UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Gustavo Vicenzi

Desenvolvimento de Sistema de Gerenciamento de Reservas e Controle Automatizado de Dispositivos IoT para Quadras Esportivas

Blumenau
2025

Gustavo Vicenzi

Desenvolvimento de Sistema de Gerenciamento de Reservas e Controle Automatizado de Dispositivos IoT para Quadras Esportivas

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Carlos Barros Montez, Dr.
Supervisor: Matheus Fischer, Eng.

Blumenau
2025

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

http://portalbu.ufsc.br/ficha

Gustavo Vicenzi

Desenvolvimento de Sistema de Gerenciamento de Reservas e Controle Automatizado de Dispositivos IoT para Quadras Esportivas

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, <dia(número)> de <mês> de <ano(número)>.

Prof. Marcelo De Lellis Costa De Oliveira, Dr.
Coordenador do Curso

Banca Examinadora:
[preencher somente após a defesa para a Versão Final da BU]

Prof(a). Carlos Barros Montez, Dr.
Orientador(a)
UFSC/CTC/EAS

Matheus Fischer, Eng.
Supervisor(a)
Fischertec Tecnologia

Prof(a). xxxx, Dr(a).
Avaliador(a)
Instituição xxxx

Prof. xxxx, Dr.
Presidente da Banca
UFSC/CTC/EAS

Este trabalho é dedicado aos meus colegas de classe e
aos meus queridos pais.

AGRADECIMENTOS

Inserir os agradecimentos aos colaboradores da execução do trabalho.
Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

DECLARAÇÃO DE PUBLICIDADE

Blumenau, 17 de janeiro de 2025.

Na condição de representante da <instituição de realização do PFC> na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.

_____

Matheus Fischer, Eng.
Fischertec Tecnologia

RESUMO

Instruções do padrão genérico de TCCs da BU: No Resumo são ressaltados o objetivo da pesquisa, o método utilizado, as discussões e os resultados com destaque apenas para os pontos principais. O resumo deve ser significativo, composto de uma sequência de frases concisas, afirmativas, e não de uma enumeração de tópicos. Não deve conter citações. Deve usar o verbo na voz ativa e na terceira pessoa do singular. O texto do resumo deve ser digitado, em um único bloco, sem espaço de parágrafo. O espaçamento entre linhas é simples e o tamanho da fonte é 12. Abaixo do resumo, informar as palavras-chave (palavras ou expressões significativas retiradas do texto) ou, termos retirados de thesaurus da área. Deve conter de 150 a 500 palavras. O resumo é elaborado de acordo com a NBR 6028.

Instruções da Coordenação de PFC: O Resumo deve descrever de forma sucinta: o contexto/motivação/problema tratado no PFC; a solução proposta; a implementação/desenvolvimento; a metodologia e as principais técnicas e ferramentas utilizadas; os principais resultados obtidos e a importância/impactos de tais resultados para a empresa/clientes da empresa/instituto de pesquisa. Escrever todos esses pontos de forma bem resumida e direta, e sem entrar em detalhes técnicos. O tamanho do Resumo deve ocupar praticamente esta página inteira, e num **único** parágrafo. Além disso, Resumo + Palavras-Chave não podem ultrapassar esta página.

Palavras-chave: Palavra-chave 1. Palavra-chave 2. Palavra-chave 3. *[essas palavras-chave devem obrigatoriamente ser utilizadas no Resumo]*

ABSTRACT

Resumo traduzido para outros idiomas, neste caso, inglês. Segue o formato do resumo feito na língua vernácula. As palavras-chave traduzidas, versão em língua estrangeira, são colocadas abaixo do texto precedidas pela expressão “Keywords”, separadas por ponto.

Keywords: Keyword 1. Keyword 2. Keyword 3.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas

LISTA DE SÍMBOLOS

C	Circunferência de um círculo
π	Número pi
r	Raio de um círculo
A	Área de um círculo

SUMÁRIO

1 INTRODUÇÃO

Instruções do padrão genérico de TCCs da BU:

As orientações aqui apresentadas são baseadas em um conjunto de normas elaboradas pela Associação Brasileira de Normas Técnicas (ABNT). Além das normas técnicas, a Biblioteca também elaborou uma série de tutoriais, guias, *templates* os quais estão disponíveis em seu site, no endereço `http://portal.bu.ufsc.br/normalizacao/`.

Paralelamente ao uso deste *template* recomenda-se que seja utilizado o **Tutorial de Trabalhos Acadêmicos** (disponível neste link `https://repositorio.ufsc.br/handle/123456789/180829`).

Este *template* está configurado apenas para a impressão utilizando o anverso das folhas, caso você queira imprimir usando a frente e o verso, acrescente a opção *openright* e mude de *oneside* para *twoside* nas configurações da classe *abntex2* no início do arquivo principal *main.tex* (Araujo, 2015).

Conforme a Resolução NORMATIVA nº 46/2019/CPG as dissertações e teses não serão mais entregues em formato impresso na Biblioteca Universitária. Consulte o Repositório Institucional da UFSC ou sua Secretaria de Pós Graduação sobre os procedimentos para a entrega.

Instruções da Coordenação do PFC:

A Introdução deve apresentar um panorama geral do PFC de modo resumido, sem entrar em detalhes técnicos, para que o leitor já entenda aqui na Introdução o começo, meio e fim do PFC. Assim, é muito importante deixar bem claro na Introdução os seguintes pontos (a Introdução pode ser vista como um resumo geral do PFC, e a Seção Resumo como um resumo desse resumo):

- O contexto e a motivação do PFC, com uma breve descrição da empresa/instituto de pesquisa (histórico, clientes, produtos, serviços, projetos, etc) em que o PFC foi realizado e do projeto global da empresa em que o PFC está inserido (se for o caso)
- Breve descrição do problema tratado no PFC;
- A importância de tal problema para a empresa/clientes da empresa/instituto de pesquisa;
- Objetivos: aqui são descritos os objetivos, que podem ser estratificados em objetivo geral e objetivos específicos. Outra opção é colocar os objetivos específicos na forma de passos de uma metodologia de trabalho, ou seja, os procedimentos e ferramentas adotadas em cada fase do projeto (um plano de trabalho).
- Breve descrição da solução proposta;

- Breve descrição da implementação/desenvolvimento realizado;
- Breve descrição da metodologia e das principais técnicas e ferramentas utilizadas (do ponto de vista técnico). **Importante**: como *Scrum* é uma metodologia geral de execução de projetos, ele não se enquadra aqui, pois não se trata de uma metodologia técnica específica para o desenvolvimento do PFC;
- Breve descrição dos principais resultados obtidos e da importância/impactos de tais resultados para a empresa/clientes da empresa/instituto de pesquisa;
- Deixar bem claro o que foi de fato foi realizado pelo(a) autor(a), diferenciando do que foi aproveitado de trabalhos anteriores/outras times da empresa. **Importante**: esta preocupação em diferenciar o trabalho realizado pelo(a) autor(a) daquele de possíveis colegas de um time deve permear todo o documento.

Ressaltamos a seguir alguns aspectos cruciais para a escrita do PFC:

- Apesar de o tamanho da monografia não ter uma correlação direta com a nota, bons trabalhos costumam ter de 60 a 70 páginas efetivamente escritas (desde a Introdução até a Conclusão, excluindo-se Capa, Resumo, Sumário, Referências Bibliográficas, etc). Por outro lado, acima de 100 páginas a monografia pode se tornar "massante", discorrendo além do necessário para o entendimento do trabalho e, consequentemente, perdendo o foco do leitor.
- A linguagem a ser utilizada em um trabalho acadêmico deve ser técnico-científica e, portanto, formal (e não informal, como se o trabalho estivesse sendo explicado a um colega ou familiar). Desse modo, não devem ser usadas gírias. Além disso, não se deve escrever "o trabalho feito por mim", mas sim algo como "o presente trabalho trata de/o trabalho realizado pelo(a) autor(a)/etc".
- Utilize corretor ortográfico, verifique a gramática, e revise com muito cuidado e atenção todo o texto: pontuação, uso da vírgula, concordância, coesão e clareza textual, encadeamento entre frases, sentenças e parágrafos, etc.
- Este documento deve corresponder a uma monografia acadêmica, em que se deve fundamentar e justificar as ideias, afirmações, métodos e deduções apresentadas com base em técnicas, ferramentas, metodologias, equações, normas, literatura especializada (livros e artigos), etc, e não a um relatório descritivo voltado a um departamento/time de uma empresa.
- As referências bibliográficas não podem conter apenas sites, blogs e manuais técnicos: devem também conter livros, artigos, dissertações, teses, etc. Além disso, há uma diferença entre citação *direta* (comando `\citeas`) e citação *indireta* (comando `\cite`).

ESTRUTURA DO DOCUMENTO

Ao final da Introdução (que é o Capítulo 1), costuma-se apresentar como o documento está organizado, descrevendo brevemente o que é tratado nos demais capítulos do Sumário, começando pelo Capítulo 2. A estrutura de capítulos mostrada no Sumário deste documento é apenas uma sugestão geral e não precisa ser seguida à risca: dependendo da área e do foco do trabalho, tanto a estrutura detalhada (Capítulos, Seções e Subseções) quanto o número de capítulos pode variar. O(A) estudante deve consultar seu(sua) Orientador(a) Acadêmico(a) para definir a estrutura detalhada do documento.

Exemplo:

O presente documento está organizado da seguinte maneira. O Capítulo 2 apresenta a fundamentação teórica sobre os principais conceitos e técnicas necessárias para o entendimento do problema abordado e da solução proposta. O problema tratado neste PFC é descrito em detalhes no Capítulo 3, juntamente com os requisitos técnicos a serem atendidos. O Capítulo 4 aborda a solução proposta e a metodologia envolvida. O desenvolvimento realizado e a análise dos resultados obtidos são mostrados no Capítulo 5. Por fim, no Capítulo 6, são apresentadas as conclusões deste trabalho e algumas sugestões de trabalhos futuros são elencadas.

2 FUNDAMENTAÇÃO TEÓRICA

Instruções da Coordenação do PFC:

Deve-se colocar um parágrafo introdutório no início de **cada** capítulo, descrevendo os assuntos que serão abordados e a relação com o restante do trabalho. Por exemplo: *A Seção 2.1 apresenta . . . . Os resultados obtidos são analisados na Seção 2.2.* Pode-se fazer o mesmo no início de seções maiores, explicando para o leitor, em uma ou duas sentenças, o que está por vir no texto e o porquê. Outra boa prática é, ao final de cada capítulo, fazer uma ligação com o capítulo seguinte por meio de parágrafo curto.

Neste capítulo, deve-se apresentar as principais teorias, conceitos, técnicas, modelos, etc que são essenciais para o entendimento do problema tratado e da solução proposta.

Figuras, tabelas, quadros e equações devem ser introduzidos e explicados no texto: não se pode simplesmente "jogá-los" no texto, sem referência nem explicação. Por exemplo, deve-se escrever algo como: *O circuito projetado é mostrado na Figura 10. O resistor R_1 faz o papel de um limitador de corrente, enquanto o capacitor C_2 juntamente com o resistor R_5 formam um filtro passa-baixa. Este circuito tem a vantagem de . . .*

Com relação às equações, não se faz referência a uma equação que ainda não foi apresentada. Por exemplo, não se escreve: *A relação entre a tensão e a corrente de um resistor é dada pela Equação (1):*

$$V = RI\,. \tag{1}$$

O correto é algo como: *A relação entre a tensão e a corrente de um resistor é dada por (Lei de Ohm)*

$$V = RI\,, \tag{2}$$

na qual V é a tensão sobre o resistor, R a resistência e I a corrente elétrica. Da Equação (2), obtemos que

$$I = \frac{V}{R}\,. \tag{3}$$

Por outro lado, . . .

É importante observar que as equações fazem parte do texto e, assim, deve-se inserir uma vírgula ou ponto ao seu final. Se o parágrafo segue, pode-se eliminar o recuo na próxima linha com o comando `\noindent`. Além disto, se a frase segue, inicia-se a linha com letra minúscula. Veja os exemplos das Equações (2) e (3).

A seguir encontra-se uma equação na linha de texto: $\hat{y}(t+k \mid t) = \sum_{i=1}^{\infty} g_i \Delta u(t+k-i \mid t)$. E também, mais à frente, um exemplo de referência cruzada da Figura 1 e da Equação (4).

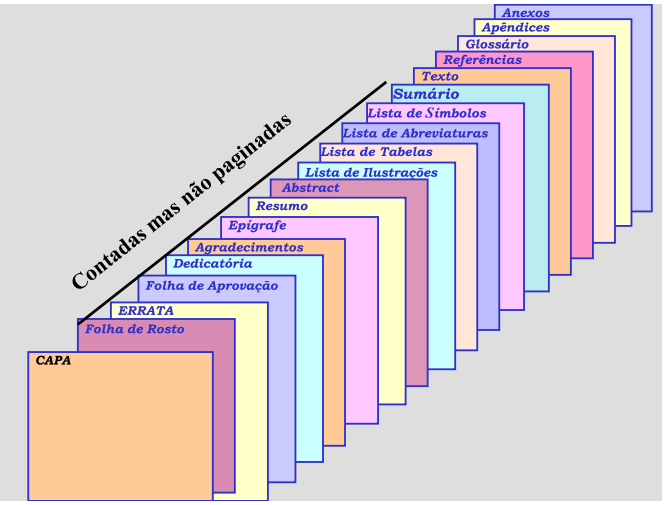
Instruções do padrão genérico de TCCs da BU:
Deve-se inserir texto entre as seções.

2.1 EXPOSIÇÃO DO TEMA OU MATÉRIA

É a parte principal e mais extensa do trabalho. Deve apresentar a fundamentação teórica, a metodologia, os resultados e a discussão. Divide-se em seções e subseções conforme a NBR 6024 (ABNT, 2012a).

Quanto à sua estrutura e projeto gráfico, segue as recomendações da ABNT para preparação de trabalhos acadêmicos, a NBR 14724, de 2011 (ABNT, 2011).

Figura 1 – Elementos do trabalho acadêmico.



Fonte: Universidade Federal do Paraná (1996).

2.1.1 Formatação do texto

No que diz respeito à estrutura do trabalho, recomenda-se que:

a) o texto deve ser justificado, digitado em cor preta, podendo utilizar outras cores somente para as ilustrações;

b) utilizar papel branco ou reciclado para impressão;

c) os elementos pré-textuais devem iniciar no anverso da folha, com exceção da ficha catalográfica ou ficha de identificação da obra;

d) os elementos textuais e pós-textuais devem ser digitados no anverso e verso das folhas, quando o trabalho for impresso. As seções primárias devem começar sempre em páginas ímpares, quando o trabalho for impresso. Deixar um espaço entre o título da seção/subseção e o texto e entre o texto e o título da subseção.

No Quadro 1 estão as especificações para a formatação do texto.

Quadro 1 – Formatação do texto.

Formato do papel	A4.
Impressão	A norma recomenda que caso seja necessário imprimir, deve-se utilizar a frente e o verso da página.
Margens	Superior: 3, Inferior: 2, Interna: 3 e Externa: 2. Usar margens espelhadas quando o trabalho for impresso.
Paginação	As páginas dos elementos pré-textuais devem ser contadas, mas não numeradas. Para trabalhos digitados somente no anverso, a numeração das páginas deve constar no canto superior direito da página, a 2 cm da borda, figurando a partir da primeira folha da parte textual. Para trabalhos digitados no anverso e no verso, a numeração deve constar no canto superior direito, no anverso, e no canto superior esquerdo no verso.
Espaçamento	O texto deve ser redigido com espaçamento entre linhas 1,5, excetuando-se as citações de mais de três linhas, notas de rodapé, referências, legendas das ilustrações e das tabelas, natureza (tipo do trabalho, objetivo, nome da instituição a que é submetido e área de concentração), que devem ser digitados em espaço simples, com fonte menor. As referências devem ser separadas entre si por um espaço simples em branco.
Paginação	A contagem inicia na folha de rosto, mas se insere o número da página na introdução até o final do trabalho.
Fontes sugeridas	Arial ou Times New Roman.
Tamanho da fonte	**Fonte tamanho 12 para o texto**, incluindo os títulos das seções e subseções. As citações com mais de três linhas, notas de rodapé, paginação, dados internacionais de catalogação, legendas e fontes das ilustrações e das tabelas devem ser de tamanho menor. Adotamos, neste *template* **fonte tamanho 10**.
Nota de rodapé	Devem ser digitadas dentro da margem, ficando separadas por um espaço simples por entre as linhas e por filete de 5 cm a partir da margem esquerda. A partir da segunda linha, devem ser alinhadas embaixo da primeira letra da primeira palavra da primeira linha.

Fonte: ABNT (2011).

2.1.1.1 As ilustrações

Independentemente do tipo de ilustração (quadro, desenho, figura, fotografia, mapa, entre outros), a sua identificação aparece na parte superior, precedida da palavra designativa.

> Após a ilustração, na parte inferior, indicar a fonte consultada (elemento obrigatório, mesmo que seja produção do próprio autor), legenda, notas e outras informações necessárias à sua compreensão (se houver). A ilustração deve ser citada no texto e inserida o mais próximo possível do texto a que se refere. (ABNT, 2011, p. 11).

2.1.1.2 Equações e fórmulas

As equações e fórmulas devem ser destacadas no texto para facilitar a leitura. Para numerá-las, usar algarismos arábicos entre parênteses e alinhados à direita. Pode-se adotar uma entrelinha maior do que a usada no texto (ABNT, 2011).

Exemplos, Equação (4) e Equação (5). Observe que o comando `\gls{}` é usado para utilizar para criar um *hyperlink* com a definição do símbolo na lista de símbolos (veja linha 153 de *main.tex*.

$$C = 2\pi r\sqrt{\gamma} + 10\,. \tag{4}$$

$$A = \pi r^2\,. \tag{5}$$

Aqui não há recuo porque o parágrafo não terminou, apenas foi iniciada uma nova frase após a equação. As equações fazem parte do texto, portanto estão sujeitas à pontuação (ponto final, vírgula etc.).

2.1.1.2.1 Exemplo tabela

De acordo com IBGE (1993), tabela é uma forma não discursiva de apresentar informações em que os números representam a informação central. Ver Tabela 3.

Tabela 3 – Médias concentrações urbanas 2010-2011.

Média concentração urbana	**População**		**Produto Interno Bruto – PIB (bilhões R$)**	**Número de empresas**	**Número de unidades locais**
Nome	**Total**	**No Brasil**			
Ji-Paraná (RO)	116 610	116 610	1,686	2 734	3 082
Parintins (AM)	102 033	102 033	0,675	634	683
Boa Vista (RR)	298 215	298 215	4,823	4 852	5 187
Bragança (PA)	113 227	113 227	0,452	654	686

Fonte: IBGE (2016).

2.2 APIS E APIS RESTFUL

APIs (*Application Programming Interfaces*) são interfaces que permitem a comunicação entre diferentes sistemas, softwares ou componentes. Elas funcionam como

pontes que facilitam o intercâmbio de dados e funcionalidades, promovendo integração e interoperabilidade. A ideia de API remonta aos primórdios da computação, quando surgiram as primeiras necessidades de criar padrões para que sistemas independentes pudessem se comunicar. Inicialmente, as APIs eram utilizadas em bibliotecas locais para abstrair funcionalidades complexas de hardware e software. Com o avanço da internet, especialmente nas décadas de 1990 e 2000, as APIs evoluíram para um modelo remoto, permitindo que sistemas distribuídos interagissem de maneira mais eficiente.

Nesse contexto, surgiu o conceito de APIs REST, um estilo arquitetural proposto por Roy Fielding em sua tese de doutorado em 2000. REST, acrônimo para *Representational State Transfer*, baseia-se em princípios como uniformidade, ausência de estado (*statelessness*), utilização de métodos HTTP - *Hypertext Transfer Protocol* (GET, POST, PUT, DELETE, etc.), e representação de recursos por meio de URLs (*Universal Resource Locators*). Fielding idealizou REST como uma maneira de padronizar a comunicação entre sistemas na web, tirando proveito das funcionalidades já existentes no protocolo HTTP. APIs REST ganharam popularidade devido à sua simplicidade, flexibilidade e capacidade de escalar, sendo amplamente adotadas por grandes empresas como Google, Twitter e Facebook (Fielding, 2000).

A ligação entre APIs de forma geral e APIs REST está na evolução das necessidades de integração entre sistemas. Enquanto APIs genéricas servem como um conceito abrangente para qualquer forma de comunicação programada entre sistemas, as APIs REST trouxeram um conjunto específico de diretrizes para implementar essas interações de maneira eficiente e alinhada com os padrões da web. Ao adotar REST, as empresas passaram a criar interfaces mais padronizadas e acessíveis, o que facilitou a interoperabilidade entre sistemas desenvolvidos por diferentes organizações.

As vantagens de utilizar APIs, de modo geral, incluem a possibilidade de reuso de funcionalidades, redução de redundâncias no desenvolvimento e facilidade de integração entre diferentes tecnologias. Já as APIs REST se destacam por sua simplicidade e aderência aos padrões da web, tornando-as fáceis de implementar e consumir. Por serem baseadas em HTTP, um protocolo amplamente suportado, as APIs REST são agnósticas em relação à linguagem de programação e oferecem suporte para uma ampla gama de clientes, desde navegadores até dispositivos IoT *(Internet das Coisas)*.

No livro "REST API Design Rulebook", Mark Masse destaca a importância de seguir boas práticas no design de APIs REST: "Uma API RESTful bem projetada deve ter como objetivo fornecer uma interface limpa e intuitiva para que os desenvolvedores interajam com recursos e serviços"(Masse, 2011). Essa ênfase na simplicidade e clareza reflete as razões pelas quais APIs REST têm sido tão amplamente adotadas. Além de facilitar a integração entre sistemas, elas promovem a consistência no design, o que reduz a curva de aprendizado para desenvolvedores e aumenta a produtividade

nas equipes de desenvolvimento. Assim, APIs e APIs REST desempenham um papel fundamental no desenvolvimento de soluções modernas e escaláveis, sendo uma base sólida para a inovação tecnológica.

2.3 CLOUD COMPUTING

A computação em nuvem, ou *Cloud Computing*, é um modelo de fornecimento de recursos computacionais, como armazenamento, processamento, bancos de dados e softwares, por meio da internet. A ideia de disponibilizar recursos computacionais de maneira remota surgiu na década de 1960, com o conceito de *time-sharing*, promovido por empresas como IBM e pesquisadores como John McCarthy, que previu que a computação poderia um dia ser organizada como um serviço público. No entanto, a computação em nuvem, como conhecida atualmente, começou a tomar forma nos anos 2000, com o surgimento de grandes provedores de serviços de nuvem, como Amazon Web Services (AWS), Microsoft Azure e Google Cloud Platform (GCP). A AWS, lançada em 2006, é amplamente reconhecida por popularizar a computação em nuvem com serviços como EC2 e S3, que forneceram acesso flexível e escalável a servidores e armazenamento.

> A computação em nuvem representa uma mudança significativa na forma como os recursos de TI são projetados, implantados e gerenciados, oferecendo um modelo para permitir acesso de rede conveniente e sob demanda a um conjunto compartilhado de recursos de computação configuráveis (Erl; Puttini; Mahmood, 2013).

Na atualidade, a computação em nuvem desempenha um papel essencial no desenvolvimento de software, especialmente no contexto de aplicações web e soluções de Internet das Coisas (IoT). No desenvolvimento de software web, a nuvem permite que empresas de todos os tamanhos utilizem infraestrutura escalável para hospedar seus aplicativos, garantindo alta disponibilidade e desempenho. Por exemplo, plataformas como AWS Elastic Beanstalk e Google App Engine facilitam a implantação e o gerenciamento de aplicativos web, reduzindo a complexidade da configuração de servidores. Além disso, ferramentas de integração e entrega contínuas (CI/CD), como GitHub Actions e Jenkins, são frequentemente integradas com serviços de nuvem, permitindo o desenvolvimento ágil e a atualização frequente de softwares.

No contexto de IoT, a computação em nuvem é fundamental para lidar com o grande volume de dados gerado por dispositivos conectados. Plataformas como AWS IoT Core, Azure IoT Hub e Google Cloud IoT oferecem soluções específicas para conectar, monitorar e gerenciar dispositivos IoT de forma centralizada. Essas plataformas permitem que sensores e dispositivos enviem dados para a nuvem, onde podem ser analisados e utilizados para decisões em tempo real. Por exemplo, em uma aplicação industrial, sensores de máquinas podem enviar informações sobre temperatura

e vibração para a nuvem, onde algoritmos de análise preditiva identificam possíveis falhas antes que ocorram. No ambiente residencial, dispositivos como termostatos inteligentes e câmeras de segurança conectadas utilizam serviços de nuvem para armazenamento de dados e controle remoto via aplicativos.

A computação em nuvem oferece vantagens significativas, como escalabilidade sob demanda, custos reduzidos de infraestrutura e maior flexibilidade para equipes de desenvolvimento. Para projetos de software, ela possibilita o processamento de grandes volumes de dados, integração de dispositivos globais e implantação rápida de novas funcionalidades. Com a crescente demanda por soluções conectadas e inovadoras, a nuvem continua a se consolidar como uma tecnologia essencial para o avanço da computação moderna.

2.4 SOFTWARE AS A SERVICE (SAAS)

Software como Serviço (SaaS, do inglês *Software as a Service*) é um modelo de distribuição de software baseado na nuvem, em que as aplicações são hospedadas por um provedor e acessadas pelos usuários por meio da internet, geralmente por meio de um navegador. Essa abordagem elimina a necessidade de instalação e manutenção de softwares locais, proporcionando uma experiência mais acessível e simplificada (Turner; Budgen; Brereton, 2003). A origem do conceito pode ser rastreada até a década de 1960, com a introdução do *time-sharing*, um modelo de computação em que múltiplos usuários podiam acessar sistemas centrais compartilhados. Entretanto, o modelo SaaS começou a ganhar relevância com o avanço da computação em nuvem nos anos 2000, especialmente após a popularização de empresas como Salesforce, que lançou sua plataforma de CRM (*Customer Relationship Management*) baseada na web em 1999, marcando um marco importante para o setor.

Na atualidade, o SaaS é amplamente utilizado em diversos setores, oferecendo soluções que vão desde produtividade e colaboração até análises de dados e gestão empresarial. Exemplos populares incluem ferramentas como o Google Workspace, que oferece aplicativos como Google Docs, Google Sheets e Google Drive, acessíveis diretamente no navegador e armazenados na nuvem. Plataformas como Slack e Microsoft Teams são usadas para comunicação e colaboração em equipe, eliminando a necessidade de servidores locais de e-mail ou mensagens instantâneas. Softwares de gestão empresarial, como SAP e NetSuite, também utilizam o modelo SaaS para oferecer recursos avançados de ERP (*Enterprise Resource Planning*) de forma escalável e acessível.

No setor de entretenimento, plataformas como Netflix e Spotify exemplificam o modelo SaaS ao oferecerem serviços de streaming de vídeo e música, respectivamente, sem a necessidade de que o conteúdo seja baixado ou armazenado localmente. Na área da educação, o SaaS tem permitido o crescimento de plataformas de apren-

dizado online, como Coursera e Khan Academy, que fornecem acesso a cursos e materiais educacionais diretamente pela web. Até mesmo áreas como saúde e finanças adotaram soluções SaaS, com softwares que ajudam na gestão de pacientes ou no controle financeiro de empresas.

As vantagens do SaaS incluem custos iniciais mais baixos, escalabilidade, acessibilidade de qualquer lugar com conexão à internet e atualizações automáticas realizadas pelo provedor. Além disso, o modelo permite que as empresas se concentrem em suas atividades principais, sem a necessidade de gerenciar infraestrutura de TI complexa. Embora existam desafios relacionados à segurança de dados e à dependência de conectividade, o SaaS continua a ganhar popularidade devido à sua flexibilidade e capacidade de atender às demandas de um mercado global em constante evolução.

2.5 CONTEINERIZAÇÃO

A containerização é uma tecnologia que revolucionou a maneira como aplicações são desenvolvidas, implantadas e gerenciadas, promovendo maior eficiência e consistência no ambiente de execução. Sua história remonta à década de 1970, com conceitos iniciais relacionados à virtualização e à criação de ambientes isolados em sistemas Unix, como o *chroot*. No entanto, foi apenas na década de 2000 que a containerização começou a ganhar tração significativa, especialmente com o surgimento de ferramentas como o Linux Containers (LXC) e, mais tarde, o Docker em 2013. O Docker popularizou a tecnologia ao torná-la acessível, padronizada e mais eficiente, contribuindo para sua ampla adoção em ambientes de desenvolvimento e produção.

A principal função da containerização é fornecer um ambiente isolado para aplicações, encapsulando todo o necessário para sua execução, como código, bibliotecas e dependências, em um único contêiner. Esses contêineres são leves, portáteis e consistentes, podendo ser executados em qualquer ambiente que suporte a tecnologia, como servidores locais, nuvens públicas ou privadas. Atualmente, a containerização é amplamente utilizada no desenvolvimento de software, especialmente em arquiteturas de microsserviços, onde cada serviço é executado em seu próprio contêiner, facilitando a escalabilidade e a manutenção. Além disso, é empregada em projetos de Internet das Coisas (IoT), onde dispositivos podem executar contêineres específicos para gerenciar funcionalidades de forma modular.

Entre as tecnologias que utilizam a containerização, destacam-se o Docker e o Kubernetes. O Docker é uma plataforma que permite criar, gerenciar e executar contêineres de maneira simplificada, enquanto o Kubernetes é um sistema de orquestração que automatiza o gerenciamento de contêineres em larga escala, garantindo alta disponibilidade, balanceamento de carga e escalabilidade automática. Essas ferramentas são amplamente adotadas em empresas de todos os tamanhos para aprimorar fluxos de trabalho de DevOps, modernizar aplicações legadas e facilitar a entrega contínua

de software.

A containerização trouxe diversas vantagens em relação às abordagens tradicionais de virtualização, como menor sobrecarga, maior eficiência no uso de recursos e tempos de inicialização mais rápidos. Além disso, ao garantir que as aplicações sejam executadas de forma consistente em diferentes ambientes, a tecnologia reduz problemas de compatibilidade e acelera os ciclos de desenvolvimento e implantação. Com sua versatilidade e eficiência, a containerização continua a transformar a forma como o software é projetado, implementado e executado em diversos setores.

2.6 ARQUITETURA EM CAMADAS

A arquitetura em camadas é um modelo amplamente adotado no desenvolvimento de aplicações modernas, proporcionando uma estrutura clara e organizada que facilita a manutenção, escalabilidade e reutilização de código. Esta abordagem divide a aplicação em camadas distintas, cada uma com responsabilidades específicas e bem definidas. No desenvolvimento da aplicação, a arquitetura em camadas é utilizada para separar as preocupações, garantindo que cada componente funcione de forma independente e coesa. (Fowler, 2002) descreve a arquitetura em camadas como uma abordagem fundamental para a construção de aplicações corporativas, destacando sua popularidade e uso extensivo em aplicações empresariais. Cada uma das camadas é apresentada da Seção 2.6.1 à Seção 2.6.5.

2.6.1 Camada de Apresentação

A camada de apresentação é responsável pela interface com o usuário final e pode ser desenvolvida utilizando diversas tecnologias e frameworks, como Next.js com a biblioteca React.js, Vue.js, Angular.js, HTML e CSS entre outros. O Next.js, por exemplo, com sua capacidade de renderização híbrida (SSR e SSG), oferece uma experiência de usuário rápida e otimizada para SEO, enquanto o React possibilita a criação de componentes reutilizáveis e uma interface dinâmica e responsiva. Nesta camada, as interações do usuário são capturadas e encaminhadas para a camada de aplicação geralmente através de chamadas HTTP a APIs RESTful, garantindo uma separação clara entre a interface e a lógica de negócios, porém outros protocolos podem ser usados para a comunicação.

2.6.2 Camada de Aplicação

A camada de aplicação atua como intermediária entre a interface do usuário e a lógica de negócios. Executando no servidor, esta camada gerencia a lógica de controle e orquestra as operações entre a camada de apresentação e a camada de negócios. Diversas tecnologias podem ser usadas para o desenvolvimento da aplicação tais como

Node.js, Nest.js, Java, Spring, C#, .NET, PHP, Python entre outros. Nesta camada, são definidos controladores que lidam com as requisições recebidas, e as enchaminham para a camada de negócios adequada para o processamento dos dados.

2.6.3 Camada de Negócios

A camada de negócios encapsula a lógica principal do sistema, garantindo que as regras de negócios sejam aplicadas de forma consistente. Esta camada é responsável pela implementação das regras de autenticação e autorização de usuários, além de toda a lógica necessária para o funcionamento adequado da aplicação. Ao concentrar as regras de negócio nesta camada, a aplicação assegura que todas as operações críticas são tratadas de forma centralizada e independente das outras camadas, simplificando o desenvolvimento, manutenção e melhoria do código da aplicação.

2.6.4 Camada de Persistência

A camada de persistência é responsável por gerenciar a interação com o banco de dados de forma abstrata ou direta. Pode-se utilizar tecnologias como TypeORM e Prisma nesta camada para abstrair a comunicação ou usar diretamente as *queries* para interação com o banco de dados, permitindo realizar operações de criação, leitura, atualização e exclusão de dados de forma simplificada e eficiente.

2.6.5 Camada de Banco de Dados

A camada de banco de dados é o próprio sistema de armazenameto escolhido para o projeto, podendo ser um banco de dados estruturado (como PostgreSQL, MySQL, Oracle entre outro) ou não (como MongoDB, Cassandran InfluxDB, etc). Esta camada é responsável por armazenar e recuperar os dados persistidos pela camada de persistência. A escolha adequada de um sistema de banco de dados garante a escalabilidade, consistência e disponibilidade dos dados, além de permitir o uso de recursos avançados para otimizar o desempenho.

2.6.6 Benefícios da Arquitetura em Camadas

A arquitetura em camadas traz vários benefícios para o desenvolvimento desta aplicação. A separação de preocupações facilita a manutenção do código, permitindo que alterações em uma camada específica não afetem diretamente as demais. Isso também melhora a escalabilidade da aplicação, pois novas funcionalidades podem ser adicionadas ou modificadas de forma isolada. Além disso, a modularização contribui para uma melhor reutilização de componentes, tornando o desenvolvimento mais eficiente e ágil. (Martin, Robert C., 2017) discute como a arquitetura em camadas permite

uma divisão clara de responsabilidades, promovendo independência no desenvolvimento e manutenção.

Portanto a adoção de uma arquitetura em camadas proporciona uma estrutura organizada e clara, que facilita o desenvolvimento, manutenção e evolução da aplicação. Cada camada desempenha um papel essencial, garantindo que a aplicação seja robusta, escalável e fácil de manter. A utilização desta abordagem, aliada às tecnologias escolhidas, assegura a entrega de um sistema eficiente, confiável e alinhado às melhores práticas do desenvolvimento de software moderno.

2.7 ARQUITETURA MODULAR

A arquitetura modular, também conhecida como arquitetura em módulos, é um estilo de design de software que organiza o sistema em componentes independentes, chamados módulos. Cada módulo encapsula uma funcionalidade específica e é projetado para ser autônomo, permitindo que diferentes partes do sistema possam ser desenvolvidas, testadas, mantidas e implantadas de forma independente. Esse tipo de arquitetura promove a separação de responsabilidades, facilitando o trabalho colaborativo entre equipes e reduzindo a complexidade no gerenciamento do sistema como um todo.

Os módulos comunicam-se entre si através de interfaces bem definidas, garantindo que as dependências entre eles sejam minimizadas e bem gerenciadas. A modularidade não apenas melhora a organização do código, mas também torna o sistema mais escalável, permitindo que novos recursos sejam adicionados sem comprometer as funcionalidades existentes.

Alguns dos principais benefícios da arquitetura modular são: a reusabilidade do código, já que um módulo pode ser utilizado em diferentes contextos; a facilidade na adição ou remoção de funcionalidades; a escalabilidade, simplificando a expansão do sistema; a capacidade de gerenciar alterações independentemente, além de facilitar a manutenção e desenvolvimento paralelo do código em diferentes times ou projetos.

2.8 INVERSÃO E INJEÇÃO DE DEPENDÊNCIAS

A inversão de dependências é um dos princípios fundamentais do desenvolvimento de software orientado a objetos, sendo parte dos cinco princípios do SOLID, apresentados pela primeira vez por Robert C. Martin. Esse conceito está diretamente relacionado à ideia de desacoplar componentes de software, de modo a tornar o código mais flexível, reutilizável e de fácil manutenção. No artigo **Design Principles and Design Patterns**, Martin descreve o **Princípio de Inversão de Dependências (DIP - *Dependency Injection Principle*)** como a prática de depender de abstrações, e não de implementações concretas (Martin, Robert C, 2000, p. 12). Esse princípio defende

que módulos de alto nível não devem depender de módulos de baixo nível diretamente; ambos devem depender de abstrações, como interfaces ou classes abstratas. Essa abordagem reduz o impacto de mudanças no sistema, pois a implementação concreta pode ser alterada sem afetar os módulos que a utilizam.

A injeção de dependências, por sua vez, é uma técnica para implementar o princípio de inversão de dependências. Ela consiste em fornecer as dependências que uma classe precisa a partir de fora, em vez de a própria classe criar ou localizar essas dependências. Em outras palavras, a responsabilidade de instanciar ou gerenciar dependências é transferida para um container ou outro componente externo. Existem três formas principais de realizar a injeção de dependências: por meio de construtores, métodos ou atributos da classe. Essa prática é amplamente adotada por frameworks modernos, como o Spring no Java e o NestJS no TypeScript, que oferecem containers de injeção de dependências para gerenciar automaticamente a criação e a injeção de objetos.

A ligação entre o princípio de inversão de dependências e a injeção de dependências é clara: a injeção de dependências é uma maneira prática de aplicar o DIP, permitindo que as classes dependam de abstrações em vez de implementações concretas. Por exemplo, em uma aplicação que utiliza um repositório para acessar o banco de dados, o repositório deve ser definido como uma interface (abstração), e a implementação concreta desse repositório deve ser fornecida à classe dependente por meio de injeção de dependências. Isso promove a substituição fácil da implementação do repositório, caso seja necessário, como ao trocar um banco de dados relacional por um banco NoSQL - *Not Only Structured Query Language*.

As vantagens de adotar o princípio de inversão de dependências e a injeção de dependências são muitas. Primeiramente, essas práticas reduzem o acoplamento entre os componentes do sistema, tornando o código mais modular e fácil de manter. A testabilidade também é significativamente melhorada, já que as dependências podem ser facilmente substituídas por objetos simulados (*mocks*) durante a execução de testes. Além disso, a flexibilidade do sistema aumenta, pois mudanças em uma implementação concreta têm impacto limitado nas demais partes do sistema. Essas práticas promovem também a reutilização de código, já que módulos de alto nível não são diretamente ligados às implementações de baixo nível.

Como afirma Robert C. Martin em seu artigo, módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações (Martin, Robert C, 2000). Esse princípio continua sendo um dos pilares para o desenvolvimento de software robusto e escalável, demonstrando sua relevância mesmo em arquiteturas modernas e frameworks contemporâneos.

O princípio da inversão de dependência é uma fundamental característica da arquitetura modular, que busca reduzir a dependência direta entre componentes e pro-

mover a independência das camadas. Como essa abordagem sugere que as classes ou módulos devem depender de abstrações gerais em vez de específicas, é possível alterar uma implementação sem afetar outras partes do sistema, tornando a manutenção e evolução mais fáceis.

2.9 IOT

A Internet das Coisas (IoT, do inglês *Internet of Things*) é um conceito que se refere à interconexão de dispositivos físicos com a internet, permitindo que eles coletem, compartilhem e atuem com base em dados, promovendo automação e eficiência em diversos setores (Gokhale; Bhat, O.; Bhat, S., 2018). A ideia de conectar dispositivos remonta à década de 1980, quando surgiu o conceito de "computação ubíqua", proposto por Mark Weiser, que vislumbrava a tecnologia como parte integral do cotidiano, funcionando de maneira invisível aos usuários. Entretanto, a IoT começou a ganhar forma prática nos anos 1990 com a popularização da internet e dos avanços em sensores, processadores e redes sem fio. Um marco importante foi a introdução de dispositivos como a máquina de venda de refrigerantes da Coca-Cola, em 1982, considerada um dos primeiros dispositivos "inteligentes", capaz de reportar seu status em tempo real.

Na atualidade, a IoT está presente em diversos setores, tanto na indústria quanto em residências e cidades. No setor industrial, a IoT é aplicada na automação de processos produtivos, monitoramento de máquinas e manutenção preditiva, configurando o que é chamado de Indústria 4.0. Sensores conectados podem, por exemplo, monitorar condições de equipamentos, como temperatura, vibração ou consumo energético, alertando sobre possíveis falhas antes que elas ocorram. Isso reduz custos operacionais e aumenta a eficiência. Além disso, a IoT é usada na logística, rastreando mercadorias em tempo real e otimizando rotas de transporte.

Em ambientes residenciais, a IoT está transformando as casas em espaços inteligentes. Dispositivos como termostatos conectados, lâmpadas controladas por aplicativos, assistentes virtuais (como Alexa ou Google Assistant) e câmeras de segurança inteligentes são exemplos de como a IoT está sendo integrada ao dia a dia das pessoas. Esses dispositivos permitem automação, controle remoto e até mesmo o uso de inteligência artificial para aprendizado dos hábitos dos moradores, aumentando conforto, segurança e eficiência energética.

Nas cidades, a IoT tem papel fundamental na construção de "cidades inteligentes". Exemplos incluem semáforos que ajustam automaticamente seus ciclos de acordo com o fluxo de tráfego, sistemas de coleta de lixo que otimizam rotas de caminhões com base no nível de preenchimento das lixeiras e sensores ambientais que monitoram a qualidade do ar em tempo real. Esses avanços visam melhorar a qualidade de vida dos cidadãos, reduzindo custos e impactos ambientais.

A IoT também é amplamente utilizada em setores como saúde, agricultura e varejo. No campo da saúde, dispositivos vestíveis como relógios inteligentes monitoram sinais vitais e enviam alertas em casos de anomalias. Na agricultura, sensores conectados ajudam no monitoramento do solo, otimizando o uso de água e fertilizantes. No varejo, prateleiras inteligentes e etiquetas RFID (*Radio Frequency Identification*) permitem rastrear estoques em tempo real e melhorar a experiência do cliente.

A IoT está revolucionando a forma como as pessoas interagem com o mundo ao seu redor, tornando processos mais inteligentes e conectados. Embora ainda enfrente desafios relacionados à segurança, privacidade e interoperabilidade, seu impacto já é visível em diversos aspectos da sociedade moderna, com um potencial imenso de expansão nos próximos anos.

3 DESCRIÇÃO DO PROBLEMA E REQUISITOS TÉCNICOS

Instruções da Coordenação do PFC:

Neste capítulo, deve-se apresentar (de forma mais detalhada e aprofundada tecnicamente que na Introdução):

- O contexto e a motivação do PFC;
- Descrição da empresa/instituto de pesquisa (histórico, clientes, produtos, serviços, projetos, etc) em que o PFC foi realizado e do projeto global da empresa em que o PFC está inserido (se for o caso);
- Descrição do problema tratado no PFC;
- Requisitos técnicos (funcionais e não-funcionais).

Procure utilizar equações, tabelas, diagramas e fluxogramas para ilustrar e explicar melhor as ideias.

3.1 CONTEXTUALIZAÇÃO

3.2 DESCRIÇÃO DO PROBLEMA

3.3 SOLUÇÃO PROPOSTA

3.4 REQUISITOS TÉCNICOS A SEREM ATENDIDOS

4 DESENVOLVIMENTO DA SOLUÇÃO PROPOSTA

Este capítulo apresenta o processo de desenvolvimento do projeto de fim de curso relativo ao desenvolvimento de um sistema de gerenciamento de reservas e controle automatizado de dispositivos IoT para quadras esportivas. Na Seção 4.1 são apresentadas as ferramentas e tecnologias utilizadas no desenvolvimento do sistema.

Instruções da Coordenação do PFC:

Neste capítulo, deve-se apresentar:

- A solução proposta para o problema descrito no capítulo anterior;
- Explicar a metodologia utilizada na solução proposta;
- Deixar bem claro e justificar tecnicamente para o leitor como que a solução proposta resolve o problema abordado e atende aos requisitos técnicos, explicando tecnicamente as decisões que foram tomadas para se chegar a tal solução.

Sugere-se colocar uma diagrama/fluxograma/casos de uso/etc ilustrando a solução proposta, e depois explicar em detalhes cada parte/bloco do diagrama/fluxograma ao longo do texto.

Ressaltamos que, em princípio, há uma infinidade de soluções possíveis para o problema abordado no PFC. Desse modo, é preciso explicar detalhadamente e justificar tecnicamente a solução proposta no PFC.

Alguns pontos que serão explicados nesse capítulo:

- **Ferramentas**: As tecnologias utilizadas na implementação da solução proposta, como linguagens de programação, frameworks, bibliotecas, etc.
- **Arquitetura da solução**: A arquitetura utilizada na implementação da solução proposta, como monolítica, microsserviços, cliente-servidor, etc.
- **Modelagem do banco de dados**: O modelo utilizado para representar os dados da aplicação, incluindo as entidades, relacionamentos, atributos, etc.
- **Backend**: A parte da solução que lida com a manipulação dos dados e a regra de negócio da aplicação.

  Registro e autenticação de usuários: Como o backend implementa a funcionalidade de registro e autenticação de usuários, incluindo validações, segurança, etc.

  Autorização de usuários: Como o backend implementa a funcionalidade de autorização de usuários, incluindo permissões, roles, etc.

Rotas da aplicação: Como o backend implementa as rotas da aplicação, incluindo endpoints de CRUD para tenanants, quadras, agendamentos de horários e credenciais de acesso a API relacionada à internet das coisas (IoT).

Detalhamento do agendamentos de horários: Como o backend implementa a funcionalidade de agendamentos de horários de quadras, incluindo informações sobre os locais, os tipos de atividades, as datas e as horas.

Detalhamento da integração com IoT: Como o backend implementa a integração com dispositivos IoT e comanda o acionamento e desligamento de acordo com os agendamentos feitos pelosus usuários.

- **Análise de resultados**: Análise dos resultados obtidos com o desenvolvimento da aplicação. Detalhando os principais aspesctos relevantes, como a eficiência do agendamento de horários, a segurança das informações e a integração com IoT.

4.1 FERRAMENTAS E TECNOLOGIAS UTILIZADAS

A seção de ferramentas e tecnologias utilizadas no desenvolvimento deste projeto apresenta um panorama detalhado dos recursos técnicos empregados para a construção do sistema de gerenciamento de reservas e controle automatizado de dispositivos IoT para quadras esportivas.

4.1.1 Git

O Git é um sistema de controle de versão distribuído criado por Linus Torvalds em 2005, inicialmente para o desenvolvimento do kernel do Linux. Antes do Git, o projeto Linux utilizava o BitKeeper, mas devido a restrições de licenciamento, Torvalds decidiu desenvolver uma ferramenta própria. O objetivo principal era criar um sistema rápido, eficiente e que suportasse grandes projetos com facilidade.

As principais funções do Git incluem rastreamento de alterações no código-fonte, permitindo que múltiplos desenvolvedores trabalhem simultaneamente em um mesmo projeto sem conflitos. O Git possibilita a criação de ramificações (branches) para desenvolver novos recursos ou corrigir bugs de forma isolada, facilitando a integração dessas alterações ao projeto principal posteriormente. Além disso, como um sistema distribuído, cada desenvolvedor possui uma cópia completa do repositório, o que aumenta a segurança e a integridade dos dados. Atualmente, o Git é a ferramenta de controle de versão mais popular do mundo, amplamente utilizada por empresas de software, desenvolvedores independentes e projetos de código aberto.

Decidiu-se utilizar o Git devido a familiaridade da equipe com a ferramenta, já que é amplamente usada em outros projetos na empresa, além de sua robustez e confiabilidade.

O Git revolucionou a forma como o desenvolvimento de software é conduzido, proporcionando uma maneira eficiente e colaborativa de gerenciar projetos complexos. Sua flexibilidade, velocidade e robustez tornaram-no uma ferramenta essencial no arsenal de qualquer desenvolvedor, suportando desde pequenos projetos individuais até grandes sistemas empresariais.

4.1.2 GitHub

O GitHub é uma plataforma de hospedagem e repositório de código-fonte e arquivos que utiliza o Git para controle de versão. A plataforma permite que programadores ou qualquer usuário cadastrado contribuam em projetos, sejam eles privados ou de código aberto, de qualquer lugar do mundo. Amplamente adotada por desenvolvedores, o GitHub facilita a divulgação de trabalhos e a colaboração em projetos, além de oferecer recursos que simplificam a comunicação, como a identificação de problemas e a mesclagem de repositórios remotos por meio de issues e pull requests.

Atualmente, o GitHub é utilizado globalmente, contando com mais de 36 milhões de usuários ativos que contribuem em projetos comerciais ou pessoais. A plataforma hospeda mais de 100 milhões de projetos, incluindo alguns de renome mundial, como WordPress, GNU/Linux, Atom e Electron. Além disso, o GitHub oferece suporte a recursos de organização, amplamente utilizados por aqueles que desejam expandir seus projetos.

O Github é uma ferramenta usada por padrão na Fischertec. Embora outras plataformas como o Gitlab tenham funções semelhantes, optou-se por manter o padrão da empresa.

4.1.3 PostgreSQL

O SQL (Structured Query Language) foi desenvolvido nos anos 1970 pela IBM como uma linguagem padrão para gerenciar e manipular bancos de dados relacionais. PostgreSQL, um sistema de gerenciamento de banco de dados relacional, surgiu na Universidade da Califórnia em Berkeley, no final dos anos 1980, como parte do projeto POSTGRES (Post Ingres). Em 1996, o sistema foi renomeado para PostgreSQL, refletindo sua compatibilidade com SQL. Desde então, PostgreSQL evoluiu significativamente, tornando-se uma das bases de dados relacionais mais robustas e avançadas disponíveis.

PostgreSQL oferece uma ampla gama de funções, incluindo suporte a transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), integridade referencial, e extensões para manipulação de dados complexos como JSON e XML. Além

disso, suporta procedimentos armazenados, gatilhos e uma variedade de tipos de índices para otimizar consultas. Atualmente, PostgreSQL é amplamente utilizado por grandes empresas, startups e projetos de código aberto devido à sua confiabilidade, flexibilidade e conformidade com os padrões SQL.

Existem outras soluções de banco de dados SQL como MySQL, que é conhecido por sua facilidade de uso e performance em aplicações web, e soluções NoSQL como MongoDB, que é ideal para armazenar grandes volumes de dados não estruturados e aplicações que exigem alta escalabilidade e flexibilidade. Optou-se por usar o PostgreSQL no desenvolvimento do projeto devido à sua capacidade de lidar com transações complexas, suporte a dados estruturados e não estruturados, e forte conformidade com os padrões SQL, o que garante integridade e consistência dos dados.

O PostgreSQL se destaca como uma solução de banco de dados robusta e versátil, adequada para uma ampla gama de aplicações. Sua evolução ao longo dos anos e a capacidade de suportar funcionalidades avançadas o tornam uma escolha excelente para projetos que exigem confiabilidade e flexibilidade. A decisão de utilizá-lo no projeto foi fundamentada em sua capacidade de atender às necessidades específicas de gerenciamento de dados de maneira eficiente e segura.

4.1.4 Typescript

O JavaScript, criado em 1995 por Brendan Eich da Netscape, rapidamente se tornou a linguagem padrão para desenvolvimento web, permitindo a criação de páginas dinâmicas e interativas. No entanto, à medida que aplicações web se tornavam mais complexas, as limitações de JavaScript em termos de tipagem estática e suporte para grandes projetos se tornaram evidentes. Para reduzir essas limitações, a Microsoft lançou o TypeScript em 2012, uma linguagem de programação de código aberto que é um superconjunto (superset) estrito de JavaScript. TypeScript adiciona tipagem estática e outros recursos avançados ao JavaScript, ajudando os desenvolvedores a escrever código mais seguro e escalável.

As principais funções do TypeScript incluem a adição de tipos estáticos, interfaces, classes e módulos, que não existem nativamente no JavaScript. Essas funcionalidades permitem a detecção de erros durante o desenvolvimento, antes do tempo de execução, melhorando a qualidade do código. Além disso, TypeScript se transpila (é convertido) para JavaScript, garantindo compatibilidade total com os navegadores e plataformas que suportam JavaScript. Atualmente, TypeScript é amplamente adotado em projetos de grande escala devido à sua capacidade de melhorar a produtividade e a manutenção do código. Empresas como Google, Microsoft e Airbnb utilizam TypeScript em seus projetos.

No desenvolvimento do projeto backend, optou-se por TypeScript em vez de JavaScript por várias razões. TypeScript proporciona uma melhor experiência de de-

senvolvimento, oferecendo autocompletar, navegação de código e verificação de tipos em tempo de compilação, o que ajuda a evitar muitos erros comuns em JavaScript. Isso é especialmente importante em projetos backend, onde a robustez e a previsibilidade do código são cruciais. Além disso, TypeScript facilita a colaboração entre desenvolvedores, permitindo que eles entendam e mantenham o código com mais facilidade.

O TypeScript se destaca como uma ferramenta poderosa que complementa e aprimora JavaScript, tornando o desenvolvimento de software mais eficiente e menos propenso a erros. Sua adoção no desenvolvimento da aplicação backend de gerenciamento de reservas e controle automatizado de dispositivos IoT permitiu criar um código mais robusto e sustentável.

4.1.5 Node.js

O Node.js é uma plataforma de desenvolvimento open-source baseada no motor JavaScript V8 do Google Chrome, criada em 2009 por Ryan Dahl. Seu objetivo inicial era possibilitar a execução de JavaScript no lado do servidor, permitindo o desenvolvimento de aplicações web com maior eficiência e escalabilidade. Desde seu lançamento, o Node.js rapidamente ganhou popularidade devido à sua arquitetura orientada a eventos e sua capacidade de lidar com operações de entrada e saída de forma não bloqueante, o que é particularmente útil para aplicações em tempo real.

Atualmente, o Node.js é amplamente utilizado para construir desde APIs e microservices até aplicações completas de grande escala. Sua capacidade de executar JavaScript tanto no cliente quanto no servidor facilita o desenvolvimento fullstack, enquanto a vasta biblioteca de pacotes disponíveis através do npm (Node Package Manager) oferece soluções para praticamente qualquer necessidade de desenvolvimento. O Node.js tem sido uma escolha popular em diversas indústrias devido à sua performance, escalabilidade e ao suporte contínuo de uma grande comunidade de desenvolvedores.

Uma das principais vantagens do Node.js em comparação com outras soluções é sua natureza assíncrona e orientada a eventos, que permite o gerenciamento eficiente de múltiplas conexões simultâneas sem sobrecarregar os recursos do servidor. Além disso, a utilização de JavaScript, uma linguagem amplamente conhecida e utilizada, reduz a curva de aprendizado para novos desenvolvedores e facilita a integração entre equipes de frontend e backend. A modularidade do Node.js e sua grande comunidade de apoio também contribuem para o desenvolvimento mais rápido e eficiente de aplicações.

Portanto o Node.js se destaca como uma solução versátil e eficiente para o desenvolvimento de aplicações modernas. Sua combinação de performance, escalabilidade e uma vasta gama de ferramentas e bibliotecas tornam-no uma escolha robusta

para desenvolvedores que buscam criar aplicações rápidas e escaláveis, atendendo às demandas de um mercado cada vez mais dinâmico e competitivo.

4.1.6 Nest.js

O Nest.js é um framework de desenvolvimento backend criado em 2017 por Kamil Myśliwiec. Inspirado nos princípios de programação modular e fortemente influenciado pelo Angular, o Nest.js foi projetado para oferecer uma estrutura robusta e escalável para a construção de aplicações do lado do servidor. Desde o seu lançamento, o framework tem crescido em popularidade, especialmente entre desenvolvedores que buscam uma abordagem moderna para o desenvolvimento de aplicações backend em Node.js.

O Nest.js fornece uma arquitetura modular que facilita a criação e a manutenção de aplicações escaláveis e bem estruturadas. Ele suporta diversos paradigmas de programação, incluindo orientação a objetos, programação funcional e reativa. Com suporte nativo para TypeScript, o Nest.js oferece tipagem estática, o que melhora a confiabilidade e a legibilidade do código. Atualmente, é amplamente utilizado para construir APIs RESTful, microservices e aplicativos monolíticos, sendo uma escolha frequente em projetos que exigem alta escalabilidade e flexibilidade.

Entre as principais vantagens do Nest.js estão sua modularidade, que permite a criação de aplicações altamente organizadas e de fácil manutenção, e seu suporte integral a TypeScript, que traz maior segurança e produtividade no desenvolvimento. Em comparação com outros frameworks, como Express.js, o Nest.js oferece uma estrutura mais robusta e orientada a boas práticas, facilitando o desenvolvimento de aplicações complexas. No contexto do projeto de fim de curso, o Nest.js foi escolhido por sua capacidade de suportar a criação de uma aplicação multi-tenant complexa, com necessidades específicas de escalabilidade, organização e integração com outras tecnologias.

O Nest.js se destaca como uma ferramenta poderosa para o desenvolvimento de aplicações backend modernas, combinando a performance do Node.js com uma arquitetura flexível e escalável. Sua adoção neste projeto de fim de curso reflete a busca por soluções eficientes e de alta qualidade, alinhadas com as demandas atuais do mercado de tecnologia.

4.1.7 Docker

O Docker foi lançado em 2013 por Solomon Hykes, inicialmente como um projeto interno da empresa dotCloud. A plataforma foi criada com o objetivo de simplificar o processo de desenvolvimento, distribuição e execução de aplicações por meio da virtualização de contêineres. Desde seu lançamento, o Docker tem revolucionado a

maneira como aplicações são desenvolvidas e implantadas, tornando-se uma das ferramentas mais populares no ecossistema de DevOps.

Atualmente, o Docker é amplamente utilizado para criar, distribuir e executar aplicações em contêineres, que são ambientes leves e portáteis que contêm tudo o que uma aplicação precisa para ser executada. Isso inclui código, bibliotecas e dependências, garantindo que a aplicação funcione de maneira consistente em qualquer ambiente. O Docker é uma escolha comum em projetos que exigem portabilidade, escalabilidade e uma integração contínua eficiente, sendo adotado por empresas de todos os tamanhos para modernizar seus fluxos de trabalho de desenvolvimento e implantação.

Entre as principais vantagens do Docker estão a portabilidade de aplicações, a eficiência no uso de recursos e a facilidade de integração com pipelines de CI/CD (integração contínua e entrega contínua). Comparado com máquinas virtuais tradicionais, os contêineres do Docker são mais leves e rápidos, o que resulta em tempos de inicialização menores e melhor utilização de recursos. No contexto deste projeto, o Docker foi escolhido para garantir que o ambiente de desenvolvimento e produção seja consistente, facilitando a implantação em servidores AWS e reduzindo problemas de compatibilidade.

O Docker se destaca como uma solução essencial para o desenvolvimento e a implantação de aplicações modernas, oferecendo uma maneira eficiente de gerenciar ambientes de software. Sua escolha neste projeto reflete a busca por uma solução que ofereça portabilidade, eficiência e flexibilidade, alinhando-se às melhores práticas do mercado e garantindo um processo de desenvolvimento mais ágil e confiável.

4.1.8 Postman

O Postman foi lançado em 2012 por Abhinav Asthana como uma ferramenta auxiliar para o desenvolvimento de APIs. Inicialmente criado como uma extensão para o Google Chrome, o Postman rapidamente evoluiu para uma aplicação completa e independente, tornando-se uma das ferramentas mais populares para o teste e desenvolvimento de APIs. Ao longo dos anos, a plataforma expandiu suas funcionalidades para atender às crescentes demandas de desenvolvedores e equipes de API.

O Postman é amplamente utilizado para testar, documentar e monitorar APIs, oferecendo uma interface amigável que facilita a realização de requisições HTTP, o gerenciamento de coleções de APIs e a automatização de testes. Além disso, ele permite a colaboração em equipe, possibilitando o compartilhamento de coleções e ambientes de teste. Atualmente, o Postman é uma ferramenta indispensável no fluxo de trabalho de desenvolvedores e engenheiros de qualidade, sendo utilizado por milhões de usuários ao redor do mundo.

As vantagens do Postman incluem sua interface intuitiva, que reduz a complexi-

dade do teste de APIs, e suas capacidades avançadas de automação e documentação. Comparado a outras ferramentas, como cURL ou alternativas mais básicas, o Postman oferece uma experiência mais integrada e acessível, especialmente para equipes que precisam colaborar em projetos complexos. No desenvolvimento deste projeto, o Postman foi escolhido por sua facilidade de uso e suas capacidades de automação de testes, o que contribui para um processo de desenvolvimento mais eficiente e menos propenso a erros.

Portanto o Postman se consolida como uma ferramenta essencial para o desenvolvimento e manutenção de APIs, oferecendo funcionalidades abrangentes que facilitam o trabalho de desenvolvedores e equipes de qualidade. Sua utilização neste projeto destaca a importância de ferramentas que otimizam o fluxo de trabalho, garantindo maior eficiência e qualidade no desenvolvimento de sistemas modernos.

5 ANÁLISE DOS RESULTADOS OBTIDOS

Instruções da Coordenação do PFC:
Neste capítulo, deve-se:

- Descrever detalhadamente como foi feita a implementação/desenvolvimento da solução proposta;
- Deixar bem claro e justificar tecnicamente para o leitor como que o desenvolvimento realizado implementa de fato a solução proposta, explicando tecnicamente as decisões que foram tomadas para se chegar a tal implentação;
- Analisar os resultados obtidos com base em indicadores, gráficos, estatísticas, etc:
  - A implementação realizada solucionou de fato o problema tratado?
  - Obteve-se o resultado esperado?
  - Mostrou-se melhor que o método anterior?
  - Vantagens e desvantagens;
  - Problemas encontrados;
  - Impacto dos resultados obtidos nos processos/projetos/produtos/serviços/clientes da empresa/instituto de pesquisa;
  - Impactos organizacionais, tecnológicos, financeiros, éticos, ecológicos, etc.

Sugere-se colocar uma diagrama/fluxograma ilustrando como que a solução proposta foi implementada/desenvolvida, e depois explicar em detalhes cada parte/bloco do diagrama/fluxograma ao longo do texto.

Ressaltamos que, em princípio, existe uma infinidade de maneiras diferentes de implementar a solução proposta. Desse modo, o diagrama/fluxograma da solução proposta apresentado no capítulo anterior é mais geral e abstrato que o diagrama/fluxograma da implementação: a implementação realizada no PFC é uma maneira específica de se chegar à solução proposta a partir das técnicas, ferramentas e métodos utilizados.

Alguns pontos que serão explicados nesse capítulo:

- **Análise de resultados**: Análise dos resultados obtidos com o desenvolvimento da aplicação. Detalhando os principais aspesctos relevantes, como a eficiência do agendamento de horários, a segurança das informações e a integração com IoT.

6 CONCLUSÃO

Instruções da Coordenação do PFC:
A Conclusão deve apresentar:

- Uma **síntese** do problema tratado, da solução proposta e dos principais resultados obtidos;
- Uma discussão sobre o que foi atingido no PFC em relação aos objetivos inicialmente traçados e sobre as limitações encontradas;
- Uma discussão sobre possíveis aprimoramentos;
- Destacar os impactos do PFC para a empresa/clientes da empresa/instituto de pesquisa, além de possíveis impactos organizacionais, tecnológicos, financeiros, éticos, ecológicos, etc.
- Sugestão de trabalhos futuros (como se poderia dar continuidade ao PFC; aplicar o desenvolvimento realizado no PFC a outros problemas/processos; etc)

REFERÊNCIAS

ARAUJO, Lauro César. **A classe abntex2**: Modelo canônico de trabalhos acadêmicos brasileiros compatível com as normas ABNT NBR 14724:2011, ABNT NBR 6024:2012 e outras. [S.l.], 2015. Disponível em: http://www.abntex.net.br/. Acesso em: 16 ago. 2019.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 10520**: Informação e documentação — Citações em documentos — Apresentação. Rio de Janeiro, ago. 2002a.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 14724**: Informação e documentação — Trabalhos acadêmicos — Apresentação. Rio de Janeiro, mar. 2011.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: Informação e documentação — Referências — Apresentação. Rio de Janeiro, ago. 2002b.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6024**: Informação e documentação — Numeração progressiva das seções de um documento escrito — Apresentação. Rio de Janeiro, fev. 2012a.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6027**: Informação e documentação — Sumário — Apresentação. Rio de Janeiro, dez. 2012b.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6028**: Informação e documentação — Resumo — Apresentação. Rio de Janeiro, nov. 2003.

ERL, Thomas; PUTTINI, Ricardo; MAHMOOD, Zaigham. **Cloud Computing: Concepts, Technology Architecture**. [S.l.]: Pearson, 2013. p. 528. ISBN 9780133387520.

FIELDING, Roy T. **Architectural styles and the design of network-based software architectures**. [S.l.]: UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

FOWLER, M. **Patterns of Enterprise Application Architecture**. [S.l.]: Addison-Wesley, 2002.

GOKHALE, Pradyumna; BHAT, Omkar; BHAT, Sagar. Introduction to IOT. **International Advanced Research Journal in Science, Engineering and Technology**, v. 5, n. 1, p. 41–44, 2018.

IBGE. **Arranjos populacionais e concentrações urbanas no Brasil**. 2. ed. Rio de Janeiro: [s.n.], 2016. Disponível em: https://biblioteca.ibge.gov.br/visualizacao/livros/liv99700.pdf. Acesso em: 16 ago. 2019.

IBGE. **Normas de apresentação tabular**. 3. ed. Rio de Janeiro: Centro de Documentação e Disseminação de Informações., 1993. Disponível em: http://biblioteca.ibge.gov.br/visualizacao/livros/liv23907.pdf. Acesso em: 16 ago. 2019.

MARTIN, Robert C. Design principles and design patterns. **Object Mentor**, v. 1, n. 34, p. 597, 2000.

MARTIN, Robert C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. [S.l.]: Pearson, 2017. ISBN 0134494164.

MASSE, Mark. **REST API design rulebook**. [S.l.]: "O'Reilly Media, Inc.", 2011.

TURNER, M.; BUDGEN, D.; BRERETON, P. Turning software into a service. **Computer**, v. 36, n. 10, p. 38–44, 2003. DOI: 10.1109/MC.2003.1236470.

APÊNDICE A – DESCRIÇÃO 1

Textos elaborados pelo autor, a fim de completar a sua argumentação. Deve ser precedido da palavra APÊNDICE, identificada por letras maiúsculas consecutivas, travessão e pelo respectivo título. Utilizam-se letras maiúsculas dobradas quando esgotadas as letras do alfabeto.

ANEXO A – DESCRIÇÃO 2

São documentos não elaborados pelo autor que servem como fundamentação (mapas, leis, estatutos). Deve ser precedido da palavra ANEXO, identificada por letras maiúsculas consecutivas, travessão e pelo respectivo título. Utilizam-se letras maiúsculas dobradas quando esgotadas as letras do alfabeto.