INSTITUTO FEDERAL DE CIÊNCIA E TECNOLOGIA DE PERNAMBUCO

Campus Recife

Departamento Acadêmico de Cursos Superiores

Tecnologia em Análise e Desenvolvimento de Sistemas

TÁSSIO RICARDO SILVA DE MORAES

DESENVOLVIMENTO DE API REST COM SPRING BOOT PARA INTEGRAÇÃO DE DADOS DE PRODUÇÃO DO HOSPITAL DAS CLÍNICAS DA UNIVERSIDADE FEDERAL DE PERNAMBUCO

Recife

2024

TÁSSIO RICARDO SILVA DE MORAES

DESENVOLVIMENTO DE API REST COM SPRING BOOT PARA INTEGRAÇÃO DE DADOS DE PRODUÇÃO DO HOSPITAL DAS CLÍNICAS DA UNIVERSIDADE FEDERAL DE PERNAMBUCO

Trabalho de conclusão de curso apresentado ao Departamento Acadêmico de Cursos Superiores do Instituto Federal de Ciência e Tecnologia de Pernambuco, como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Ms. Paulo Abadie Guedes

Recife

2024

Ficha elaborada pela bibliotecária Maria do Perpétuo Socorro Cavalcante Fernandes CRB4/1666

M687d

2024 Moares, Tássio Ricardo Silva de

Desenvolvimento de API REST com Spring Boot para integração de dados de Produção do Hospital das Clínicas da Universidade Federal de Pernambuco./ Tássio Ricardo Silva de Moraes. ---- Recife: o autor, 2024.

47. il. Color.

Trabalho de Conclusão (Curso Superior Tecnológico em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Pernambuco, Recife, 2024.

Inclui Referências

Orientador: Prof. Paulo Abadie Guedes

1.Base dados. 2. API. 3. API REST. 4. Dados públicos. 5. Sistemas de informação em saúde. 6. Spring boot. I. Título. II. Guedes, Paulo Abadie (orientador). III. Instituto Federal de Pernambuco.

CDD 005.13 (21 ed.)

Trabalho de Conclusão de Curso apresentado pelo estudante **Tássio Ricardo Silva de Moraes** à coordenação de Análise e Desenvolvimento de Sistemas, do Instituto Federal de Pernambuco, sob o título de "**DESENVOLVIMENTO DE API REST COM SPRING BOOT PARA INTEGRAÇÃO DE DADOS DE PRODUÇÃO DO HOSPITAL DAS CLÍNICAS DA UNIVERSIDADE FEDERAL DE PERNAMBUCO**", orientado pelo **Prof. Ms. Paulo Abadie Guedes** e aprovado pela banca examinadora formada pelos professores:

Recife, 19 de janeiro de 2024.

_____

Prof. Ms. Paulo Abadie Guedes
CTADS/DACS/IFPE

_____

Prof. Dr. Henrique Correia Torres Santos
CTADS/DACS/IFPE

_____

Prof. Dr. Samuel Victor Medeiros Macedo
CTADS/DACT/IFPE

Dedico este trabalho a todos

que acreditaram em mim.

AGRADECIMENTOS

Primeiramente a Deus, pelo Seu amor infinito e ter me presenteado com os amigos e conhecimentos adquiridos ao longo do curso.

Gratidão à minha família por toda compreensão e incentivo demonstrados durante minha jornada nesta instituição. Cheguei até aqui graças ao apoio incondicional de mãe, esposa, tios, tias e pais adotivos que a vida me concedeu. Obrigado por sempre me enxergarem como um ser humano capaz, principalmente nas horas em que o caminho foi mais íngreme.

Agradeço de maneira especial ao Professor Paulo Abadie, que por tantas vezes não permitiu que eu desistisse. Obrigado pela confiança, motivação e ensinamentos tão valiosos.

À minha esposa Isaura e à minha filha Taísa, razões da minha busca constante por evolução. Vocês são dois faróis que iluminam minha vida. Quer chova ou faça Sol, é por vocês que sigo em frente.

RESUMO

Este trabalho de conclusão de curso é baseado num relato de experiência sobre o desenvolvimento de uma API REST com Spring Boot para integrar dados de produção e faturamento ambulatorial e hospitalar do Hospital das Clínicas da Universidade Federal de Pernambuco. As bases de dados foram extraídas a partir de sistemas de informação em saúde disponibilizados pelo SUS e convertidas para o formato JSON, para então serem manipuladas através da API. Para a construção desta ferramenta, foram utilizados Spring Boot 3, Java 17 LTS e banco de dados PostgreSQL 15. Como resultado, obteve-se uma API que atende os requisitos da arquitetura REST, permitindo salvar, consultar, editar e excluir dados de toda a produção SUS do hospital. A ferramenta se mostrou bastante eficiente, podendo ser facilmente escalada para outros estabelecimentos de saúde. A API também pode suprir a ausência deste tipo de serviço na rede SUS, uma vez que os sistemas de informação do governo ainda não fornecem dados de produção e faturamento via API. Tal cenário ratifica o caráter inovador e estratégico do produto desenvolvido no âmbito da gestão pública hospitalar. Por fim, a disponibilização de informações neste formato facilita a integração com ferramentas de análise de dados, como o Microsoft Power BI, que, por sua vez, já é utilizado neste hospital. Dessa forma, cria-se um cenário bastante favorável para embasar tomadas de decisão pelos gestores, pois as informações podem ser acessadas, analisadas e apresentadas de forma ainda mais eficiente.

Palavras-chave: api; api rest; dados públicos; sistemas de informação em saúde; spring boot.

ABSTRACT

This final course project is based on an experience report about the development of a REST API with Spring Boot to integrate outpatient and hospital production and billing data from the Hospital das Clínicas of the Federal University of Pernambuco. The databases were extracted from health information systems provided by SUS and converted to JSON format, to then be manipulated through the API. For the construction of this tool, Spring Boot 3, Java 17 LTS, and PostgreSQL 15 database were used. As a result, an API was obtained that meets the requirements of the REST architecture, allowing to save, consult, edit, and delete data from all SUS production of the hospital. The tool proved to be quite efficient, and can be easily scaled to other health establishments. The API can also supply the absence of this type of service in the SUS network, since the government's information systems still do not provide production and billing data via API. This scenario ratifies the innovative and strategic character of the product developed within the scope of public hospital management. Finally, the provision of information in this format facilitates integration with data analysis tools, such as Microsoft Power BI, which, in turn, is already used in this hospital. In this way, a very favorable scenario is created to support decision-making by managers, as the information can be accessed, analyzed, and presented even more efficiently.

Keywords: api; rest api; public data; health information systems; spring boot.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AOP – Aspect Oriented Programming

API – Interface de Programação de Aplicativo

CID10 – Classificação Internacional de Doenças

CNES – Cadastro Nacional de Estabelecimentos de Saúde

CNS – Cartão Nacional de Saúde

CRUD – Create, Read, Update e Delete

CSV – Comma-separated values

DATASUS – Departamento de Informática do Sistema Único de Saúde

DBC – DataBase Compressed

DBF – Data Base File

EBSERH – Empresa Brasileira de Serviços Hospitalares

HATEOAS – Hypermedia As the Engine Of Application State

HC-UFPE – Hospital das Clínicas da Universidade Federal de Pernambuco

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

JPA – Jakarta Persistence API

JPQL – Jakarta Persistence Query Language

JSON – JavaScript Object Notation

LTS – Long Time Support

MVC – Model, View, Controller

POX – Plain Old XML

REST – Representional State Transfer

SIA/SUS – Sistema de Informações Ambulatoriais do SUS

SIH/SUS – Sistema de Informações Hospitalares do SUS

SIS – Sistemas de Informação em Saúde

SQL – Structured Query Language

SUS – Sistema Único de Saúde

URI – Universal Resource Identifier

XML – Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

O Hospital das Clínicas da Universidade Federal de Pernambuco (HC-UFPE), estabelecimento vinculado à Empresa Brasileira de Serviços Hospitalares (EBSERH), desempenha importante papel assistencial à população, oferecendo diversos serviços habilitados pelo Sistema Único de Saúde (SUS).

O quantitativo de procedimentos realizados pela instituição é informado mensalmente ao Ministério da Saúde, originando um grande volume de dados que, por sua vez, podem ser consultados abertamente por gestores e população através do site do Departamento de Informática do Sistema Único de Saúde (DATASUS).

Com mais de 25 anos de atuação, o DATASUS está presente em todas as regiões do país e segue desenvolvendo soluções e sistemas que auxiliam os diversos órgãos gestores de saúde (DATASUS, 2023).

No âmbito da gestão hospitalar, a análise desse conjunto de dados configura excelente mecanismo de monitoramento do desempenho assistencial, possibilitando ao gestor conhecer mais a fundo o perfil dos serviços, detectar e agir pontualmente nas deficiências.

Nesse contexto, o poder da análise exploratória de dados vinculado a ferramentas de gestão como *dashboards* gera um cenário bastante favorável para impulsionar o processo de tomada de decisão, uma vez que concilia facilidade de uso, interação e visualização clara e objetiva das informações.

Visando oferecer mais praticidade e objetividade aos gestores do HC-UFPE, este artigo tem como principal objetivo a criação de uma API REST que possibilite integração de dados do DATASUS com o Microsoft Power BI.

Uma API REST é uma interface utilizada para troca de informações entre sistemas que segue alguns padrões pré-estabelecidos de implementação (AWS, 2023).

Já o Power BI é uma ferramenta de análise de dados desenvolvida pela Microsoft que permite a criação de *dashboards* dinâmicos a partir das mais diversas fontes de dados, auxiliando seus usuários a tomarem melhores decisões (MICROSOFT, 2023).

Foram utilizados como fontes dos dados o Sistema de Informações Ambulatoriais do SUS (SIA/SUS) e o Sistema de Informações Hospitalares do SUS (SIH/SUS). Ambos os sistemas disponibilizam seus dados para consulta pública através do *site* do DATASUS. Os dados foram armazenados em um banco de dados PostgreSQL 15.

Para a codificação da API, foi utilizada a linguagem Java na versão 17 *Long Time Support* (LTS). Java é uma plataforma de desenvolvimento e linguagem de programação gratuita. Sua popularidade é evidenciada pelas mais de 60 bilhões de máquinas virtuais da plataforma rodando no mundo. Atualmente na versão 21, Java continua concentrando esforços em melhorias de desempenho, estabilidade e segurança de suas aplicações (ORACLE, 2023).

Definidos o contexto e os objetivos, este relato de experiência trará na sequência um referencial teórico sobre APIs REST, sistemas de saúde e *Spring*. O *Spring* é um *framework* para desenvolvimento de software em Java com foco em rapidez, flexibilidade, produtividade, velocidade e segurança. Com o *Spring*, é possível criar desde microsserviços a aplicativos *web* e na nuvem de forma mais simplificada (SPRING, 2024b).

A metodologia utilizada para o desenvolvimento deste artigo está dividida em duas partes: preparação das bases de dados e implementação da API. Por fim, serão apresentados resultados e discussão do trabalho, além das atividades a serem desenvolvidas futuramente.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Interface de Programação de Aplicativo (API)

APIs oferecem uma forma simples de integrar sistemas, fornecendo dados para aplicações na nuvem, *web* e *mobile*, por exemplo.

Dentre suas principais vantagens, podemos citar a simplicidade, a clareza e a facilidade de uso. Um exemplo do poder e versatilidade das APIs se mostra na possibilidade de conexão com várias fontes de dados para assim disponibilizá-los de forma centralizada, viabilizando o acesso por outras aplicações.

Importante destacar que uma API não oferece diretamente uma interface de usuário. Ela atua praticamente na comunicação entre máquinas, retornando dados através de chamadas padronizadas realizadas por outros sistemas (BIEHL, 2016).

2.2 Arquitetura REST

A arquitetura REST (*Representional State Transfer*) foi criada para orientar, padronizar e facilitar o desenvolvimento de aplicações cliente-servidor. Uma de suas principais características é o uso do protocolo HTTP para troca de informações entre sistemas (SUBRAMANIAN; RAJ, 2019).

Uma API REST é considerada RESTful quando atende todas as restrições propostas pela arquitetura. Fielding (2000) estabeleceu seis restrições que devem ser observadas:

1) Cliente/servidor. A troca de mensagens seguindo padrão *request-reponse*, no qual o cliente faz uma requisição e o servidor retorna algum tipo de resposta;

2) *Stateless*. Todas as requisições ao servidor já dispõem de todas as informações necessárias, de forma que o servidor processa as informações e as trata de maneira independente, favorecendo a escalabilidade da aplicação;

3) Capacidade de realizar cache. Resume-se à capacidade de armazenar informações acessadas com frequência, de forma a minimizar a carga no servidor;

4) Interface uniforme. Recursos devem ser bem definidos, manipulados em formatos específicos (JSON, XML, HTML etc.), possuir conteúdo e metadados auto descritivos e disponibilizar links para recursos relacionados;

5) Construção em camadas. Os sistemas são implementados em camadas com diferentes funcionalidades. Camadas podem ser adicionadas, removidas, modificadas ou reordenadas facilmente, de acordo com a evolução da aplicação;

6) Código sob demanda. Permite que o servidor envie código para ser executado na máquina do cliente, dependendo da requisição.

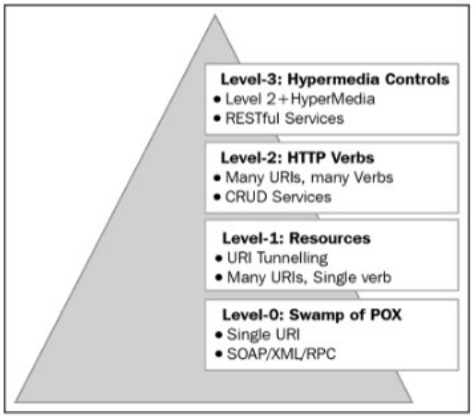
2.2.1. Modelo de maturidade de Richardson

Como afirmado por Brito (2023), existem casos mais simples em que não é viável aplicar todas as regras propostas por Roy Fielding. Nessas situações, a maturidade de uma aplicação REST também pode ser medida através do modelo de Richardson, que classifica sistemas REST em quatro níveis. Cada nível possui as características do anterior e agrega mais funcionalidades. A classificação proposta por Richardson se dá da seguinte forma:

- Nível 0 - *Swamp of POX* (*Plain Old* XML): Utiliza um único *Universal Resource Identifier* (URI) para todas as operações. Uso predominante da requisição POST. Dados transmitidos no formato XML;
- Nível 1 – *Resources*: Aqui, os recursos começam a ser usados, mas de uma forma limitada. Um único recurso é representado por vários URIs. As requisições neste nível ainda são realizadas via POST;
- Nível 2 – Verbos HTTP: Cada recurso passa a ser identificado por uma única URI. Os verbos HTTP são utilizados, permitindo que um único recurso realize mais de uma operação.
- Nível 3 – *Hypermedia Controls*: Inclusão de links na resposta do servidor, permitindo navegação através da API. Apresenta o conceito de HATEOAS (*Hypermedia As the Engine Of Application State*), no qual links são fornecidos ao cliente pelo servidor, de forma que aquele consiga navegar e descobrir recursos que estão disponíveis para serem acessados (WIKIPEDIA, 2023d). Uma aplicação também é considerada RESTful ao atingir este grau de maturidade pois, para chegar neste nível de implementação, é preciso atender todas as restrições da arquitetura REST.

Considerando o contexto da aplicação a ser desenvolvida neste trabalho, constatou-se que o nível dois da escala de Richardson atende perfeitamente os objetivos almejados, passando a ser adotado como referencial de qualidade e maturidade para a ferramenta proposta. A figura 1 exibe de maneira resumida os níveis de maturidade propostos por Richardson:

Figura 1 - Níveis de maturidade de Richardson



Fonte: Subramanian e Raj, 2019.

2.2.2. API REST

Uma das premissas básicas do REST é utilizar o protocolo HTTP para transferência de dados entre sistemas. Através do conceito de recursos, componentes podem ser acessados através dos verbos HTTP: GET, POST, PUT e DELETE (SUBRAMANIAN, 2019), conforme a tabela 1.

Tabela 1 - Verbos HTTP e respectivas funções

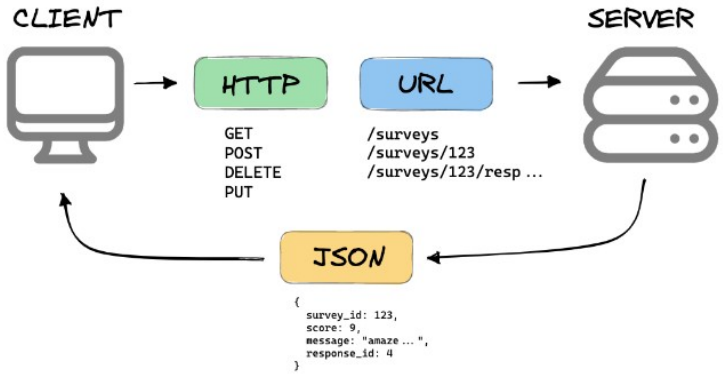
VERBO HTTP	**FUNÇÃO**
GET	Retorna dados de um recurso
POST	Cria um item no recurso
PUT	Altera/substitui um item no recurso
DELETE	Exclui um item existente

Fonte: Próprio Autor, 2023

A figura 2 explica de forma simplificada o funcionamento de uma API REST. Uma requisição é feita ao servidor através de um URI. Na implementação de uma

API, URIs representam recursos e são vinculados a verbos HTTP para requisitar consulta, criação, alteração ou exclusão de itens. Dessa forma, o servidor responde de acordo com o URI informado e pode ou não retornar dados, de acordo com o verbo HTTP ao qual foi vinculado. No caso do exemplo abaixo, o servidor retorna dados no formato JSON, através da requisição GET disparada através da URI “/surveys”.

Figura 2 - Representação de API REST



Fonte: Mann, 2023.

2.3. Spring Framework

O *Spring Framework* é o projeto base para todo o ecossistema *Spring* e oferece um modelo para desenvolvimento de aplicações com foco na implementação das regras de negócio.

Dentre os vários benefícios que apresenta, destaca-se o ganho na produtividade, já que seu uso diminui bastante o tempo gasto com configurações relacionadas ao projeto.

Em sua versão 6.1.2, o *Spring Framework* (figura 3) é composto pelos seguintes módulos:

- *Data Access/Integration*: responsável pela integração com bancos de dados diversos;

- Web: Oferece os recursos necessários para criação de aplicações Web, criação de Web Services REST e implementação do padrão de projeto MVC. O padrão MVC funciona da seguinte forma: o modelo (*Model*) representa a lógica ou objeto do sistema, a visão (*View*) é a camada responsável pela apresentação das informações na tela e o controlador (*Controller*) gerencia a comunicação entre modelo e visão (GAMMA et al. 2008).

- *Aspect Oriented Programming* (AOP): Módulo voltado à programação orientada a aspectos;

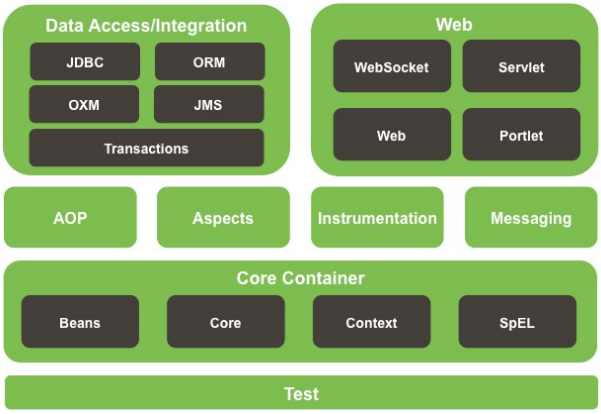
- *Instrumentation*: Oferece recursos para medição de desempenho de partes específicas do código, como tempo de execução de determinado método, por exemplo;

- *Messaging*: Fornece suporte para trabalhar com APIs e protocolos de mensagens;

- *Core Container*: Reúne os módulos fundamentais do *framework* e classes principais, como o *container* de inversão de controle, injeção de dependência (ver seção 2.3.1) e gerenciamento de *Beans*.

- *Test*: Módulo voltado a testes. Faz uso do JUnit, ferramenta de código aberto para testes de software em ambientes Java. Possibilita criar testes unitários, bem como conjuntos de testes (JUNIT, 2023). Testes unitários avaliam de forma isolada os menores componentes de um sistema.

Figura 3 – Arquitetura do Spring Framework



Fonte: Spring, 2023.

2.3.1. Inversão de Controle e Injeção de Dependência

Inversão de controle é um padrão de projeto no qual o fluxo de controle principal da aplicação passa a ser gerido por um componente externo, que pode ser um *framework* ou uma biblioteca.

No caso do *Spring Framework*, o *container* de inversão de controle assume o fluxo principal e a implementação das dependências que serão utilizadas pelos objetos da aplicação. Em outras palavras, o *container* fica responsável por configurar e conectar objetos da aplicação e *framework*, assim como gerenciar seus ciclos de vida (JOHNSON et al. 2005).

Já a injeção de dependência especifica onde serão instanciadas/injetadas as classes que possuem as dependências desejadas. Por exemplo, se uma classe A precisa acessar um método de uma classe B, basta criar um ponto de injeção de B em A, seja através do construtor ou declaração de objeto. Dessa forma, o contêiner de inversão de controle do *Spring Framework* fica responsável por instanciar o objeto da classe B e fornecer suas dependências para A, sempre que necessário (BRITO, 2023).

Inversão de controle e injeção de dependência são recursos presentes no *Core Container* do *Spring Framework*.

2.3.2. Spring Boot

Spring Boot é uma ferramenta baseada no *Spring Framework* que facilita ainda mais o desenvolvimento de aplicações, por gerenciar de forma ágil toda a parte de dependências e configuração do ambiente de desenvolvimento. O *Spring Boot* possui três componentes principais (GEEKHUNTER, 2022):

- *Spring Boot Starter*: agrega várias dependências em apenas uma. A dependência *spring-boot-starter-web*, por exemplo, agrupa dependências relacionadas a configuração de servidor, banco de dados e web. Dessa forma, os *starters* simplificam bastante o gerenciamento de dependências.

- *Spring Boot AutoConfigurator*: Gerencia as configurações padrão de uma aplicação Spring Boot e permite customizá-las através de anotações. Por exemplo, “@SpringBootAplication”, que fica logo acima do método de inicialização da aplicação, é um caso de uso do *AutoConfigurator*.
- *Spring Boot Actuator*: permite gerenciar e monitorar aplicativos em tempo real.

2.4. Sistemas de Informação em Saúde

Sistemas de Informação em Saúde (SIS) são plataformas que coletam e armazenam dados dos serviços de saúde. A partir deles, podem ser obtidas informações relevantes para ajudar os gestores a tomarem decisões, impactando de forma direta na qualidade dos serviços prestados à sociedade.

O DATASUS disponibiliza todos os SIS em uso no Brasil, bem como manuais e programas auxiliares para download. Informações sobre indicadores, dados financeiros, assistenciais e epidemiológicos, demográficos e socioeconômicos podem ser acessadas livremente, tanto pela população como pelos profissionais de saúde (FRANCO, 2023).

O Cadastro Nacional de Estabelecimentos de Saúde (CNES) tem como objetivos o gerenciamento eficaz do SUS, automação da coleta de dados dos estabelecimentos de saúde para fins de planejamento e prestação de contas à sociedade. Também visa atuar junto ao Cartão Nacional de Saúde (CNS), como elo entre os sistemas que compõem o SUS (IBGE, 2023a).

O Tabwin é um SIS auxiliar disponibilizado pelo DATASUS. Ele permite que o usuário compile e baixe bases de dados de diversas fontes gerenciadas pelo SUS (SINAN, 2023).

Para a manipulação dos dados que foram baixados, existe a versão *desktop* do Tabwin, cujo *download* pode ser feito através da página do DATASUS. O programa permite a realização de operações aritméticas, estatísticas e a criação de diversos tipos de gráficos a partir das bases obtidas. Além disso, o programa facilita

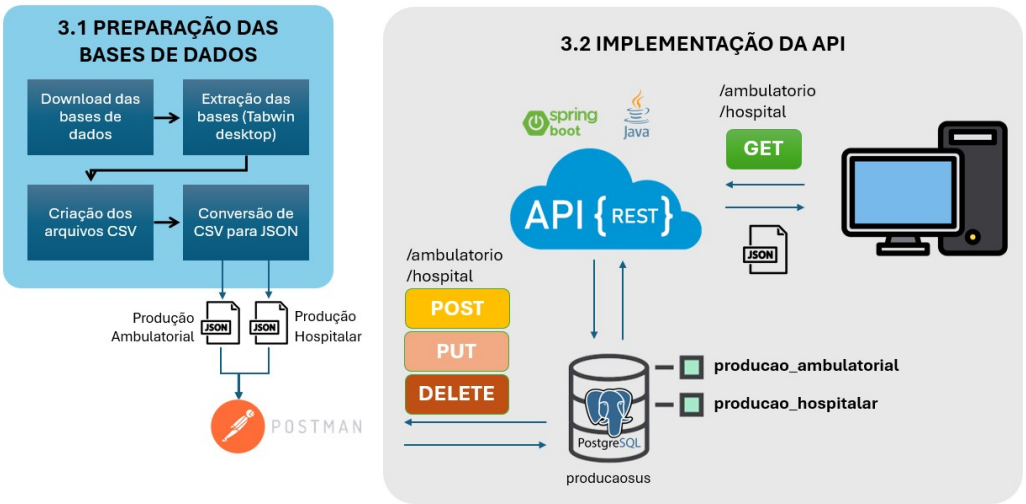
a construção de indicadores, o planejamento, a tomada de decisões com base na alocação de recursos e avaliação do impacto destas nos serviços de saúde.

Os dois principais sistemas utilizados neste relato foram o Sistema de Informações Ambulatoriais do SUS (SIA/SUS) e o Sistema de Informações Hospitalares do SUS (SIH/SUS). O primeiro reúne informações sobre atendimentos realizados em ambulatórios (IBGE, 2023b). O segundo armazena dados de procedimentos executados em caráter de internação (IBGE, 2023c). Além disso, ambos os sistemas geram relatórios que viabilizam o pagamento dos serviços prestados.

3 METODOLOGIA

A figura 4 oferece uma visão geral das duas fases que irão compor esta seção, esquematizando o processo de preparação das informações e a estrutura da API.

Figura 4 – Preparação dos dados e estrutura da API



Fonte: Próprio Autor, 2023

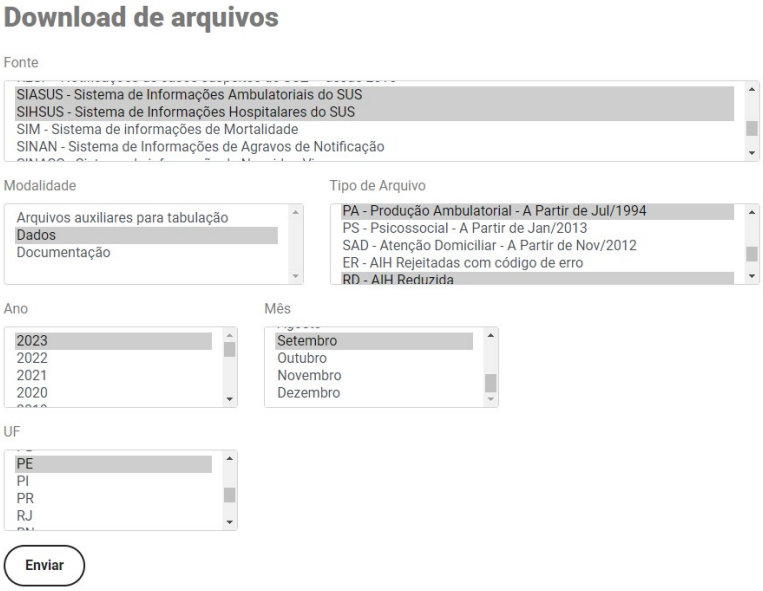
3.1. Preparação das bases de dados

Esta etapa tem início com o download das bases de dados através do serviço Tabwin, no site do DATASUS.

As opções oferecidas para filtragem dos dados englobam Fonte (bases disponibilizadas pelo serviço), Modalidade, Tipo de Arquivo (*datasets* específicos de acordo com a fonte selecionada), Ano, Mês e Unidade Federativa.

Dessa forma, foram realizadas as seguintes seleções, mostradas na figura 5:

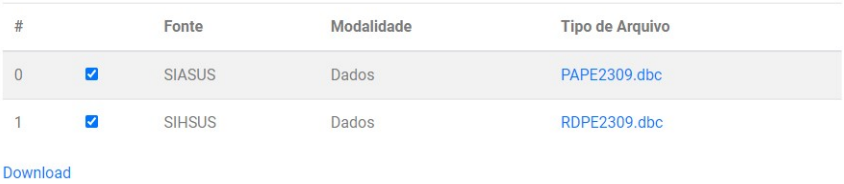
Figura 5 - Seleção de dados para download



Fonte: Próprio Autor, 2023.

Ao clicar no botão Enviar, a solicitação é recebida pelo servidor que retorna uma lista com as bases organizadas por fonte, modalidade e tipo de arquivo. Este último possui formato DBC (*DataBase Compressed*) e denominação composta pela abreviação do tipo de arquivo (PA e RD, respectivamente), UF, ano e mês selecionados, como demonstrado na figura 6:

Figura 6 – Lista de dados compilados para download



Fonte: Próprio Autor, 2023.

Em seguida, para descarregar as informações, é necessário clicar na opção de *download* que aparece logo após a lista mostrada na figura 5.

O site então comprime os arquivos gerados e os disponibiliza através do *link* destacado em azul, como mostra a figura 7:

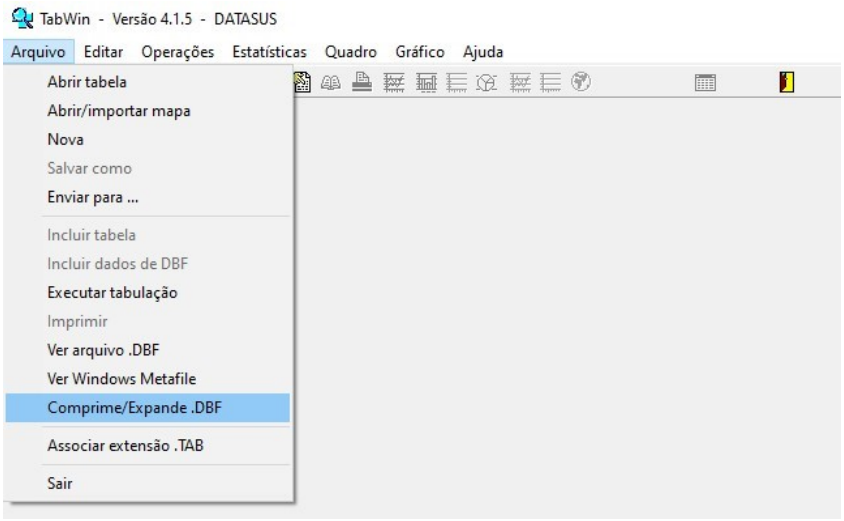
Figura 7 - Link para download dos arquivos em formato .zip

Fonte: Próprio Autor, 2023.

Assim, um único arquivo compactado contendo as bases é disponibilizado ao usuário para ser baixado.

O próximo passo da etapa de preparação dos dados consiste na descompressão das bases e criação dos arquivos CSV. Para isso, foi utilizada a versão 4.1.5 do Tabwin Desktop para conversão dos arquivos DBC para o formato DBF (*Data Base File*), permitindo assim a manipulação dos dados pela ferramenta. A opção para iniciar o processo de expansão das bases está disponível através do menu Arquivo > Comprime/Expande .DBF, conforme mostrado na figura 8:

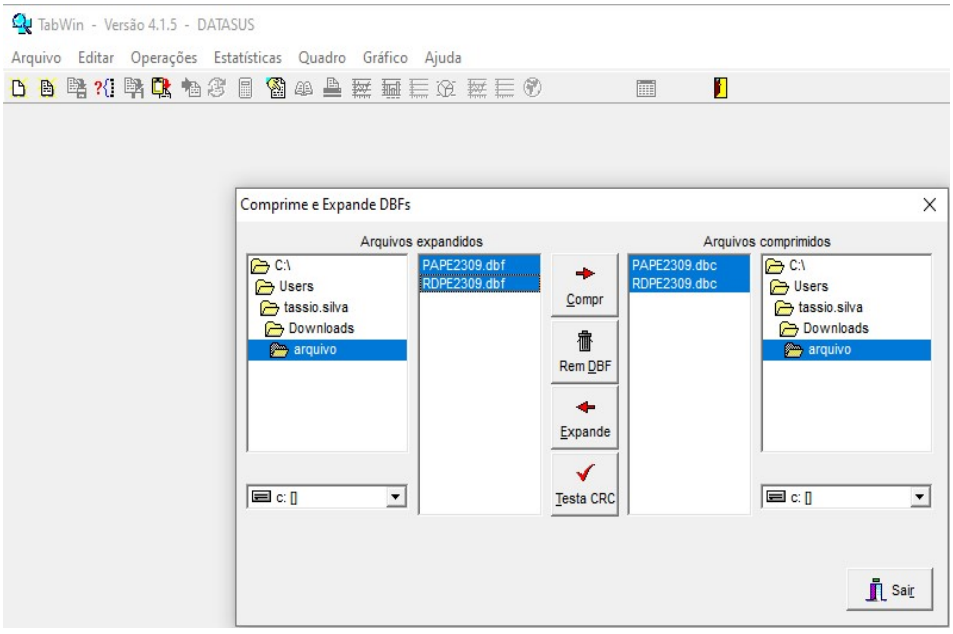
Figura 8 – Comprimir/Expandir DBF



Fonte: Próprio Autor, 2023.

Em seguida, o programa exibe uma janela para seleção dos arquivos comprimidos (DBC) e escolha do local onde ficarão as bases expandidas (DBF) que serão lidas pelo Tabwin Desktop. Selecionados os arquivos DBC, a descompressão é iniciada através da opção “Expande”, gerando finalmente os arquivos DBF em local previamente definido, como mostrado na figura 9:

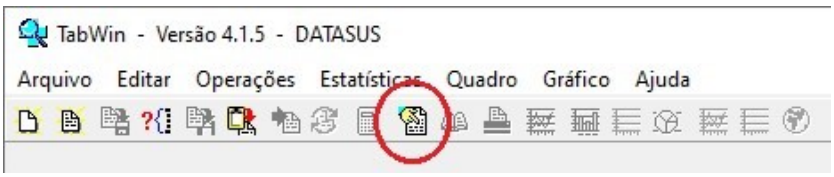
Figura 9 – Expansão de arquivos DBC



Fonte: Próprio Autor, 2023.

Uma vez expandidos, o próximo passo é visualizar os dados através da opção “Mostrar DBF”, como indicado na figura 10:

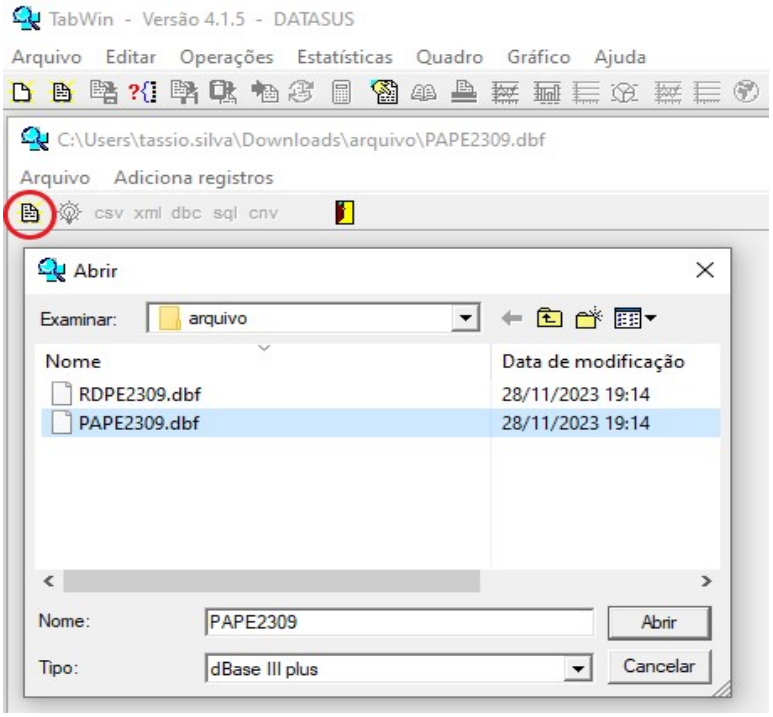
Figura 10 - Mostrar DBF



Fonte: Próprio Autor, 2023.

Selecionada a opção, é necessário abrir um arquivo DBF por vez, como ilustrado na figura 11:

Figura 11 – Abrir arquivo DBF

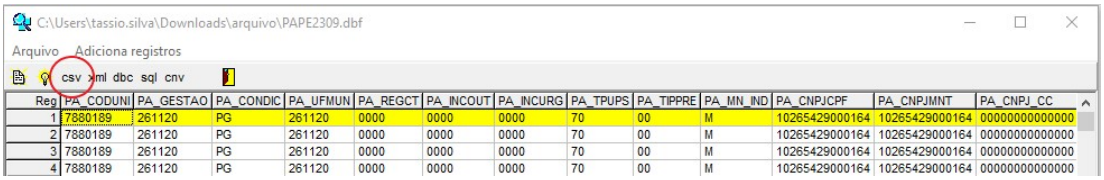


Fonte: Próprio Autor, 2023.

Infelizmente o Tabwin ainda não gera arquivos JSON a partir de bases DBF, evidenciando então a necessidade de obter dados naquele formato a partir das opões que o programa oferece para exportação (CSV, XML, DBC, SQL e CNV).

Por questões de praticidade, a opção escolhida foi o CSV por ser um formato leve e de fácil manipulação. A figura 12 mostra como gerar um arquivo CSV a partir de uma base DBF no Tabwin:

Figura 12 - Gerar arquivo no formato CSV



Reg	PA_CODUNI	PA_GESTAO	PA_CONDIC	PA_UFMUN	PA_REGCT	PA_INCOUT	PA_INCURG	PA_TPUPS	PA_TIPPRE	PA_MN_IND	PA_CNPJCPF	PA_CNPJMNT	PA_CNPJ_CC
1	7880189	261120	PG	261120	0000	0000	0000	70	00	M	10265429000164	10265429000164	00000000000000
2	7880189	261120	PG	261120	0000	0000	0000	70	00	M	10265429000164	10265429000164	00000000000000
3	7880189	261120	PG	261120	0000	0000	0000	70	00	M	10265429000164	10265429000164	00000000000000
4	7880189	261120	PG	261120	0000	0000	0000	70	00	M	10265429000164	10265429000164	00000000000000

Fonte: Próprio Autor, 2023.

O processo de expansão e exportação para formato CSV foi o mesmo para ambos as bases (RDPE2309 e PAPE2309).

A última etapa da preparação dos dados é a conversão das bases para JSON. Nesta etapa, foi implementado código em Java para manipular os arquivos CSV de forma que fossem consideradas apenas as linhas com procedimentos realizados pelo HC-UFPE.

A identificação dos procedimentos realizados no HC-UFPE está implícita nas colunas “PA_CODUNI” (PAPE2309) e “CNES” (RDPE2309). Ambas as colunas citadas representam o identificador do CNES, único para cada estabelecimento de saúde.

O HC-UFPE possui identificador CNES de número 0000396. Dessa forma, é possível identificar exclusivamente a produção do hospital, usando este valor como referência.

Para ambas as bases – já convertidas em CSV – o algoritmo recebe o endereço onde o arquivo se encontra e o local desejado para salvar o resultado em formato JSON.

Basicamente, o código percorre linha por linha das bases CSV, localiza aquelas que possuem o valor 0000396 na coluna PA_CODUNI ou CNES, armazena essas linhas numa lista e por fim cria um arquivo JSON no diretório especificado.

Dessa forma, gerados os arquivos JSON com a produção ambulatorial e hospitalar do HC-UFPE, é possível armazenar as informações no banco de dados através de requisição POST, com o conteúdo dos arquivos como corpo da mensagem.

Importante destacar que todo esse processo (*download* das bases em DBC, expansão para DBF, criação dos arquivos CSV e posterior conversão para JSON) foi pensado para aproveitar todas as colunas disponíveis das bases de dados, ampliando as possibilidades de aplicação em cenários futuros.

3.2 Implementação da API REST com Spring Boot

O projeto foi iniciado na IDE Eclipse, versão 4.29.0, através da opção *New* > *Maven Project*. O *Maven* é uma ferramenta de gerenciamento e automação que simplifica a configuração de um projeto graças ao seu poderoso gerenciamento de dependências. Além do vasto e crescente repositório central, o *Maven* fornece estrutura padrão para vários projetos (MAVEN, 2023b).

Arquétipos são modelos de projetos com configurações pré-definidas. Seu uso, além de ser considerado boa prática, inclui benefícios como: reutilização de conhecimentos e padrões, além da redução de erros e ganho de produtividade (MAVEN, 2023a). Neste projeto, foi utilizado o arquétipo *springboot-rest-api*, disponível no repositório central do *Maven*.

Selecionado o arquétipo, o *Maven* assume o gerenciamento da estrutura do projeto, dependências e construção, tornando o desenvolvimento mais rápido. Por fim, como linguagem de programação, a escolha foi o Java 17 LTS por ser uma versão com tempo de suporte estendido e trabalhar bem com o *Spring Boot* 3.

Seguindo o padrão REST, a API apresenta em sua estrutura os seguintes aspectos, elencados a seguir:

3.2.1 Recursos

A ferramenta expõe dados de procedimentos SUS através das entidades *produção_hospitalar* e *produção_ambulatorial*, ambas com dados sobre tipos de financiamento, complexidade, quantidades, valores, Classificação Internacional de Doenças (CID10), dentre outros.

3.2.2 Design de URI's e verbos HTTP

O design dos Identificadores Universais de Recursos foi pensado de forma a referenciar intuitivamente os dados aos quais apontam, seguindo uma estrutura padrão, iniciando com a modalidade (ambulatório ou hospital) e especificando em seguida alguns dos parâmetros que deverão ser informados. As tabelas 2 e 3 especificam os tipos de requisição, URI e respostas:

Tabela 2 - URIs para dados hospitalares

Verbo	URI	Resposta
GET	/hospital	Produção hospitalar geral.
GET	/hospital/anomescompetencia/{ano_cmpt}/{mes_cmpt}	Produção por ano e mês de competência.
GET	/hospital/ano/{ano_cmpt}	Produção do ano informado como parâmetro.
GET	/hospital/anogrupo/{ano_cmpt}/{proc_grp}	Produção por ano e grupo de procedimentos.
GET	/hospital/anosubgrupo/{ano_cmpt}/{proc_subgrp}	Produção por ano e subgrupo de procedimentos.
GET	/hospital/anoforma/{ano_cmpt}/{proc_forma}	Produção por ano e forma de organização.
GET	/hospital/anomesgrupo/{ano_cmpt}/{mes_cmpt}/{proc_grp}	Produção por ano, mês e grupo de procedimentos.
GET	/hospital/anomessubgrupo/{ano_cmpt}/{mes_cmpt}/{proc_subgrp}	Produção por ano, mês e subgrupo de procedimentos.
GET	/hospital/anomesforma/{ano_cmpt}/{mes_	Produção por ano, mês e

	cmpt}/{proc_forma}	forma de organização.
POST	/hospital	Salva a lista de procedimentos informada.
PUT	/hospital	Edita procedimento hospitalar
DELETE	/hospital/deletaanocomp/{ano_cmpt}/{mes_cmpt}	Deleta todos os procedimentos realizados no ano e mês informados.

Fonte: Próprio Autor, 2023

Tabela 3 - URIs para dados ambulatoriais

Verbo	Recurso	Resposta
GET	/ambulatorio	Produção ambulatorial geral.
GET	/ambulatorio/mesanocompetencia/{mes_cmp}/{ano_cmp}	Produção por mês e ano de competência.
GET	/ambulatorio/anocompetencia/{ano_cmp}	Produção do ano informado como parâmetro.
GET	/ambulatorio/anogrupo/{ano_cmp}/{proc_grp}	Produção por ano e grupo de procedimentos.
GET	/ambulatorio/anosubgrupo/{ano_cmp}/{proc_subgrp}	Produção por ano e subgrupo de procedimentos.
GET	/ambulatorio/anoforma/{ano_cmp}/{proc_forma}	Produção por ano e forma de organização.
GET	/ambulatorio/anomesgrupo/{ano_cmp}/{mes_cmp}/{proc_grp}	Produção por ano, mês e grupo de procedimentos.
GET	/ambulatorio/anomessubgrupo/{ano_cmp}/{mes_cmp}/{proc_subgrp}	Produção por ano, mês e subgrupo de procedimentos.
GET	/ambulatorio/anomesforma/{ano_cmp}/{mes_cmp}/{proc_forma}	Produção por ano, mês e forma de organização.
POST	/ambulatorio	Salva a lista de procedimentos informada.
PUT	/ambulatorio	Edita procedimento ambulatorial.
DELETE	/ambulatorio/deletarmesano/{mes_cmp}/{ano_cmp}	Deleta todos os procedimentos realizados no mês e ano

		informados.

Fonte: Próprio Autor, 2023

3.2.3 Representação de recursos

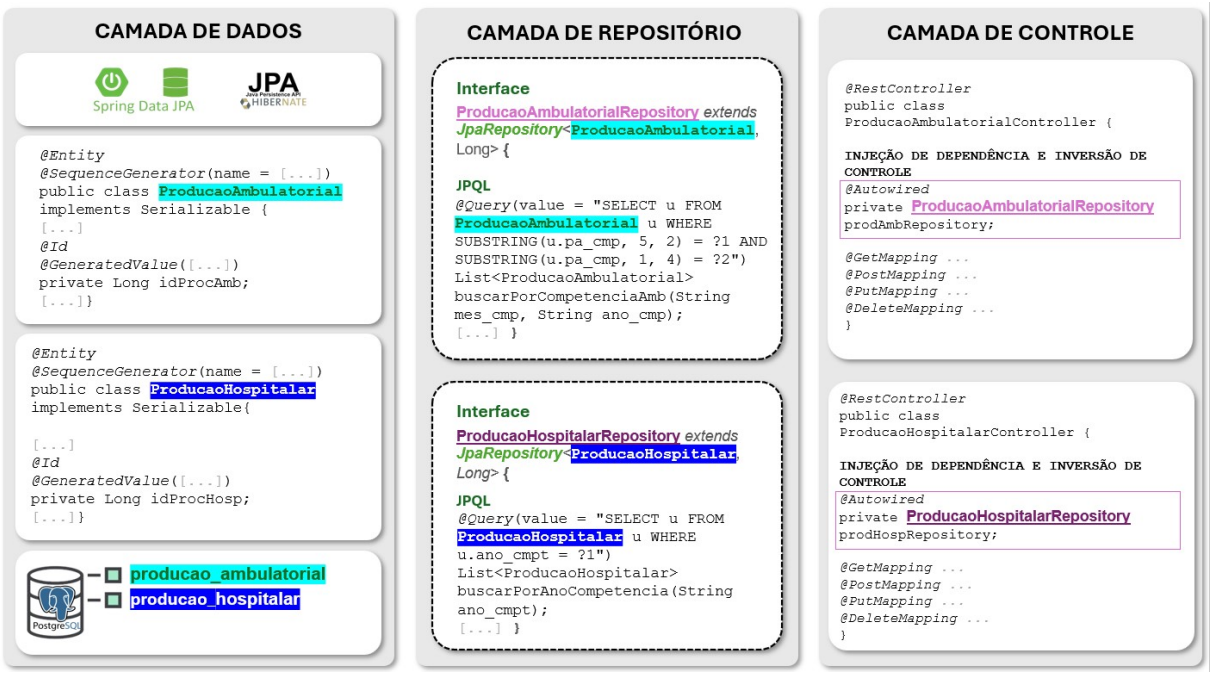
Os dados estão representados em JSON, por ser um formato bastante difundido quando se trata de APIs. As informações são apresentadas em *arrays* de objetos das classes *ProducaoAmbulatorial* ou *ProducaoHospitalar*, com resultados compilados de acordo com a requisição HTTP realizada.

A escolha pelo JSON se deu pela facilidade de uso do formato, que foi adotado como padrão de formatação de dados já nas etapas de extração de informações dos arquivos CSV gerados pelo Tabwin Desktop. O formato teve preferência em relação ao XML por ser legível e de fácil entendimento, possuir amplo suporte e estrutura simples graças à dualidade chave-valor.

3.2.4 Camadas e componentes

As camadas e componentes da API estão representados na figura 13 a seguir. A camada de dados é responsável por modelar e armazenar as informações da aplicação num banco. A camada de repositório, por sua vez, gerencia a persistência das informações, se comunicando com a camada de dados. É composta por interfaces que estendem as operações de repositório do *Spring Data JPA*. Já a camada de controle fica responsável pelo controle geral da aplicação. Nela, são implementadas as requisições HTTP. Também é nesta camada que os conceitos de inversão de controle e injeção de dependência são colocados em prática.

Figura 13 – Representação geral das camadas da aplicação



Fonte: Próprio Autor, 2023.

3.2.4.1 Camada de dados ou modelo

Utiliza recursos do *Spring Data JPA* para facilitar o mapeamento dos dados para as tabelas correspondentes no banco de dados. Membro da família *Spring Data*, o *Spring Data JPA* facilita a implementação de repositórios baseados em *Jakarta Persistence API* (JPA), além de automatizar a integração e criação de consultas (SPRING, 2024a).

O JPA, por sua vez, é uma API Java para gerenciamento de dados que estabelece padrões de mapeamento objeto-relacional. Oferece abstração para operações de bancos de dados, tornando o desenvolvimento mais eficiente (KONDA, 2014).

A camada ou modelo de dados abriga a definição das classes que compõem os recursos oferecidos pela API: *ProducaoAmbulatorial* e *ProducaoHospitalar* (figura 13). O *Spring Data JPA* gerencia a criação das tabelas no banco de dados PostgreSQL e faz o mapeamento automático das classes através da anotação @Entity. Os atributos, então, tornam-se colunas e o nome da tabela é definido com

base no nome da classe ou através da anotação @Table. No caso deste projeto, @Table não foi utilizado.

A estratégia de atualização foi definida através da propriedade *spring.jpa.hibernate.ddl-auto=update* definida no arquivo pom.xml. A opção *update* faz com que o *Hibernate* sincronize o esquema do banco com o modelo de dados, sempre que houver alguma alteração. No entanto, o uso dessa estratégia apresenta riscos de perda de dados caso alguma atualização mais complexa seja realizada no código. Mesmo assim, a solução foi adotada neste trabalho pela facilidade que agrega no processo de desenvolvimento e pelo fato das operações que alteram o banco estarem restritas ao desenvolvedor da aplicação, diminuindo assim os riscos de perda de informação.

O *Hibernate* é um *framework* baseado em JPA que visa facilitar o mapeamento objeto-relacional através de arquivo XML ou *anotações.* Pode ser usado para organizar relacionamentos um-pra-muitos e muitos-para-muitos. Também dispõe de recursos para recuperação e consulta de dados, além de poder gerar consultas em SQL (KONDA, 2014).

Como se trata de uma aplicação simples, não houve necessidade de modelar relacionamentos entre as classes, visto que as informações de ambas são independentes, resultando num modelo de dados bastante simplificado, mas que atende perfeitamente à proposta do projeto.

3.2.4.2 Camada de repositório

Para esta camada, foram criadas duas interfaces: uma para gerenciar consultas referentes à produção hospitalar e outra responsável pelo acesso aos dados ambulatoriais. Ambas as interfaces necessitaram de métodos personalizados com consultas definidas através de *Jakarta Persistence Query Language* (JPQL), linguagem de consulta orientada a objetos e utilizada para acessar dados em bancos relacionais (CORDEIRO, 2012). Tais métodos foram implementados na camada de controle, para retornar dados específicos através de requisições GET.

3.2.4.3 Camada de controle:

Conforme a figura 13, suas classes recebem a anotação @RestController para tornar explícito que se trata de classes que processam requisições HTTP. As classes retornam dados em algum tipo de formato diretamente para o cliente. A camada abarca também a implementação dos métodos declarados nas interfaces da camada de repositório, definindo as requisições e parâmetros para cada *endpoint*, além de estruturar as saídas de cada método.

Através da anotação *@Autowired*, o container de inversão de controle do Spring passa a gerenciar e instanciar as dependências da camada de repositório. Essas dependências serão necessárias para a execução das consultas ao banco de dados, através das requisições HTTP.

A figura 14 demonstra a injeção das dependências contidas em *ProducaoAmbulatorialRepository* na classe *ProducaoAmbulatorialController*. A criação do objeto *prodAmbRepository* fica então sob responsabilidade do *Spring Core Container*, configurando um exemplo claro de inversão de controle. O objeto em questão fornecerá as dependências relacionadas à busca, escrita e exclusão de dados. As operações citadas são parte crucial das requisições HTTP, presentes na API. O Spring Data JPA possui papel de destaque nestas operações com dados, uma vez que já dispõe de diversas codificações prontas para uso.

Figura 14 - Inversão de controle e injeção de dependências com @Autowired

```
@RestController
public class ProducaoAmbulatorialController {

    @Autowired
    private ProducaoAmbulatorialRepository prodAmbRepository;
```

Fonte: Próprio Autor, 2023.

Como a API trabalha com informações ambulatoriais e hospitalares, foram desenvolvidas camadas individuais para cada tipo de informação. A implementação referente aos dados hospitalares não está sendo mostrada porque segue

estritamente a mesma lógica da contraparte ambulatorial. As diferenças estão apenas nos nomes de classes, variáveis, interfaces, métodos e URIs.

3.2.5 Requisição GET

A figura 15 exibe o trecho do código responsável por retornar para o usuário a listagem de todos os procedimentos ambulatoriais. As anotações *@GetMapping* e *@ResponseBody* são componentes do *Spring framework*. A primeira aciona o método “listarProcedimentoAmb()”, sempre que uma requisição GET for realizada através do endereço “/ambulatorio”. A segunda indica ao *Spring* que o retorno do método deverá ser usado como corpo da resposta HTTP.

Figura 15 – Implementação da requisição GET

```
@GetMapping(value = "ambulatorio")
@ResponseBody
public ResponseEntity<List<ProducaoAmbulatorial>> listarProcedimentosAmb() {

    List<ProducaoAmbulatorial> procedimentos = prodAmbRepository.findAll();

    return new ResponseEntity<List<ProducaoAmbulatorial>>(procedimentos, HttpStatus.OK);
}
```

Fonte: Próprio Autor, 2024.

Na declaração do método, foi utilizada a classe *ResponseEntity* do *Spring*. Esta classe permite gerar uma resposta mais detalhada, incluindo o código de *status*, por exemplo.

Os códigos de *status* são formados por três dígitos e constituem parte fundamental do protocolo HTTP. Eles desempenham papel importante na comunicação entre cliente e servidor, por informarem o resultado de uma solicitação. A tabela 4 relaciona e dá alguns exemplos de códigos de *status*:

Tabela 4 - Códigos de status HTTP

1xx (*Informational*)	Indica recebimento da solicitação e iminência de novas informações para o cliente.

2xx (*Successful*)	Solicitação recebida e aceita com sucesso. Exemplos: código “200 *OK*” e “201 *CREATED*”.
3xx (*Redirection*)	Informa que novas ações ou redirecionamentos serão necessários para completar a requisição.
4xx (*Client Error*)	Representa um erro por parte do cliente. Exemplo: “404 *Not Found*” denota que o recurso alvo da requisição não foi encontrado.
5xx (*Server Error*)	Significa que houve um erro do servidor. Exemplo: “500 *Internal Server Error*” expõe falha interna no servidor.

Fonte: IETF, 2022.

O objeto *prodAmbRepository*, através do método *findAll()* do *Spring Data* JPA, realiza a busca de todos os procedimentos e os armazena numa lista.

O método “ListarProcedimentosAmb()” terá então, como retorno, um objeto da classe *ResponseEntity*, encapsulando a lista de procedimentos obtida através de *prodAmbRepository.findAll. O Spring* converte a lista automaticamente para o formato JSON e retorna as informações na tela para o usuário, como o corpo da resposta HTTP. Ainda na declaração de retorno, o *status* é definido como *OK* (200).

3.2.6 Requisição POST

A figura 16 representa a codificação do método “SalvarProducaoAmb()”, cujo propósito é salvar uma lista de procedimentos no banco. A anotação *@PostMapping* dispara este método sempre que uma requisição POST é realizada através do endereço “/ambulatorio”. Já *@ResponseBody* indica que o retorno do método deverá ser incorporado ao corpo da resposta HTTP, ou seja: após salvar uma lista de procedimentos, a mesma será retornada ao usuário.

Figura 16 - Salvar produção ambulatorial

```
@PostMapping(value = "ambulatorio")
@ResponseBody
public ResponseEntity<List<ProducaoAmbulatorial>> SalvarProducaoAmb(
        @RequestBody List<ProducaoAmbulatorial> prodAmb) {

    List<ProducaoAmbulatorial> producaoAmbulatorial = prodAmbRepository.saveAll(prodAmb);

    return new ResponseEntity<>(producaoAmbulatorial, HttpStatus.CREATED);
}
```

Fonte: Próprio Autor, 2024.

Esta implementação faz uso da anotação *@RequestBody*, também do *Spring*, que tem por função garantir que o conteúdo integrante do corpo da mensagem seja utilizado e convertido para Java. Neste caso, tal conteúdo será a lista de procedimentos a ser salva, sendo, portanto, parte obrigatória desta requisição.

O objeto *prodAmbRepository* recebe a lista da produção ambulatorial (“prodAmb”) informada no cabeçalho do método e utiliza a dependência *saveAll() do Spring Data JPA* para converter a mesma produção para Java e armazená-la na lista “producaoAmbulatorial”.

A resposta é então encapsulada num objeto *ResponseEntity*, contendo a lista que foi salva e o *status CREATED*, indicando que as informações foram devidamente salvas no banco.

3.2.7 Requisição PUT

O funcionamento da requisição PUT é demonstrado na figura 17. Nela, a anotação *@PutMapping* do *Spring* invoca o método “atualizarProcedimento()” sempre que uma solicitação PUT for requisitada através do endereço “/ambulatório”. O método citado recebe o procedimento que deverá ser alterado. Novamente, a anotação *@RequestBody* define que os dados fornecidos no corpo da mensagem – “procedimentoAmbulatorial”, neste caso – deverão ser considerados e convertidos para um objeto Java.

Figura 17 - Editar procedimento ambulatorial

```
@PutMapping(value="ambulatorio")
@ResponseBody
public ResponseEntity<?> atualizarProcedimento(@RequestBody ProducaoAmbulatorial procedimentoAmbulatorial){

    if(procedimentoAmbulatorial.getIdProcAmb() == null) {
        return new ResponseEntity<String>("ID do procedimento não informado.", HttpStatus.OK);
    }

    ProducaoAmbulatorial procAmb = prodAmbRepository.saveAndFlush(procedimentoAmbulatorial);

    return new ResponseEntity<ProducaoAmbulatorial>(procAmb,HttpStatus.OK);
}
```

Fonte: Próprio Autor, 2024.

O sinal de interrogação logo após o *ResponseEntity* indica um tipo genérico. Seu uso serve para definir que o método pode retornar qualquer tipo de resposta. No caso deste trecho de código, o retorno pode ser uma mensagem de texto ou um objeto JSON, que corresponde ao procedimento ambulatorial que foi alterado com sucesso.

Dentro do método, também há uma validação do identificador do procedimento (idProcAmb), declarado na camada de dados. Caso o valor do idProcAmb fornecido seja inválido, o método retorna uma mensagem de texto informando erro.

Uma vez que, dentro do corpo da mensagem, o identificador do procedimento a ser editado esteja correto, as alterações serão registradas. Para isso, o objeto *prodAmbRepository* recebe o procedimento ambulatorial devidamente editado e grava as informações no banco, através do método *saveAndFlush()*. Este último método é parte do *Spring Data* JPA.

Como retorno, o objeto alterado é obtido em JSON e o código exibe *status OK*.

A requisição PUT foi implementada para satisfazer os requisitos do nível 2 do modelo de Richardson, que pressupõe operações CRUD (*Create, Read, Update* e *Delete*) em uma API. No entanto, a alteração de qualquer procedimento descaracteriza as informações obtidas através do DATASUS, tornando os dados inconsistentes.

3.2.8 Requisição DELETE

Como mostra a figura 18, a operação de exclusão é estruturada a partir da anotação *@DeleteMapping* do *Spring, que* dispara o método “deletarPorCompetenciaAmb()” todas as vezes em que uma requisição DELETE for realizada através do endereço “ambulatorio/deletarmesano/{mes_cmp}/{ano_cmp}”. Os campos entre chaves correspondem ao período em que se encontram os dados a serem excluídos. O usuário deve então informá-los, de forma a viabilizar a operação.

Figura 18 - Deletar procedimentos realizados no mês

```
@DeleteMapping(value = "ambulatorio/deletarmesano/{mes_cmp}/{ano_cmp}")
public ResponseEntity<Void> deletarPorCompetenciaAmb(@PathVariable String mes_cmp, @PathVariable String ano_cmp) {
    List<ProducaoAmbulatorial> procedimentos = prodAmbRepository.buscarPorCompetenciaAmb(mes_cmp, ano_cmp);

    if (procedimentos.isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }

    prodAmbRepository.deleteAll(procedimentos);

    return new ResponseEntity<>(HttpStatus.OK);
}
```

Fonte: Próprio Autor, 2024.

Uma vez fornecidos os parâmetros, a anotação *@PathVariable* mapeia mês e ano para dentro do método “deletarPorCompetenciaAmb()”.

A partir disso, os procedimentos são localizados na base de dados através da função “buscarPorCompetenciaAmb()”, definida em JPQL na interface “ProducaoAmbulatorialRepository”.

A figura 19 exibe a consulta JPQL definida para localizar o conjunto de dados que serão excluídos. Acima do método, é declarada a anotação *@Query* que recebe uma instrução SQL.

Figura 19 – Uso do JPQL para retornar procedimentos de acordo com mês e ano informados.

```
@Query(value = "SELECT u FROM ProducaoAmbulatorial u WHERE SUBSTRING(u.pa_cmp, 5, 2) = ?1 AND SUBSTRING(u.pa_cmp, 1, 4) = ?2")
List<ProducaoAmbulatorial> buscarPorCompetenciaAmb(String mes_cmp, String ano_cmp);
```

Fonte: Próprio Autor, 2024.

Assim que o método “buscarPorCompetenciaAmb()” é executado, os parâmetros (mês e ano) são repassados para a consulta. No banco, mês e ano são representados em uma única coluna chamada “pa_cmp”. Portanto, foi necessário fazer uso da função SUBSTRING para separar os dados. Os primeiros quatro dígitos da coluna “pa_cmp” correspondem ao ano, enquanto os dois últimos são referentes ao mês.

Dando sequência à lógica da requisição DELETE, o método “buscarPorCompetenciaAmb()” retorna um conjunto de procedimentos que atendem aos parâmetros informados e são armazenados numa lista.

Caso a lista esteja vazia, o *status NOT_FOUND* é exibido. Do contrário, “prodAmbRepository” excluirá todos os procedimentos da lista, através do método “deleteAll()”, do *Spring Data* JPA, e o *status OK* será mostrado ao usuário.

4 TESTES E ANÁLISE DE RESULTADOS

Como resultado das etapas descritas na seção anterior, obteve-se uma aplicação focada nos requisitos indispensáveis ao usuário, evitando gasto de energia com funcionalidades desnecessárias. O produto contempla os princípios do padrão REST e se mostrou bastante eficiente nas operações com os dados da produção do HC-UFPE.

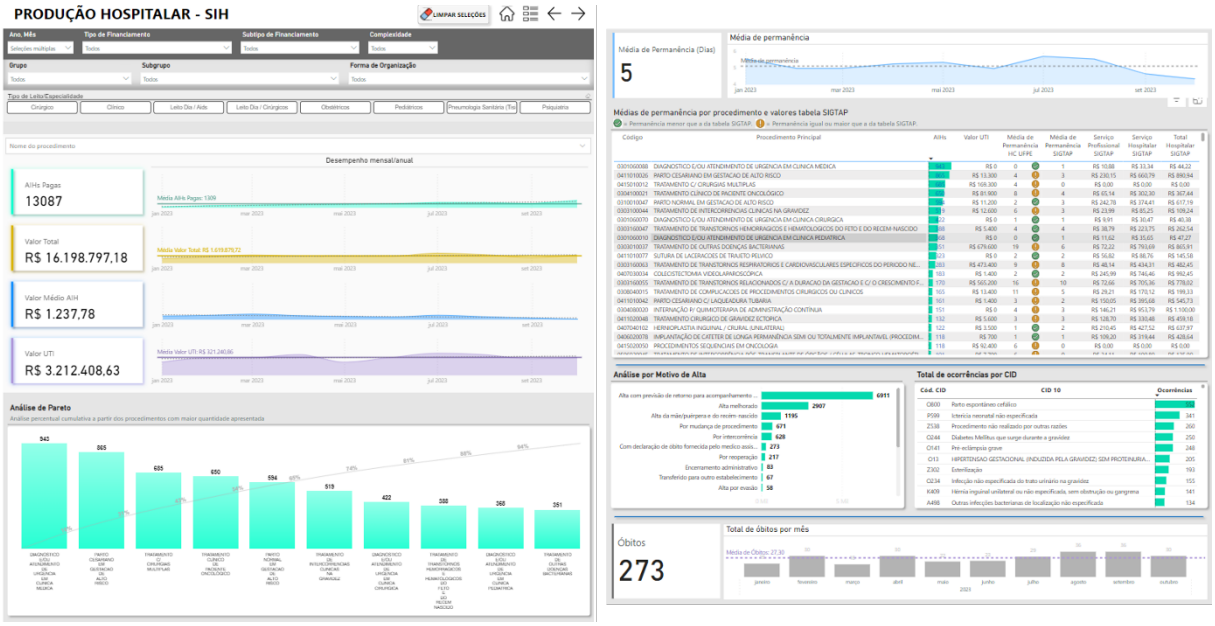
Foram realizados testes via *browser* das requisições GET através dos URIs, retornando com sucesso dados de produção de acordo com os parâmetros informados. As requisições POST, PUT e DELETE foram testadas e validadas através da plataforma de API *Postman*. Esta ferramenta oferece um ambiente onde

é possível executar operações diversas em APIs, tais como: requisições CRUD, testes de integração, testes de cenários, visualização de dados e criação de documentação (POSTMAN, 2023).

Houve sucesso na integração dos dados da API com o Power BI, evidenciando ainda mais a relevância do trabalho. A ferramenta da Microsoft obteve acesso aos recursos da API através de requisição GET. As informações providas puderam ser lidas, tratadas e utilizadas para construção de gráficos através da plataforma. O processo de desenvolvimento de *dashboards* envolve diversas outras etapas que não estão previstas para este artigo.

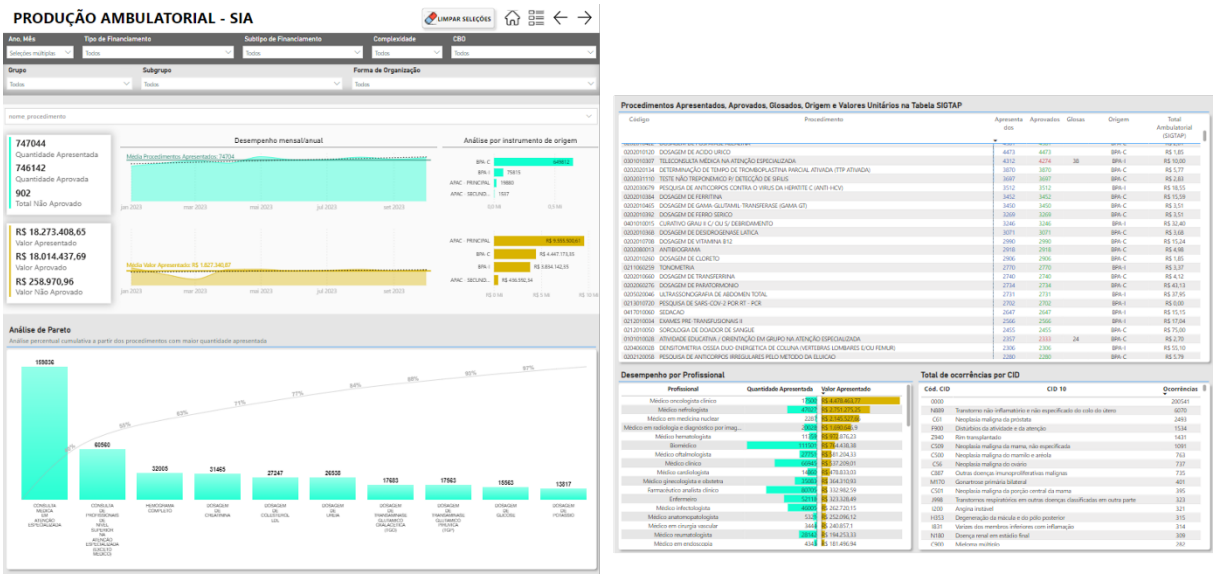
As figuras 20 e 21 representam seções de um *dashboard* em Power BI que consome dados da API. O referido painel está disponível para acesso ao público através do site do HC-UFPE, na seção “Governança” -> “Gestão por Indicadores”.

Figura 20 - Painel de Produção Hospitalar



Fonte: Próprio Autor, 2024.

Figura 21 - Painel de Produção Ambulatorial



Fonte: Próprio Autor, 2024.

O uso com o Power BI é o principal cenário de uso desta API e a eleva a um patamar estratégico junto às demais ferramentas de gestão disponíveis no hospital. Os gestores agora podem contar com mais uma fonte de dados oficiais para geração de métricas que irão alimentar as tomadas de decisão. Com isso, espera-se que novas estratégias sejam definidas para otimizar o uso de recursos públicos.

O resultado alcançado foi considerado um sucesso, por resolver a necessidade estratégica do hospital de possuir um serviço inovador que armazene e disponibilize informações de produção para todos os gestores. Isto também corrobora para a criação e fortalecimento da cultura de dados na instituição, abrindo novas possibilidades para melhorias nos serviços oferecidos à população através do SUS.

A principal dificuldade encontrada no projeto foi a impossibilidade de baixar as bases já em formato JSON contendo exclusivamente a produção do HC-UFPE. Este fato foi contornado na etapa de extração dos dados através da criação de algoritmo de limpeza e conversão dos arquivos CSV gerados pelo Tabwin Desktop.

5 CONSIDERAÇÕES

O desenvolvimento deste trabalho proporcionou a criação de um serviço indispensável para gestores que, até o momento, não é oferecido pelo DATASUS. Isto, por si só, configura um ato genuíno de inovação no âmbito da administração pública hospitalar no estado. Ferramentas dessa natureza agregam muito valor e impactam positivamente a gestão, graças ao interesse das lideranças em dispor de mais subsídios para nortear a condução dos estabelecimentos que governam.

O projeto executado também contribui de maneira sólida para que os benefícios de uma cultura voltada a dados sejam devidamente reconhecidos. As informações agora estão facilmente disponíveis para serem integradas a ferramentas de análise de dados. Esse cenário favorece o processo de tomada de decisão, tornando-o mais ágil, graças à geração de mais *insights* através dos dados analisados no Power BI.

A análise dos dados fornecidos pela API também é útil para revelar pontos críticos até então despercebidos pela gestão. A partir destas informações, novas estratégias podem ser criadas para reduzir custos, mitigar riscos e amenizar impactos no serviço à população. Além disso, a integração da ferramenta com o Power BI torna possível a criação de instrumentos de transparência institucional, corroborando com as boas práticas de gestão, através da prestação de contas à sociedade.

Os resultados podem se estender ao nível operacional, traduzindo-se em mais tempo livre para os gestores focarem em tarefas críticas. O processo manual de obtenção de dados através do DATASUS demanda certo tempo e esforço. Com a API, os dados do hospital estarão disponíveis a qualquer momento.

Ao utilizar a ferramenta, o gestor também promove o bom uso do gasto público, uma vez que todo o esforço do governo para manter os SIS é compensado através das decisões embasadas pelos dados obtidos.

Graças à padronização das bases de dados do DATASUS, os benefícios desta API poderão ser estendidos a outros estabelecimentos SUS, sendo necessário apenas informar, durante o processo de extração de dados, o identificador CNES do hospital onde será implantada.

Por fim, como trabalhos futuros, estão previstos a disponibilização da API a nível institucional, a implementação da camada de serviços e *hiperlinks*, expansão da base de dados, integração com *Spring Security* para autenticação de usuários e o desenvolvimento de novos painéis a partir dos dados da API.

REFERÊNCIAS

AWS. **O que é a API RESTful? - Explicação sobre a API RESTful - AWS**. [S.l.]. Amazon, 2023. Disponível em: https://aws.amazon.com/pt/what-is/restful-api/. Acesso em: 29 dez. 2023.

BIEHL, Matthias. **RESTful API Design: API-University Press**, v. 3, 2016. 296 p. (API-University Series). ISBN: 978-1514735169.

BRITO, Michelli (org.). **Spring Boot: Da API REST aos Microservices**. 2023. E-book (55p.) color. Disponível em: https://github.com/MichelliBrito/springboot-api-ebook. Acesso em: 1 set. 2023.

CORDEIRO, Gilliard. **Aplicações Java para a web com JSF e JPA**. São Paulo: Casa do Código, 2012.

DATASUS. **DATASUS**. [S.l.]. Ministério da Saúde, 2023. Disponível em: https://DATASUS.saude.gov.br/sobre-o-DATASUS/. Acesso em: 25 nov. 2023.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese (Doutorado) - Information and Computer Science, University of California, Irvine, 2000. Disponível em: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 27 jan. 2024.

FRANCO, Joel Levi Ferreira. **Sistemas da Informação**. Unifesp. Disponível em: https://www.unasus.unifesp.br/biblioteca_virtual/esf/13/Unidade3/Sistemas_de_Informacao/p_04.html. Acesso em: 19 dez. 2023

GAMMA, Erich. et al. **Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos**. Porto Alegre: BOOKMAN® COMPANHIA EDITORA, 2008.

GEEKHUNTER. **Spring Boot: Tudo que você precisa saber!** - Blog de TI. Disponível em: https://blog.geekhunter.com.br/tudo-o-que-voce-precisa-saber-sobre-o-spring-boot/. Acesso em: 19 dez. 2023.

IBGE. **Cadastro Nacional de Estabelecimentos de Saúde – CNES**. [S.l.]. IBGE - Instituto Brasileiro de Geografia e Estatística, 2023a. Disponível em: https://ces.ibge.gov.br/base-de-dados/metadados/ministerio-da-saude/cadastro-nacional-de-estabelecimentos-de-saude-cnes.html. Acesso em: 19 dez. 2023.

IBGE. **Sistema de Informações Ambulatoriais do SUS – SIA/SUS**. [S.l.]. IBGE - Instituto Brasileiro de Geografia e Estatística, 2023b. Disponível em: https://ces.ibge.gov.br/base-de-dados/metadados/ministerio-da-saude/sistema-de-informacoes-ambulatoriais-do-sus-sia-sus.html. Acesso em: 19 dez. 2023.

IBGE. **Sistema de Informações Hospitalares do SUS – SIH/SUS**. [S.l.]. IBGE - Instituto Brasileiro de Geografia e Estatística, 2023c. Disponível em: https://ces.ibge.gov.br/base-de-dados/metadados/ministerio-da-saude/sistema-de-informacoes-hospitalares-do-sus-sih-sus.html. Acesso em: 19 dez. 2023.

JOHNSON, Rod. et al. **Professional Java Development with the Spring Framework**. 1ª ed. Indianapolis: Wrox, 2005.

JUNIT. **JUnit 5 User Guide**. [S.l.]. JUnit.org, 2023. Disponível em: https://junit.org/junit5/docs/current/user-guide/. Acesso em: 29 dez. 2023.

KONDA, Madhusudhan. **Introdução ao Hibernate**. Tradução: Lúcia Ayako Kinoshita. 1ª ed. São Paulo: Novatec Editora Ltda., 2014. Título original: Just Hibernate.

MANN, Howie. (2023). **REST API** [Diagrama]. Mann Howie. https://mannhowie.com/rest-api

MAVEN. **Introduction to Archetypes**. [S.l.]. The Apache Software Foundation, 2023a. Disponível em: https://maven.apache.org/guides/introduction/introduction-to-archetypes.html. Acesso em: 19 dez. 2023.

MAVEN. **Maven Features**. [S.l.]. The Apache Software Foundation, 2023b. Disponível em: https://maven.apache.org/maven-features.html. Acesso em: 29 dez. 2023.

MICROSOFT. **Power BI - Visualização de dados**. [S.l.]. Microsoft, 2023. Disponível em: https://www.microsoft.com/pt-br/power-platform/products/power-bi. Acesso em: 7 dez. 2023.

ORACLE. **Java Software**. [S.l.]. Oracle, 2023. Disponível em: https://www.oracle.com/java/. Acesso em: 29 dez. 2023.

POSTMAN. **Postman API Platform**. [S.l.]. Postman, Inc., 2023. Disponível em: https://www.postman.com/home. Acesso em: 29 dez. 2023.

IETF. **RFC 9110: HTTP Semantics**. [S.l.]. Internet Engineering Task Force (IETF), 2022. Disponível em: https://www.rfc-editor.org/rfc/rfc9110.html#name-status-codes. Acesso em: 27 jan. 2024.

SINAN. **SINANWEB - Tabwin**. [S.l.]. SISTEMA DE INFORMAÇÃO DE AGRAVOS DE NOTIFICAÇÃO, 2023. Disponível em: https://portalsinan.saude.gov.br/sistemas-auxiliares/TABWIN. Acesso em: 19 dez. 2023.

SPRING. **Spring Data JPA**. [S.l.]. Broadcom Inc., 2024a. Disponível em: https://spring.io/projects/spring-data-jpa/. Acesso em: 2 jan. 2024.

SPRING. **Why Spring**. [S.l.]. Broadcom Inc., 2024b. Disponível em: https://spring.io/why-spring/. Acesso em: 2 jan. 2024.

SUBRAMANIAN, Harihara; RAJ, Pethuru. **Hands-On RESTful API Design Patterns and Best Practices**: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs. Birmingham: Packt, 2019.