****

**FACULDADE DE TECNOLOGIA DE TAUBATÉ**

**GUILHERME COSTA DE AGUIAR**

**SISTEMA INTEGRADOR DE DADOS BÁSICOS DO MUNÍCIPE**

**TAUBATÉ**

**2022**

****

**FACULDADE DE TECNOLOGIA DE TAUBATÉ**

**GUILHERME COSTA DE AGUIAR**

**SISTEMA INTEGRADOR DE DADOS BÁSICOS DO MUNÍCIPE**

Trabalho de Graduação apresentado à Coordenação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Centro Estadual de Educação Tecnológica Paula Souza para a obtenção do diploma de Tecnólogo em Análise e Desenvolvimento de Sistemas.

**Orientador: Prof. Luis Felipe Feres Santos**

**TAUBATÉ**

**2022**

**GUILHERME COSTA DE AGUIAR**

**SISTEMA INTEGRADOR DE DADOS BÁSICOS DO MUNÍCIPE**

Trabalho de Graduação apresentado a Faculdade de Tecnologia de Taubaté, como parte das exigências para a obtenção do diploma de Tecnólogo em Análise e Desenvolvimento de Sistemas.

**Orientador: Prof. Luis Felipe Feres Santos**

Taubaté, \_\_\_\_\_ de \_\_\_\_\_\_\_\_\_\_ de 2022.

**BANCA EXAMINADORA**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof.

**LISTA DE ILUSTRAÇÕES**

|  |  |
| --- | --- |
| **FIGURA 1: VISÃO GERAL DO SPRING FRAMEWORK** \_\_\_\_\_\_\_\_\_\_\_ | **10** |
| **FIGURA 2: ARQUITETURA DO SISTEMA** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **17** |
| **FIGURA 3: DIAGRAMA DE CASO DE USO** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **19** |
| **FIGURA 4: DIAGRAMA DE ATIVIDADE: INCLUSÃO DE NOVO MUNÍCIPE** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **20** |
| **FIGURA 5: MODELO CONCEITUAL** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **20** |
| **FIGURA 6: MODELO FÍSICO** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **21** |
| **FIGURA 7: SWAGGER** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **22** |
| **FIGURA 8: RESPOSTA DA REQUISIÇÃO GET** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **23** |

**LISTA DE TABELAS**

|  |  |
| --- | --- |
| **TABELA 1: MÉTODOS HTTP NO PADRÃO RESTFUL**\_\_\_\_\_\_\_\_\_\_\_\_ | 13 |
| **TABELA 2: REQUISITOS FUNCIONAIS**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | 14 |
| **TABELA 3: REQUISITOS NÃO FUNCIONAIS**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | 15 |
| **TABELA 4: CRONOGRAMA** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | 23 |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**LISTA DE ABREVIATURAS E SIGLAS**

|  |  |
| --- | --- |
| **API** | Application Programming Interface |
| **EE** | Entrerprise Edition |
| **HTTP** | HyperText Transfer Protocol |
| **IDE** | Integrated Development Environment |
| **REST** | Representational State Transfer |
| **SGDB** | Sistema Gerenciador de Banco de Dados |
| **SQL** | Standard Query Language |
| **XML** | eXtensible Markup Language |

SUMÁRIO

[**1 INTRODUÇÃO 7**](#_Toc121426157)

[**2 CONTEXTUALIZAÇÃO TECNOLÓGICA 9**](#_Toc121426158)

[**2.1 QUALIDADE DE DADOS 9**](#_Toc121426159)

[**2.2 JAVA 9**](#_Toc121426160)

[**2.3 SPRING FRAMEWORK 10**](#_Toc121426161)

[**2.3.1 SPRING BOOT 11**](#_Toc121426162)

[**2.3.2 SPRING DATA 11**](#_Toc121426163)

[**2.4 IDE ECLIPSE 11**](#_Toc121426164)

[**2.5 MARIADB 12**](#_Toc121426165)

[**2.6 WEB SERVICES 12**](#_Toc121426166)

[**2.7 APPLICATION PROGRAMMING INTERFACE 12**](#_Toc121426167)

[**2.8 SWAGGER 13**](#_Toc121426168)

[**2.9 ARQUITETURA RESTFUL 13**](#_Toc121426169)

[**3 DESENVOLVIMENTO 15**](#_Toc121426170)

[**3.1 LEVANTAMENTO DE REQUISITOS 15**](#_Toc121426171)

[**3.2 ARQUITETURA DO SISTEMA 17**](#_Toc121426172)

[**3.3 DIAGRAMA DE CASO DE USO 18**](#_Toc121426173)

[**3.4 DIAGRAMA DE ATIVIDADE 19**](#_Toc121426174)

[**3.5 MODELAGEM DO BANCO DE DADOS 20**](#_Toc121426175)

[**3.5.1 MODELO CONCEITUAL 20**](#_Toc121426176)

[**3.5.2 MODELO FÍSICO 21**](#_Toc121426177)

[**4 RESULTADOS OBTIDOS 22**](#_Toc121426178)

[**5 CRONOGRAMA 24**](#_Toc121426179)

[**REFERÊNCIAS 25**](#_Toc121426180)

# INTRODUÇÃO

Nesse momento tecnológico, quando todos os setores da sociedade procuram soluções tecnológicas para atender suas demandas, o setor público também se moderniza através de sistemas computacionais.

Tem sido comum municípios contratarem diferentes soluções para os diferentes setores dentro da Administração, criando um ambiente tecnológico heterogêneo.

Diante deste cenário diverso sem a devida interoperabilidade, observa-se: A impossibilidade de extrair dados confiáveis em razão de registros múltiplos onde deveriam ser únicos; O retrabalho em setores que realizam cadastro de dados básicos dos munícipes; A dificuldade na migração de dados durante a substituição dos fornecedores de sistemas informatizados. Esta pesquisa busca formas de permitir a normalização e centralização dos dados básicos dos munícipes das soluções em operação na administração municipal.

Sabendo que a responsabilidade final dos dados armazenados nestes sistemas ainda é do órgão público, e que é de interesse dele a visão geral do cenário, independentemente de quem é o prestador do serviço de informatização, uma abordagem centralizada e reguladora pode sanar as inconsistências e prover a interoperabilidade entre sistemas no aspecto que tange os dados básicos dos munícipes.

Com o objetivo de centralizar e regular o acesso aos dados básicos dos munícipes, é proposto o desenvolvimento em linguagem *Java* com o *framework Spring MVC* uma estrutura centralizada, dotada de um banco de dados relacional utilizando o Sistema Gerenciador de Banco de Dados (SGBD) *MariaDB*, com uma plataforma de gerenciamento para as regras e autorizações de acesso, além de *Application Programming Interface* (APIs) no padrão *Representational State Transfer* (REST) disponibilizadas aos sistemas terceirizados.

A unidade dos dados básicos dos cidadãos reflete em um melhor serviço prestado ao usuário final, evitando retrabalho ao cadastrar e atualizar os mesmos dados em vários sistemas. Também provê dados mais confiáveis para as análises e planejamento de políticas públicas.

Como prover a consistência dos dados básicos dos cidadãos no âmbito da administração municipal através de integração de sistemas terceirizados de forma padronizada?

Para atingir seus objetivos, esta pesquisa apresenta artigos acerca da consistência de dados e então embasa a escolha das tecnologias para o desenvolvimento da aplicação proposta.   Busca através de entrevistas com usuários finais o entendimento de suas necessidades, e cria uma lista de requisitos funcionais e requisitos não funcionais.

Esta pesquisa está estruturada em capítulos e apresenta a fundamentação teórica para seu desenvolvimento, o detalhamento da elaboração da solução proposta e por fim, a conclusão.

# CONTEXTUALIZAÇÃO TECNOLÓGICA

Nesse capítulo serão expostos os fundamentos que permeiam o desenvolvimento do sistema, com a apresentação de textos sobre a importância da qualidade de dados, passando pelas tecnologias e ferramentas propostas para a construção da solução.

## QUALIDADE DE DADOS

De acordo com Rêgo (2013), dados são a matéria-prima necessária para almejar a utilização das informações afim de tomar decisões ágeis corretas.

A importância da qualidade dos dados de um sistema computacional fica evidente quando Barbieri (2020) aponta que dados sem qualidade comumente causam problemas em tomadas de decisões e em aspectos de segurança.

McGilvray (2008, APUD Barbieri, 2020) define um conjunto de características de qualidade de dados, considerando algumas dimensões: Integridade dos dados, completude e validade, unicidade, acurácia, consistência e sincronização, temporalidade, facilidade de uso e acessibilidade, segurança e privacidade, entre outros.

## JAVA

Segundo Oracle (2022), “Java é uma linguagem de programação e plataforma computacional lançada pela primeira vez pela Sun Microsystems em 1995[...]”.

Mendes (2009, p. 17) conceitua:

A linguagem Java é considerada simples porque permite o desenvolvimento de sistemas em diferentes sistemas operacionais e arquiteturas de hardware, sem que o programador tenha que se preocupar com detalhes de infra-estrutura. Dessa forma, o programador consegue desempenhar seu trabalho de uma forma mais produtiva e eficiente.

Ainda de acordo com Mendes (2009), a linguagem Java foi concebida seguindo o paradigma de orientação a objetos, diferenciando da programação estruturada e trazendo formas mais próximas do mundo real para o gerenciamento de um sistema.

A plataforma Java *Entrerprise Edition* (EE) contém um conjunto de tecnologias que reduzem a complexidade e custo de desenvolvimento de aplicações centradas em servidores. Essa plataforma oferece uma coleção de APIs que permitem o desenvolvimento e execução de aplicações robustas e confiáveis no lado do servidor, de acordo com Apache (2022).

## SPRING FRAMEWORK

Segundo Santana (2021), o Spring é um *framework* Java que possui uma grande quantidade de projetos relacionados, como o Spring Boot e o Spring Data. É um projeto robusto e estável, com sua primeira versão publicada em 2002.

Spring (2022) em sua documentação oficial explica que, o Spring Framework constitui-se de uma coleção de recursos organizados em aproximadamente 20 módulos. Esses módulos são reunidos em grupos maiores que possuem conceitos similares em seus propósitos. Os grupos de módulos são: o *Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging* e *Test*, conforme mostrado no diagrama a seguir.

Figura 1: Visão geral do Spring Framework.



Fonte: Spring (2022)

Ainda de acordo com Spring (2022), este *framework* fornece um modelo amplo de programação e configuração para aplicativos desenvolvidos em Java, permitindo o desenvolvedor se concentrar na lógica de negócios no nível de aplicativo, sem se preocupar com detalhes desnecessários dos ambientes de implantação.

### SPRING BOOT

Santana (2021) explica que o Spring Boot é uma maneira simplificada de criar projetos baseados no Spring Framework. A utilização do Spring Boot também facilita a configuração das aplicações através de arquivos específicos ou diretamente na aplicação, descartando a necessidade do uso de arquivos eXtensible Markup Language (XML).

Spring (2022) elenca as principais características do Spring Boot: Possibilita a criação de aplicativos Spring autônomos, permite incorporar servidores Tomcat, Jetty ou Undertow, disponibiliza dependências para simplificar sua configuração de compilação, possibilita a configuração automática de bibliotecas Spring e de terceiros e, por fim, fornece recursos prontos para produção, como métricas, verificações de integridade e configuração externa.

### SPRING DATA

O Spring Data é um projeto que possui diversos módulos relacionados ao acesso de dados em múltiplos sistemas de armazenamento de dados. Não se trata de uma especificação ou padrão, é uma abstração que tenta trazer consistência ao acesso de dados. (PHALTANKAR, 2022)

Kainulainen (2012) complementa ao afirmar que objetivo do Spring Data é fornecer uma maneira fácil para criar aplicações que usem tanto banco de dados relacionais quantos novas tecnologias de armazenamento, como banco de dados não relacionais ou baseados em nuvem, tudo isso utilizando o framework Spring.

Spring (2022) cita como características do Spring Data, o poderoso repositório de abstrações de mapeamento de objetos, suporte para auditoria transparente (criação, última alteração) e integração avançada com controladores do Spring MVC.

### SPRING SECURITY

## IDE ECLIPSE

O ambiente integrado de desenvolvimento, ou *Integrated Development Environment* (IDE) Eclipse é uma plataforma criada a partir do Projeto Eclipse, originalmente criado em novembro de 2001 pela IBM. A Eclipse Foundation foi criada em janeiro de 2004 como uma organização independente e sem fins lucrativos para agir como um mantenedor da comunidade Eclipse, de acordo com Eclipse (2022).

Partindo de um desenvolvimento colaborativo, a comunidade desenvolveu Spring Tools, que, de acordo com Eclipse (2022), são ferramentas para a IDE Eclipse para a escrita de grandes aplicativos, aplicativos modernos e microsserviços baseados no framework Spring Boot.

## MariaDB

O MariaDB é um Sistema Gerenciador de Bando de Dados (SGDB), necessário para a implementação de um banco de dados relacional e confiável.

Segundo MariaDB (2022, tradução nossa) “O servidor Maria DB é um dos servidores de banco de dados mais populares do mundo. É feito pelos desenvolvedors originais do MySQL e garantido em continuar em código aberto. Usuário notáveis incluem Wikipedia, WordPress. Com e Goolge”.

## POSTMAN

De acordo com Postman (2023, tradução nossa), a ferramenta Postman é um software utilizado para testar e documentar APIs (Application Programming Interfaces). Ele foi criado em 2012 por Abhinav Asthana, Ankit Sobti e Abhijit Kane, com o objetivo de facilitar o desenvolvimento e a integração de APIs. Com o Postman, é possível criar e enviar requisições HTTP para uma API, testar suas funcionalidades, validar as respostas e documentar as informações obtidas. O software permite ainda automatizar testes de rotina e colaborar com outros membros da equipe na documentação e desenvolvimento da API.

## WEB SERVICES

Wittig (2015) define *web services* como serviços que podem ser controlados por uma interface web, podendo ser usada por outros sistemas e ou por pessoas, através de uma interface gráfica de usuário.

O cenário heterogêneo e dependente de integração requer soluções que possibilitem isso. Devmedia (2016) afirma que umas tecnologias que possibilitam essa integração são os *web services,* definindo-os como aplicações cliente-servidor que operam através do protocolo *HyperText Transfer Protocol* (HTTP) com o objetivo de possibilitar serviços entre aplicações, mesmo que estas estejam sendo executadas em plataformas distintas. Esses softwares utilizam a linguagem XML como padrão para fornecer as marcações que podem ser interpretadas por outros sistemas, dessa forma, possibilitando inúmeros programas a interagir uns com os outros para fornecer soluções para o cenário de tecnologias e plataformas diversas.

## APPLICATION PROGRAMMING INTERFACE

Uma *Application Programming Interface* (API), permite aos desenvolvedores explorar funções de um sistema de computador em outras aplicações. Com o passar dos anos, as APIs evoluíram apoiadas em tecnologias avançadas, como a velocidade das redes e segurança. (DANIEL et al., 2014)

Ainda de acordo com Daniel et al. (2014), companhias como Google, Facebook e Twitter têm utilizado APIs para expor seus serviços e permitir à desenvolvedores explorar suas funcionalidades.

## SWAGGER

Swagger é um framework usado para descrever API usando linguagem comum que é familiar à todos. Pode ser comparado como uma planta baixa de uma caisa, segundo Prompt Softech (2019).

De acordo com Swagger.io (2022), Swagger começou como uma simples especificação open source para criação de APIs RESTful, em 2010. Em 2015, o projeto foi adquirido pela empresa SmartBear Software, e então a especificação foi doada para a fundação Linux e renomeada para OpenAPI. Desde então, o Swagger se tornou o conjunto de ferramentas mais popular para explorar o potencial da especificação criada inicialmente.

## ARQUITETURA RESTFUL

A *Representational State Transfer* (REST) é um popular estilo de arquitetura de software, comumente utilizado para a criação de web services e na integração de sistemas, segundo Lecheta (2015).

Devmedia (2016) complementa afirmando que o estilo REST determina como deve ser efetuada a Transferência de Estado Representacional, ou *a Representational State Transfer*, em inglês. Em outros termos, significa a representação correspondente a um conjunto de valores caracteriza uma determinada entidade em um determinado instante.

O REST utiliza o protocolo HTTP em seus serviços que respondem às requisições com dados geralmente nos formatos JSON ou XML, servindo de alternativa ao SOAP, de acordo com Ferreira (2015).

O modelo de arquitetura REST possui alguns princípios básicos, elencados por Ferreira (2015): Identificação de recursos, utilização de URIs legíveis, padronização na identificação dos recursos, não utilização do formato desejado da representação do recurso na URI, utilização dos métodos HTTP para manipulação dos recursos (GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS), suporte à diferentes representações e comunicação sem estado, ou *stateless*.

A seguir, na tabela 1 Agrawal (2018) exemplifica o uso padronizado dos métodos HTTP:

Tabela 1: Métodos HTTP no padrão RESTful

|  |  |  |  |
| --- | --- | --- | --- |
| **Nome da rota** | **URL** | **Verbo HTTP** | **Descrição** |
| Index | /blogs | GET | Exibe a lista de todos os blogs |
| New | /blog/new | GET | Mostra um formulário para criar novos blogs |
| Create | /blogs | POST | Adiciona um novo blog no banco de dados, então redireciona |
| Show | /blogs/:id | GET | Mostra as informações sobre um blog |
| Edit | /blogs/:id/edit | GET | Mostra um formulário para edição de um blog existente |
| Update | /blogs/:id | PUT | Atualiza um blog em particular, então redireciona |
| Destroy | /blogs/:id | DELETE | Apaga um blog em particular, então redireciona |

Fonte: Traduzido e adaptado de Agrawal (2018)

Lecheta (2015) conclui: “Então, se um *web service* seguir estes princípios básicos, podemos dizer que, em termos gerais, este é um *web service* RESTful”

O capítulo 3 deste projeto de pesquisa científica apresenta as metodologias de estudo que serão utilizadas no desenvolvimento de uma monografia a respeito do tema aqui proposto.

# DESENVOLVIMENTO

Neste capítulo serão apresentadas as atividades realizadas e os artefatos produzidos durante o desenvolvimento do sistema.

Neste ponto, cabe salientar que a solução proposta é composta de duas aplicações: uma para disponibilizar os recursos e outra para consumir esses recursos. Aqui será apresentado conteúdo referente às duas aplicações, e quando se fizer necessário, será apontado a qual aplicação o conteúdo se refere.

## Levantamento de requisitos

De acordo com Nilcain (2018), o levantamento de requisitos é a etapa inicial do ciclo de desenvolvimento de um software, onde são identificadas as funcionalidades de escopo do projeto.

Durante o desenvolvimento desse trabalho, foram identificados os requisitos funcionais demonstrados a seguir na tabela 2.

Tabela - Requisitos funcionais

|  |  |  |
| --- | --- | --- |
| **Requisitos funcionais** | | |
| **Código** | **Descrição do requisito funcional** | **Detalhamento** |
| RF001 | Possuir aplicação web para administração dos aplicativos cadastrados | Inclusão, alteração e inativação dos aplicativos |
| RF002 | Possuir aplicação web para acesso aos dados dos munícipes | Inclusão e alteração dos dados dos munícipes |
| RF003 | Possuir aplicação web para administração dos usuários do sistema cadastrados | Inclusão, alteração e inativação dos usuários |
| RF004 | Permitir a emissão de relatórios através da aplicação web |  |
| RF005 | Possuir perfis de permissões aos usuários do sistema |  |
| RF006 | Permitir a vinculação de usuários do sistema aos perfis de permissões |  |
| RF007 | Possibilitar a inclusão de novos dados através de API na arquitetura REST |  |
| RF008 | Deverá registrar o aplicativo que realizou alterações ou inclusões de dados |  |
| RF009 | Deverá registrar o usuário que realizou alterações ou inclusões de dados |  |
| RF010 | Permitir a recuperação dos dados de forma individual | Informar ID ou CPF |
| RF011 | Permitir a recuperação dos dados em lote | Informar uma série de ID |
| RF012 | Gerar ID único automaticamente para os munícipes cadastrados |  |
| RF013 | Deverá validar o formato dos dados de entrada | Em conformidade com o tipo armazenado em banco de dados |
| RF014 | Deverá verificar a duplicidade de munícipes no momento da inclusão | Utilizar o CPF como chave |
| RF015 | Permitir somente a inclusão de CPF validado |  |
| RF016 | Deverá validar dados de entrada de acordo com as tabelas base do sistema | CEP |
| RF017 | Deverá armazenar as tentativas de inclusão ou alteração de dados não sucedidas | Tentativas não sucedidas em razão de dados de entrada incorretos ou duplicados |
| RF018 | Possuir função que possibilite a exclusão de dados, em conformidade com a LGPD |  |
| RF019 | Possuir função que possibilite o compartilhamento de dados de mediante autorização do munícipe |  |
| RF020 | Possuir função que receba dos sistemas terceiros a solicitação de exclusão de dados, conforme a LGPD |  |
| RF021 | Possuir função que notifique os sistemas terceiros da solicitação de exclusão de dados, conforme a LGPD |  |
| RF022 | Possuir função de verificação de dados entrada | A função deve retornar se a inclusão foi nem sucedida |

A tabela 3, a seguir, mostra os requisitos não funcionais identificados durante a fase de levantamento de requisitos.

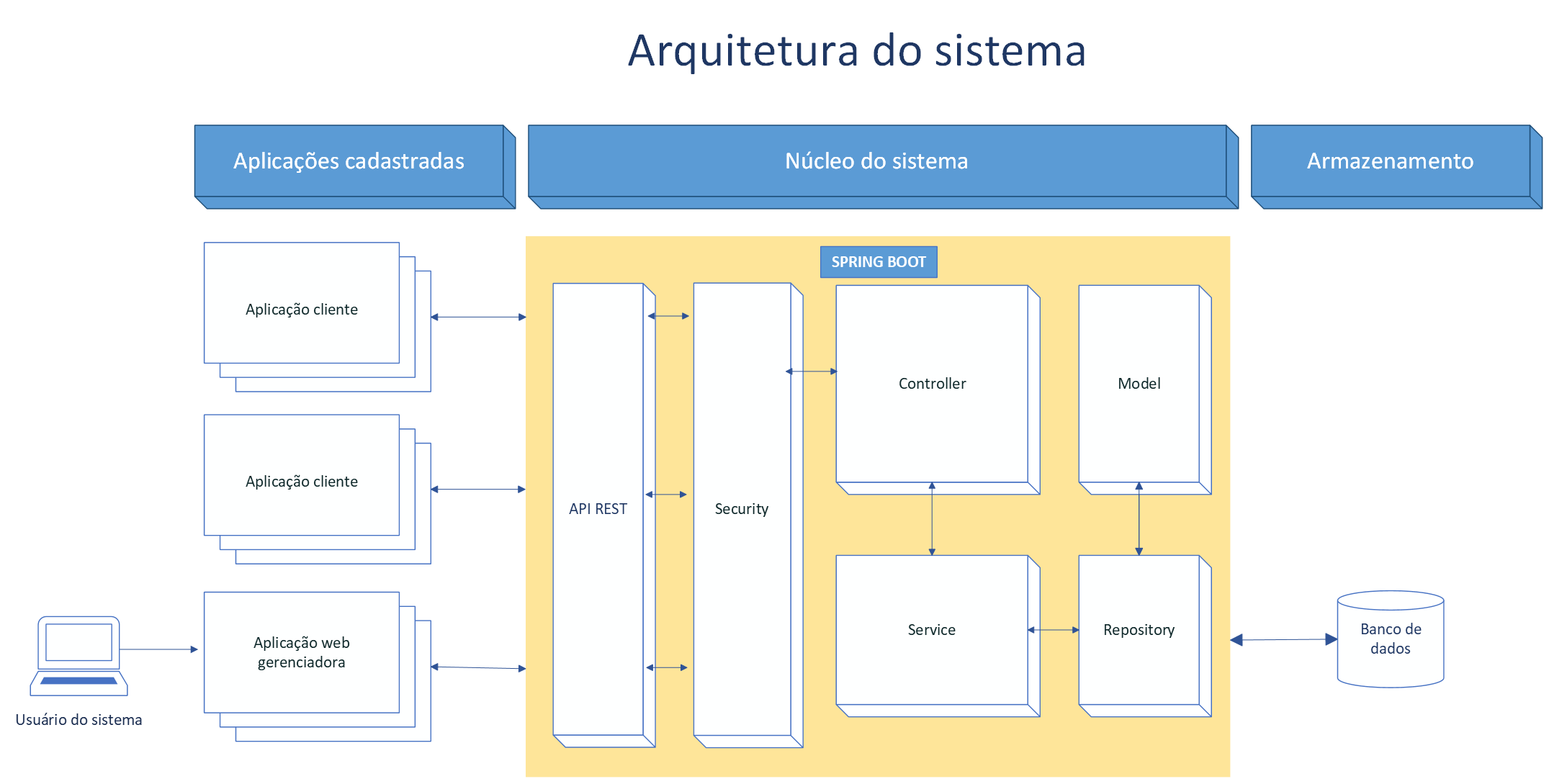
Tabela - Requisitos não funcionais

|  |  |  |
| --- | --- | --- |
| **Requisitos não funcionais** | | |
| **Código** | **Descrição do requisito não funcional** | **Detalhamento** |
| RNF001 | Possuir banco de dados relacional |  |
| RNF002 | Deverá ser um sistema web |  |
| RNF003 | Utilizar o padrão RESTful nas APIs disponibilizadas |  |
| RNF004 | Possuir documentação para a utilização das APIs por parte dos aplicativos cadastrados |  |
| RNF005 | Possuir campos no cadastro do munícipe para conformidade com LGPD |  |
| RNF006 | Deverá atender a legislação vigente |  |
| RNF007 | Não deverá permitir a exclusão de dados | Exceto por solicitação explícita do usuário ou por ordem judicial |
| RNF008 | Possuir acesso somente aos aplicativos cadastrados |  |

## Arquitetura do sistema

A arquitetura da solução proposta neste trabalho foi definida de maneira a ser facilmente desacoplada e independente entre seus próprios componentes. A figura X mostra um esboço dos componentes do sistema e a forma de comunicação entre si.

Figura - Arquitetura do sistema



Fonte: De autoria própria

## Diagrama de caso de uso

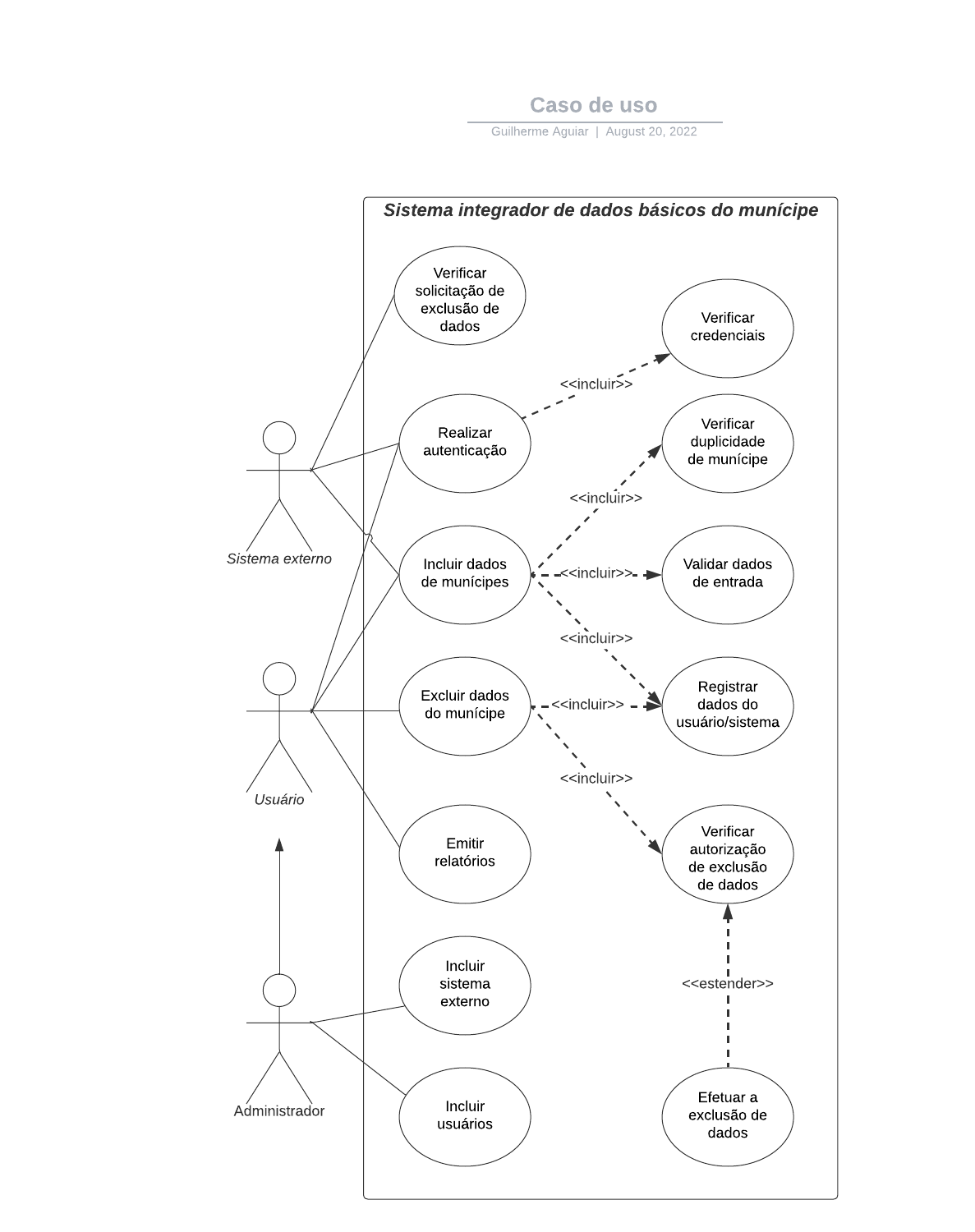
Segundo Pressman (2011), um diagrama UML de caso de uso é uma visão de todos os casos de usos e como eles se relacionam, desta forma, fornecem uma visão macro do sistema.

Pressman (2011, p. 731) também explica o caso de uso em si:

Um caso de uso descreve como um usuário interage com o sistema definindo os passos necessários para atingir um objetivo específico (por exemplo, gravar uma lista de músicas em um CD). Variações na sequência de passos descrevem vários cenários (por exemplo, o que acontece se as músicas da lista não couberem em um CD?).

A figura 3 apresenta o diagrama de caso de uso do sistema.

Figura - Diagrama de caso de uso



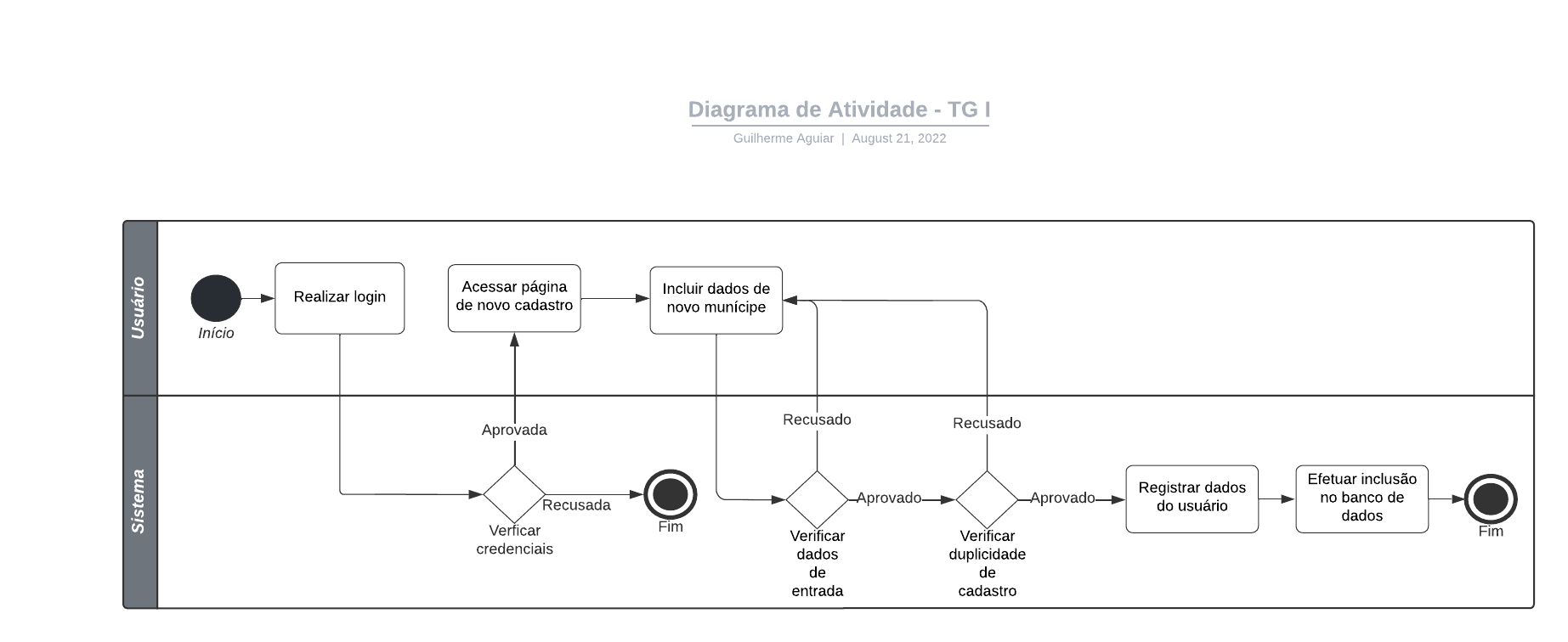
Fonte: De autoria própria

## Diagrama de Atividade

Um diagrama de atividade representa o comportamento de um sistema ou parte dele por meio de um fluxo de controle que percorre as ações executadas pelo sistema, de acordo com Pressman (2011).

A figura 4 mostra o diagrama atividade do processo de inclusão de dados de um novo munícipe a partir da aplicação de gerenciamento.

Figura - Diagrama de atividade: Inclusão de novo munícipe



Fonte: De autoria própria

## Modelagem do banco de dados

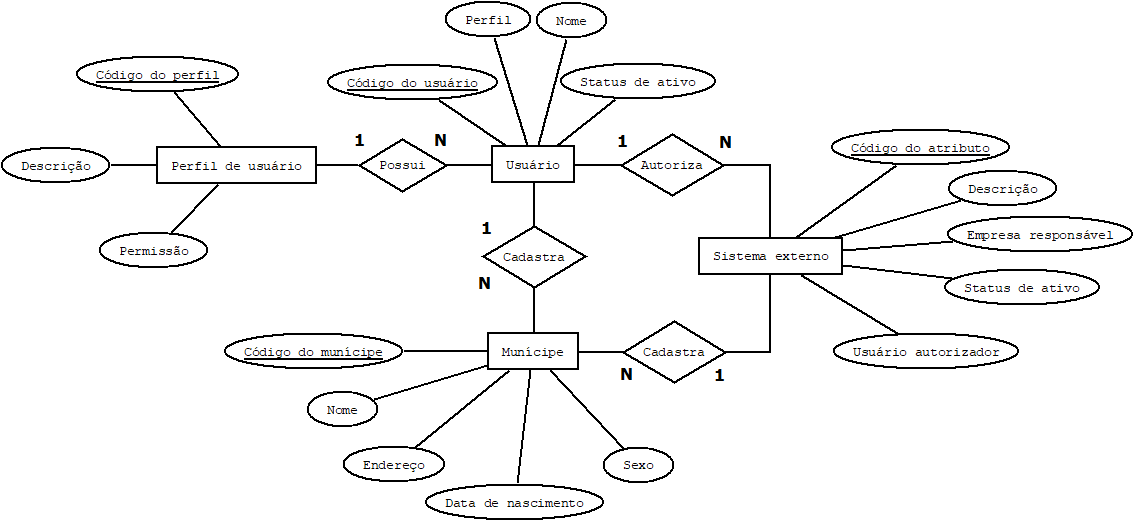
### MODELO CONCEITUAL

Fernigrini (2021) explana que um modelo conceitual normalmente abrange somente os principais conceitos (entidades) para armazenar as informações e os relacionamentos que existem entre as entidades. Este modelo é uma etapa inicial para a criação de um banco de dados.

Ainda de acordo com Fernigrini (2021), a especificação das entidades em alto nível, usando nomes comerciais no lugar de nomes técnicos, permite o bom entendimento do público geral, como gestores de negócios e usuários.

A figura 5 possui a representação do modelo conceitual utilizado como base para a criação do banco de dados do sistema.

Figura - Modelo conceitual



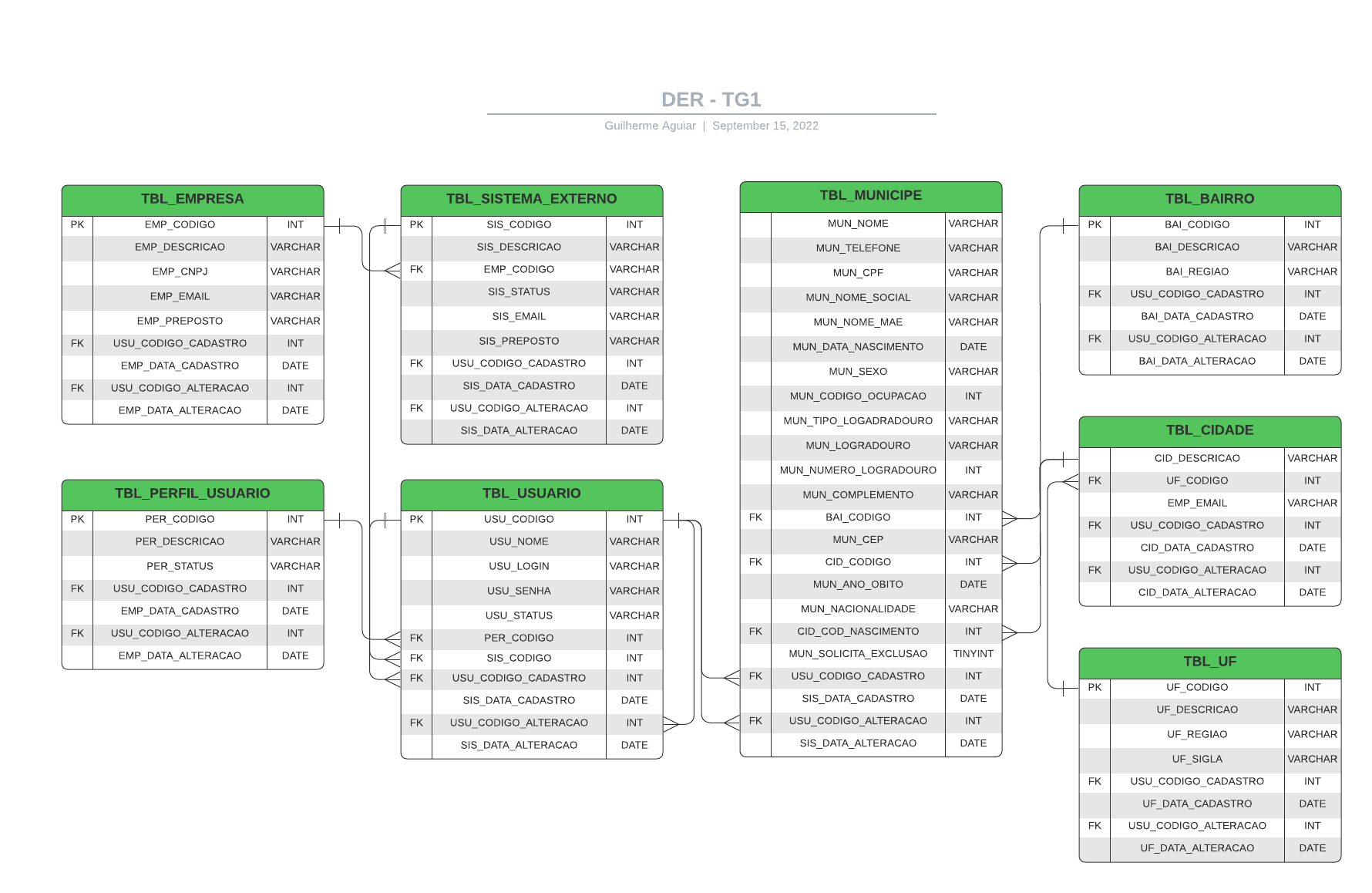
Fonte: De autoria própria

### MODELO FÍSICO

Segundo Fernigrini (2021), um modelo de dados físico é derivado de um modelo lógico para determinado SGBD. A grande diferença é que aqui são utilizados os nomes das tabelas e colunas ao invés das entidades e atributos. Além disso, são fornecidos os tipos dos dados, as restrições e tabelas adicionais. Isso possibilita ajustar-se aos limites e convenções do banco de dados pretendido.

A figura 6 possui a representação do modelo físico do banco de dados.

Figura - Modelo físico



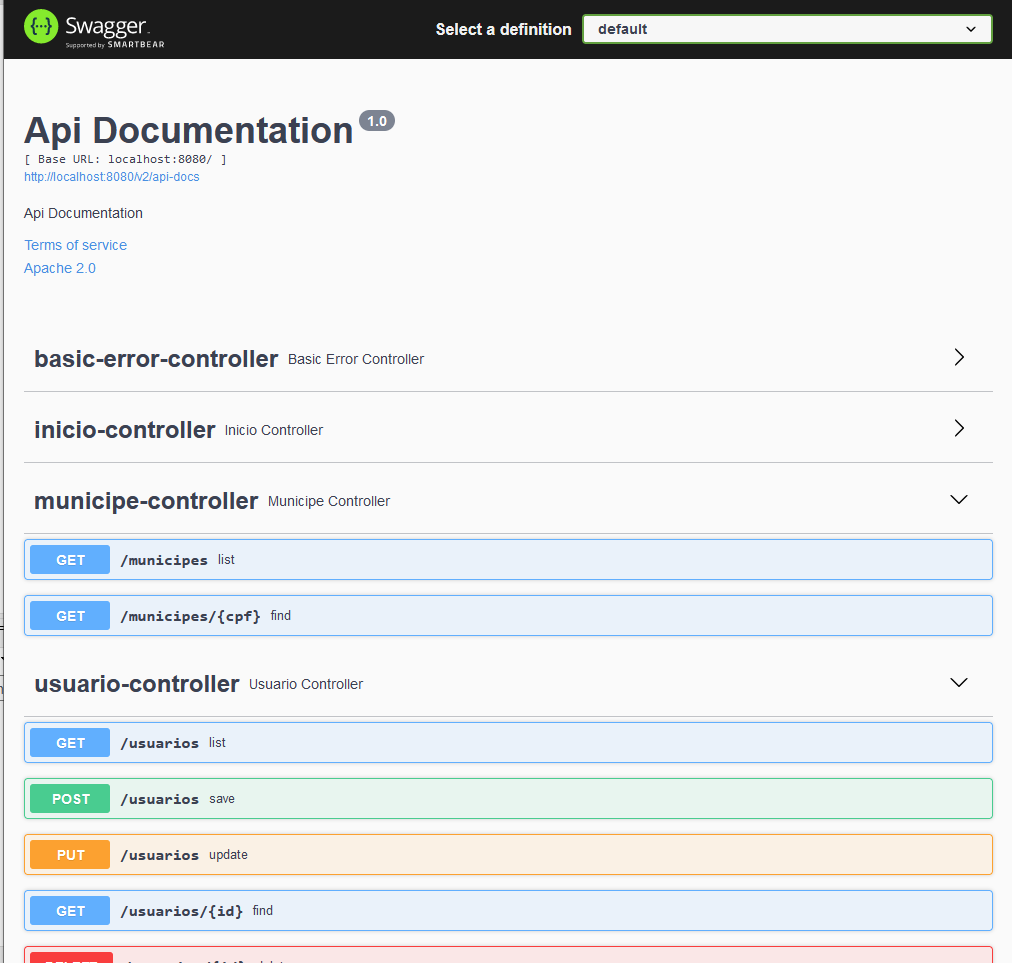
Fonte: De autoria própria

# Resultados obtidos

Aqui serão apresentados os resultados obtidos nas fases de desenvolvimento concluídas.

A utilização do framework Swagger aliado ao Spring Boot resultou na disponibilização de uma página dedicada à documentação da API criada, conforme é observado na figura 7.

Figura - Swagger

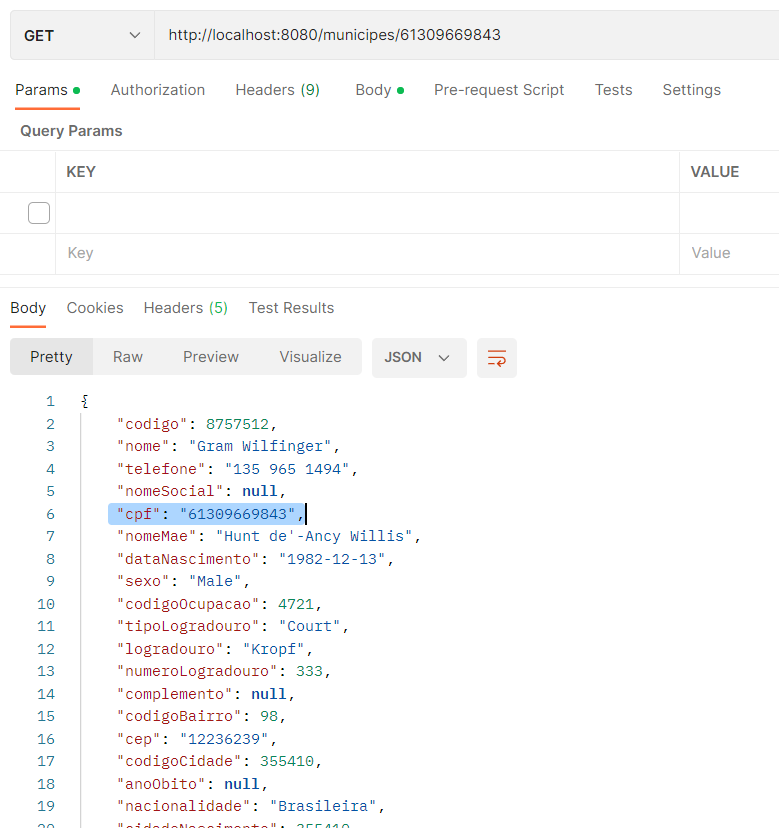


Fonte: De autoria própria

A disponibilização da documentação atende ao requisito não funcional RNF007.

A figura 8 apresenta o resultado de uma requisição GET realizada através do endpoint “/municipes/{cpf}” passando como parâmetro um CPF cadastrado. Foi utilizada a ferramenta Postman para o envio da requisição.

Figura - Resposta da requisição GET



Fonte: De autoria própria

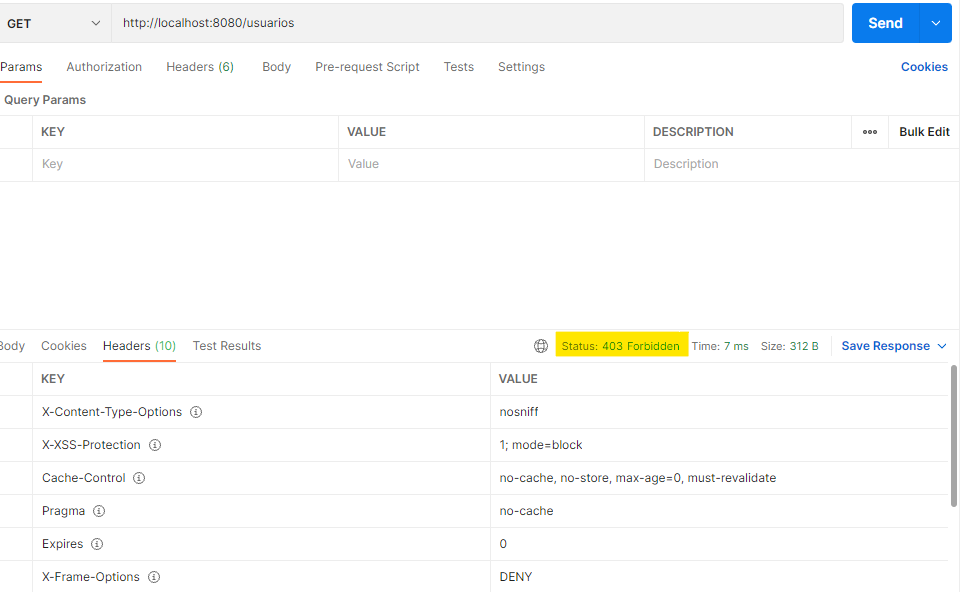
## Implementação do Json Web Token

Para assegurar os requisitos de segurança, foi implementada a autenticação via JWT (*Json Wen Token*). A tabela abaixo relaciona os *endpoints* e métodos autorizados para cada um dos perfis de acesso utilizado na plataforma.

Tabela

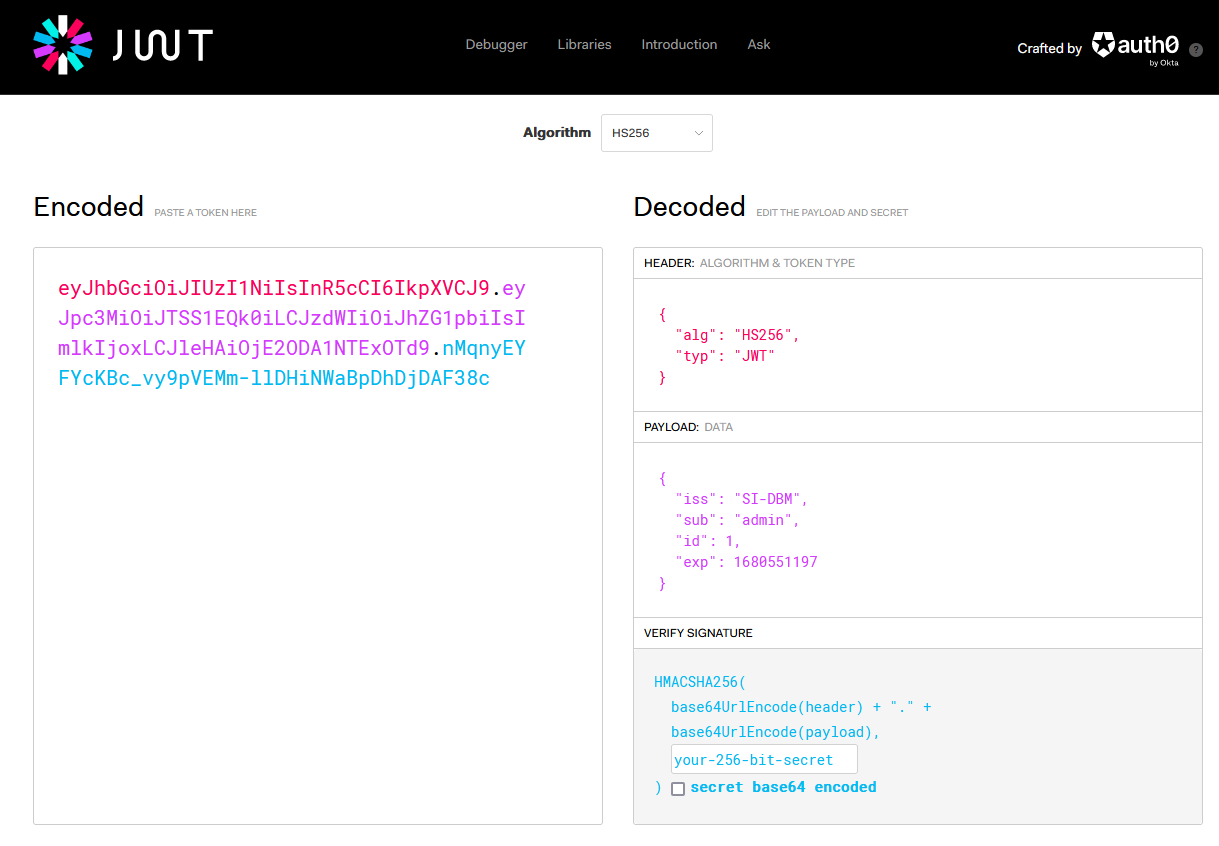
A figura 9 mostra uma tentativa de acesso ao endpoint /usuários sem a devida autenticação, que retorna a requisição com o *status 403 Forbidden,* exibido no destaque.

Figura - Resposta da requisição GET



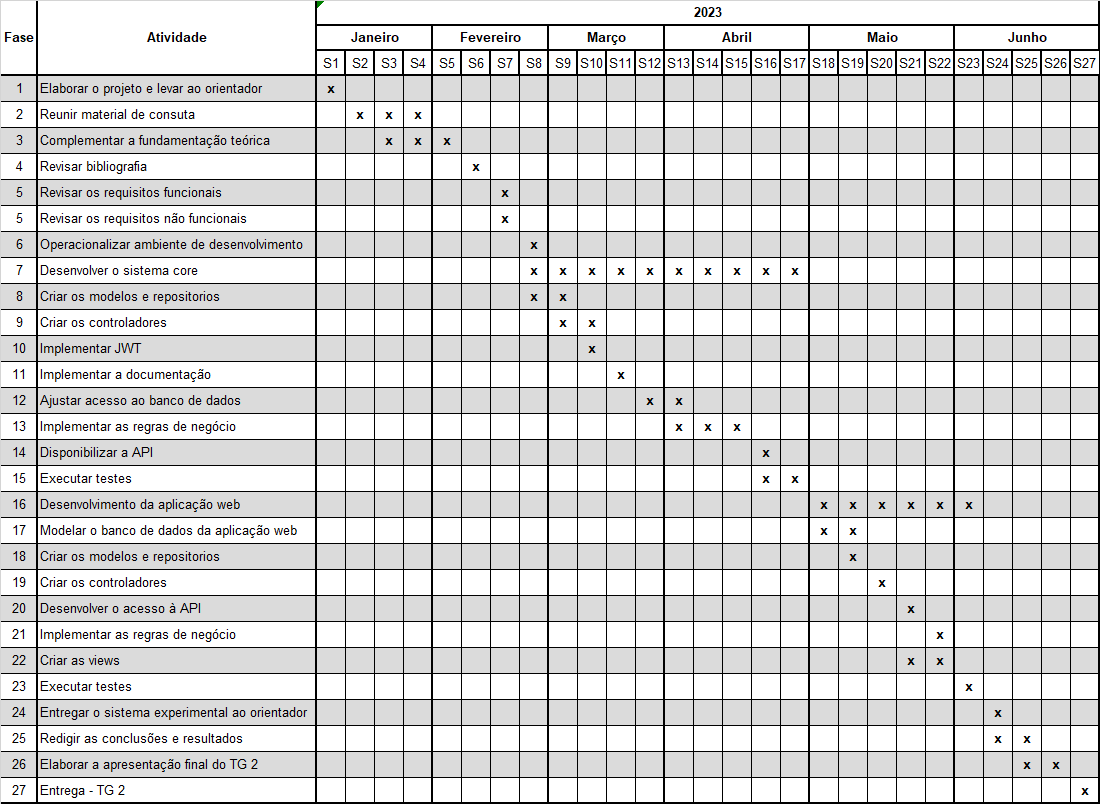
Fonte: De autoria própria

A utilização das bibliotecas do Spring Security permite a geração do JWT e entrega ao cliente autenticado. A figura 10



# CRONOGRAMA

Tabela - Cronograma



Fonte: De autoria própria

# REFERÊNCIAS

AGRAWAL, Shubhangi Raj. **The 7 RESTful routes!** 2018. Disponível em: https://medium.com/@shubhangirajagrawal/the-7-restful-routes-a8e84201f206. Acesso em: 22 maio 2022.

APACHE SOFTWARE FOUNDATION. **Trilha do Aprendizado do Java EE e Java Web**. Disponível em: https://netbeans.apache.org/kb/docs/java-ee\_pt\_BR.html. Acesso em: 22 maio 2022.

BARBIERI, Carlos. **Governança de Dados**: práticas, conceitos e novos caminhos. Rio de Janeiro: Alta Books, 2020. 288 p.

DANIEL, Andrew et al. **Exposing and Managing Enterprise Services with IBM API Management.** Lindon: Vervanté, 2014. 266 p.

DEVMEDIA. **Introdução a web services RESTful**. 2016. Disponível em: https://www.devmedia.com.br/introducao-a-web-services-restful/37387. Acesso em: 26 maio 2022.

DUBOIS, Paul. **MySQL**: the definitive guide to using, programming, and administering mysql 5.0 and 5.1. 4. ed. Reino Unido: Addison-Wesley, 2008. 1224 p. (Developer's Library).

FERREIRA, Rodrigo. **REST:** princípios e boas práticas. Princípios e boas práticas. 2017. Disponível em: https://www.alura.com.br/artigos/rest-principios-e-boas-praticas. Acesso em: 29 maio 2022.

KAINULAINEN, Petri. **Spring Data**. Birmingham: Packt Publishing, 2012. 160 p.

LECHETA, Ricardo. **Web services RESTful**: aprenda a criar web services restful em java na nuvem do google. São Paulo: Novatec Editora, 2015. 432 p.

MARIADB. **About MariaDB Server**. 2022. Disponível em: https://mariadb.org/about/. Acesso em: 14 nov. 2022.

MENDES, Douglas Rocha. **Programação Java com Ênfase em Orientação a Objetos**. São Paulo: Novatec Editora, 2009.

MILANI, André. **MySQL:** guia do programador. São Paulo: Novatec Editora, 2007. 400 p.

NILCAIN, Mário. **Como realizar o Levantamento de Requisitos no desenvolvimento de software.** 2018. Disponível em: https://blog.cedrotech.com/levantamento-de-requisitos-e-desenvolvimento-de-softwares. Acesso em: 14 nov. 2022.

PHALTANKAR, Amit. **What is JPA, Spring Data and Spring Data JPA.** Disponível em: https://www.amitph.com/jpa-and-spring-data-jpa/. Acesso em: 14 maio 2022.

POSTMAN. **About Postman**. Disponível em: https://www.postman.com/about/. Acesso em: 03 abr. 2023.

PRESSMAN, Roger S.. **Engenharia de Software**: uma abordagem profissional. 7. ed. Porto Alegre: Amgh Editora Ltda., 2011. 780 p.

PROMPT SOFTTECH. **How to Use Swagger Tool for API Documentation?** 2019. Disponível em: https://www.promptsoftech.com/blog/how-to-use-swagger-tool-for-api-documentation/. Acesso em: 14 nov. 2022.

RÊGO, Bergson Lopes. **Gestão e Governança de Dados:** promovendo dados como ativo de valor nas empresas. Rio de Janeiro: Brasport, 2013. 312 p.

SANTANA, Eduardo Felipe Zambom. **Back-end Java:** microsserviços, spring boot e kubernetes. São Paulo: Casa do Código, 2021. 155 p.

SPRING. **2. Introduction to the Spring Framework:** part i. overview of spring framework. Part I. Overview of Spring Framework. Disponível em: https://docs.spring.io/spring-framework/docs/4.2.1.RELEASE/spring-framework-reference/html/overview.html. Acesso em: 29 maio 2022

\_\_\_\_\_\_\_ **Spring Data.** 2021. Disponível em: https://spring.io/projects/spring-data. Acesso em: 26 maio 2022.

SWAGGER. **About Swagger**: the history behind swagger. The History Behind Swagger. 2022. Disponível em: https://swagger.io/about/. Acesso em: 14 nov. 2022.

FERNIGRINI, Lisandro. **What Are Conceptual, Logical, and Physical Data Models?** 2021. Disponível em: https://vertabelo.com/blog/conceptual-logical-physical-data-model/. Acesso em: 14 nov. 2022.

WEISSMANN, Henrique Lobo. **Vire o jogo com Spring Framework**. São Paulo: Casa do Código, 2014. 296 p.

WITTIG, Andreas; WITTIG, Michael. **Amazon Web Services em ação**. São Paulo: Novatec Editora, 2015. 512 p.