baralhos.

# Refereˆncias Refereˆncias

Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format.

Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., and Lea, D. (2006). *Java Concurrency in Practice*. Addison-Wesley Professional.

Kurose, J. F. and Ross, K. W. (2016). *Redes de computadores e a internet: uma aborda- gem top-down*. Pearson, 7 edition.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. In *Linux Journal*, volume 2014.

Tanenbaum, A. S. and Wetherall, D. J. (2011). *Redes de computadores*. Pearson Prentice Hall, 5 edition.