

**CENTRO PAULA SOUZA  
FACULDADE DE TECNOLOGIA DE MOGI MIRIM**

**Brenda Ingrid Gaudêncio  
Gabriel Coelho Soares  
Marcos Moreira Martins**

**START IOT**

**Plataforma Web e Tecnologia IoT na Sincronização e Controle de  
Largada e Gestão de Tempo na Descida da Ladeira**

**Mogi Mirim  
2025**

**Brenda Ingrid Gaudêncio**  
**Gabriel Coelho Soares**  
**Marcos Moreira Martins**

## **START IOT**

Plataforma Web e Tecnologia IoT na Sincronização e Controle de  
Largada e Gestão de Tempo na Descida da Ladeira

Trabalho de Graduação apresentado ao  
Curso de Tecnologia em Análise e De-  
senvolvimento de Sistemas da Faculdade  
de Tecnologia de Mogi Mirim como pré-  
requisito para a obtenção do Título de Tec-  
nólogo em Análise e Desenvolvimento de  
Sistemas. **Orientador:** Prof. Me. Marcio

Rodrigues Sabino

**Mogi Mirim**  
**2025**

**FICHA CATALOGRÁFICA**

(Elaborada pela Biblioteca)

*Ao Professor Fioravante Willi Nesto, nosso muito obrigado por idealizar a "Descida da Ladeira" em 2013, inspirando o espírito de equipe e a multidisciplinaridade que fundamentaram este projeto.*

## **AGRADECIMENTOS**

Texto dos agradecimentos...

*“A única coisa que a Internet das Coisas  
ainda não faz é achar a meia que sumiu na  
lavanderia”*

— The Primeagen

## **RESUMO**

Texto do resumo em um único parágrafo, contendo introdução, metodologia, resultados e conclusões. Máximo de 500 palavras.

**Palavras-chave:** Palavra1; Palavra2; Palavra3; Palavra4; Palavra5.

## **ABSTRACT**

Translation of the abstract in English. **Keywords:** Keyword1; Keyword2; Keyword3; Keyword4; Keyword5.

## **LISTA DE FIGURAS**

3.1 Título da figura . . . . .	6
--------------------------------	---

## **LISTA DE TABELAS**

3.1 Título da tabela . . . . .	6
--------------------------------	---

## **LISTA DE ABREVIATURAS E SIGLAS**

**ABNT** Associação Brasileira de Normas Técnicas

**TCC** Trabalho de Conclusão de Curso

# **Sumário**

## **AGRADECIMENTOS**

## **RESUMO**

## **ABSTRACT**

## **LISTA DE ABREVIATURAS E SIGLAS**

<b>1 INTRODUÇÃO</b>	<b>1</b>
1.1 Objetivos . . . . .	1
1.1.1 Objetivo Geral . . . . .	1
1.1.2 Objetivos Específicos . . . . .	1
<b>2 METODOLOGIA</b>	<b>2</b>
2.1 Pesquisa Bibliográfica . . . . .	3
2.2 Design Thinking . . . . .	3
2.3 Scrum . . . . .	4
2.4 Cronograma . . . . .	5
<b>3 FUNDAMENTAÇÃO TEÓRICA</b>	<b>6</b>
3.1 Seção . . . . .	6
3.1.1 Subseção . . . . .	6
<b>4 METODOLOGIA</b>	<b>7</b>
<b>5 RESULTADOS ESPERADOS</b>	<b>8</b>
<b>6 CONCLUSÃO</b>	<b>9</b>
<b>A PRIMEIRO APÊNDICE</b>	<b>11</b>
<b>ANEXO A – Título do Anexo</b>	<b>12</b>

# **1 INTRODUÇÃO**

Texto da introdução. Deve conter o problema, justificativa, quadro teórico de referência, objetivos e hipótese.

## **1.1 Objetivos**

### **1.1.1 Objetivo Geral**

Texto do objetivo geral.

### **1.1.2 Objetivos Específicos**

- Objetivo específico 1;
- Objetivo específico 2;
- Objetivo específico 3.

## 2 METODOLOGIA

A metodologia constitui o conjunto de procedimentos sistemáticos e técnicas empregadas para conduzir uma investigação científica, definindo o caminho percorrido para alcançar os objetivos propostos e garantir a confiabilidade dos resultados obtidos. Segundo Lakatos and Marconi(2017), “a metodologia é o conjunto das atividades sistemáticas e racionais que, com maior segurança e economia, permite alcançar o objetivo - conhecimentos válidos e verdadeiros -, traçando o caminho a ser seguido, detectando erros e auxiliando as decisões do cientista”. Assim, para o contexto do desenvolvimento de sistemas tecnológicos aplicados à gestão de eventos esportivos, a definição metodológica torna-se fundamental para estabelecer critérios rigorosos de análise, implementação e validação das soluções propostas.

A complexidade multidisciplinar dos projetos contemporâneos de tecnologia demanda uma evolução das abordagens metodológicas tradicionais em direção a *frameworks* híbridos que integrem rigor científico com flexibilidade adaptativa. Conforme destaca Brown(2008), “Design Thinking é uma disciplina que utiliza a sensibilidade e métodos do designer para encontrar/descobrir o que as pessoas necessitam, juntamente com o que é tecnologicamente praticável e o que é viável para os negócios”. Esta perspectiva complementa a metodologia tradicional ao focar na experiência do usuário e na inovação centrada no ser humano, aspectos essenciais para o desenvolvimento de sistemas efetivos em ambientes esportivos. Adicionalmente, Souza et al.(2020) Souza, Cavassini, and Sabino demonstram que “a metodologia design thinking propõe soluções inovadoras dirigindo o foco às necessidades do público alvo, e a metodologia Scrum, eficiente a complementa, no sentido da organização e produção de softwares complexos”.

Este trabalho caracteriza-se pela integração de múltiplas abordagens metodológicas, considerando tanto a natureza técnica do desenvolvimento de sistemas IoT quanto a necessidade de compreender profundamente o contexto organizacional do evento Descida da Ladeira. A pesquisa classifica-se como aplicada quanto à natureza, pois visa “gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos” (Prodanov and Freitas(2013)), e adota abordagem quali-quantitativa para contemplar tanto a análise dos requisitos funcionais quanto a mensuração de eficiência dos processos automatizados. O framework metodológico proposto integra pesquisa bibliográfica para fundamentação teórica (Gil(2017)), Design Thinking para descoberta e definição de soluções centradas no usuário (Vianna et al.(2012) Vianna, Vianna, Adler, e desenvolvimento ágil baseado em Scrum para implementação iterativa e validação contínua das funcionalidades do sistema.

Esta pesquisa caracteriza-se, também, como exploratória, uma vez que lidar com o problema da cronometragem automatizada em eventos esportivos de pequeno e médio porte, área ainda pouco investigada no contexto acadêmico nacional. Gil(2017) esclarece que “pesquisas exploratórias têm como propósito proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a construir hipóteses”. A natureza exploratória justifica-se pela necessidade de investigar tecnologias emergentes como redes LoRa e protocolos MQTT em contextos esportivos educacionais, além de mapear os requisitos específicos dos múltiplos stakeholders envolvidos - organizadores, participantes e espectadores - para construir uma solução tecnológica que atenda efetivamente às demandas identificadas.

A escolha por metodologias ágeis no desenvolvimento deste projeto fundamenta-se na necessidade de adaptabilidade e validação contínua inerente aos projetos de inovação tecnológica aplicada. Severino(2017) ressalta que “a metodologia científica não pode ser entendida como um conjunto de regras estáticas, mas deve adaptar-se à natureza do objeto de estudo e aos objetivos da pesquisa”. Neste sentido, a integração entre Design Thinking e Scrum permite equilibrar o planejamento estruturado com a flexibilidade necessária para ajustes em tempo real durante o desenvolvimento, característica fundamental quando se trabalha com hardware IoT e comunicação sem fio, tecnologias que frequentemente apresentam comportamentos imprevistos durante a implementação prática.

A presente seção estrutura-se de forma a detalhar cada componente metodológico empregado no desenvolvimento desta pesquisa. Inicialmente, será apresentada a pesquisa bibliográfica que fundamentou teoricamente o trabalho, seguida pela descrição da aplicação do Design Thinking nas fases de descoberta e definição das soluções, e finalmente pela exposição dos procedimentos ágeis baseados em Scrum utilizados na implementação e validação do sistema. Esta organização visa demonstrar como a integração metodológica proposta viabiliza tanto o rigor científico quanto a inovação prática, atendendo simultaneamente aos requisitos acadêmicos e às demandas reais do evento esportivo estudado.

## **2.1 Pesquisa Bibliográfica**

Para dar fundamento ao trabalho, realizou-se uma pesquisa bibliográfica abrangente sobre os temas relacionados ao desenvolvimento de sistemas Web com integração em componentes IoT para a gestão de eventos esportivos. Deste modo, garantimos o contato adequado com fontes seguras já estabelecidas, como Prodanov and Freitas(2013) recomendam: “A pesquisa bibliográfica coloca o pesquisador em contato direto com toda a produção escrita sobre a temática que está sendo estudada”.

Todo o levantamento desta base para o trabalho foram realizados nas bases de dados do Google Scholar, livros digitais e físicos, tendo o recorte temporal priorizado a partir dos anos 2000, considerando a evolução das tecnologias Web, a gestão de eventos e os avanços em pesquisas com IoT.

A pesquisa abrangendo toda bibliografia já tornada pública em relação ao tema (Lakatos and Marconi(2017)), notou-se uma lacuna significativa na literatura nacional sobre aplicações de redes LoRa em eventos esportivos de pequeno e médio porte, justificando a relevância do projeto e da pesquisa, permitindo identificar o estado e as oportunidades de contribuição científica.

## **2.2 Design Thinking**

O Design Thinking constitui uma abordagem centrada no ser humano para a resolução criativa de problemas, amplamente adotada no desenvolvimento de produtos e serviços inovadores. Brown(2008) define Design Thinking como “uma disciplina que usa a sensibilidade e os métodos do designer para suprir as necessidades das pessoas com o que é tecnologicamente factível”, convertendo estratégias de negócios em valor para os *stakeholders*.

A escolha do Design Thinking como abordagem metodológica complementar para este projeto justifica-se pela natureza inovadora da solução proposta, que demanda compreensão profunda das necessidades dos usuários do evento Descida da Ladeira e dos organizadores. Conforme destaca Vianna et al.(2012)Vianna, Vianna, Adler, Lucena o Design Thinking traz uma visão holística para a inovação através de equipes multidisciplinares que compreendem consumidores no contexto onde se encontram.

O processo de Design Thinking adotado neste trabalho seguiu o modelo em cinco fases proposto pela d.school de Stanford: (1) Empatia - para compreender necessidades de competidores, organizadores e público; (2) Definição - para estabelecer claramente os problemas a serem resolvidos pelo sistema e sua integração com a IoT; (3) Ideação - para gerar alternativas de solução técnica; (4) Prototipação - para desenvolver versões testáveis do sistema; e (5) Teste - para validar a solução com usuários reais em ambiente controlado (Interaction Design Foundation(2025)).

Cabe destacar que, conforme enfatizam Vianna et al.(2012)Vianna, Vianna, Adler, Lucena, as etapas do Design Thinking possuem natureza não-linear, permitindo que sejam moldadas conforme a natureza específica do projeto. Assim, ao longo do desenvolvimento, foi possível realizar ciclos iterativos, unidos ao SCRUM, retornando a fases anteriores sempre que novos insights eram obtidos, característica fundamental desta abordagem metodológica.

### 2.3 Scrum

O Scrum, embora se enquadre como metodologia, é mais conhecido como um framework leve para gerenciamento de projetos. Amplamente utilizado para o desenvolvimento de projetos, ele “ajuda pessoas, equipes e organizações a gerar valor através de soluções adaptativas para problemas complexos” (Schwaber and Sutherland(2020)).

A escolha deste framework para se alinhar com o desenvolvimento do projeto, munido da metodologia de Design Thinking, justifica-se pela natureza iterativa e incremental do trabalho, onde os requisitos e solicitações foram se refinando ao longo do processo de desenvolvimento. Assim, o Scrum possibilitou a entrega funcional, ainda que parcial, que puderam ser validadas pelos *stakeholders* do evento Descida da Ladeira, bem como da defesa deste material de estudo.

A compreensão do Scrum é simples, pois se baseia em três responsabilidades fundamentais dentro da equipe que participam do projeto onde, segundo Schwaber and Sutherland são: (1) o *Product Owner* ou Dono do Produto que é responsável por maximizar o valor do produto resultante do trabalho do time; (2) o *Scrum Master* responsável por ajudar todos do time a entenderem a teoria e a prática do Scrum; e (3) os Desenvolvedores responsáveis por criar o plano da sprint e garantir a qualidade do produto. Assim, neste projeto, definimos estas responsabilidades do PO para o orientador, que possui o contato com os organizadores do evento, tendo a plena capacidade de definir as prioridades e validar as entregas. Já as duas outras responsabilidades foram rotativas durante todo o projeto entre os integrantes de defesa do material de estudo, definindo semanalmente os papéis de cada um para aquela *sprint*.

O método incremental que o Scrum adota, compõe-se de quatro atividades a serem realizadas: *Sprint Planning* que definirá um planejamento do ciclo; *Daily Scrums* sendo as reuniões diárias de no máximo quinze minutos; *Sprint Review* uma reunião semanal para discutir os avanços; e o *Sprint Retrospective* que visa trazer

reflexões e validações do orientador para a melhoria do processo.

Ainda conforme Schwaber and Sutherland(2020) o Scrum possui alguns artefatos, que para o projeto foram escolhidos três, sendo: o *Product Backlog* - uma listagem com prioridade das funcionalidades a serem implementadas; o *Sprint Backlog* - como um subconjunto da primeira lista com itens selecionados para o ciclo de Sprint; e o *Increment*, entregue em *releases* do Github, como versões funcionais do produto gerado no fim de cada sprint.

## 2.4 Cronograma

O cronograma de desenvolvimento do projeto foi estruturado para contemplar todas as fases metodológicas propostas, distribuídas ao longo de dois semestres acadêmicos conforme apresentado no Quadro ??.

[htb]

2*Tarefas	2025 - Semestre 2						2026 - Semestre 1					
	8	9	10	11	12	1	2	3	4	5	6	7
Busca de tema, imersão com o orientador	X	X										
Imersão: Conhecer o problema		X	X									
Ideação: Proposta de solução			X	X								
Pesquisa e escrita de introdução			X	X	X							
Pesquisa e escrita do referencial teórico				X	X	X						
Escrita do capítulo de desenvolvimento					X	X	X					
Prototipação: Elaboração de diagramas						X	X	X				
Escrita do capítulo de resultados							X	X	X			
Testes de bancada							X	X				
Escrita: documento TG1								X	X			
Qualificação do TG1 (banca)									X			
Desenvolvimento do sistema backend/frontend										X	X	X
Teste no evento com o público-alvo										X		
Adaptações e correções										X	X	
Escrita: documento TG2										X	X	
Entrega e Defesa do TG2												X

**Fonte:** Dos próprios autores (2025).

### **3 FUNDAMENTAÇÃO TEÓRICA**

O presente capítulo tem como objetivo apresentar a fundamentação teórica que embasa este trabalho, por meio da revisão de conceitos, metodologias e tecnologias relacionadas ao tema em estudo. Serão abordados os principais referenciais da literatura científica, contemplando definições, princípios e estudos já consolidados, a fim de oferecer suporte teórico para a compreensão do problema e a construção da solução proposta.

#### **3.1 Engenharia de Software**

A Engenharia de Software emerge como disciplina estruturada em resposta à chamada "Crise do Software" da década de 1960, quando o aumento da complexidade dos sistemas evidenciou a necessidade de métodos sistemáticos para o desenvolvimento, operação e manutenção de software (PRESSMAN; MAXIM, 2021). Diferentemente da programação isolada, a engenharia de software estabelece processos, técnicas e ferramentas que garantem a construção de sistemas confiáveis, eficientes e que atendam às necessidades dos usuários. Seu objetivo central é produzir software de qualidade dentro de prazos e custos estabelecidos, considerando não apenas a codificação, mas todo o ciclo de vida do produto, desde a concepção até a manutenção (SOMMERVILLE, 2019). Historicamente, o campo evoluiu de modelos prescritivos e sequenciais, como o modelo Cascata, para abordagens iterativas e incrementais, culminando nas metodologias ágeis contemporâneas, que valorizam a adaptação rápida a mudanças e a colaboração contínua com stakeholders. Essa evolução reflete a necessidade de maior flexibilidade diante da dinâmica acelerada do mercado tecnológico e das expectativas dos usuários por entregas frequentes e incrementais de valor (BECK et al., 2001, tradução nossa). A engenharia de software, portanto, não se limita a aspectos técnicos, mas engloba gestão de pessoas, processos e tecnologias, sendo fundamental para o sucesso de projetos de qualquer escala ou complexidade.

##### **3.1.1 Levantamento de Requisitos**

O levantamento de requisitos constitui a fase inicial e crítica do desenvolvimento de software, na qual são identificadas e documentadas as necessidades e expectativas dos stakeholders em relação ao sistema a ser construído (SOMMERVILLE, 2019). Requisitos são declarações que descrevem o que o sistema deve realizar e sob quais condições deve operar, sendo tradicionalmente categorizados em requisitos funcionais e não funcionais. Os requisitos funcionais especificam as funcionalidades que o sistema deve oferecer, tais como operações de cadastro, consulta, processamento de dados e geração de relatórios, representando as ações observáveis que o sistema executa em resposta a entradas ou eventos (PRESSMAN; MAXIM, 2021). Já os requisitos não funcionais estabelecem critérios de qualidade que o sistema deve satisfazer, abrangendo aspectos como desempenho, segurança, usabilidade, escalabilidade, confiabilidade e manutenibilidade. Esses requisitos definem restrições sobre como o sistema implementa suas funcionalidades, sendo muitas vezes determinantes para a

satisfação dos usuários e o sucesso da aplicação em ambientes reais. A elicitação de requisitos demanda o emprego de técnicas estruturadas que facilitem a comunicação entre desenvolvedores e stakeholders. Entre as técnicas amplamente utilizadas destacam-se entrevistas, que permitem obter informações detalhadas diretamente de usuários e especialistas do domínio; questionários, adequados para coletar dados de um número maior de participantes de forma padronizada; workshops colaborativos, nos quais diferentes partes interessadas discutem e negociam requisitos em conjunto; prototipação, que viabiliza a validação precoce de conceitos por meio de versões simplificadas do sistema; e análise de documentação existente, útil quando há sistemas legados ou processos já estabelecidos (SOMMERVILLE, 2019). A escolha das técnicas varia conforme o contexto do projeto, a natureza do domínio e a disponibilidade dos stakeholders. A documentação clara e precisa dos requisitos, frequentemente formalizada em especificações de requisitos de software, é essencial para evitar re trabalho e garantir que o produto final atenda às expectativas, constituindo a base para todas as etapas subsequentes do desenvolvimento (PRESSMAN; MAXIM, 2021).

### 3.1.2 Modelagem do Sistema

A modelagem de sistemas de software consiste na criação de representações abstratas que facilitam a compreensão, a comunicação e a análise da estrutura e do comportamento do sistema antes de sua implementação. A *Unified Modeling Language* (UML), consolidada como padrão internacional pela *Object Management Group* em meados dos anos 1990, oferece um conjunto rico de diagramas para modelagem de sistemas orientados a objetos (BOOCH; RUMBAUGH; JACOBSON, 2005, tradução nossa). A UML organiza-se em diagramas estruturais, que representam aspectos estáticos do sistema, e diagramas comportamentais, que capturam sua dinâmica e interações. Entre os diagramas mais relevantes no contexto de engenharia de software, destacam-se o Diagrama de Casos de Uso, o Diagrama de Classes, o Diagrama de Sequência e o Diagrama de Atividades, cada um atendendo a propósitos específicos no processo de análise e projeto. O Diagrama de Casos de Uso documenta as funcionalidades do sistema sob a perspectiva dos usuários finais, identificando atores (agentes externos que interagem com o sistema) e casos de uso (funcionalidades específicas fornecidas pelo sistema). Esse diagrama é particularmente útil na fase de levantamento de requisitos, pois fornece uma visão de alto nível do escopo funcional e facilita a comunicação com stakeholders não técnicos (SOMMERVILLE, 2019). O Diagrama de Classes, por sua vez, representa a estrutura estática do sistema, especificando as classes que compõem a solução, seus atributos, métodos e os relacionamentos entre elas, como associações, agregações, composições e heranças. Este diagrama é fundamental para o projeto orientado a objetos, pois define a arquitetura lógica do sistema e orienta a implementação (FOWLER, 2003, tradução nossa). O Diagrama de Sequência ilustra a interação temporal entre objetos ao longo da execução de um caso de uso ou cenário específico, evidenciando a ordem em que mensagens são trocadas e os métodos invocados. Esse tipo de modelagem é valioso para compreender fluxos de controle complexos e para identificar dependências entre componentes (BOOCH; RUMBAUGH; JACOBSON, 2005, tradução nossa). Finalmente, o Diagrama de Atividades modela o fluxo de trabalho ou processos dentro do sistema, representando atividades, decisões, bifurcações e sincronizações. Esse

diagrama é útil tanto para documentar processos de negócio quanto para descrever a lógica interna de operações complexas, apoiando a compreensão de fluxos não lineares e condições de execução (FOWLER, 2003, tradução nossa). Em conjunto, esses diagramas UML constituem uma base sólida para a documentação arquitetural e facilitam a transição da análise de requisitos para o projeto detalhado e a implementação.

### **3.1.3 Arquitetura e Tecnologias**

A arquitetura de software refere-se à estrutura fundamental de um sistema, definindo seus componentes principais, as propriedades externamente visíveis desses componentes e os relacionamentos entre eles (BASS; CLEMENTS; KAZMAN, 2021, tradução nossa). A escolha arquitetural tem impacto direto sobre atributos de qualidade como desempenho, escalabilidade, manutenibilidade e segurança, sendo uma das decisões mais críticas no ciclo de desenvolvimento. Diferentes estilos arquiteturais oferecem trade-offs específicos e são adequados a contextos distintos. A arquitetura cliente-servidor, por exemplo, separa responsabilidades entre clientes que solicitam serviços e servidores que os proveem, promovendo centralização de dados e lógica de negócio. Já a arquitetura em camadas organiza o sistema em níveis hierárquicos, onde cada camada oferece serviços à camada superior e consome serviços da camada inferior, facilitando a separação de responsabilidades e a manutenibilidade (PRESSMAN; MAXIM, 2021). A arquitetura de microsserviços, popularizada nas últimas duas décadas, propõe a decomposição do sistema em serviços pequenos, independentes e fracamente acoplados, cada um responsável por uma funcionalidade específica. Essa abordagem favorece escalabilidade horizontal, resiliência e facilitação de deploys frequentes, alinhando-se a práticas DevOps e ao desenvolvimento ágil (NEWMAN, 2021, tradução nossa). Além da estrutura arquitetural, a seleção de tecnologias é igualmente determinante para o sucesso do projeto. A escolha de linguagens de programação, frameworks, sistemas gerenciadores de banco de dados e ferramentas de desenvolvimento deve considerar fatores como maturidade da tecnologia, disponibilidade de bibliotecas e comunidade ativa, compatibilidade com requisitos não funcionais e expertise da equipe de desenvolvimento (BASS; CLEMENTS; KAZMAN, 2021, tradução nossa). Decisões arquiteturais mal fundamentadas podem resultar em sistemas difíceis de manter, com baixo desempenho ou incapazes de escalar adequadamente, evidenciando a importância de uma análise criteriosa e documentada desde as fases iniciais do projeto.

### **3.1.4 Metodologias e Processos de Desenvolvimento**

As metodologias de desenvolvimento de software definem processos estruturados que orientam as atividades de concepção, construção, teste e entrega de sistemas. Historicamente, predominaram abordagens prescritivas e sequenciais, como o modelo Cascata, no qual o desenvolvimento ocorre em fases bem delimitadas (levantamento de requisitos, projeto, implementação, teste e manutenção), com entregas únicas ao final do ciclo (PRESSMAN; MAXIM, 2021). Embora proporcione previsibilidade e documentação extensa, esse modelo revela-se pouco adaptável a mudanças, sendo mais adequado a projetos com requisitos estáveis e bem compreendidos

desde o início. O modelo Espiral, proposto por Boehm, introduz iterações e análise de riscos, mas ainda mantém características de planejamento extenso e documentação detalhada (SOMMERVILLE, 2019). A partir do final dos anos 1990, metodologias ágeis emergiram como resposta às limitações dos modelos tradicionais, priorizando indivíduos e interações sobre processos e ferramentas, software funcionando sobre documentação abrangente, colaboração com o cliente sobre negociação contratual e resposta a mudanças sobre seguir um plano (BECK et al., 2001, tradução nossa). Entre as metodologias ágeis mais difundidas, destaca-se o Scrum, um framework iterativo e incremental que organiza o trabalho em sprints de duração fixa, promovendo entregas frequentes, feedback contínuo e adaptação rápida a mudanças de requisitos. O Extreme Programming (XP) enfatiza práticas técnicas como programação em pares, desenvolvimento orientado a testes e integração contínua, visando elevar a qualidade do código e a satisfação do cliente. Já o Kanban foca na visualização do fluxo de trabalho e na limitação do trabalho em progresso, otimizando a eficiência e reduzindo gargalos (BECK; ANDRES, 2004, tradução nossa). Complementarmente, o Design Thinking pode ser integrado ao processo de desenvolvimento como abordagem centrada no ser humano para identificação de problemas e geração de soluções inovadoras, promovendo empatia com usuários e prototipação rápida (BROWN, 2008, tradução nossa). A escolha da metodologia mais adequada depende do contexto do projeto, da natureza dos requisitos, da maturidade da equipe e das expectativas dos stakeholders, sendo comum a adoção de práticas híbridas que combinam elementos de diferentes abordagens.

### **3.1.5 Validação e Testes**

A validação e verificação de software são atividades essenciais para assegurar que o sistema construído atende aos requisitos especificados e está livre de defeitos que comprometam seu funcionamento. A verificação responde à questão "estamos construindo o produto corretamente?", enquanto a validação responde "estamos construindo o produto correto?"(SOMMERVILLE, 2019). Testes de software constituem o principal mecanismo de validação e verificação, sendo classificados em diferentes níveis conforme o escopo de aplicação. Os testes unitários focam na validação de componentes individuais, como funções ou classes, isoladamente, verificando se cada unidade produz os resultados esperados para entradas específicas. Testes de integração avaliam a interação entre módulos ou componentes, identificando falhas na comunicação entre partes do sistema. Testes de sistema validam o comportamento do sistema completo em relação aos requisitos funcionais e não funcionais, enquanto testes de aceitação verificam se o software atende às expectativas dos usuários finais e stakeholders, sendo frequentemente conduzidos em ambientes reais de operação (PRESSMAN; MAXIM, 2021). Além da classificação por nível, os testes podem ser categorizados quanto à técnica empregada. Testes de caixa-branca, também conhecidos como testes estruturais, examinam a lógica interna do código, buscando exercitar diferentes caminhos de execução, condições e laços. Testes de caixa-preta, ou testes funcionais, concentram-se nas entradas e saídas do sistema, sem considerar sua estrutura interna, sendo baseados exclusivamente nas especificações de requisitos (SOMMERVILLE, 2019). A prototipação, embora não seja um teste formal, desempenha papel relevante na validação precoce de conceitos e na obtenção de feedback dos

usuários antes da implementação completa, reduzindo riscos de retrabalho (PRESSMAN; MAXIM, 2021). A execução eficaz de testes é suportada por ferramentas automatizadas que aumentam a produtividade e a confiabilidade do processo. Frameworks como JUnit para Java, Jest para JavaScript e Pytest para Python facilitam a criação e execução de testes unitários. Para testes de interface de usuário, ferramentas como Selenium e Cypress permitem automatizar interações com elementos visuais. Testes de APIs são comumente realizados com ferramentas como Postman e Insomnia, que validam requisições HTTP e respostas de serviços web. Métricas de qualidade, como cobertura de testes (percentual de código exercitado pelos testes) e densidade de defeitos (número de defeitos por mil linhas de código), auxiliam na avaliação objetiva da qualidade do software e na tomada de decisões sobre sua liberação para produção (SOMMERVILLE, 2019). Em suma, um processo rigoroso de validação e testes é indispensável para a entrega de software confiável, seguro e alinhado às expectativas dos usuários.

### 3.2 Linguagem Java

A linguagem de programação Java, lançada pela Sun Microsystems em 1995, consolidou-se como uma das tecnologias mais influentes no desenvolvimento de software contemporâneo, sendo amplamente adotada em aplicações empresariais, sistemas embarcados e dispositivos móveis (DEITEL; DEITEL, 2017). Concebida com o lema "Write Once, Run Anywhere" (Escreva uma vez, execute em qualquer lugar), Java introduziu o conceito de portabilidade por meio da Java Virtual Machine (JVM), que permite que programas compilados em bytecode sejam executados em qualquer plataforma que possua uma implementação da JVM, abstraindo as particularidades de cada sistema operacional. A linguagem fundamenta-se nos paradigmas de programação orientada a objetos, promovendo encapsulamento, herança e polimorfismo como pilares para a construção de software modular e reutilizável. Adicionalmente, Java incorpora gerenciamento automático de memória através do coletor de lixo (garbage collector), reduzindo erros relacionados a vazamentos de memória e aumentando a segurança e confiabilidade dos sistemas desenvolvidos (SIERRA; BATES, 2005). Ao longo das décadas, a linguagem evoluiu significativamente, incorporando recursos modernos como expressões lambda, streams e programação funcional a partir do Java 8, mantendo-se relevante diante das demandas tecnológicas atuais. O ecossistema Java caracteriza-se pela diversidade de frameworks e bibliotecas que facilitam o desenvolvimento de aplicações robustas e escaláveis. Entre as soluções mais relevantes, destaca-se o Spring Framework, surgido em 2002 como resposta à complexidade dos Enterprise JavaBeans (EJB), oferecendo uma abordagem mais leve e flexível baseada em Inversão de Controle e Injeção de Dependências. O Spring Framework revolucionou a maneira como aplicações Java são estruturadas, promovendo baixo acoplamento entre componentes e facilitando a testabilidade do código (DEITEL; DEITEL, 2017). A partir de 2014, o projeto Spring Boot emergiu como uma extensão do Spring Framework, com o objetivo de simplificar drasticamente o processo de configuração e inicialização de aplicações. Fundamentado no princípio de convenção sobre configuração, o Spring Boot oferece auto-configuração inteligente, dependências inicializadoras (starters) e servidores de aplicação embutidos, permitindo que desenvolvedores concentrem-se na lógica de negócios em vez de em configurações repetitivas.

(WALLS, 2016, tradução nossa). Essa abordagem alinha-se às metodologias ágeis de desenvolvimento, nas quais a entrega rápida de valor ao cliente é priorizada, reduzindo significativamente o tempo de desenvolvimento de protótipos e aplicações em produção. A combinação de Java com Spring Boot tornou-se padrão de facto para o desenvolvimento de APIs RESTful, microsserviços e sistemas distribuídos na indústria de tecnologia. O Spring Boot oferece integração nativa com tecnologias amplamente utilizadas, incluindo bancos de dados relacionais e não relacionais, sistemas de mensageria, ferramentas de segurança e plataformas de nuvem, constituindo um ecossistema completo para aplicações modernas (DEITEL; DEITEL, 2017). A presença de uma comunidade ativa e extensa documentação oficial contribui para sua adoção em projetos de diferentes escalas e complexidades. Além disso, a arquitetura promovida pelo Spring Boot facilita a implementação de práticas DevOps, como integração contínua e implantação automatizada, essenciais para a manutenção de sistemas em ambientes de produção dinâmicos. Em síntese, Java e seu ecossistema, especialmente por meio do Spring Boot, consolidam-se como ferramentas indispensáveis para o desenvolvimento de software empresarial moderno, oferecendo produtividade, escalabilidade e alinhamento com as melhores práticas de engenharia de software.

### 3.3 Linguagem React

O ReactJS, frequentemente denominado apenas React, é uma biblioteca JavaScript de código aberto amplamente empregada no desenvolvimento de interfaces de usuário para aplicações *web*. Em sua definição essencial, React é “uma biblioteca JavaScript para desenvolver interfaces de usuário” (STEFANOV, 2016, tradução nossa). Diferentemente de *frameworks* mais abrangentes, o React se concentra unicamente na camada de visualização (*View*), adotando uma abordagem declarativa para renderizar dados de modo eficiente e consistente. Em vez de manipular o *Document Object Model* (DOM) de forma imperativa, o desenvolvedor descreve apenas o estado final desejado da interface, cabendo ao React realizar as atualizações necessárias (WIERUCH, 2024, tradução nossa). A arquitetura do React se sustenta em três pilares conceituais principais. O primeiro e mais fundamental é a componentização. Nesse modelo, a interface do usuário é decomposta em blocos independentes e reutilizáveis denominados componentes. Segundo Lerner et al. (2017), os componentes “são os elementos fundamentais das aplicações React, que capacitam os desenvolvedores a construírem componentes de interface de usuário modulares e passíveis de reutilização”. Cada componente encapsula tanto sua lógica quanto sua apresentação, promovendo a modularidade e a manutenção do código, além de possibilitar a construção de interfaces complexas a partir de unidades menores e coesas. O segundo pilar da arquitetura é o *Virtual DOM* (VDOM), responsável pelo alto desempenho na renderização das interfaces. Para otimizar o processo de atualização da interface, o React utiliza o VDOM como uma representação leve e mantida em memória do DOM real do navegador (STEFANOV, 2016, tradução nossa). Quando o estado de um componente sofre alterações, o React atualiza primeiramente o VDOM e, em seguida, realiza uma operação de *diffing* (comparação) entre a versão anterior e a nova. Apenas as diferenças identificadas são refletidas no DOM real. Essa estratégia reduz significativamente as operações diretas sobre o DOM, conhecidas por serem lentas, resultando em renderizações mais rápidas e eficientes (FLANAGAN, 2020, tradução nossa). O terceiro

pilar é o JSX (JavaScript XML), uma extensão de sintaxe que simplifica o desenvolvimento de interfaces no React. De acordo com Wieruch (2024, tradução nossa), o JSX é “uma extensão de sintaxe para JavaScript usada no React que permite aos desenvolvedores escreverem código semelhante ao HTML diretamente dentro do JavaScript”. Essa característica promove a integração entre a lógica de renderização e a estrutura visual do componente, tornando o código mais legível e favorecendo a compreensão da hierarquia de elementos. A adoção do React em projetos de desenvolvimento web contemporâneos justifica-se por sua eficiência técnica, escalabilidade e maturidade consolidada no mercado. Wieruch (2024, p. 7-8) destaca que, com a ascensão do React, o conceito de componentes popularizou-se no desenvolvimento web, onde cada componente encapsula sua apresentação visual e comportamento, permitindo reutilização em hierarquias complexas para construir aplicações completas. Embora o React mantenha foco como biblioteca de componentes, seu ecossistema circundante o transforma em um framework flexível, caracterizado por uma API enxuta e um ecossistema estável em constante evolução. Nesse contexto, o React evoluiu para além de uma biblioteca de interface, tornando-se um ecossistema completo que suporta o desenvolvimento *full-stack* por meio de integrações com tecnologias complementares, como o Next.js, React Native e diversas soluções de gerenciamento de estado. Conforme destaca Derks (2022, tradução nossa), o React representa atualmente “um ecossistema integral que viabiliza o desenvolvimento completo (*full-stack*), estendendo suas funcionalidades para além dos limites convencionais do frontend”. Essa abrangência o consolida como uma das tecnologias mais relevantes e adotadas no desenvolvimento web moderno, sustentando aplicações escaláveis, performáticas e com experiências de usuário aprimoradas.

### 3.4 API – Application Programming Interface

A arquitetura de software contemporânea é amplamente dependente da comunicação eficiente e padronizada entre diferentes aplicações, um conceito materializado pela API (Application Programming Interface), ou Interface de Programação de Aplicações (ORACLE, 2024). Essencialmente, a API é um contrato de serviço que define um conjunto de regras, protocolos e ferramentas para a interação entre um cliente e um servidor. Sua principal função é atuar como um intermediário, que permite que aplicações troquem dados e usem funcionalidades umas das outras de forma segura e desacoplada, sem que a aplicação cliente precise conhecer a complexidade de implementação interna do servidor (AWS, 2024). Este modelo de separação de responsabilidades é vital para a construção de sistemas distribuídos robustos, como preconizado por Coulouris et al. (2013). O mecanismo de funcionamento de uma API é fundamentado no modelo de requisição-resposta. A aplicação cliente inicia a comunicação enviando uma requisição à API, especificando a ação desejada sobre um recurso. A API, após validação e autorização, encaminha a solicitação ao sistema servidor, que processa a lógica de negócios e devolve os dados resultantes. Por fim, a API formata esses dados, geralmente utilizando a notação leve JSON (JavaScript Object Notation), e os entrega ao cliente como resposta. Na arquitetura web, o estilo dominante para estruturar essa comunicação é o REST (Representational State Transfer). Formalizado por Roy Fielding em sua tese de doutorado no ano 2000, o REST é um conjunto de restrições arquitetônicas que conferem ao sistema características como escalabilidade

e confiabilidade (FIELDING, 2000, tradução nossa). As APIs que aderem a essas restrições, denominadas RESTful, utilizam o protocolo HTTP para manipular recursos (identificados por Endpoints - URLs específicas) por meio de métodos padronizados, como GET para leitura e POST para criação. Uma restrição crucial do REST é a ausência de estado (Stateless), exigindo que cada requisição do cliente inclua todas as informações necessárias para ser processada, o que simplifica o gerenciamento de sessões no lado do servidor (FIELDING, 2000, tradução nossa). Em suma, a API se estabelece como um facilitador da interoperabilidade e da inovação. Ao fornecer um ponto de comunicação estável e documentado, as APIs promovem a reutilização de código e a aceleração do desenvolvimento, permitindo que sistemas utilizem serviços complexos de terceiros — como pagamento, autenticação ou geolocalização — sem a necessidade de construí-los do zero. Além disso, elas são essenciais para o desacoplamento entre a lógica de negócios e a interface do usuário, assegurando que o sistema seja modular, escalável e apto a ser consumido por múltiplos tipos de clientes (web, mobile, dispositivos IoT), alinhando-se às boas práticas de arquitetura de software distribuído (RICHARDS; FORD, 2020).

### 3.5 Banco de Dados

A gestão eficiente de dados constitui um pilar fundamental para sistemas de informação, sendo materializada por meio de Sistemas Gerenciadores de Banco de Dados (SGBDs), que são softwares especializados projetados para criar, manipular e administrar grandes volumes de dados de forma estruturada e segura (ELMASRI; NAVATHE, 2019). Historicamente, a evolução dos SGBDs acompanhou o crescimento exponencial da produção de dados, partindo de sistemas hierárquicos e em rede nas décadas de 1960 e 1970, até a consolidação do modelo relacional, proposto por Edgar F. Codd em 1970, que introduziu fundamentos matemáticos baseados na teoria dos conjuntos e na álgebra relacional (DATE, 2004). Esse modelo estabeleceu-se como padrão da indústria devido à sua capacidade de representar relacionamentos complexos entre entidades de forma intuitiva e flexível. Atualmente, observa-se também o crescimento de bancos de dados não relacionais, conhecidos como NoSQL, que atendem demandas específicas de escalabilidade horizontal e flexibilidade de esquema, características essenciais em aplicações de big data e sistemas distribuídos (SILBERSCHATZ; KORTH; SUDARSHAN, 2020). O modelo relacional organiza informações em tabelas compostas por linhas e colunas, onde cada tabela representa uma entidade do domínio do problema, e as colunas definem os atributos dessa entidade. As relações entre tabelas são estabelecidas por meio de chaves primárias e estrangeiras, garantindo integridade referencial e eliminando redundâncias por meio do processo de normalização (ELMASRI; NAVATHE, 2019). A manipulação desses dados é realizada através da Structured Query Language (SQL), uma linguagem declarativa padronizada pela ISO que permite operações de consulta, inserção, atualização e exclusão de dados de forma eficiente e expressiva. Os SGBDs relacionais implementam o conceito de transações, que são conjuntos de operações executadas de forma atômica, consistente, isolada e durável, propriedades conhecidas pelo acrônimo ACID (DATE, 2004). Essas garantias são cruciais para a confiabilidade de sistemas críticos, como instituições financeiras e sistemas de saúde, onde a integridade dos dados não pode ser comprometida. Além disso, os SGBDs oferecem mecanismos de controle

de concorrência, que asseguram que múltiplos usuários possam acessar e modificar dados simultaneamente sem causar inconsistências. A escolha adequada do sistema de banco de dados impacta diretamente a performance, escalabilidade e manutenibilidade de aplicações modernas. Em sistemas web e APIs RESTful, os bancos de dados relacionais frequentemente atuam como camada de persistência, armazenando informações que são expostas através de endpoints HTTP, enquanto bancos NoSQL são preferidos em cenários que demandam alta disponibilidade e particionamento de dados (SILBERSCHATZ; KORTH; SUDARSHAN, 2020). No contexto de Internet das Coisas, a capacidade de processar grandes volumes de dados gerados continuamente por dispositivos distribuídos exige estratégias híbridas que combinam SGBDs relacionais para dados estruturados e soluções NoSQL para dados de telemetria em tempo real. A integração entre bancos de dados e frameworks de desenvolvimento, facilitada por tecnologias como Object-Relational Mapping (ORM), abstrai a complexidade das operações de banco, permitindo que desenvolvedores manipulem dados por meio de objetos de programação, acelerando o desenvolvimento e reduzindo erros (ROB; CORONEL, 2011). Em síntese, os SGBDs são componentes essenciais da arquitetura de software contemporânea, viabilizando a construção de sistemas confiáveis, escaláveis e alinhados às necessidades de negócio.

### 3.6 Internet das Coisas IoT

A Internet das Coisas, representa uma tecnologia que possibilita a interconexão de dispositivos físicos por meio de redes de comunicação, permitindo a coleta, transmissão e processamento de dados. Segundo Queiroz et al. (2023), a IoT pode ser compreendida como a infraestrutura que possibilita a conexão e comunicação entre dispositivos inteligentes, com o objetivo de coletar, transmitir e receber informações de maneira coordenada, transformando objetos convencionais em entidades digitais capazes de interagir entre si e com o ambiente. Os sistemas IoT fundamentam-se em três pilares funcionais principais: a conectividade ubíqua, que permite a comunicação contínua entre dispositivos por meio de tecnologias sem fio como Bluetooth, WiFi, LoRa e redes celulares; a capacidade de sensoriamento ambiental, onde sensores convertem grandezas físicas em sinais digitais; e o processamento distribuído de dados, que envolve tanto processamento local nos dispositivos (*edge computing*) quanto processamento centralizado em servidores remotos (*cloud computing*), otimizando latência e consumo de recursos (VASCONCELOS FILHO et al., 2023; QUEIROZ et al., 2023). Os sistemas IoT estruturam-se em arquiteturas em camadas que organizam funcionalidades específicas. Queiroz et al. (2023) propõem uma arquitetura de cinco camadas fundamentais: camada de dispositivo (sensores, atuadores e microcontroladores); camada de comunicação (protocolos de transmissão de dados); camada de rede (infraestrutura de conectividade); camada de aplicação (interface com usuário e lógica de negócio); e camada de segurança (privacidade e confiança na transmissão). Essa estruturação permite modularidade e escalabilidade, facilitando a integração de diferentes tecnologias. A comunicação entre as camadas utiliza protocolos especializados que atendem diferentes requisitos de largura de banda, latência, alcance e consumo energético. Embora protocolos como MQTT (*Message Queuing Telemetry Transport*) e CoAP (*Constrained Application Protocol*) sejam amplamente utilizados, a escolha deve considerar as especificidades de cada aplicação (VASCONCELOS FI-

LHO et al., 2023). Para aplicações que demandam comunicação de longa distância em ambientes externos, tecnologias como LoRa apresentam vantagens significativas em termos de alcance e consumo energético. A aplicação de tecnologias IoT em eventos esportivos tem se consolidado como estratégia para automação de processos e melhoria da precisão de medições. A Maratona de Boston utiliza tecnologia RFID (Radio-Frequency Identification) UHF para cronometragem automática de mais de 30 mil corredores simultaneamente, eliminando a necessidade de intervenção manual. Entretanto, a tecnologia RFID apresenta limitações em termos de alcance (tipicamente inferior a 10 metros) e custo elevado por dispositivo (DTB NFC, 2025). Na Fórmula 1, a IoT atinge níveis ainda mais elevados, com cada veículo equipado com mais de 300 sensores que geram aproximadamente 1,1 milhão de pontos de dados por segundo. Esses dados são transmitidos em tempo real via telemetria wireless, permitindo monitoramento de desempenho mecânico e otimização de estratégias de corrida (AWS SPORTS, 2025; CATAPULT, 2025). Os benefícios da aplicação de IoT em cronometragem incluem: precisão aumentada na medição de tempos, eliminando variações decorrentes de erro humano; automação de processos, reduzindo dependência de comunicação manual; rastreabilidade completa, permitindo registro detalhado com *timestamp* sincronizado; e escalabilidade, facilitando expansão para eventos com maior número de participantes. Os desafios técnicos envolvem conectividade em ambientes externos, onde interferências podem comprometer a comunicação wireless; autonomia energética para operação contínua; confiabilidade operacional, exigindo mecanismos de redundância; e sincronização temporal, crucial para cronometragem precisa em sistemas distribuídos (VASCONCELOS FILHO et al., 2023).

### 3.7 Embarcamento

Os sistemas embarcados, constituem sistemas computacionais microprocessados especializados, projetados para executar funções específicas dentro de um dispositivo ou equipamento maior. Segundo Almeida, Moraes e Seraphim (2016), diferentemente dos computadores de propósito geral, os sistemas embarcados caracterizam-se pela integração completa entre *hardware* e *software*, operando de forma dedicada a tarefas predefinidas. Constatase que esses sistemas se fundamentam em três pilares: especialização funcional, otimização de recursos e operação em tempo real, características que os tornam adequados a aplicações com restrições de espaço físico, consumo energético e capacidade de processamento. Conforme Denardin e Barriquello (2018, p. 38), os sistemas embarcados são completamente encapsulados e dedicados ao dispositivo ou sistema que controlam, realizando um conjunto de tarefas predefinidas com requisitos específicos. Verifica-se que esses sistemas apresentam atributos distintos que os diferenciam de sistemas computacionais tradicionais, destacando-se a operação com recursos limitados de memória e processamento, a resposta a eventos externos em tempo real e a necessidade de alta confiabilidade e estabilidade. Ademais, esses sistemas operam frequentemente em ambientes restritos ou hostis, demandando robustez e eficiência energética. Observa-se ainda que grande parte dos sistemas embarcados são projetados para funcionar continuamente, sem intervenção humana, executando suas funções de forma autônoma e ininterrupta. A arquitetura de um sistema embarcado comprehende elementos fundamentais de hardware e software integrados para atender a requisitos funcionais específicos.

No âmbito do hardware, Almeida, Moraes e Seraphim (2016) destacam que os microcontroladores constituem o componente central desses sistemas, integrando em um único chip a unidade de processamento, memória e periféricos de entrada e saída. Diferentemente dos microprocessadores, que demandam componentes externos para operação completa, os microcontroladores oferecem uma solução compacta e econômica, ideal para aplicações com restrições de espaço e custo. Quanto à memória, verifica-se a presença de memória volátil para processamento de dados temporários e memória não volátil para armazenamento permanente do programa e configurações do sistema. As arquiteturas clássicas de sistemas embarcados baseiam-se em dois modelos fundamentais. A arquitetura de von Neumann caracteriza-se pelo armazenamento de instruções e dados em um mesmo espaço de endereçamento, compartilhando um único barramento de comunicação. Em contraste, a arquitetura de Harvard emprega espaços de endereçamento distintos para programa e dados, possibilitando acesso simultâneo e, consequentemente, maior desempenho em aplicações que demandam processamento intensivo. Observa-se que a escolha entre essas arquiteturas depende dos requisitos específicos da aplicação, considerando fatores como velocidade de processamento, complexidade do sistema e restrições de custo. O desenvolvimento de software para sistemas embarcados apresenta particularidades que o distinguem da programação convencional. Segundo Almeida, Moraes e Seraphim (2016, p. 23), as linguagens de programação mais empregadas nesse contexto são C, C++ e Assembly, selecionadas por sua eficiência, controle próximo do hardware e capacidade de gerenciar recursos limitados de forma otimizada. A linguagem C destaca-se por sua grande utilização na comunidade de desenvolvedores embarcados, oferecendo equilíbrio entre controle de baixo nível e produtividade de desenvolvimento. Adicionalmente, linguagens como Python aparecem como alternativas para prototipagem rápida, embora com maior consumo de recursos computacionais. O ciclo de desenvolvimento de sistemas embarcados compreende etapas distintas e iterativas. Inicialmente, o código-fonte é escrito em uma linguagem de programação e, posteriormente, compilado por ferramentas específicas que geram código executável adequado ao microcontrolador alvo. O código compilado é então transferido para a memória não volátil do dispositivo através de interfaces de programação especializadas. A crescente integração dos sistemas embarcados com tecnologias de conectividade, especialmente no contexto da Internet das Coisas, expande significativamente suas possibilidades de aplicação. Denardin e Barriquello (2018) destacam que os sistemas embarcados são a espinha dorsal de inovações tecnológicas modernas, presentes em dispositivos que variam desde eletrodomésticos até veículos autônomos e equipamentos médicos críticos. Esses sistemas executam funções específicas com restrições temporais bem definidas, interagindo com eventos externos síncronos ou assíncronos, características essenciais para aplicações que demandam determinismo e confiabilidade. Na questão de eventos e monitoramento em tempo real, os sistemas embarcados viabilizam aplicações diversas. Em eventos esportivos, dispositivos embarcados integrados a sensores possibilitam rastreamento de atletas, monitoramento de condições ambientais e coleta automática de dados de desempenho. Na automação industrial, controlam processos críticos com alta confiabilidade e resposta em tempo real. Em dispositivos médicos garantem operação precisa e segura, processando sinais vitais continuamente. Entretanto, o desenvolvimento e implementação de sistemas embarcados apresentam desafios consideráveis. Destacam-se as restrições de recursos computacionais, demandando otimização rigorosa do código e eficiê-

cia no uso de memória e processamento. A necessidade de operação confiável em ambientes adversos exige robustez no design de hardware e software. Ademais, em aplicações conectadas, aspectos de segurança da informação tornam-se críticos, requerendo implementação de mecanismos de criptografia, autenticação e atualizações seguras de firmware.

### 3.8 LoRa

LoRa, abreviação de Long Range, constitui tecnologia de radiofrequência sem fio de longo alcance e baixo consumo energético, desenvolvida especificamente para aplicações de Internet das Coisas, possibilitando enlaces de dados extremamente longos que alcançam até cinco quilômetros em áreas urbanas e quinze quilômetros ou mais em ambientes rurais com linha de visada (SEMTECH CORPORATION, 2024, p. 4, tradução nossa). A tecnologia LoRa compreende a camada física de comunicação baseada na modulação *Chirp Spread Spectrum* (CSS), técnica derivada de aplicações militares desenvolvidas originalmente para sistemas de radar na década de 1940, que utiliza pulsos *chirp* modulados para codificar dados, demonstrando especial resiliência contra interferência, efeitos Doppler e multipercorso (SEMTECH CORPORATION, 2015, p. 9; LEA, 2018, tradução nossa). A modulação LoRa, patenteada pela Semtech em 2014, estabelece compromisso entre sensibilidade e taxa de dados operando em larguras de banda fixas de 125 kHz ou 500 kHz, permitindo que sinais *chirp* possuam amplitude constante e atravessem toda a largura de banda de forma linear, proporcionando robustez mesmo em ambientes com elevado nível de ruído eletrromagnético (HAXHIBEQIRI et al., 2018; SEMTECH CORPORATION, 2024, p. 7, tradução nossa). O consumo energético ultrabaixo constitui característica fundamental da tecnologia, permitindo que dispositivos alimentados por bateria alcancem vida útil de até dez anos (SEMTECH CORPORATION, 2024, p. 4-5, tradução nossa). A tecnologia LoRa fundamenta-se em princípios que priorizam eficiência energética e alcance ao custo da taxa de transferência de dados. Conforme Bertoleti (2019, p. 15), dispositivos LoRa operam em faixas de frequência não licenciadas, conhecidas como bandas ISM (*Industrial, Scientific and Medical*), que variam conforme regulamentações regionais. No Brasil, a Agência Nacional de Telecomunicações homologou as faixas de 902 a 907,5 MHz e 915 a 928 MHz para operação de dispositivos LoRa. Nota-se que as taxas de transmissão variam entre 0,3 e 50 kbps, característica ideal para aplicações que necessitam realizar envios constantes de pequenos pacotes de dados. LoRaWAN, vem de Long Range Wide Area Network, e compreende protocolo de rede de área ampla que opera sobre a camada física LoRa, definindo a arquitetura do sistema e os mecanismos de comunicação entre dispositivos. Segundo Bertoleti (2019, p. 18), LoRaWAN constitui protocolo aberto desenvolvido e mantido pela LoRa Alliance, organização global que padroniza e certifica dispositivos compatíveis. A arquitetura de rede LoRaWAN adota topologia em estrela, onde os dispositivos finais comunicam-se diretamente com gateways, que por sua vez encaminham os dados para um servidor de rede central através da internet. Esta configuração elimina a necessidade de sincronização entre nós e reduz o consumo energético dos dispositivos terminais. Nas especificações, LoRaWAN define três classes de dispositivos baseados em suas características operacionais. A Classe A representa a configuração padrão de menor consumo energético, onde os dispositivos transmitem dados de forma

assíncrona e abrem janelas de recepção apenas após transmissões próprias, sendo obrigatoriamente implementada por todos os dispositivos LoRaWAN. A Classe A permite comunicação bidirecional onde cada transmissão ascendente do dispositivo é seguida por duas janelas curtas de recepção descendente, constituindo a configuração de menor consumo energético e o conjunto básico de opções obrigatórias que todo dispositivo final necessita implementar. A Classe B estende capacidades da Classe A abrindo periodicamente janelas extras de recepção denominadas *ping slots* para receber mensagens descendentes, utilizando *beacons* sincronizados para derivar e alinhar seus relógios internos com a rede. A Classe C mantém janelas de recepção continuamente abertas exceto durante transmissões, possibilitando receber mensagens descendentes a qualquer momento e oferecendo latência muito baixa para enlaces servidor-dispositivo (LORA ALLIANCE, 2018, p. 9; HAXHIBEQIRI et al., 2018, tradução nossa). A rede transmite *beacons* a cada 128 segundos para dispositivos Classe B, garantindo comunicação em horários programados com melhor previsibilidade de latência (JABBAR et al., 2022, tradução nossa). Dispositivos Classe C não dependem de alimentação por bateria e operam continuamente ativos, resultando em consumo energético significativamente maior, sendo adequados para aplicações alimentadas diretamente por rede elétrica (SEMTECH CORPORATION, 2024, p. 25, tradução nossa). Esta classificação hierárquica permite adaptação da tecnologia conforme requisitos específicos da aplicação, estabelecendo equilíbrio entre consumo energético, latência de comunicação e responsividade do sistema (JABBAR et al., 2022, tradução nossa). A segurança constitui aspecto fundamental na especificação LoRaWAN. Bertoleti (2019, p. 21) destaca que o protocolo implementa criptografia AES (*Advanced Encryption Standard*) de cento e vinte e oito bits em duas camadas distintas. A *Application Session Key* protege o conteúdo dos dados entre dispositivo e servidor de aplicação, garantindo confidencialidade das informações transmitidas. A *Network Session Key* garante a integridade e autenticidade das mensagens, permitindo que o servidor de rede valide pacotes sem acessar seu conteúdo. Cada dispositivo possui identificador único, e o processo de ativação na rede demanda autenticação mútua, prevenindo acesso não autorizado. Em contrapartida, a tecnologia possui limitações intrínsecas em seu design. A baixa taxa de transferência de dados torna LoRa inadequada para aplicações que demandam transmissão contínua de grandes volumes de informação. Ademais, a natureza assíncrona da comunicação introduz latência que pode ser incompatível com sistemas de controle em tempo real de natureza crítica. Regulamentações de uso de espectro impõem restrições adicionais, limitando o tempo de transmissão por dispositivo para evitar saturação das faixas não licenciadas.

### 3.9 ESP32

O ESP32 constitui microcontrolador desenvolvido pela Espressif Systems em 2016 como sucessor do ESP8266. Trata-se de sistema computacional completo integrado em único *chip* de baixo custo e baixo consumo energético que integra processador *dual-core* Tensilica Xtensa LX6 de 32 bits operando a até 240 MHz, incorporando capacidade de processamento, memória e módulos de comunicação sem fio WiFi e Bluetooth (OLIVEIRA, 2021), característica que o posiciona como solução adequada para desenvolvimento de aplicações em Internet das Coisas. A família de microcontroladores ESP8266 e ESP32 da Espressif constitui opção extremamente in-

teressante para aplicações de Internet das Coisas devido ao custo reduzido e aos recursos suficientes para diversas implementações IoT (OLIVEIRA, 2021). O ESP32 caracteriza-se pela alta integração, incorporando comutadores de antena, balun RF, amplificador de potência, amplificador de recepção de baixo ruído, filtros e módulos de gerenciamento de energia, o que o posiciona como solução adequada para desenvolvimento de dispositivos conectados em Internet das Coisas (ESPRESSIF SYSTEMS, 2025). Na questão dos recursos de hardware, o ESP32 apresenta memória integrada suficiente para armazenamento de código de aplicação e dados operacionais. Disponibiliza múltiplos pinos de entrada e saída digitais configuráveis, conversores que permitem leitura de sensores analógicos e geração de sinais de controle modulados. Segundo Bertoleti (2019, p. 42), o dispositivo oferece diversas interfaces de comunicação padrão da indústria eletrônica, permitindo conexão com ampla variedade de sensores, *displays* e outros periféricos. O módulo WiFi permite conexão com redes domésticas e corporativas, possibilitando comunicação com internet e serviços em nuvem. O módulo Bluetooth integrado suporta tanto comunicação tradicional quanto modo de baixo consumo energético, viabilizando conexão com dispositivos móveis e periféricos diversos. Conforme Oliveira (2020, p. 91), esta dupla conectividade permite que o microcontrolador atenda simultaneamente requisitos de comunicação de longo alcance e proximidade, característica particularmente vantajosa em aplicações que demandam múltiplos canais de interação. O desenvolvimento de software para ESP32 beneficia-se de ecossistema abrangente de ferramentas. O ambiente oficial desenvolvido pela fabricante, baseado em sistema operacional de tempo real FreeRTOS, provê acesso completo aos recursos de hardware. Alternativamente, apresenta compatibilidade com plataforma Arduino, ambiente amplamente conhecido que facilita prototipagem rápida e reduz curva de aprendizado. Conforme Bertoleti (2019, p. 35), esta versatilidade de ambientes de desenvolvimento torna o microcontrolador acessível tanto para desenvolvedores experientes quanto para iniciantes na área de sistemas embarcados. No contexto de aplicações práticas, o ESP32 demonstra versatilidade em automação e Internet das Coisas. Em automação residencial, viabiliza controle remoto de iluminação, monitoramento de temperatura e umidade, além de integração com assistentes virtuais. Em ambientes industriais, possibilita aquisição de dados de sensores e comunicação com sistemas de supervisão. Em aplicações veiculares, permite interface com sistemas eletrônicos automotivos para diagnóstico e telemetria. Para dispositivos portáteis alimentados por bateria, oferece modos de operação de baixíssimo consumo energético, possibilitando autonomia prolongada quando o sistema permanece em estado de espera, acordando periodicamente para realizar medições e transmissões. Aspectos de segurança integram a arquitetura do dispositivo. Implementa recursos de criptografia acelerados por hardware, permitindo proteção eficiente de dados transmitidos e armazenados. Disponibiliza mecanismos para geração de números aleatórios seguros e proteção de código armazenado na memória. O suporte a protocolos de comunicação segura possibilita conexões protegidas com servidores remotos, aspecto crítico em aplicações conectadas à nuvem que manipulam informações sensíveis. Entretanto, o dispositivo apresenta limitações que devem ser consideradas. A necessidade de alimentação em tensão específica demanda circuitos reguladores adicionais quando integrado a sistemas que operam em tensões diferentes. O consumo energético durante transmissão contínua de dados sem fio, embora otimizado, pode comprometer a autonomia de aplicações alimentadas exclusivamente por bateria. Ademais, a riqueza de recursos disponíveis exige investimento maior em

aprendizado comparado a microcontroladores de arquitetura mais simples. Não obstante, a integração de processamento paralelo, conectividade WiFi e Bluetooth nativa, amplo ecossistema de desenvolvimento e custo reduzido consolidam o ESP32 como alternativa preferencial para implementação de sistemas embarcados conectados em aplicações de Internet das Coisas e automação.

## **4 DESENVOLVIMENTO**

Este capítulo apresenta o desenvolvimento do projeto fundamentado em duas metodologias consolidadas: Design Thinking e Scrum. A primeira orienta a progressão desde a compreensão profunda do problema (Imersão), passando pela estruturação de soluções (Ideação), até a materialização tecnológica (Prototipação). A segunda, aplicada especificamente na fase de Prototipação, organiza o trabalho técnico em entregas iterativas que permitiram a validação contínua das funcionalidades pelos stakeholders.

### **4.1 Imersão**

A etapa da imersão do Design Thinking, segundo Vianna et al. (2012), é o momento em que uma equipe de projeto busca aproximação com o contexto de um problema latente, identificando necessidades e oportunidades que nortearão a geração de soluções. Assim, esta fase do projeto teve como objetivo compreender os fundamentos e contexto do evento, bem como seus processos operacionais e os desafios enfrentados pela equipe. Afim de garantir uma boa imersão e ideação da situação atual, bem como identificar as lacunas atuais que necessitam de inovação tecnológica, foi aplicado um questionário estruturado à Comissão Organizadora do evento Descida da Ladeira, entre os dias 20 a 25 de outubro de 2025, totalizando 6 inquiridos entre professores e membros da organização com diferentes níveis de envolvimento e experiência com o evento.

#### **4.1.1 Contextualização do Evento**

O evento Descida da Ladeira teve sua primeira edição realizada em fevereiro de 2014, por iniciativa do Prof. Fioravante, com o objetivo de criar um momento de integração entre alunos de diversos cursos que estudavam em períodos diferentes. Ao longo de sua trajetória, o evento consolidou-se como projeto de extensão de caráter pedagógico na Fatec Mogi Mirim, tendo realizado um total de 10 edições até 2025, com uma interrupção motivada pela pandemia de COVID-19. Segundo os dados coletados, o evento mantém uma média consistente de 25 equipes participantes por edição, variando entre 20 e 30 competidores conforme a mobilização em cada ano. A competição envolve carrinhos de rolimã projetados e construídos pelos próprios estudantes, que descem a ladeira central da instituição em baterias eliminatórias até a definição dos vencedores, sendo facultada a participação de alunos de todos os cursos oferecidos pela unidade. O processo operacional do evento inicia-se na fase pré-competição com o sistema de inscrições, atualmente realizado através de formulário digital na plataforma Google Forms. Esta mudança do formato físico para o digital ocorreu durante o período da pandemia e foi mantida pelas vantagens operacionais observadas. No formulário, são coletadas informações essenciais como nome da equipe, identificação de um piloto responsável, identificação de um "motor"(membro responsável pelo empurrão inicial), relação completa dos membros da equipe com seus respectivos cursos e períodos, além da possibilidade de upload de uma imagem que servirá como lo-

gotipo oficial da equipe durante o evento. Paralelamente ao sistema de inscrições, é disponibilizado online através do site institucional um regulamento detalhado contendo todas as regras técnicas de segurança, especificações dos carrinhos, critérios de desclassificação e funcionamento das baterias.

#### **4.1.2 Fluxo do Evento**

No dia do evento, realizado tipicamente das 8h00 às 13h00, o fluxo operacional inicia-se com a organização dos espaços e demarcação precisa da pista de competição. A equipe organizadora distribui os diversos postos necessários ao longo do percurso, desde a linha de largada até a linha de chegada na ladeira central da Fatec. Após a liberação oficial da pista, realiza-se uma vistoria técnica detalhada de todos os carrinhos inscritos, verificando requisitos de segurança, conformidade com o regulamento e condições gerais dos equipamentos. Concluída a vistoria, inicia-se a fase de tomadas de tempo classificatórias, onde cada equipe realiza entre 2 e 3 descidas cronometradas. As baterias são organizadas com grupos de 3 carrinhos descendo simultaneamente, permitindo comparação direta de desempenho. Dependendo do número total de equipes participantes, o formato de eliminação pode incluir uma fase intermediária de classificação antes da final, onde são selecionados os três melhores tempos acumulados para disputar o título. A bateria final é disputada pelos três finalistas em descida única, sendo declarado vencedor aquele que cruzar primeiro a linha de chegada. O processo de cronometragem, elemento central e crítico da competição, ocorre de forma distribuída e manual entre dois pontos principais do percurso. Na linha de largada, posicionam-se dois fiscais de pista responsáveis por garantir o posicionamento correto dos carrinhos e acionar os dispositivos de sinalização. A largada oficial é marcada simultaneamente por uma buzina sonora e pelo abaixamento de uma bandeira, ambos operados manualmente pelos fiscais. A meio do trajeto, encontra-se o fiscal de bandeira, responsável por liberar visualmente a pista para o público. Na linha de chegada, um fiscal munido de cronômetro digital (tipicamente um smartphone com aplicativo de cronometragem) assume a função crítica de registrar os tempos. Ao ouvir a buzina de largada, este fiscal inicia manualmente o cronômetro e, à medida que cada carrinho cruza a linha de chegada, registra os três tempos individuais utilizando um cronômetro de múltiplos estágios. Os tempos capturados são então anotados em papel ou mantidos na tela do dispositivo e imediatamente transmitidos via aplicativo WhatsApp para a equipe de controle, responsável pelo lançamento manual dos dados em planilha Excel, cálculo de médias quando aplicável e elaboração da classificação atualizada para divulgação.

#### **4.1.3 Infraestrutura do Evento**

Quanto à infraestrutura disponível para realização do evento, identificou-se uma situação de recursos limitados mas adaptáveis. A área da ladeira utilizada para a competição não dispõe de infraestrutura elétrica permanente, sendo necessário adaptar ligações provisórias a partir de postes próximos especificamente no dia do evento. Esta adaptação fornece energia suficiente para os equipamentos essenciais como televisor, computador e sistema de som, mas apresenta limitações de capacidade e

distribuição espacial dos pontos de energia. Um desafio de infraestrutura particularmente crítico é a ausência completa de conectividade Wi-Fi estável na área da ladeira, impossibilitando qualquer solução que dependa de internet durante a competição. A distância entre os pontos de largada e chegada não foi quantificada precisamente nas respostas, mas é suficiente para criar desafios de comunicação e sincronização entre as equipes posicionadas em cada extremidade. Os recursos de comunicação atuais limitam-se a rádios convencionais e sistema de som com microfone para chamamento das equipes e comunicação com o público presente.

#### 4.1.4 Problemática

A partir da análise consolidada das 27 questões aplicadas e suas respectivas respostas, foi possível identificar e categorizar três grupos principais de problemas no processo operacional atual, com diferentes níveis de impacto e criticidade. Esta categorização baseou-se na frequência de menções pelos respondentes, no grau de preocupação expressado e na avaliação dos impactos diretos sobre a qualidade, justiça e fluidez da competição. Os problemas identificados distribuem-se entre questões de precisão técnica na captura de dados, deficiências nos fluxos de comunicação e informação, e limitações operacionais decorrentes da natureza manual e distribuída dos processos atuais. O primeiro e mais crítico problema identificado refere-se ao processo de cronometragem manual, apontado por todos os respondentes como fonte de preocupação quanto à precisão e confiabilidade dos resultados. A captura do tempo de chegada depende inteiramente da reação humana do fiscal, que deve observar visualmente o momento exato em que cada carrinho cruza a linha e acionar manualmente o cronômetro. Esta dependência do fator humano introduz uma margem de erro inerente devido ao tempo de reação, variações de atenção e diferenças de interpretação sobre o momento exato da passagem pela linha. Conforme relatado pelos respondentes, o processo torna-se "trabalhoso e repetitivo", especialmente considerando que cada bateria envolve três carrinhos e o evento pode ter mais de 20 equipes realizando múltiplas descidas. A incerteza de medição resultante cria uma situação onde diferenças de décimos ou mesmo centésimos de segundo - potencialmente decisivas para a classificação - podem ser questionadas pelos competidores, gerando contestações e comprometendo a percepção de justiça do processo competitivo. Um respondente sintetizou a questão afirmando que "a medição do tempo ocorre ainda de maneira um tanto arcaica, o que destoa do viés tecnológico que a unidade e seus cursos apregoam", evidenciando não apenas uma limitação técnica mas também uma incongruência entre o perfil institucional e os métodos empregados. O segundo problema identificado relaciona-se à latência significativa entre a captura dos tempos na linha de chegada e sua efetiva divulgação para participantes e público. O fluxo atual envolve múltiplas etapas sequenciais: o fiscal de chegada registra os tempos no cronômetro, anota ou fotografa os valores, envia via mensagem de WhatsApp para a equipe de controle, que então realiza o lançamento manual dos dados em planilha Excel, executa cálculos de média quando necessário, atualiza a classificação geral e finalmente projeta os resultados na televisão disponível ao público. Este processo, embora funcional, consome tempo considerável e gera um atraso perceptível entre a conclusão de cada bateria e o conhecimento de seus resultados. Conforme apontado pelos respondentes, "a divulgação de cada corrida geralmente não acon-

tece, somente é divulgado no som de rádio local no dia do evento os participantes que avançaram para próximas fases", e os resultados são mostrados na TV "só após lançados", sem qualquer visualização em tempo real do andamento da competição. Esta latência impacta negativamente a experiência tanto dos competidores, que ficam em suspense aguardando seus resultados, quanto do público, que perde o dinamismo de acompanhar instantaneamente o desempenho de cada equipe. Adicionalmente aos problemas de cronometragem e divulgação, identificou-se uma deficiência significativa na comunicação operacional com as equipes participantes durante o evento. O sistema atual baseia-se em chamamento via microfone e rádio para convocar as equipes que devem se posicionar para a próxima bateria. Entretanto, conforme relatado explicitamente por um dos respondentes, "frequentemente os membros não ouvem o chamamento no microfone", gerando atrasos na sequência das baterias, necessidade de chamamentos repetidos e eventual desorganização do cronograma do evento. Este problema de comunicação contribui para as dificuldades de "gestão de tempo" citadas como um dos maiores desafios operacionais do dia da competição. A natureza difusa do áudio em ambiente aberto, combinada com a dispersão natural das equipes pela área do evento enquanto aguardam sua vez, cria uma situação onde a responsabilidade de estar atento aos chamamentos recai inteiramente sobre os participantes, sem um mecanismo confiável de notificação ou confirmação de recebimento da convocação.

#### **4.1.5 Interações Tecnológicas Anteriores**

O levantamento também revelou um histórico de tentativas anteriores de incorporação de tecnologia ao evento, com resultados mistos que fornecem lições importantes para o projeto atual. Segundo os respondentes, duas iniciativas tecnológicas foram implementadas em edições passadas: um sistema de sirene e luzes para sinalização de largada, que "funcionou bem" e permanece em uso até hoje; e uma tentativa de cronometragem automática baseada em um pórtico eletrônico instalado na linha de chegada, que "não funcionou bem" e foi descontinuada. Adicionalmente, foi mencionado o sistema desenvolvido no TCC de Eugênio realizado em 2025, embora os respondentes não tenham fornecido detalhes sobre suas funcionalidades específicas ou resultados de uso, indicando que o conhecimento sobre este sistema não está uniformemente distribuído entre a comissão organizadora ou que sua implementação foi limitada. O uso atual de Excel para lançamento de resultados e TV para divulgação representa a linha de base tecnológica que tem sido funcional, ainda que com as limitações já descritas de latência e trabalho manual intensivo. A experiência mal-sucedida com a tentativa de automação completa via pórtico eletrônico demonstra que soluções excessivamente complexas ou que removem inteiramente a supervisão humana podem não ser adequadas ao contexto específico do evento. As razões para a falha do sistema de pórtico não foram detalhadas pelos respondentes, mas tipicamente sistemas deste tipo enfrentam desafios relacionados à sensibilidade dos sensores, calibração para diferentes condições ambientais (luminosidade, velocidade variável dos carrinhos), necessidade de manutenção técnica especializada e dificuldade de validação imediata em caso de leituras duvidosas. Esta experiência negativa criou, naturalmente, certa cautela entre os organizadores quanto a propostas de automação, como evidenciado pela preocupação expressa sobre a "confiabilidade" de

sistemas automatizados. Entretanto, os mesmos respondentes manifestaram entusiasmo com a possibilidade de cronometragem automática com sensores, desde que "funcione" adequadamente, e sugeriram explicitamente que a confiabilidade poderia ser assegurada "mantendo-se em paralelo a medição manual, para conferência". Esta receptividade condicionada indica uma preferência por abordagens híbridas que combinam tecnologia e supervisão humana, maximizando precisão sem eliminar a possibilidade de validação.

#### **4.1.6 Propostas de Soluções**

A partir da análise consolidada dos dados coletados na fase de imersão e considerando as experiências anteriores com tecnologia no evento, identificou-se que as necessidades prioritárias para o novo sistema concentram-se em três frentes principais que respeitam as severas limitações de infraestrutura identificadas e mantêm conscientemente o elemento humano no processo de validação dos resultados. Esta abordagem alinha-se à lição aprendida com a falha do sistema de pórtico automático, optando por uma estratégia de digitalização híbrida que potencializa as capacidades humanas através da tecnologia ao invés de tentar substituí-las completamente. O escopo definido para o projeto foca, portanto, em resolver os problemas de latência, integração de dados e comunicação operacional, deixando explicitamente de fora a automação completa da cronometragem com sensores, que demandaria infraestrutura e validação além das possibilidades atuais do evento. A primeira necessidade prioritária identificada é a integração da cronometragem manual ao sistema digital, eliminando as etapas intermediárias de transcrição e transmissão que atualmente consomem tempo e introduzem possibilidades de erro. O novo sistema deve permitir que o fiscal de chegada, munido de um dispositivo digital, insira os tempos capturados diretamente no sistema através de uma interface simplificada e intuitiva, adequada ao contexto de uso em ambiente externo e sob a pressão temporal da competição. Esta entrada direta deve substituir imediatamente o fluxo atual de anotação em papel, fotografia de cronômetros, envio via WhatsApp e lançamento manual em Excel, reduzindo drasticamente o número de etapas e a possibilidade de erros de transcrição. É importante ressaltar que esta funcionalidade mantém o aspecto manual e humano da captura do tempo através do cronômetro, validado e aceito ao longo das 10 edições do evento, mas moderniza e agiliza tudo que ocorre após esta captura inicial, transformando um processo analógico-digital fragmentado em um fluxo digital integrado. A segunda necessidade prioritária refere-se à divulgação de resultados em tempo real, eliminando a latência atualmente experimentada entre a conclusão de uma bateria e o conhecimento de seus resultados por participantes e público. O sistema deve expor dados atualizados instantaneamente sobre a classificação geral, resultados da bateria mais recente, próximas equipes a competir e outras informações relevantes. A atualização em tempo real significa que, no momento em que o fiscal de chegada confirma a entrada dos tempos no sistema, estes dados tornam-se imediatamente disponíveis em todas as interfaces conectadas, sem necessidade de processamento manual intermitente. Esta funcionalidade transforma radicalmente a experiência do evento, permitindo que a emoção de cada descida seja acompanhada instantaneamente por seus resultados objetivos. A terceira e mais inovadora necessidade identificada relaciona-se à redução da latência na comunicação entre os pontos operacionais do evento,

especificamente entre a linha de chegada e a linha de largada, utilizando tecnologia de comunicação de longo alcance LoRa (Long Range). Atualmente, a comunicação entre estes pontos depende de rádios convencionais sujeitos a interferências, alcance limitado e necessidade de operação manual. O sistema proposto implementará uma rede LoRa onde a estação de chegada, operada pelo cronometrista, terá a capacidade de acionar remotamente a buzina de largada, estabelecendo uma sincronização precisa e eliminando a dependência de coordenação por rádio entre equipes fisicamente distantes. Este acionamento remoto resolve simultaneamente dois problemas: garante que o cronometrista esteja pronto e atento no exato momento da largada (pois é ele próprio quem a aciona), e elimina o delay de comunicação que existe no fluxo atual onde a equipe de largada aciona a buzina e precisa comunicar via rádio o momento exato para a equipe de chegada iniciar a cronometragem. A escolha pela tecnologia LoRa justifica-se pela necessidade de funcionar em ambiente sem Wi-Fi, pelo longo alcance necessário entre os pontos do percurso, pelo baixo consumo de energia que permite operação com baterias portáteis, e pela confiabilidade superior a rádios convencionais em ambientes com múltiplas fontes de interferência. Adicionalmente às três necessidades funcionais prioritárias, os respondentes foram unânimes em destacar requisitos técnicos não-funcionais essenciais que devem permeiar todas as decisões de arquitetura e implementação do sistema. O primeiro e mais crítico destes requisitos é a capacidade de operação completamente offline, sem qualquer dependência de conectividade com a internet durante o evento. Esta exigência decorre diretamente da constatação de que não há Wi-Fi estável disponível na área da ladeira e de que a conectividade móvel pode ser intermitente ou congestionada devido à concentração de pessoas no local. O sistema deve, portanto, ser arquitetado para funcionar de forma autônoma através de rede local, com todos os componentes comunicando-se diretamente entre si sem necessidade de servidores externos ou serviços em nuvem durante a operação. O segundo requisito essencial refere-se à resiliência e confiabilidade do sistema, que não pode apresentar falhas ou indisponibilidades durante o evento. Diferentemente de sistemas corporativos onde uma falha temporária pode ser contornada, o evento Descida da Ladeira ocorre uma única vez por ano em um período concentrado de aproximadamente 5 horas, não havendo possibilidade de "tentar novamente mais tarde". Por fim, o terceiro requisito crítico é a simplicidade operacional, considerando que o sistema será utilizado por pessoas sob pressão temporal e em ambiente externo, muitas vezes sem treinamento técnico aprofundado. As interfaces devem ser intuitivas, os fluxos de operação devem ser diretos e inequívocos, e o sistema como um todo deve "desaparecer" permitindo que os operadores foquem na condução do evento ao invés de lutar com a tecnologia.

#### 4.1.7 Conclusão da fase

Esta fase de imersão forneceu, portanto, uma compreensão abrangente e empiricamente fundamentada do contexto operacional do evento Descida da Ladeira, permitindo direcionar o desenvolvimento do sistema para solucionar problemas reais, prioritários e consensualmente reconhecidos pela equipe organizadora. A abordagem metodológica de coletar dados diretamente dos *stakeholders* através de questionário estruturado, complementada pela análise do histórico de tentativas tecnológicas anteriores, resultou em um conjunto bem definido de requisitos funcionais e não-funcionais

que orientarão as fases subsequentes de ideação e prototipação. Mais importante ainda, a imersão evidenciou que a solução tecnológica adequada para este contexto não é necessariamente a mais sofisticada ou automatizada, mas sim aquela que se adapta às limitações reais de infraestrutura, respeita os processos validados ao longo de 10 edições do evento, e foca em eliminar os gargalos específicos que causam os maiores impactos negativos na experiência de organizadores, competidores e público. A próxima fase de ideação partirá destes *insights* para explorar alternativas de arquitetura, justificar escolhas tecnológicas e detalhar como cada componente do sistema proposto atenderá às necessidades identificadas dentro das restrições estabelecidas.

## 4.2 Ideação

A etapa de Ideação, segundo Vianna et al. (2012), representa o momento de transformar os *insights* coletados na fase de Imersão em oportunidades concretas e soluções inovadoras. Nesta fase, a equipe de projeto analisa os dados levantados, sintetiza os principais desafios identificados e projeta alternativas tecnológicas que respondam às necessidades dos *stakeholders*. Assim, a Ideação constitui a ponte entre a compreensão profunda do problema e a materialização da solução proposta, fundamentando as escolhas arquiteturais e tecnológicas que orientarão o desenvolvimento do sistema. A análise das respostas obtidas no questionário aplicado junto à Comissão Organizadora revelou padrões consistentes que evidenciam tanto as potencialidades quanto às fragilidades do processo atual de gerenciamento do evento Descida da Ladeira. A síntese desses *insights* permitiu identificar cinco categorias principais de oportunidades de melhoria, as quais nortearam a concepção da solução tecnológica. O processo atual de cronometragem, realizado manualmente com cronômetros digitais e registro em planilhas, apresenta-se como o principal gargalo operacional identificado. Os organizadores destacaram a dificuldade em garantir precisão absoluta nos registros, especialmente em situações de múltiplas equipes descendendo simultaneamente ou em rápida sucessão. A transcrição manual dos tempos para planilhas eletrônicas introduz riscos de erro humano e atrasos na divulgação dos resultados, gerando períodos de incerteza para as equipes participantes. Ademais, a ausência de um sistema centralizado dificulta a validação cruzada dos dados e a manutenção de histórico confiável entre edições. A comunicação entre os organizadores ocorre predominantemente através de grupos de WhatsApp e rádios comunicadores durante o evento. Embora funcionais, esses canais apresentam limitações significativas: mensagens importantes podem ser perdidas no fluxo contínuo de conversas, não há garantia de que todos os membros da equipe recebam as informações, e a comunicação via rádio depende de proximidade física e disponibilidade de equipamentos. Os respondentes manifestaram interesse em mecanismos mais eficazes de broadcast de informações críticas, como mudanças de horário, resultados de baterias e convocações para vistoria. A publicação dos resultados atualmente ocorre de forma manual, com considerável intervalo entre o término de cada bateria e a divulgação oficial das classificações. Esse hiato temporal, embora compreensível dadas as limitações do processo manual, reduz o engajamento do público presente e das equipes já eliminadas, que frequentemente deixam o local sem conhecer os resultados finais. Os organizadores reconheceram que a modernização desse aspecto poderia aumentar significativamente a experiência de todos os envolvidos, transformando o evento em

algo mais dinâmico e transparente.

#### 4.2.1 Definição de Atores e Casos de Uso

A partir dos *insights* sintetizados, procedeu-se à identificação dos perfis de usuário que irão interagir com o sistema proposto, bem como das funcionalidades necessárias para atender suas necessidades específicas. Segundo Pressman e Maxim (2016), a modelagem de casos de uso constitui ferramenta fundamental para capturar requisitos funcionais sob a perspectiva dos usuários, facilitando a comunicação entre *stakeholders* técnicos e não-técnicos. Foram identificados quatro perfis principais de usuários, cada qual com necessidades e níveis de acesso distintos: a) O Organizador (Administrador) representa a Comissão Organizadora do evento, composta por professores e coordenadores responsáveis pela gestão completa. Este perfil possui acesso irrestrito a todas as funcionalidades do sistema, desde a configuração de uma nova edição até a geração de relatórios finais, incluindo a aprovação de inscrições, validação de tempos e gerenciamento de usuários. b) O Cronometrista (Juiz) corresponde ao membro da equipe responsável pela operação da cronometragem durante o evento. Sua interação com o sistema concentra-se nas funcionalidades de registro de tempos e sua validação, exigindo interface otimizada para agilidade e precisão nas operações realizadas sob pressão temporal. c) O Público (Espectador) engloba a comunidade acadêmica e visitantes presentes no evento. Para este perfil, o sistema oferece acesso de visualização aos resultados em tempo real e histórico de corridas, sem necessidade de autenticação, promovendo transparência e engajamento. A modelagem dos casos de uso organizou as funcionalidades do sistema em cinco módulos principais, conforme ilustrado na Figura X. O módulo de Gestão do Evento contempla as funcionalidades administrativas fundamentais: criação de novas edições anuais, configuração da estrutura de baterias (eliminatórias, semifinais e finais), definição de cronograma e geração automática de chaveamento das equipes. O módulo de Gestão de Equipes concentra as funcionalidades relacionadas ao ciclo de vida das inscrições: a importação das inscrições, o processo de aprovação e o registro de vistoria técnica dos carrinhos através de checklist.



vertopal\_8b17decc23bf4529b4e7a127829a3ec2/media/image4.png

Figura X - Diagrama de Casos de Uso do Sistema de Gerenciamento do Evento Descida da Ladeira  
Fonte: Dos próprios autores (2025).

O módulo de Cronometragem representa o núcleo operacional do sistema no dia do evento. Engloba a funcionalidade de iniciar corridas (marcando o momento da largada), o registro dos tempos de chegada através de interface otimizada e a validação posterior desses tempos pelos juízes. A decisão de manter a cronometragem manual, ao invés de implementar sensores automáticos, fundamentou-se na análise de custo-benefício e na confiabilidade já estabelecida do processo atual, direcionando os esforços de inovação para a otimização da interface de registro e para a transparência na divulgação dos dados. O módulo de Resultados e Comunicação foca na experiência do usuário final, oferecendo visualização de ranking atualizado em tempo real,

consulta ao histórico de tempos de baterias anteriores, geração de relatórios finais oficiais, e exportação de dados para análises externas. A atualização em tempo real do ranking constitui um dos diferenciais principais da solução proposta, eliminando os hiatos temporais atualmente existentes entre a conclusão das corridas e a divulgação dos resultados. Por fim, o módulo de Gestão de Usuários provê as funcionalidades de autenticação no sistema e gerenciamento de permissões, garantindo controle de acesso apropriado conforme o perfil de cada usuário. A estrutura hierárquica de atores, com relações de generalização entre Público, Cronometrista e Organizador, simplifica significativamente o modelo de permissões do sistema. Ao invés de duplicar casos de uso ou criar relacionamentos complexos, a herança de permissões garante que perfis superiores automaticamente possuem capacidades dos perfis inferiores, refletindo naturalmente a realidade operacional onde organizadores frequentemente assumem funções de cronometragem. A ausência de relacionamentos «*include*» e «*extend*» entre casos de uso, comuns em diagramas mais detalhados, justifica-se pelo nível conceitual desta modelagem na fase de Ideação. Tais relacionamentos poderão ser explicitados durante a Prototipação, quando as especificações textuais detalhadas de cada caso de uso revelarão dependências e extensões específicas.

#### **4.2.2 Modelagem Conceitual de Dados**

A estrutura de dados do sistema foi concebida considerando os requisitos funcionais identificados e as necessidades de rastreabilidade e auditoria inerentes a eventos competitivos. O Modelo Entidade-Relacionamento conceitual, apresentado na Figura Y, organiza as informações em cinco agrupamentos lógicos principais. O primeiro agrupamento, Gestão do Evento, estabelece a hierarquia fundamental através das entidades EVENTO e EDIÇÃO. A separação dessas entidades, ao invés de uma estrutura única, permite manter histórico completo de todas as realizações do evento ao longo dos anos, possibilitando análises comparativas e evolução temporal. Cada EVENTO (entidade permanente) possui múltiplas EDIÇÕES (instâncias anuais/semestrais), as quais por sua vez agregam as EQUIPES participantes e as BATERIAS que estruturam a competição. O segundo agrupamento, Participantes, modela a composição das equipes através das entidades EQUIPE, MEMBRO e CARRINHO. Cada EQUIPE é composta por múltiplos MEMBROS (alunos identificados por RA e função - piloto ou mecânico) e possui um único CARRINHO, sobre o qual são registradas as informações de vistoria técnica. A cardinalidade 1:1 entre EQUIPE e CARRINHO reflete a regra de negócio que determina uma equipe por carrinho. Figura Y - Modelo Entidade-Relacionamento Conceitual do Sistema

Fonte: Dos próprios autores (2025). O terceiro agrupamento, Competição, estrutura a organização temporal do evento através das entidades BATERIA e CORRIDA. Cada BATERIA (fase eliminatória, semifinal ou final) contém múltiplas CORRIDAS (descidas individuais), estabelecendo a hierarquia que permite gerenciar a progressão do evento desde as eliminatórias até a final. O quarto agrupamento, Dados de Cronometragem, centra-se na entidade REGISTRO\_TEMPO, que armazena todas as medições realizadas. A decisão de criar uma entidade independente para os registros de tempo, ao invés de vinculá-los diretamente à entidade CORRIDA, fundamenta-se em três necessidades principais: permitir múltiplos registros por equipe (para casos de tentativas ou validações), suportar o sistema híbrido de registro (manual com possibi-

lidade de correção), e facilitar a auditoria através do rastreamento de quem registrou, quando registrou, e se o tempo foi validado. Cada REGISTRO\_TEMPO relaciona-se com a EQUIPE que realizou a descida, com a CORRIDA específica, e com o USUÁRIO responsável pelo registro, criando trilha completa de auditoria. O quinto agrupamento, Controle de Acesso, representa os USUÁRIOS do sistema (organizadores, cronometristas e demais operadores), vinculando-os aos registros de tempo que validaram ou corrigiram, garantindo rastreabilidade das operações críticas. As cardinalidades estabelecidas refletem as regras de negócio do evento: um EVENTO possui múltiplas EDIÇÕES (1:N), cada EDIÇÃO envolve múltiplas EQUIPES (1:N) e organiza-se em múltiplas BATERIAS (1:N), cada BATERIA contém múltiplas CORRIDAS (1:N), e cada CORRIDA gera múltiplos REGISTROS\_TEMPO (1:N). O relacionamento N:N entre EQUIPE e REGISTRO\_TEMPO permite que uma equipe participe de múltiplas corridas ao longo do evento.

#### 4.2.3 Arquitetura Proposta do Sistema

A arquitetura do sistema foi concebida seguindo o padrão de camadas, amplamente reconhecido na engenharia de software por promover separação de responsabilidades, facilitar manutenção e permitir evolução independente de componentes (PRESSMAN; MAXIM, 2016). A solução proposta organiza-se em quatro camadas principais, conforme será detalhado na Figura Z na seção de Prototipação. A Camada de Apresentação (Frontend) será desenvolvida como aplicação web responsiva, garantindo acesso via navegadores desktop e dispositivos móveis. A interface deve adaptar-se aos diferentes perfis de usuário, apresentando dashboards simplificados para cronometristas (foco em agilidade), painéis administrativos completos para organizadores, e visualizações públicas otimizadas para acompanhamento dos resultados. A escolha por aplicação web, ao invés de aplicativos nativos, justifica-se pela necessidade de manutenção centralizada e pela dispensa de instalação por parte dos usuários, reduzindo barreiras de adoção. A Camada de Lógica de Negócio (Backend) concentrará todas as regras de negócio do evento, incluindo validação de inscrições, cálculo de classificações, geração de chaveamentos, e controle de permissões de usuários. Esta camada apresentará APIs RESTful para comunicação com o frontend, seguindo princípios de arquitetura orientada a serviços que facilitam escalabilidade e integração futura com outros sistemas. A separação clara entre frontend e backend permite que diferentes interfaces possam consumir os mesmos serviços, possibilitando futuramente, por exemplo, a criação de aplicativo mobile nativo sem alterações na lógica de negócios. A Camada de Persistência (Banco de Dados) será responsável pelo armazenamento confiável e eficiente de todos os dados do sistema. A estratégia de evolução gradual prevê inicialmente a utilização de SQLite durante as fases de desenvolvimento e testes, migrando posteriormente para PostgreSQL quando necessidades de concorrência e volume de dados justificarem banco de dados cliente-servidor robusto. Esta abordagem permite validar o modelo de dados com menor complexidade operacional, postergando investimentos em infraestrutura até o momento apropriado. A Camada de Comunicação implementará mecanismos de atualização em tempo real para o módulo de visualização de resultados. A utilização de WebSockets ou Server-Sent Events permitirá que alterações nos tempos e classificações sejam automaticamente propagadas para todos os clientes conectados, eliminando a necessidade de

recarregamento manual das páginas e proporcionando experiência verdadeiramente dinâmica aos espectadores. Uma consideração arquitetural fundamental refere-se à resiliência do sistema frente às limitações de infraestrutura identificadas na fase de Imersão. A arquitetura contempla estratégias de funcionamento offline-first no front-end, permitindo que funcionalidades críticas como o registro de tempos continuem operacionais mesmo em caso de perda temporária de conectividade. Dados registrados offline serão armazenados localmente e sincronizados automaticamente quando a conexão for restabelecida, garantindo continuidade operacional do evento.

#### 4.2.4 Escolha de Tecnologias

A seleção do conjunto tecnológico que materializará a solução proposta pautou-se por critérios de maturidade, suporte da comunidade, aderência aos requisitos identificados e capacitação da equipe de desenvolvimento. As escolhas foram realizadas através de análise comparativa de alternativas, considerando vantagens e limitações de cada opção. Para a camada de lógica de negócio, optou-se pela utilização do framework Spring Boot sobre a plataforma Java. Esta decisão fundamenta-se em diversos fatores convergentes. Primeiramente, Spring Boot oferece ecossistema maduro e abrangente para desenvolvimento de APIs RESTful, incluindo componentes prontos para autenticação (Spring Security), acesso a dados (Spring Data JPA), e exposição de serviços web (Spring Web). A curva de aprendizado da equipe, composta por estudantes familiarizados com Java através das disciplinas do curso, reduz riscos de projeto relacionados à capacitação técnica. Adicionalmente, Spring Boot promove convenção sobre configuração, reduzindo significativamente a quantidade de código boilerplate necessário e acelerando o desenvolvimento. A robustez e escalabilidade da plataforma Java garantem que o sistema poderá crescer conforme demandas futuras sem necessidade de reescrita completa. Por fim, a vasta documentação disponível e a comunidade ativa facilitam a resolução de problemas e a adoção de melhores práticas. Para a camada de apresentação, selecionou-se a biblioteca React como base para construção das interfaces. React destaca-se pela abordagem declarativa de construção de interfaces, onde componentes reutilizáveis encapsulam tanto lógica quanto apresentação, promovendo modularidade e manutenibilidade do código. A arquitetura baseada em componentes alinha-se perfeitamente com a necessidade de diferentes painéis para diferentes perfis de usuário, permitindo compartilhamento de elementos comuns enquanto especializa-se funcionalidades específicas. O ecossistema React oferece vasto conjunto de bibliotecas complementares para necessidades comuns: gerenciamento de estado (Redux ou Context API), rotas (React Router), comunicação em tempo real (Socket.io-client), e componentes de interface pré-construídos (Material-UI, Ant Design). A curva de aprendizado, embora presente, é suavizada pela abundância de recursos educacionais e pela crescente adoção da biblioteca no mercado. A estratégia de banco de dados prevê evolução em duas etapas. Inicialmente, SQLite será utilizado durante desenvolvimento e testes, aproveitando sua simplicidade de configuração (banco de dados em arquivo único) e adequação para cenários de concorrência limitada. Esta escolha permite à equipe focar no desenvolvimento da lógica de negócio e na validação do modelo de dados sem investir tempo em configuração e administração de servidor de banco de dados. Conforme o sistema amadurecer e aproximar-se de uso em produção, migração para PostgreSQL será realizada.

PostgreSQL oferece recursos fundamentais para aplicações multi-usuário em produção: suporte robusto a transações ACID, controle de concorrência otimista, replicação para alta disponibilidade, e performance superior para operações de leitura/escrita simultâneas. A transição de SQLite para PostgreSQL será facilitada pela utilização de ORM (Object-Relational Mapping) através do Spring Data JPA, que abstrai detalhes específicos do banco de dados e permite mudança de provider com mínimas alterações no código. A ausência de rede WiFi estável na área da pista, identificada como limitação crítica na fase de Imersão, demandou pesquisa intensa visando uma solução arquitetural que garantisse comunicação em tempo real entre os dispositivos dos cronometristas, organizadores e público, sem depender de infraestrutura de internet externa. A solução adotada utiliza dois microcontroladores ESP32 configurados como Access Point (ponto de acesso WiFi) e servidores, criando uma rede autônoma à qual dispositivos móveis se conectam diretamente. O ESP32, além de suas capacidades de comunicação sem fio, possui recursos computacionais suficientes para hospedar servidor web leve e broker de mensagens, tornando-se o núcleo da infraestrutura de rede do evento. Dispositivos móveis (smartphones e tablets) dos usuários conectam-se à rede WiFi criada pelo ESP32, acessando a aplicação web servida localmente e recebendo atualizações em tempo real através de protocolo de mensageria. Para implementação da funcionalidade de ranking ao vivo e sincronização de dados entre múltiplos clientes conectados, avaliou-se três abordagens principais: polling periódico, WebSockets (FETTE; MELNIKOV, 2011), e protocolos de mensageria baseados em publish-subscribe como MQTT (BANKS; GUPTA, 2014) ou AMQP (Advanced Message Queuing Protocol, implementado pelo RabbitMQ) (OASIS, 2012). A técnica de polling, embora simples, gera tráfego desnecessário e introduz latência significativa na atualização, sendo descartada como opção viável. WebSockets oferecem comunicação bidirecional full-duplex eficiente (FETTE; MELNIKOV, 2011), porém demandam manutenção de conexões persistentes entre servidor e cada cliente, o que pode sobrecarregar o ESP32 quando múltiplos usuários estão conectados simultaneamente. Protocolos baseados em publish-subscribe, particularmente MQTT (Message Queuing Telemetry Transport), apresentam-se como solução mais adequada ao contexto do projeto. MQTT foi projetado especificamente para comunicação em dispositivos com recursos limitados e redes de baixa largura de banda (BANKS; GUPTA, 2014), características alinhadas com o cenário do ESP32 operando como servidor. O protocolo implementa padrão de mensageria onde publishers (publicadores) enviam mensagens para tópicos específicos, e subscribers (assinantes) recebem automaticamente todas as mensagens dos tópicos aos quais estão inscritos (HOHPE; WOOLF, 2003), sem necessidade de conexões diretas ponto-a-ponto. Esta arquitetura reduz significativamente a carga sobre o servidor em comparação com WebSockets, pois o broker MQTT gerencia eficientemente a distribuição de mensagens. Alternativamente, RabbitMQ implementa protocolo AMQP, oferecendo recursos mais avançados de roteamento de mensagens, garantias de entrega e persistência. Embora mais robusto, RabbitMQ demanda maior consumo de recursos computacionais e memória, potencialmente excedendo as capacidades do ESP32 quando operando simultaneamente como Access Point, servidor web e broker de mensagens. A decisão final entre MQTT e implementação simplificada baseada em AMQP será definida durante a fase de Prototipação, baseada em testes de carga que avaliarão o comportamento do ESP32 sob condições realistas de uso (estimados 50-100 dispositivos conectados simultaneamente durante o evento). O backend da aplicação, desenvolvido em Spring Boot, integrará cliente

MQTT através da biblioteca Eclipse Paho ou Spring Integration MQTT, estabelecendo ponte entre as operações de banco de dados (persistência de registros de tempo) e a distribuição de mensagens para os clientes conectados. Quando o cronometrista registra tempo através da interface web, o backend persiste dados no banco local, valida o registro, e publica mensagem MQTT notificando todos os clientes assinantes, que atualizam suas interfaces imediatamente sem necessidade de recarregamento manual.

#### **4.2.5 Considerações sobre Tecnologias Não Adotadas**

A decisão de utilizar microcontrolador ESP32 como infraestrutura de rede fundamenta-se em análise pragmática que equilibra inovação tecnológica, viabilidade de implementação e alinhamento com os objetivos centrais do projeto. É fundamental distinguir entre dois usos potenciais de IoT (MINERVA; BIRU; ROTONDI, 2015) no contexto deste trabalho: (1) sensores automáticos de cronometragem, e (2) infraestrutura de comunicação e rede. Enquanto o primeiro foi deliberadamente descartado, o segundo constitui elemento essencial da arquitetura proposta. O ESP32, configurado como Access Point, cria rede WiFi autônoma com alcance suficiente para cobrir área do evento (estimados 100-150 metros lineares da pista). Sua capacidade de processar requisições HTTP, servir conteúdo estático (arquivos HTML/CSS/JavaScript da aplicação frontend), e operar broker MQTT simultaneamente, embora exija otimizações cuidadosas, está dentro das especificações técnicas do hardware. Arquitetura híbrida também foi considerada: múltiplos ESP32 distribuídos geograficamente (largada, meio da pista, chegada), cada um criando Access Point independente mas sincronizados através de rede mesh (AKYILDIZ et al., 2005) ou backbone cabeado. Esta abordagem aumenta a complexidade mas oferece redundância e melhor cobertura. A decisão entre arquitetura centralizada (um único ESP32) ou distribuída (múltiplos ESP32) será baseada em testes de campo que irão mensurar a qualidade do sinal nas diferentes posições da pista. Importante ressaltar que esta decisão não descarta evolução futura para cronometragem automática. Esta flexibilidade arquitetural permite que o projeto inicie com escopo gerenciável, valide conceitos em produção real, e evolua incrementalmente (PRESSMAN; MAXIM, 2016) conforme recursos e maturidade técnica da equipe permitirem.

### **4.3 Prototipação**

Xxxxxxxxxxxxxxxxxxxxxx xxxxxxxxx xxxxxxxx xxxxxxxxxxxx xxxxxxxxx. X xxxxxxxxx  
 xxxxxxxxxxxx xxxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx x xxxxxxxxxxxx. XXXXXXXXXXXXXXXXXXXX  
 xxxxxxxxx xxxxxxxx xxxxxxxxxxxx xxxxxxxx. X xxxxxxxxxxxx

## **5 RESULTADOS ESPERADOS**

Apresentação dos resultados.

## **6 CONCLUSÃO**

Texto da conclusão.

## REFERÊNCIAS BIBLIOGRÁFICAS

aaaa() aaaa.

Brown(2008) Tim Brown. Design thinking. *Harvard Business Review*, 86(6):84–92, 2008.

Gil(2017) Antonio Carlos Gil. *Como elaborar projetos de pesquisa*. Atlas, São Paulo, 6 edition, 2017.

Interaction Design Foundation(2025) Interaction Design Foundation. What is design thinking?, 2025. URL <<https://www.interaction-design.org/literature/topics/design-thinking>>.

Lakatos and Marconi(2017) Eva Maria Lakatos and Marina de Andrade Marconi. *Fundamentos de metodologia científica*. Atlas, São Paulo, 8 edition, 2017.

Prodanov and Freitas(2013) Cleber Cristiano Prodanov and Ernani Cesar de Freitas. *Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico*. Feevale, Novo Hamburgo, 2 edition, 2013.

Schwaber and Sutherland(2020) Ken Schwaber and Jeff Sutherland. *O Guia do Scrum: o guia definitivo para o Scrum: as regras do jogo*, Novembro 2020. Disponível em: <<https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Portuguese-European.pdf>>. Acesso em: 02 out. 2025.

Severino(2017) Antonio Joaquim Severino. *Metodologia do trabalho científico*. Cortez, São Paulo, 24 edition, 2017.

Souza et al.(2020) Souza, Cavassini, and Sabino Ana Paula de Melo Souza, Gabrielly Alves Cavassini, and Marcio Rodrigues Sabino. Design thinking e scrum no desenvolvimento de software para gerência de petições. *REFAS: Revista FATEC Zona Sul*, 7(1):112, 2020.

Vianna et al.(2012) Vianna, Vianna, Adler, Lucena, and Russo Maurício Vianna, Ysmar Vianna, Isabel K Adler, Brenda F Lucena, and Beatriz Russo. *Design Thinking: inovação em negócios*. MJV Press, Rio de Janeiro, 2012.

## **Apêndice A**

### **PRIMEIRO APÊNDICE**

Texto do apêndice.

## **ANEXO A – TÍTULO DO ANEXO**

Texto do anexo.