

INSTITUTO FEDERAL SUL-RIO-GRANDENSE - IFSUL
CÂMPUS CHARQUEADAS
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

MARIA EDUARDA CARVALHO LEMOS

**HEALTHGATE: UM GATEWAY PARA
INTEROPERABILIDADE E INTEGRAÇÃO DE
APLICAÇÕES DE SAÚDE DIGITAL**

**Orientador: Prof. Dr. André Luís Del
Mestre Martins**

CHARQUEADAS, 2025

RESUMO

O gerenciamento e a coleta de dados médicos sempre foram assuntos fundamentais na área da saúde. Com o avanço tecnológico, a digitalização dos registros médicos tem ganhado cada vez mais destaque nesse meio. Embora essa transformação possibilite uma maior eficiência nos processos clínicos e administrativos, ela também apresenta desafios, como a privacidade dos dados e a interoperabilidade entre sistemas de saúde. O grupo de pesquisa IF4Health, do IFSUL Campus Charqueadas, desenvolveu soluções inovadoras, como FASS-ECG, um sistema de APIs para a digitalização e transmissão de eletrocardiogramas (ECGs), o IF-Cloud, um software que automatiza scripts e processa dados médicos e o H2Cloud, um sistema de autenticação baseado no Smart on FHIR. Entretanto, para obter uma integração eficiente entre esses três sistemas, é necessária uma solução centralizada, capaz de otimizar o roteamento de requisições entre eles. Neste trabalho, foi desenvolvido e homologado o HealthGate, um gateway de APIs baseado no padrão FHIR (Fast Healthcare Interoperability Resources), capaz de intermediar e gerenciar requisições entre os sistemas do ecossistema IF4Health. O sistema permite o cadastro dinâmico de rotas, conta com interface gráfica intuitiva e oferece monitoramento em tempo real das requisições, promovendo integração, segurança e observabilidade. O código-fonte foi disponibilizado publicamente no GitHub e o sistema encontra-se implantado na infraestrutura do IFSul.

Palavras-chave: Gateway. Saúde digital. APIs médicas. Interoperabilidade.

ABSTRACT

The management and collection of medical data have always been fundamental in the healthcare field. With technological advances, the digitization of medical records has become increasingly prominent. While this transformation enhances clinical and administrative efficiency, it also introduces challenges such as data privacy and interoperability among health systems. The IF4Health research group at IFSUL Charqueadas Campus has developed innovative solutions including FASS-ECG, an API system for digitizing and transmitting electrocardiograms (ECGs); IF-Cloud, software that automates scripts and processes medical data; and H2Cloud, an authentication system based on the SMART on FHIR protocol. However, to ensure efficient integration among these three systems, a centralized solution is required to optimize request routing. In this project, HealthGate was developed and validated—a FHIR-based API gateway capable of mediating and managing requests across the IF4Health ecosystem. The system supports dynamic route registration, features an intuitive graphical interface, and offers real-time monitoring of requests, fostering integration, security, and observability. The source code is publicly available on GitHub, and the system is deployed on the IFSUL infrastructure.

Keywords: Gateway, Digital health, Medical APIs, Interoperability.

LISTA DE FIGURAS

1.1	Arquitetura do ecossistema do grupo IF4Health antes e depois do Health-Gate centralizando as requisições entre os sistemas.	9
2.1	Captura de tela do Visualizador de ECG mostra a onda ECG ao centro e a frequência cardíaca no canto inferior direito.	19
3.1	Arquitetura do hub de integrações GIRLS	23
4.1	Tela cadastro de rotas HealthGate	30
4.2	Tela de rotas cadastradas	31
5.1	Sistemas FASS-ECG e IF-Cloud reproduzidos localmente	36
5.2	Sistema H2Cloud reproduzido localmente	36
5.3	Estrutura de rotas no Postman	37
5.4	Tela de login HealthGate	38
5.5	Tela de logs HealthGate	38
5.6	Otimização do código-fonte do Visualizador de ECG (DUARTE, 2024) para evitar roteamento do front-end	39

LISTA DE CÓDIGOS-FONTE

2.1	Exemplo de recurso FHIR do tipo <i>Observation</i> descrevendo um ECG	12
2.2	Exemplo do JSON resposta da requisição na rota <i>/.well-known/smart-configuration</i>	14
2.3	Exemplo do corpo de requisição API <i>/auth/register</i>	14
2.4	Exemplo do corpo de requisição API <i>/auth/authorize</i>	15
2.5	Exemplo do JSON de resposta API <i>/auth/token</i>	15
2.6	Exemplo do JSON de configuração da API IF-Cloud	18
4.1	Corpo da requisição do cadastro de rota para consulta de observações em inter- valo de tempo	28
4.2	Exemplo de <i>payload</i> das requisições de registro e login	31

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
AWS	Amazon Web Services
ECG	Eletrocardiograma
EHR	Electronic Health Records
FHIR	Fast Healthcare Interoperability Resources
HTTP	Hypertext Transfer Protocol
IFSul	Instituto Federal Sul-rio-grandense
IoT	Internet of Things
JSON	JavaScript Object Notation
REST	Representational State Transfer
TSI	Tecnólogo em Sistemas para Internet

SUMÁRIO

1	INTRODUÇÃO	8
1.1	Objetivo	9
1.1.1	Objetivos específicos	9
1.2	Organização do Trabalho	10
2	REFERENCIAL TEÓRICO	11
2.1	O Padrão FHIR	11
2.2	SMART on FHIR	13
2.3	API FASS-ECG	15
2.4	API IF-Cloud	17
2.5	Front-end - Visualizador de exames de eletrocardiograma	18
2.6	Gateway	19
2.6.1	Kong Gateway	20
2.6.2	Cadastro de rotas no Kong	21
3	TRABALHOS RELACIONADOS	22
3.1	IBM API Connect e API Hub para Saúde Digital	22
3.2	GIRLS, a Gateway for Interoperability of electronic health Record in Low-cost System	22
3.3	Comparação de Desempenho entre Soluções de Interoperabilidade	24
3.4	Comparativo entre os trabalhos relacionados	24
4	METODOLOGIA E SOLUÇÃO PROPOSTA	27
4.1	Definição de tecnologias e estrutura do sistema	27
4.2	Rotas disponíveis do HealthGate	27
4.2.1	Gestão e Cadastro de rotas via Requisição Web	28
4.2.2	Gestão e Cadastro de rotas via Interface Gráfica	30
4.2.3	Rotas para usuário	31
4.2.4	Rotas criadas por usuários e roteamento de requisições	32
4.3	Expressão Regular - Regex	33
5	RESULTADOS E DISCUSSÕES	35
5.1	Reprodução dos sistemas FASS-ECG, IF-Cloud e H2Cloud	35
5.2	Prova de Conceito	35
5.3	Interface Gráfica	37
5.4	Apontamento do Front-end Visualizador ao HealthGate	38

6 CONCLUSÕES E TRABALHOS FUTUROS	40
REFERÊNCIAS	42

1 INTRODUÇÃO

A coleta de dados médicos é uma demanda que está sempre crescendo, impulsionada pelo avanço tecnológico e pela necessidade de registros precisos e acessíveis (DUARTE, 2024). A digitalização dos registros médicos tornou-se essencial para diversos aspectos da assistência à saúde, desde diagnósticos até o acompanhamento contínuo de pacientes, trazendo maior eficiência nos processos clínicos e administrativos. Entretanto, essa digitalização também apresenta desafios como a segurança e privacidade das informações, além da necessidade de interoperabilidade entre os sistemas de saúde (OEMIG; SNELICK, 2016).

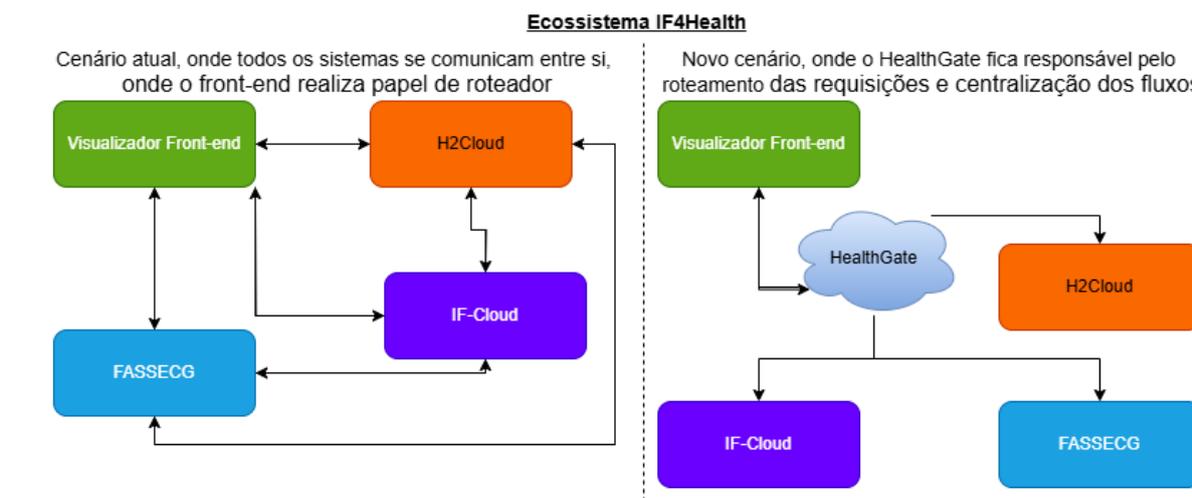
O grupo de pesquisa IF4Health, do IFSUL Campus Charqueadas, trabalha com tecnologias direcionadas à saúde, buscando integrar tecnologias emergentes para aprimorar a conectividade e eficiência dos sistemas biomédicos. Dentre os trabalhos do grupo, destacam-se o FASS-ECG (PEREIRA et al., 2023), um sistema que utiliza o padrão FHIR para digitalizar, transmitir e armazenar eletrocardiogramas (ECGs) de longa duração, direcionado para operações CRUD (*Create-Read-Update-Delete*), o IF-Cloud (VIEIRA et al., 2024), que automatiza scripts Python utilizando dados FHIR, possuindo também uma camada de processamento a esse ecossistema digital, uma plataforma visualizadora de front-end (DUARTE, 2024), com objetivo de trazer graficamente os dados médicos e ECGs e o H2Cloud (SANTOS et al., 2023), um sistema de autenticação de aplicações relacionadas a saúde que sejam compatíveis com SMART on FHIR.

Esses quatro sistemas, embora robustos, ainda necessitam de um componente central que facilite a comunicação entre eles, seja de requisições feitas do front-end para o back-end ou em requisições feitas entre as três nuvens de processamento e autenticação também. As aplicações web não deveriam lidar com o roteamento de requisições entre as nuvens FASS-ECG, IF-Cloud e H2Cloud.

Sendo assim, o desenvolvimento de um gateway surge como uma solução essencial para centralizar e simplificar essa integração, garantindo eficiência e segurança no gerenciamento e transmissão dos dados. A solução permite o roteamento inteligente das requisições entre os sistemas do ecossistema IF4Health, garantindo que aplicações front-end possam interagir com os serviços por meio de um ponto único de entrada, padronizado e seguro, como demonstrado na Figura 1.1.

Além de cumprir seu papel como middleware, o HealthGate conta com outras funcionalidades desejadas em um gateway: interface gráfica de administração, painel de métricas para monitoramento e lógica de roteamento baseada em expressões regulares. Assim HealthGate se apresenta como uma ferramenta essencial para garantir a rastreabilidade e eficiência dentro do ecossistema.

Figura 1.1: Arquitetura do ecossistema do grupo IF4Health antes e depois do HealthGate centralizando as requisições entre os sistemas.



Fonte: Autoria Própria

1.1 Objetivo

O objetivo geral deste trabalho é desenvolver e homologar o gateway HealthGate para aplicações de saúde digital, visando a centralização e otimização do roteamento de requisições entre diferentes nuvens no ecossistema de saúde digital.

1.1.1 Objetivos específicos

O presente trabalho tem os seguintes objetivos específicos:

1. Estudar e compreender o padrão de interoperabilidade FHIR (HL7, 2019) que todo o ecossistema de saúde digital segue;
2. Analisar detalhadamente os trabalhos anteriores do grupo IF4Health: FASS-ECG, IF-Cloud e H2Cloud, e entender sua arquitetura e funcionamento;
3. Realizar *deploy* dos três sistemas do grupo IF4Health, assegurando que estejam configurados corretamente para integração com o gateway;
4. Desenvolver e integrar o gateway, estabelecendo uma comunicação eficiente com os sistemas FASS-ECG, IF-Cloud e H2Cloud, garantindo o roteamento de requisições entre as duas nuvens;
5. Projetar e conduzir experimentos que validem o funcionamento do gateway, executando uma sequência de requisições para validar o roteamento e a troca de dados;
6. Publicar o código-fonte no repositório público do GitHub (<https://github.com/if4health>), com documentação completa (README.md) para facilitar o uso e a reprodução dos experimentos;

7. Realizar o deploy do gateway na infraestrutura do IFSul para garantir sua escalabilidade e disponibilidade.

1.2 Organização do Trabalho

O Capítulo 2 apresenta os conceitos específicos necessários para a compreensão deste trabalho e os trabalhos anteriores do grupo de pesquisa IF4Health. O Capítulo 3 revisa os trabalhos relacionados. O Capítulo 4 detalha a solução proposta destacando a construção das rotas e cadastro das mesmas. O Capítulo 5 apresenta os resultados deste trabalho. Finalmente, o Capítulo 6 apresenta as conclusões e percepção de trabalhos futuros

2 REFERENCIAL TEÓRICO

Este Capítulo sumariza os principais conceitos para o pleno entendimento deste trabalho. A Figura 1.1 mostra que o gateway proposto neste trabalho integra um ecossistema de variadas aplicações já existentes. Logo, é necessário compreender cada um das aplicações do ecossistema e a forma como elas se comunicam entre si.

Inicialmente, Seção 2.1 apresenta a norma de registro de dados em Saúde. Em seguida, a Seção 2.2 o protocolo e os *endpoints* para autenticação em sistemas de saúde. A seguir, Seções 2.3 e 2.4 apresentam as APIs que compõem a arquitetura de microsserviço do ecossistema de aplicações que HealthGate vai integrar. Na sequência, a Seção 2.5 destaca uma aplicação front-end a ser refatorada para se beneficiar do desenvolvimento de HealthGate. Finalmente, Seção 2.6 apresenta uma visão geral sobre gateways aplicado à arquiteturas de software de microsserviços e monolitos.

2.1 O Padrão FHIR

O padrão *Fast Healthcare Interoperability Resources*, popularmente conhecido como FHIR, foi desenvolvido pela HL7 International e um dos seus objetivos principais é facilitar a interoperabilidade entre sistemas na área da saúde. Por meio de estruturas padronizadas para a troca de dados clínicos e administrativos, o FHIR promove uma comunicação eficiente e consistente entre diferentes softwares (GOMES et al., 2019).

O FHIR combina características de padrões anteriores, como HL7 V2, HL7 V3 e CDA utilizando tecnologias modernas da web, incluindo REST, JSON e XML e tornando sua utilização mais acessível e eficiente no ambiente de desenvolvimento (HL7, 2019). Essa abordagem baseada em recursos permite a modelagem de entidades comuns no contexto da saúde, como pacientes, consultas, exames, medicamentos e dispositivos médicos de forma padronizada.

Os recursos do FHIR são flexíveis e podem ser combinados para atender a cenários mais complexos. A arquitetura RESTful do FHIR possibilita a realização de operações padrão como criação (POST), leitura (GET), atualização (PUT, PATCH) e exclusão (DELETE) de dados clínicos. Esse formato é muito utilizado para integrar sistemas modernos, reduzindo custos de desenvolvimento e sustentação.

O padrão FHIR é aplicado em diversos cenários, podendo destacar:

- Troca de informações clínicas entre hospitais, laboratórios e sistemas governamentais;
- Compartilhamento de dados médicos em prontuários eletrônicos;
- Integração de dispositivos IoT para monitoramento remoto de pacientes;
- Implementação de serviços específicos, como o gerenciamento de registros de eletrocardiogramas contínuos no contexto da telemedicina.

A interoperabilidade garantida pelo FHIR é essencial no projeto HealthGate, que utiliza

APIs baseadas nesse padrão para unificar dados provenientes de diferentes sistemas de saúde, pois existem diversas implementações de APIs FHIR de diferentes autores (AYAZ et al., 2021). Ao utilizar o padrão FHIR, o projeto HealthGate poderia, em tese, rotear para qualquer API FHIR. Assim, o padrão desempenha um papel central na construção de soluções escaláveis e modernas, essenciais para a transformação digital na saúde.

Código-fonte 2.1: Exemplo de recurso FHIR do tipo *Observation* descrevendo um ECG

```
{
  "resourceType": "Observation",
  "status": "preliminary",
  "subject": {
    "reference": "Patient/f001",
  },
  "device": {
    "display": "12 lead EKG Device Metric"
  },
  "component": [
    {
      "code": {
        "coding": [
          {
            "display": "MDC_ECG_ELEC_POTL_I"
          }
        ]
      },
      "valueSampledData": {
        "period": 10,
        "lowerLimit": -3300,
        "upperLimit": 3300,
        "data": "2041 2043 2037 ... 2066 2077 2073"
      }
    }
  ]
}
```

Fonte: Adaptado de PEREIRA et al. (2023)

Como o estudo-de-caso de HealthGate são serviços relacionados ao registro, envio e visualização de ECGs de forma remota, o Código-Fonte 2.1 mostra um exemplo simplificado de um ECG registrado como um recurso FHIR em formato JSON. Dentre as diversas chaves do recurso FHIR que descrevem o ECG, destacam-se:

- `resourceType`: Define o tipo do recurso. No exemplo, indica que este recurso é uma *Observation*, ou seja, uma observação clínica.
- `component`: É um array que representa um ECG possuindo até 12 derivações, onde cada componente representa uma dessas derivações.
- `valueSampledData`: Representa os dados do ECG associados a uma derivação espe-

cífica de um componente. Contém os valores do sinal, intervalo de amostragem, e limites de amplitude.

- `period`: Representa o intervalo de tempo entre os pontos de amostragem do sinal.
- `upper/lowerLimit`: Valor máximo e mínimo permitido para o sinal registrado, usado para identificar o limite superior e inferior da amplitude.
- `data`: Sequência de valores que representam o sinal registrado do ECG em uma derivação específica. Esses valores, como mostrado no exemplo, são números inteiros correspondendo à amplitude do sinal em diferentes pontos no tempo.

2.2 SMART on FHIR

O protocolo SMART on FHIR (Substitutable Medical Applications and Reusable Technologies on FHIR) surgiu como uma solução recomendada para garantir autenticação e controle de acesso em aplicações de saúde que utilizam o padrão FHIR. Este padrão propõe a estruturação e o envio de dados clínicos de forma interoperável, como citado anteriormente, porém ele não define especificações sobre segurança e autenticação, o que se torna crítico considerando legislações de proteção de dados, como a LGPD no Brasil e a GDPR na Europa (SANTOS et al., 2023).

O SMART on FHIR propõe uma solução baseada no protocolo OAuth2, permitindo que aplicações web com interface gráfica se conectem a servidores FHIR de maneira segura e autenticada. Através de fluxos de autenticação e autorização, é possível garantir que apenas aplicações devidamente autorizadas possam acessar recursos clínicos sensíveis. Ademais, a autenticação é realizada de forma RESTful, alinhando-se com as práticas modernas de desenvolvimento de software.

No estudo de SANTOS et al. (2023), foi desenvolvido o servidor H2Cloud, um sistema baseado em nuvem que implementa o protocolo SMART on FHIR para autenticação de de diferentes tipos de aplicações FHIR. O ecossistema de microserviços no qual HealthGate vai ser integrado prevê a autenticação com SMART on FHIR, portanto apresenta-se os principais *endpoints* utilizados no processo de autenticação a seguir:

- `GET /.well-known/smart-configuration`: Retorna um arquivo JSON com as configurações iniciais do servidor SMART, como os endpoints de autorização e troca de token. O Código-Fonte 2.2 ilustra um exemplo de retorno deste *endpoint*.
- `POST /auth/register`: Este endpoint é utilizado apenas por aplicações com interface de usuário para realizar o registro inicial no H2Cloud, no Código-Fonte 2.3 temos um exemplo de corpo da requisição.
- `POST /auth/authorize`: Endpoint utilizado para autorizar a aplicação com base no escopo e no paciente informado, onde é retornado um code que será utilizado na chamada

Código-fonte 2.2: Exemplo do JSON resposta da requisição na rota */.well-known/smart-configuration*

```
{
  "authorization_endpoint": "https://if4health.charqueadas.ifsul.edu.br/biosignalinfhir/auth/register",
  "token_endpoint": "https://if4health.charqueadas.ifsul.edu.br/biosignalinfhir/auth/token",
  "token_endpoint_auth_methods_supported": [
    "private_key_jwt"
  ],
  "grant_types_supported": [
    "authorization_code",
    "client_credentials"
  ],
  "scopes_supported": [
    "patient/*.rs"
  ],
  "response_types_supported": [
    "code"
  ],
  "capabilities": [
    "client-confidential-asymmetric"
  ],
  "code_challenge_methods_supported": [
    "S256"
  ],
  "token_endpoint_auth_signing_alg_values_supported": [
    "RS384",
    "ES384"
  ]
}
```

Código-fonte 2.3: Exemplo do corpo de requisição API */auth/register*

```
{
  "redirect_uri": "/api-docs",
  "scope": "patient/*.rs",
  "token_endpoint": "/auth/token",
  "client_id": "gateway",
  "aud": "placeholder",
  "state": "placeholder"
}
```

abaixo.

- POST */auth/token*: Após a autorização, o gateway utiliza esse endpoint para resgatar token de acesso, que será utilizado nas requisições futuras.

No contexto do HealthGate, o processo de autenticação segue um fluxo próprio para aplicações backend e dispositivos IoT, conforme previsto no protocolo SMART on FHIR. As rotas

Código-fonte 2.4: Exemplo do corpo de requisição API /auth/authorize

```
{
  "redirect_uri": "/api-docs",
  "scope": "patient/*.rs",
  "token_endpoint": "/auth/token",
  "client_id": "<id cadastrado>",
  "paciente_id": "<id cadastrado>",
  "aud": "placeholder",
  "state": "placeholder"
}
```

Código-fonte 2.5: Exemplo do JSON de resposta API /auth/token

```
{
  "access_token": "<token gerado>",
  "token_type": "bearer",
  "expires_in": 3600,
  "client_id": "gateway"
}
```

/auth/register e /auth/authorize são utilizadas em cenários onde há interação com o usuário, como aplicações web ou mobile, permitindo o registro e a autorização explícita do usuário final. Já no caso de aplicações sem interface de usuário, como o HealthGate, utiliza-se o método de pré-autorização, no qual o *client_id* e o *client_code* são previamente cadastrados e armazenadas no banco de dados da aplicação. Assim, o gateway obtém tokens de acesso diretamente através da rota /auth/token, utilizando essas credenciais fixas. Essa abordagem é amplamente empregada em serviços backend e dispositivos IoT, garantindo autenticação segura e automatizada sem a necessidade de fluxos interativos de usuário.

Com o token de acesso em mãos, o HealthGate consegue realizar chamadas autenticadas às APIs FHIR disponíveis, como as do sistema FASS-ECG. A autenticação é um pré-requisito indispensável para o acesso às operações de leitura, escrita e atualização de recursos clínicos sensíveis que estão protegidos nesse ecossistema de saúde digital.

2.3 API FASS-ECG

O FASS-ECG (PEREIRA et al., 2023) é um sistema de APIs baseado no padrão FHIR, projetado para viabilizar o streaming e o armazenamento de registros de eletrocardiogramas contínuos de longo prazo, com foco em aplicações de telemedicina. A solução aborda desafios relacionados à integração de sistemas IoT e ao grande volume de dados trafegados, implementando protocolos que permitem a transmissão distribuída de ECGs.

Nesse cenário, os dispositivos IoT podem dividir os registros de ECG em segmentos e enviá-los por requisições HTTP separadas que ao serem processadas em conjunto, formam um recurso de observação FHIR unificado. O sistema suporta funcionalidades como leitura

e gravação de registros de ECG, utilizando tecnologias de armazenamento baseadas em particionamento e objetos, como o AWS S3, para assegurar escalabilidade e baixa latência nas respostas.

Como o FASS-ECG é uma das APIs que será conectada ao HealthGate, é necessário detalhar os *endpoints* FHIR de leitura e escrita de ECGs.

- **POST /Observation**: Cria um novo registro de ECG e marca o início do streaming. Como podemos observar no exemplo de request no código 2.1, o campo *status* é marcado como *preliminary* representando o estado do recurso como inicial. Além disso, cada *component* representa uma derivação do ECG.
- **PATCH /Observation/{id}**: Adiciona novas amostras ao ECG previamente cadastrado. Ao mandar um POST, como mencionado anteriormente, é retornado um *id* do ECG cadastrado, que será utilizado para adicionar novas derivações. Esta operação segue o padrão JSON Patch, que define operações para atualizar partes específicas do recurso JSON sem substituí-lo completamente.
- **PUT /Observation/{id}**: Finaliza o streaming, enviando no corpo da request apenas o campo *status* como *final*;
- **GET /Observation/{id}**: Leitura do ECG completo, retornando os dados e as derivações cadastradas anteriormente.
- **GET /Observation/{id}/data/{minute}**: Leitura de um minuto específico do ECG, conforme o parâmetro passado.
- **GET /Observation/{id}/data/{start}/{end}**: Leitura de ECG em um intervalo de tempo especificado em minutos.

Para garantir a segurança e o controle de acesso às rotas do FASS-ECG, todas as requisições devem ser realizadas utilizando um *Bearer Token* válido. Na versão utilizada neste trabalho, o FASS-ECG foi integrado diretamente ao H2Cloud, de modo que seus recursos FHIR estão protegidos por um fluxo de autenticação baseado no protocolo SMART on FHIR. Esse mecanismo de segurança não estava presente na versão original descrita no artigo de (PEREIRA et al., 2023)pereira2023fass, sendo uma melhoria implementada para atender aos requisitos de segurança e controle de acesso.

O processo de autenticação consiste em chamar a rota `/auth/token` no H2Cloud, enviando o *client_id* e um *code* previamente cadastrado. Com essas credenciais fixas, o token de acesso é retornado e possui um tempo de expiração definido, devendo ser incluído no cabeçalho das requisições (*Authorization: Bearer <token>*) para que as chamadas às rotas do FASS-ECG sejam aceitas e processadas corretamente.

A utilização dessa arquitetura no sistema oferece uma base flexível e escalável para aplicações de telemedicina, otimizando o armazenamento e a transmissão de registros de ECGs, além de garantir interoperabilidade com plataformas compatíveis com o padrão FHIR e assegurando os mecanismos de controle de acesso e proteção de dados sensíveis conforme as boas práticas

de segurança da informação.

2.4 API IF-Cloud

A API IF-Cloud foi desenvolvida para a prototipação e integração de dispositivos IoT e aplicações web no contexto da saúde digital, com foco na interoperabilidade entre sistemas utilizando o padrão FHIR. Essa aplicação permite que scripts em Python sejam carregados e executados para o processamento de dados de biosinais, incluindo tarefas como compressão de dados e cálculo de frequência cardíaca (VIEIRA et al., 2024). Por meio de uma interface gráfica, o IF-Cloud promove a integração de funcionalidades dentro de um ecossistema unificado de aplicações de saúde.

O sistema realiza as consultas de dados a partir de uma API que realiza operações CRUD no padrão FHIR, o que dispensa a necessidade de gerenciamento direto de bancos de dados. A API de CRUD pode ser a FASS-ECG (Seção 2.3) ou outra que siga o padrão FHIR. É importante destacar que o IF-Cloud foi previamente autorizado no servidor H2Cloud para poder realizar as requisições ao FASS-ECG. Para validar a interoperabilidade, o IF-Cloud foi conectado ao servidor de testes HAPI FHIR (CDR, 2019) que tem todo o padrão implementado e é bastante utilizado academicamente.

Para operacionalizar seu funcionamento, o IF-Cloud implementa duas categorias principais de rotas. Tais recursos foram implementados no HealthGate, onde as chamadas que são recebidas apontam para ele e, conseqüentemente, buscam informações no HAPI FHIR.

1. **Rotas diretas:** Permitem ao usuário testar scripts Python previamente carregados, sem realizar processamento nos dados oriundos da API de CRUD.
 - GET `/run_script/direct/:script_name`: Executa o script `script_name` que foi salvo no IF-Cloud sem os parâmetros de entrada.
 - POST `/run_script/direct/params`: Executa o script salvo com parâmetros de entrada. O nome do script e os parâmetros de entrada são passados por um JSON.
2. **Rota principal:** Realiza a execução do script com base em um recurso FHIR proveniente da API de CRUD, retornando o recurso processado. Sempre que uma requisição é recebida na rota principal, o IF-Cloud acessa os dados na API de CRUD e sobrescreve as informações processadas conforme especificado no script configurado.
 - POST `/run_script/operation`: Executa um script configurado com base em um JSON de configuração, modificando o conteúdo de uma chave de um recurso FHIR oriundo da API de CRUD.

O Código-Fonte 2.6 ilustra a configuração do IF-Cloud que contem um script Python que calcula a Frequência Cardíaca (medida em Batimentos por Minuto – BPM) para um determi-

Código-fonte 2.6: Exemplo do JSON de configuração da API IF-Cloud

```

{
  "resourceType": "Observation",
  "id": "64e8f55031cb42fdd73a3b5a",
  "scriptName": "calcBPM.py",
  "returnOnlyFieldsComponents": true,
  "components": [
    {
      "index": "0",
      "changeField": "data"
    }
  ]
}

```

Fonte: Adaptado de VIEIRA et al. (2024)

nado ECG salvo em uma API de CRUD:

- `resourceType`: especifica o tipo de recurso FHIR que o IF-Cloud está buscando na API de CRUD. No caso, refere-se ao FASS-ECG.
- `id`: identificador único do recurso FHIR que o IF-Cloud está consultando ou manipulando na API de CRUD.
- `scriptName`: nome do script que será executado para realizar alterações ou processamentos no recurso FHIR.
- `component`: define qual chave específica do recurso FHIR será alterada pelo script configurado em `scriptName`
- `index`: especifica o índice da chave `changeField` que será modificada no recurso FHIR.
- `changeField`: determina qual chave ou campo do recurso FHIR será alterado durante a execução do script.
- `returnOnlyFieldComponent`: indica se o IF-Cloud retornará apenas os campos alterados ou se fornecerá o recurso FHIR completo ao solicitante.

Essas funcionalidades tornam o IF-Cloud uma ferramenta versátil para atender às demandas de processamento de dados no ecossistema de saúde digital, promovendo automação de tarefas complexas.

2.5 Front-end - Visualizador de exames de eletrocardiograma

Um visualizador de exames de eletrocardiogramas é o principal componente de front-end para este estudo de caso. A solução disponível (DUARTE, 2024) simula um ECG tradicional analógico com um gráfico digital sob uma folha cartográfica e também exibir a onda do ECG em velocidade real, similar aos monitores multiparamétricos de hospitais Figura (2.1).

Figura 2.1: Captura de tela do Visualizador de ECG mostra a onda ECG ao centro e a frequência cardíaca no canto inferior direito.



Fonte: Adaptado de DUARTE (2024)

O visualizador de ECG utilizado neste trabalho utiliza tecnologias modernas como React e a biblioteca CanvasJS, permitindo a renderização de gráficos dinâmicos e interativos para representar os sinais de ECG em tempo real, garantindo ao usuário acessar e interagir com os exames de maneira prática e eficiente .

O visualizador de ECGs busca por recursos FHIR das APIs FASS-ECG e IF-Cloud para buscar, processar e exibir os dados de eletrocardiogramas. Conforme ilustrado na Figura 2.1, as ondas de ECG são obtidas do FASS-ECG a partir de observações previamente cadastradas. A frequência cardíaca é calculada e exibida com base em scripts pré-configurados executados pelas rotas do IF-Cloud. O visualizador de ECG atual se conecta diretamente às APIs FASS-ECG e IF-Cloud, realizando um trabalho de roteamento. Entretanto, após a implementação do HealthGate, o front-end será integrado diretamente ao gateway, garantindo um fluxo contínuo e padronizado de informações.

2.6 Gateway

O termo gateway, do inglês, significa “portão” ou “portal”. Como o próprio nome sugere, trata-se de um intermediário entre clientes e serviços, desempenhando um papel essencial em sistemas distribuídos. Sua principal função é gerenciar as requisições recebidas, aplicar regras de negócio, autenticar usuários, validar dados e encaminhar essas requisições para os destinos corretos (HAT, 2019). Mais do que um simples roteador, o gateway centraliza recursos, tornando a comunicação entre sistemas mais segura, eficiente e organizada, sendo indispensável em arquiteturas.

No desenvolvimento de sistemas, a escolha entre arquiteturas monolíticas e baseadas em microserviços é importante para a adoção de um gateway. Em arquiteturas monolíticas, onde todas as funcionalidades estão concentradas em um único bloco, o roteamento interno é simplificado. No entanto, conforme o sistema vai crescendo, a escalabilidade e a manutenção tornam-se desafiadoras, especialmente com o aumento de funcionalidades e APIs. Já nas arquiteturas baseadas em microserviços, onde o sistema é dividido em componentes independentes e especializados, o gateway é fundamental para gerenciar a comunicação entre esses serviços. Ele reduz a complexidade de múltiplas APIs ao fornecer um ponto único de entrada, simplificando o gerenciamento do ecossistema distribuído (HARRIS, 2024).

Embora o front-end seja responsável por interagir diretamente com os usuários, ele não deve assumir o papel de roteador entre as APIs. Delegar essa função ao front-end aumenta a complexidade do código, dificulta a manutenção e a implementação de novas funcionalidades, além de tornar a documentação mais trabalhosa. O gateway, por outro lado, é projetado para lidar com essas responsabilidades. Ele recebe as requisições do front-end, identifica os serviços e APIs apropriados e realiza o roteamento de forma eficiente. Dessa forma, o front-end pode se concentrar em sua função principal: entregar uma interface amigável e eficaz ao usuário.

Em sistemas compostos por serviços interconectados, o gateway atua como ponto central de integração, facilitando o roteamento de requisições em ecossistemas complexos de APIs. Ele permite que cada serviço opere de forma isolada, enquanto gerencia de forma centralizada a comunicação entre eles. Tudo isso é feito sem sobrecarregar os serviços individuais ou o front-end, como no caso deste trabalho. Assim, o gateway não apenas otimiza o fluxo de requisições, mas também contribui para a segurança e manutenção do sistema como um todo.

2.6.1 Kong Gateway

O Kong é um gateway de APIs construído sobre o NGINX, criado para gerenciar, proteger e escalar aplicações em microsserviços. Ele funciona como uma camada intermediária entre *clients* e os serviços de backend, realizando o roteamento de requisições, autenticações, monitoramento, entre outras funções (KONG, 2025). Ele é oferecido em duas versões:

- Kong Gateway (OSS) – Versão gratuita e open source, com funcionalidades principais como criação de serviços, rotas, e uso de plugins essenciais (autenticação, rate limiting, logging).
- Kong Gateway Enterprise – Versão comercial, que inclui recursos adicionais como:
 - Interface gráfica oficial (Kong Manager);
 - Controle de acesso com RBAC (Role-Based Access Control);
 - Monitoramento avançado e analytics;
 - Suporte empresarial e ferramentas de automação.

Para usuários da versão gratuita, pode-se utilizar a Konga, uma alternativa comunitária popular à interface gráfica, onde seu papel é facilitar o gerenciamento visual de rotas, serviços, plugins e consumidores a partir de um painel administrativo (PANTSEL, 2024).

2.6.2 Cadastro de rotas no Kong

O Kong utiliza uma arquitetura baseada em serviços e rotas:

- Um serviço representa uma API de backend.
- Uma rota define como esse serviço será exposto externamente, incluindo caminhos, métodos HTTP e regras de correspondência.

Na versão Enterprise com Kong Manager, o cadastro é feito de forma visual, com suporte direto a expressões regulares. Já na versão OSS, esse processo é feito por linha de comando ou pela interface Konga.

Exemplo de Cadastro com Kong OSS:

Suponha que desejamos cadastrar uma API de backend do FASS-ECG, cujo endpoint seja: `$_URL_FASS-ECG/Observation/{id}/data/{minute}`, onde os valores entre chaves (`{}`) são parâmetros variáveis na rota.

Para roteá-la corretamente, seria necessário:

- Cadastrar o serviço apontando para a URL base: `$_URL_FASS-ECG`
- Cadastrar a rota utilizando expressão regular para capturar os parâmetros `{id}` e `{minute}`.

No Kong OSS, isso deve ser feito via API Admin ou Konga, usando a seguinte expressão:

```
/Observation/(?<id>[^\s/]+)/data/(?<minute>[^\s/]+)
```

Embora o Kong seja uma solução robusta e forte no mercado, sua utilização exige um esforço adicional de configuração e conhecimento técnico, especialmente no que diz respeito ao uso de expressões regulares e à ausência de recursos gráficos nativos na versão gratuita. Além disso, a autenticação padrão oferecida não contempla nativamente os fluxos exigidos por protocolos específicos da saúde, como o SMART on FHIR, o que obriga a adoção de plugins adicionais.

Nesse contexto, o HealthGate se diferencia por ser uma solução desenvolvida especificamente para o ecossistema da saúde digital. Ele permite o cadastro de rotas por meio de uma interface gráfica intuitiva, sem a necessidade de expressões regulares. Os parâmetros da URL podem ser declarados diretamente em campos próprios, facilitando a criação e manutenção de rotas dinâmicas mesmo por usuários sem conhecimento técnico avançado. Além disso, a ferramenta possui recursos de monitoramento incorporado nativamente, como dashboards com métricas de acesso, taxa de sucesso e falhas nas requisições.

3 TRABALHOS RELACIONADOS

Por se tratar de um tema de trabalho bastante específico e voltado para a área da saúde, três trabalhos relacionados foram revisados. Ao final desta Seção, é apresentada uma análise comparativa entre a revisão desses estudos e o presente trabalho.

3.1 IBM API Connect e API Hub para Saúde Digital

O trabalho de SIU et al. (2024) traz uma abordagem prática e robusta para a publicação, segurança e gestão de APIs voltadas para a área da saúde digital utilizando as plataformas IBM API Connect e IBM API Hub. A proposta se baseia na integração de quinze APIs desenvolvidas no contexto do sistema Health Guardian, que permeia áreas como dor crônica, mobilidade e cognição. Essas APIs foram desenvolvidas em Flask, seguindo especificações OpenAPI, e após isso, integrados ao IBM API Connect para o gerenciamento do ciclo de vida das APIs e publicação do IBM API Hub.

O IBM API Connect possibilita o controle completo de autenticação, versionamento, testes, e segurança, com suporte a REST e GraphQL das APIs. Já o IBM API Hub funciona como uma vitrine pública que permite a descoberta e experimentação dessas APIs com ferramentas como Jupyter Notebook, Python, R, Kubeflow e Apache Airflow.

O estudo destaca o uso do Pain States API, um sistema que classifica pacientes com dor crônica em cinco estados clínicos a partir de dados de sensores móveis e relatos dos pacientes. A integração com IBM API Hub permite que esse tipo de API seja acessado via interface gráfica ou via código, o que promove a reutilização e adoção em ambientes clínicos e de pesquisa.

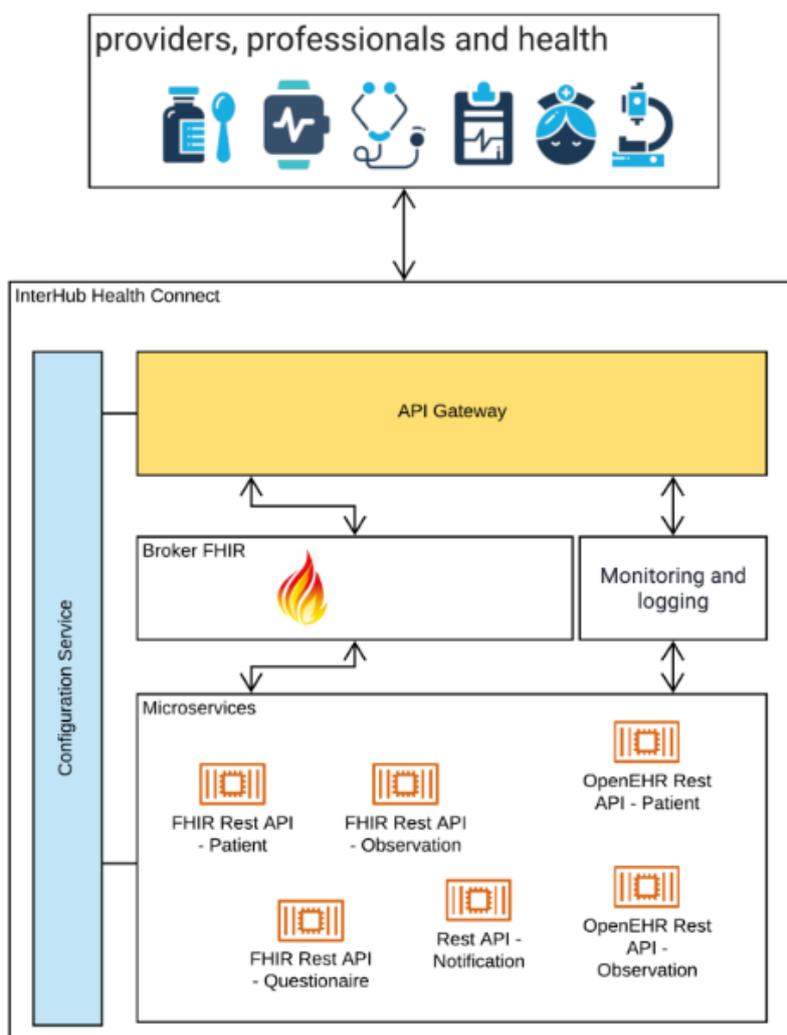
Além disso, os autores discutem desafios e tendências futuras, como a adoção de infraestrutura híbrida, uso de APIs em larga escala para inferência de modelos de IA e a importância de protocolos como o HL7 FHIR para garantir interoperabilidade entre sistemas.

3.2 GIRLS, a Gateway for Interoperability of electronic health Record in Low-cost System

O GIRLS é um portal de baixo custo para interoperabilidade entre registro eletrônicos de saúde utilizando os padrões FHIR e OpenEHR. Ele possui uma estrutura baseada em microserviços projetada para atuar como um hub de integrações entre OpenEHR e FHIR, promovendo a interoperabilidade semântica entre diferentes sistemas que consomem dados com padrões distintos (GOMES et al., 2019). Desta forma, as organizações e profissionais da área da saúde podem acessar e compartilhar informações clínicas de forma contínua e eficiente, especialmente em casos em que os pacientes necessitam de atenção integrada entre sistemas públicos e privados.

A arquitetura utilizada nesse software se baseia em um gateway de APIs que facilita a troca

Figura 3.1: Arquitetura do hub de integrações GIRLS



Fonte: GOMES et al. (2019)

de dados entre padrões distintos, onde é utilizado um broker FHIR que converte as informações de formatos próprios dos sistemas ou baseadas no padrão OpenEHR para o padrão FHIR.

A tecnologia utilizada para a gestão de microserviços é Spring Cloud e para a comunicação entre os componentes do sistema foi utilizado Apache Kafka. A Figura 3.1 representa a arquitetura do sistema e seus componentes, bem como a função de cada um, onde podemos destacar os seguintes componentes:.

- API Gateway: Realiza o gerenciamento dos endpoints para a troca de dados e coordena os microserviços.
- Configuration Service: Facilita a comunicação e gerenciamento de múltiplos microserviços.
- Broker FHIR: Realiza a conversão dos dados dos sistemas proprietários e de padrões distintos para FHIR, realizando todos os mapeamentos necessários.

- **Monitoring and logging:** Monitora e registra o fluxo de dados entre diversos microserviços.
- **Microservices:** Implementações específicas para cada endpoint e funcionalidade do sistema.

3.3 Comparação de Desempenho entre Soluções de Interoperabilidade

O trabalho de ALENCAR et al. (2022) apresenta uma análise comparativa entre as soluções de interoperabilidade governamental X-Road e uma arquitetura moderna baseada no API Gateway Kong. O X-Road foi criado originalmente para ambientes públicos, utilizando a pilha SOAP/WSDL e, posteriormente, aderindo suporte a REST. Apesar dessas evoluções, sua estrutura manteve-se voltada para aplicações monolíticas, impondo limitações em cenários de alta escalabilidade e integrações baseadas em microsserviços.

Por outro lado, o Kong se destaca por sua abordagem orientada a APIs REST, com uma arquitetura flexível e extensível, onde podemos utilizar vários plugins para suportar diferentes protocolos, como GraphQL e SOAP. Além disso, oferece recursos centralizados para autenticação, autorização e monitoramento das requisições trafegadas, o que contribui para uma melhor observabilidade das comunicações entre os serviços.

A análise experimental realizada pelos autores avaliou três cenários de requisições de APIs: chamadas diretas, por meio do X-Road e utilizando o Kong. Os resultados mostraram que a solução baseada no Kong apresentou tempos de resposta inferiores em chamadas sequenciais e concorrentes, enquanto o X-Road demonstrou maior sobrecarga de comunicação, atribuída, em parte, à sua arquitetura legada e aos processos de controle rigoroso de identidade.

Ainda que o Kong tenha apresentado melhor desempenho e maior flexibilidade tecnológica, os autores citam que a solução não contempla nativamente mecanismos mais robustos de controle de identidade, o que pode ser uma limitação em cenários que lidam com informações sensíveis, como os dados de saúde. Nesses casos, seria necessária a adoção de soluções complementares de autenticação e autorização, como o uso de SSO (Single Sign-On) ou protocolos especializados, a exemplo do SMART on FHIR.

3.4 Comparativo entre os trabalhos relacionados

A partir da revisão dos trabalhos relacionados, foram identificados aspectos importantes que contribuem para a definição de soluções de interoperabilidade em ecossistemas que demandam integração de sistemas e segurança de dados, como é o caso da área da saúde. A seguir, são apresentados os principais pontos observados nas soluções analisadas.

- **Ponto Único de Interação:**
 - **IBM API Connect/API Hub:** Centraliza o gerenciamento e a exposição de múlti-

plas APIs, promovendo a interoperabilidade, reuso e descoberta pública de serviços voltados à saúde digital.

- GIRLS: utiliza um gateway centralizado para facilitar a comunicação entre diferentes padrões de interoperabilidade, como FHIR e OpenEHR, atuando como um ponto único de acesso para os serviços disponibilizados.
- X-Road/Kong: Ambas as soluções centralizam o acesso a serviços, com o X-Road voltado a ambientes institucionais e governamentais e o Kong oferecendo uma abordagem mais flexível e orientada a APIs REST para diferentes domínios de aplicação.
- HealthGate: Assim como as soluções anteriores, centraliza o acesso a serviços, com foco em ambientes da saúde e integração a plataformas que constituem o ecossistema IF4Health, contemplando requisitos específicos de interoperabilidade nesse contexto.
- Segurança e Autenticação:
 - IBM API Connect/API Hub: Adota autenticação via IBMid e API Keys, com controle de acesso, versionamento de endpoints, segurança e integração com ambientes seguros, atendendo aos padrões corporativos da área da saúde.
 - GIRLS: Utiliza mecanismos de segurança em nível de API, mas não incorpora protocolos específicos de autenticação voltados à saúde.
 - X-Road: Apresenta forte controle de identidade e autenticação via certificados digitais, oferecendo alto nível de segurança e rastreabilidade, ainda que com maior complexidade de implantação.
 - Kong: Foca em segurança genérica baseada em tokens JWT, OAuth2 e certificados SSL, sem suporte nativo a padrões específicos para o setor da saúde.
 - HealthGate: Utiliza autenticação baseada em OAuth2 e contempla o uso do SMART on FHIR, proporcionando controle maior de escopos de acesso, em conformidade com regulamentações como a LGPD.
- Coleta e Monitoramento de Logs:
 - IBM API Connect/API Hub: Oferece o monitoramento completo e centralizado via dashboard, com estatísticas de uso, rastreamento de chamadas e análise de performance para cada API publicada, promovendo transparência e auditoria.
 - GIRLS: Dispõe de um módulo de monitoramento e registro de logs.
 - X-Road: Implementa registros detalhados de transações e time-stamping, o que permite alta rastreabilidade.
 - Kong: Permite a coleta de logs por meio de plugins, mas depende de integrações externas para oferecer recursos completos de auditoria.

- HealthGate: Inclui monitoramento de requisições voltado para a auditoria de transações clínicas, permitindo o acompanhamento de acessos a dados sensíveis.

4 METODOLOGIA E SOLUÇÃO PROPOSTA

Este Capítulo descreve o processo de desenvolvimento do HealthGate, destacando os passos realizados para a criação, implementação, tecnologias adotadas, estrutura do sistema, e o funcionamento das rotas disponíveis.

4.1 Definição de tecnologias e estrutura do sistema

Primeiramente, para garantir um desenvolvimento ágil, eficiente e com boa performance, foram escolhidas tecnologias modernas e amplamente utilizadas. No back-end, optou-se pelo Node.js, uma linguagem leve e escalável que utiliza JavaScript, proporcionando um desenvolvimento prático e eficiente. Para a criação das APIs RESTful, foi utilizado o framework Express, que facilita a construção de serviços web com uma arquitetura flexível.

Além disso, o armazenamento de dados foi realizado com o banco de dados MongoDB, que é orientado a documentos, oferecendo escalabilidade e flexibilidade no gerenciamento das informações. Por fim, para o desenvolvimento da interface gráfica, escolheu-se o EJS, um mecanismo de templates que permite combinar HTML com JavaScript de maneira prática.

Durante o desenvolvimento, também foram utilizadas algumas bibliotecas para auxiliar o processo. Para realizar requisições HTTP, empregou-se a biblioteca Axios, que oferece chamadas de APIs simples e eficiente para comunicação com serviços externos.

A estrutura dos diretórios foi organizada em três pastas principais. Abaixo, temos as especificações de cada pasta:

- `/assets`: Contém a arquitetura macro do sistema para melhor entendimento do funcionamento geral;
- `/backend`: Responsável por receber e processar as requisições das APIs, realizando o roteamento para os endpoints corretos com base nos dados armazenados no banco. Também abriga os arquivos de renderização da interface;
- `/db`: Armazena informações essenciais, como as rotas cadastradas e suas configurações, permitindo consultas rápidas e seguras para o roteamento.

4.2 Rotas disponíveis do HealthGate

O HealthGate permite cadastrar, buscar e deletar rotas tanto por requisições web (Seção 4.2.1) quanto por interface gráfica (Seção 4.2.2). O cadastro de novas rotas no sistema é essencial para configurar como o *gateway* irá redirecionar as requisições recebidas para os sistemas em que ele foi integrado.

O sistema oferece suporte a três tipos principais de rotas:

1. Rotas para gerenciamento de usuários administradores do sistema (Seção 4.2.3);

2. Rotas para gerenciamento e criação de integrações e novas rotas com sistemas externos (Seções 4.2.1 e 4.2.2);
3. Rotas criados pelos próprios usuários, permitindo o roteamento entre diferentes serviços (Seção 4.2.4).

4.2.1 Gestão e Cadastro de rotas via Requisição Web

Para configurar o HealthGate com as rotas das APIs dos sistemas, as seguintes rotas específicas são dedicadas à gestão das rotas, permitindo:

- Listar as rotas já cadastradas (GET - `/api/admin/routes`).
- Criar novas rotas no gateway (POST - `/api/admin/routes`) onde o payload é o JSON do Código-fonte 4.1.
- Deletar rota (POST - `/api/admin/routes/:id`), onde passamos o id da rota cadastrada no banco de dados.

O endpoint POST - `/api/admin/routes` realiza a inserção de novas rotas para serem roteadas pelo HealthGate. Assumindo como exemplo o cadastro de uma das rotas do sistema FASS-ECG (Seção 2.3), no corpo dessa requisição (retratado no Código-fonte 4.1), observa-se diversas chaves necessárias para o cadastro da nova rota:

Código-fonte 4.1: Corpo da requisição do cadastro de rota para consulta de observações em intervalo de tempo

```
{
  "nameProject": "FASS_ECG",
  "sourcePath": "/Observation/:id/data/:minute",
  "routeParams": {
    "0": "id",
    "1": "minute"
  },
  "queryParams": {},
  "targetUrl": "https://biosignalinfhir.if4health.com.br/base_
  ↪ R4/Observation/{:id}/data/{:minute}",
  "method": "GET",
  "headers": {
    "accept": "application/fhir+json",
    "content-type": "application/fhir+json"
  },
  "description": "Permite consultar os dados de uma observao
  ↪ clinica de um paciente especifica ({id}) em um intervalo
  ↪ de tempo especifico ({minute})"
}
```

- `nameProject`: Representa o nome do projeto ao qual a rota pertence. Permite categorizar e agrupar rotas de acordo com o contexto do sistema. Por exemplo: "FASS_ECG" e "IF-Cloud"
- `sourcePath`: Define o caminho de origem da rota. Este campo especifica o padrão de URL que o gateway deve observar para redirecionar as requisições.
- `routeParams`: Um mapeamento contendo os parâmetros dinâmicos de rota extraídos da URL de origem. Esses parâmetros podem ser usados para personalizar o redirecionamento.
- `queryParams`: Um mapeamento contendo parâmetros de query que podem ser utilizados para enriquecer a requisição redirecionada. Exemplo de configuração de `queryParams` para uma determinada URL do sistema H2Cloud:

```

${H2Cloud-URL} \
  auth/register?redirect_uri=/api-docs&scope=patient/*.rs \
  &token_endpoint=auth/token&client_id=12345 \
  &aud=asd&state=asd

"queryParams": {
  "redirect_uri": "/api-docs",
  "scope": "patient/*.rs",
  "token_endpoint": "/auth/token",
  "client_id": "{:id}",
  "aud": "{:aud}",
  "state": "{:state}"
}

```

- `targetUrl`: URL de destino para onde a requisição será encaminhada. Esse campo pode conter *placeholders*, como `{:id}`, que serão substituídos pelos valores extraídos da requisição original.
- `method`: Especifica o método HTTP permitido para essa rota. Valores possíveis: GET, POST, PUT, DELETE, PATCH.
- `headers`: Um mapeamento de cabeçalhos adicionais que devem ser incluídos na requisição redirecionada.
- `description`: Descrição da rota, fornecendo informações adicionais sobre seu propósito e uso.

O processamento do cadastro de rotas pelo HealthGate requer cuidados que estão categorizados em três etapas:

- **Validação de campos obrigatórios**, onde `nameProject`, `sourcePath`, `targetUrl` e `method` são verificados.

- Substituição dinâmica, onde *placeholders* (ex.: `{:id}` e `{:minute}`) são automaticamente substituídos pelos valores correspondentes extraídos de `routeParams`, da `sourcePath` e da `queryParams`.
- Headers dinâmicos, onde *headers* adicionais definidos no cadastro são incluídos na requisição redirecionada, permitindo autenticações específicas ou outros ajustes. O exemplo mostra o cadastro de uma rota no padrão FHIR (Seção 2.1).

Com essa estrutura, o cadastro de novas rotas garante a flexibilidade e a escalabilidade do gateway, permitindo que ele se adapte a diferentes cenários e requisitos de integração.

4.2.2 Gestão e Cadastro de rotas via Interface Gráfica

Alternativamente, o cadastro de novas rotas também pode ser realizado por interface gráfica. Na Figura 4.1 podemos observar a tela de cadastro de novas rotas, onde os campos do formulário são similares ao corpo da requisição de cadastro de rotas via Requisição Web, situado no código-fonte 4.1.

Figura 4.1: Tela cadastro de rotas HealthGate

The screenshot shows the 'Cadastrar Nova Rota' form in the HealthGate interface. The form is structured as follows:

- Nome do Projeto:** A dropdown menu with 'FASS_ECG' selected.
- Source Path:** A text input field.
- Route Params (JSON):** A text input field containing `{ "0": "id", "1": "minute" }`. Below it is an example: `Exemplo: { "0": "id", "1": "minute" }`.
- Query Params (JSON):** A text input field containing `{}`. Below it is an example: `Exemplo: { "key": "value" }`.
- Headers (JSON):** A text input field containing `{ "accept": "application/json", "Authorization": "Bearer <token>" }`. Below it is an example: `Exemplo: { "accept": "application/json", "Authorization": "Bearer " }`.
- Target URL:** A text input field.
- Método:** A dropdown menu with 'GET' selected.
- Descrição:** A text input field.
- Cadastrar:** A green button at the bottom.

4.2.2.1 Outras funcionalidades da Interface Gráfica

A interface gráfica também permite gerenciar as rotas do HealthGate, pois além do cadastro de novas rotas, ela fornece uma visão de todas as rotas cadastradas, opção de deletar rota, conforme demonstrado na Figura 4.2 e acompanhar, por meio de um painel, o tráfego de requisições que passam pelo sistema. Esse painel contribui para a observabilidade e o moni-

toramento do funcionamento do gateway. Todas essas funcionalidades são possíveis graças às informações armazenadas no banco de dados MongoDB do HealthGate, que inclui dados dos usuários administradores, rotas cadastradas e logs das requisições trafegadas.

Figura 4.2: Tela de rotas cadastradas

Projeto	Source Path	Target URL	Método	Descrição	Ações
FASS_ECG	/Observation/id	https://if4health.charqueadas.ifsul.edu.br/biosignalinfrir/Observation/id	GET	Recupera informações completas de uma observação clínica específica usando o ID do recurso (id).	Excluir
FASS_ECG	/Observation	https://if4health.charqueadas.ifsul.edu.br/biosignalinfrir/Observation/	POST	Cria uma nova observação clínica no servidor FHIR.	Excluir

4.2.3 Rotas para usuário

Para que o sistema seja utilizado de forma segura e controlada, é necessário que os usuários se autenticem antes de acessar as funcionalidades oferecidas. Por isso, o sistema disponibiliza rotas voltadas a gestão de usuários, que possibilitam o registro de novos administradores e o login de usuários já cadastrados.

- O acesso e utilização das rotas via requisição web utiliza o protocolo *Basic Auth*, exclusivo para usuários previamente cadastrados;
- Para gerenciar o sistema por meio da interface gráfica, é utilizado *Token JWT*, com sessão válida por uma hora.

Os *endpoints* utilizados para realizar o login e o registro de usuário são, respectivamente:

1. POST - /api/auth/login
2. POST - /api/auth/register

onde o corpo de requisição de ambos *endpoints* requer nome de usuário e senha (Código-fonte 4.2).

Código-fonte 4.2: Exemplo de *payload* das requisições de registro e login

```
{
  "username": "admin3",
  "password": "admin123"
}
```

4.2.4 Rotas criadas por usuários e roteamento de requisições

As rotas criadas por usuários seguem três caminhos principais, que definem o destino das requisições:

- FASS-ECG: endpoint `/api/fassecg/`
- IF-Cloud: endpoint `/api/ifcloud/`
- H2Cloud: endpoint `/api/neoFassEcg/`

Essas rotas são processadas pelo HealthGate, que realiza um mapeamento inteligente com base nas URLs cadastradas. Para identificar se a URL contém um parâmetro ou query, o HealthGate utiliza uma lógica com o auxílio de expressões regulares (*regex*), que permite encontrar e contabilizar os parâmetros. Na Seção 4.3, será detalhado o funcionamento do *regex* e como ele foi implementado nesta aplicação.

Quando uma rota correspondente é encontrada, os parâmetros da URL chamada são extraídos e mapeados. Esses parâmetros são então adicionados ao corpo da requisição, que será enviada ao sistema de destino.

No caso das requisições destinadas ao sistema FASS-ECG, que é baseado no padrão FHIR, os headers devem ser configurados de maneira distinta. Os valores utilizados são:

- `content-type: 'application/fhir+json'`
- `accept: 'application/fhir+json'`

Esses headers indicam que os dados seguem o padrão FHIR e estão codificados em JSON, reforçando a interoperabilidade e a conformidade com o protocolo utilizado pelo sistema.

No caso do NeoFASS-ECG, trata-se da evolução dos sistemas FASS-ECG e H2Cloud, agora unificados para melhor integração. Ainda que sirvam a propósitos distintos, a fusão dos sistemas foi implementada gradualmente no ecossistema. Essa rota destinada ao H2Cloud desempenha um papel estratégico dentro do HealthGate, uma vez que, em determinados fluxos, as requisições encaminhadas ao FASS-ECG devem passar previamente por uma etapa de autenticação SMART on FHIR (Seção 2.2), onde é retornado um token com tempo de expiração para as requisições destinadas ao FASS-ECG poderem ser realizadas.

Para lidar com essas situações, o HealthGate implementa um processo de obtenção e cache do token de acesso, evitando chamadas repetidas ao serviço de autenticação e otimizando o desempenho do sistema. O token é armazenado temporariamente na aplicação e reaproveitado enquanto estiver válido, sendo renovado automaticamente cinco segundos antes de expirar.

Dessa forma, o roteamento das requisições no HealthGate não apenas direciona chamadas para os serviços apropriados, como também atua como agente de autenticação e padronização, garantindo que os dados estejam formatados corretamente, com os headers exigidos e os parâmetros devidamente resolvidos.

4.3 Expressão Regular - Regex

As expressões regulares (*regex*) são amplamente utilizadas no desenvolvimento de software para identificar, extrair e manipular padrões em textos. No contexto do sistema HealthGate, elas são fundamentais para implementar uma lógica flexível de roteamento, capaz de corresponder dinamicamente URLs recebidas com as rotas cadastradas no banco de dados.

O sistema utiliza a biblioteca *path-to-regexp*, uma ferramenta que transforma caminhos de rotas com parâmetros dinâmicos em expressões regulares reais. Esses parâmetros são marcados no padrão `{:param}`, no banco de dados são como `/api/fassecg/Observation/{:id}` para ser realizado o processamento da requisição.

Funcionamento da lógica de roteamento

O processo executado pelo sistema pode ser dividido em etapas:

1. Busca de rotas compatíveis com o método HTTP: o sistema consulta o banco de dados buscando apenas as rotas que possuam o mesmo método da requisição (GET, POST, etc.) e que pertençam ao projeto correspondente (`nameProject`).
2. Ordenação por especificidade: as rotas são ordenadas com base no número de parâmetros presentes em seus caminhos (`{:param}`), da mais específica (menos parâmetros) para a mais genérica (mais parâmetros). Também utilizamos a função *localeCompare* para a ordenação alfabéticas das rotas.
3. Identificação da rota correspondente: cada `sourcePath` da rota é transformado em um *regex* e comparado com a URL da requisição recebida, utilizando a função *match()* da biblioteca *path-to-regexp*. Caso haja correspondência, os parâmetros dinâmicos são extraídos automaticamente e armazenados.
4. Substituição dinâmica de parâmetros: os valores extraídos são utilizados para substituir os *placeholders* na `targetUrl`, resultando na URL final de redirecionamento.
5. Montagem da requisição redirecionada: os *headers* são montados conforme o projeto. Os parâmetros de *query* são preservados e adicionados à URL final. A requisição é então enviada para o sistema de destino, com os dados e *headers* apropriados.

Exemplo prático

Dada a rota cadastrada com os seguintes valores:

```
sourcePath: /Observation/:id
targetUrl: $FHIR-API-URL/Observation/:id
```

e uma requisição recebida no HealthGate no seguinte *endpoint*:

- GET - `/api/fassecg/Observation/12345`

o sistema irá:

1. Identificar que a URL corresponde à rota cadastrada;
2. Extrair o parâmetro `id = 12345`

3. Substituir o `:id` por 12345 na `targetUrl`

4. Redirecionar a requisição final para URL abaixo com o método utilizado:

```
$FHIR-API-URL/Observation/12345
```

A combinação da ordenação por especificidade com o uso de expressões regulares permite que o sistema priorize rotas mais específicas, garantindo precisão no roteamento mesmo em cenários com múltiplas rotas semelhantes. Ademais, essa abordagem possibilita a expansão dinâmica do sistema, sem a necessidade de reescrever a lógica de roteamento das requisições. Com isso, o HealthGate alcança um alto nível de flexibilidade, fator essencial para ambientes que demandam múltiplas integrações e interoperabilidade entre diversos sistemas.

5 RESULTADOS E DISCUSSÕES

Esta Seção descreve o entendimento realizado dos sistemas FASS-ECG, IF-Cloud e H2Cloud, bem como a reprodução dos mesmos no ambiente local para, posteriormente, montar e cadastrar os recursos no sistema HealthGate. Em seguida, temos as rotas que foram cadastradas no gateway, além da arquitetura criada no Postman para validação das rotas.

5.1 Reprodução dos sistemas FASS-ECG, IF-Cloud e H2Cloud

Para compreensão e reprodução dos sistemas FASS-ECG, IF-Cloud e H2Cloud, foram configurados no ambiente local, utilizando portas específicas para cada um, evitando o conflito das mesmas. O sistema FASS-ECG foi alocado na porta 3000, o IF-Cloud foi configurado na porta 8000 e o H2Cloud na porta 8080. Durante o processo, foi realizada uma análise detalhada das configurações e recursos existentes nos projetos, a fim de garantir a compatibilidade entre as APIs e a estrutura proposta pelo HealthGate. O sucesso na reprodução dos sistemas possibilitou pleno entendimento das funcionalidades e recursos de cada um. As Figuras 5.1 e 5.2 ilustram os sistemas FASS-ECG, IF-Cloud e H2Cloud operando localmente. É importante reforçar que a versão do H2Cloud utilizada é a versão mesclada com o FASS-ECG, denominada ‘neoFASS-ECG’¹, pois esta é a versão mais atual do repositório IF4Health que implementa o SMART on FHIR.

A reprodução do ecossistema de aplicações é colocado como um resultado preliminar, porque a premissa para demonstrar o funcionamento do HealthGate é ter os outros serviços rodando online. Ou seja, antes de implementar HealthGate, é necessário estudar e saber implantar os demais serviços do ecossistema.

5.2 Prova de Conceito

Uma vez que as 3 aplicações do ecossistema estão online, foi realizado o desenvolvimento do HealthGate, bem como a implementação de todas as rotas existentes nos três sistemas FASS-ECG, IF-Cloud e H2Cloud. O desenvolvimento está disponível no GitHub ², onde temos o README com todo passo a passo detalhado para clonar e subir o serviço localmente.

O cadastro e a organização das rotas foram realizados no ambiente do Postman, permitindo uma gestão centralizada das configurações. Abaixo, temos o detalhamento das rotas cadastradas e a forma de organização para garantir um bom entendimento e agilidade durante o processo.

A estrutura de rotas foi organizada em cinco pastas principais, conforme ilustrado na Figura 5.3:

¹<https://github.com/if4health/neoFASS-Ecg>

²<https://github.com/if4health/healthgate>

Figura 5.1: Sistemas FASS-ECG e IF-Cloud reproduzidos localmente

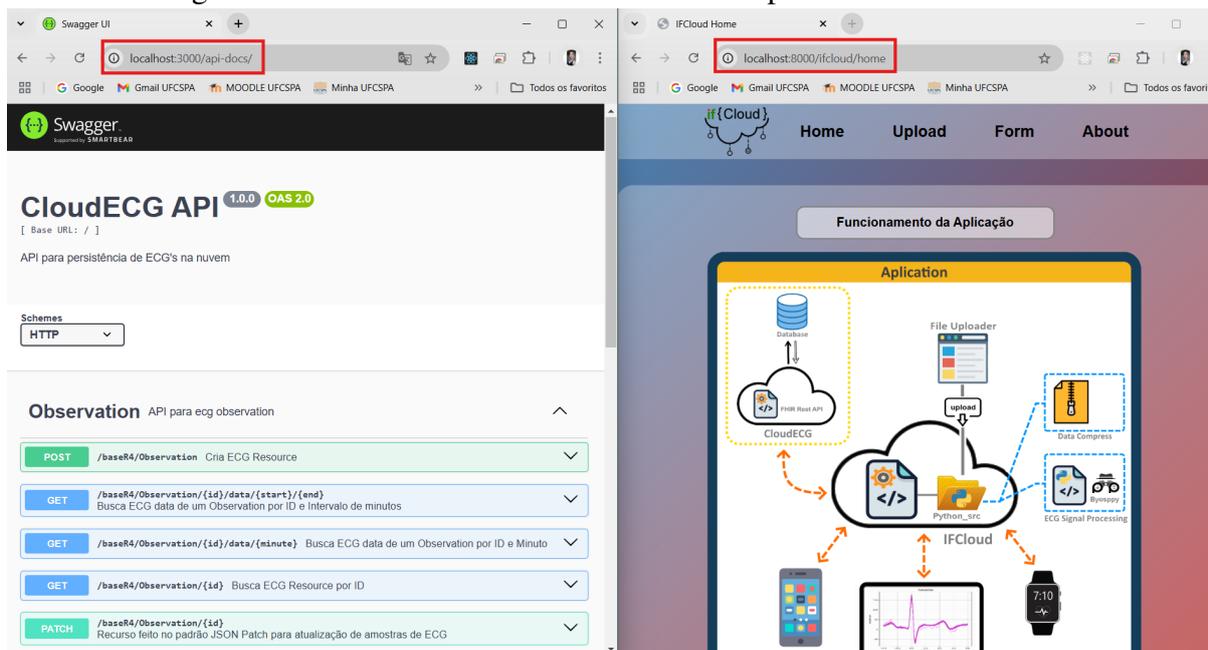
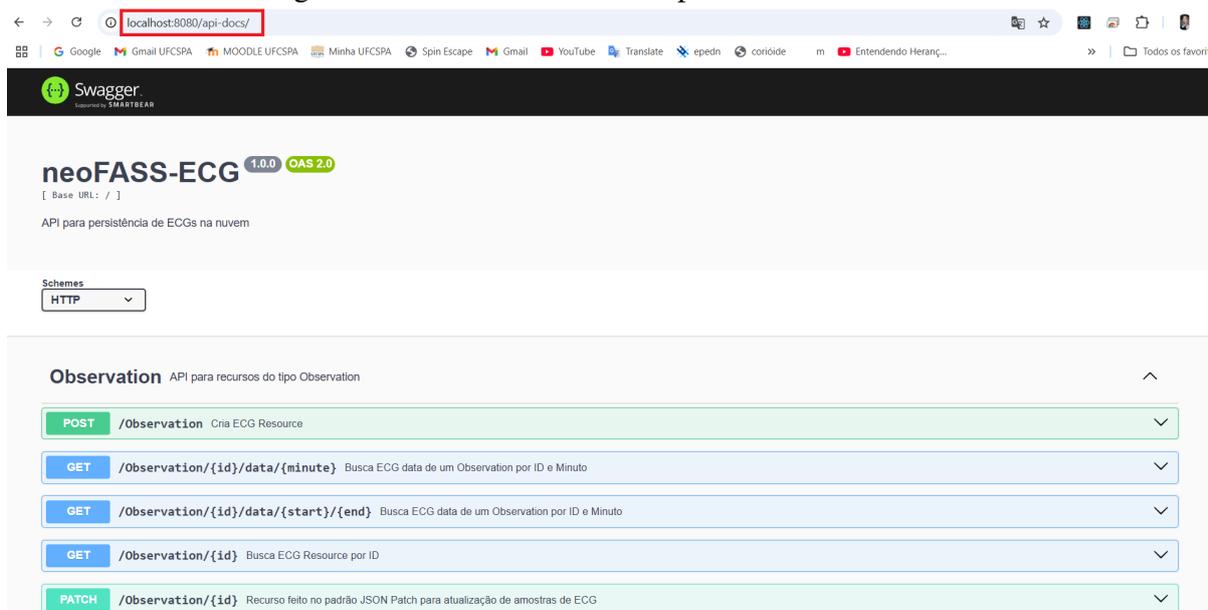
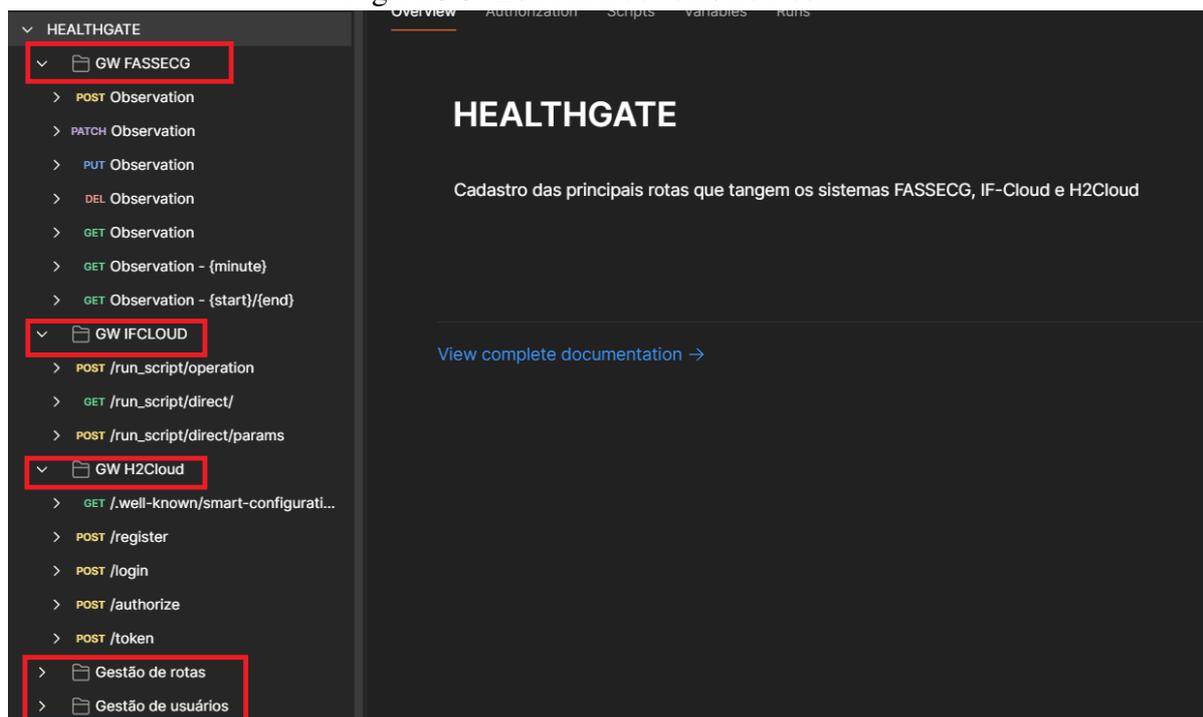


Figura 5.2: Sistema H2Cloud reproduzido localmente



- GW FASSECG: Contém as rotas relacionadas ao sistema FASS-ECG.
- GW IFCLOUD: Inclui as rotas do sistema IF-Cloud, com foco na execução de scripts e operações específicas.
- GW H2Cloud: Compreende as rotas do sistema H2Cloud, com foco no registro e autenticação de usuários.
- Gestão de rotas: Pasta destinada à gestão centralizada das rotas do HealthGate, permitindo ações como listagem e criação de rotas (detalhadas no Código-fonte 4.1).
- Gestão de usuários: Inclui os *endpoints* específicos para o gerenciamento de usuários administradores do sistema.

Figura 5.3: Estrutura de rotas no Postman



Todas as *collections* compatíveis para importação no Postman apresentadas na Figura 5.3 estão disponíveis no repositório GitHub.

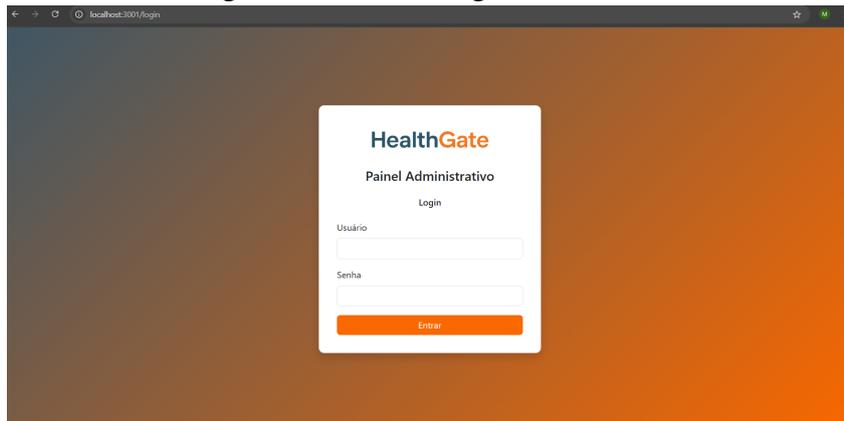
5.3 Interface Gráfica

Com o objetivo de facilitar a interação dos usuários com o sistema HealthGate e proporcionar maior controle e visibilidade, foi desenvolvida uma interface gráfica acessível via web. Essa interface foi projetada com foco na usabilidade e organização dos principais recursos oferecidos pelo gateway, centralizando as operações administrativas em um ambiente intuitivo e funcional.

A aplicação conta com uma tela de login, ilustrado na Figura 5.4, garantindo que apenas usuários previamente cadastrados possam acessar os recursos administrativos. Após o login, o usuário tem acesso a uma visão geral das rotas cadastradas (Figura 4.2), organizadas por projeto (FASS-ECG, IF-Cloud e H2Cloud), com possibilidade de cadastrar novas rotas diretamente pela interface (Figura 4.1), sem a necessidade de manipulação manual via ferramentas como Postman ou bancos de dados.

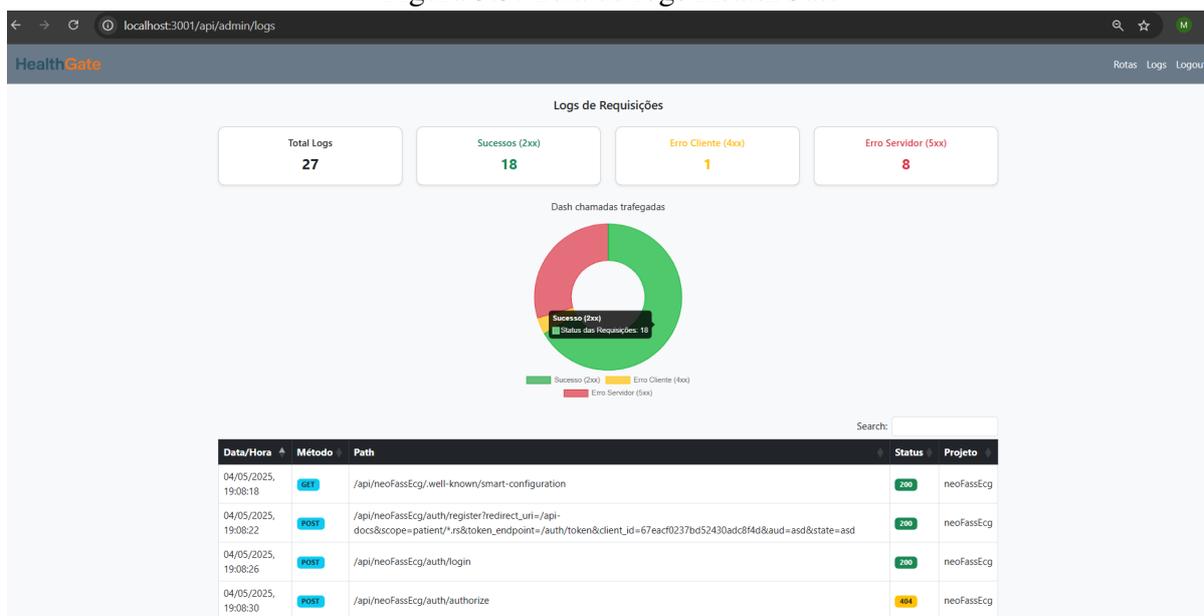
Outro destaque importante a ser feito é a tela de logs de requisições, situada na Figura 5.5, que não apenas lista os acessos trafegados pelo gateway em tempo real, mas também apresenta gráficos visuais e indicadores quantitativos, como o número total de requisições, taxa de sucesso e ocorrência de erros por tipo de status HTTP (2xx, 4xx, 5xx). Esse painel contribui significativamente para a observabilidade do sistema, oferecendo ao usuário uma visão clara do comportamento das requisições e facilitando a identificação de gargalos, falhas e padrões

Figura 5.4: Tela de login HealthGate



de uso. A interface gráfica facilita muito a manutenção e a observabilidade do ecossistema como um todo comparado a abordagem por requisições web porque promove uma camada de gerenciamento centralizado e amigável ao usuário.

Figura 5.5: Tela de logs HealthGate



5.4 Apontamento do Front-end Visualizador ao HealthGate

Um visualizador de ECG básico (Seção 2.5) mostra a onda ECG e a frequência cardíaca e essas informações estão disponíveis em duas APIs diferentes (FASS-ECG e IF-Cloud). Para demonstrar que o Front-end não precisa mais atuar como roteador entre os sistemas, foi realizado um experimento de redirecionamento das requisições do Visualizador de ECG (Figura 2.1) para o HealthGate.

Por se tratar de um projeto relativamente antigo, surgiram alguns desafios relacionados à obsolescência de bibliotecas e da própria versão do Node.js utilizada. Após a atualização dessas

dependências, foi realizada uma análise da arquitetura e do código-fonte do sistema, a fim de identificar os pontos necessários para a alteração das URLs.

A Figura 5.6 mostra os trechos de código que realizam fetch nas APIs FASS-ECG e IF-Cloud, respectivamente. O código original está comentado para possibilitar a comparação clara entre o código anterior e o código atualizado, destacando as alterações implementadas. O código-fonte original, de fato, fazia fetch em duas URLs diferentes:

```
biosignalinfhir.if4heath.com.br e ifcloud.if4heath.com.br
```

Esse cenário é especialmente ruim porque as URLs não existem mais e deveriam ser duas variáveis de ambiente. Assim, a mudança foi substituir as URLs por uma única variável de ambiente (BASE_URL) para permitir um roteamento unificado e centralizado. Embora a alteração seja simples em termos técnicos, ela representa um avanço significativo no que diz respeito à manutenção do código, pois sempre que as APIs tiverem suas URLs alteradas, o código-fonte do Visualizador não precisa ser alterado.

Figura 5.6: Otimização do código-fonte do Visualizador de ECG (DUARTE, 2024) para evitar roteamento do front-end

```

JS service.js
C: > TCC1 > ecgmonitor > src > services > JS service.js > getBPMByResu
1
2 import axios from 'axios';
3 //url base utilizada para as requisições ao
  HealthGate em Produção
4 const BASE_URL = "https://if4health.charqueadas.
  ifsul.edu.br/healthgate/api/";
5 //url base utilizada para as requisições ao
  HealthGate em Desenvolvimento
6 // const BASE_URL = "http://localhost:3001/
  healthgate/api/";
7
8 const ApiClient = axios.create({
9   baseURL: `${BASE_URL}fassecg/`
10  //como estava anteriormente
11  //baseURL: "https://biosignalinfhir.
  if4health.com.br/baseR4/"
12 })
13

```

(a) Fetch em FASS-ECG

```

49 const getBPMByResult = async (params) => {
50   //nova implementação do BASEURL
51   return await axios.post(`${BASE_URL}ifcloud/
  run_script/direct/params`, {
52     //como estava anteriormente
53     // return await axios.post("https://ifcloud.
  if4health.com.br/run_script/direct/params", {
54       "scriptName": "calcBPM.py",
55       "params": [params]
56     }).then(res => res);
57 }
58
59 const getRPEAKSByResult = async (params) => {
60   //nova implementação do BASEURL
61   return await axios.post(`${BASE_URL}ifcloud/
  run_script/direct/params`, {
62     //como estava anteriormente
63     // return await axios.post("https://ifcloud.
  if4health.com.br/run_script/direct/params", {
64       "scriptName": "calcRPEAKS.py",
65       "params": [params]
66     }).then(res => res);
67 }
68

```

(b) Fetch em IF-Cloud

Fonte: Autoria própria, 2025

Com as mudanças devidamente aplicadas, foi possível validar o funcionamento do Front-end localmente acessando o endereço `http://localhost:8080`. A partir das alterações, o sistema passou a encaminhar as requisições para um ponto único, o HealthGate, centralizando as chamadas e mantendo as funcionalidades de visualização e cálculo de ECG e BPM.

6 CONCLUSÕES E TRABALHOS FUTUROS

O desenvolvimento do HealthGate representou um avanço significativo na busca por maior integração, interoperabilidade e observabilidade no ecossistema de saúde digital proposto pelo grupo de pesquisa IF4Health. A criação de um gateway centralizado, capaz de intermediar as requisições entre sistemas distintos baseados no padrão FHIR, mostrou-se eficaz do ponto de vista técnico e funcional. O sistema foi homologado em ambiente local e demonstrou capacidade de rotear as chamadas entre as aplicações FASS-ECG, IF-Cloud e H2Cloud, assegurando corretamente a autenticação, padronização e entrega dos dados médicos às APIs destinos.

Além disso, a implementação de uma interface gráfica amigável e funcional para usuários elevou o nível de acessibilidade e administração do sistema. A visualização em tempo real dos logs de requisição junto aos dashboards informativos, reforça a importância da observabilidade como componente essencial para a manutenção, rastreabilidade e desempenho do sistema. Em um contexto onde o volume de dados médicos tende a crescer rapidamente, esse tipo de recurso torna-se cada vez mais estratégico e utilizável em setores da área da tecnologia.

De um ponto de vista reflexivo, este trabalho também evidenciou um desafio frequente em projetos baseados na interoperabilidade: a padronização formal nem sempre implica numa integração prática de ser realizada. Foi necessário um esforço considerável de adaptação, homologação e entendimento de cada um dos serviços envolvidos no ecossistema IF4Health para que a comunicação fluísse de forma consistente entre eles. Esse fator reforça a necessidade de soluções como o gateway HealthGate, que encapsulam essa complexidade de roteamento de requisições e oferecem uma camada de abstração maior para facilitar a conexão entre sistemas distintos.

Nos trabalhos futuros aponta-se como principais direcionamentos:

- Cadastro dinâmico de sistemas externo: atualmente, o sistema está preparado para lidar com APIs específicas (FASS-ECG, IF-Cloud e H2Cloud). A generalização do cadastro de novos sistemas com base em especificações FHIR padronizadas permitiria maior escalabilidade e adaptação a outros cenários clínicos.
- Plugar o visualizador frontend ao HealthGate: conectar diretamente o visualizador frontend (citado no Capítulo 2) ao HealthGate, de modo que a própria interface se beneficie da camada intermediária de abstração e roteamento, sem a necessidade de gerenciar individualmente as conexões com cada sistema, promovendo maior simplicidade, coesão e manutenção.
- Agregação de resultados entre sistemas distintos: uma possibilidade futura é o aprimoramento da lógica de agregação dos dados. Em vez de exigir múltiplas chamadas diretas a diferentes serviços, o HealthGate pode consolidar as respostas de diferentes sistemas em um único fluxo de resposta unificada para o cliente. Essa funcionalidade melhora a experiência do usuário e aumenta a eficiência reduzindo o número de chamadas de rede necessárias.

- Containerização com Docker e gerenciamento com Portainer: considerando que o HealthGate faz parte de um ecossistema maior utilizado pelo grupo IF4Health e que, atualmente, estão todos hospedados no mesmo servidor, propõe-se a criação de containers separados para cada sistema por meio do Docker. Isso permitiria um melhor isolamento, facilidade de manutenção e replicação de ambientes. Com o uso do Portainer, o gerenciamento dos containers seria facilitado por uma interface gráfica acessível. Além disso, também sugere-se a utilização do Jenkins para configurar uma esteira de integração e entrega contínua (CI/CD), automatizando a publicação de novas versões do projeto e promovendo práticas alinhadas ao DevOps.

REFERÊNCIAS

ALENCAR, J. M. U. et al. Comparação de Desempenho entre Soluções de Interoperabilidade. In: INSIGHT LAB – UNIVERSIDADE FEDERAL DO CEARÁ, 2022. **Anais...** [S.l.: s.n.], 2022. p.1–12.

AYAZ, M. et al. The Fast Health Interoperability Resources (FHIR) standard: systematic literature review of implementations, applications, challenges and opportunities. **JMIR Medical Informatics**, [S.l.], v.9, n.7, p.e21929, 2021.

CDR, S. **HAPI FHIR - The Open Source FHIR API for Java.**, 2019. Acessado em 11/12/2022, Disponível em: <https://hapifhir.io/>.

DUARTE, E. W. **Desenvolvimento front-end para Visualização de Exames de Eletrocardiograma**, 2024. [S.l.]: IFSul câmpus Charqueadas, 2024.

GOMES, F. et al. GIRLS, a gateway for interoperability of electronic health record in low-cost system: interoperability between fhir and openehr standards. In: IEEE INTERNATIONAL CONFERENCE ON E-HEALTH NETWORKING, APPLICATION & SERVICES (HEALTHCOM), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1–6.

HARRIS, C. **Microserviços vs. arquitetura monolítica**, 2024. Acessado em 05/11/2024, Disponível em: <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>.

HAT, R. **O que faz um gateway de API?**, 2019. Acessado em 10/12/2024, Disponível em: <https://www.redhat.com/pt-br/topics/api/what-does-an-api-gateway-do>.

HL7. **FHIR Release 4**, 2019. Acessado em 11/12/2018, Disponível em: <https://www.hl7.org/fhir/>.

KONG. **Kong Gateway Documentation**, 2025. Acessado em 23/05/2025, Disponível em: <https://docs.konghq.com/gateway/latest/>.

OEMIG, F.; SNELICK, R. Healthcare interoperability standards compliance handbook. **Cham: Springer international publishing AG**, [S.l.], 2016.

PANTSEL. **Konga**: gui for kong admin api, 2024. Acessado em 23/05/2025, Disponível em: <https://github.com/pantssel/konga>.

PEREIRA, C. R. et al. FASS-ECG: a fhir cloud api to enable streaming and storage of continuous 12-leads ecgs. In: IEEE 36TH INTERNATIONAL SYMPOSIUM ON COMPUTER-BASED MEDICAL SYSTEMS (CBMS), 2023., 2023. **Anais...** [S.l.: s.n.], 2023. p.898–903.

SANTOS, L. S. dos et al. Interoperabilidade e Segurança na Implementação de Aplicações Web de Saúde com SMART on FHIR. **Journal of Health Informatics**, [S.l.], v.15, n.Especial, 2023.

SIU, V. S. et al. API Management in Digital Health: exploring ibm api connect and ibm api hub in enabling healthcare innovation. In: IEEE INTERNATIONAL CONFERENCE ON DIGITAL HEALTH (ICDH), 2024., 2024. **Anais...** [S.l.: s.n.], 2024. p.189–195.

VIEIRA, J. M. et al. IF-Cloud: api fhir para integração de projetos de saúde digital. **Journal of Health Informatics**, [S.l.], v.16, n.Especial, 2024.