

SOCIEDADE EDUCACIONAL PINHALZINHO

HORUS FACULDADES

Adilson Amorim Gonçalves Junior

**DESENVOLVIMENTO DE UM PROTÓTIPO DE SISTEMA WEB PARA O
GERENCIAMENTO DE LEITOS E PROCEDIMENTOS CIRÚRGICOS EM
HOSPITAIS**

Pinhalzinho/SC

2023

ADILSON AMORIM GONÇALVES JUNIOR

**DESENVOLVIMENTO DE UM PROTÓTIPO DE SISTEMA WEB PARA O
GERENCIAMENTO DE LEITOS E PROCEDIMENTOS CIRÚRGICOS EM
HOSPITAIS**

Trabalho de Conclusão de Curso apresentado à Horus
Faculdades, como parte dos requisitos para obtenção do grau
de Bacharel em Sistemas de Informação

Orientador(a): Prof. Esp. Ricardo Jeferson Hendges

Pinhalzinho/SC

2023

AGRADECIMENTO

Antes de tudo gostaria de agradecer a todos meus familiares e amigos, por todo auxílio e que, com absoluta certeza, tem uma contribuição para conclusão desta pesquisa.

Ao meu pai, Adilson Amorim Gonçalves, que mesmo longe, sempre se manteve presente quando o assunto era relacionado ao meu crescimento profissional.

A minha mãe, Leila Aparecida dos Santos, que também está longe, se manteve presente e auxiliando, principalmente, no desenvolvimento desta pesquisa.

A minha namorada, Milena Triches Knop, pelo apoio e companheirismo, estando junto comigo desde o início da minha trajetória na instituição.

Aos professores, colegas de curso e a instituição Horus Faculdades, pelo apoio e comprometimento, buscando auxiliar na melhor forma para o desenvolvimento profissional.

“A primeira regra de qualquer tecnologia utilizada nos negócios é que a automação aplicada a uma operação eficiente aumentará a eficiência. A segunda é que a automação de uma operação ineficiente aumentará a ineficiência”.

Bill Gates.

RESUMO

A pandemia do COVID-19 designou diversas mudanças na saúde pública e privada, como na segurança dos atendimentos hospitalares (BETHA, 2021). Diante disso, uma das mudanças com mais notoriedade foi relacionada à interrupção de serviços essenciais. Onde estudos realizados, apontava que diversos países com diversas rendas diferentes registravam uma interrupção de cerca de 23% nas cirurgias de emergência e de 59% nas cirurgias eletivas, causando um acúmulo futuro (OPAS, 2022). Em meio a pandemia, o Brasil registrou também um aumento expressivo no número total de leitos, onde no período de dezembro de 2019 até abril de 2020, passando de 46.045 para 60.265, representando um acréscimo de 23,59% (CABRAL et al, 2020). Em meio a este cenário complexo, o presente projeto propõe-se a desenvolver um protótipo de software de gestão hospitalar, com o objetivo de otimizar e transparecer os processos relacionados às cirurgias, tanto eletivas quanto emergenciais e a gestão de leitos de forma eficaz, utilizando tecnologias como Vue, Node e Postgres como base para desenvolvimento do software.

Palavras-chave: Pandemia do COVID-19. Interrupção de serviços essenciais. Gestão hospitalar.

ABSTRACT

The COVID-19 pandemic has brought about various changes in both public and private healthcare, particularly in the safety of hospital services (BETHA, 2021). One of the most notable changes was the interruption of essential services. Studies indicated that many countries, regardless of income levels, experienced a 23% disruption in emergency surgeries and a 59% disruption in elective surgeries, leading to potential future backlogs (OPAS, 2022). Amid the pandemic, Brazil also saw a significant increase in the total number of hospital beds. From December 2019 to April 2020, the count rose from 46,045 to 60,265, representing a 23.59% increase (CABRAL et al., 2020). In the midst of this complex scenario, this project aims to develop a prototype of hospital management software to optimize and streamline processes related to both elective and emergency surgeries, as well as effective bed management. Technologies such as Vue, Node, and Postgres will serve as the foundation for the software's development.

Keywords: COVID-19 pandemic, Interruption of essential services, Hospital management.

LISTA DE FIGURAS

Figura 01: Fluxograma de Unidades de emergência.....	20
Figura 02: Fluxograma de Hospitais.....	20
Figura 03: Número de leitos por mil habitantes.....	22
Figura 04: Regiões de Santa Catarina e a matriz de risco.....	23
Figura 05: Total de cirurgias realizadas no Brasil no período de 2016 até 2020.....	24
Figura 06: Regiões do Brasil e seu acúmulo de cirurgias em 2020.....	25
Figura 07: Diagrama de Casos de uso do protótipo.....	33
Figura 08: Diagrama de Classes de uso do protótipo.....	34
Figura 09: Diagrama de Entidade e Relacionamento do protótipo.....	35
Figura 10: Diagrama de Sequência - Consultar.....	37
Figura 11: Diagrama de Sequência - Parte 1 Inserir.....	38
Figura 12: Diagrama de Sequência - Parte 2 Inserir.....	39
Figura 13: Diagrama de Sequência - Parte 1 Editar utilizando o Inserir.....	40
Figura 14: Diagrama de Sequência - Parte 2 Editar.....	41
Figura 15: Diagrama de Sequência - Inativar.....	42
Figura 16: Diagrama de Sequência (Manter Usuário).....	43
Figura 17: Diagrama de Atividade - Criar um procedimento cirúrgico.....	45
Figura 18: Diagrama de Atividade - Atribuir leito.....	46
Figura 19: Diagrama de Máquina de Estados - Procedimentos Cirúrgicos.....	47
Figura 20: Exemplificação da estrutura MVC.....	49
Figura 21: Comparação de uso de Contêiner e Máquinas Virtuais.....	53
Figura 22: Diferença do funcionamento de um servidor Node.js para o Tradicional.....	58
Figura 23: Exemplificação de uso do Node.js.....	59
Figura 24: Exemplificação de uso do JWT.....	61
Figura 25: Prototipação do projeto realizada no Figma.....	67
Figura 26: Repositório do protótipo no GitHub.....	68
Figura 27: Dockerfile Postgres presente no protótipo.....	70
Figura 28: Uso do docker compose entre a API e o Banco de Dados no projeto.....	70
Figura 29: Docker-compose no protótipo.....	71
Figura 30: Exemplificação do uso do Postgres: criar tabelas.....	72
Figura 31: Exemplificação do uso do Postgres: criar tabelas.....	73
Figura 32: Estrutura das pastas no Backend do protótipo.....	74
Figura 33: Exemplo de um POST (inserção) na API - Services.....	75
Figura 34: Exemplo de um POST (inserção) na API - Controller.....	75
Figura 35: Exemplo de um POST (inserção) na API - Routes.....	76
Figura 36: Exemplo de documentação usando swagger.....	77
Figura 37: Criação do projeto em Vue.....	78

Figura 38: Exemplo de configuração ao criar um projeto Vue.....	79
Figura 39: Instalação do Vuetify no projeto.....	80
Figura 40: Configuração pré-configuradas do Vuetify.....	80
Figura 41: Configuração pré-configuradas do Vuetify.....	81
Figura 42: Estrutura das pastas no Frontend do protótipo.....	82
Figura 43: Exemplo de uso do Vue Router.....	83
Figura 44: Exemplo de uso do Vue Router.....	84
Figura 45: Protótipo de Tela de Login.....	85
Figura 46: Exemplificação do cadastro de Alas no protótipo.....	85
Figura 47: Gerenciamento de leitos no protótipo.....	86
Figura 48: Aplicação do Kanban no protótipo.....	87

LISTA DE ABREVIACÕES

API	<i>Application Programming Interface</i>
BD	<i>Banco de Dados</i>
COREN	<i>Conselho Regional de Enfermagem</i>
CSS	<i>Cascading Style Sheets</i>
DataSUS	<i>Departamento de Informática do Sistema Único de Saúde</i>
DIEESE	<i>Departamento Intersindical de Estatística e Estudos Socioeconômicos</i>
DOM	<i>Document Object Model</i>
EC 95	<i>Emenda Constitucional n° 95</i>
GISAID	<i>Global Initiative on Sharing All Influenza Data</i>
HOST	<i>Hosting</i>
HS256	<i>HMAC SHA-256</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IBM	<i>International Business Machines Corporation</i>
IDE	<i>Integrated Development Environment</i>
JS	<i>JavaScript</i>
JSON	<i>JavaScript Object Notation</i>
JWT	<i>Json Web Token</i>
MVC	<i>Model-View-Controller</i>
OMS	<i>Organização Mundial da Saúde</i>
SARS	<i>Síndrome respiratória aguda grave</i>
SQL	<i>Structured Query Language</i>

<i>SRC</i>	<i>Source</i>
<i>SUS</i>	<i>Sistema Único de Saúde</i>
<i>UF</i>	<i>União Federativa</i>
<i>UML</i>	<i>Unified Modeling Language</i>
<i>URL</i>	<i>Uniform Resource Locator</i>
<i>UTI</i>	<i>Unidade de Terapia Intensiva</i>
<i>VPN</i>	<i>Virtual Private Network</i>
<i>WEB</i>	<i>World Wide Web</i>
<i>XSS</i>	<i>Cross-site Scripting</i>
<i>YAML</i>	<i>Yet Another Markup Language</i>

SUMÁRIO

1 INTRODUÇÃO.....	14
2 OBJETIVOS DA PESQUISA.....	15
2.1 OBJETIVO GERAL.....	15
2.2 OBJETIVOS ESPECÍFICOS.....	15
3 REFERENCIAL TEÓRICO.....	16
3.1 ORIGEM E EVOLUÇÃO DO SISTEMA ÚNICO DE SAÚDE.....	16
3.1.1 Desafios do Sistema Único de Saúde.....	17
3.2 ORIGEM E CONCEITOS DA PANDEMIA DO COVID-19.....	17
3.2.1 Grupos de riscos e sua gravidade.....	19
3.2.2 Campanhas de conscientização COVID-19.....	19
3.2.3 Métodos preventivos no Brasil.....	20
3.2.4 Interferência direta da pandemia aos hospitais e aos profissionais de saúde.....	20
3.3 ESCASSEZ DE LEITOS POR HABITANTES E SUA LOTAÇÃO EM TEMPOS DE PANDEMIA.....	22
3.4 CONSEQUÊNCIAS DA PANDEMIA RELACIONADAS À PRORROGAÇÃO DE CIRURGIAS.....	25
4 PROCEDIMENTOS METODOLÓGICOS.....	27
5 ANÁLISE DOS REQUISITOS.....	28
5.1 LEVANTAMENTO E VALIDAÇÃO DOS REQUISITOS.....	28
5.2 DESCRIÇÃO DO PROTÓTIPO DO SISTEMA.....	29
5.3 REQUISITOS FUNCIONAIS.....	30
5.4 REQUISITOS NÃO FUNCIONAIS.....	31
5.5 UML.....	32
5.5.1 Diagrama de Casos de Uso.....	33
5.5.2 Diagrama de Classes.....	35
5.5.3 Diagrama de Entidade e Relacionamento.....	36
5.5.4 Diagrama de Sequência.....	37
5.5.4.1 Diagrama: manterProcedimentosCirurgicos.....	37
5.5.4.1.1 Operação: Consultar.....	38
5.5.4.1.2 Operação: Inserir.....	39
5.5.4.1.3 Operação: Editar.....	40
5.5.4.1.4 Operação: Inativar.....	42
5.5.4.2 Diagrama: manterUsuario.....	43
5.5.5 Diagrama de Atividade.....	45
5.5.5.1 Criar um procedimento cirúrgico.....	45
5.5.5.2 Criar uma ocupação de leitos.....	46
5.5.6 Diagrama de Máquina de Estados.....	48

6 FERRAMENTAS, SISTEMAS, TECNOLOGIAS E METODOLOGIAS ENVOLVIDAS DO PROTÓTIPO.....	49
6.1 SISTEMAS E APLICAÇÕES.....	49
6.2 ARQUITETURA.....	50
6.3 GIT E GITHUB.....	51
6.4 GITHUB COPILOT.....	51
6.5 VISUAL STUDIO CODE.....	52
6.6 DOCKER.....	53
6.6.1 Container e Imagem.....	53
6.6.2 Docker Compose.....	55
6.7 BANCO DE DADOS.....	55
6.7.1 PostgreSQL.....	56
6.8 JAVASCRIPT.....	57
6.8.1 Node.js.....	58
6.8.2 Express.js.....	60
6.9 BCRYPT.....	61
6.10 JSON WEB TOKEN.....	62
6.11 POSTMAN.....	63
6.12 SWAGGER.....	63
6.13 VUE.JS.....	64
6.14 VUETIFY.....	65
6.15 VUE ROUTER.....	65
6.16 AXIOS.....	66
7 DESENVOLVIMENTO DO PROJETO.....	67
7.1 PLANEJAMENTO E ORGANIZAÇÃO.....	67
7.1.1 Prototipação no Figma.....	67
7.1.2 Versionamento de Código.....	69
7.2 BACKEND.....	70
7.2.1 Configurações do Docker.....	70
7.2.2 Estrutura do Banco de Dados.....	73
7.2.3 Desenvolvimento da API.....	74
7.2.4 Documentação utilizando Swagger.....	77
7.3 FRONTEND.....	78
7.3.1 Instalação do Vue.....	79
7.3.2 Configuração do Vuetify.....	80
7.3.3 Estruturas de Pastas.....	82
7.3.4 Interfaces.....	85
8 CRONOGRAMA.....	89
9 CONSIDERAÇÕES FINAIS.....	90
9.1 TRABALHOS FUTUROS.....	91
REFERÊNCIAS BIBLIOGRÁFICAS.....	92

1 INTRODUÇÃO

A origem do Sistema Único de Saúde ou popularmente conhecido como “SUS”, foi de forma gradual sendo a partir da Reforma Sanitária Brasileira, na década de 70, sendo considerado um momento histórico para a saúde, inclusive mundialmente, lutando diretamente contra a ditadura militar, fornecendo acesso e oportunidade de cuidados à saúde à toda população (FERNANDO et al, 2019).

Com ênfase em melhorar a expectativa de vida da população e a partir de todas as ações realizadas desde sua existência, o Sistema Único de Saúde é considerado exemplo para todo o mundo, principalmente quando relacionado aos sistemas de transplantes, hemocentros e na distribuição de medicamentos para controle de doenças crônicas, como hipertensão e diabetes (VICTORA et al., 2011; IBGE, 2018).

Contando, desde sua criação e estruturação, que permitiu e permite o acesso a diversos brasileiros, recentemente vivenciamos uma pandemia, a qual estudos provenientes do projeto Monitora Covid-19 da Fiocruz, levantou-se que houve, durante um ano e meio, cerca de 1.7 milhões de internações a menos, que resulta em 9,9% a menos, o que conseqüentemente atrasou-se no quesito de cirurgias, sendo visto como uma das sequelas provenientes da Covid, esta fila que ficou para trás (FIOCRUZ, 2021).

Por outro lado, ainda na pandemia, a exemplo de Santa Catarina, em 2022, foi decretado situação de emergência contendo cerca de 97.7% dos leitos ocupados, sendo que nas regiões de: Foz do Rio Itajaí, Meio Oeste e Serra Catarinense e a região Grande Oeste lidava com 100% da ocupação e com filas de esperas (MÁXIMO, 2022).

Portanto, a presente pesquisa visa analisar, conceituar, compreender e desenvolver um sistema de gerenciamento de leitos e procedimentos cirúrgicos em hospitais. Justifica-se a pesquisa pois vivenciamos recentemente uma pandemia, a qual houve uma superlotação nos hospitais devido ao ápice do COVID-19 em que encadeou uma alta demanda por atendimentos, ocasionando a ampliação de leitos, além da postergação de procedimentos cirúrgicos, o que acaba evidenciando ainda mais a importância de se ter um sistema que possibilite o monitoramento e a otimização do uso desses recursos.

2 OBJETIVOS DA PESQUISA

Neste tópico, delinearemos os objetivos da pesquisa, os quais são fundamentais para orientar o desenvolvimento e a documentação do protótipo de sistema proposto. O escopo geral da pesquisa visa abordar as necessidades de gestão e controle de leitos e procedimentos cirúrgicos em ambientes hospitalares, considerando tanto aspectos históricos quanto às repercussões causadas pela pandemia.

2.1 OBJETIVO GERAL

Desenvolver e documentar um protótipo de sistema que atenda às necessidades de gestão e controle de leitos e procedimentos cirúrgicos em ambientes hospitalares, por meio de análises dos aspectos históricos e das consequências causadas pela pandemia.

2.2 OBJETIVOS ESPECÍFICOS

Estudar conceitos referentes ao sistema hospitalar e também referente ao Sistema Único de Saúde (SUS), afins de compreensão do contexto e as características da saúde brasileira no período pós pandêmico;

Realizar um levantamento dos principais aspectos relacionados à gestão de leitos e procedimentos cirúrgicos em ambientes hospitalares no âmbito do SUS;

Identificar os modelos de gestão e controle mais adequados visando otimizar o uso de leitos e procedimentos cirúrgicos;

Estudar e identificar as melhores tecnologias do mercado para desenvolvimento web;

Desenvolver um protótipo de sistema de gestão e controle de procedimentos cirúrgicos e leitos que atenda às necessidades identificadas e que seja facilmente integrado ao sistema hospitalar.

3 REFERENCIAL TEÓRICO

Para a realização do referencial teórico foram utilizadas referências de autores que abordam sobre a saúde no Brasil, suas mudanças decorrentes da pandemia, além de temas relacionados que possam auxiliar na elaboração desse projeto. Os principais tópicos levantados foram a concepção do Sistema Único de Saúde, origem da pandemia, consequências da pandemia em relação a escassez de leitos e a prorrogação das cirurgias.

3.1 ORIGEM E EVOLUÇÃO DO SISTEMA ÚNICO DE SAÚDE

Sendo considerado, por grande parte, o maior sistema público de saúde do mundo, o Sistema Único de Saúde (SUS) garante acesso total e gratuito para toda a população brasileira, desde atendimentos como avaliar a pressão arterial até mesmo ao transplante de órgãos (Ministério da Saúde, 2020). Onde antes de sua criação, o acesso à saúde era destinado diretamente aos contribuintes da Previdência Social, formando uma saúde centralizada e constituída de forma federal (GOV, 2020).

Atualmente com mais de 32 anos, o Sistema Único de Saúde teve sua criação na década de 70, em razão do cidadão, lutando contra a ditadura militar e visando melhorar a qualidade de vida da população brasileira, que, nos dias atuais, beneficia mais de 180 milhões de brasileiros, contribuindo diretamente em campanhas de vacinação, ações relacionadas à vigilância sanitária, como fiscalização de alimentos e o registro de medicamentos, além de, em um geral, realizar mais de dois bilhões de atendimentos em geral (FERNANDO et al, 2019; FIOCRUZ, 2020).

3.1.1 Desafios do Sistema Único de Saúde

Os desafios enfrentados pelo Sistema Único de Saúde são bem visíveis, onde entra em pauta, principalmente a força de trabalho, a gestão e controle do sistema e também sua administração são pontos levantados por especialistas, além de, claro, a questão financeira (FIOCRUZ, 2020). Juntamente nesta questão financeira, está em vigor o seu processo constante de retirada de recursos, em especial desde 2016, com a aprovação da EC 95, que congelou investimentos em saúde até 2036 (SAÚDE, Conselho, 2021).

Por outro lado, ocorre também, o surgimento de desafios inesperados, como o caso da pandemia, que deixou como legado um aumento da demanda por UTIs, respiradores e outros equipamentos médicos, os quais poderão ser requisitados devido à falta de recursos para sua manutenção (SAÚDE, Ministério, 2020).

Neste caso da Pandemia, ressaltou pontos que também necessitam de atenção direta, como as desigualdades na distribuição de recursos de saúde e a necessidade de expansão da oferta de leitos, com destaque para a necessidade de leitos de UTI (VARGAS, Tatiane, 2022).

Ainda de acordo com a pesquisadora Margareth Portela, com o grande crescimento no número de óbitos em tempos de Pandemia, o Brasil começou a expandir a capacidade de leitos. Ela citou ainda, a disputa global por equipamentos, dando como exemplo o colapso do sistema de saúde em Manaus, na primeira grande onda da Covid-19 (VARGAS, Tatiane, 2022).

3.2 ORIGEM E CONCEITOS DA PANDEMIA DO COVID-19

A pandemia que vivenciamos recentemente foi marcada por uma doença altamente contagiosa, de propagação rápida, sendo causada pelo coronavírus (SARS-CoV-2) e que acaba afetando a saúde e também a economia da população mundial, sendo diretamente ou

indiretamente. Seu surgimento ocorreu em dezembro de 2019, a partir do anúncio da Organização Mundial da Saúde (OMS), até então, seus primeiros casos eram de pneumonia, sendo causados por um agente ainda desconhecido, em Wuhan, na China. Em janeiro de 2020 a doença começou a se propagar rapidamente pelo mundo, de acordo com o levantamento de dados internacional realizado pela *Global Initiative on Sharing All Influenza Data* (GISAID) (BRITO et. al, 2020).

Conhecido anteriormente, nos anos de 2002 e 2003 durante a epidemia de SARS sendo causada por outro coronavírus, onde naquela época se manteve restrita somente a alguns países, como: Canadá, China e Estados Unidos. O novo coronavírus (SARS-CoV), até então batizado como SARS-CoV-2, é responsável pela rápida propagação e disseminação da doença em nível mundial, sendo a China o primeiro país a reportar a doença (BRITO et. al, 2020).

Com alta taxa de disseminação do COVID-19 mundialmente, o primeiro caso no Brasil foi confirmado em Fevereiro de 2020, no estado de São Paulo. No mês de Março já haviam diversos casos, incluindo a notificação do Ministério da Saúde quanto à pandemia. Neste mesmo mês tivemos a primeira pessoa curada do COVID-19 e também a primeira notificação de morte por COVID-19 (SANARMED, 2020).

Cerca de 30 dias do primeiro caso confirmado, no Brasil, o número cresceu rapidamente, ultrapassando três mil pessoas contaminadas pelo covid-19. Esse número cresce ainda mais em relação ao termo de desde o primeiro caso até que se chegasse aos mil casos demorou cerca de 25 dias, mas foram apenas seis dias para que alcançasse a marca de dois mil casos. Estes números se elevaram até então ao que foi considerado o pior mês da Pandemia, sendo ele Fevereiro de 2022, onde no Brasil fechou-se com o número de mais de 28 milhões de casos confirmados e quase 650 mil mortes confirmadas pela COVID-19. (SANARMED).

3.2.1 Grupos de riscos e sua gravidade

Durante a pandemia, devido à alta taxa de infecção do COVID-19 e o fato da idade ser um fator de risco para casos graves, houve uma necessidade de dar prioridade aos grupos

considerados de maior risco, como idosos e trabalhadores da saúde, no acesso às vacinas (NACIONAL, Plano, 2020). O denominado grupo de risco é caracterizado por, potencialmente, haver mais riscos e evolução mais grave caso tenha Covid-19. Este grupo é composto por gestantes, idosos, pessoas com doenças crônicas, com deficiência ou baixa mobilidade (CDD, 2020).

3.2.2 Campanhas de conscientização COVID-19

No decorrer da pandemia, várias campanhas de conscientização surgiram com o objetivo de promover práticas preventivas, como o uso de máscaras e a questão sobre lavar as mãos. Uma dessas campanhas ganhou maior destaque na televisão e nas redes sociais ao enfatizar a mensagem "Fique em casa", que foi adotada pelas autoridades sanitárias e civis como uma estratégia para reduzir a transmissão da doença por meio do isolamento social (SOARES, et al., 2020).

Essa campanha tem como objetivo evitar aglomerações e proteger vidas, sendo diretamente relacionada ao isolamento social. Ela impactou significativamente a vida das pessoas, introduzindo novos hábitos de higiene pessoal e causando mudanças importantes na economia, incluindo uma redução significativa de pessoas nas ruas (GIBBIN, et al, 2020).

3.2.3 Métodos preventivos no Brasil

A Organização Mundial de Saúde declarou o surto de coronavírus como uma emergência de saúde pública, sendo este o nível mais alto de alarme e consequentemente levantando a proposta de acelerar o desenvolvimento de vacinas. Havia cerca de 200 projetos

registrados na OMS, destes 200 projetos, cerca de 13 estavam em fase 3 de avaliação de eficácia, a última etapa antes da aprovação pelas agências reguladoras (DOMINGUES, S.A.M.C, 2021).

Inicialmente, 170 países aderiram à estratégia da Covax Facility, inclusive o Brasil, que foi criada pela OMS para acelerar o desenvolvimento e fabricação de vacinas, garantindo o acesso justo e equitativo para todos os países. No Brasil, foram reservadas cerca de 40 milhões de doses por meio dessa iniciativa. Para garantir mais doses no Brasil, foram firmados três acordos: o Instituto de Tecnologia em Imunobiológicos da Fundação Oswaldo Cruz com a AstraZeneca, para fornecimento inicial de 100 milhões de doses; o Instituto Butantan com a Sinovac (Coronovac), para fornecimento inicial de 46 milhões de doses; e o Instituto de Tecnologia do Paraná com o Instituto Gamaleya, da Rússia (DOMINGUES, S.A.M.C, 2021).

No Brasil, as vacinas iniciaram a partir do dia 17 de Janeiro de 2021 e de acordo com a Fiocruz, um ano após já haviam cerca de 78,8% da população vacinada com a primeira dose e 68% totalmente imunizada, sendo com dose única ou segunda dose (FIOCRUZ, 2022).

3.2.4 Interferência direta da pandemia aos hospitais e aos profissionais de saúde

O Ministério da Saúde elaborou fluxogramas de atendimento em todos os hospitais e unidades de emergência, visando fornecer assistência abrangente, organizada e colaborativa aos pacientes com COVID-19 (THOMAS, et. al, 2020).

Figura 01: Fluxograma de Unidades de emergência



Fonte: Ministério da Saúde, 2020.

Figura 02: Fluxograma de Hospitais



Fonte: Ministério da Saúde, 2020.

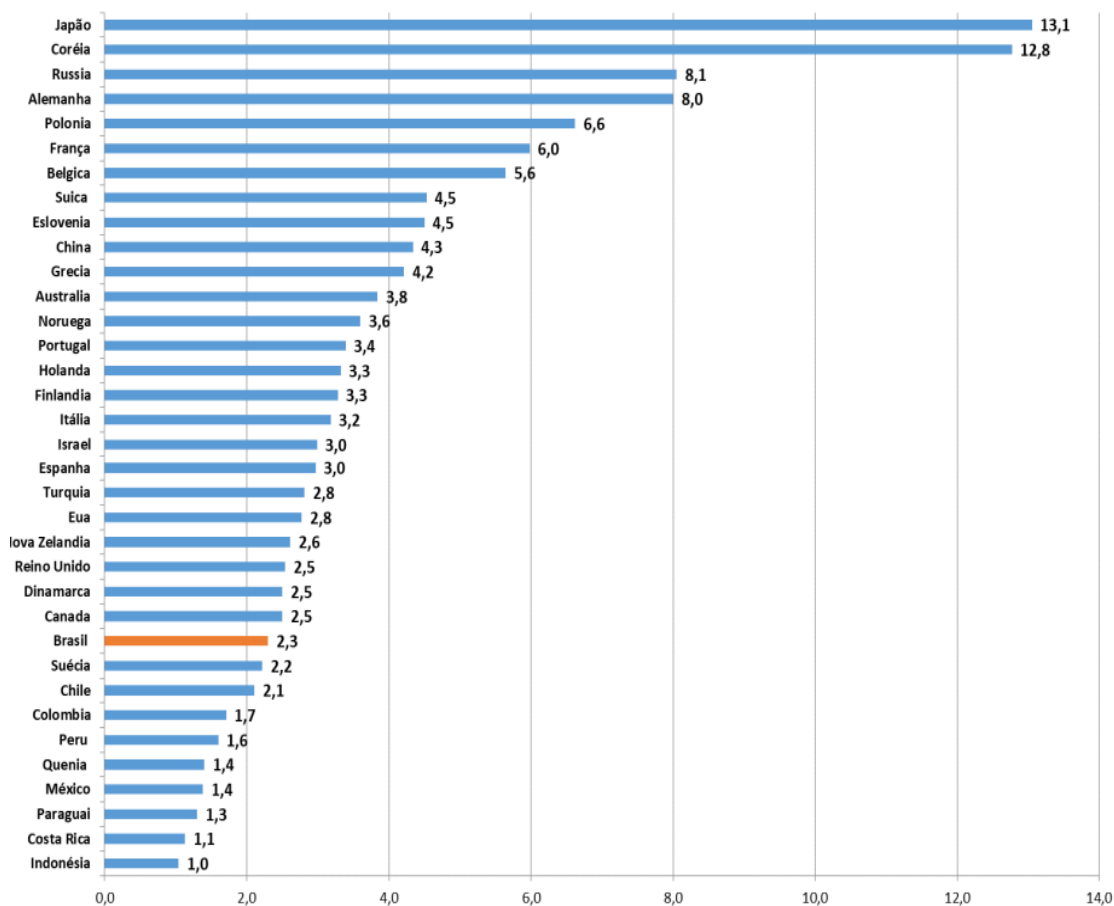
O grande desafio no âmbito hospitalar em tempos de pandemia se deu em razão da reorganização dos atendimentos, ampliação de leitos, o abastecimento de máscaras e aventais para proteção individual e a obtenção de testes suficientes para o diagnóstico (MEDEIROS, 2020).

3.3 ESCASSEZ DE LEITOS POR HABITANTES E SUA LOTAÇÃO EM TEMPOS DE PANDEMIA

Em 2019, o Brasil contava com aproximadamente 8.139 hospitais e 490.397 leitos, dos quais 270.880 em leitos gerais (clínicos e cirúrgicos) e cerca de 34.464 eram leitos de UTI adultos. Desses números, especificamente destinados ao Sistema Único de Saúde, cerca de 66% dos leitos gerais e 48% dos leitos de UTI adultos (NORONHA et al., 2020).

De acordo com estudos realizados pelo Departamento Intersindical de Estatística e Estudos Socioeconômicos (DIEESE), o Brasil possui uma média de 2,3 leitos a cada mil habitantes, e acaba tornando-se mais um país em que a sua média é menor do que a média global, que é de 3,2 leitos por mil habitantes. Ao analisar os números e esta média do Brasil, em cada estado, observa-se uma significativa disparidade, com apenas seis dos 27 estados brasileiros atingindo a recomendação do Ministério da Saúde. Esses estados são: Rio Grande do Sul (2,9), Goiás (2,8), Paraná (2,7), Distrito Federal (2,6), Rondônia (2,5) e Piauí (2,5). Além disso, ainda existem aproximadamente seis estados que possuem menos de 2 leitos a cada mil habitantes, são eles: Acre (1,9), Alagoas (1,9), Pará (1,8), Amazonas (1,6), Sergipe (1,6) e Amapá (1,5) (DIEESE, 2020).

Figura 03: Número de leitos por mil habitantes



Fonte: DIEESE, 2020.

Durante o pico do Covid no Brasil, em meados de Fevereiro de 2021, um boletim da Fiocruz revelou que 17 capitais do Brasil apresentavam uma taxa de ocupação de leitos igual ou superior à 80% e especificamente destas 17 capitais, cerca de 7 delas, incluindo Florianópolis, ultrapassaram 90% (MODELLI, 2021).

Neste mesmo período, em 2021, no estado de Santa Catarina, foi divulgado pelo governo, a matriz de risco referente à transmissão do vírus, onde citava que apenas a região do Extremo Oeste do estado estava na faixa grave, enquanto o restante se adentrava na última faixa, a gravíssima (G1 SC, 2021).

Figura 04: Regiões de Santa Catarina e a matriz de risco



Fonte: Governo/SC, 2021.

Em meados de 2022, em específico no mês de Junho, um ano após o auge da Pandemia, o estado de Santa Catarina decretou situação de emergência devido à alta taxa de atendimentos nos hospitais do SUS, ocasionando a lotação, em cerca de 97%, dos leitos devido ao aumento de doenças respiratórias durante o outono e inverno. Regiões como Foz do Rio Itajaí, Meio Oeste, Serra Catarinense e Grande Oeste a ocupação chegou a 100% e ainda havia pacientes na lista de espera no aguardo por atendimento (BRASIL, 2022).

Já em maio de 2023, a Organização Mundial de Saúde (OMS) anunciou o fim da emergência da Covid-19, considerando que o vírus não representa mais uma ameaça sanitária internacional (CHADE, 2023). No entanto, o estado de Santa Catarina voltou a enfrentar uma alta demanda nas Unidades de Terapia Intensiva (UTIs), desta vez devido a doenças respiratórias e casos de dengue (Batistella, 2023). A ocupação dos leitos atingiu cerca de 97%, sendo que na região da Grande Florianópolis, todas as unidades estavam completamente lotadas. Diante dessa situação, foi anunciada a intenção de abrir 63 novos leitos de UTI para lidar com essa alta demanda mesmo no período pós pandemia (AMORIM, 2023).

3.4 CONSEQUÊNCIAS DA PANDEMIA RELACIONADAS À PRORROGAÇÃO DE CIRURGIAS

Com alta taxa de propagação do novo coronavírus, serviços como as cirurgias eletivas foram suspensas temporariamente, buscando a segurança dos pacientes, protegendo-os de contaminação em períodos pré-operatório e pós-operatório nos hospitais, além de aumentar os recursos para os pacientes com covid-19. Estima-se que, nas primeiras semanas do pico de COVID-19 no mundo, cerca de 70% de todas as cirurgias foram canceladas, o que consequentemente irá gerar um acúmulo em breve (ROCCO et al, 2019).

Em uma breve comparação, no período de 2016 até 2020, com dados obtidos através do DataSUS, mostra o quanto caiu o número de cirurgias, tanto emergenciais quanto eletivas, principalmente quando se é comparado no período 2019-2020, onde o número corresponde a cerca de 19% no total, sendo 40% em específico aos números de cirurgias eletivas (TRUCHE, et al).

Figura 05: Total de cirurgias realizadas no Brasil no período de 2016 até 2020



Fonte: Truche, et al. 2022. Título do artigo: Association between government policy and delays in emergent and elective surgical care during the COVID-19 pandemic in Brazil: a modeling study, 3-3.

Após a Pandemia, especificamente no Brasil, cerca de um milhão de procedimentos cirúrgicos foram suspensos ou revogados e o acúmulo de cirurgias eletivas supera a marca de

900.000 casos. Dentre estes números, destaca-se que todas as regiões do país, que tiveram um acúmulo de cirurgias em 2020 até então, com destaque, principalmente, para as Regiões Nordeste e Sudeste, onde obtiveram o maior número de cirurgias emergenciais canceladas ou suspensas, com números superiores a 45 mil e em cirurgias eletivas, onde os números superaram 220 mil casos, sendo que, no Sudeste a marca passou aproximadamente de 474 mil casos (TRUCHE, et al).

Figura 06: Regiões do Brasil e seu acúmulo de cirurgias em 2020

Region	Total Backlog		
	Total	Emergent	Elective
Midwest	65,705(30,472–114,123)	15,803(1097–47,846)	49,327(30,017–69,979)
North	70,398(32,588–115,454)	25,118(10,985–53,901)	34,774(11,236–69,508)
Northeast	281,355(177,829–411,413)	47,693(9585–123,875)	224,120(145,164–315,719)
South	209,229(147,171–271,446)	25,596(5064–62,057)	186,373(145,549–228,113)
Southeast	492,746(374,603–611,560)	47,110(10,737–107,799)	434,164(343,237–525,449)
Total	1119,432 (762,663 - 1523,996)	161,329(37,468–395,478)	928,758(675,203–1208,768)

Fonte: Truche, et al. 2022. Título do artigo: Association between government policy and delays in emergent and elective surgical care during the COVID-19 pandemic in Brazil: a modeling study, 4-4.

Diante de toda demanda acumulada, ocasionada diretamente pela pandemia, algumas medidas cabíveis serão necessárias para otimização desta espera, afins de evitar longas filas, tais como uma reestruturação dos processos de trabalho e das unidades que realizam assistências, aumento da oferta em questão financeiras e também na ampliação dos serviços cirúrgicos, além de haver um critério para priorização, unificando as listas de esperas (FILHO, 2021).

4 PROCEDIMENTOS METODOLÓGICOS

No que diz respeito à metodologia, foi realizado diversas pesquisas em documentos, artigos e portais de informações relacionados à Saúde, para entender melhor como o Sistema Único de Saúde se relaciona com a situação atual, especialmente durante a pandemia, visando ver como isso afetou a administração dos leitos e salas de cirurgia nos hospitais.

A justificativa se dá por pesquisas do porque e como a pandemia do Covid-19 trouxe muitos desafios e mudanças na área da saúde em todo o mundo, incluindo aqui no Brasil, com foco principal na lotação dos leitos e na postergação das cirurgias.

Quando entendemos esses conceitos e a relação com a situação atual e também com os objetivos citados na pesquisa, podemos encontrar oportunidades para melhorar a prestação de serviços aos pacientes e também um controle interno mais eficiente. Uma ideia que surgiu é o desenvolvimento de protótipo de software web para gerenciar os leitos e procedimentos cirúrgicos no período pós-pandemia.

Para realização deste protótipo foi definido a escolha do Banco de Dados *PostgreSQL*, pela sua confiabilidade e escalabilidade, além das tecnologias Node.js e Vue.js, ambas baseadas no JavaScript, mas cada uma com suas responsabilidades, onde Vue.js é um framework utilizado para aplicações web, proporcionando agilidade para o desenvolvedor e uma interatividade ao lado do usuário, além da facilidade de ser incorporado a projetos existentes. Quanto ao Node.js, este é o responsável pela execução de códigos JavaScript no servidor, sendo utilizado para criar a lógica do servidor, manipular solicitações e respostas, e interagir com o Banco de Dados PostgreSQL.

5 ANÁLISE DOS REQUISITOS

Nesta etapa abordaremos os passos necessários para o desenvolvimento bem sucedido do protótipo, estabelecendo uma base consistente de requisitos e compreendendo as necessidades elencadas.

5.1 LEVANTAMENTO E VALIDAÇÃO DOS REQUISITOS

O levantamento e validação dos requisitos se deu por meios de estudos presentes no referencial teórico além de pesquisas com 4 profissionais da área da Saúde, sendo auxiliares em Enfermagem, enfermeiros e profissionais da secretaria de Saúde. Desses profissionais, 2 residem e atuam diretamente no trabalho voltado ao município de Presidente Olegário, em Minas Gerais e os outros dois atuam no município de Pinhalzinho, Santa Catarina. A primeira, Leila Aparecida dos Santos, auxiliar de enfermagem, *COREN*: 245356, atuou diretamente durante cerca de 32 anos no Hospital Darci José Fernandes, em Presidente Olegário. Em segundo, Vívian Lara dos Santos Araújo, enfermeira, *COREN*: 41397, atuando diretamente no município de Presidente Olegário. Além de Ceonara Balen, *COREN*: 611368, auxiliar em enfermagem, atuou na linha de frente na pandemia no município de Pinhalzinho, Santa Catarina e Neli Maria Triches Knop, profissional da Secretaria de Saúde do município.

De modo geral foi relatado a dificuldade no gerenciamento de leitos e das cirurgias como um todo, mesmo após o período pandêmico. Diante deste cenário, em ambos municípios, visando a segurança dos pacientes, profissionais e demais envolvidos, foi interrompido de forma geral todas as cirurgias eletivas constatadas que poderiam esperar, como por exemplo cirurgia de Vesícula, Hérnia e afins. Quanto aos leitos, na pandemia, o Hospital do Município de Pinhalzinho aderiu a uma denominada central de leitos, responsável pelo controle de disponibilidade dos mesmos, visto que toda a região estava em estado de emergência devido à lotação nos hospitais e o município de Pinhalzinho ficou a poucos passos

de aderir a um Hospital de Campanha. Essa central tinha o papel de designar, no ato, o paciente que necessitava ser internado e para onde ele deveria ser enviado para internação, onde tivemos como exemplos, pacientes de Pinhalzinho designados a Joinville e pacientes de Concórdia internados no município de Pinhalzinho. Já no município de Presidente Olegário, a ordem foi conforme a disponibilidade dos leitos no próprio hospital, através de um sistema do *SUS* para redirecionamento para outro município conforme a urgência, ou seja, a disponibilidade dos leitos era separada, sendo leitos gerais e leitos destinados ao COVID-19, estes controlado através da ficha dos pacientes, impressa, onde a profissional que atuava na recepção buscava a informação se havia leito ou não disponível e no caso da não disponibilidade, era encaminhado a uma sala de observação até uma liberação dos leitos.

Quanto a isso, em ambos municípios e nos demais estudos realizados no referencial teórico, foi identificado que a maior dificuldade se dá em relação à gestão dos leitos de forma eficiente e automatizada, visando melhorar o fluxo dos pacientes, além de um controle para agendamento das cirurgias prorrogadas, principalmente as eletivas.

Frente à necessidade identificada, deu-se início ao processo de definição dos requisitos funcionais, não funcionais e da descrição do protótipo, tendo como ponto de partida a exigência de que a plataforma opere em um ambiente online, demandando acessos simultâneos e que seja apta a integração. Ao analisar o contexto de maneira abrangente, identificamos dois potenciais atores envolvidos nesse processo: o Administrador e o Profissional de Saúde, onde há a possibilidade de ambos se tornarem um ator só: Profissional da Saúde.

5.2 DESCRIÇÃO DO PROTÓTIPO DO SISTEMA

O protótipo em questão, é voltado à gestão hospitalar, viabilizando a otimização do gerenciamento de leitos e procedimentos cirúrgicos em ambientes hospitalares. O intuito é ter uma interface intuitiva, com dados concisos e de fácil acesso, como quantidade de pacientes e cirurgias realizadas, além de agendar cirurgias, realizar o cadastro de forma geral, como alas,

médicos, enfermeiros e demais possibilidades. Contém também o controle dos leitos, agrupados por alas, o que proporciona eficiência na hora de buscar a disponibilidade de leitos quando chega um novo paciente ao hospital, melhorando a qualidade de atendimento ao paciente.

5.3 REQUISITOS FUNCIONAIS

Os requisitos funcionais são especificações das funcionalidades e capacidades que um sistema, software ou produto deve oferecer para atender às necessidades e expectativas dos usuários, clientes ou stakeholders, visando a satisfação de maneira ideal. Com este embasamento, os requisitos funcionais identificados do protótipo são:

- Autenticação dos Usuários;
 - Os usuários devem fornecer credenciais válidas para acesso ao sistema;
 - Sistema de recuperação de senha para usuários autorizados.
- Cadastros;
 - Registro de Alas, Cirurgias, Enfermeiros, Especialidades, Leitos, Médicos, Pacientes, Pessoas e Tipos de Cirurgias, fornecendo as funcionalidades de inserção, consulta, edição e inativação/deleção dos dados.
- Controle de Cirurgias;
 - Agendamento de cirurgias, atualizações e exclusões/inativações ao agendá-las;
 - Listagem diária, mensal e semanal das cirurgias agendadas;
 - No Kanban, uma listagem de acordo com o período da cirurgia, mantendo um controle a partir de novo, em andamento ou finalizado, fornecendo a possibilidade de atualizações em tempo real.
- Controle de Leitos;
 - Registro de Ocupação dos Leitos;
 - Indicativo da disponibilidade de leitos disponíveis por ala;
 - Possibilidade de informações dos leitos ocupados em um único local, com informativo de paciente, observações e o próprio enfermeiro responsável.

- Cadastro de Usuários;
 - Novos usuários podem se registrar no sistema, fornecendo informações como e-mail e senha;
 - Verificação de dados ao cadastrar, para manter a integridade dos registros.
- Dashboard;
 - Apresentação de dados principais do protótipo;
 - Visualização de estatísticas e informações cruciais referente às cirurgias, leitos, pacientes e procedimentos cirúrgicos;
 - Permitir a navegação intuitiva para outras telas.

5.4 REQUISITOS NÃO FUNCIONAIS

Requisitos não funcionais referem-se a critérios que não descrevem as funcionalidades específicas do sistema, mas sim qualidades ou características que são importantes para o seu desempenho global. Os requisitos não funcionais do protótipo são:

- Desempenho: O requisito desempenho refere-se à eficiência operacional do sistema, incluindo a rapidez com que responde a solicitações do usuário, processa dados e executa operações. Esse aspecto é crucial para garantir uma experiência do usuário satisfatória e para otimizar o uso de recursos computacionais.
- Disponibilidade: Disponibilidade, como um requisito não funcional, refere-se à capacidade de um sistema estar operacional e acessível quando necessário. Este é um aspecto crítico, especialmente para sistemas web, onde a interrupção do serviço pode resultar em perda de receita, insatisfação do usuário e danos à reputação.
- Escalabilidade: O requisito escalabilidade está relacionado à capacidade do sistema de lidar com um aumento de demandas significativas no desempenho. Sistemas bem escaláveis podem crescer em dois modelos: Horizontalmente,

adicionando mais recursos e Verticalmente, aumentando a capacidade dos recursos existentes.

- Segurança: Por último, quando falamos do requisito segurança, nos referimos a abordagem relacionada à proteção do sistema contra ameaças internas e externas. Para um sistema web, isso envolve a implementação de práticas de segurança, como autenticação robusta, autorização adequada, criptografia de dados sensíveis, prevenção de ataques como SQL injection e cross-site scripting (XSS), além de monitoramento contínuo para detectar e responder a possíveis violações de segurança.

5.5 UML

O *UML* é o acrônimo para *Unified Modeling Language* e é uma linguagem que nos auxilia na tarefa de modelar e documentar sistemas (DevMedia, 2018).

Surgindo em 1996, com a ajuda de Grady, Booch e Rumbaugh, a linguagem de modelagem unificada, ou popularmente *UML* foi criada com o objetivo de padronizar a modelagem no desenvolvimento, design e arquitetura de um software e é dividido em dois grandes grupos: diagramas estruturais e diagramas comportamentais. (LucidChart, 2023).

Os diagramas estruturais são ferramentas usadas para descrever e representar de maneira visual os elementos estáticos de um sistema, como classes, métodos, interfaces e a arquitetura geral. Eles são úteis para especificar, construir e documentar a estrutura do sistema de forma clara e compreensível. Os principais diagramas estruturais são os diagramas de classe, diagramas de objetos e diagramas de estruturas (ANDRADE, 2019).

Quanto aos diagramas comportamentais, por sua vez, são usados para mostrar como o sistema se comporta dinamicamente. Eles detalham o funcionamento de funcionalidades específicas, como processos de negócios, interações do usuário e o tratamento de diversas operações. Esses diagramas ajudam a exemplificar o comportamento esperado do sistema em

diferentes situações e seus principais diagramas são os de casos de uso, diagramas de sequência e diagramas de atividades (ANDRADE, 2019).

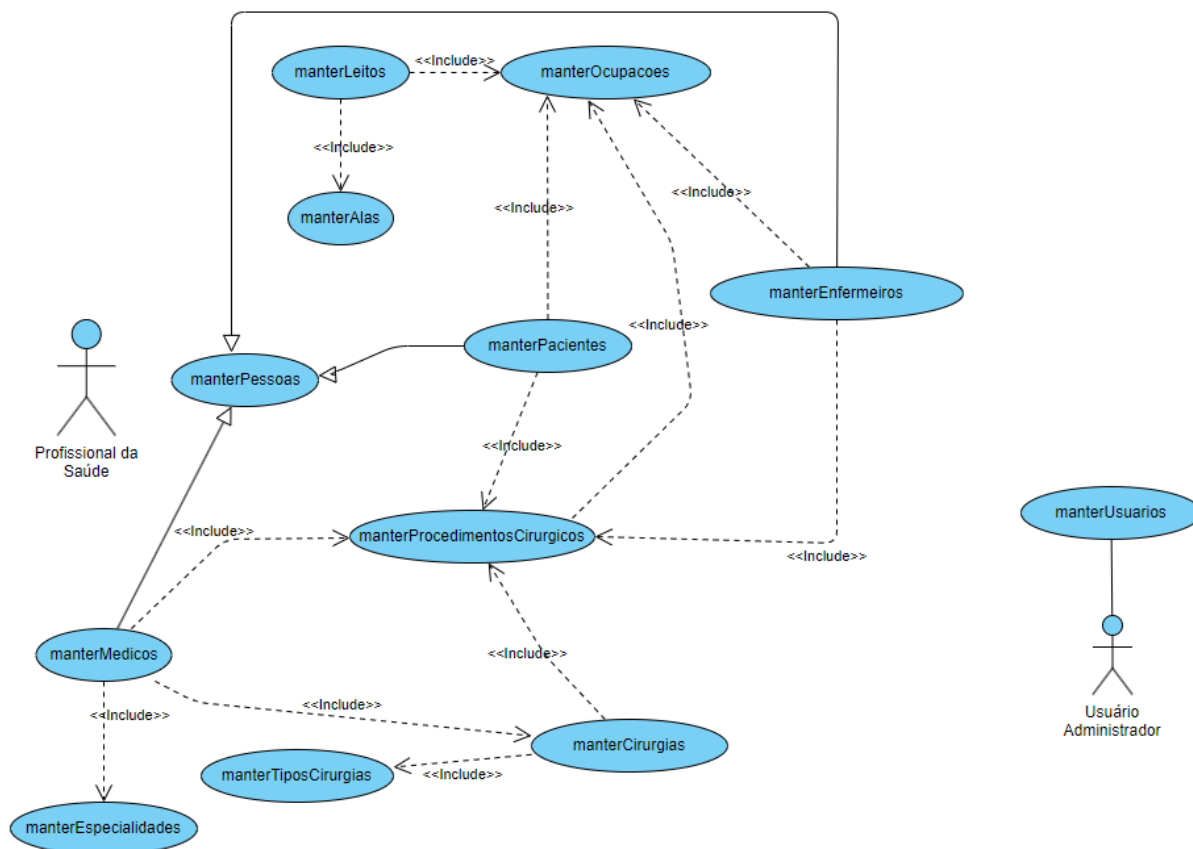
5.5.1 Diagrama de Casos de Uso

Os casos de uso desempenham um papel fundamental na comunicação entre os usuários e desenvolvedores, sendo considerado uma ferramenta de simulação que modela como os clientes interagem com um produto ou sistema e resulta na criação de diagramas de modelo de caso de uso, os quais visualizam os usuários, suas interações com o sistema e como o sistema responde, representando visualmente o fluxo, comportamento e ilustrando as funcionalidades do sistema de acordo com as interações do próprio usuário (Mosaico, 2022).

Segundo Macoratti (2011), os casos de uso descrevem a sequência de ações que compõem o cenário principal, bem como cenários alternativos. O objetivo é demonstrar o comportamento de um sistema por meio das interações dos atores. Os principais componentes dos casos de uso incluem:

- Atores: Quem interage com o sistema;
- Relacionamentos: Como os atores se conectam com os casos de uso;
- Casos de Uso: Descrições dos cenários ideais e alternativos com base nas ações do usuário.

Figura 07: Diagrama de Casos de uso do protótipo



Fonte: elaborado pelo autor.

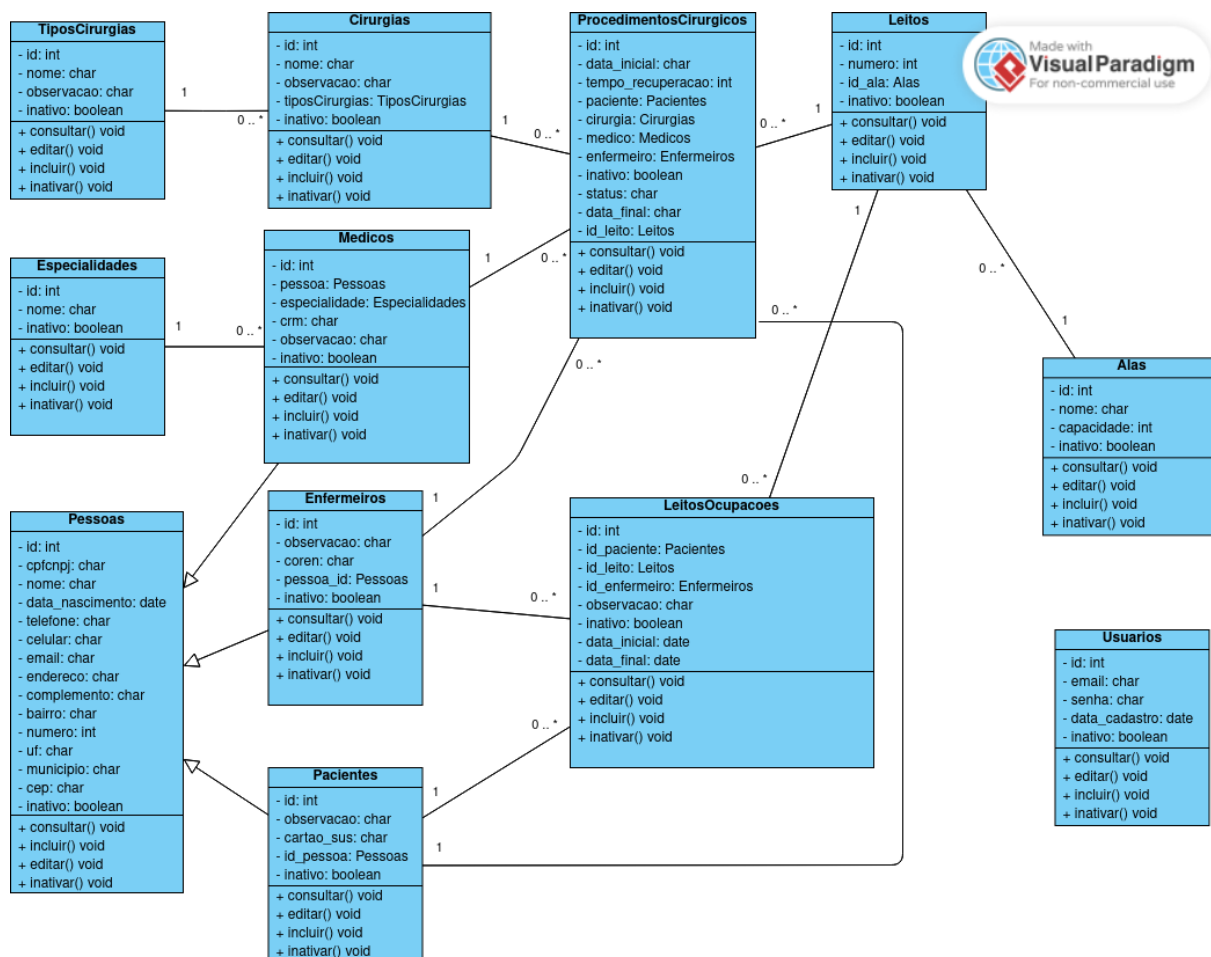
O caso de uso da imagem acima representa a sequência de ações possíveis no protótipo do sistema, onde o ator, definido por um profissional da saúde pode realizar interações relacionadas à inserção, edição, deleção e consulta de dados como leitos, procedimentos cirúrgicos, médicos, pacientes e demais. A outra possibilidade é o ator, denominado então administrador, realizar as operações de consultas, inserção, edição e deleção relacionada aos usuários.

5.5.2 Diagrama de Classes

Este diagrama é um dos seis diagramas estruturais presentes no *UML* e é considerado fundamental para o processo de modelagem dos objetos e da estrutura do sistema, visando exibir os relacionamentos, serviços e utilidades dos objetos (IBM, 2021).

No diagrama de classes, uma classe é representada por três camadas, como nome, atributos e métodos (Douglas, 2016).

Figura 08: Diagrama de Classes de uso do protótipo



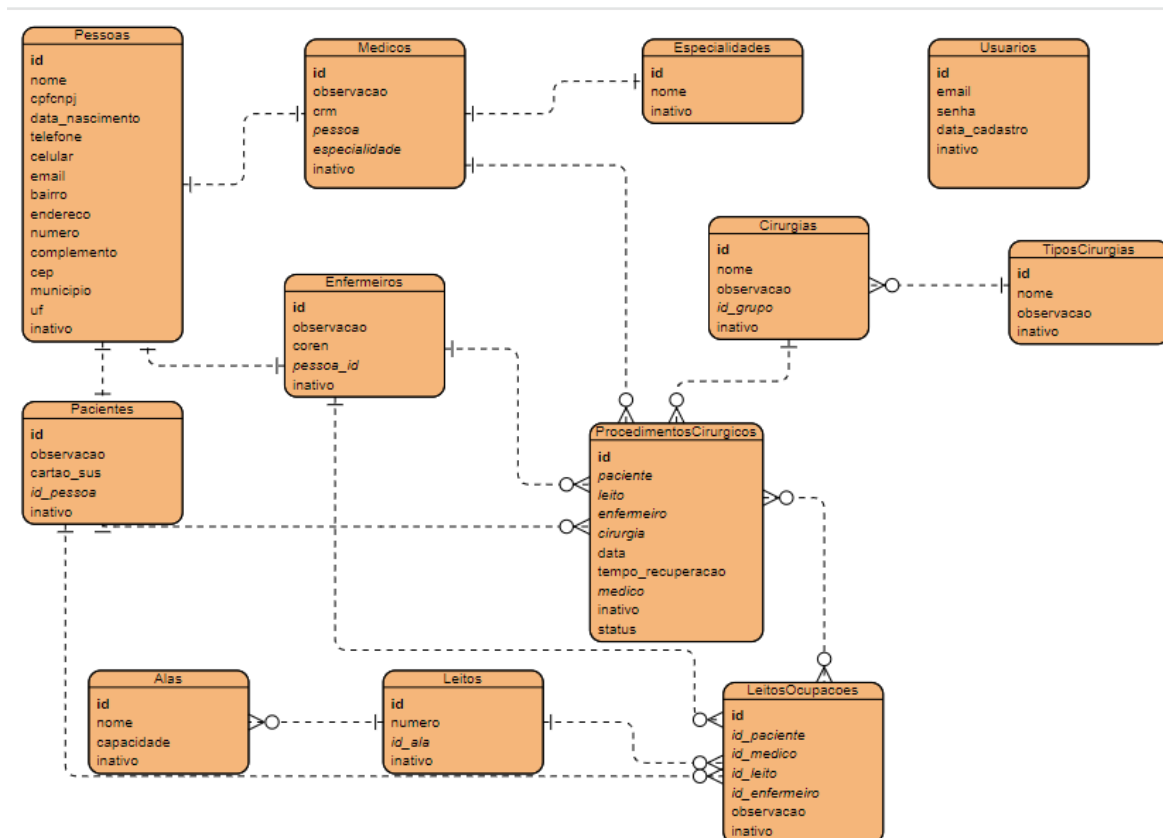
Fonte: elaborado pelo autor.

Na Figura 08, representada acima, está representada toda a estrutura do diagrama de classes e seus métodos e atributos, além das relações entre as classes definidas para posterior criação das tabelas no banco de dados.

5.5.3 Diagrama de Entidade e Relacionamento

Diagrama de entidade relacionamento é um fluxograma que ilustra como entidades, pessoas, conceitos ou objetos se relacionam em relação a um sistema, tendo seu uso voltado a depuração de banco de dados relacionais, definindo assim a estrutura de um banco de dados (FRANK et al., 2021).

Figura 09: Diagrama de Entidade e Relacionamento do protótipo



Fonte: elaborado pelo autor.

A representação gráfica fornecida na Figura 09 oferece um exemplo concreto da aplicação do diagrama de entidade e relacionamento, contextualizando-o no âmbito do protótipo em processo de desenvolvimento. Essa visualização não apenas ilustra a estrutura do banco de dados, mas também destaca as conexões entre as entidades, proporcionando uma compreensão mais profunda do sistema em evolução.

5.5.4 Diagrama de Sequência

O diagrama de sequência, ou também conhecido como diagrama de eventos ou cenários de eventos, é uma solução dinâmica de modelagem *UML* e serve como um diagrama de interação, descrevendo como e em qual ordem um grupo de objetos deve trabalhar em conjunto, sendo utilizado por desenvolvedores para entender as necessidades de um novo sistema (LucidChart, 2023).

Um diagrama de sequência compreende um conjunto de objetos representados por linhas de vida, e ilustra as mensagens que eles trocam durante sua interação. Além disso, esses diagramas exibem a ordem das mensagens transmitidas entre os objetos e revelam as estruturas de controle entre eles (IBM, 2021).

5.5.4.1 Diagrama: manterProcedimentosCirurgicos

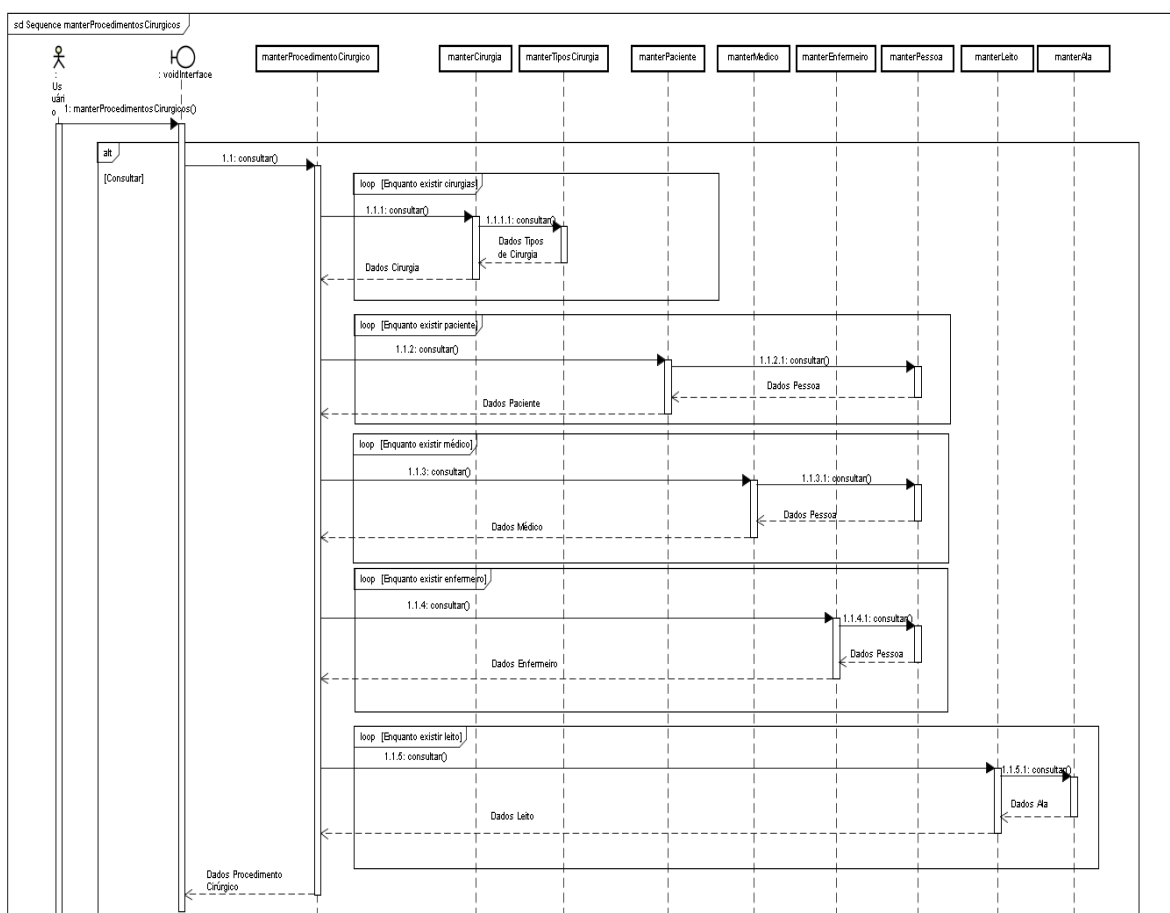
O diagrama em questão retrata os processos lógicos utilizados para retratar a interação referente às operações de consultar, editar, inserir e inativar envolvendo os procedimentos cirúrgicos e demais relações presentes quando utilizado o caso de uso “manterProcedimentosCirurgicos”. Devido ao seu extenso tamanho, este diagrama foi

dividido em partes, onde cada parte é responsável por uma operação, sendo elas: Consultar, Editar, Inserir e Inativar.

5.5.4.1.1 Operação: Consultar

A interação Consultar é responsável pela listagem dos procedimentos cirúrgicos realizados. Para isso ocorrer, é necessário a consulta de demais interações, que neste caso se referem a consulta de paciente, médico, enfermeiro e o leito, o que por fim, ao listar todas as interações relacionadas, retorna uma lista de procedimentos cirúrgicos.

Figura 10: Diagrama de Sequência - Consultar



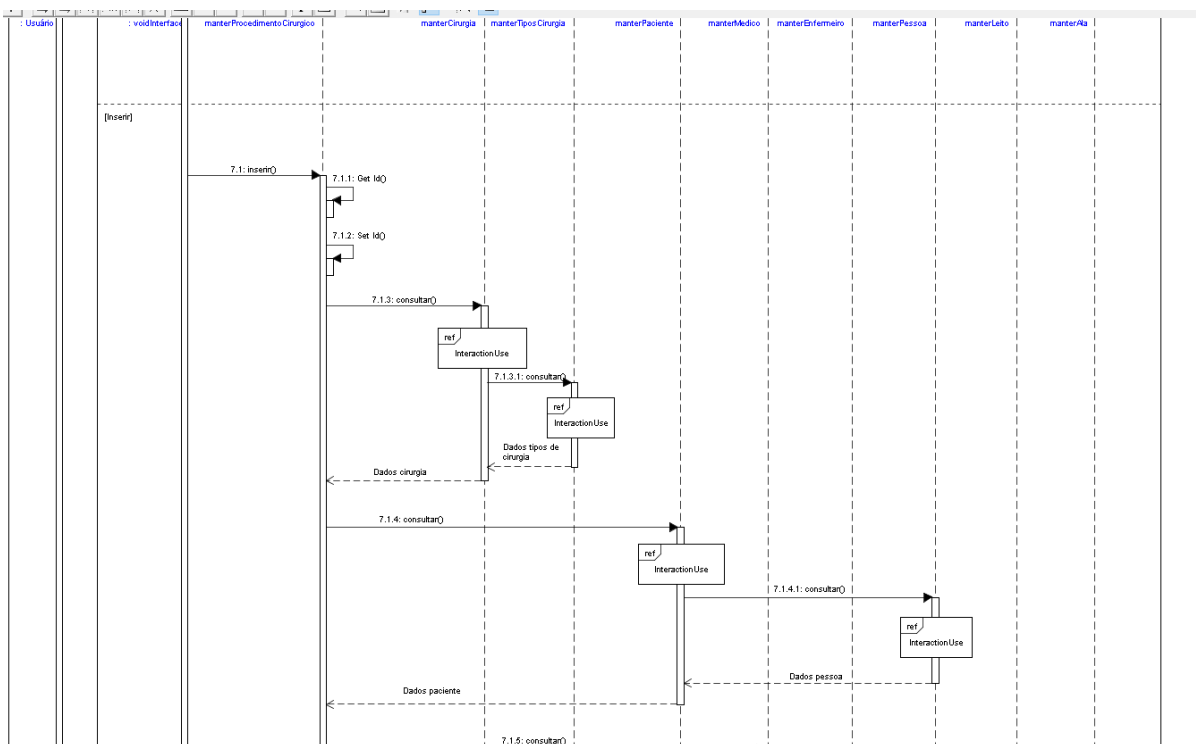
Fonte: elaborado pelo autor.

A Figura 10 fornece exemplos práticos da interação Consultar designada no Diagrama de Sequência, fornecendo a listagem dos procedimentos cirúrgicos realizados.

5.5.4.1.2 Operação: Inserir

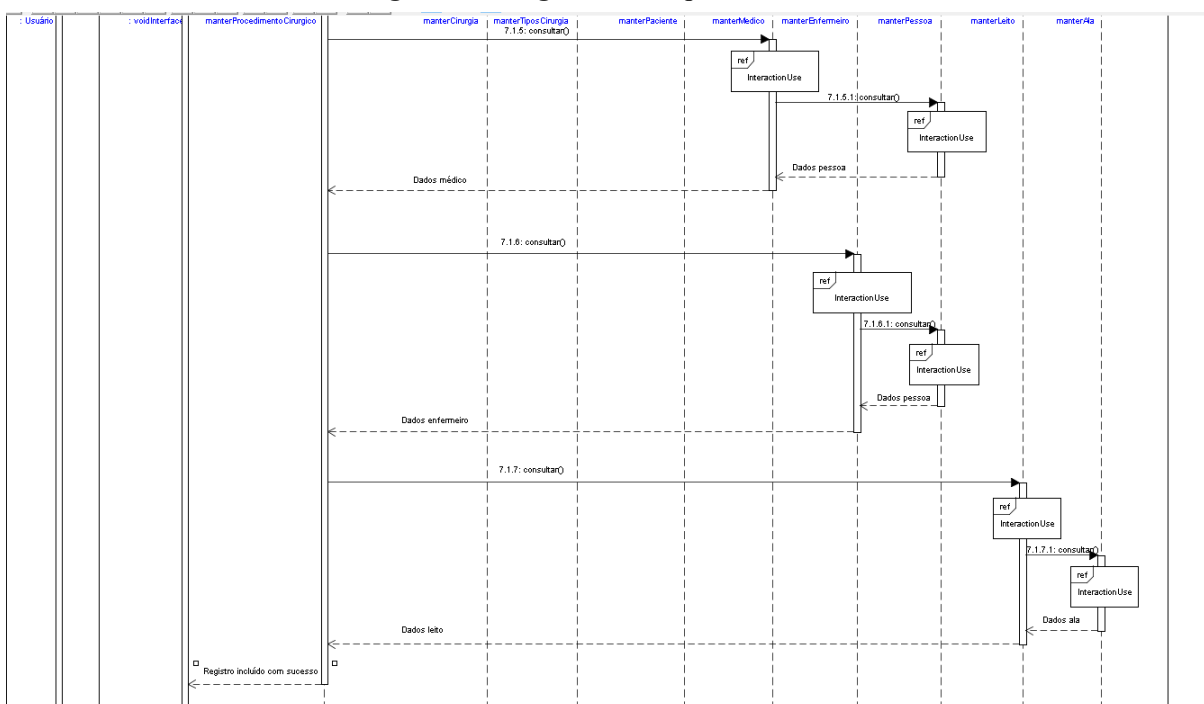
A interação Inserir é responsável pela rotina de interação de uma inserção, repassando os pontos necessários para que seja possível, neste caso, inserir um novo procedimento cirúrgico. Nesta primeira parte, para realizar a inserção, foi necessário buscar o código sequencial, posteriormente buscar o tipo de cirurgia e também o paciente, verificando se há a ambos registros já previamente existentes para associar a interação da inserção.

Figura 11: Diagrama de Sequência - Parte 1 Inserir



Fonte: elaborado pelo Autor.

Figura 12: Diagrama de Sequência - Parte 2 Inserir



Fonte: elaborado pelo autor.

Na Figura 12, complementado a Figura 11, foi necessário consultar o enfermeiro, médico e o leito, ambas interações validando se há ou não a possibilidade de associar cada um deles na interação.

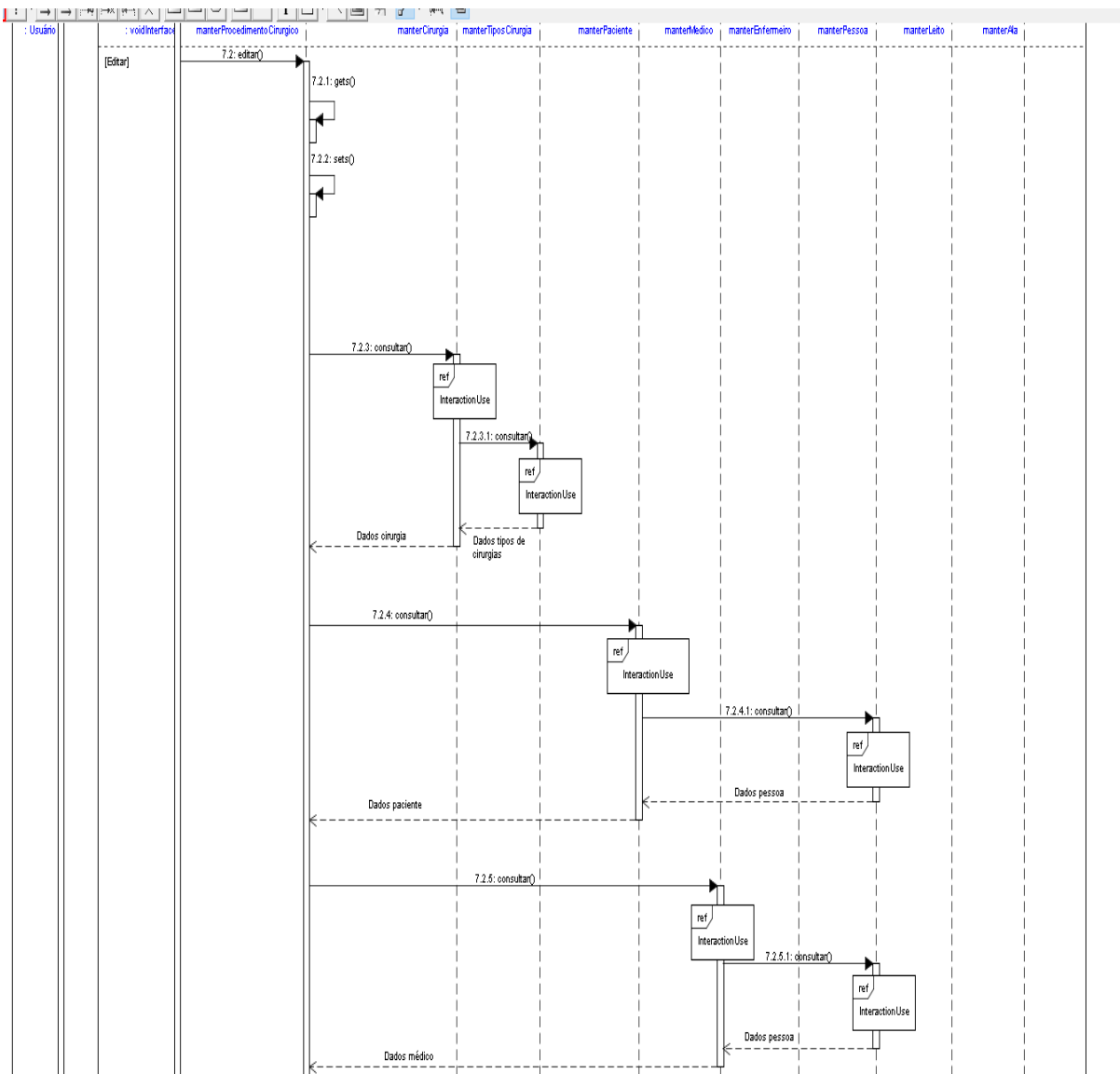
5.5.4.1.3 Operação: Editar

A interação de Editar se refere a uma interação responsável por modificar um procedimento cirúrgico já existente, mas que para isso, é necessário implementar uma sequência lógica para que a própria edição funcione, seguindo a sequência estipulada no diagrama em questão.

No Diagrama criado, foi estipulado as ações referentes ao diagrama, como as consultas de cirurgias, tipos, médicos, enfermeiros, pacientes, leitos e alas.

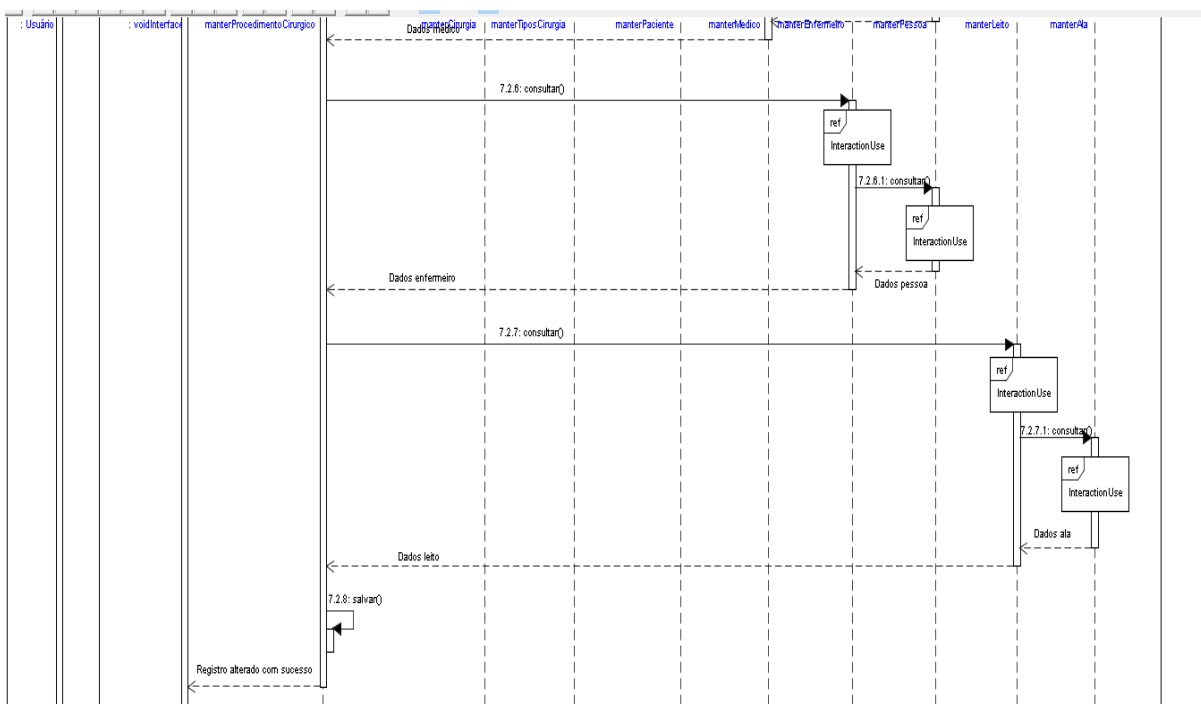
Na parte um, representada pela Figura 13, temos a interação voltada às ações de consultar a cirurgia e seu tipo, paciente, médico e a pessoa referente.

Figura 13: Diagrama de Sequência - Parte 1 Editar utilizando o Inserir



Fonte: elaborado pelo Autor.

Figura 14: Diagrama de Sequência - Parte 2 Editar



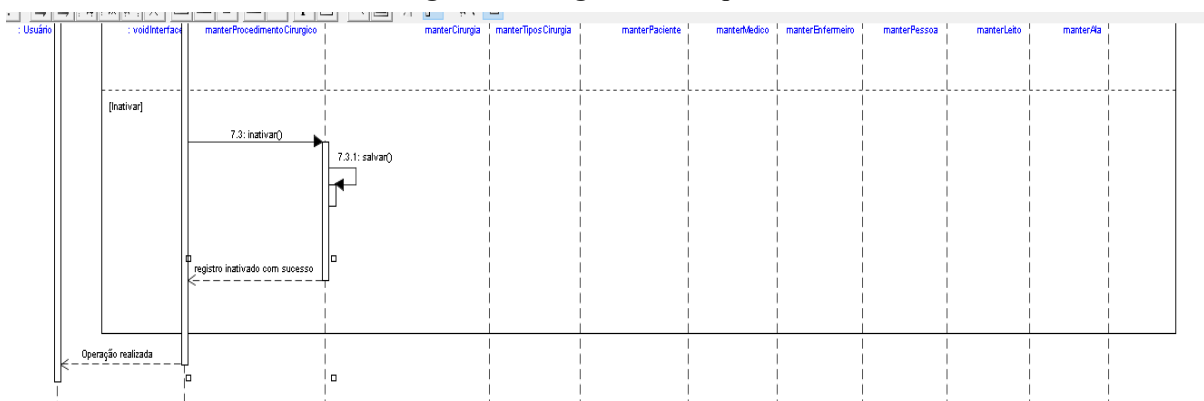
Fonte: elaborado pelo autor.

Na parte dois, representada pela Figura 14, ao Editar, nesta etapa temos a interação relacionada ao médico e leito, consultando também a pessoa e ala referentes. Por fim, este diagrama representa em si os processos realizados para edição de um procedimento cirúrgico, realizando todos os passos necessários para uma edição de sucesso.

5.5.4.1.4 Operação: Inativar

A interação “inativar” é responsável por modificar um procedimento cirúrgico existente, tornando-o oculto, onde a sequência de interação é restrita à inativação do procedimento cirúrgico, salvar a interação realizada e retornar que foi inativado de fato.

Figura 15: Diagrama de Sequência - Inativar



Fonte: elaborado pelo autor.

Representada pela Figura 16, essa operação contribui para a integridade do sistema, garantindo que as modificações sejam rastreáveis e se mantenham ocultas a nível operacional, sendo responsável por inativar um procedimento cirúrgico.

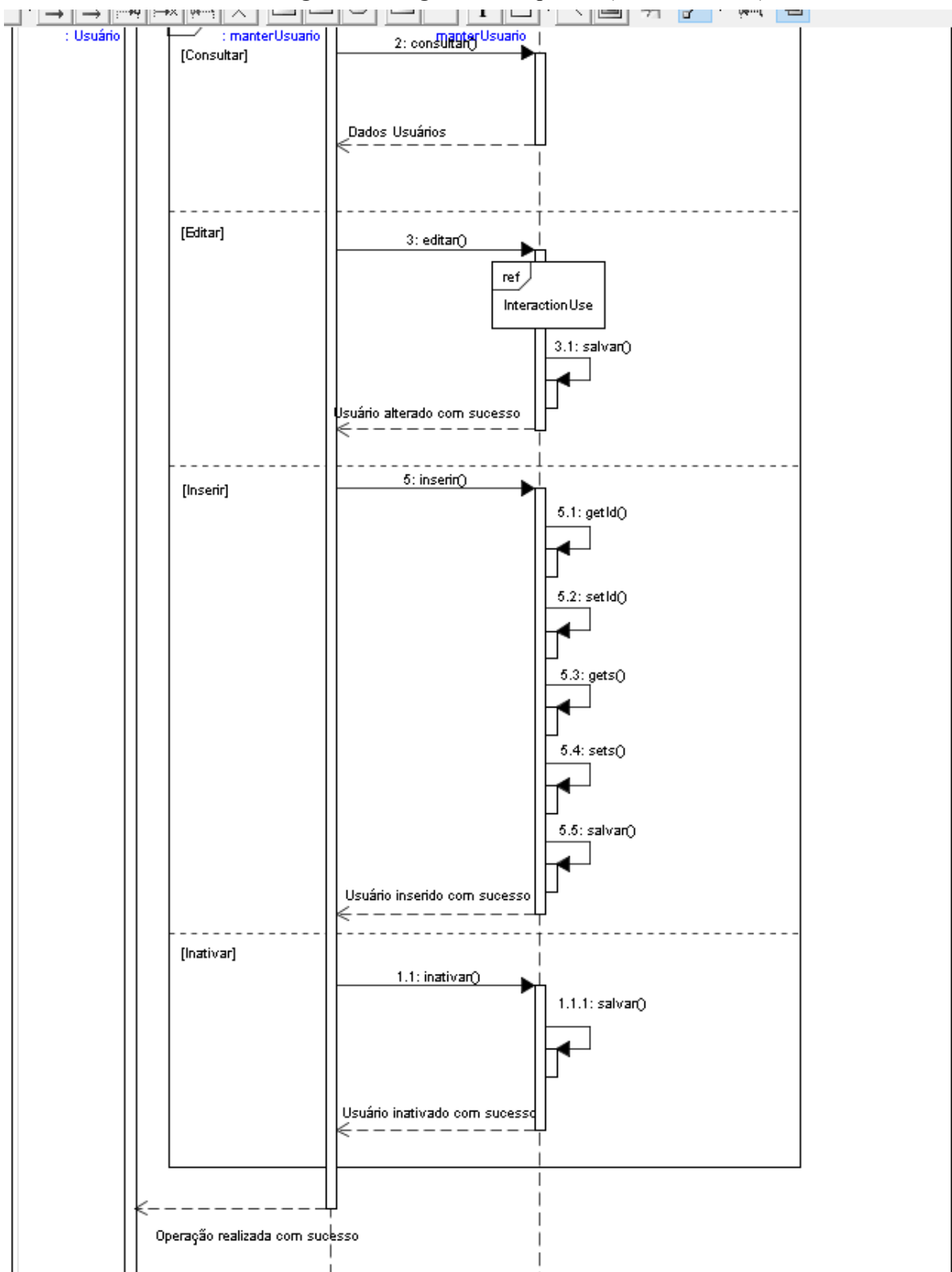
5.5.4.2 Diagrama: manterUsuario

O diagrama em questão retrata os processos lógicos utilizados para retratar a interação referente às operações envolvendo os usuários quando utilizado o caso de uso “manterUsuários”.

Este diagrama retrata quatro possíveis alterações, que são:

- Consultar: responsável pela interação de consulta dos usuários cadastrados no protótipo do sistema, retornando uma lista de dados dos usuários;
- Editar: responsável pela interação de edição dos usuários cadastrados no protótipo do sistema;
- Inserir: responsável pela interação de inserção de um novo usuário no protótipo do sistema;
- Inativar: responsável pela interação de inativação de um usuário no protótipo do sistema.

Figura 16: Diagrama de Sequência (Manter Usuário)



Fonte: elaborado pelo Autor.

Representado na Figura 17, as operações Editar, Inativar e Inserir têm a necessidade de uma interação extra, comparado a interação do “consultar”, que seria em relação ao salvar, para que concretize a interação realizada.

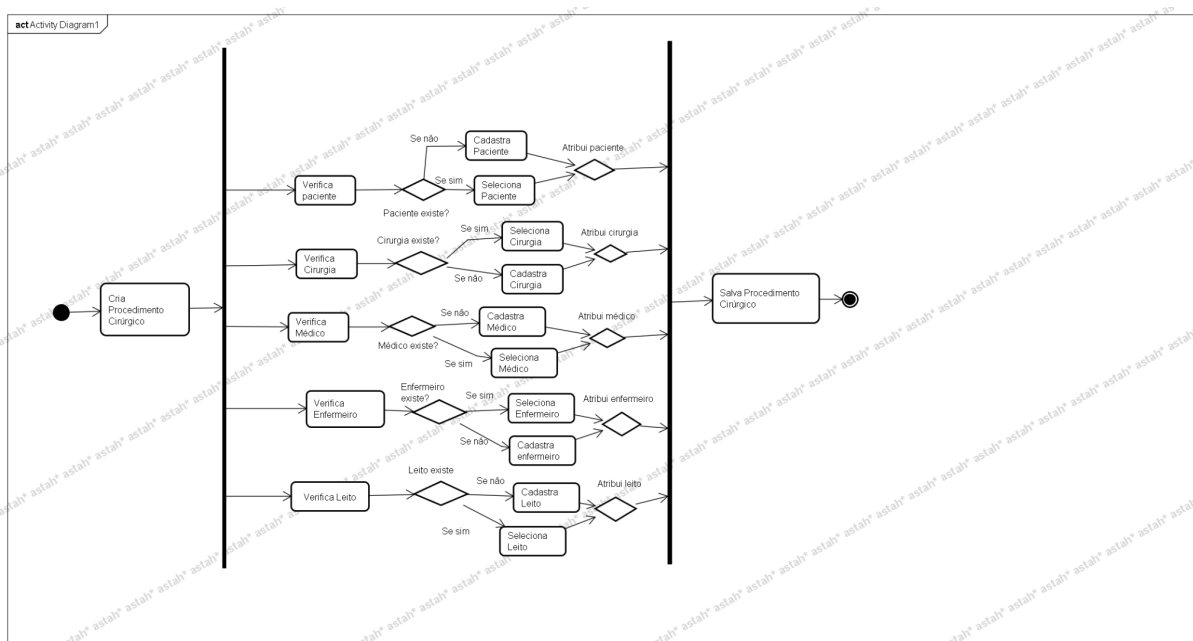
5.5.5 Diagrama de Atividade

O diagrama de atividades é considerado um dos diagramas de comportamento, visto que propõe o que é necessário que aconteça no sistema que está sendo modelado. Este diagrama em específico ajuda a unir as pessoas de outras áreas para entender o processo como um todo, além do comportamento, visto que tende a oferecer a descrição das etapas realizadas em um caso de uso *UML* e a simplificação e melhoria de qualquer processo ao esclarecer os casos de uso complicados (LucidChart, 2023).

5.5.5.1 Criar um procedimento cirúrgico

O diagrama de atividade em questão retrata o comportamento necessário para que seja possível criar um procedimento cirúrgico, realizando todo o seu passo-a-passo para que seja bem sucedido. Para que seja possível é estipulado diversas situações para efetivar que a atividade seja concluída, como por exemplo criar um paciente caso o mesmo não esteja cadastrado no momento em questão.

Figura 17: Diagrama de Atividade - Criar um procedimento cirúrgico



Fonte: elaborado pelo autor.

Na imagem acima, representa o diagrama de atividades de criação de um procedimento cirúrgico, onde está inicialmente separado por um *fork* para utilizar múltiplas atividades em paralelo e no fim utilizado o *join* para finalizar as atividades realizadas, por fim, finalizando e salvando o procedimento.

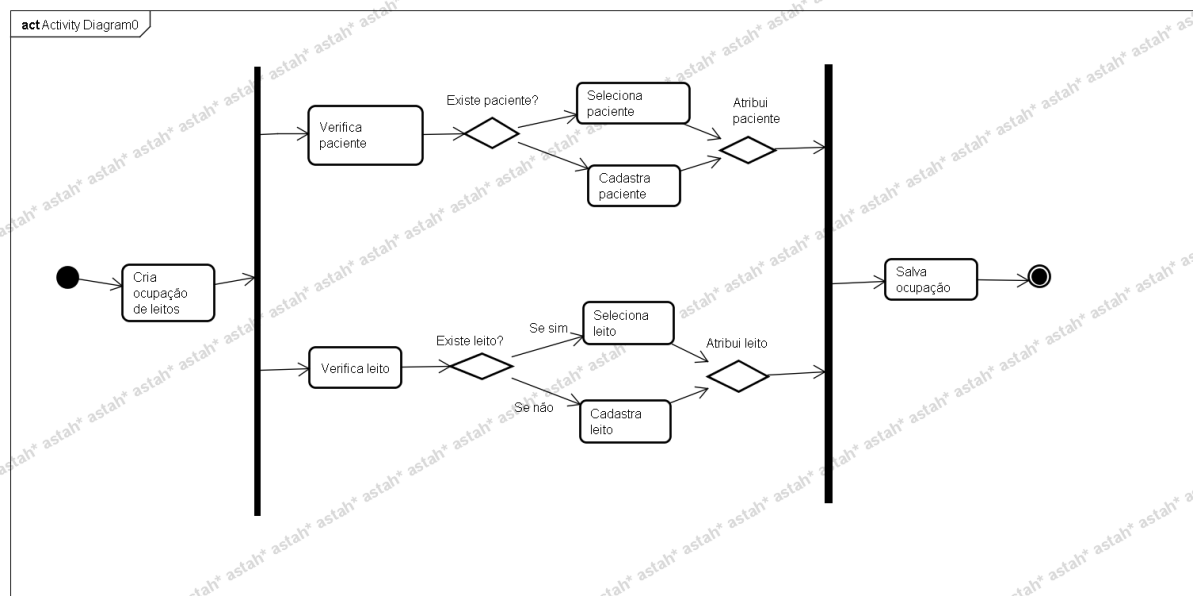
Ao realizar as atividades, temos a verificação dos leitos, cirurgias, médicos, enfermeiros e pacientes como atividades a serem concluídas e atribuídas a um procedimento cirúrgico.

5.5.5.2 Criar uma ocupação de leitos

O diagrama de atividade em questão retrata o comportamento necessário para que seja possível criar um atribuir um leito a um paciente, ou seja, processo de ocupação de um leito por um paciente, realizando todo o seu passo-a-passo para que seja bem sucedido. Para que seja possível é estipulado diversas situações para efetivar que a atividade seja concluída,

como por exemplo consultar a disponibilidade de leitos, caso não tenha leitos disponíveis, a operação se encerra.

Figura 18: Diagrama de Atividade - Atribuir leito



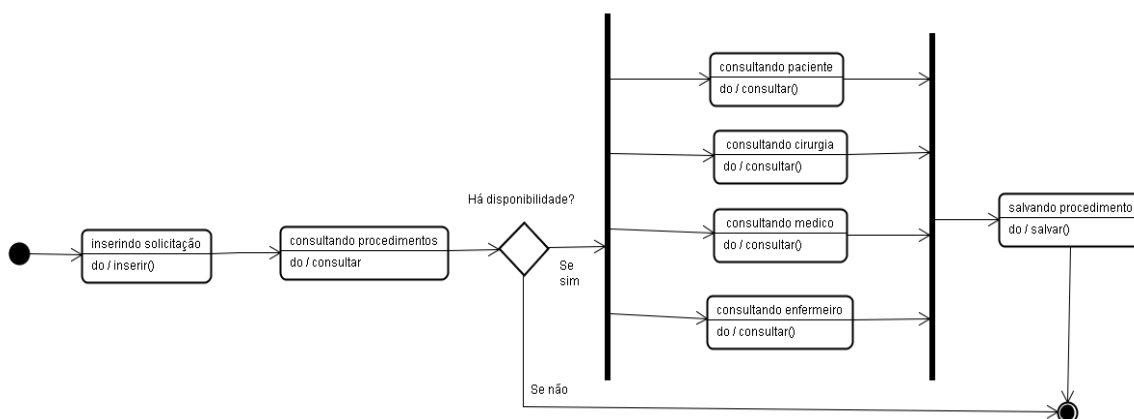
Fonte: elaborado pelo Autor.

Na Figura 19, representa um diagrama de atividade, também separado por um *fork* e feito a junção utilizando um *join*, ocorre as atividades de verificação do paciente e do leito, sendo atribuídos a uma ocupação de leito. A utilização do *fork* visa utilizar consultas em paralelos que não comprometem, na ocasião, a atribuição do leito, como por exemplo, verificar o cadastro do paciente e do leito. Quanto ao *join*, este é utilizado para fazer a junção das múltiplas consultas realizadas, para no fim concluir a atividade realizada, sendo possível, neste caso, salvar a ocupação do leito.

5.5.6 Diagrama de Máquina de Estados

Este diagrama se refere a qualquer dispositivo que armazena o status de um objeto em algum momento, podendo mudar o status e causar demais ações de acordo com as ações realizadas, indicando diferentes combinações de informações que um objeto pode conter e não como ele se comporta em si. O intuito é retratar, principalmente, estados e transições, descrevendo como um objeto se move por vários estados em seu tempo de vida e mostrar o comportamento em geral da máquina de estados ou do conjunto relacionado dos estados (LucidChart, 2023).

Figura 19: Diagrama de Máquina de Estados - Procedimentos Cirúrgicos



Fonte: elaborado pelo autor.

Na imagem acima, representada pela Figura 20, demonstra o diagrama de máquina de estados referente aos agendamentos do procedimento cirúrgico. Este modelo de diagrama é responsável por retratar os estados em que o protótipo do sistema realiza, mensurando as transições nestes estados, como por exemplo a abertura do chamado para solicitar um novo procedimento cirúrgico a ser realizado, posteriormente a verificação de datas disponíveis, consultando os procedimentos existentes e assim em diante.

6 FERRAMENTAS, SISTEMAS, TECNOLOGIAS E METODOLOGIAS ENVOLVIDAS DO PROTÓTIPO

Neste tópico, abordaremos detalhadamente as tecnologias, ferramentas, sistemas e metodologias empregadas no desenvolvimento do protótipo. Cada elemento desempenha um papel crucial na criação e implementação eficiente do projeto, contribuindo para a funcionalidade, desempenho e sucesso global do produto em desenvolvimento.

6.1 SISTEMAS E APLICAÇÕES

Um sistema pode ser aplicado em diversas áreas do conhecimento, desde ciências exatas até mesmo ciências naturais, visto que de acordo com a teoria geral dos sistemas, a definição de um sistema é: qualquer organismo formado por partes interligadas e interdependentes, mantendo a integridade do mesmo (ARAUJO, A. C. M; GOUVEIA, L. B, 2016).

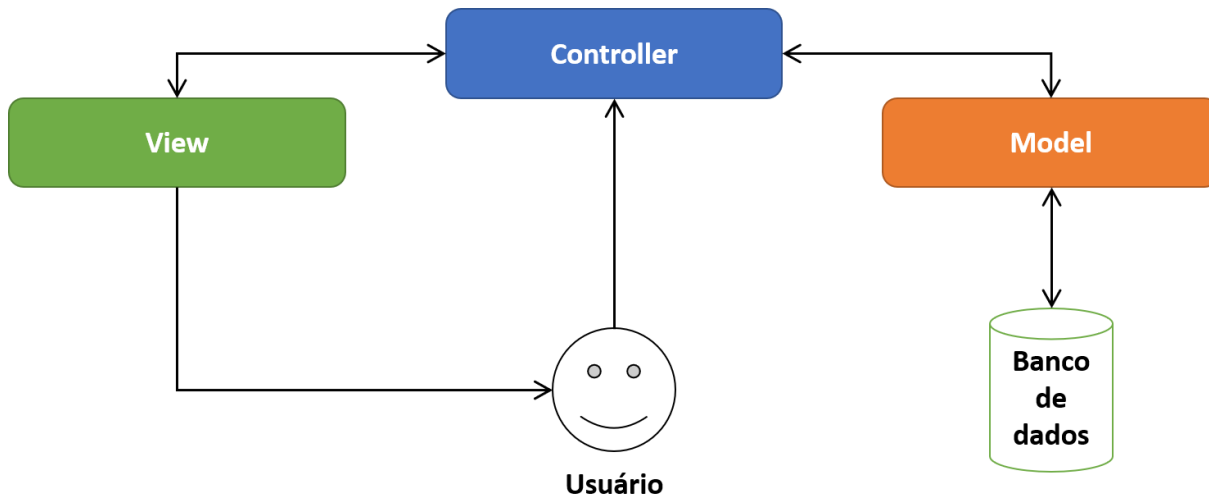
Quando citamos um Sistema, logo vem em mente o termo *Software*, que basicamente é a junção de instruções e programas que permitem o funcionamento de um equipamento em si (Monitora, 2023).

6.2 ARQUITETURA

A definição para arquitetura de software, dentre as mais comuns, se dá pela preocupação do projeto em um nível maior, deixando de lado a organização e interfaces individuais e passa a valorizar as unidades em um tamanho maior, como pacotes, componentes, módulos, camadas ou serviços (DCC/UFMG, 2023).

Outro tipo de definição, expresso por Ralph Johnson, é que arquitetura inclui as decisões mais importantes de um sistema, visto que a partir do momento em que estas forem decididas, dificilmente poderão ser revertidas no futuro, considerando então a arquitetura como um conjunto de decisões (DCC/UFMG, 2023).

Figura 20: Exemplificação da estrutura MVC



Fonte: GUEDES, 2023.

Dentre os padrões existentes, foi optado pelo uso do padrão *MVC*, que vem do acrônimo de: *Model*, *View* e *Controller*. Onde cada camada tem sua importância, visto que *Model* se refere ao negócio, acesso e manipulação dos dados, a *View* se refere a interface ao usuário e por último *Controller* se refere a camada de controle, responsável por ligar o *Model* até a *View* (GUEDES, 2023).

6.3 GIT E GITHUB

O Git é um sistema de controle de versão distribuído, desenvolvido por Linus Torvalds, também criador do Linux. Amplamente adotado, o Git permite que os colaboradores de um projeto tenham cópias completas do repositório, possibilitando o rastreamento detalhado das modificações em arquivos e facilitando a reversão a estados anteriores. Diferentemente de outros sistemas de controle de versão, o Git oferece uma abordagem descentralizada, o que contribui para sua popularidade, especialmente em projetos de código aberto. Além disso, o Git permite um histórico preciso das alterações, facilitando a identificação de bugs e a análise de otimizações (SILVEIRA, 2023).

O GitHub é uma plataforma que utiliza o Git como sistema de controle de versão, facilitando o gerenciamento e colaboração em projetos de desenvolvimento de software (SILVEIRA, 2023). Além disso, o Github hospeda mais de 100 milhões de repositórios e a partir destes repositórios é possível consultar suas versões, mantendo o registro das alterações realizadas (L. Andrei, 2023).

6.4 GITHUB COPILOT

O Github Copilot é uma inteligência artificial que oferece sugestões de preenchimento automático conforme você programa, isso facilita no processo do desenvolvedor, como em tarefas repetitivas (Copilot, Github, 2023).

Atuando como um copiloto para programadores, o GitHub Copilot funciona como um “copiloto”, oferecendo sugestões automáticas em tempo real com base em um algoritmo alimentado por uma ampla variedade de códigos abertos. Sua principal vantagem está na melhoria da eficiência da codificação, acelerando o processo de desenvolvimento. Com

suporte para diversos editores de código e linguagens de programação populares, como Python e JavaScript, o Copilot oferece versatilidade aos usuários (EDUCAÇÃO, 2022).

6.5 VISUAL STUDIO CODE

O *Visual Studio Code*, como *IDE*, é conhecido pela sua praticidade e customização, possuindo diversas extensões, as quais podemos adicionar diversas funcionalidades extras ao desenvolver. É considerado um dos editores de códigos mais utilizados no mundo, além de estar disponível no Linux, Mac e Windows (Hanashiro, Akira, 2021).

O Visual Studio Code é uma ferramenta amplamente utilizada para escrever e editar códigos de programação levando destaque devido ao extenso suporte a diversas linguagens de programação. Sua popularidade se deve não apenas à interface amigável e leve, mas também à robusta integração com sistemas de controle de versionamento, como o Git. Além disso, o Visual Studio Code oferece recursos avançados, como sugestões inteligentes de código, como o Copilot ou Intellisense, além de extensões personalizáveis, proporcionando uma experiência altamente flexível e produtiva para desenvolvedores (Walakys, 2023).

6.6 DOCKER

Docker pode ser definido como um sistema de virtualização não convencional, visto que não precisa de um software instalado na máquina *HOST* para gerenciar a máquina virtual, assim utilizando o conceito de container (Furtado, Fernando, 2017).

Além disso, o Docker oferece diversas vantagens, abrangendo portabilidade, automação e uma comunidade ativa. Em relação à portabilidade, possibilita aos usuários criar

e instalar aplicativos com facilidade, independentemente do ambiente, devido à virtualização de contêineres. Quanto à automação, simplifica a vida do desenvolvedor ao eliminar tarefas repetitivas, permitindo a criação de ambientes consistentes e prontos para execução. Além disso, o Docker possui uma comunidade grande, com vários fóruns e grupos dedicados, como por exemplo o fórum Slack e a comunidade do Stack Overflow (DANIELA, 2023).

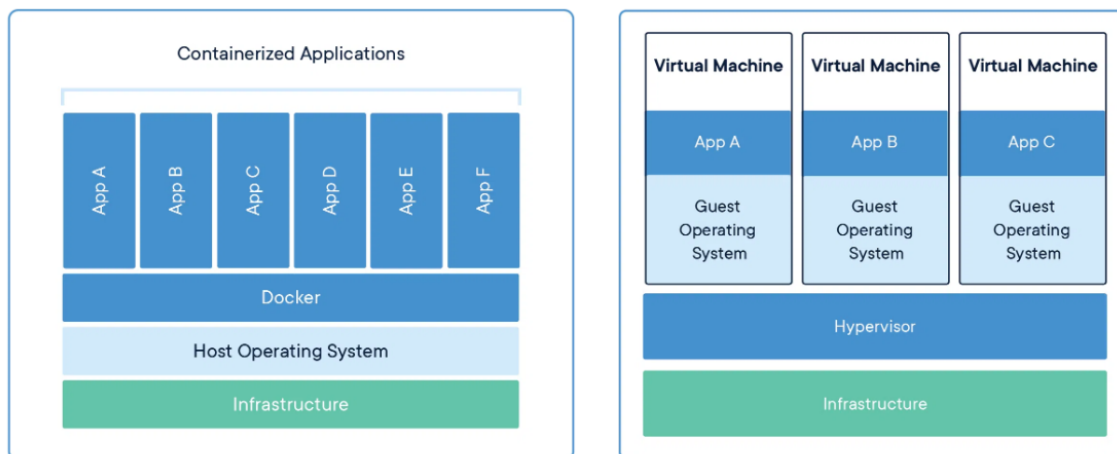
6.6.1 Container e Imagem

Um container agrega uma aplicação junto com todas as suas dependências, garantindo que o aplicativo possa ser executado em qualquer ambiente. Já a imagem do container é uma representação leve desse pacote, contendo o código e as configurações necessárias e podem ser transformadas em containers quando são executadas. A grande vantagem do uso dos container se dá em relação ao funcionamento de maneira consistente, indiferente do sistema operacional (DOCKER, 2023).

Figura 21: Comparação de uso de Contêiner e Máquinas Virtuais

Comparing Containers and Virtual Machines

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.



Fonte: DOCKER, 2023.

A utilização de contêineres e máquinas virtuais são semelhantes, desde isolamento a alocação de recursos, mas, por fim, funcionam de maneiras diferentes, visto que os contêineres virtualizam o sistema operacional, empacotando código e dependências juntos, ocupando menos espaço, geralmente megabytes, se tornando mais eficiente e portátil enquanto as máquinas virtuais abstraem do hardware, transformando um servidor em muitos porém cada máquina virtual contém uma cópia do sistema operacional e da aplicação, ocupando, geralmente, dezenas de gigabytes (DOCKER, 2023).

De acordo com a *IBM*, a grande diferença entre o uso do Docker e das máquinas virtuais, é que, a tecnologia de containers sobressai ao uso das máquinas virtuais quando citamos maior eficiência de recursos, leveza e produtividade ao desenvolver.

6.6.2 Docker Compose

Para gerenciar os containers, é utilizado o Docker Compose, que, através de um arquivo `docker-compose.yml` e a partir destas instruções é possível configurar diversos containers, instruindo a forma em que eles funcionam (Trucco, Cristian, 2019).

Ou seja, o Docker Compose é uma ferramenta essencial para definir e compartilhar aplicações que envolvem múltiplos contêineres, configurado através de um arquivo com extensão *YAML* é possível configurar os serviços da aplicação, e, com um simples comando, iniciar ou encerrar todo o ambiente. A principal vantagem do Docker Compose é a capacidade de consolidar a configuração da aplicação em um arquivo, armazenado na raiz do repositório do projeto e sujeito a controle de versão. Isso simplifica significativamente a colaboração, permitindo que outras pessoas contribuam para o projeto com facilidade, bastando clonar o repositório e iniciar a aplicação usando o Docker Compose. Essa prática é comum e eficiente, sendo adotada por muitos projetos no GitHub/GitLab (DOCKER, 2023).

6.7 BANCO DE DADOS

Originando do termo inglês *Databanks* e posteriormente trocado para *Database*, um banco de dados é considerado um conjunto de dados persistentes e armazenados com um determinado objetivo e que atende a diversos usuários ou de uma organização. As vantagens da utilização de um banco de dados para armazenamento dos dados se dá em relação ao controle centralizado, independência e a garantia de integridade dos dados em questão (COSTA, 2011).

Ainda quando falamos de banco de dados, temos dois tipos:

- Banco de dados relacional: Um banco de dados relacional é constituído por tabelas, criando relacionamento entre as próprias tabelas, por meio de chaves denominadas

primárias e estrangeiras, permitindo o uso dos dados de forma associada além de garantir o uso destes dados de forma eficiente e com integridade (CALANCA, 2016);

- Banco de dados não relacional: Um banco de dados não relacional não é constituído por tabelas e existem diversos modelos de uso de um banco não relacional, como: grafos e chave-valor, o que oferece uma flexibilidade e escalabilidade muito vantajosa quando se tem um alto número de dados (CALANCA, 2016).

Para o desenvolvimento deste projeto, atendendo aos requisitos especificados e identificado que há a necessidade da relação dos dados, foi optado pelo uso do modelo relacional dos bancos de dados, em específico o uso do *PostgreSQL*.

6.7.1 PostgreSQL

O PostgreSQL é um sistema de gerenciamento de banco de dados relacional de código aberto amplamente aclamado por sua robustez, desempenho e capacidade de extensibilidade. Destaca-se por sua aderência aos padrões *SQL* e pelo suporte a recursos avançados de banco de dados, como arrays, *JSON* e *JSONB* (POSTGRESQL, 2023).

Conhecido por sua reputação, desde a confiabilidade e flexibilidade até mesmo por aderir ao código aberto, o *PostgreSQL*, que surgiu em 1986 como "*POSTGRES*" a partir da ideia de Michael Stonebraker, vem crescendo ao longo do tempo e tem como características principais o desempenho, escalabilidade, suporte de simultaneidade e código aberto (IBM, 2023).

Neste contexto, destacamos três fundamentos que fundamentam a escolha do PostgreSQL. Primeiramente, a natureza de software livre do PostgreSQL não apenas promove acessibilidade, mas também propicia inovação e flexibilidade, eliminando custos associados a licenciamento. Em segundo lugar, a notável capacidade de escalabilidade do sistema facilita a administração eficiente de grandes volumes de dados. Por fim, o ecossistema robusto do PostgreSQL é marcado por uma comunidade engajada de colaboradores, constantemente aprimorando o sistema e oferecendo suporte, resolução de problemas e compartilhamento de

conhecimento. Esses aspectos combinados fundamentam a escolha do PostgreSQL como uma solução confiável e eficiente para aqueles que almejam excelência em gerenciamento de banco de dados (EDUCAÇÃO, 2022).

6.8 JAVASCRIPT

JavaScript, criado por Brendan Eich em 1996, é uma linguagem de programação não compilada, compatível com diversos tipos de plataformas, como web, mobile e desktop (Carlos, 2023).

O *JavaScript*, inicialmente era utilizado como um complemento para o navegador da Netscape e posterior a diversas evoluções, tornando-o mais organizado, rápido e com mais funcionalidades, o tornou uma linguagem versátil utilizada em diversas áreas, não somente restrita ao Frontend, mas também em quesito dos testes, com o uso do *Cypress* e até mesmo ao lado do servidor, com o uso do *Express*, além de ser possível realizar desenvolvimentos mobile e desktop (HANASHIRO, 2018).

JavaScript ou popularmente *JS*, é uma linguagem de programação orientada a objetos e interpretada baseada em protótipos, multi-paradigmas e dinâmica, suportando estilos de orientação a objetos, funcionais e imperativos (MOZILLA, 2023).

A utilização do *JavaScript* pode ser tanto ao lado do cliente, funcionando no navegador e sendo restrito aos recursos que o navegador permite, quanto ao lado do servidor, podendo acessar todos os recursos do servidor conforme a necessidade. A grande diferença presente entre ambas situações é a forma dinâmica em que o *JavaScript* funciona, gerando um novo conteúdo, enquanto ao lado do servidor gera um novo conteúdo dinamicamente utilizando lógica e modificando dados provenientes do banco de dados, ao lado do cliente, a geração de um novo conteúdo é utilizado dinamicamente usando lógica da interface com o usuário e sua modificação de conteúdos. Por fim, ambas abordagens funcionam em sinergia melhorando a experiência em si do usuário (AWS, 2023).

Devido a grande comunidade de desenvolvedores e com diversas bibliotecas e frameworks disponíveis para uso, facilitando o desenvolvimento como um todo e também por ser uma linguagem definida por ter uma curva de aprendizado acentuada, além da flexibilidade de uso, como já mencionado, tanto ao lado do servidor quanto ao lado da interface gráficas, com as interações dos usuários.

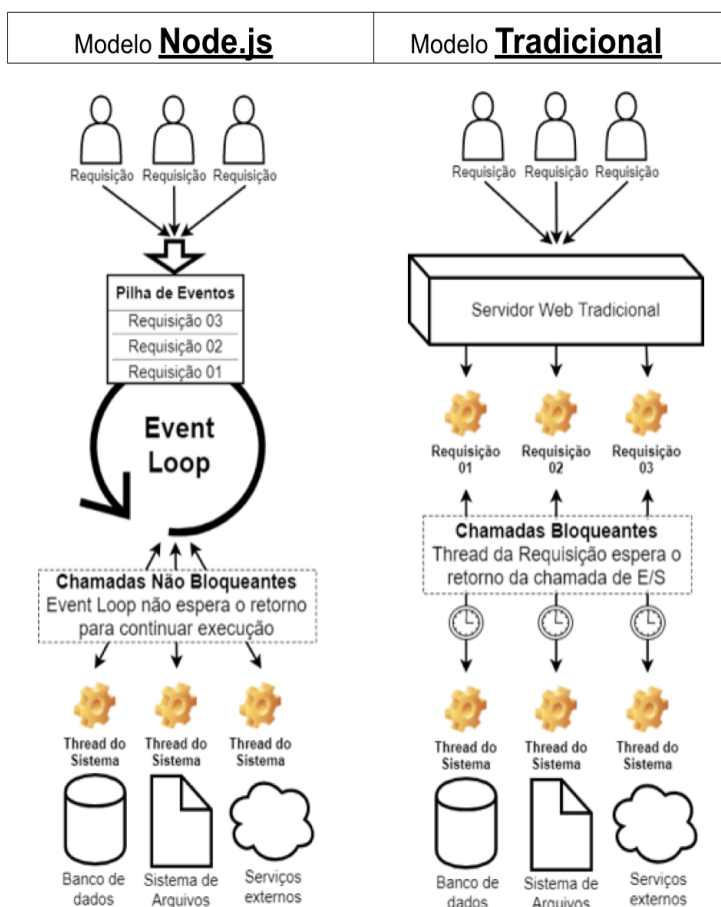
6.8.1 Node.js

Node.js é um ambiente para executar códigos JavaScript pelo lado do servidor, sendo criado em 2009 e seu intuito era justamente o uso do JavaScript para o lado do servidor sem a necessidade de instalação de ferramentas complexas (BESSA, 2023).

Utilizado geralmente para criação de APIs além de aplicações de servidores e redes escaláveis, sendo considerado uma boa solução para realização de tarefas com uso intenso de dados ou para análises em tempo real, devido a sua característica de arquitetura assíncrona orientada a eventos (Carlos, 2023).

A grande característica e marcante do Node.js, diferenciando-o dos demais, é o uso da execução single-thread, onde uma thread é responsável por executar o código JavaScript de toda a aplicação e para utilizar o efeito do multi-threads é utilizado por meio de operações assíncronas, onde não bloqueia a thread (Opus, 2018).

Figura 22: Diferença do funcionamento de um servidor Node.js para o Tradicional



Fonte: Opus, 2018.

Na Figura 22, observamos de forma clara representa exatamente a diferença entre um servidor utilizando Node.js e um servidor tradicional, ressaltando a importância e praticidade da utilização da pilha de eventos. A representação visual destaca como o servidor *Node.js* é capaz de executar tarefas de forma simultânea, otimizando a manipulação de requisições e promovendo um desempenho mais ágil. Essa comparação visual ressalta a vantagem fundamental do modelo de servidor *Node.js*, enfatizando sua capacidade de lidar eficazmente com operações concorrentes, resultando em servidores mais eficientes e adaptáveis.

O Node.js, um ambiente de execução de programas assíncrono e orientado a eventos em JavaScript, foi criado para o desenvolvimento de aplicações de rede escaláveis ao lado do servidor. Seu intuito é que várias conexões possam ser processadas simultaneamente onde a cada nova conexão, a função de retorno é ativada. No entanto, se não houver tarefa pendente, o Node.js entra em modo de espera (NODE, 2023).

Figura 23: Exemplificação de uso do Node.js

```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Fonte: NODE, 2023.

Na Figura 23, representada acima, temos um exemplo disponibilizado na documentação do Node onde trata-se de permitir várias conexões simultâneas e onde cada chamada de conexão, é ativada a função de retorno e em caso que não há chamadas, entrará em modo de espera.

6.8.2 Express.js

O *Express.js* é um framework rápido e um dos mais utilizados junto ao *Node*, o Express é um facilitador no processo de desenvolvimento da *API*, sendo escrito em *JavaScript* e com as características voltadas ao tratamento de exceções, gerenciamento diferente de requisições *HTTP* e o tratamento completo das rotas (ANDRADE, 2021).

Um *framework* do *Node.js*, o Express, que fornece um conjunto sólido de recursos para desenvolvimento de aplicações, seu intuito é a criação de *APIs* de uma forma concreta e fácil, fornecendo camadas de recursos fundamentais para a aplicação (EXPRESS, 2023). Sendo considerado o *framework Node* mais popular, o express oferece diversas soluções para gerenciar requisições, adicionar novos processos por meio de *middlewares* (MOZILLA, 2023).

6.9 BCrypt

O BCrypt foi criado por Niels Provos e David Mazières em 1999 com o objetivo de proteger as senhas dos usuários, transformando-as em dados indecifráveis por meio de um algoritmo de hash (KERLYN, 2019).

Este modelo de criptografia ficou popularmente conhecido no desenvolvimento back-end devido a robustez dos recursos de segurança e também a sua adaptabilidade a diversas plataformas, empregando uma combinação denominada única por meio de procedimentos como hashing adaptativo, salting e técnicas de fortalecimento de chave afins de proteger as informações confidenciais contra acessos não autorizados e violações de dados (APPMASER, 2023).

6.10 JSON WEB TOKEN

Json Web Token é um padrão para autenticar e trocar informações definido pela *RFC7519*, sendo possível, de forma segura e compacta, o armazenamento de objetos em

JSON, código base64 e assinado por um ou mais chaves públicas ou privadas (SEGUINS, 2022).

Para sua utilização, o *JWT* de forma assinada, existe dois modelos: usando criptografia ou por meio de um par de chaves públicas e privadas, isso faz com que, o cenário mais comum para seu uso é no quesito de autorização, onde após a solicitação, cada solicitação incluirá um token *JWT*, permitindo então que o usuário acesse determinadas rotas e recursos, além de conter uma estrutura separada em três camadas: *Header*, *Payload* e *Signature* (JWT, 2023).

Figura 24: Exemplificação de uso do JWT



Fonte: JWT, 2023.

A imagem representada pela Figura 24, retrata como um *JWT* é obtido para acessar recursos ou *APIs*, onde o aplicativo ou cliente solicita autorização ao servidor e quando a autorização é concedida, o servidor de autorização retorna um token de acesso ao aplicativo e esse token é utilizado pela aplicação para acessar um recurso considerado protegido, como a própria *API*.

6.11 POSTMAN

Postman é uma ferramenta utilizada para auxiliar nos testes executados chamando a API em questão, trabalhando de modo mais eficaz, construindo pastas para organização das rotas e chamadas, além de cada método necessário (VERSIANI, 2023).

O postman é uma ferramenta abrangente que oferece a capacidade de armazenar, catalogar e colaborar em torno de todos os elementos relacionados às APIs, desde especificações e documentação até casos de teste e métricas. Com uma variedade de ferramentas que abrangem design, teste, documentação e simulação, o Postman facilita o compartilhamento e a descoberta de APIs. Sua abordagem abrangente para governança em todo o ciclo de vida promove práticas de desenvolvimento de alta qualidade e colaboração entre equipes de desenvolvedores e design de API. Os espaços de trabalho do Postman oferecem organização e colaboração eficientes em diferentes níveis, adaptando-se às necessidades pessoais, de equipe, parceiros e espaços de trabalho públicos. Além disso, a integração com ferramentas-chave no pipeline de desenvolvimento de software e a extensibilidade através da API e tecnologias de código aberto destacam o Postman como uma solução versátil e poderosa para a construção e uso de APIs de forma eficaz (POSTMAN, 2023).

6.12 SWAGGER

Swagger é um software criado em 2011, constituído por um conjunto de ferramentas capaz de criar, projetar e documentar uma API (MARTINS, 2022).

Considerado uma abrangente coleção de ferramentas de código aberto projetadas em torno da especificação *OpenAPI*, focadas em facilitar o design, construção, documentação e consumo de *APIs REST*. Essas ferramentas incluem o Swagger Editor, um editor baseado em navegador para definições *OpenAPI*, o Swagger UI, que renderiza definições *OpenAPI* como

documentação interativa, o Swagger Codegen, capaz de gerar stubs de servidor e bibliotecas de cliente a partir de definições *OpenAPI*, e diversas outras bibliotecas e utilitários como o Swagger Parser e Swagger APIDom. A principal vantagem do uso do *Swagger* é a capacidade das *APIs* de descreverem sua própria estrutura, possibilitando impulsionar eficientemente o desenvolvimento da *API* após a escrita da especificação (SWAGGER, 2023).

6.13 VUE.JS

Vue é um framework e ecossistema que oferece suporte à maioria dos recursos comuns necessários no desenvolvimento frontend. Adaptando-se à diversidade da web, o *Vue* foi projetado para ser flexível e adaptável de forma incremental. Essa abordagem permite que ele seja utilizado de diversas maneiras, desde aprimorar *HTML* estático sem etapas de compilação até incorporar-se como Componentes da Web em qualquer página. Além disso, o Vue é aplicável em contextos como *Single-Page Application*, Fullstack com Renderização do Lado do Servidor, arquiteturas Jamstack com Geração de Sites Estáticos, e pode ser direcionado para ambientes desktop, mobile e até mesmo o terminal. Independentemente do caso de uso, o Vue oferece flexibilidade e facilidade de adoção, sendo acessível para desenvolvedores com conhecimentos básicos em *HTML* e *JavaScript* (VUE, 2023).

Vue é um dos mais populares frameworks JavaScript do mundo, voltado ao construções de interfaces ao usuário, lançado em 2014 por Evan You, este framework conta com a versatilidade e facilidade em aprendizado, sendo cada vez mais popular (PICOLLO, 2019).

Baseado em *HTML*, *CSS* e *JavaScript*, oferece um modelo de programação baseado em componentização, auxiliando no desenvolvimento eficiente de interfaces de usuário, sendo elas simples ou complexas. Suas principais características são em relação a renderização declarativa e a reatividade, onde a primeira estende o *HTML* convencional incorporando uma sintaxe de modelo permitindo que os seja descrita de forma expressiva a saída *HTML*, alinhando-a com o estado *JavaScript*, tendo como resultado, o código torna-se mais intuitivo

e mais fácil de manter, já a reatividade é responsável por monitorar autonomamente as alterações no estado JavaScript, garantindo uma interface do usuário dinâmica e responsiva e quando ocorre, o Vue se torna responsável por atualizar o *DOM* de forma eficiente (VUE, 2023).

6.14 VUETIFY

Vuetify, criado em 2016 por John Leider, é uma biblioteca de componentes do Vue.js, ou seja, serve como uma coleção de componentes personalizáveis e reutilizáveis que podem ser utilizados ao criar uma interface ao usuário (SILVA, 2023).

Vuetify destaca-se como um robusto Framework de Componentes Vue, projetado para facilitar a aprendizagem e proporcionar uma experiência gratificante ao usuário, onde sua coleção de componentes de interface mantém um estilo consistente, com amplas opções de personalização para atender a diversas necessidades. Além de ser gratuito e de código aberto sob a licença MIT, a flexibilidade de seus componentes, alinhados com a especificação de Design Material do Google, oferece aos desenvolvedores a capacidade de criar aplicações Vue concisas e visualmente atraentes (VUETIFY, 2023).

6.15 VUE ROUTER

Vue Router é o roteador oficial para Vue.js. Ele se integra profundamente ao núcleo do Vue.js para tornar a construção de aplicações de página única com Vue.js muito mais simples. Suas características incluem o mapeamento de rotas, roteamento dinâmico, configuração baseada em componentes, parâmetros de rota, consulta, efeitos de transição de visualização

impulsionados pelo sistema de transição do Vue.js, controle de navegação refinado, links com classes *CSS* ativas automáticas, modo de histórico *HTML* ou modo de hash, comportamento de rolagem personalizável e codificação adequada para *URLs*. Vue Router fornece uma solução abrangente para a navegação em aplicativos Vue.js, facilitando a implementação de roteamento avançado e oferecendo controle preciso sobre a experiência do usuário (ROUTER, 2023).

Permitindo o gerenciamento das rotas e como os usuários navegam entre si, além de informações da rota atual, o Vue.js disponibiliza o uso do Vue Router, responsável por este gerenciamento de forma íntegra (JAVASCRIPT, 2023).

6.16 AXIOS

Axios é um cliente HTTP baseado em promises para o node e para o navegador, sendo considerado isomórfico, ou seja, pode rodar tanto no node.js quanto no navegador com a mesma base de código. Suas características são: Faz requisições HTTP, suporte a promises API, intercepta requisições e respostas, automaticamente transformando os dados para JSON, além da possibilidade de cancelar as requisições (AXIOS, 2023).

O Axios é uma escolha amplamente adotada para facilitar a comunicação de dados entre o *frontend* e o *backend* em aplicações web. Esta biblioteca oferece uma API simples e flexível, permitindo a realização de chamadas de API assíncronas. Essas chamadas podem incluir a obtenção de dados de uma API, o envio de dados para um servidor ou a atualização de informações em tempo real. Com a simplicidade e eficácia do Axios, torna-se mais conveniente implementar interações dinâmicas e atualizações de dados em aplicações web (AWARI, 2023).

7 DESENVOLVIMENTO DO PROJETO

Nesta seção do projeto, iremos explorar profundamente a criação do protótipo de software, levando em consideração as tecnologias anteriormente elencadas. Após uma análise aprofundada dessas tecnologias, nosso foco estará em como essas ferramentas desempenharam um papel essencial na concepção, arquitetura e implementação do software, transformando o estudo em prática.

Esta parte foi dividida em três ocasiões, sendo elas: Planejamento e Organização, *Backend* e *Frontend*, onde cada tópico contém, respectivamente, sua responsabilidade e contribuição no desenvolvimento do protótipo.

7.1 PLANEJAMENTO E ORGANIZAÇÃO

Nesta etapa abordaremos os passos designados antes do processo de desenvolvimento em si da aplicação, abordando tópicos de planejamentos e organização do projeto como sua prototipação e versionamento de códigos.

7.1.1 Prototipação no Figma

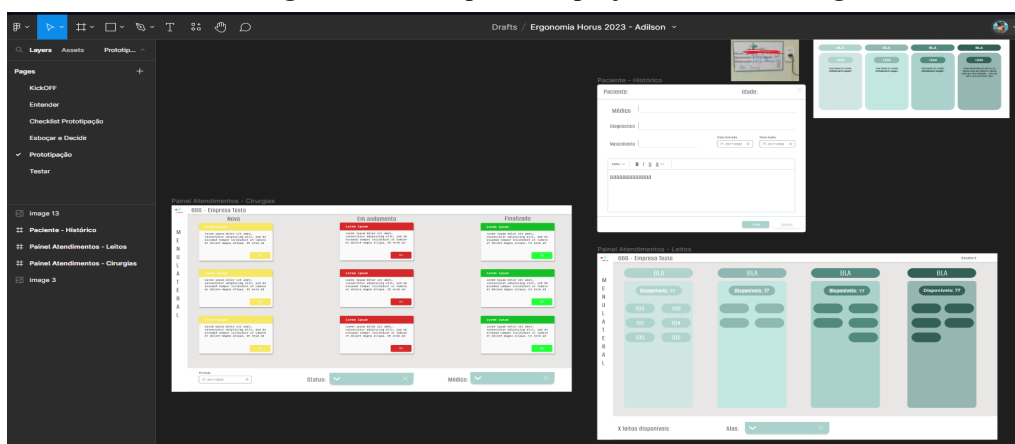
Nesta etapa, exploraremos a aplicação prática do Figma e o processo de prototipação realizado. A prototipação realizada no Figma não apenas facilita a visualização das funcionalidades, fluxos e interações previstas, mas também serve como um instrumento valioso para refinamento contínuo e validação do design antes da implementação. Este

processo dinâmico no Figma representa uma etapa crucial no desenvolvimento, promovendo a comunicação eficaz e a convergência para soluções visualmente impactantes e funcionalmente sólidas.

O termo “prototipar” se refere ao processo de criar uma versão inicial ou representação palpável de uma ideia ou conceito, com o propósito de avaliá-la, validando sua viabilidade e obtendo feedback dos usuários antes de alocar recursos para sua implementação final. Essa prática abrange representações tanto físicas quanto digitais do produto ou serviço, indo desde simples esboços até modelos complexos e funcionais. Essa abordagem possibilita que equipes de projeto testem e aprimorem suas ideias, identificando potenciais problemas, ajustando detalhes e refinando a solução antes de sua implementação definitiva (MOSHE, 2023).

Diante deste cenário, o Figma é uma plataforma comumente utilizada para uma prática coletiva de desenvolvimento de interfaces e protótipos, que foi ao ar em 2016 por Dylan Field e Evan Wallace com o objetivo de ser uma ferramenta gratuita com a ambição da busca de colaboração entre os times (SILVEIRA, 2023).

Figura 25: Prototipação do projeto realizada no Figma



Fonte: efetuado pelo autor.

Representado na Figura 25, temos a prototipação realizada na plataforma Figma referente às telas iniciais de cadastros, controle de leitos por alas além do modelo Kanban. Este passo foi essencial para entender e estudar ainda mais como seria a experiência do usuário ao utilizar o sistema.

7.1.2 Versionamento de Código

Antes de iniciar o desenvolvimento do projeto, realizamos uma organização cuidadosa do ambiente de trabalho. Para isso, adotamos a prática de armazenar todo o progresso na nuvem. Essa abordagem garante que, em casos de eventualidades, nenhum trabalho seja perdido. Para atender a essa necessidade, optamos por utilizar o software de versionamento Git e a plataforma GitHub.

Figura 26: Repositório do protótipo no GitHub



Fonte: elaborado pelo autor.

Dessa forma, conforme representado através da Figura 26, já configuramos os repositórios no GitHub, o que possibilita o desenvolvimento em qualquer máquina, mantendo sempre a versão mais atualizada. Além disso, essa prática nos permite preservar um histórico detalhado de todas as alterações realizadas ao longo do projeto.

7.2 BACKEND

Considerado a ponte entre os dados do próprio navegador e do banco de dados, a definição de *backend* consiste na ideia de tudo que há por trás de uma aplicação, sendo constituído por toda a regra de negócio e validações (SOUTO, 2023).

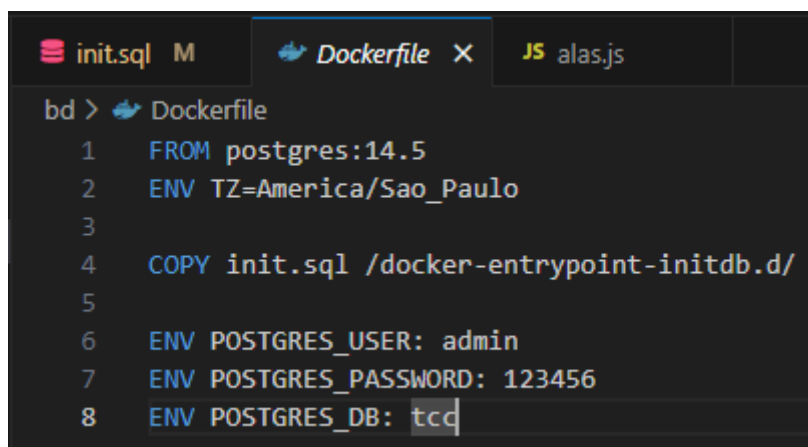
O desenvolvimento *backend* consiste na construção da parte "não visível" da aplicação, que lida com a regra de negócios, processamento de dados, interações com o banco de dados, controle e tratamento das requisições e outras funcionalidades que não são diretamente visíveis aos usuários finais.

Neste tópico abordaremos as configurações e estruturas necessárias para o desenvolvimento em si do *backend*, como as configurações utilizadas no Docker, estrutura do banco, da api e as pastas utilizadas.

7.2.1 Configurações do Docker

Para uso do Docker no protótipo, a exemplo da imagem 26, foi necessário a criação de um arquivo denominado Dockerfile, utilizado tanto no banco de dados para o *Postgres* quanto para a api, no *Node*. Este arquivo é composto inicialmente utilizando o *from* que indica a imagem a ser utilizada no Docker, que neste caso foi o *postgres:14.5* e o *node:18-alpine* além de demais informações essenciais como as credenciais do banco de dados, por exemplo.

Figura 27: Dockerfile Postgres presente no protótipo

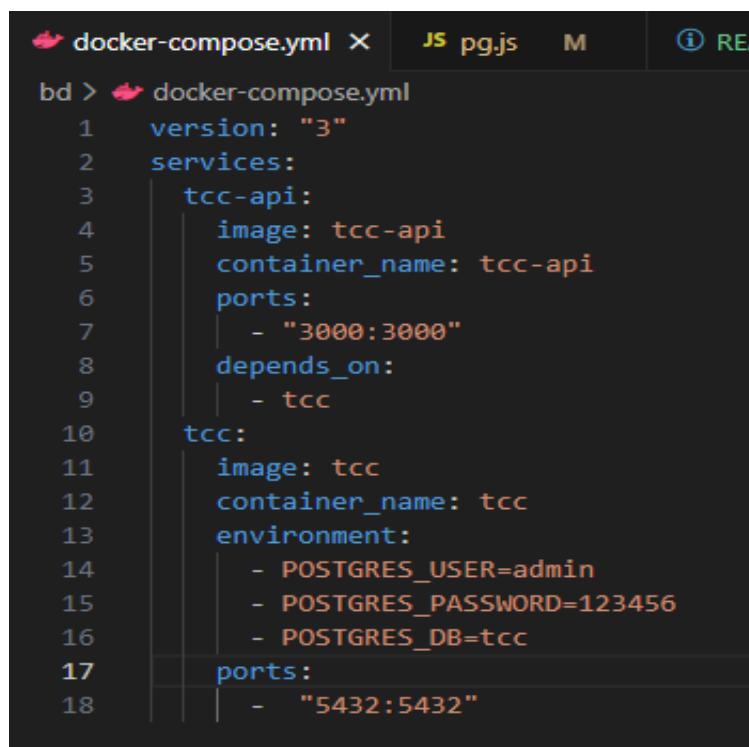


```
bd > Dockerfile
1 FROM postgres:14.5
2 ENV TZ=America/Sao_Paulo
3
4 COPY init.sql /docker-entrypoint-initdb.d/
5
6 ENV POSTGRES_USER: admin
7 ENV POSTGRES_PASSWORD: 123456
8 ENV POSTGRES_DB: tcc
```

Fonte: elaborado pelo Autor.

Através deste arquivo, conforme a imagem abaixo, estão as configurações dos containers, tanto do banco de dados quanto da API, além das portas utilizadas e nome da imagem, onde é necessário ter o vínculo com o arquivo Dockerfile previamente criado.

Figura 28: Uso do docker compose entre a API e o Banco de Dados no projeto



```
bd > docker-compose.yml
1 version: "3"
2 services:
3   tcc-api:
4     image: tcc-api
5     container_name: tcc-api
6     ports:
7       - "3000:3000"
8     depends_on:
9       - tcc
10  tcc:
11    image: tcc
12    container_name: tcc
13    environment:
14      - POSTGRES_USER=admin
15      - POSTGRES_PASSWORD=123456
16      - POSTGRES_DB=tcc
17    ports:
18      - "5432:5432"
```

Fonte: elaborado pelo autor.

Após ter configurado o ambiente do docker-compose, é necessário criar as imagens, dentro da pasta do backend, que podem ser utilizadas pelos seguintes comandos no terminal, onde o primeiro se refere a api e o segundo ao banco de dados:

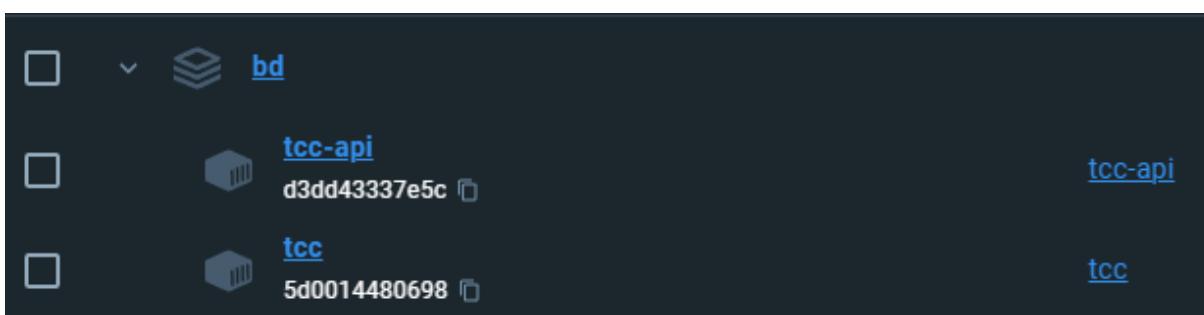
- `docker build -t tcc-api .`
- `docker build -t tcc .`

Posteriormente da criação de ambas imagens, é necessário rodar o último comando:

- `docker compose up -d`

Este último comando faz com que gere o container com a API e o Banco de dados.

Figura 29: Docker-compose no protótipo



Fonte: elaborado pelo Autor.

Após todos os comandos elencados e utilizados, é possível identificar, através da Figura 28, a utilização do docker-compose, onde um container contém duas imagens representadas por “tcc-api” que se refere ao uso do *Node* e “tcc” que se refere ao uso do *Postgres*.

7.2.2 Estrutura do Banco de Dados

Seguindo com base na estrutura dos diagramas, foram criadas as tabelas no banco de dados. Essas tabelas criadas no *PostgreSQL* são integradas juntamente com a API, pois será onde irá manipular os dados de acordo com o tipo, como consulta, inserção, edição ou deleção.

Para criação das tabelas foi definido um arquivo denominado *init.sql*, onde está presente todas as tabelas, chaves primárias e estrangeiras para que quando criado o docker, automaticamente crie as tabelas no *PostgreSQL*.

Figura 30: Exemplificação do uso do Postgres: criar tabelas

```
create table if not exists procedimentos_cirurgicos (
  id serial primary key,
  data_cirurgia timestampz not null,
  tempo_recuperacao int not null,
  paciente int not null,
  cirurgia int not null,
  medico int not null,
  enfermeiro_responsavel int not null,
  id_leito int not null,
  inativo boolean default false,
  status varchar(100),
  constraint fk_paciente_to_procedimentos_cirurgicos foreign key (paciente) references pacientes (id),
  constraint fk_cirurgia_to_procedimentos_cirurgicos foreign key (cirurgia) references cirurgias (id),
  constraint fk_medico_to_procedimentos_cirurgicos foreign key (medico) references medicos (id),
  constraint fk_enfermeiro_responsavel_to_procedimentos_cirurgicos foreign key (enfermeiro_responsavel) references enfermeiros (id)
  constraint fk_id_leito_to_procedimentos_cirurgicos foreign key (id_leito) references leitos (id)
);

create table if not exists leitos_ocupacoes (
  id serial primary key,
  id_paciente int not null,
  id_leito int not null,
  id_procedimento_cirurgico int not null,
  id_enfermeiro_responsavel int not null,
  observacao varchar(250),
  inativo boolean default false,
  constraint fk_id_paciente_to_leitos_ocupacao foreign key (id_paciente) references pacientes (id),
  constraint fk_id_leito_to_leitos_ocupacao foreign key (id_leito) references leitos (id),
  constraint fk_id_procedimento_cirurgico_to_leitos_ocupacao foreign key (id_procedimento_cirurgico) references procedimentos_cirurgicos (id),
  constraint fk_id_enfermeiro_responsavel_to_leitos_ocupacao foreign key (id_enfermeiro_responsavel) references enfermeiros (id)
);
```

Fonte: elaborado pelo autor.

Através da representação da Figura 30 temos duas tabelas, de exemplo, criadas no banco de dados, se referindo aos procedimentos cirúrgicos e as ocupações dos leitos, utilizando conceitos e boas práticas de programação ao criar as tabelas.

7.2.3 Desenvolvimento da API

No *backend* foi onde foi desenvolvido a *API* e a conexão dela com o banco de dados *PostgreSQL* para posteriormente ser possível manipular os dados através de requisições. Neste ponto foi aplicado todas as tecnologias abordadas até então, além de boas práticas de programação, visando organização e padronização no código.

No desenvolvimento da *API*, foi instalado diversos pacotes com diversos intuítos diferentes, citados nos tópicos abaixo, sendo instalado a partir dos comandos:

- `npm install jsonwebtoken`: responsável pela instalação do pacote do *Json Web Token*;
- `npm i express`: responsável pela instalação do pacote do *Express*, para desenvolvimento da *API*;
- `npm install bcryptjs`: responsável pela instalação do pacote de criptografia, utilizado nas senhas ao desenvolver a *api*;
- `npm i swagger-ui-express`: responsável pela instalação do pacote do *swagger*, afins de documentação dos *endpoints*;
- `npm i swagger-autogen`: responsável pela instalação do pacote que gera automaticamente a documentação através dos *endpoints* criados.

Figura 31: Exemplificação do uso do Postgres: criar tabelas

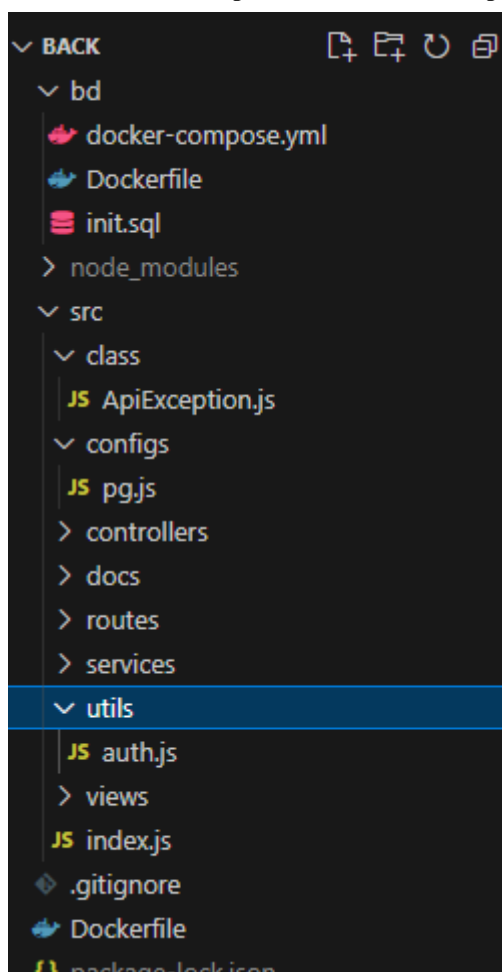
```
license : ISC ,
"dependencies": {
  "bcryptjs": "^2.4.3",
  "cors": "^2.8.5",
  "express": "^4.18.2",
  "jsonwebtoken": "^9.0.1",
  "pg": "^8.11.1",
  "swagger-autogen": "^2.23.7",
  "swagger-ui-express": "^5.0.0"
},
"devDependencies": {
```

Fonte: elaborado pelo autor.

Cada comando citado, representa um pacote instalado e que foi utilizado no desenvolvimento do protótipo, podendo também ser consultado através do arquivo *package-lock*, representado pela Figura 31.

A figura a seguir demonstra toda a estrutura de pastas criadas no backend, onde temos a divisão entre duas pastas: *BD* e *SRC*.

Figura 32: Estrutura das pastas no Backend do protótipo



Fonte: efetuado pelo autor.

Na pasta denominada BD temos o arquivo Dockerfile que se refere a criação do container da imagem do banco *Postgresql*, no docker-compose temos a composição do uso tanto do container do banco de dados quanto do node, fazendo o uso do Docker Compose e por último o init, que é o responsável pela criação das tabelas e ligações do banco de dados.

Na pasta denominada SRC temos diversos arquivos, dentre os principais temos:

- Class: onde contém um arquivo que se refere a uma classe responsável por retornar exceções das chamadas a API de forma padronizada;
- Configs: pasta responsável por arquivos que contém a conexão com o banco de dados;
- Controllers: pasta responsável pelos arquivos de controle dos retornos das chamadas da API além dos parâmetros de entrada;
- Docs e Views: pastas utilizadas para configurações e uso do Swagger;
- Routes: pasta responsável por conter arquivos destinados ao redirecionamento proveniente da chamada da api;
- Services: pasta responsável pelos arquivos que contém as regras de negócio e manipulação dos dados;

Figura 33: Exemplo de um POST (inserção) na API - Services

```
const db = require('../configs/pg')
const postCirurgia = async (params) => {
  const sqlInsert = `insert into cirurgias (nome, observacao, id_grupo) values ($1, $2, $3)`
  const { nome, observacao, id_grupo } = params
  await db.query(sqlInsert, [nome, observacao, id_grupo])
}
```

Fonte: efetuado pelo autor.

Figura 34: Exemplo de um POST (inserção) na API - Controller

```
const cirurgiasService = require('../services/cirurgias')
const postCirurgia = async (req, res, next) => {
  try {
    await cirurgiasService.postCirurgia(req.body)
      .then(ret => res.status(201).send(ret))
      .catch(err => res.status(500).send(err))
  } catch (err) {
    res.status(500).json(err);
  }
}
```

Fonte: efetuado pelo autor.

Figura 35: Exemplo de um POST (inserção) na API - Routes

```
routes > ./cirurgias.js > <unknown> > exports
const cirurgiasController = require('../controllers/cirurgias');
module.exports = (app) => {
  app.post('/cirurgia', cirurgiasController.postCirurgia)
  app.get('/cirurgia', cirurgiasController.getCirurgias)
  app.get('/allCirurgias', cirurgiasController.getAllCirurgias)
  app.get('/getCirurgiaById/:id', cirurgiasController.getCirurgiaById)
  app.delete('/cirurgia/:id', cirurgiasController.deleteCirurgia)
  app.put('/cirurgia/:id', cirurgiasController.putCirurgia)
  app.patch('/cirurgia/:id', cirurgiasController.patchCirurgia)
}
```

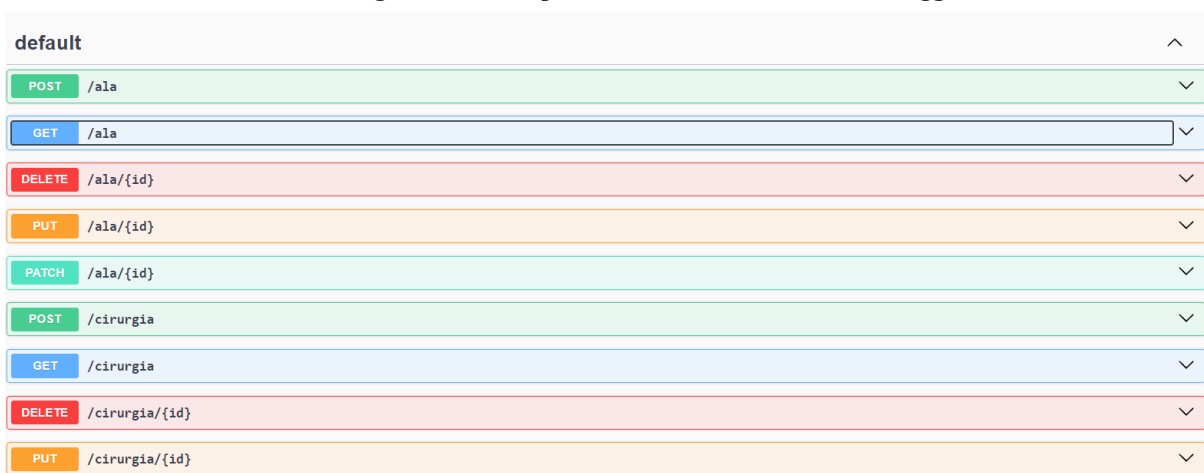
Fonte: efetuado pelo autor.

E nas demais figuras, representadas pelas Figuras 33, 34 e 35, é exemplificado o desenvolvimento da *API*, responsável pela inserção dos dados de uma cirurgia na tabela de “cirurgias” no banco de dados *PostgreSQL*. A primeira imagem sendo a indicação do service responsável pela manipulação dos dados, conexão com o banco e das regras de negócio, seguido do controller que realiza o tratamento os parâmetros de entrada e retorno da chamada e por fim a routes, responsável pelo redirecionamento dos métodos para posteriormente ir para o controller.

7.2.4 Documentação utilizando Swagger

O uso do Swagger segue no conceito de manter as boas práticas de programação, neste caso de organização, agora documentando a *API*, visto que a *API* em questão pode ser utilizada por outras integrações e a partir disso, é necessário uma descrição do que é capaz, de qual forma deve ser utilizada e o que faz cada requisição disponibilizada.

Figura 36: Exemplo de documentação usando swagger



The image shows a Swagger API documentation interface. At the top, there is a tab labeled 'default' with an upward-pointing arrow. Below the tab, there is a list of API endpoints, each with a colored header indicating the HTTP method and a dropdown arrow on the right. The endpoints are:

Method	Endpoint
POST	/ala
GET	/ala
DELETE	/ala/{id}
PUT	/ala/{id}
PATCH	/ala/{id}
POST	/cirurgia
GET	/cirurgia
DELETE	/cirurgia/{id}
PUT	/cirurgia/{id}

Fonte: efetuado pelo autor.

Com isso, conforme a Figura 36, acima, o Swagger nos auxilia diretamente permitindo uma documentação rápida de todos os pontos disponíveis da API em um local só, criando uma interface com todos os métodos e parâmetros.

7.3 FRONTEND

Definido por ser a parte visual de uma aplicação, *frontend* é a parte responsável pela interface gráfica em que o usuário consegue interagir (SOUTO, 2023).

O *frontend* é a parte visível e interativa de uma aplicação web, representando a interface com a qual os usuários interagem diretamente. Composto por tecnologias como *HTML*, responsável pela estruturação da página, o *CSS*, responsável pela estilização da página e por último *JavaScript*, responsável pela interatividade e dinamismo na página. O *frontend* molda a apresentação e o layout das páginas, garantindo uma experiência visualmente atraente e intuitiva para os usuários. Essa camada é essencial para a comunicação eficiente de informações com a *API* já criada, recebendo informações da mesma.

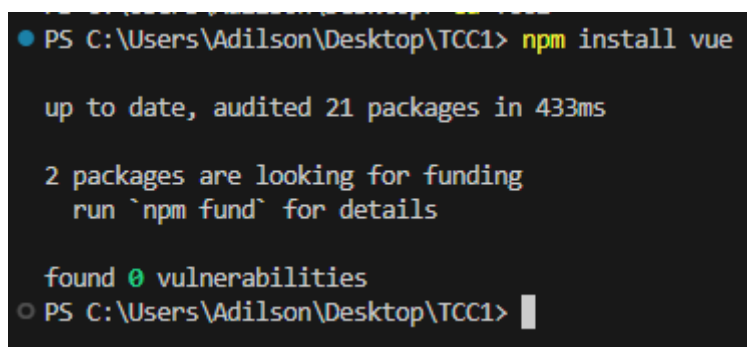
7.3.1 Instalação do Vue

A utilização do Vue no projeto serve como um facilitador e segue os mesmos conceitos de *HTML*, *CSS* e JavaScript padrão, porém com outras nomenclaturas, que podem ser separadas por seções, que são:

- **Template:** A parte do componente que define a estrutura e a apresentação visual, onde contém o *HTML* apresentado ao usuário;
- **Script:** Considerado seção onde a lógica do componente é definida usando JavaScript, onde é definido a lógica de dados, métodos e outros comportamentos do componente;
- **Style:** A parte que lida com a estilização do componente, contendo o CSS para modificar e estilizar conforme a necessidade.

Para iniciarmos, é necessário utilizar o comando de instalação do vue, representado pelo comando: `npm install vue`;

Figura 37: Instalação do Vue



```
PS C:\Users\Adilson\Desktop\TCC1> npm install vue
up to date, audited 21 packages in 433ms
2 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\Adilson\Desktop\TCC1>
```

Fonte: efetuado pelo autor.

Posteriormente a instalação do vue, pode ser criado um projeto com o comando:

- `vue create <nome>`

Figura 38: Exemplo de configuração ao criar um projeto Vue

```
Vue CLI v5.0.8
? Please pick a preset: (Use arrow keys)
> Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
  Manually select features
```

Fonte: efetuado pelo autor.

Ao utilizar o comando de criação, automaticamente será possível seguir os próximos passos para configurações do projeto, selecionando, por exemplo, a versão do Vue a ser utilizada, conforme exemplificado na Figura 38.

7.3.2 Configuração do Vuetify

A decisão de incorporar o Vuetify ao protótipo foi motivada pelas inúmeras oportunidades e facilidades oferecidas por seus componentes prontos para uso. Essa escolha permitirá uma implementação mais eficiente e ágil, aproveitando a variedade de elementos visuais e funcionais fornecidos pelo Vuetify, o que contribuirá para aprimorar a experiência do usuário no protótipo.

Para realizar a instalação, conforme disponibilizada na própria documentação do Vuetify, basta apenas utilizar o comando: `npm create vuetify`.

Figura 39: Instalação do Vuetify no projeto

```
To get started with Vuetify 3, simply paste the following code into your terminal:
```

```
yarn  npm  pnpm  bun
```

```
npm create vuetify
```

```
This command prompts you with a few options before generating your scaffolded Vue / Vuetify 3 project.
```

```
success Installed "create-vuetify@x.x.x" with binaries:
  - create-vuetify

? Project name: > vuetify-project // the folder to generate your application
? Use TypeScript?: > No / Yes
? Would you like to install dependencies with yarn, npm, or pnpm?:
  > yarn
    npm
    pnpm
    bun
    none
```

Fonte: Vuetify, 2023.

Este comando, representado através da Figura 39, disponibilizará as funcionalidades e irá, em forma de sugestão, declarar quais configurações pré-configuradas é necessário para instalação do *framework*.

Figura 40: Configuração pré-configuradas do Vuetify

```
PS C:\Users\Adilson\Desktop\TCC\Back> npm create vuetify
```

```
Vuetify.js - Material Component Framework for Vue
```

```
✓ Project name: ... tttttttttt
```

```
? Which preset would you like to install? » - Use arrow-keys. Return to submit.
```

```
  Default (Vuetify)
```

```
> Base (Vuetify, VueRouter)
```

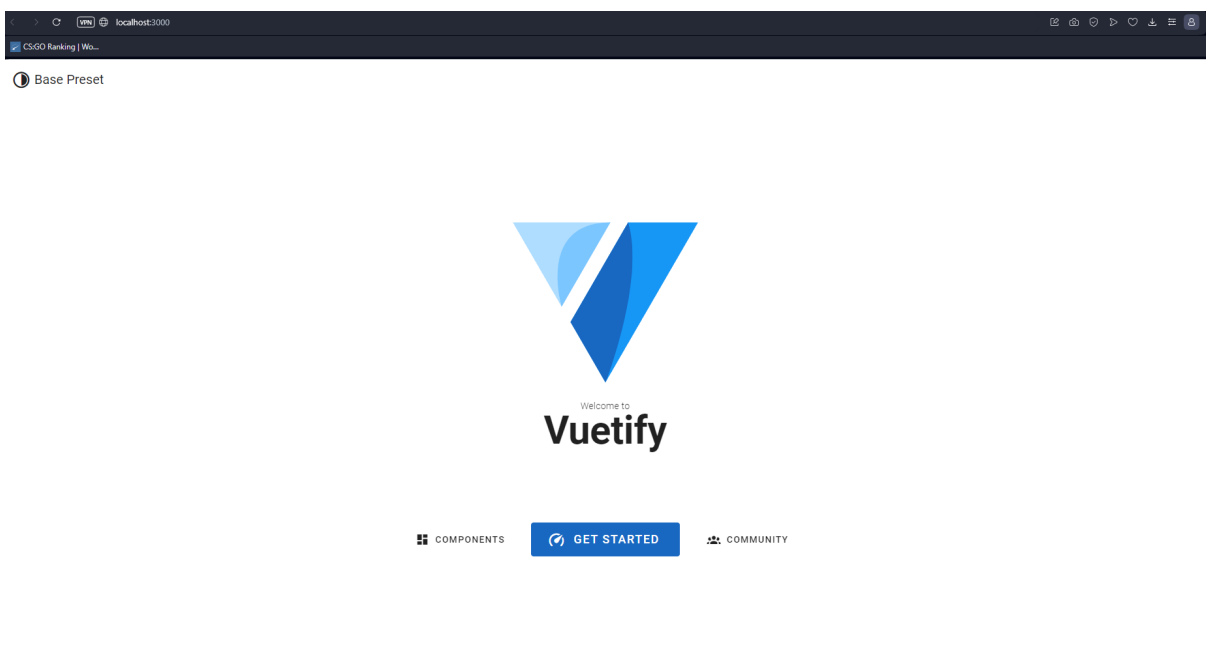
```
  Essentials (Vuetify, VueRouter, Pinia)
```

```
  Custom (Choose your features)
```

Fonte: Elaborado pelo autor.

Na configuração inicial, Figura 40, adotamos a instalação do Vuetify com o Vue Router, sem a utilização do TypeScript e como pacote de instalação de dependências utilizado o próprio npm.

Figura 41: Configuração pré-configuradas do Vuetify



Fonte: Elaborado pelo autor.

Posteriormente a instalação base, é possível visualizar a aplicação funcionando juntamente com a utilização do framework Vuetify, representado através da Figura 41. A partir deste ponto já é possível ter uma aplicação Vue com Vuetify funcionando e apto ao uso de componentes provenientes do próprio Vuetify.

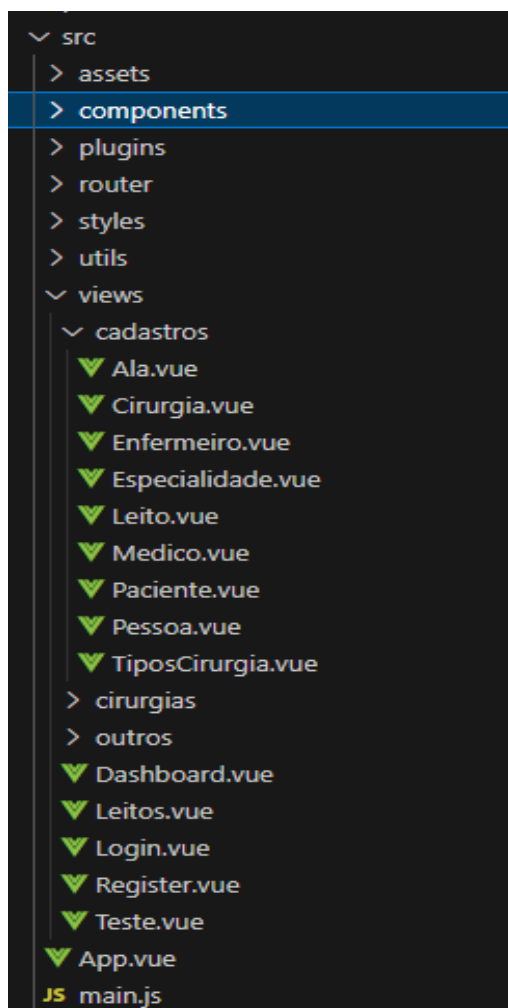
7.3.3 Estruturas de Pastas

Nesta etapa, estará descrito a organização da estrutura de pastas, um aspecto crucial no processo de desenvolvimento. A estrutura de pastas desempenha um papel fundamental na

manutenção e organização do projeto, facilitando a localização e gerenciamento eficiente dos arquivos.

A Figura 42, representada a seguir retrata a estrutura de pastas criadas no *frontend*, após a instalação do *Vuetify*, onde cada pasta tem sua responsabilidade.

Figura 42: Estrutura das pastas no Frontend do protótipo



Fonte: efetuado pelo autor.

A pasta “assets”, pasta responsável por conter conteúdos como imagens e fontes, sendo importados nos arquivos de componentes para demonstração ao usuário.

A pasta “components” é a pasta responsável por conter todos os componentes criados e que serão reutilizados no protótipo.

A pasta “plugins” é a pasta de criação e configurações do *Vuetify*.

A pasta “router” é responsável pelo redirecionamento entre as interfaces, sendo definidas por uma rota em direção a um componente.

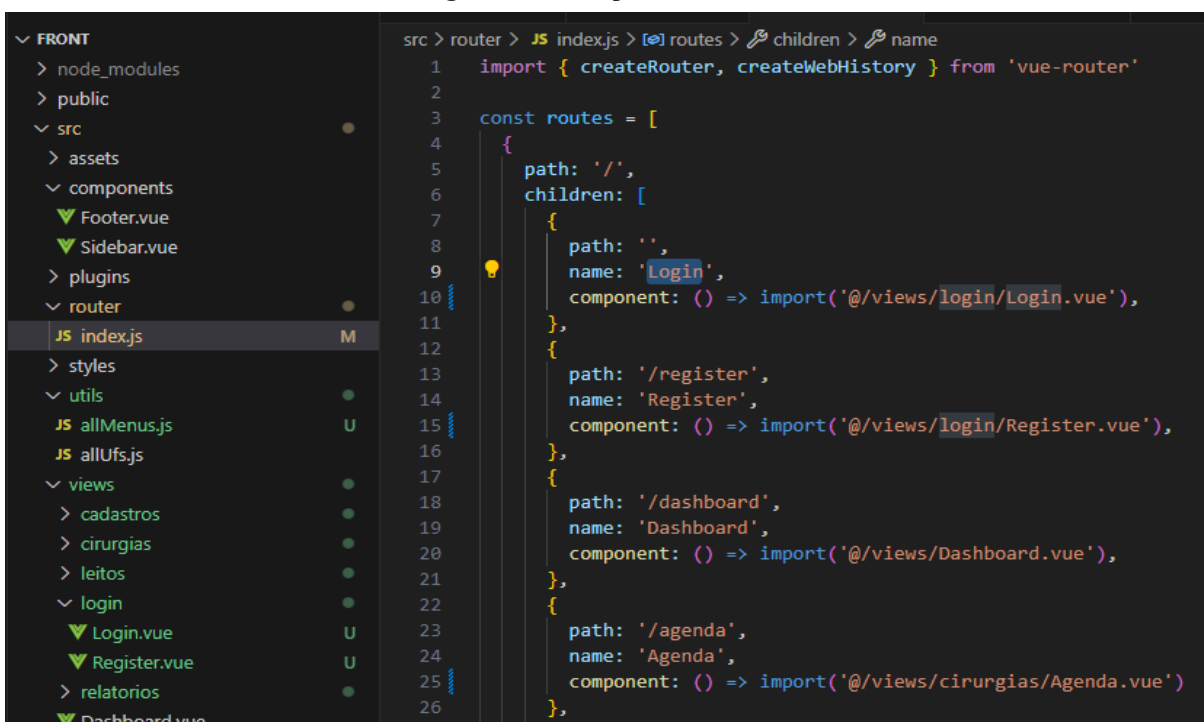
A pasta “style”, que é uma pasta privada de configuração do *Vuetify*.

A pasta “utils”, sendo responsável por arquivos auxiliares, como a sigla e nomenclatura de todas as UFs.

A pasta “views” que representa as principais páginas, geralmente sendo a junção dos componentes criados e prontas para serem demonstradas ao usuário.

O arquivo “main.js”, que é responsável pela criação da aplicação e de configuração de funcionalidades extras, como instanciar e criar uma *URL* base para utilização do *Axios*.

Figura 43: Exemplo de uso do Vue Router



```
src > router > JS index.js > routes > children > name
1  import { createRouter, createWebHistory } from 'vue-router'
2
3  const routes = [
4    {
5      path: '/',
6      children: [
7        {
8          path: '',
9          name: 'Login',
10         component: () => import('@/views/login/Login.vue'),
11       },
12       {
13         path: '/register',
14         name: 'Register',
15         component: () => import('@/views/login/Register.vue'),
16       },
17       {
18         path: '/dashboard',
19         name: 'Dashboard',
20         component: () => import('@/views/Dashboard.vue'),
21       },
22       {
23         path: '/agenda',
24         name: 'Agenda',
25         component: () => import('@/views/cirurgias/Agenda.vue')
26       },
27     ],
28   },
29 ]
```

Fonte: efetuado pelo autor.

Na imagem acima, figura 43, está representado todo o sistema de rotas disponível no protótipo, referenciando cada componente e seu redirecionamento quando chamado através da propriedade “to” ou do uso do router push no código.

Figura 44: Exemplo de uso do Vue Router

```
</v-form>
<div class="register">
  <v-btn to="/register" x-small color="#9E9EFF" dark>Registre-se</v-btn>
</div>
<div class="btnLogin">
```

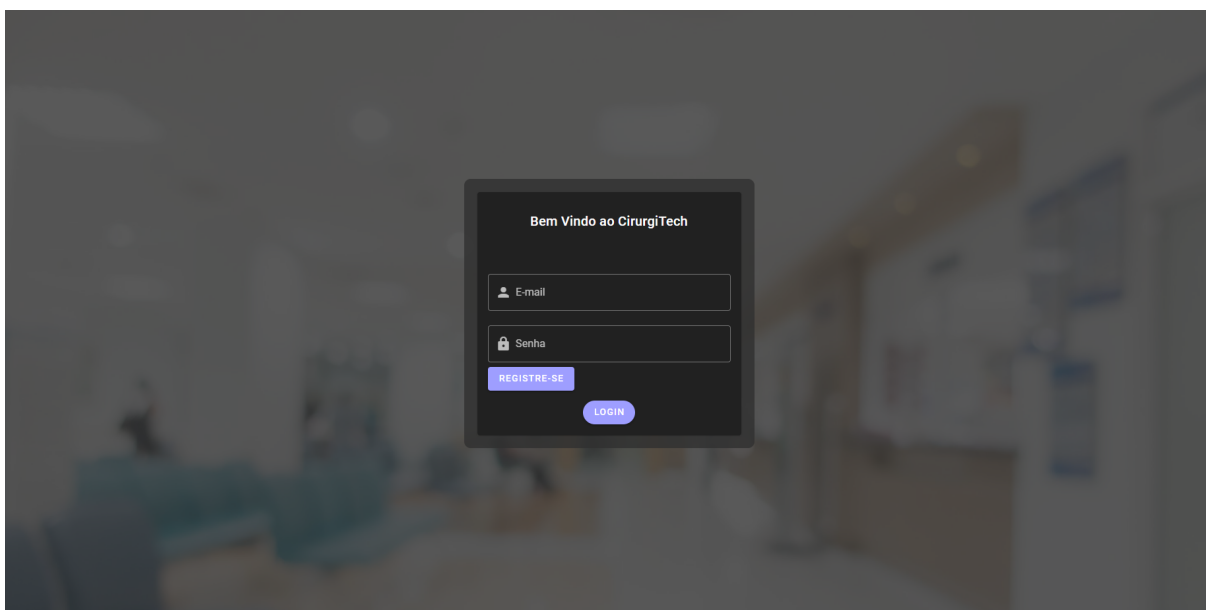
Fonte: efetuado pelo autor.

Na representação acima, figura 44, representamos o uso do Vue Router ao clicar em um botão, onde existe a propriedade “to”, que significa o caminho de destino do ao realizar a ação de clicar, que neste caso, irá para a *URL* de registro.

7.3.4 Interfaces

Neste segmento, exploraremos as interfaces do sistema, um aspecto essencial que desempenha um papel significativo na interação entre o usuário e o protótipo em questão.

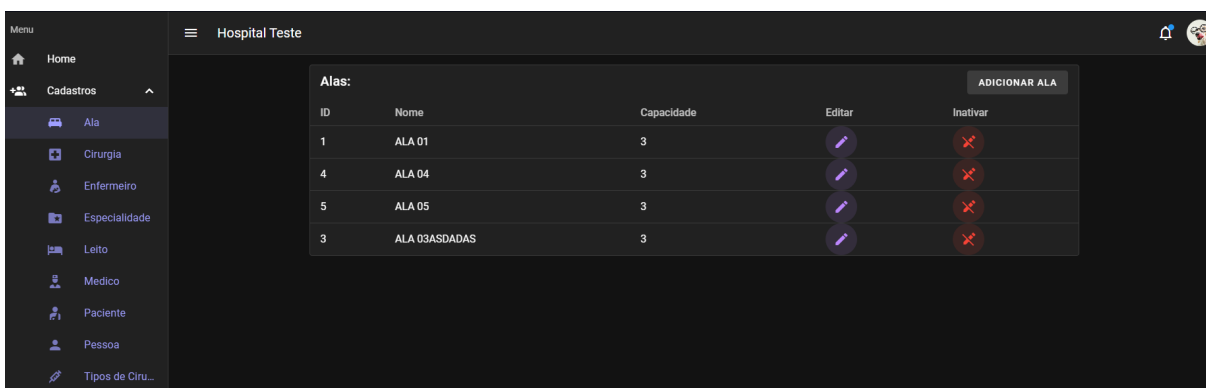
Inicialmente temos a tela de *Login*, derivado do termo inglês *logging in*, que significa conectar, refere-se ao conjunto de credenciais e procedimentos utilizados para identificar um usuário em diversas plataformas. Este método tem como objetivo conceder acesso a uma plataforma específica ou proporcionar os meios para que o usuário assuma o controle de uma função, tarefa ou sistema (BLASI, 2022).

Figura 45: Protótipo de Tela de Login

Fonte: efetuado pelo autor.

Na Figura 45 temos o primeiro protótipo de interface gráfica criado, responsável pelo acesso do usuário ao sistema, sendo necessário informar os campos e-mail e senha, caso seja permitido, irá acessar o sistema em questão.

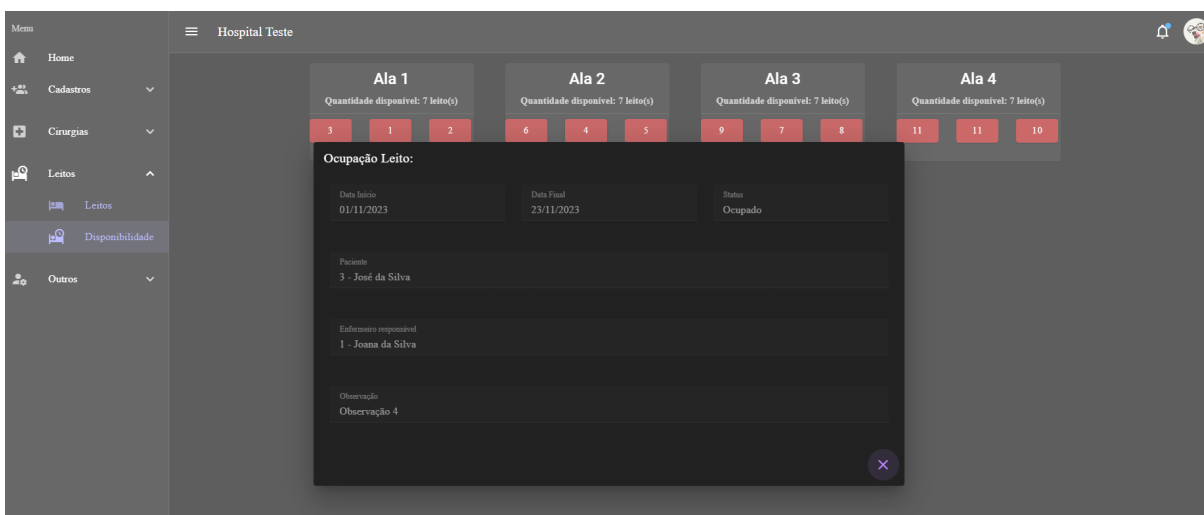
A gestão de cadastros é crucial para as empresas, pois envolve o controle de informações de todos os tipos de dados presentes no sistema e com uma gestão eficiente, mantendo a integridade dos dados, acaba proporcionando uma base sólida para decisões estratégicas, permitindo que a empresa seja mais ágil (ABDOUN, 2023).

Figura 46: Exemplificação do cadastro de Alas no protótipo

Fonte: efetuado pelo autor.

Na Figura 46 temos o menu lateral aberto com todas as possibilidades de cadastros, como cadastro de Alas, de Cirurgias, de Médicos e demais funcionalidades, onde todos eles têm as ações de inserção, edição e inativação.

Figura 47: Gerenciamento de leitos no protótipo



Fonte: efetuado pelo autor.

Na Figura 47 temos a listagem de leitos ocupados, este que ao clicar, apresentará os dados referentes ao leito ocupado em questão, como o paciente, enfermeiro responsável e observações, fazendo com que seja possível filtrar e verificar qual paciente está em qual leito, além da possibilidade de visualização rápida da quantidade de leitos disponíveis por ala.

O Kanban é uma parte fundamental da gestão de projetos, oferecendo uma abordagem direta e eficiente para a administração de demandas, visto que seu foco principal é priorizar as atividades e organizar as entregas (TOTVS, 2023).

Figura 48: Aplicação do Kanban no protótipo

Fonte: efetuado pelo autor.

Na figura 48, temos o conceito do Kanban aplicado ao projeto, onde há uma lista de cirurgias e a partir desta lista é possível controlá-las, através do arrasta e solta, viabilizando gerenciamento dos procedimentos cirúrgicos através dos status: novo, em andamento ou finalizadas. Além disso, é possível, facilmente, perceber o paciente e o tipo de cirurgia que está sendo realizada.

9 CONSIDERAÇÕES FINAIS

Diante das significativas mudanças no setor da saúde, com ênfase no Brasil, tanto aquelas causadas diretamente pela pandemia como as influências indiretas decorrentes desse cenário, a tarefa implementar um eficiente controle e gerenciamento de atividades hospitalares torna-se ainda mais desafiadora. Com isso, o presente trabalho apresenta um estudo do cenário da saúde no Brasil como um todo além do desenvolvimento de um protótipo, com as tecnologias Vue, Node e Postgres, que busca auxiliar, de forma intuitiva e fácil, a gestão hospitalar com foco em cirurgias e leitos.

A escolha de tecnologias web, como as citadas, permite uma maior acessibilidade e flexibilidade no acesso ao protótipo. Isso significa que profissionais de saúde e demais usuários podem utilizar a ferramenta a partir de diferentes computadores, o que proporciona uma integração mais eficiente com as rotinas hospitalares.

Além disso, a interoperabilidade oferecida no desenvolvimento das APIs possibilita uma integração eficiente com outros sistemas e ferramentas utilizadas na área da saúde, permitindo uma troca de informações mais fluida no gerenciamento hospitalar, visto que no protótipo há documentado de forma simples e prática todos os *endpoints* disponíveis para integração.

O protótipo entregue atende ao objetivo geral estabelecido, referente a gestão hospitalar concentrado principalmente nas cirurgias e nos leitos, possibilitando, de forma intuitiva, realizar os diversos controles necessários para um agendamento de cirurgia e para controle de ocupação dos leitos.

9.1 TRABALHOS FUTUROS

O levantamento de dados possibilitou diversos cenários de melhorias presentes para o sucesso do projeto em questão, principalmente com foco em segurança e na integridade dos dados, como as seguintes:

- Compreender e implantar o uso de VPN para garantir a segurança e privacidade, a utilização da VPN garante uma conexão segura entre locais autorizados ao acessar o sistema;
- Explorar a viabilidade de utilização de *middlewares* para o controle de autenticação e autorização, além de validações relacionadas a verificação dos ips, permitindo os autorizados;
- Implementar certificados SSL/TLS para garantir a comunicação segura entre o sistema web e os usuários. Isso é particularmente importante se o acesso for feito pela internet;
- Utilização de Firewalls para restringir o tráfego de entrada e saída no servidor onde o sistema está hospedado para proteger contra acessos não autorizados.

REFERÊNCIAS BIBLIOGRÁFICAS

ABDOUN, Alexandre. **Gestão de cadastros - O que é a gestão de dados cadastrais?**

Disponível em

<<https://www.stibosystems.com/pt-br/blog/gestao-de-cadastros-o-que-e-gestao-de-dados-caada-strais>> Acesso em: 15 nov. 2023.

AMORIM, Luana. **SC pretende abrir mais 63 leitos de UTI para conter lotação em hospitais, diz secretária.** Disponível em

<<https://www.nsctotal.com.br/noticias/sc-pretende-abrir-mais-63-leitos-de-uti-para-conter-lota-cao-em-hospitais-diz-secretaria>> Acesso em: 04 jun. 2023.

ANDRADE, Ana Paula. **O que é Express.js?** Disponível em

<<https://www.treinaweb.com.br/blog/o-que-e-o-express-js>> Acesso em: 08 nov. 2023.

ANDRADE, Ana Paula. **O que é UML?** Disponível em

<<https://www.treinaweb.com.br/blog/o-que-e-uml>> Acesso em: 05 dez. 2023.

APPMASTER. **BCrypt.** Disponível em <<https://appmaster.io/pt/glossary/bcrypt-5>> Acesso em: 15 nov. 2023.

ARAÚJO, A.C.M; GOUVEIA, L.B. Uma revisão sobre os princípios da Teoria Geral dos Sistemas, 2016.

AWARI. **Axios e React: Uma dupla poderosa para comunicação de dados.** Disponível em

<<https://awari.com.br/axios-react/>> Acesso em: 05 dez. 2023.

AWS. **O que é JavaScript?** Disponível em <<https://aws.amazon.com/pt/what-is/javascript/>>

Acesso em: 05 dez. 2023.

AXIOS. **Introdução.** Disponível em <<https://axios-http.com/ptbr/docs/intro>> Acesso em: 09 nov. 2023.

BATISTELA, Paulo. **Por que SC voltou a ter UTIs lotadas um ano após abertura emergencial de leitos.** Disponível em

<<https://www.nsctotal.com.br/noticias/por-que-sc-voltou-a-ter-utis-lotadas-um-ano-apos-abertura-emergencial-de-leitos>> Acesso em: 04 jun. 2023.

BESSA, André. **Node.JS: o que é, como funciona esse ambiente de execução JavaScript e um Guia para iniciar.** Disponível em <<https://www.alura.com.br/artigos/node-js>> Acesso em: 08 nov. 2023.

BETHA. **Perspectivas e desafios para a saúde pública pós-pandemia.** Disponível em <<https://www.betha.com.br/blog/saude-publica-pos-pandemia/>> Acesso em: 15 nov. 2023.

BRASIL, Agência. **Com hospitais superlotados, Santa Catarina decreta situação de emergência.** Disponível em <<https://agenciabrasil.ebc.com.br/saude/noticia/2022-06/santa-catarina-decreta-situacao-emergencia-por-hospitais-superlotados>> Acesso em: 03 jun. 2023.

BRITO, S.B.P; BRAGA, I.O; CUNHA, C.C; PALÁCIO, M.A.V; TAKENAMI, IUKARY. Pandemia da COVID-19: o maior desafio do século XXI.

CALANCA, Paulo. **SQL e NoSQL: trabalhando com bancos relacionais e não relacionais.** Disponível em <www.alura.com.br/artigos/sql-nosql-bancos-relacionais-nao-relacionais > Acesso em: 08 nov. 2023.

CABRAL, L.M.S; JUNIOR, D.F.C. Crescimento dos leitos de UTI no país durante a pandemia de Covid-19: desigualdades entre o público x privado e iniquidades regionais, 2020.

CARLOS. **O que é JavaScript e para que serve na programação web.** Disponível em <https://www.hostinger.com.br/tutoriais/o-que-e-javascript#O_Que_e_JavaScript> Acesso em: 08 nov. 2023.

CARLOS. **O que é Node.Js: Aplicações práticas e como instalá-lo.** Disponível em <https://www.hostinger.com.br/tutoriais/o-que-e-node-js#O_Que_e_Nodejs> Acesso em: 05 dez. 2023.

CDD, Redação. **O que é grupo de risco para Covid-19.** Disponível em <<https://cdd.org.br/noticia/saude-publica/o-que-e-grupo-de-risco-para-covid-19/>> Acesso em: 22 abr. 2023.

CHADE, Jamil. **Após milhões de mortos em 3 anos, OMS decreta fim da emergência da covid.** Disponível em <<https://noticias.uol.com.br/colunas/jamil-chade/2023/05/05/oms-decreta-fim-de-emergencia-por-covid-19.htm>> Acesso em: 04 jun. 2023.

CIVIL, Casa. **SUS completa 30 anos da criação.** Disponível em <<https://www.gov.br/casacivil/pt-br/assuntos/noticias/2020/setembro/sus-completa-30-anos-da-criacao>> Acesso em: 13 abr. 2023.

CONASS. **O SUS foi importante para a pandemia e terá papel fundamental no período pós Covid, avaliam especialistas durante debate organizado pelo Conass.** Disponível em <<https://www.conass.org.br/o-sus-foi-importante-para-pandemia-e-tera-papel-fundamental-no-periodo-pos-covid-avaliam-especialistas-durante-debate-organizado-pelo-conass/>> Acesso em: 13 abr. 2023.

COPILOT, Github. **Sobre o GitHub Copilot for Individuals.** Disponível em <<https://docs.github.com/pt/copilot/overview-of-github-copilot/about-github-copilot-for-individuals>> Acesso em: 06 nov. 2023.

COSTA, E.R; Banco de Dados Relacionais, 2011.

DANIELA. **O que é Docker e como ele funciona?** Disponível em <<https://www.hostinger.com.br/tutoriais/o-que-e-docker>> Acesso em: 18 nov. 2023.

DCC/UFMG. **Cap. 7: Arquitetura - Engenharia de Software Moderna.** Disponível <<https://engsoftmoderna.info/cap7.html>> Acesso em: 08 nov. 2023.

DevMedia. O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML. Disponível em <<https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>> Acesso em: 06 nov. 2023.

DIEESE. Brasil tem 2,3 leitos por mil habitantes, abaixo do mínimo desejável, 2020.

DOCKER. **What is a Container?** Disponível em <<https://www.docker.com/resources/what-container/>> Acesso em: 06 nov. 2023.

DOCKER. **Use Docker Compose.** Disponível em <https://docs.docker.com/get-started/08_using_compose/> Acesso em: 05 dez. 2023.

DOMINGUES, S.A.M.C. Desafios para a realização da campanha de vacinação contra a Covid-19 no Brasil.

DOUGLAS. Diagrama de classes (UML): Orientações básicas na elaboração de um diagrama de classes. Disponível em

<<https://www.devmedia.com.br/orientacoes-basicas-na-elaboracao-de-um-diagrama-de-classes/37224>> Acesso em: 07 nov. 2023.

EDUCAÇÃO, Redação XP. Banco de dados PostgreSQL: o que é e quais são os tipos?

Disponível em <<https://blog.xpeducacao.com.br/banco-de-dados-postgresql/>> Acesso em: 05 dez. 2023

EDUCAÇÃO, Redação XP. O que é e para que serve o GitHub Copilot? Disponível em

<<https://blog.xpeducacao.com.br/github-copilot/>> Acesso em: 03 dez. 2023.

EXPRESS. Express - Framework web rápido, flexível e minimalista para Node.js.

Disponível em <<https://expressjs.com/pt-br/>> Acesso em: 05 dez. 2023.

FERNANDO, B. V. A; SALES, P. O; MARTINS, M. A; GARCIA, G. L; FERREIRA, A.

K. R. O sistema único de saúde: Desafios, avanços e debates em 30 anos de história, 2019.

FILHO, S. A.; VELASCO, W.; VIEIRA, L.; LIMA, A.; COVID-19: DEMANDA REPRIMIDA DE CIRURGIAS ELETIVAS, 2021.

FIOCRUZ. Brasil celebra um ano da vacina contra o Covid-19. Disponível em

<<https://portal.fiocruz.br/noticia/brasil-celebra-um-ano-da-vacina-contr-covid-19>> Acesso em: 01 mai. 2023.

FIOCRUZ. Estudo revela como a pandemia afetou os atendimentos no SUS. Disponível

em <<https://agencia.fiocruz.br/estudo-revela-como-pandemia-afetou-os-atendimentos-no-sus>> Acesso em: 24 mar. 2023.

FIOCRUZ, PenseSus. SUS: O que é? Leia mais no PenseSUS. Disponível em

<<https://pensesus.fiocruz.br/sus#:~:text=Em%201988%2C%20por%20ocasi%C3%A3o%20da,gratuito%20a%20servi%C3%A7os%20de%20sa%C3%ADde>> Acesso em: 12 abr. 2023.

FRANK, K.M; PEREIRA, R.F; FILHO, J.V.D; DIAGRAMA

ENTIDADE-RELACIONAMENTO: UMA FERRAMENTA PARA MODELAGEM DE DADOS CONCEITUAIS.

FURTADO, Fernando. O que é o Docker e quais as vantagens de usá-lo? Disponível em

<<https://www.alura.com.br/artigos/comecando-com-docker>> Acesso em: 06 nov. 2023.

G1, SC. **Governo de SC prorroga decreto de calamidade pública por causa do coronavírus até 31 de outubro.** Disponível em

<<https://g1.globo.com/sc/santa-catarina/noticia/2021/06/28/governo-de-sc-prorroga-decreto-d-e-calamidade-publica-por-causa-do-coronavirus-ate-31-de-outubro.ghhtml>> Acesso em: 03 jun. 2023.

GIBBIN, Curtti Fernando; RUIZ, Almeida Antonio Marco. **Fique em casa.** Disponível em <<https://informasus.ufscar.br/fique-em-casa-2/>> Acesso em: 01 mai. 2023.

GUEDES, Marylene. **O que é MVC?** Disponível em <<https://www.treinaweb.com.br/blog/o-que-e-mvc>> Acesso em: 08 nov. 2023.

HANASHIRO, Akira. **O que se pode fazer com o JavaScript hoje em dia?** Disponível em <<https://www.treinaweb.com.br/blog/o-que-se-pode-fazer-com-javascript-hoje-em-dia>> Acesso em: 18 nov. 2023.

HANASHIRO, Akira. **VS Code - O que é e por quê você deve usar?** Disponível em <<https://www.treinaweb.com.br/blog/vs-code-o-que-e-e-por-que-voce-deve-usar>> Acesso em: 06 nov. 2023.

IBM. **O que é PostgreSQL?** Disponível em <<https://www.ibm.com/br-pt/topics/postgresql>> Acesso em: 08 nov. 2023.

IBM. **O que é o Docker?** Disponível em <<https://www.ibm.com/br-pt/topics/docker>> Acesso em: 06 nov. 2023.

IBM. **Diagramas de Sequência.** Disponível em <<https://www.ibm.com/docs/pt-br/rsm/7.5.0?topic=uml-sequence-diagrams>> Acesso em: 07 nov. 2023.

JWT. **JSON Web Token Introduction.** Disponível em <<https://jwt.io/introduction>> Acesso em: 05 dez. 2023.

KERLYN. **Uma breve introdução sobre BCrypt.** Disponível em <<https://medium.com/reprogramabr/uma-breve-introdução-sobre-bcrypt-f2fad91a7420>> Acesso em: 08 nov. 2023.

L. ANDREI. **O que é GitHub, Para Que Serve e Como Usar.** Disponível em <https://www.hostinger.com.br/tutoriais/o-que-github#O_Que_e_Git> Acesso em: 06 nov. 2023.

LUCIDCHART. **O que é um diagrama UML?** Disponível em
<<https://www.lucidchart.com/pages/pt/o-que-e-uml>> Acesso em: 06 nov. 2023.

LUCIDCHART. **O que é um diagrama de sequência UML?** Disponível em
<<https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-sequencia-uml>> Acesso em: 07 nov. 2023.

LUCIDCHART. **O que é um diagrama de atividades UML?** Disponível em
<<https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-atividades-uml>> Acesso em: 08 nov. 2023.

MACORATTI. **UML - Casos de Uso - Conceitos (revisão).** Disponível em
<https://macoratti.net/11/10/uml_rev1.htm> Acesso em: 07 nov. 2023.

MARTINS, Antonio. **Documentação de APIs: você conhece o Swagger?** Disponível em
<<https://medium.com/inside-picpay/documentação-de-apis-você-conhece-o-swagger-fd8b403d27ed>> Acesso em: 08 nov. 2023.

MÁXIMO, Wellton. **Com hospitais superlotados, Santa Catarina decreta situação de emergência.** Disponível em
<<https://agenciabrasil.ebc.com.br/saude/noticia/2022-06/santa-catarina-decreta-situacao-emergencia-por-hospitais-superlotados>> Acesso em: 23 mar. 2023.

MODELLI, Laís. **7 capitais têm leitos de UTI do SUS com mais de 90% de ocupação: ‘pior cenário já observado’, diz Fiocruz.** Disponível em
<<https://g1.globo.com/bemestar/coronavirus/noticia/2021/02/26/lotacao-nos-leitos-de-uti-do-sus-e-pior-cenario-ja-observado-desde-o-inicio-da-pandemia-alerta-fiocruz.ghtml>> Acesso em: 03 jun. 2023.

MONITORA. **Software: o que é e por quê aplicá-lo na sua empresa?** Disponível em
<<https://www.monitoretec.com.br/blog/software-o-que-e>> Acesso em: 06 nov. 2023.

MOSAICO, Blog. **Use cases: por que você precisa deles?** Disponível em
<<https://blogmosaico.medium.com/use-cases-por-que-voce-precisa-deles-584b99e3e670>>
Acesso em: 07 nov. 2023

MOSHE, Lucas. **Prototipar: o que é, como fazer e sua importância?** Disponível em
<<https://escolakoru.com.br/blog/prototipar-o-que-e-como-fazer/>> Acesso em: 15 nov. 2023.

MOZILLA. **Introdução Express/Node.** Disponível em <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction> Acesso em: 05 dez. 2023.

MOZILLA. **Sobre o JavaScript.** Disponível em <<https://developer.mozilla.org/pt-BR/docs/conflicting/Web/JavaScript>> Acesso em: 05 dez. 2023.

NACIONAL, Conselho de Saúde. **30 anos da lei que regulamentou o SUS: CNS segue em luta por mais orçamento.** Disponível em <<http://conselho.saude.gov.br/ultimas-noticias-cns/1379-30-anos-da-lei-que-regulamentou-o-sus-cns-segue-em-luta-por-mais-orcamento>> Acesso em: 13 abr. 2023.

NACIONAL, Plano. **Saiba quais são os primeiros grupos que serão vacinados contra a covid-19 no Brasil.** Disponível em <<https://gauchazh.clicrbs.com.br/saude/noticia/2020/12/saiba-quais-sao-os-primeiros-grupos-que-serao-vacinados-contracovid-19-no-brasil-ckirpu9hp0000017wo9fobiwe.html>> Acesso em: 23 abr. 2023.

NODE. **About Node.js.** Disponível em <<https://nodejs.org/en/about>> Acesso em: 05 dez. 2023.

NORONHA, S.M.V.K.; GUEDES, R.G.; TURRA, M.C.; ANDRADE, V.M.; BOTEGA, L.; NOGUEIRA, D; CALAZANS, A.J.; CARVALHO, L.; SERVO, L.; FERREIRA, F.M; Pandemia por COVID-19 no Brasil: análise da demanda e da oferta de leitos hospitalares e equipamentos de ventilação assistida segundo diferentes cenários, 2020.

OPAS. **Serviços essenciais de saúde enfrentam interrupções contínuas durante pandemia de COVID-19.** Disponível em <<https://www.paho.org/pt/noticias/7-2-2022-servicos-essenciais-saude-enfrentam-interruptoes-continuas-durante-pandemia-covid>> Acesso em: 15 nov. 2023.

OPUS.**Node.js – O que é, como funciona e quais as vantagens.**Disponível em <<https://www.opus-software.com.br/insights/node-js/>> Acesso em: 08 nov. 2023.

PICOLLO, Lucas. **Vue JS: o que é, como funciona e vantagens.** Disponível em <https://blog.geekhunter.com.br/vue-js-so-vejo-vantagens-e-voce/#O_que_e_Vue_JS> Acesso em 09 nov. 2023.

POSTGRES. **PostgreSQL: About**. Disponível em <<https://www.postgresql.org/about/>> Acesso em 05 dez. 2023.

POSTMAN. **What is Postman?** Disponível em <<https://www.postman.com/product/what-is-postman/>> Acesso em: 05 dez. 2023.

ROCCO, M; OLIVEIRA, L.B.; RIZZARDI, A.A.D.; RODRIGUES, G.; OLIVEIRA, G.; GUERREIRO, G.M.; CRUZ, S.V; JUNIOR, N.R.C; TCBC-BR.; Impacto da Pandemia por COVID-19 nos Procedimentos Cirúrgicos Eletivos e Emergenciais em Hospital Universitário, 2022.

ROUTER. **Vue Router**. Disponível em <<https://router.vuejs.org>> Acesso em: 05 dez. 2023.

SANTOS, C. Marilene. COVID-19: O flagelo do século 21, 2020.

SANAR. **Linha do tempo do Coronavírus no Brasil**. Disponível em <<https://www.sanarmed.com/linha-do-tempo-do-coronavirus-no-brasil>> Acesso em: 21 abr. 2023.

SEGUINS, Neilton. **O que é JSON Web Tokens?** Disponível em <<https://www.alura.com.br/artigos/o-que-e-json-web-tokens>> Acesso em: 08 nov. 2023.

SILVA, Gizele. **O que é Vuetify?** Disponível em <<https://coodesh.com/blog/dicionario/o-que-e-vuetify/>> Acesso em: 09 nov. 2023

SILVEIRA, Maria Isabelle. **Figma: o que é a ferramenta, Design e uso**. Disponível em <<https://www.alura.com.br/artigos/figma>> Acesso em: 15 nov. 2023

SILVEIRA, Paulo. **O que é Git e Github: como configurar os primeiros passos**. Disponível em <<https://www.alura.com.br/artigos/o-que-e-git-github>> Acesso em: 06 nov. 2023.

SOARES, D.H.K.; OLIVEIRA, S.L.; SILVA, F.K.R.; SILVA, A.C.D.; FARIAS, N.C.A.; MONTEIRO, M.L.M.E.; COMPAGNON, C.M. Medidas de prevenção e controle da covid-19: revisão integrativa, 2020.

SOUTO, Mario. **Front-end, Back-end e Full Stack**. Disponível em <<https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end>> Acesso em: 15 nov. 2023.

SWAGGER. **About Swagger Specification**. Disponível em <<https://swagger.io/docs/specification/about/>> Acesso em: 05 dez. 2023.

THOMAS, S.L.; PIETROWSKI, K.; KINALSKI, S. S.; BITTENCOURT, L.L.V.; SANGOI, M.C.K. Atuação do enfermeiro emergencista na pandemia de covid-19: Revisão narrativa da literatura, 2020.

TOTVS. **Kanban: como funciona, vantagens e implementação.** Disponível em <<https://www.totvs.com/blog/negocios/kanban/>> Acesso em: 09 nov. 2023.

TRUCCO, Cristian. **Docker Compose: O que é? Para que serve? O que come?** Disponível em <<https://imasters.com.br/banco-de-dados/docker-compose-o-que-e-para-que-serve-o-que-com-e>> Acesso em: 06. nov. 2023.

TRUCHE, P.; CAMPOS, N.L.; MARAZZO, B. E.; RANGEL, G. A; BERNARDINO, R.; BOWDER, N.A.; BUDA, M.A; FARIA, I.; POMPERMAIER, L.; RICE, E.H.; WATTER, D.; DANTAS, L.L.F.; MOONEY, P.D.; BOTELHO, F; FERREIRA, V.R.; ALONSO, N. Association between government policy and delays in emergent and elective surgical care during the COVID-19 pandemic in Brazil: a modeling study, 2021.

VARGAS, Tatiane. **Abrascão 2022: mesa aborda desafios e lições do SUS no enfrentamento da pandemia de Covid 19.** Disponível em <<https://informe.ensp.fiocruz.br/noticias/53594>> Acesso em: 14 abr. 2023.

VERSIANI, Rafael. **O que é o Postman?** Disponível em <<https://enotas.com.br/blog/postman/>> Acesso em: 11 nov. 2023.

VICTORA, C. G.; AQUINO, E. M. L.; LEAL, M. D. C.; MONTEIRO, C. A.; BARROS, F. C; SZWARCWALD, C. L. Maternal and child health in Brazil: progress and challenges. Lancet, v.377, n.9780, p.1863-76, dez. 2011.

VUE. **Introduction Vue.js.** Disponível em <<https://vuejs.org/guide/introduction.html>> Acesso em: 05 dez. 2023.

VUETIFY. **Why Vuetify?** Disponível em <<https://vuetifyjs.com/en/introduction/why-vuetify/>> Acesso em: 11 nov. 2023.

WALAKYS. **Visual Studio Code: A importância de uma poderosa ferramenta de desenvolvimento.** Disponível em: <<https://www.dio.me/articles/visual-studio-code-a-importancia-de-uma-poderosa-ferramenta-de-desenvolvimento>> Acesso em: 05 dez. 2023.