ARQUITETURA DE MICROSSERVIÇOS PARA RETAGUARDA DO PROJETO FURBOT

Jorge Guilherme Kohn, Mauro Mattos – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

jkohn@furb.br, mattos@furb.br

Resumo: *Arquiteturas de sistemas tradicionais sofrem degradação e apresentam dificuldades em escalar serviços quando expostos a picos de demanda ou experimentam crescimento exponencial de acessos. A arquitetura de microsserviços é uma das soluções que através de módulos separados autônomos entregam uma melhor performance e disponibilidade para o sistema como um todo. O projeto Furbot é uma iniciativa educacional que vem sendo preparada para ser disponibilizada nas lojas da Apple e Google e tem por objetivo auxiliar no desenvolvimento de habilidades de pensamento computacional em estudantes do ensino fundamental. Objetivando prover uma arquitetura para suportar muitos acessos provenientes a estes futuros usuários do jogo nas lojas de aplicativos, contudo trabalho aqui apresentado desenvolveu uma solução de infraestrutura de retaguarda para o projeto baseada na arquitetura de microsserviços com grande potencial de escalabilidade utilizando Java Spring Boot.*

Palavras-chave*:* Microsserviço. Performance. Disponibilidade. Escalabilidade. FURBOT. Java, Spring Boot.

1 INTRODUÇÃO

O mundo de jogos educacionais vem crescendo exponencialmente no Brasil, devido à forte inclusão digital e o rápido aprendizado passado aos alunos de forma interativa e dinâmica, e como afirma Silveira (1999, p. 15), "Os jogos educativos computadorizados são elaborados para divertir e aumentar a chance na aprendizagem de conceitos, conteúdos e habilidades embutidas no jogo.". Neste contexto, Vahldick e Mattos (2008) introduzem o projeto Furbot, criado com o intuito de apoiar atividades de ensino de habilidades em Pensamento Computacional (PC) em alunos de graduação.

O projeto evoluiu na forma de um projeto de extensão e, a partir de 2017, teve seu foco alterado para estudantes do ensino fundamental (entre 1º e 5º ano) (SCHLÖGL 2017), pois segundo Carvalho (1992, p. 14), "(...) desde muito cedo o jogo na vida da criança é de fundamental importância, pois quando ela brinca, explora e manuseia tudo aquilo que está a sua volta, através de esforços físicos e mentais …". Vem sendo mantidas duas novas versões do Furbot (além daquela da graduação): uma versão desplugada (KOHLER *et al*., 2019) e uma versão digital (MATTOS *et al*., 2019). A versão digital do Furbot sofreu um processo de refactoring e atualmente vem sendo desenvolvido em Unity, ferramenta esta que permite a disponibilização de duas releases: Android e WebGL (para execução em navegadores). Este processo demandou o desenvolvimento de novos leiautes, personagens, ambientes e desafios voltados para o novo público-alvo: crianças e adolescentes (MATTOS *et al*., 2019).

O Furbot vem sendo utilizado em oficinas, em escolas públicas estaduais do município de Blumenau desde 2017, tendo sido realizadas mais de 100 oficinas com crianças de 1º ao 5º ano do ensino fundamental. A partir dos resultados positivos do experimento descrito em Xavier *et al*. (2019), a equipe de projeto identificou a necessidade do desenvolvimento de uma infraestrutura de retaguarda que possibilite a aquisição e persistência de dados sobre a performance dos jogadores a medida em que eles avançam pelas fases do jogo com vistas a viabilizar o desenvolvimento de pesquisas em Learning Analytics (IFENTHALER;2015). Paralelamente a isto, esta demanda está alinhada com o *roadmap* do projeto que prevê o lançamento do Furbot nas plataformas Apple Store e Google Play disponibilizando o jogo em larga escala o que deve produzir um grande número de acessos e a produção de um volume de dados de utilização bastante expressivo.

Devido a estas mudanças de foco no FUBOT surgir a necessidade de aproveitar de diferentes formas os dados gerados através das jogatinas e o presente projeto foi concebido para disponibilizar uma solução de retaguarda na forma de uma arquitetura distribuída em microsserviços para receber e persistir o grande volume de dados produzidos pelo Furbot mobile a partir da sua publicação nas lojas Apple e Google. Objetivamente o projeto contempla:

a) a concepção, implementação e configuração de uma arquitetura microsserviços compatível com servidores de nuvem;
b) disponibilizar uma interface (API) de acoplamento entre o jogo e a infraestrutura;
c) disponibilizar uma forma de consulta para professores acompanharem o desenvolvimento de seus alunos.

O trabalho está organizado da seguinte forma: a seção 2 apresenta a fundamentação teórica dos temas envolvidos, a seção 3 descreve a arquitetura desenvolvida, a seção 4 apresenta os testes realizados e a seção 5 apresenta as conclusões do trabalho.

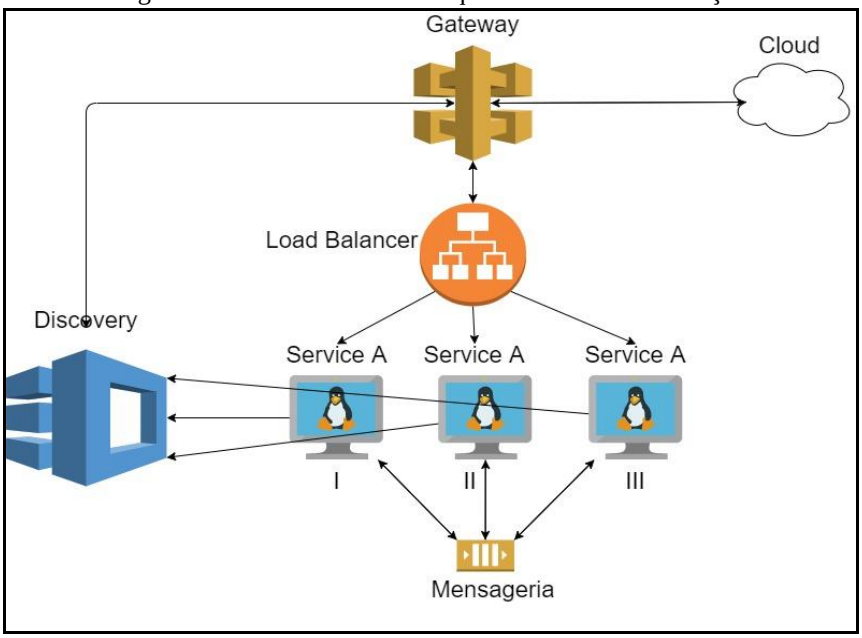
2 FUNDAMENTAÇÃO TEÓRICA

Serão apresentados os conceitos e como funcionam os microsserviços, Gateway, Discovery, Load Balancer e é apresentada uma visão geral do projeto Furbot.

2.1 CONCEITOS DE MICROSSERVIÇOS

Segundo Dragoni *et al*. (2017. p. 1), "Microsserviços é um estilo arquitetônico inspirado pela computação orientada a serviços" onde os chamados Serviços consistem em módulos específicos, resilientes, autônomos, escaláveis, integrados, além de um balanceador de carga, uma porta de entrada de requisições e um servidor de nomes. A Figura 1 apresenta como a arquitetura funciona, nela é possível identificar que as requisições da nuvem atingem o Gateway o qual por sua vez ativa o Discovery que localiza qual o serviço interno deve ser chamado. Após esta etapa a requisição é encaminhada ao Load Balancer que por sua vez escolhe qual instância do serviço será ativada. Em solicitações complexas outros serviços internos são ativados via sistema de mensagens.

Figura 1 - Funcionamento da arquitetura de microsserviços



Fonte: elaborado pelo autor.

A Figura 1 apresenta o componente de Discovery onde Pina (2018, p. 7) conclui que "Cada micro-serviço é construído consoante as necessidades da arquitetura.". Um ponto muito importante conforme Barrozo (2016, p. 13) é que é necessário que os demais módulos descubram onde esses serviços estão localizados, seja para monitorá-los ou mesmo para acessá-los, lembrando que é típico dessa arquitetura a volatilidade. O mecanismo responsável por este processo chama-se descoberta de serviço ou Discovery. No framework da Netflix ele é chamado de Eureka e todos os serviços são obrigados a registrarem-se nele de tal forma que o Discovery acaba por conhecer todas as instâncias de serviços on-line da arquitetura (PINA;2018, p. 18), além do conhecimento de todas as instancias o Discovery também é o responsável por fazer um broadcast para saber como está a vida útil dos microsserviços.

A partir do Discovery, o próximo componente chama-se Gateway (ou porta de entrada) representado na Figura 1 e tem por objetivo disponibilizar uma camada de abstração da Application Program Interface (API) que os serviços internos disponibilizam. Para Lima (2015, p. 155) "o gateway é responsável por receber as chamadas para os sistemas internos e será a única parte acessível externamente na forma de uma Application Program Interface (API)". A partir deste modelo, de um lado os clientes externos interagem com *endpoints* simples e o gateway trata internamente da complexidade de ligação com os vários *endpoints* internos desacoplando a estrutura de acesso da estrutura da implementação (LIMA;2015).

O terceiro aspecto da arquitetura é responsável pelo balanceamento de carga (Load Balancer) representado na Figura 1 o distribuindo a carga de dados entre várias instâncias de um mesmo serviço que pode estar rodando paralelamente em uma divisão lógica ou física da estrutura interna. Pina (2018, p. 14) afirma que "Um balanceador de carga é um componente que distribui tráfego de rede através de um aglomerado de servidores.", ou seja, para a arquitetura

ter escalabilidade e disponibilidade este módulo é indispensável, pois o mesmo irá gerenciar o tráfego de dados entre as instâncias de um mesmo microsserviço.

Normalmente a comunicação entre os serviços internos da arquitetura é realizada via o protoloco Hyper Text Transference Protocol (HTTP) ou via serviço de mensagens embora a segunda opção seja a mais adotada tendo em vista que ela provê recursos que facilitam o uso e a resiliência quando comparados com as chamadas HTTP. O RabbitMQ é uma das opções disponíveis e Weber (2016, p. 26) “RabbitMQ é um message broker e gerenciador de filas que permite o envio e recebimento de mensagens entre aplicações, sendo uma implementação do protocolo Advanced Message Queuing Protocol (AMQP)”.

2.2 PROJETO FURBOT

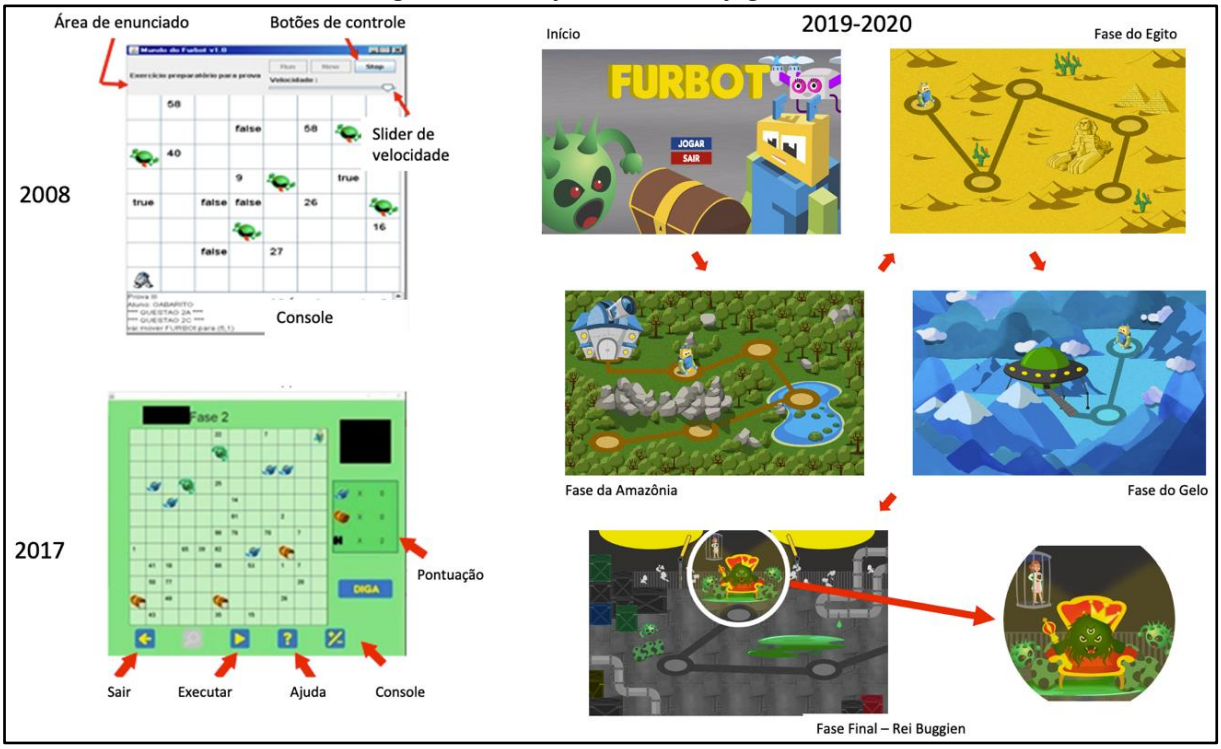
Conforme Tridapalli (2017), o projeto Furbot, criado em 2008, está inserido no contexto do ensino fundamental desde 2017 e busca promover a inclusão digital cidadã por meio de oficinas de programação que permitem o desenvolvimento de aptidões em pensamento computacional com uso de uma ferramenta lúdica. O Furbot é apresentado em duas versões: uma versão desplugada e uma versão digital. A versão desplugada compreende: um jogo de tabuleiro, onde o aluno é apresentado ao conjunto de comandos que serão utilizados na versão digital (Figura 2A) e, um jogo de tabuleiro “vivo” onde as crianças iniciam brincando com regras originais do Furbot, mas a medida em que a brincadeira fica interessante, lhes é permitido incrementar ou alterar as regras criando seus próprios jogos (Figura 2B).

Figura 2 – Componentes do Furbot desplugado.

Fonte: Mattos *et al.* (2018).

A versão digital iniciada em 2008 e desenvolvida em java, vem sendo utilizada desde então em disciplinas introdutórias de programação nos cursos de Ciência da Computação e Sistemas de Informação da FURB (Figura 3). Em 2017 houve uma operação de adequação da interface para utilização no ensino fundamental. A concepção da versão em Unity a partir de 2019 envolveu um completo redesenho da arquitetura do projeto introduzindo-se um apelo mais forte à gamificação (MATTOS; KOHLER, 2019).

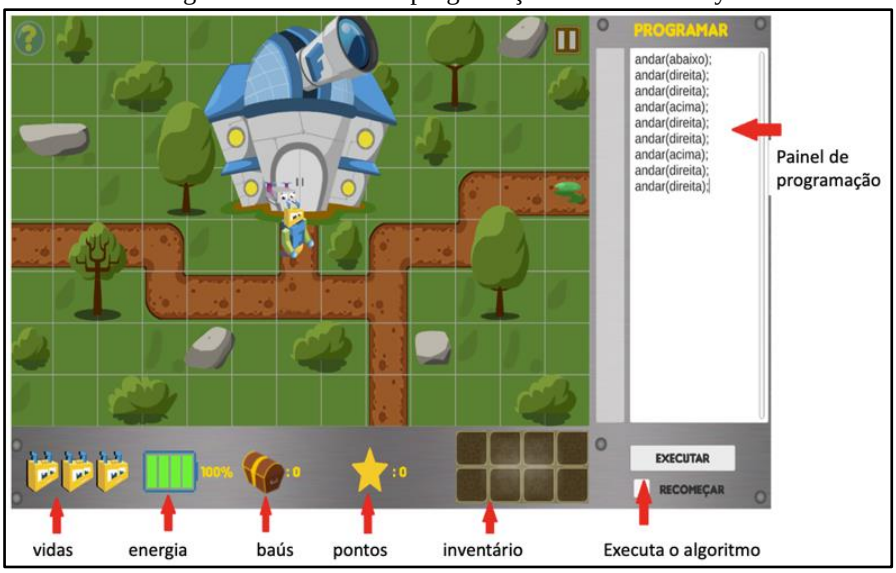
Figura 3 – Evolução histórica do jogo Furbot



Fonte: elaborado pelo autor.

A história tem como personagens a Professora Sam, o Furbot, o drone S223 e os malvados Buggiens. A Professora Sam cria o Furbot como um robô assistente. Contudo durante o desenvolvimento do robô, a terra é invadida pelos malvados Buggiens e o aluno é convocado a programar o Furbot ao longo de 5 fases e 20 etapas para resgatar a Professora Sam e salvar o mundo. Conforme Tridapalli (2017), na versão Unity, para viabilizar a movimentação do Furbot, o aluno precisa construir uma sequência lógica de passos. A Figura 4 apresenta a interface, onde o jogador pode introduzir (e editar) a sequência de comandos que serão executados pelo Furbot. Na base da tela, é possível controlar o número de vidas, a quantidade de energia (o Furbot perde energia quando bate em obstáculos e a programação precisa ser alterada para capturar-se energia suficiente para concluir uma fase), o número de tesouros obtidos, vidas, e outros elementos pertencentes à história do jogo.

Figura 4 – Interface de programação do Furbot Unity.



Fonte: digitalizado pelo autor.

A versão para dispositivos móveis, objeto da atenção deste projeto é apresentada na Figura 5 onde a interface é adaptada para as restrições de tela daquela plataforma. Nesta figura é possível identificar o painel de comandos (a) que permite que a criança construa os comandos necessários para movimentar o Furbot através do cenário da fase (b) e o

painel (c) onde o "código fonte" é apresentado e a partir do qual é possível acompanhar o efeito da execução de cada passo do programa construído (uma espécie de depurador).

Figura 5 - Furbot versão para dispositivos móveis

Fonte: digitalizado pelo autor.

Conforme afirmado anteriormente, o Furbot possui um roteiro estático composto de 20 etapas e a medida em que o jogador avança os desafios de programação são incrementados demandando mais atenção e habilidades. Entre 2017 e 2020 foram realizadas mais de 300 oficinas com aproximadamente 200 alunos do ensino fundamental (1º ao 5º ano) de escolas públicas em Blumenau. A partir desta experiência, em 2020 foi construído um protótipo para coleta de dados acerca dos movimentos que o jogador faz de forma a permitir o acompanhamento da sua evolução com vistas a construção de um módulo de learning analytics. A intenção é adequar o jogo para adaptar-se conforme as dificuldades que o jogador apresenta ao longo da estrutura do jogo. Conforme o *roadmap* do projeto, a equipe pretende publicar o jogo nas lojas da Apple e Google ampliando a disponibilidade do jogo para um universo muito mais amplo. Neste novo cenário, projeta-se que poderá haver um grande volume de jogadores realizando as atividades o que pode gerar picos de demanda e a adequação da estrutura de retaguarda para fazer frente a esta nova realidade. Neste contexto, a opção por uma infraestrutura baseada em microsserviços surge como uma opção a ser explorada e é o foco do presente projeto.

2.3 TRABALHOS CORRELATOS

São apresentados projeto correlatos, que possuem características semelhantes a proposta deste trabalho. O Quadro 1 apresenta o trabalho de Walpita (2017), o Quadro 2 apresenta o trabalho de Chiaradia (2018) e o Quadro 3 apresenta o trabalho de Lima (2015).

Quadro 1 – Trabalho correlato 1

Referência	Walpita (2017)
Título	Defense In-Depth Security Framework For Netflix Oss Microservices
Objetivos	Proteger os microsserviços internos de uma aplicação com o Framework da Netflix.
Principais funcionalidades	Revalidar o Token de autorização em cada microsserviço para garantir a integridade dos dados caso um invasor tenha acesso a arquitetura interna dos serviços.
Ferramentas de desenvolvimento	Java Spring Boot, Framework Netflix OSS, RabbitMQ.
Resultados e conclusões	O auto faz a seguinte citação: "A Netflix possui mais de 80 milhões de assinantes de streaming em todo o mundo e média de 20 milhões assinantes acessam o streaming simultaneamente". (WALPITA, 2017, p. 1)

Fonte: elaborado pelo autor.

Quadro 2 – Trabalho correlato 2

Referência	Chiaradia (2018)
Título	Uma Proposta de Arquitetura de Microsserviços Aplicada em um Sistema de CRM Social
Objetivos	Desenvolver um sistema de CRM em microsserviço.
Principais funcionalidades	A aplicação possui um gateway que repassa as requisições para os microsserviços de contatos, geolocalização, análise, relatório entre outros.
Ferramentas de desenvolvimento	Java e Ruby com o Framework Apache Spark.

Resultados e conclusões	O autor não traz dados quantitativos de performance, mas relata que "A utilização de microsserviços oferece uma série de benefícios relacionados à performance e disponibilidade do sistema, no entanto, os custos para manter uma estrutura desta forma podem ser relativamente altos" (CHIARADIA, 2018, p. 157).

Fonte: elaborado pelo autor.

Quadro 3 – Trabalho correlato 3

Referência	Lima (2015)
Título	Implementação de uma Arquitetura Baseada em Microserviços
Objetivos	Desenvolver um sistema baseado em microsserviços para pesquisa de preços em lojas físicas.
Principais funcionalidades	A aplicação possui um gateway que repassa as requisições para os microsserviços de: interesses, produtos, crawling e monitoramento.
Ferramentas de desenvolvimento	Python com o framework Flask e Play Framework.
Resultados e conclusões	O autor relata que a execução da arquitetura em uma máquina de 1,75 GB a mesma se manteve estável com até 3 mil usuários durante 5 minutos.

Fonte: elaborado pelo autor.

3 DESCRIÇÃO DA ARQUITETURA

A arquitetura do projeto atual é baseada no Framework Netflix OSS utilizando Java Spring Boot, que traz uma série de recursos já implementados e padronizados além do gerenciador de filas RabbitMQ para suporte à mensageria necessária para garantir a integridade dos dados trafegados nos serviços internos. Conforme apresentado anteriormente, a proposta pretende disponibilizar uma arquitetura com escalonamento horizontal e vertical voltada para o escopo do projeto Furbot.

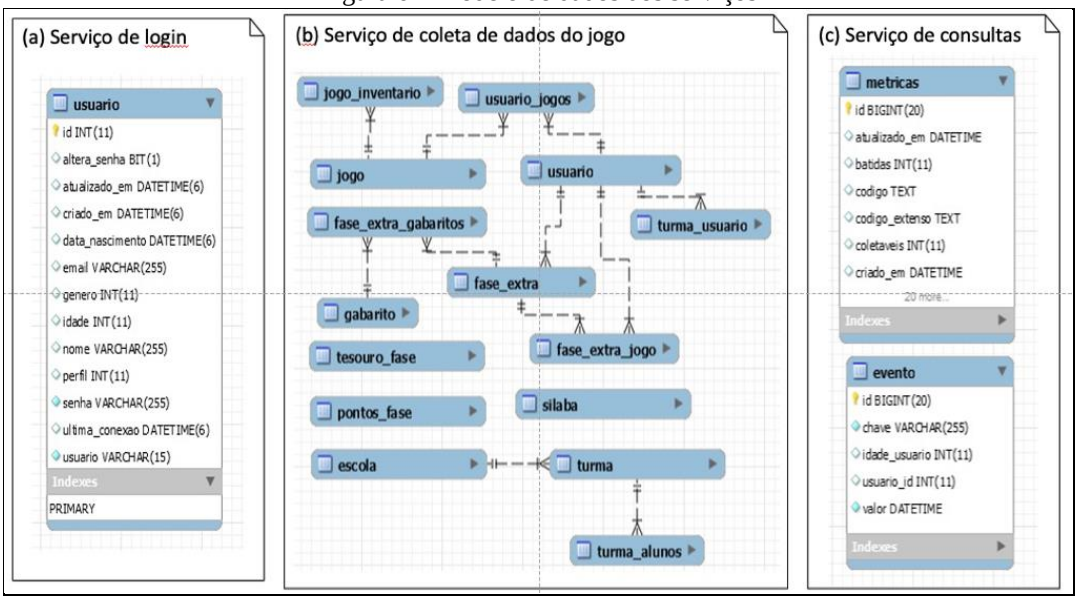
3.1 ESPECIFICAÇÃO

O projeto possui o seguinte conjunto de requisitos funcionais (RF) e não funcionais (RNF):

a) suportar CRUD de todas as entidades relacionadas ao jogo Furbot (RF);
b) auto descobrir novas instâncias da aplicação para desvio de fluxo de dados (RF);
c) prover uma API Gateway para abstrair a arquitetura interna dos microsserviços (RF);
d) prover uma dashboard básico de consulta a dados do jogo para o professor (RF);
e) permitir escalamento horizontal e vertical da arquitetura (RNF);
f) comportar o fluxo de dados de várias escolas ao mesmo tempo, persistindo e analisando os dados recebidos e consumidos pelo jogo (Requisito Não Funcional - RNF);
g) utilizar a linguagem Java com os Frameworks Spring Boot, Netflix OSS e RabbitMQ. (RNF);
h) utilizar containers Docker para implantação em servidores nuvem (RNF).

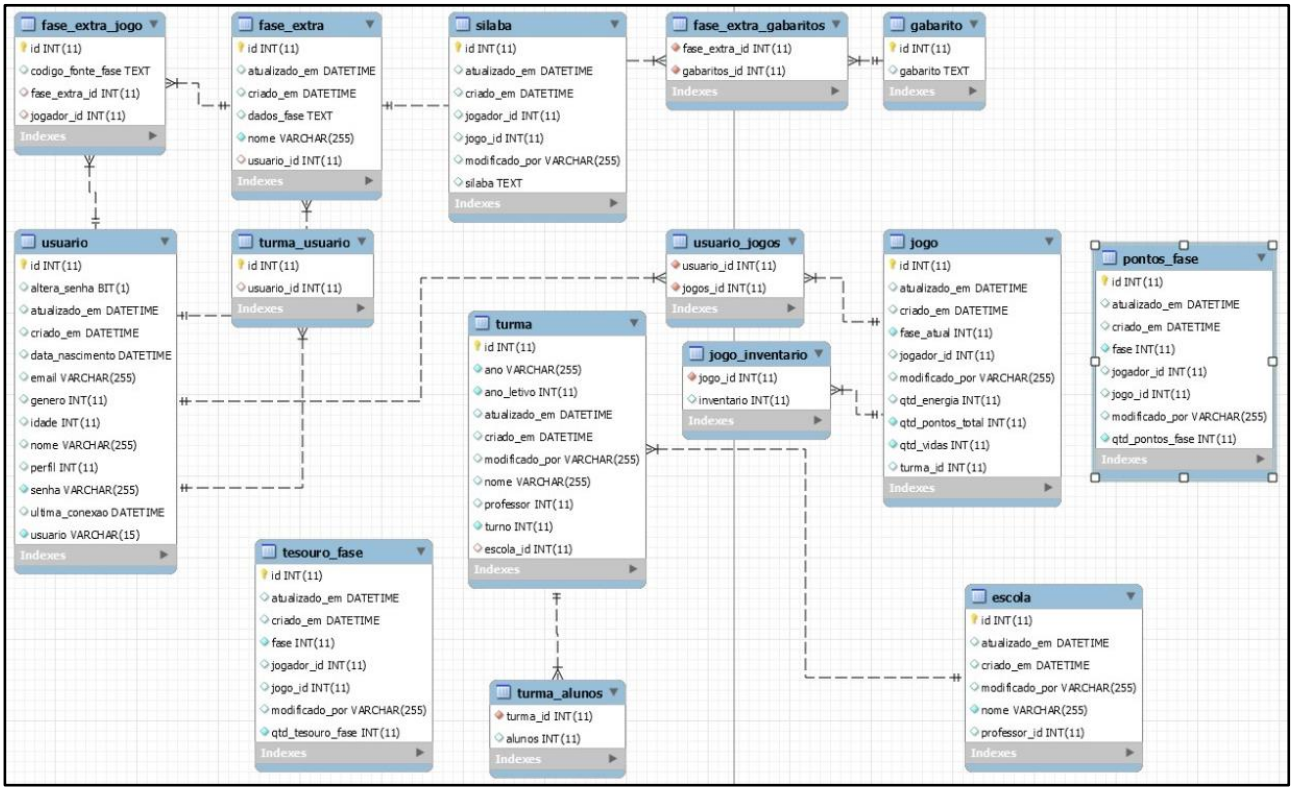
Seguindo o modelo citado anteriormente a arquitetura é composta pelos seguintes módulos: Gateway, Discovery, Load Balancer, microsserviço de usuário (para registro e autenticação) (Figura 6a), microsserviço de persistência de dados (para persistência dos dados do Furbot) (Figura 6b) e microsserviço de consultas (que armazena as métricas coletadas durante a execução dos jogos) (Figura 6c) . A Figura 6 apresenta o diagrama de modelo de dados das bases criadas para atender aos microsservicos correspondentes destacando que cada uma delas pode estar sendo executada em instâncias físicas separadas ou em instâncias lógicas do tipo container (Docker, Kubernetes etc.). A figura 7 apresenta em destaque o modelo de Entidades e Relacionamentos (MER) do serviço de coleta de dados do jogo.

Figura 6 – Modelo de dados dos serviços



Fonte: elaborado pelo autor.

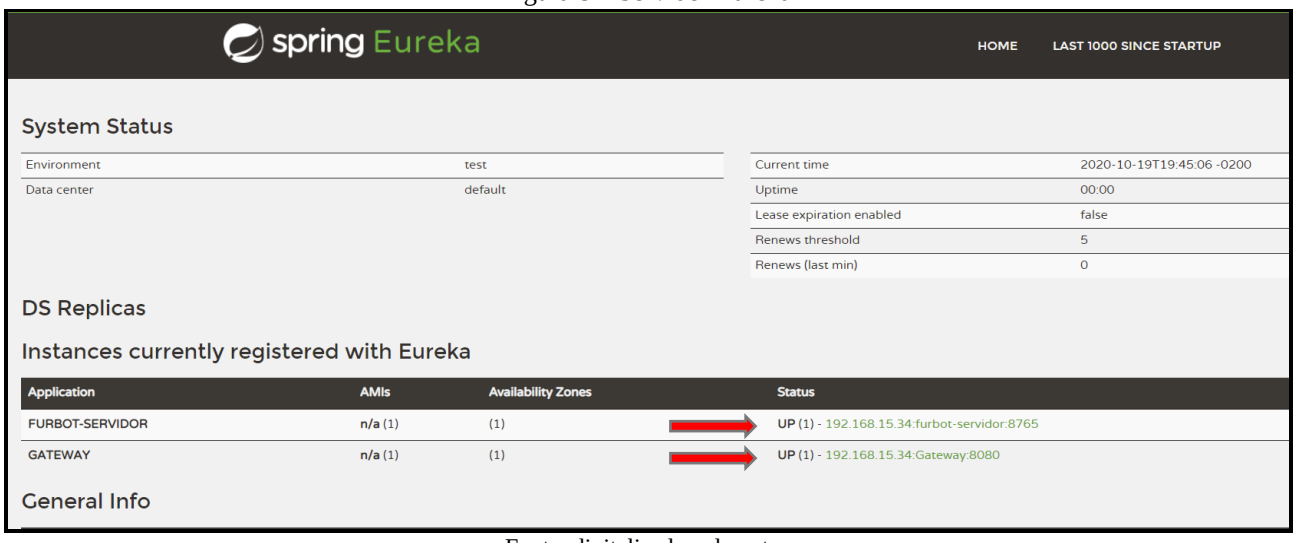
Figura 7 – Modelo de dados do serviço de coleta de dados do jogo



Fonte: elaborado pelo autor.

O módulo de Discovery (utiliza o Netflix Eureka) é responsável pela descoberta das instâncias dos serviços internos, ou seja, é o servidor de nomes, onde se encontra a localização física por meio do protocolo Protocolo de Internet versão 4 IPV4 de cada microsserviço. O Discovery também foi configurado para gerar um sinal de broadcast a cada 30 segundos para consultar se os serviços estão disponíveis. A Figura 8 representa o Eureka exibindo todas as instâncias e serviços ativos, neste exemplo o serviço de Furbot-Servidor (ativo na porta 8765) e o gateway (ativo na porta 8080). Vale lembrar que o Discovery deve ser alcançado por todos os microsserviços para que o mesmo seja eleito a processar requisições.

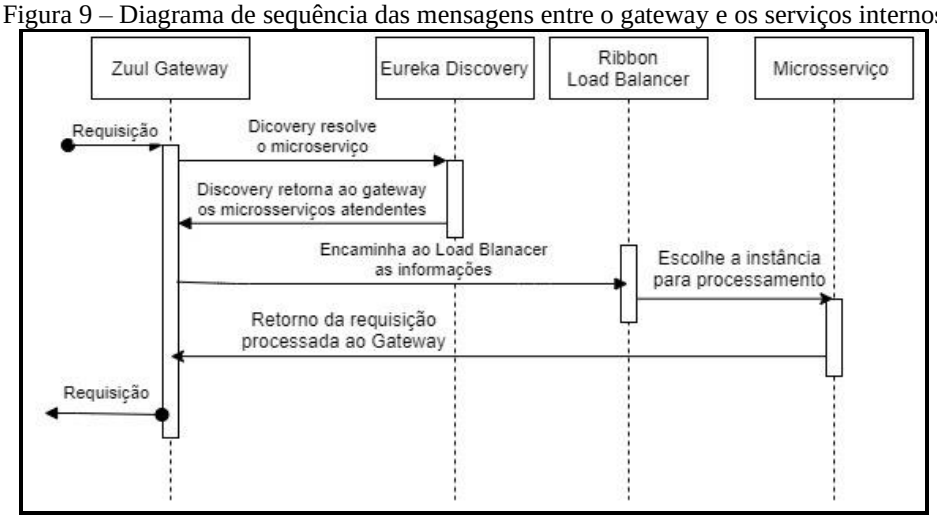
Figura 8 – Servidor Eureka



Fonte: digitalizado pelo autor

O módulo de Gateway (utiliza o Netflix Zuul) corresponde a um servidor de proxy que conversa diretamente com o Eureka e o Discovery da aplicação, para localizar os serviços disponíveis para cada chamada à API. A dinâmica de funcionamento deste processo é apresentada na Figura 9, que inicia com uma requisição chegando ao Gateway e todo fluxo de mensagens até a liberação do retorno da mesma.

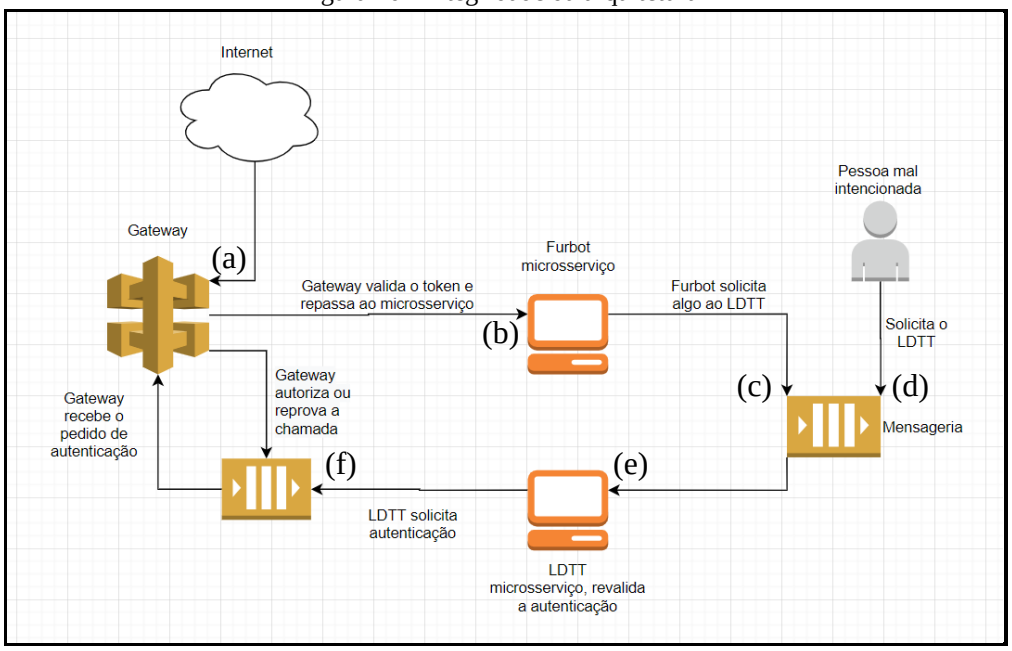
Figura 9 – Diagrama de sequência das mensagens entre o gateway e os serviços internos



Fonte: elaborador pelo autor.

A consequência natural da distribuição de responsabilidades é o caráter assíncrono desta arquitetura. Neste contexto para evitar acesso mal intencionado aos serviços internos foi desenvolvido um modelo de revalidação de token de autorização em todas as chamadas internas. Esta solução introduz um pequeno atraso nas requisições, mas garante que os dados que estão sendo acessados estão devidamente autorizados. A Figura 10 detalha o fluxo de mensagens que chegam ao Gateway (Figura 10a) e são repassadas ao primeiro microsserviço (Figura 10b), caso o mesmo solicite dados de outro microsserviço é feito pela mensageria (Figura 10c). Neste ponto é possível que alguém mal intencionado adicione chamadas não autenticadas para processamento (Figura 10d), porem assim que o microsserviço recebe uma mensagem (Figura 10e), encaminha a mesma para o Gateway novamente para confirmar a autenticidade (Figura 10f).

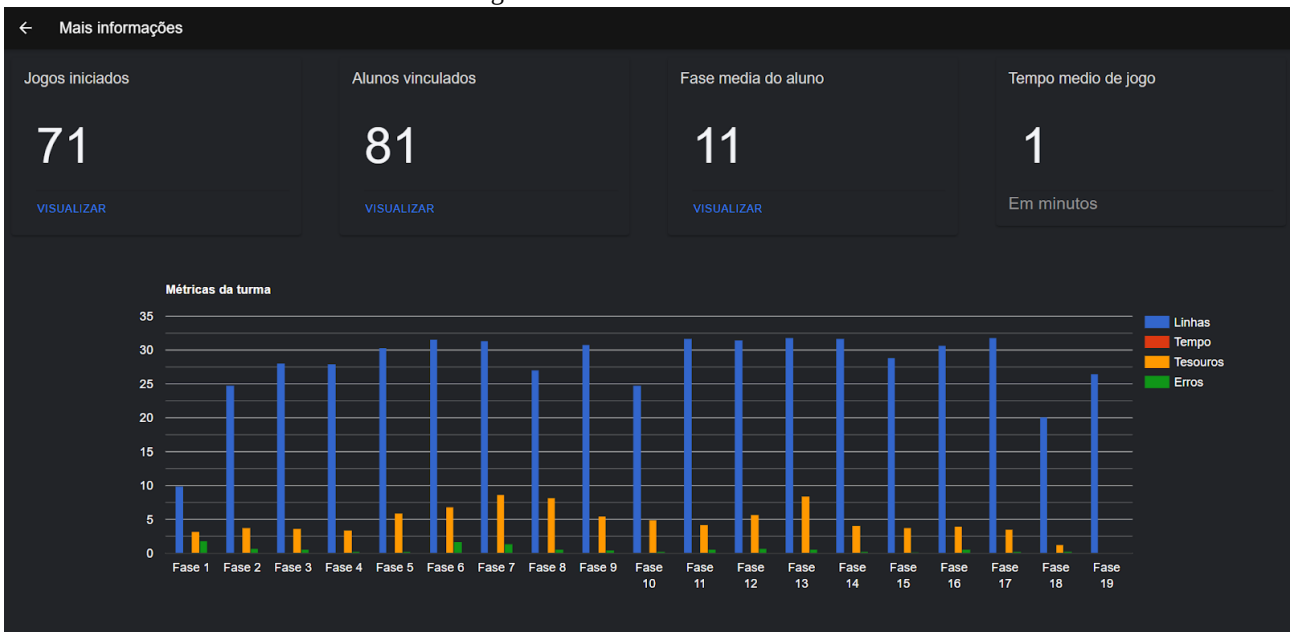
Figura 10 - Integridade da arquitetura



Fonte: elaborado pelo autor.

O dashboard da aplicação em que o professor utiliza para consulta de dados em tempo real dos seus alunos, utiliza a mesma API disponibilizada para o jogo do FURBOT para consulta de dados. É possível obter informações sobre as escolas, turmas, jogos e métricas básicas de jogos dos alunos vinculadas as turmas que o professor gere, que podemos verificar na Figura 11 o relatório sumarizado de fazes, tempo, tesouros e erros médios dos alunos da turma.

Figura 11 – Dashboard do Furbot

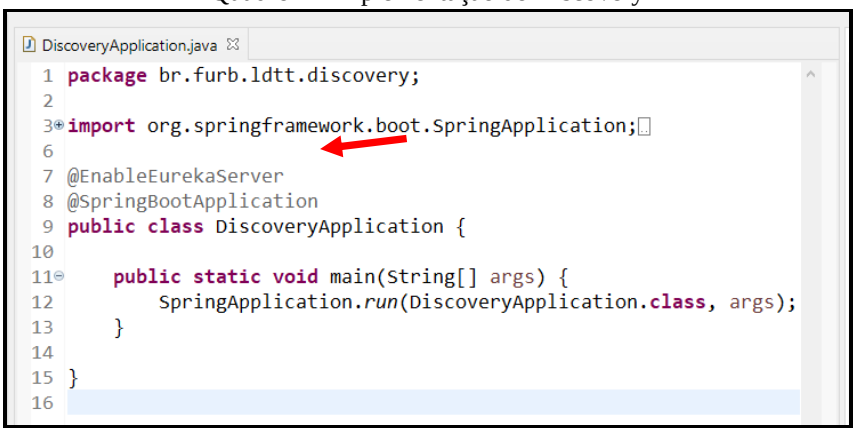


Fonte: elaborado pelo autor.

3.2 IMPLEMENTAÇÃO DA INFRAESTRUTURA

O projeto foi desenvolvido utilizando-se o Framework Java Spring Boot com as bibliotecas da Netflix conforme citado anteriormente. O quadro 4 destaca a implementação do módulo Discovery, onde basta adicionar a anotação `@EnableEurekaServer` acima do método Main do projeto que o Framework se encarrega de implementar todas as funções citadas na especificação.

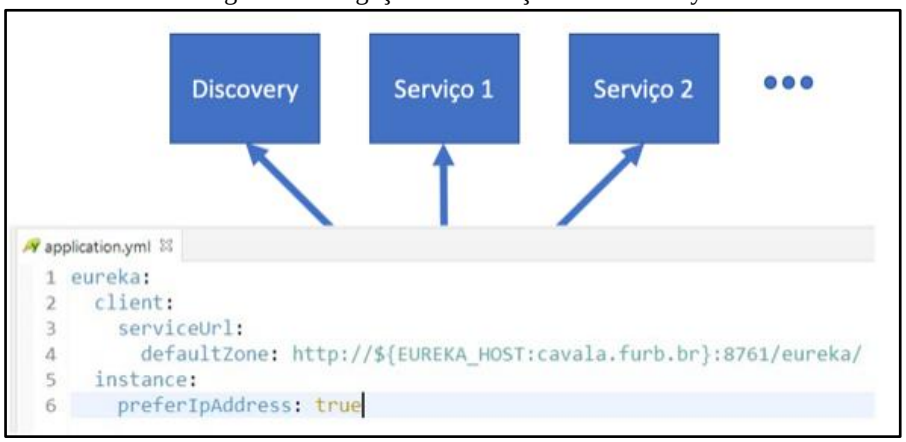
Quadro 4 - implementação do Discovery



Fonte: elaborado pelo autor.

A Figura 1212 destaca que a ligação entre o módulo Discovery e cada microsserviço ocorre através de um arquivo de configuração que possibilita que cada microsserviço se registre ao Discovery para que o mesmo possa apontar qual servidor atende qual serviço. Este é um aspecto importante porque se o arquivo de configuração de um microsserviço não estiver apontando para o local correto do Discovery esta instância de processamento não será ativada pela arquitetura.

Figura 12 – Ligação dos serviços ao Discovery



Fonte: elaborado pelo autor.

O proxy necessário para abstração da arquitetura interna possui uma configuração simplificada como aquela utilizada pelo o módulo Discovery, ou seja, basta adicionar as anotações `@EnableEurekaClient` e `@EnableZuulProxy` acima do método Main do projeto (Quadro 5).

Quadro 5 Configuração do serviço de proxy

Fonte: elaborado pelo autor.

Diferentemente do Discovery, o Gateway requer um nível maior de atenção na questão de autenticação com o módulo JWT Security (Quadro 6) para que sejam disponibilizadas uma série de garantias de segurança como: um token para cada sessão do usuário (Quadro 6B), configurações de rotas públicas (Quadro 6A), controle de origem (Quadro 6C), criptografia entre outros recursos.

Quadro 6 – Configuração do módulo JWT.

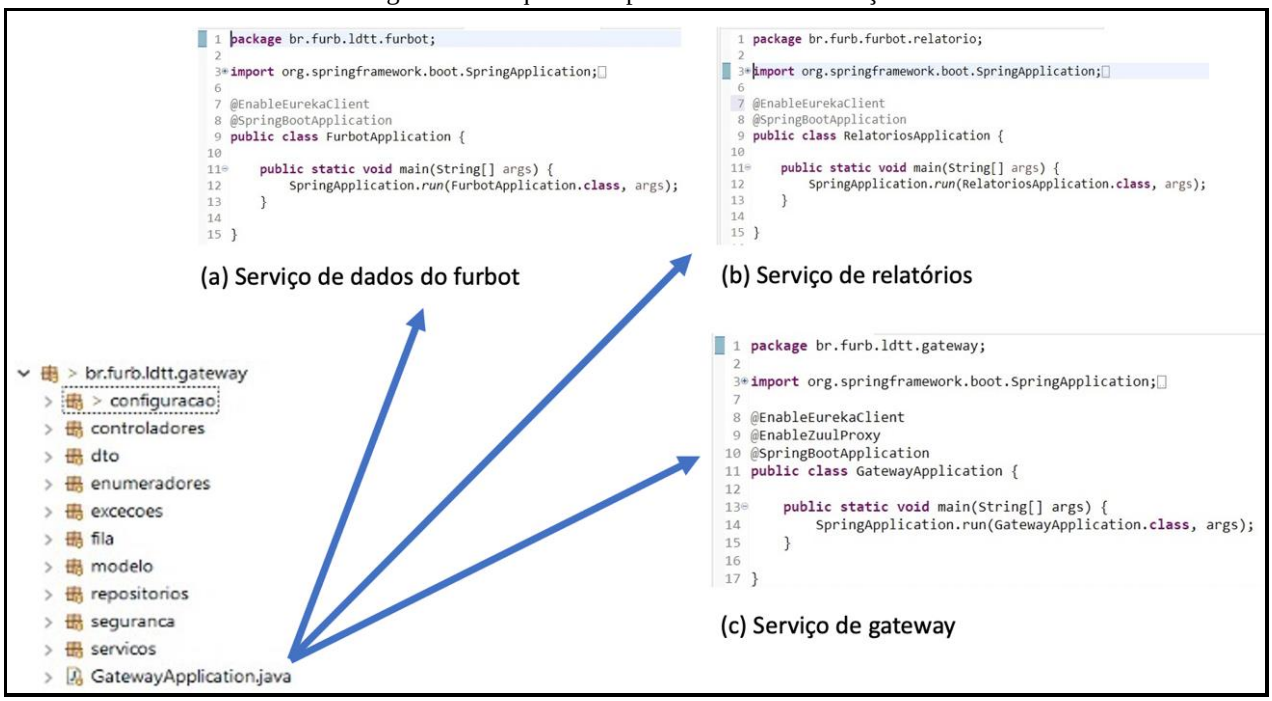
```
55    @Override
56    protected void configure(HttpSecurity http) throws Exception {
57
58        http.cors().and().csrf().disable();                                    (a)
59        http
60        .cors().and()
61        .authorizeRequests()
62            .antMatchers(HttpMethod.POST, PUBLIC_MATCHERS_POST).permitAll()
63            .antMatchers(HttpMethod.GET, PUBLIC_MATCHER_GET).permitAll()
64            .antMatchers(PUBLIC_MATCHERS).permitAll()
65            .anyRequest().authenticated();
66        http.addFilter(new AutenticacaoFiltroJWT(authenticationManager(), jwtUtil));
67        http.addFilter(new AutorizacaoFiltroJWT(authenticationManager(), jwtUtil, this.userDetailsService));
68        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
69        http.headers().frameOptions().disable();
70    }
71                                                                               (b)
72    @Override
73    public void configure(AuthenticationManagerBuilder auth) throws Exception {
74        auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder());
75    }
76
77    @Bean                                                                      (c)
78    CorsConfigurationSource corsConfigurationSource() {
79        CorsConfiguration configuration = new CorsConfiguration();
80        configuration.setAllowedOrigins(Collections.singletonList("*"));
81        configuration.setAllowedMethods(Collections.singletonList("*"));
82        configuration.setAllowedHeaders(Collections.singletonList("*"));
83        configuration.setExposedHeaders(Collections.singletonList(HttpHeaders.ACCEPT));
84        configuration.setAllowCredentials(true);
85        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
86        source.registerCorsConfiguration("/**", configuration);
87        return source;
88    }
89
90    @Bean
91    public BCryptPasswordEncoder bCryptPasswordEncoder() {
92        return new BCryptPasswordEncoder();
93    }
```

Fonte: elaborado pelo autor.

3.3 IMPLEMENTAÇÃO DOS MICROSSERVIÇOS

Após da implementação do núcleo da arquitetura, foram desenvolvidos os microsserviços responsáveis por autenticação (Figura 13c), coleta de dados do jogo (Figura 13a) e relatórios (Figura 13b) sendo que ambos utilizam Java Spring Boot e Hibernate.

Figura 13 - Esquema de pastas dos microsserviços



Fonte: elaborado pelo autor.

Vale destacar que os 3 microsserviços possuem a mesma estrutura de pastas contendo os códigos de configurações, controladores, enumeradores, exceções, modelos, repositórios, segurança e serviços, quais sejam:

a) Configurações: nesta pasta são encontrados os arquivos responsáveis por definir o RabbitMQ, o token JWT para segurança e definições de paginação nas chamadas (para redução de tráfego de dados e evitar mensagens muito grandes nas filas da mensageria);
b) Controladores: Aqui se encontram todas as rotas aceitas pela API, que serão repassadas ao Gateway para acesso público ou autenticado. A sua implementação seguiu os seguintes padrões com base nos verbos HTTP GET para busca ou listagem de uma entidade, POST para cadastrar um novo registro na entidade, PUT para atualizar um registro na entidade, DELETE para remover um registro da entidade (Quadro 7);

Quadro 7 – Configuração das rotas

```java
@RestController
@RequestMapping(value = "/api/jogos")
public class JogoControlador {

    @Autowired
    private JogoServico jogoServico;

    @GetMapping
    public ResponseEntity<Pagina<Jogo>> buscar(@RequestParam(defaultValue = "0") Integer pagina,
            @RequestParam(defaultValue = "10") Integer tamanho) {
        return ResponseEntity.ok().body(new Pagina<Jogo>(this.jogoServico.listar(pagina, tamanho)));
    }

    @GetMapping(value = "/{id}")
    public ResponseEntity<JogoDto> buscar(@PathVariable Integer id) {
        return ResponseEntity.ok().body(this.jogoServico.buscarTudo(id));
    }

    @PostMapping
    public ResponseEntity<Jogo> criar(@Valid @RequestBody Jogo jogo) {
        return ResponseEntity.ok().body(this.jogoServico.criar(jogo));
    }

    @PutMapping(value = "/{id}")
    public ResponseEntity<JogoDto> atualizar(@PathVariable Integer id, @Valid @RequestBody JogoDto jogo) {
        return ResponseEntity.ok().body(this.jogoServico.atualizar(id, jogo));
    }
```

Fonte: elaborado pelo autor.

c) Exceções: aqui estão todas as exceções mapeadas as quais podem ser lançadas a qualquer momento e que serão automaticamente convertidas para o código HTTP através de um interceptador de exceções. Esta solução evita o uso das tradicionais estruturas de TRY CATCH no meio da implementação. Deve-se observar que, caso alguma exceções não esteja devidamente mapeada aqui, será retornando o erro HTTP 500, Internal Server Error;
d) Modelos: aqui estão todas as classes que definem exatamente os modelos do banco de dados e que será utilizado pelo Hibernate para tratar com estes dados. Também nestes modelos estão as definições do banco e validações que devem ser aplicadas ao tentar persistir estes dados, pois com base nelas são realizadas as migrações na base de dados em caso de nova entidade ou atualizações no modelo;
e) Repositórios: são as interfaces da arquitetura com o banco de dados, onde já existe uma pré implementação do CRUD básico que pode ser utilizado ou adicionadas novas consultas sob demanda ou especificas para cada caso;
f) Segurança: para garantir a integridade da aplicação, todas as chamadas passam pela camada de segurança, que decriptografa o token JWT para identificar o tempo de expiração da mesma, o e-mail do usuário e perfil de acesso. Além de conter estas informações criptografadas no token, também são realizadas consultas na base de dados para validar se os dados ainda são condizentes com os mesmos utilizados na geração do token. Em caso positivo é criada uma sessão para o usuário válida durante a requisição (ela expira após o retorno da chama ao Gateway);
g) Serviços: aqui encontra-se, entre outros, a lógica da aplicação, com códigos Java, injeções de dependências (Quadro 8a), validações (Quadro 8b), controle de transação (Quadro 8c) e mapeamentos realizados. O Quadro 8 apresenta um exemplo um exemplo de serviço implementado no Furbot responsável pelo CRUD da entidade pontos por fase;

Quadro 8 – Exemplo de serviço

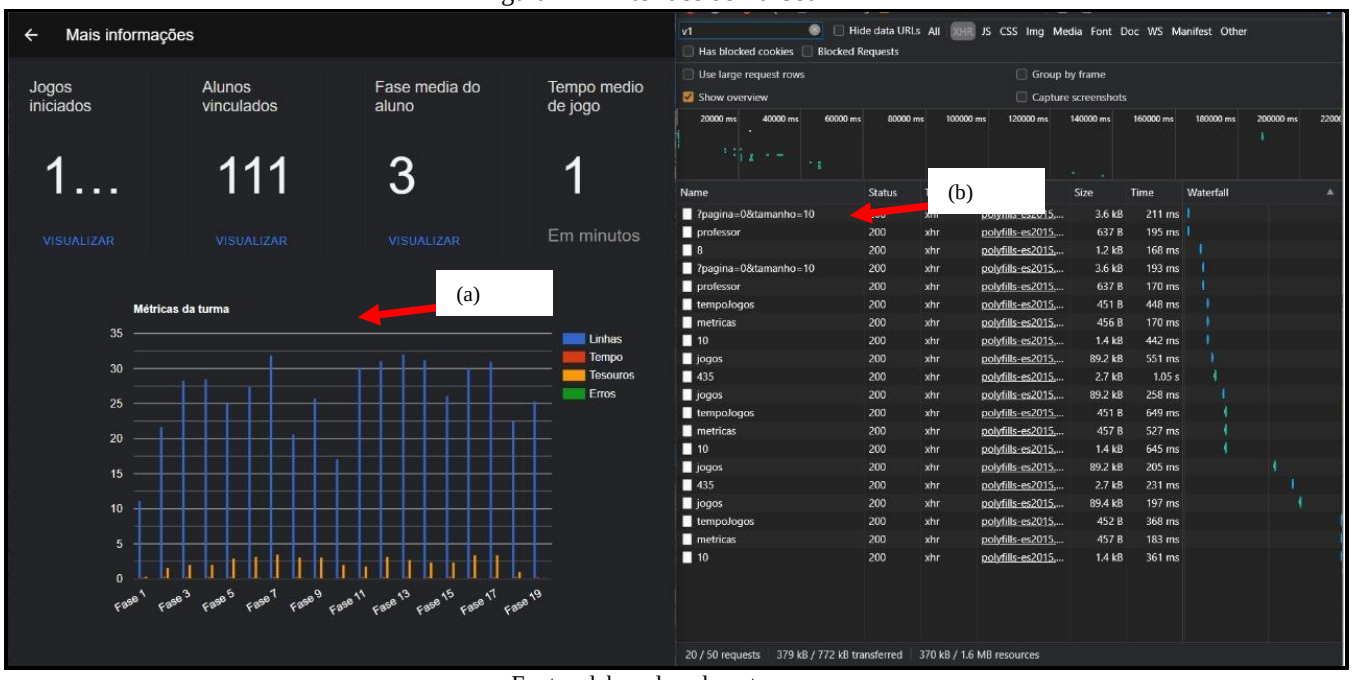
```java
package br.furb.ldtt.furbot.servico.jogo;

import java.util.List;

@Service
public class PontosFaseServico {

    @Autowired
    private PontosFaseRepositorio pontosFaseRepositorio;

    @Autowired
    private UsuarioServico usuarioServico;

    public PontosFase buscar(Integer id) {
        Optional<PontosFase> pontosFase = this.pontosFaseRepositorio.findById(id);
        if (!pontosFase.isPresent()) {
            throw new ObjetoNaoEncontradoExcecao("Ponto da fase não encontrado!");
        }
        return pontosFase.get();
    }

    public PontosFase buscarPorUsuarioLogado(Integer id) {
        return this.buscar(id);
    }

    public List<PontosFase> buscarTodas(Jogo jogo) {
        return this.pontosFaseRepositorio.findByJogoId(jogo.getId());
    }

    @Transactional
    public PontosFase criar(PontosFase pontosFase) {
        Usuario usuario = this.usuarioServico.usuarioLogado();
        pontosFase.setJogadorId(usuario.getId());
        pontosFase.setModificadoPor(usuario.getUsuario());
        return this.pontosFaseRepositorio.save(pontosFase);
    }

    public PontosFase atualizar(Integer id, PontosFase pontosFase) {
        PontosFase pontosFaseBuscado = this.buscar(id);
        if (pontosFase.getFase() != null) {
```

(a) (b) (c)

Fonte: elaborado pelo autor.

Hoje em dia implementar uma arquitetura de microsserviços se torna algo fácil e intuitivo devido a todos os recursos e benefícios que os Frameworks nos provêm. Na Figura 14 é a tela de relatório do Furbot (Figura 14a) e ao lado uma lista de endpoints com chamadas para o Gateway de diferentes microsserviços (Figura 14b). O Frontend foi criado com Ionic 5 utilizando a API disponibilizada pela arquitetura.

Figura 14 – Interface do Furbot



Fonte: elaborado pelo autor.

4 RESULTADOS

Ao final do trabalho, considera-se que os requisitos foram atendidos. O Quadro 9 apresenta um comparativo da ferramenta com os trabalhos correlatos.

Quadro 9 – Comparação dos correlatos

	Trabalhos correlatos			
Características	Walpita (2017)	Chiaradia (2019)	Lima (2015)	Jorge (2020)
Microsserviços	X	X	X	X
Estrutura em nuvem	X	X	X	X
Escalabilidade	X	X	X	X
Linguagem	Java	Java e Ruby	Python	Java
Framework	Netflix OSS e Java Spring Boot	Apache Spark	Flask e Play Framework	Netflix OSS e Java Spring Boot
Segurança	JWT	Não informada	Não informada	JWT
Mensageria	RabbitMQ	HTTP	RabbitMQ	RabbitMQ
Performance	Empresas que usam esta tecnologia trabalham com até 20 milhões de usuários simultâneos.	Sem dados quantitativos de performance, porem informa que a arquitetura tem uma série de benefícios relacionados à performance e disponibilidade.	Com 1.75 GB de memória RAM, permaneceu estável com até 3 mil usuários simultâneos.	Se manteve estável com até mil usuários simultâneos com hardware de 1,7 GB de memória RAM.

Fonte: elaborado pelo autor.

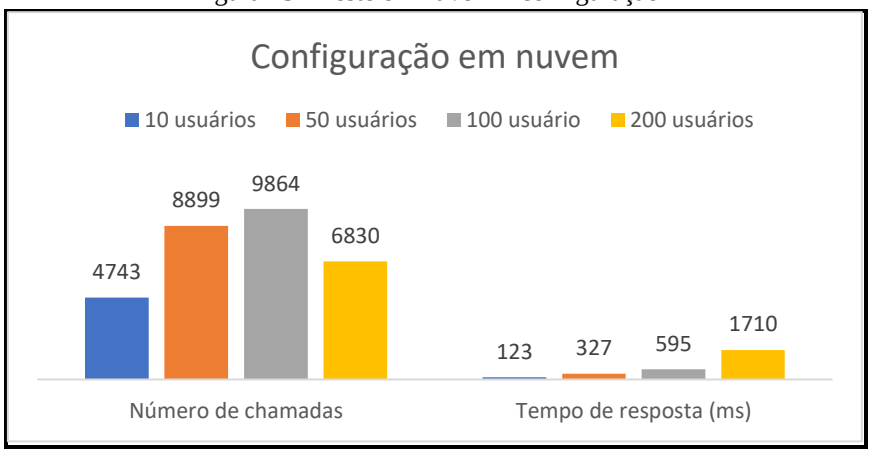
Com os resultados do quadro acima, podemos ver que existem várias formas de desenvolver uma arquitetura em microsserviços, com diferentes frameworks, diferentes linguagens e ótimos com dados de performance. A escalabilidade é um quesito tratado em todos os correlatos, desde a parte de distribuição de carga até o devido cluster em um cloud, pois as arquiteturas em microsserviços devem permitir o escalonamento lateral quanto horizontal.

Contudo o trabalho aqui apresentado se mostrou equivalente aos correlatos mesmo possuindo mecanismos de segurança não citados nos demais trabalho o que impacta diretamente na performance. De modo a validar o modelo construído e a sua equidade aos correlatos foram realizados três testes de performance, dois em laboratório simulando uma alta carga de usuários simultâneos e um teste em ambiente real de utilização com usuários reais cujos resultados são apresentados a seguir.

4.1 TESTE EM NUVEM - CONFIGURAÇÃO 1

Neste experimento foi utilizado o Google Cloud Platform (GCP) juntamente com o jMeter da Apache disparando várias requisições durante um minuto consecutivos. Neste caso a infraestrutura foi montada com três máquinas localizadas na Califórnia (EUA) com 600 Megabytes de memória RAM e uma vCpu compartilhada. Na primeira máquina foi instalado o Discovery, na segunda o Gateway e na terceira os serviços do Furbot. Os resultados são apresentados na Figura 15. Neste caso a arquitetura se manteve estável com até 100 usuários simultâneos, porém com até 200 usuários embora tenha conseguido responder sem a necessidade de subir uma nova instancia, percebeu-se uma latência maior referente ao número de chamadas e tempo de resposta. Para fins de pesquisa na data do teste o Google calculou o custo mensal das máquinas pelo valor de R$ 35,16 (valor já convertido considerando o dólar a R$ 5,74).

Figura 15 - Teste em nuvem – configuração 1

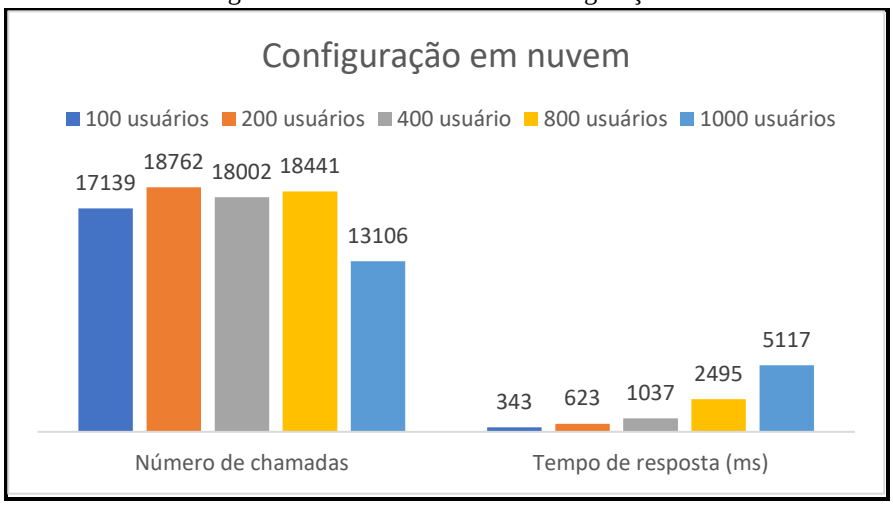


Fonte: elaborado pelo autor.

4.2 TESTE EM NUVEM - CONFIGURAÇÃO 2

Neste experimento foi utilizado o Google Cloud Platform (GCP) juntamente com o jMeter da Apache disparando várias requisições durante um minuto consecutivos, agora com três máquinas (também localizadas na Califórnia) com 1,7 Gigabytes de memória RAM e uma vCpu compartilhada. Nesta configuração a primeira máquina hospedou o Discovery, a segunda o Gateway e a terceira o Furbot. A Figura 6 apresenta os resultados obtidos. Neste cenário a arquitetura se manteve estável com até 400 usuários simultâneos, porém com até 1000 usuários, embora tenha conseguido responder sem a necessidade de subir uma nova instancia, percebeu-se uma latência em 5 segundos, o que já impactaria no tempo de resposta aos usuários do jogo. Neste cenário fica evidenciada a necessidade de subir uma nova instancia para suportar um fluxo projetado de mais de 400 usuário ativos simultaneamente. Neste cenário, para fins de pesquisa, na data do teste o Google calculou o custo mensal das máquinas pelo valor de R$ 80,67 (convertidos considerando o dólar a R$ 5,74) o que representa valores aceitáveis para o escopo do projeto Furbot.

Figura 16 - Teste em nuvem - configuração 2

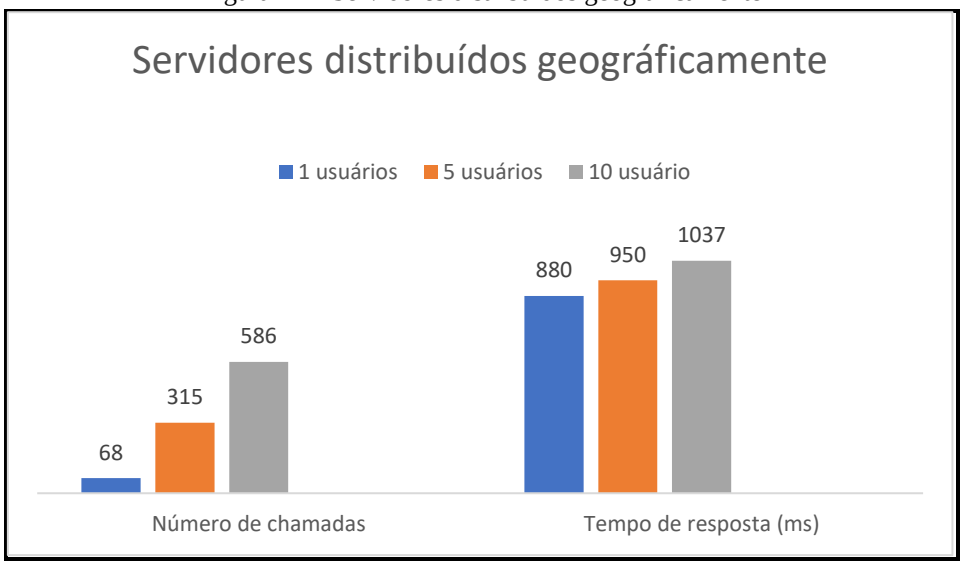


Fonte: elaborado pelo autor.

4.3 TESTE EM SERVIDORES GEOGRAFICAMENTE DISTANTES

Neste experimento foi utilizando os recursos do Experimento 4.1, foi realizado um novo teste onde os microsserviços foram configurados em diferentes locais: o Gateway foi configurado no Brasil, o Discovery no Estados unidos e os serviços na Europa. Neste caso os resultados apontaram uma grande degradação no tempo de resposta em função da necessidade de intensa comunicação interna entre os servidores. A figura 17 demonstra os resultados onde no melhor dos casos, com 1 usuário, o tempo de resposta médio ficou acima de 800 milissegundos.

Figura 17 – Servidores distribuídos geograficamente

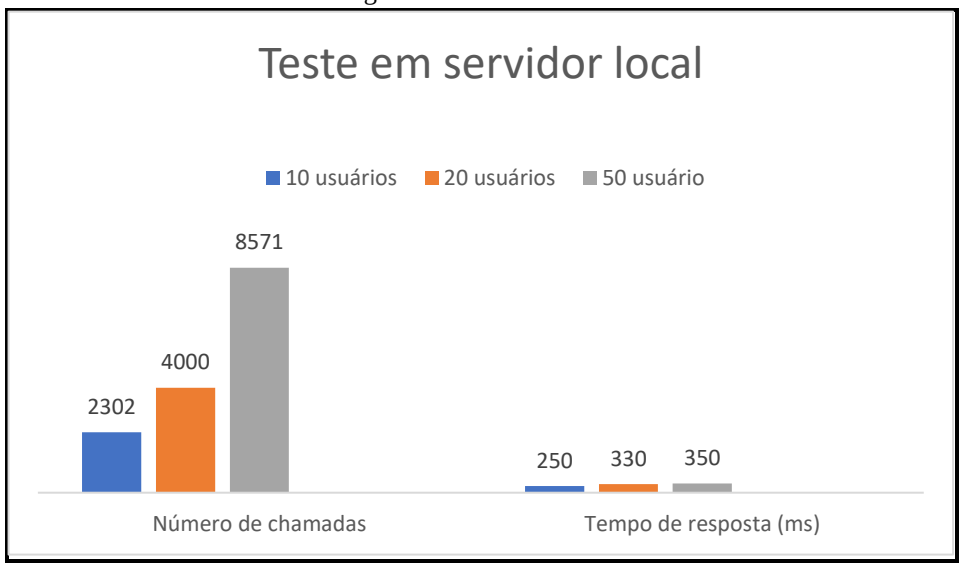


Fonte: elaborado pelo autor.

4.4 TESTE EM AMBIENTE REAL

Neste teste foi utilizada a arquitetura considerada “de produção” a qual nos servidores da Universidade Regional de Blumenau (FURB). Todos os microsserviços, Gateway, Discovery e RabbitMQ foram hospedados em uma única máquina virtual com 4 GB de memória RAM e um processador Xeon, vale lembrar que estas instancias estão com a proteção de DDOS ativa, ou seja, não é possível enviar muitas requisições por segundo em uma mesma sessão o que impacta na capacidade de teste de performance. Neste caso, o número de jogadores simultâneos atingiu um pico de 50, durante um minuto, sendo os resultados apresentados na Figura 18.

Figura 18 – Servidor local



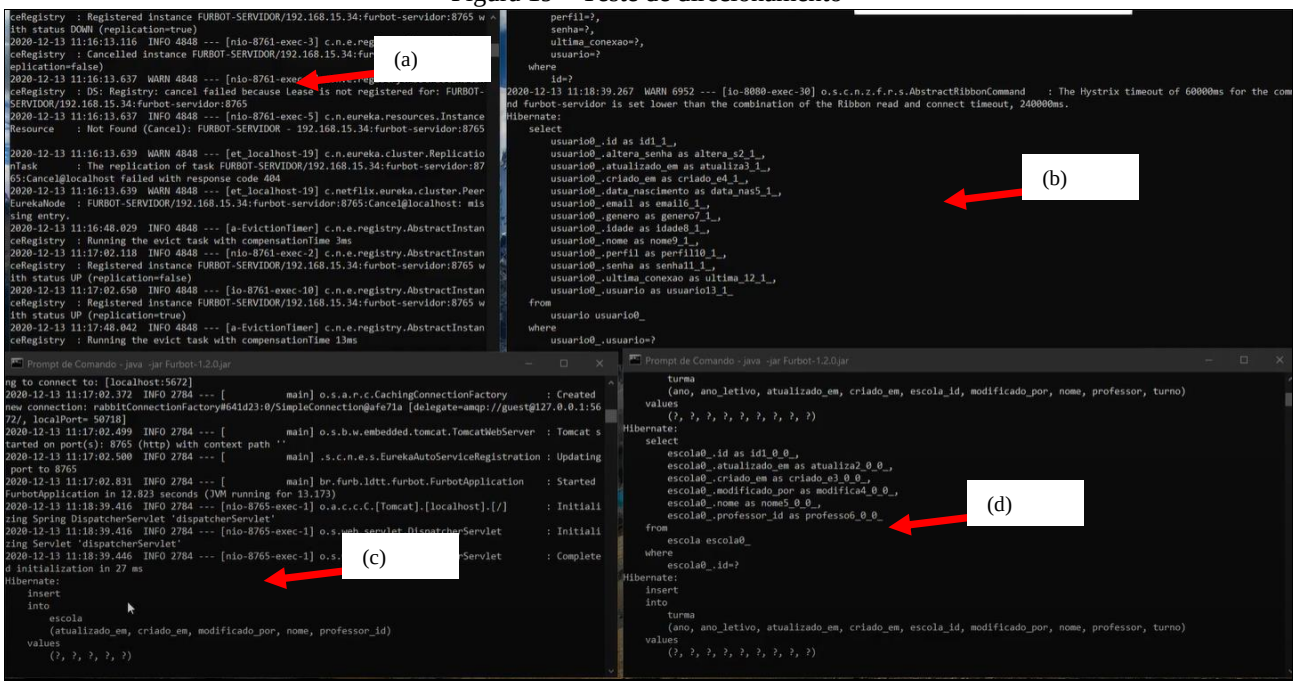
Fonte: elaborado pelo autor.

4.5 TESTE DIRECIONAMENTO

Neste experimento foi realizado em um computador local, para acompanhar o Loadbalancer. Na figura 19 temos quatro instancias rodando, Figura 19(a) o Discovery, Figura 19(b) o Gateway, Figura 19(c) e Figura 19(d) mostram duas instâncias do microsserviço do Furbot com logs de buscas na base de dados, que foram executadas de requisições diferentes de forma paralelas no dois microsserviços mesmo que o *endpoint* seja o mesmo para as duas chamadas.

Quando uma das instancias é desligada, seja ela intencional ou não, o fluxo é desviado para o microsserviço ativo no momento, mas toda via, se não existir um serviço disponível para o processamento acarretara uma exceção lançada para o usuário da API informando um erro HTTP 500.

Figura 19 – Teste de direcionamento



Fonte: elaborado pelo autor.

5 CONCLUSÕES

O presente trabalho descreveu a especificação, principais aspectos da implementação e os resultados da disponibilização de uma infraestrutura escalável baseada em microsserviços para suportar uma carga de demanda projetada para o projeto Furbot, que está limitada através de configurações a 10 mil requisições por segundos, o que é possível atingir em uma máquina com de 15GB de RAM e custo aproximado de R$ 445,64 considerando o dólar a R$ 5,48. O projeto foi desenvolvido utilizando a Netflix OSS juntamente com Spring Boot que se mostrou bastante estável, e com uma comunidade com muitos fóruns de resoluções de problemas.

Semelhantemente a API disponibilizada no presente trabalho, abstraiu toda a arquitetura interna, onde o jogo e o Frontend da aplicação somente precisam chamar o Gateway, para suas solicitações serem atendidas. Tal funcionalidade permitiu o desenvolvimento de um dashboard com métricas em tempo real dos jogos para os professores acompanharem seus alunos, mesmo que de forma sucinta, é possível conhecer o perfil de cada jogador com base nos seus resultados obtidos através da API.

Contudo a arquitetura conseguiu atingir os objetivos de performance e escalabilidade necessários considerando-se um pico de demanda de até 1000 usuários simultâneos em testes. Esta estrutura quando validada no atual servidor do projeto possibilita projetar que a atual infraestrutura consegue atender a uma demanda inicial de aproximadamente 1.500 usuários simultâneos, mas os testes realizados em nuvem caracterizam que bastará migrar os serviços para a nuvem que toda a estrutura continuará funcionando sem mudanças. Neste sentido a perspectiva de coleta de um grande volume de dados com vistas a implementação futura do módulo de Learning Analytics foi atendida. A validação de pior caso considerando os serviços instalados em diferentes regiões demográficas demonstrou que quando o tempo de latência é alto entre o Gateway e o Discovery há um impacto importante na performance em função da alta taxa de comunicação entre estes dois componentes.

5.1 EXTENSÕES E MELHORIAS

Como extensões e melhorias futuros sugere-se:

a) Aprofundar os estudos sobre a questão da latência quando os serviços são instalados em regiões demográficas distantes;
b) Melhorar a performance de processamento assíncrono no RabbitMQ;
c) Desenvolver e incorporar à arquitetura o módulo de Learning Analytics.

REFERÊNCIAS

ARAÚJO, Vania Carvalho de. **O jogo no contexto da educação psicomotora**. São Paulo: Cortez, 1992. p.106.

BARROZO, Lucas Moura. **Descoberta semântica de microservices em contêineres**. 2016. 109 f. TCC (Graduação) - Curso de Curso de Ciência da Computação, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2016. Disponível em: https://repositorio.ufsc.br/xmlui/handle/123456789/171422. Acesso em: 09 mar. 2020.

CARVALHO, A.M.C. et al. (Org.). **Brincadeira e cultura**:viajando pelo Brasil que brinca. São Paulo: Casa do Psicólogo, 1992. P.14.

CHIARADIA, Luiz Felipe Correa; MACEDO, Douglas Dyllon Jeronimo; DUTRA, Moisés Lima. Uma proposta de arquitetura de microsserviços aplicada em um sistema de CRM social. **Encontros Bibli**: revista eletrônica de biblioteconomia e ciência da informação, [S.L.], v. 23, n. 53, p. 147-159, 6 set. 2018. Universidade Federal de Santa Catarina (UFSC). http://dx.doi.org/10.5007/1518-2924.2018v23n53p147. Disponível em: https://periodicos.ufsc.br/index.php/eb/article/view/1518-2924.2018v23n53p147. Acesso em: 10 maio 2020.

DRAGONI, Nicola; GIALLORENZO, Saverio; LAFUENTE, Alberto Lluch; MAZZARA, Manuel; MONTESI, Fabrizio; MUSTAFIN, Ruslan; SAFINA, Larisa. Microservices: yesterday, today, and tomorrow. **Present And Ulterior Software Engineering**, [S.L.], p. 195-216, 2017. Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-67425-4_12. Disponível em: https://link.springer.com/chapter/10.1007%2F978-3-319-67425-4_12. Acesso em: 15 jun. 2020.

IFENTHALER, Dirk. Learning Analytics. In: SPECTOR, J. Michael. **The SAGE Encyclopedia of Educational Technology**. Thousand Oaks,ca: Sage Publications, Inc., 2015. p. 448-451. Disponível em: doi: 10.4135/9781483346397.n187. Acesso em: 7 nov. 2020.

LIMA, Leonardo Ross Torquato**. Implementação de uma arquitetura baseada em microserviços**. 2015. 77 f. TCC (Graduação) - Curso de Sistemas da Informação, Fundação de Ensino Eurípides Soares da Rocha, Marília, SP, 2015. Disponível em: https://aberto.univem.edu.br/bitstream/handle/11077/1381/TCC%20LRTL%20CORRE%c3%87%c3%95ES%20BANCA.pdf?sequence=1&isAllowed=y. Acesso em: 10 abr. 2020.

Kohler, Luciana P. de Araújo et al. **Uso da metodologia de rotação por estações com a computação desplugada**. In: VIII Congresso Brasileiro de Informática na Educação (CBIE), Anais dos Workshops do VIII Congresso Brasileiro de Informática na Educação (WCBIE 2019).

MATTOS, Mauro; KOHLER, Luciana P. de Araújo; ZUCCO, Fabrícia Diurex; WUO, Andrea; SANTOS, Bruno F. F.; FRONZA, Leonardo; SILVEIRA, Heitor Ugarte Calvet da; GIOVANELLA, Gian Carlo; LARGURA, Liz Rios; MELO, Jéssica Maria de. Aplicação de Game Design na Refatoração de um Jogo com Foco no Pensamento Computacional.**Anais Estendidos do Simpósio Brasileiro de Fatores Humanos em Sistemas Computacionais (IHC)**, [S.L.], p. 1-2, 11 out. 2019. Sociedade Brasileira de Computação - SBC. http://dx.doi.org/10.5753/ihc.2019.8379.

PINA, Fábio Figueiredo. **Monitoria de Arquiteturas de Micro-serviços**. 2018. 70 f. Dissertação (Mestrado) - Curso de Mestrado em Engenharia Informática, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, Coimbra, Portugal, 2018. Disponível em: https://eg.uc.pt/bitstream/10316/83561/1/FINAL_THESIS.pdf. Acesso em: 12 abr. 2020.

SCHLÖGL, Lucas E. et al. **Ensino do Pensamento Computacional na Educação Básica**. Revista de Sistemas e Computação, Salvador, v. 7, n. 2, p.233-304, dez. 2017.

SHARMA, Tushar; FRAGKOULIS, Marios; SPINELLIS, Diomidis. **Does your configuration code smell?**. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, 2016. p. 189-200. SCHOENFELD. A. H. (1985) Mathematical Problem Solving. New York: Academic Press.

SILVEIRA, Sidnei R. **Estudo de uma Ferramenta de Autoria Multimídia para a Elaboração de Jogos Educativos**, Porto Alegre, 1999. Disponível em: http://hdl.handle.net/10183/26551. Acesso em: 06 abril. 2020

TRIDAPALLI, Joan Gianesini. **Pensamento computacional e gamification**: relato de um experimento na plataforma furbot. 2017. 16 f. TCC (Graduação) - Curso de Ciência da Computação, Sistemas e Computação, Universidade Regional de Blumenau, Blumenau, 2017. Disponível em: http://dsc.inf.furb.br/arquivos/tccs/monografias/2019_1_joan-gianesini-tridapalli_monografia.pdf. Acesso em: 15 maio 2020.

VAHLDICK, A.; MATTOS, M. M. Relato de uma Experiência no Ensino de Algoritmos e Programação Utilizando um Framework Lúdico. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 19., 2008, Fortaleza. **Anais...** Fortaleza, 2008. Disponível em: <https://www.researchgate.net/publication/280879239_Relato_de_uma_Experiencia_no_Ensino_de_Algoritmos_e_Programacao_Utilizando_um_Framework_Ludico>. Acesso em: 19 jun. 2019.

WALPITA, P.a. **Defense In-depth security framework for Netflix OSS Micro Services**. 2017. 107 f. Tese (Doutorado) - Curso de Degree Of Master Of Science In Information Security, University Of Colombo School Of Computing, Sri Lanka, 2017. Disponível em: http://documents.ucsc.lk/jspui/bitstream/123456789/4039/1/2014MIS023_Final_Thesis.pdf . Acesso em: 25 maio 2020.

WEBER, Paulo Matheus Baehr. **Middleware com escalonamento de aplicações**. 2016. 66 f. TCC (Graduação) - Curso de Ciência da Computação, 16 Universidade Regional de Blumenau, Blumenau, Sc, 2016. Disponível em: http://www.inf.furb.br/~pericas/orientacoes/EscalonamentoAplicacoes2016.pdf. Acesso em: 13 abr. 2020.