INSTITUTO FEDERAL DO PARANÁ
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

RAFAEL FRANCISCO CECCHIN

# DESENVOLVIMENTO DE UM SOFTWARE OPEN-SOURCE PARA GERENCIAMENTO DE PROVAS DE DOWNHILL COM SUPORTE À INTEGRAÇÃO COM EMBARCADOS VIA API SERIAL

UNIÃO DA VITÓRIA

2023

RAFAEL FRANCISCO CECCHIN

# DESENVOLVIMENTO DE UM SOFTWARE OPEN-SOURCE PARA GERENCIAMENTO DE PROVAS DE DOWNHILL COM SUPORTE À INTEGRAÇÃO COM EMBARCADOS VIA API SERIAL

Trabalho de conclusão de curso apresentado ao curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal do Paraná, como requisito parcial de avaliação.

Orientador: Deividson Luiz Okopnik, Me

UNIÃO DA VITÓRIA

2023

Rafael Francisco Cecchin
DESENVOLVIMENTO DE UM SOFTWARE OPEN-SOURCE PARA GERENCIAMENTO DE PROVAS DE DOWNHILL COM SUPORTE À INTEGRAÇÃO COM EMBARCADOS VIA API SERIAL/ Rafael Francisco Cecchin. – União da Vitória, 2023-
64 p. : il. (algumas color.) ; 30 cm.

Orientador: Deividson Luiz Okopnik, Me

Trabalho de Conclusão de Curso – Instituto Federal do Paraná, 2023.

1. Downhill. 2. RFID. 3. Software. 4. Orientador: Deividson Luiz Okopnik, Me. 5. DESENVOLVIMENTO DE UM SOFTWARE OPEN-SOURCE PARA GERENCIAMENTO DE PROVAS DE DOWNHILL COM SUPORTE À INTEGRAÇÃO COM EMBARCADOS VIA API SERIAL.

## TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal do Paraná – Campus União da Vitória, foram convocados para realizar a arguição do trabalho de conclusão de curso de Rafael Francisco Cecchin, intitulado: **DESENVOLVIMENTO DE UM *SOFTWARE OPEN-SOURCE* PARA GERENCIAMENTO DE PROVAS DE *DOWNHILL* COM SUPORTE À INTEGRAÇÃO COM EMBARCADOS VIA API SERIAL**, sob orientação do Professor Me. Deividson Luiz Okopnik, que após terem inquirido o aluno e realizada a avaliação do trabalho são de parecer pela sua aprovação no rito da defesa.
A outorga do título de Tecnólogo em Análise e Desenvolvimento de Sistemas está sujeita a homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Colegiado do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas.

UNIÃO DA VITÓRIA, 14 de dezembro de 2023.

Deividson Luiz Okopnik
Presidente da banca examinadora
Instituto Federal do Paraná

Douglas Lusa Krug
Avaliador Interno – Instituto Federal do Paraná

Bruno Ricardo Resende
Avaliador Externo – Ugv Centro Universitário

## RESUMO

Este projeto propõe o desenvolvimento de um software de código aberto para gerenciamento para provas de downhill, com o objetivo principal de simplificar e aprimorar a organização de eventos competitivos. O software visa integrar funcionalidades essenciais, como o registro digitalizado de competidores, a organização eficiente de campeonatos e etapas, a categorização de competidores e a integração com hardware embarcado para cronometragem em tempo real. Os testes realizados demonstram que o sistema é promissor e também indicam que ele pode ser utilizado em competições reais para proporcionar uma gestão mais profissional e simplificada das competições.



**Palavras-chaves**: Downhill. RFID. Software.

## ABSTRACT

This project proposes the development of an open-source software for managing downhill races, with the main goal of simplifying and enhancing the organization of competitive events. The software aims to integrate essential functionalities, such as the digitized registration of competitors, efficient organization of championships and stages, categorization of competitors, and integration with embedded hardware for real-time timing. The conducted tests demonstrate that the system is promising and also indicate that it can be used in actual competitions to provide a more professional and streamlined management of the events.



**Key-words**: Downhill. RFID. Software.

**SUMÁRIO**

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| API | Application Programming Interface |
| CLI | Comand Line Interface |
| CSS | Cascading Style Sheets |
| HTML | Hyper Text Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IDE | Integrated Development Environment |
| IEEE | Institute of Electrical and Electronic Engineers |
| IoT | Internet of Things |
| JSON | JavaScript Object Notation |
| LoRa | Long Range |
| LoRaWAN | Long Range Wide Area Network |
| MTB | Mountain Bike |
| NPM | Node Package Manager |
| NoSQL | Non Structured Query Language |
| ORM | Object-Relational Mapper |
| RFID | Radio Frequency Identification |
| ROM | Read-only Memory |
| RTC | Real-time clock |
| SCSS | Syntactically Awesome Style Sheets |
| SQL | Structured Query Language |
| SRAM | Static Random Acess Memory |
| UCI | Union Cycliste Internationale |
| UML | Unified Modeling Language |
| USB | Universal Serial Bus |
| Wi-Fi | Wireless Fidelity |
| XML | eXtensible Markup Language |

# 1 INTRODUÇÃO

Este trabalho se propõe a abordar o desenvolvimento de um software inovador dedicado ao gerenciamento de competições de downhill, com o objetivo de aprimorar a organização e otimizar a execução de eventos dessa modalidade do mountain bike.

O principal objetivo deste projeto é desenvolver um *software* que simplifique o gerenciamento de competições e forneça uma plataforma aberta para integração com sistemas embarcados, permitindo que os desenvolvedores de sistemas embarcados utilizem uma API para estender as funcionalidades do *software* principal.

A relevância deste projeto reside na sua capacidade de contribuir para a modernização do gerenciamento de provas de *downhill*, simplificando processos e integrando tecnologias. Ao finalizar este trabalho, espera-se que o *software* desenvolvido demonstre sua eficácia e que os resultados obtidos possam impactar positivamente a realização de competições dessa modalidade.

# 2 OBJETIVOS

## 2.1 OBJETIVO GERAL

O principal objetivo desse projeto é desenvolver um *software* para computador capaz de gerenciar provas de *downhill*. O *software* deve disponibilizar uma API para comunicação através da interface serial, permitindo que desenvolvedores de sistemas embarcados criem soluções que trabalhem em conjunto com o *software* e estendam suas funcionalidades.

## 2.2 OBJETIVOS ESPECÍFICOS

- Entender como as provas de *downhill* funcionam e os problemas enfrentados.
- Projetar como deve ser a infraestrutura ideal para que os embarcados consigam se comunicar via API com o *software* proposto de forma eficiente.
- Projetar e desenvolver o *software* responsável por gerenciar as provas de *downhill* e também por receber informações externas via API.
- Realizar os testes necessários para o funcionamento da aplicação.
- Disponibilizar o código-fonte da aplicação para permitir o aprimoramento da tecnologia por outros desenvolvedores.

# 3 REVISÃO LITERÁRIA

## 3.1 *DOWNHILL*

O *mountain bike*, popularmente abreviado pela sigla MTB, é uma modalidade de ciclismo praticada em montanhas e ambientes com terrenos irregulares, como terra, neve e areia (INTERNATIONALE, 2023b). O regulamento geral e técnico de corridas, estabelecido pela União Ciclística Internacional (UCI), é um documento que dispõe das regras que devem ser seguidas nos campeonatos de ciclismo e será nossa principal fonte de referência sobre o assunto.

O *mountain bike* é dividido em diversas modalidades que propõem diferentes desafios aos praticantes de ciclismo. Dentre estas modalidades, podemos citar: *cross country*, *downhill*, *four cross*, *enduro*, *pump track snow bike* e *e-mountain biking* (INTERNATIONALE, 2023b).

O *downhill* é uma das subdivisões do *mountain bike* e nesse esporte o objetivo dos competidores é fazer o percurso o mais rápido possível, mas isso requer muita técnica e habilidade, pois além da pista estar localizada na descida de um morro ou montanha, há diversas irregularidades e curvas fechadas.

> "O percurso compreende seções de terrenos variados: trilhas estreitas e largas, estradas e caminhos florestais, caminhos de campo e trilhas rochosas. Normalmente há uma mistura de secções rápidas e técnicas. A ênfase do percurso é testar as habilidades técnicas dos pilotos e sua capacidade física (tradução nossa)." (INTERNATIONALE, 2023b, p. 24).

Segundo a UCI, a distância do percurso não deve exceder 3500 metros. Em campeonatos mundiais, taças mundiais, campeonatos continentais e competições de classe 1, a duração da prova deve variar entre 2 e 5 minutos. No caso de competições de classe 2, as provas devem durar entre 1 e 5 minutos e não há restrições relacionadas à duração em competições de classe 3 (INTERNATIONALE, 2023b, p. 24–25).

A prova contém um treino cronometrado ou uma descida classificatória para definir a ordem de largada, seguida da descida oficial, em que o vencedor é o competidor mais rápido. Caso haja aprovação prévia da UCI, é possível fazer um sistema de

corrida baseado em duas mangas, em que cada ciclista pode descer duas vezes e o seu melhor tempo é apurado (INTERNATIONALE, 2023b, p. 24).

A utilização de sistemas de cronometragem é essencial para as competições de *downhill*, pois o tempo de cada ciclista define sua classificação final. Para garantir a precisão na medição do tempo, são empregados diferentes tipos de cronometragem, como a manual, por fotocélula e RFID. De acordo com a UCI, alguns eventos como corridas de rua, pista, jogos olímpicos e campeonatos mundiais também exigem o uso obrigatório de *photo-finish* com cronometragem eletrônica (INTERNATIONALE, 2023a, p. 45).

Apesar da determinação da UCI, campeonatos locais e regionais de pequeno porte ainda recorrem à cronometragem manual devido aos altos custos das demais tecnologias apresentadas. Funciona da seguinte forma: os oficiais do evento, que são as pessoas responsáveis pelos resultados oficiais, ficam na linha de largada e chegada e registram o tempo e número do competidor quando o mesmo inicia a prova e também quando chega ao final do percurso (TSAI, 2011, p. 26).

A cronometragem manual pode trazer algumas consequências negativas para o registro dos tempos dos competidores, uma vez que o número do competidor deve ficar localizado em seu peito e fixado à frente do guidão de sua bicicleta (INTERNATIONALE, 2023a, p. 89) e, caso o número esteja ilegível devido a algum incidente não intencional, como respingos de barro ou queda, isso pode afetar a integridade dos resultados da corrida (TSAI, 2011, p. 26). Além disso, a cronometragem manual pode estar sujeita a imprecisões nos registros dos tempos, uma vez que depende da agilidade e integridade das pessoas encarregadas de controlar a largada e a chegada.

Uma das tecnologias mais recentes utilizadas na cronometragem de provas de *mountain bike* é o uso de chips eletrônicos RFID (*Radio Frequency Identification*), que permitem o registro automático do tempo de cada competidor ao longo do percurso. O chip é fixado no capacete ou na bicicleta do ciclista e é capaz de registrar o tempo de passagem em determinados pontos do circuito, sem a necessidade de intervenção humana, o que garante maior precisão nos registros de tempo e reduz as brechas para erros na cronometragem.

## 3.2 *HARDWARE*

Segundo Tanenbaum, *hardwares* são componentes eletrônicos que compõem um computador, incluindo resistores, capacitores, reguladores de tensão, ROM, memória e dispositivos de entrada e saída (TANENBAUM; ZUCCHI, 2013, p. 6). Entre os *hardwares* existentes, encontram-se os *hardwares* de sistemas embarcados, que são projetados para um único propósito, diferentemente dos *hardwares* de computadores convencionais que fornecem um sistema computacional geral para atender às diversas demandas do usuário (CHASE; ALMEIDA, 2007)[2].

Para facilitar a prototipagem de projetos eletrônicos foram criadas placas de desenvolvimento, que contêm um sistema embarcado em que é possível transferir um novo *software* de forma simples e rápida, de acordo com as necessidades do projeto. Um exemplo é a placa Arduino Uno (FIGURA 1), que possui um microcontrolador ATmega328P, um *clock* de 16 MHz, 14 pinos digitais, 6 pinos analógicos, uma memória *flash* de 32 KB e uma SRAM de 2 KB. De acordo com Massimo Banzi, cofundador da plataforma, "O Arduino é uma ferramenta fácil de usar para criar objetos interativos e sistemas embarcados" (BANZI; SHILOH, 2014, p. 11).

FIGURA 1 – PLACA ARDUINO UNO


FONTE: o Autor (2023)

### 3.2.1 Conexão serial

Conexão Serial é um termo que se refere a qualquer tipo de comunicação que ocorre em série, ou seja, onde os dados são transmitidos sequencialmente, um *bit* de cada vez (MENEZES; SATO, 2019)[13]. Quando é estabelecida uma conexão serial através do protocolo USB (*Universal Serial Bus*), na verdade trata-se de uma conexão serial virtual emulada pelo *hardware*, já que a conexão não ocorre por meio de uma porta convencional.

O Arduino conta com uma porta USB tipo B, que é usada para estabelecer uma conexão serial virtual com o computador. Segundo Banzi e Shiloh (2014, p. 72), este canal de comunicação entre o Arduino e o computador abre muitas possibilidade, já que existem muitas linguagens de programação que permitem escrever programas que podem se comunicar com a porta serial e, através da porta serial, esses programas podem se conectar com o Arduino.

### 3.2.2 ESP-NOW

O ESP-NOW é um protocolo de comunicação desenvolvido pela *Espressif Systems*, uma empresa chinesa de tecnologia conhecida por seus chips de micro-controladores e módulos de comunicação sem fio, como o ESP8266 e ESP32. Ele permite a comunicação direta entre esses dispositivos em uma distância de até 190 metros em campo aberto, sem a necessidade de um roteador Wi-Fi intermediário, tornando-o adequado para aplicações de IoT (Internet das Coisas) de baixa potência e alta eficiência energética (PASIC; KUZMANOV; ATANASOVSKI, 2021).

Também segundo Pasic, Kuzmanov e Atanasovski (2021), o ESP-NOW utiliza protocolos para para garantir segurança na comunicação. Além disso, ele oferece suporte a recursos como comunicação *unicast* criptografada e não criptografada, dispositivos pares mistos criptografados e não criptografados, capacidade de transportar *payloads* de até 250 *bytes* e uma função de retorno de chamada de envio para informar a camada de aplicação sobre o sucesso ou falha na transmissão.

### 3.2.3 LORA

Segundo Augustin et al. (2016), em dispositivos IoT (Internet das Coisas) espera-se um consumo menor de energia, já que as "coisas" são alimentadas por baterias e seu consumo deve ser minimizado. Para suprir essa demanda, surgiram diversas tecnologias e, dentre elas, destaca-se a tecnologia LoRa e seu protocolo LoRaWAN (*Long Range Wide Area Network*).

Devido a sua capacidade de longo alcance e baixo consumo de energia, a tecnologia LoRa pode ser aplicada em diversas áreas, como monitoramento da saúde (sensores de saúde e equipamentos médicos), medição inteligente (contadores de eletricidade, água e gás), monitoramento ambiental (monitorar a qualidade do ar, níveis de poluição, condições climáticas e qualidade da água) e em aplicações industriais. (AUGUSTIN et al., 2016, p. 1).

### 3.2.4 RFID

A tecnologia RFID (*Radio Frequency Identification*) é utilizada para identificação por radiofrequência em vários setores, permitindo rastreamento de bens, controle de tráfego, monitoramento de animais, gerenciamento de acervos em bibliotecas, cronometragem esportiva, entre outros processos (NOGUEIRA FILHO, 2005, p. 37).

Os quatro componentes essenciais do RFID - *tag*, *transceiver*, antena e *middleware* - constituem a base dessa tecnologia. Quando a antena integra-se ao *transceiver*, ela se torna o leitor. O funcionamento básico do RFID envolve o leitor que, por meio de ondas magnéticas, energiza a *tag*, que responde transmitindo informações. O leitor, então, encaminha esses dados para o *middleware*, que define a finalidade das informações (TEIXEIRA, T., 2011, p. 14).

No contexto de um mundo cada vez mais conectado, o RFID destaca-se como ferramenta fundamental para aprimorar a gestão e eficiência de processos devido a sua capacidade de coletar dados precisos e em tempo real.

## 3.3 DIAGRAMA DE CASO DE USO

Os diagramas UML constituem ferramentas indispensáveis no âmbito da engenharia de *software*, pois ajudam na modelagem e documentação, tornando mais fácil

entender a estrutura e o comportamento do software.

No diagrama de caso de uso, parte integrante da UML, a ênfase recai sobre a engenharia de requisitos. Seguindo a abordagem de Barros (2009), a elaboração cuidadosa desses diagramas, que capturam as interações entre atores e casos de uso, contribui para uma documentação clara e organizada, facilitando a comunicação e compreensão dos requisitos.

Conforme a metodologia proposta por Barros (2009), o diagrama de casos de uso detalhada de itens como nome, cenários principais e adicionais, atores, partes interessadas, precondições e dependências. Essa abordagem não apenas cria uma representação visual clara, mas também estabelece uma base sólida para a compreensão e desenvolvimento do sistema, reforçando a importância da diagramação adequada no processo de elaboração de um *software*.

## 3.4 API

Para que um *software* permita a comunicação com aplicações criadas por outros programadores, é necessário disponibilizar uma API (*Application Programming Interface*), que é um conjunto de rotinas e protocolos que permite a comunicação entre diferentes aplicações. As APIs são importantes, pois ajudam a melhorar os aspectos de eficiência, automação e personalização dos serviços disponibilizados (JAYAKODY et al., s.d., p. 42).

As APIs comunicam dados em formatos específicos, como JSON (*JavaScript Object Notation*) e XML (*eXtensible Markup Language*). Ambos são os formatos mais comuns e permitem a troca de informações estruturadas entre diferentes sistemas de forma eficiente (JAYAKODY et al., s.d., p. 42).

## 3.5 BANCO DE DADOS

O armazenamento de informações é importante para *softwares* em geral e, para que isso seja feito de forma eficiente, é necessário utilizar um banco de dados. De acordo com Sumathi e Esakkirajan (2007), o uso de banco de dados permite controlar a integridade, o acesso e as transações realizadas sobre os dados.

Existem os bancos de dados relacionais, que organizam dados em tabelas

inter-relacionadas usando SQL (*Structured Query Language*), e os bancos de dados não relacionais, também conhecidos como NoSQL (*Non Structured Query Language*), que oferecem flexibilidade com modelos como documentos e grafos para dados não estruturados (SILVA DIAS, 2023).

A complexidade e a estrutura dos dados influenciam na escolha do banco de dados a ser utilizado. Se a aplicação lida com dados estruturados e inter-relacionados de forma rígida, um banco de dados relacional oferece consistência e integridade. No entanto, se a estrutura dos dados é dinâmica ou não estruturada, os bancos de dados NoSQL proporcionam flexibilidade e escalabilidade horizontal, permitindo acomodar grandes volumes de dados variáveis (SILVA DIAS, 2023).

### 3.5.1 Modelagem de dados

Segundo Heuser (1998), um modelo de dados é uma descrição dos tipos de informações armazenadas em um banco de dados. Essa descrição é feita através de uma linguagem de modelagem, que pode ser textual ou gráfica, e pode ser definida em diferentes níveis de abstração.

É possível encontrar diferentes programas com o objetivo de facilitar a diagramação de bancos de dados, dentre eles destaca-se o BR Modelo *Web*, que é um *software* de modelagem *open-source* e fácil de usar. Uma das características do sistema que torna ele mais atrativo é a disponibilidade na internet e armazenamento em nuvem, que dispensa a instalação do programa no computador e facilita seu acesso em diferentes lugares através do navegador de internet.

O BR Modelo *Web* permite a modelagem de dados através da abordagem entidade-relacionamento que, segundo Heuser (1998), é a técnica de modelagem de dados mais difundida e utilizada. Normalmente, em um projeto de banco de dados, os principais níveis de abstração considerados são conceitual, em que o sistema gerenciador de banco de dados utilizado não é considerado, e lógico, que deve considerar as especificidades do banco de dados escolhido em sua construção (HEUSER, 1998), e o BR Modelo *Web* fornece opções para esses dois tipos de abstração.

## 3.6 IDE

As IDEs (*Integrated Development Environment*) são ferramentas importantes para o desenvolvimento de *software*, pois elas fornecem um ambiente integrado para edição de código, depuração, gerenciamento de projetos, entre outras funcionalidades (AMAZON WEB SERVICES, 2023).

O *Stack Overflow (2023)*, plataforma popular utilizada pelos desenvolvedores para tirarem dúvidas uns com os outros, realizou uma pesquisa perguntando qual IDE os usuários do site utilizavam e teve como resultado mais de 28% dos votos para IDE *Microsoft Visual Studio* e mais de 73% dos votos para o *Visual Studio Code*. Outras IDEs também destacaram-se na pesquisa, como IntelliJ IDEA (26.82% dos votos), Notepad++ (24.54% dos votos) e Vim (22.29% dos votos).

Diferente do *Microsoft Visual Studio* que precisa de licença, o *Visual Studio Code* é um projeto *open-source*, multiplataforma e que permite a adição de *plugins* caso o desenvolvedor queira estender suas funcionalidades, isso explica sua adesão por grande parte dos desenvolvedores.

## 3.7 C++

O C++ é uma linguagem de programação criada no início da década de 1980 por Bjarne Stroustrup, que tinha o objetivo de estender a linguagem C, adicionando recursos de orientação a objetos (STROUSTRUP, 1999).

Segundo Stroustrup (1999), criador do C++ e autor do artigo referenciado, a linguagem torna a programação mais agradável para programadores sérios e conta com suporte a abstração de dados, programação orientada a objetos e programação genérica.

A linguagem de programação "Arduino", assim como diversas outras linguagens de programação, herdou sua sintaxe da definida por C++ e é utilizada para desenvolvimento de *software* para placas de prototipagem eletrônica da família Arduino, como a apresentada na FIGURA 1 (ARDUINO, 2023).

## 3.8 HTML, CSS/SCSS E JAVASCRIPT

O HTML, segundo Flatschart (2011), é a principal linguagem utilizada na WEB. Ela é considerada uma linguagem de marcação e fornece a estrutura básica para

conteúdo na *internet*, permitindo que os desenvolvedores organizem informações em diferentes seções, como títulos, parágrafos e listas.

Em conjunto com HTML, o CSS permite que os desenvolvedores controlem o *design* e a aparência das páginas WEB. Como mencionado por Flatschart (2011), o CSS permite que conteúdo e estilo sejam trabalhados de forma independente, conferindo flexibilidade e modularidade ao fluxo de trabalho para WEB. O SCSS é uma tecnologia que gera um CSS após um pré-processamento do código de estilo e tem diversas vantagens, como por exemplo uma melhor estruturação e organização do código (SASS, 2023).

O JavaScript é uma linguagem de programação de alto nível que oferece interatividade e dinamismo às páginas WEB. Com JavaScript, os desenvolvedores podem criar animações, validar formulários e fornecer experiências de usuário altamente interativas. Segundo Flatschart (2011), o JavaScript é uma linguagem de programação com uma sintaxe relativamente simples e mesmo quem não é programador é capaz inserir fragmentos de código JavaScript no HTML.

O HTML, CSS e JavaScript trabalham juntos para fornecer uma experiência visual, interativa e dinâmica aos usuários de aplicações WEB. A união dessas tecnologias permite que os desenvolvedores criem *websites* e aplicativos que não apenas apresentam informações de forma visualmente atraente, mas que também envolvem os usuários por meio de interações intuitivas e claras.

## 3.9 SEQUELIZE

*Frameworks* fornecem uma estrutura base para a organização do código, definição de padrões e suporte a funcionalidades comuns. Sequelize é conhecido por ser um *framework* ORM (*Object-Relational Mapper*) que é uma técnica que permite mapear objetos e fazer relação direta com o banco de dados e, através dela, é possível economizar muito tempo de desenvolvimento, pois evita que o desenvolvedor tenha que criar as requisições com o banco de dados manualmente, utilizando comandos SQL (SEQUELIZE, 2023).

As *migrations* no Sequelize são uma maneira organizada de gerenciar alterações no banco de dados. Elas permitem que os desenvolvedores controlem e versionem

as mudanças no banco, tornando o processo de desenvolvimento e implantações mais consistente e fácil de acompanhar. Ao fornecer uma abordagem estruturada para modificar tabelas e colunas, as *migrations* mantém integridade do banco de dados em diferentes ambientes, garantindo uma evolução controlada do aplicativo (SEQUELIZE, 2023).

Segundo a documentação do Sequelize, é possível utilizá-lo com diferentes bancos de dados relacionais, como Postgres, MySQL, MariaDB, SQLite, DB2, Microsoft SQL Server, Snowflake, Oracle DB e Db2 para IBM i (SEQUELIZE, 2023).

## 3.10 NODE

O Node.js, uma plataforma de código aberto baseada no motor V8 do Google Chrome, surgiu em 2009, desenvolvido por Ryan Dahl. Inicialmente, o JavaScript era utilizado somente para desenvolvimento do lado do cliente, mas um grande diferencial foi sua introdução no lado do servidor com o Node.js, que permitiu aos desenvolvedores utilizar a mesma linguagem de programação em ambientes *front-end* e *back-end*, simplificando o desenvolvimento de aplicativos WEB (TEIXEIRA, P., 2011).

De acordo com Pedro Teixeira (2011, p. 1), o sucesso do Node.js não se deve apenas à sua capacidade de usar a mesma linguagem de programação em diferentes ambientes, já que outras tecnologias com uma abordagem semelhante não alcançaram tanta popularidade. O autor aponta três razões fundamentais para explicar a popularidade do Node:

- Primeiramente, o Node.js é conhecido por sua simplicidade. Ele simplifica a programação de E/S orientada a eventos, tornando-a mais acessível e compreensível do que nunca.
- Além disso, o Node é altamente eficiente. Ao se concentrar em fornecer uma base sólida e suporte aos protocolos básicos da Internet por meio de APIs limpas e funcionais, o Node.js não tenta resolver todos os problemas, mantendo-se leve e ágil.
- Outro aspecto crucial é que o Node.js não compromete sua visão inovadora. Ao não se prender a compatibilidades com *softwares* pré-existentes, o Node

adota uma abordagem original, o que muitos consideram a direção certa para o desenvolvimento.

O NPM (*Node Package Manager*) é o gerenciador de pacotes padrão do Node.js, que utiliza um arquivo chamado "package.json" para indicar dependências utilizadas e uma pasta "node_modules" que armazena as dependências instaladas (TEIXEIRA, P., 2011, p. 18). O NPM permite instalar bibliotecas que auxiliam no desenvolvimento de novas funcionalidades para aplicações e possui outras funcionalidades como executar *scripts* e manter versões de pacotes compatíveis com o projeto.

## 3.11 VERSIONAMENTO

O versionamento é utilizado na programação para registro das alterações de códigos fontes de *softwares*, mas pode ser utilizado para controle de versão de qualquer arquivo que esteja no computador (CHACON; STRAUB, 2014, p. 27). Aplicações de controle de versão registram alterações em um arquivo ou conjunto de arquivos ao longo do tempo e permitem consultar modificações realizadas e recuperar versões anteriores.

O Git foi desenvolvido por Linus Torvalds em 2005, com o objetivo de controlar o versionamento do kernel do Linux. O sistema foi criado com foco na velocidade, *design* simplificado, suporte ao desenvolvimento não linear (múltiplas ramificações em um projeto), total distribuição e capacidade de lidar com projetos de grande escala (CHACON; STRAUB, 2014, p. 31).

Já o Github é um host de repositórios que permite que usuários cadastrado na plataforma contribuam em projetos privados e/ou de código aberto de qualquer lugar do mundo. Segundo Chacon e Straub (2014), o git é uma ferramenta poderosa para controle de versão e o GitHub é o ponto central de colaboração para milhões de desenvolvedores e projetos.

# 4 DESENVOLVIMENTO

## 4.1 LEVANTAMENTO DE REQUISITOS

Além de realizar pesquisas sobre como funciona o gerenciamento e organização de competições de *downhill*, também foram feitas algumas reuniões com praticantes do esporte para definir as especificações técnicas do *software* e elaborar uma proposta detalhada para a sua implementação. Com base nas informações obtidas, segue uma lista com as principais demandas que o *software* deve ser capaz de suprir.

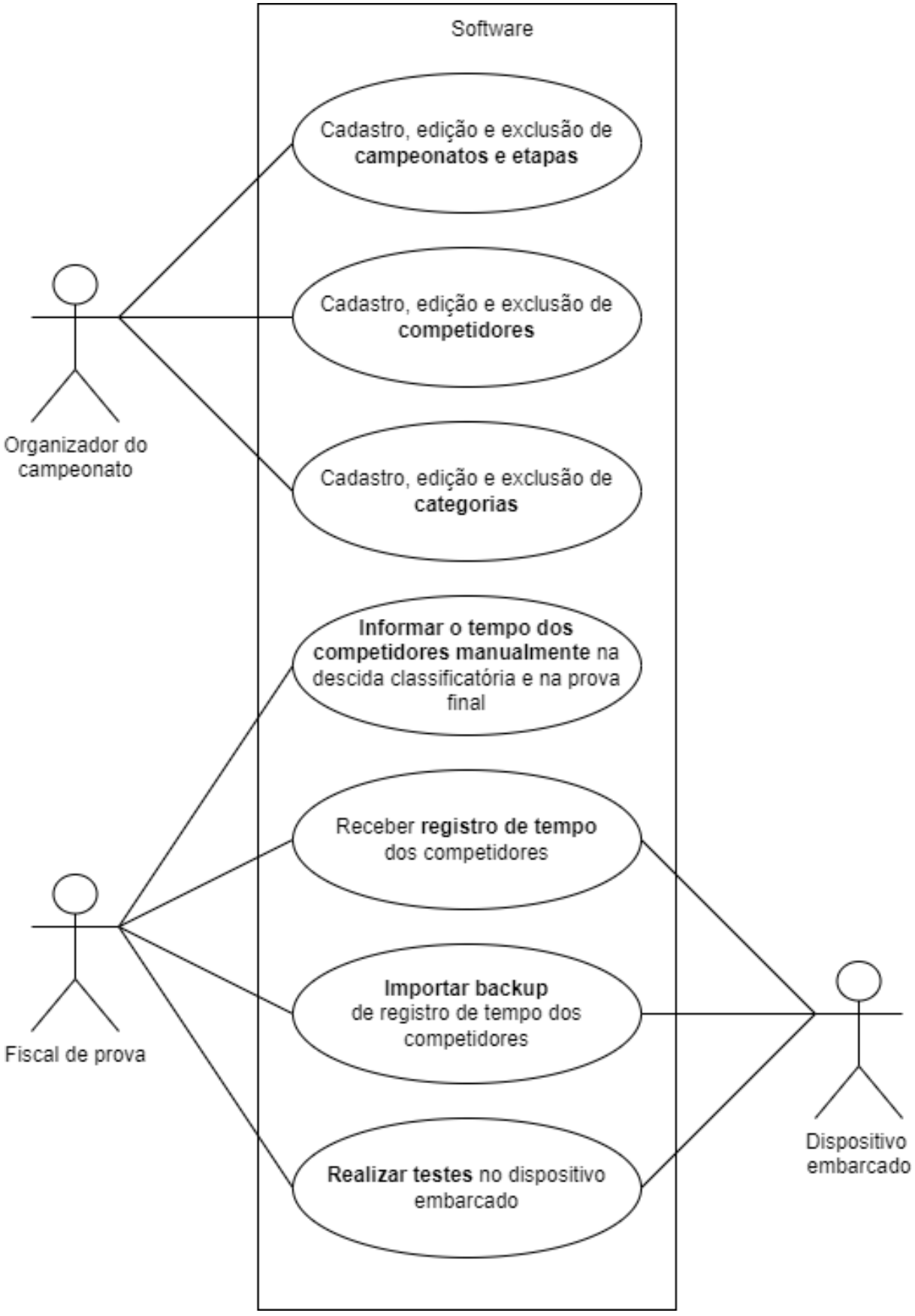
- Cadastro de campeonatos;
- Cadastro de etapas para cada campeonato em que seja possível informar o número e a data da realização das provas. Cada etapa deve permitir o registro de tempo dos competidores na descida classificatória e da prova final;
- Cadastro de categorias para os competidores, separadas por gênero;
- Cadastro de competidores com os seguintes dados: CPF, nome, gênero, data de nascimento e patrocinador. Já dentro de cada etapa, ao adicionar um competidor, deve ser selecionada uma categoria dentre as cadastradas, número da placa de identificação e código do RFID.

## 4.2 PLANEJAMENTO

Após o levantamento de requisitos, iniciou-se o planejamento do sistema e foi desenvolvido um diagrama de caso de uso através da plataforma gratuíta Diagrams.net[1]. Segundo Barros (2009), um diagrama de caso de uso é um diagrama UML que tem o objetivo de auxiliar na identificação das funcionalidades requeridas pelo *software*. Na FIGURA 2 é possível observar o resultado.

[1] https://app.diagrams.net/

FIGURA 2 – DIAGRAMA DE CASO DE USO




FONTE: o Autor (2023)

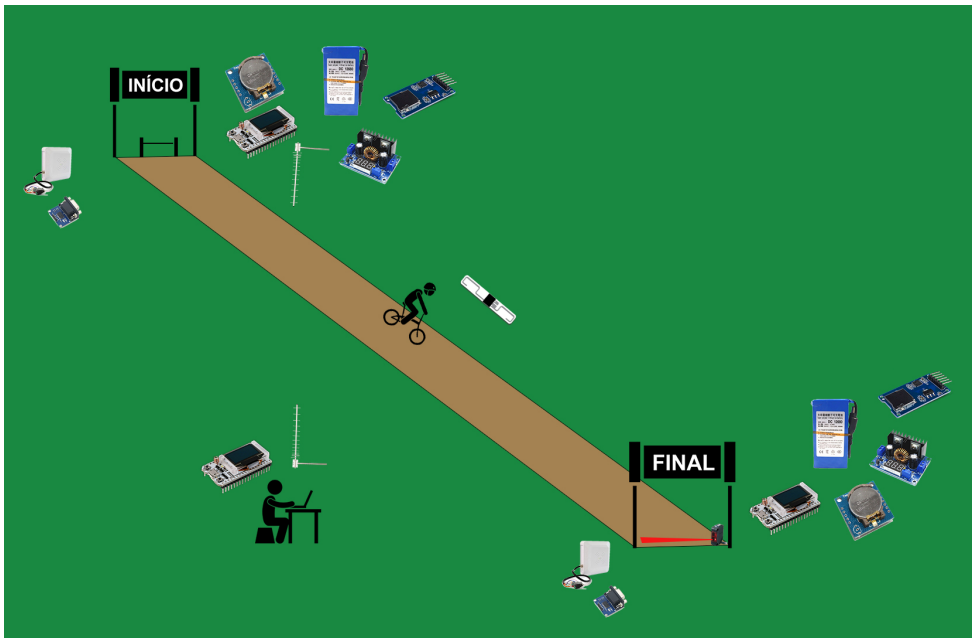
Como é possível observar no diagrama, existem três atores para o sistema e sete casos de uso. O primeiro ator, organizador do campeonato, é responsável pelos cadastros no sistema e possui 3 casos de uso relacionados a ele. O segundo ator, fiscal de prova, possui como um de seus casos de uso "informar o tempo dos competidores manualmente". Apenas com esses dois atores é possível registrar todas as informações necessárias para o gerenciamento de competições de downhill.

Existe um terceiro ator, o dispositivo embarcado, que estende as funcionalidades do sistema, possibilitando que o *software*, a comando do fiscal de prova, consiga receber o registro de tempo dos competidores durante as provas e importar *backup* com registro de tempo dos competidores. O sistema embarcado também deve permitir que o fiscal de prova consiga realizar testes para aferir se *hardware* está respondendo como esperado.

### 4.2.1 Projeção da infraestrutura

Como o *software* deve permitir a conexão com dispositivos embarcados, foi necessário projetar a infraestrutura completa do sistema para nortear as regras de comunicação entre os dispositivos. A FIGURA 3 apresenta a infraestrutura concebida para o projeto.

FIGURA 3 – INFRAESTRUTURA DO PROJETO




FONTE: o Autor (2023)

O sistema deve contar com três dispositivos embarcados, um localizado na largada, outro na chegada e um terceiro conectado ao computador por meio de conexão serial. A maneira como esses dispositivos se comunicam não é relevante para o *software* do computador, desde que as informações cheguem no formato esperado pela API. Apesar de não ser o foco principal do projeto, foram realizados estudos sobre algumas tecnologias que podem ser utilizadas em conjunto com os dispositivos embarcados para permitir uma conexão eficiente e o bom funcionamento do sistema.

Devido à grande variação do relevo e até mesmo a presença de mata fechada no *downhill*, com distâncias de até 3500 metros (INTERNATIONALE, 2023b), a tecnologia LoRa pode ser usada para permitir a conexão entre o dispositivo de largada e o dispositivo conectado ao computador. Segundo Augustin et al. (2016, p. 3), a tecnologia LoRa permite a comunicação de longa distância por meio de radiofrequência com baixo consumo de energia.

Já para permitir a conexão entre dispositivos mais próximos, como o dispositivo de chegada e o dispositivo conectado ao computador, pode-se utilizar a tecnologia ESP-NOW. O ESP-NOW é um protocolo que permite a comunicação ponto a ponto com baixo consumo de energia e opera no padrão IEEE 802.11 com frequência de 2.4GHz (PASIC; KUZMANOV; ATANASOVSKI, 2021).

A FIGURA 3 também apresenta algumas tecnologias que devem fazer parte da infraestrutura do sistema, como o sensor e a *tag* RFID, para identificação dos ciclistas; bateria e regulador de tensão, para fornecer energia na tensão correta para o embarcado e seus sensores e atuadores; módulo RTC, para fornecer informações sobre data e hora; módulo SD, para *backup* das informações da corrida; *start gate*, na largada e sensor fotoelétrico, na chegada, para informar o início e fim da cronometragem.

### 4.2.2 Prototipagem do *software*

Segundo Usability.gov (2013), um protótipo é um modelo preliminar de um produto que ajuda a testar e validar ideias antes de investir recursos significativos no desenvolvimento. Um protótipo permite que os criadores experimentem diferentes soluções de *design* e recebam *feedback* dos usuários, possibilitando ajustes e melhorias antes da produção.

Existem alguns programas que ajudam a criar protótipos de *softwares* e o programa escolhido para o projeto foi o Figma[2]. Segundo Sharma e Tiwari (2021), o Figma é um editor gráfico e vetorial gratuito, disponível na *internet* e também no aplicativo, que pode ser utilizado para prototipagem de projetos. Todos os recursos visuais desenvolvidos para o projeto em questão passaram antes pela prototipagem no Figma.

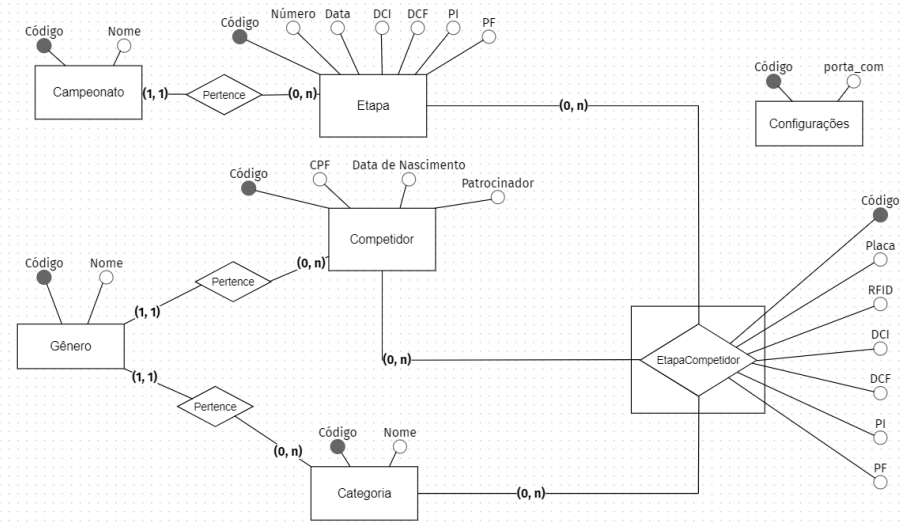
A criação do protótipo, além de auxiliar na validação do produto, foi muito importante para nortear e acelerar o desenvolvimento do *software*.

## 4.3 BANCO DE DADOS

O sistema gerenciador de banco de dados escolhido para o projeto foi o SQLite, devido à sua simplicidade e portabilidade. O SQLite é implementado como um arquivo de banco de dados único, facilitando a inclusão direta em nossa aplicação. Isso elimina a necessidade de configurações complexas e oferece uma solução eficiente para armazenar e recuperar dados, tornando o processo de desenvolvimento mais simples e acessível.

### 4.3.1 Modelagem conceitual

FIGURA 4 – MODELO CONCEITUAL DO BANCO DE DADOS



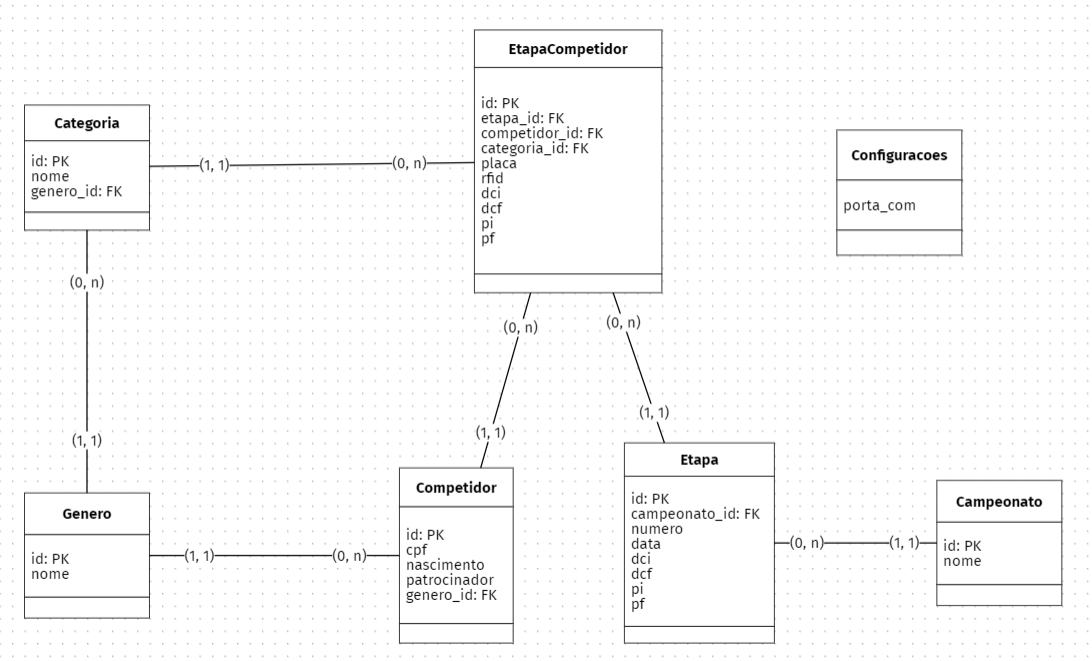

FONTE: o Autor (2023)

[2] https://www.figma.com/

No modelo conceitual apresentado na FIGURA 4, a entidade-relacionamento é representada por um retângulo com um losango em seu interior, as entidades são representadas pelos retângulos, os relacionamentos são representados por losangos, os atributos são representados pelos círculos e também há linhas que ligam os atributos às entidades e aos relacionamentos.

### 4.3.2 Modelagem lógica

FIGURA 5 – MODELO LÓGICO DO BANCO DE DADOS




FONTE: o Autor (2023)

O modelo lógico é uma representação mais próxima do que deve haver no banco de dados. Como é possível observar na FIGURA 5, as tabelas são representadas por retângulos e seus atributos, ficam em seu interior. Após os nomes de alguns atributos, há as siglas "PK" e "FK", que significam respectivamente "Chave primária" (*primary key*) e "Chave estrangeira" (*foreign key*).

## 4.4 PROGRAMAÇÃO

Para desenvolver um *software*, é essencial escolher a linguagem de programação mais adequada às necessidades do projeto, já que ela é responsável por comunicar instruções ao computador e executar tarefas específicas. Neste projeto, a linguagem

de programação utilizada para desenvolver o *firmware* do embarcado foi o Arduino, que é uma linguagem baseada no C++, enquanto para o *software* do computador optou-se pelo JavaScript em conjunto com tecnologias como *Handlebars*, HTML e SCSS/CSS para estilização e apresentação visual do sistema. O Node.js foi escolhido para ser utilizado no projeto devido a familiaridade do desenvolvedor com tecnologias WEB, que podem ser utilizadas para o desenvolvimento de aplicações *desktop* através do *framework Electron*.

O Node conta com o NPM (*Node Pack Manager*), gerenciador de pacotes do Node, que funciona através de uma CLI (*Comand Line Interface*) e permite gerenciar as bibliotecas utilizadas no projeto de forma simples e rápida. Na TABELA 2 é possível observar as principais biblitecas utilizadas no projeto e suas respectivas funções.

TABELA 2 – BIBLIOTECAS UTILIZADAS NO PROJETO

| **Biblioteca** | **Função** |
|---|---|
| HTTP | Criação de servidor WEB |
| Express | Gerenciamento de requisições HTTP de diferentes verbos |
| Express-handlebars | Criação de layouts e prevenção de repetição de código |
| Express-fileupload | Facilita o envio de arquivos |
| SQLite3 | Comunicação com o banco de dados SQLite |
| Serialport | Comunicação com portas seriais |
| Socket.io | Possibilita a comunicação entre o cliente e o servidor através de sockets |
| Gulp | Automação de tarefas de desenvolvimento |
| Gulp-sass | Fornece ao Gulp a capacidade de compilar códigos SCSS |
| Electron | Possibilita a criação de aplicativos desktop através de tecnologias web |

Fonte: Autor (2023)

O *framework* Sequelize foi utilizado no projeto para facilitar o desenvolvimento do *software*, já que através dele é possível realizar consultas no banco de dados sem utilizar comandos SQL diretamente no código da aplicação. Através da interface de linha de comando do Sequelize é possível criar *models*, que são representações de tabelas dos bancos de dados, e *migrations*, que são arquivos que permitem a criação, modificação e exclusão das tabelas. O Sequelize se mostrou mais prático quando comparado a seus concorrentes TypeORM[3] e Bookshelf.js[4].

Para este projeto, foi escolhido o VSCODE como a IDE de desenvolvimento, devido à sua flexibilidade, suporte a várias linguagens de programação e recursos de integração com outras ferramentas. Além disso, o VSCODE é uma ferramenta

[3] https://typeorm.io/
[4] https://bookshelfjs.org/

*open-source*, o que significa que é possível personalizá-lo e estendê-lo de acordo com as necessidades do projeto. Sua ampla comunidade de usuários e desenvolvedores também oferece suporte e contribuições úteis para aprimorar o desempenho da ferramenta.

Para permitir o versionamento do código-fonte e rastreabilidade das alterações realizadas durante o desenvolvimento, se fez necessário o uso do git juntamente com o github, que permite o armazenamento do repositório em nuvem.

### 4.4.1 API

Para que o *software* permita a comunicação com aplicações de embarcados criadas por outros programadores foi desenvolvida uma API que utiliza interface serial como meio de comunicação. Essa escolha se deu principalmente porque a maioria das placas de prototipagem eletrônica contam com essa interface de comunicação.

Na API desenvolvida, o JSON foi utilizado para representar os dados de requisição e resposta. Isso significa que quando um embarcado envia uma requisição para a API, ela é enviada em formato JSON, e quando a API retorna uma resposta, ela também é formatada em JSON. Dessa forma, é possível garantir a interoperabilidade entre os diferentes sistemas, independentemente da linguagem de programação ou plataforma utilizada.

FIGURA 6 – DADOS UTILIZADOS NA COMUNICAÇÃO COM A API


FONTE: o Autor (2023)

O valor referente a chave "*device*" é responsável por informar qual dispositivo enviou os dados, quando a informação parte dos embarcados, ou para qual dispositivo a

informação deve ser enviada, quando parte do *software* do computador. Conforme apresentado na seção 4.2.1, a infraestrutura projetada exige a utilização de três dispositivos embarcados. Nesse caso, são três os valores possíveis:

- 1 - Para o embarcado conectado ao computador
- 2 - Para o embarcado localizado na largada
- 3 - Para o embarcado localizado na chegada

O valor referente a chave "*operation*" é responsável por informar qual operação foi executada, quando a informação parte do embarcado, ou qual operação deve ser executada, quando parte do *software* do computador. São quatro os valores possíveis:

- 1 - Teste de conexão
- 2 - Teste de RFID e identificação dos pilotos durante a corrida
- 3 - Teste de RTC e atualização de hora
- 4 - Teste de interruptor (*start gate*, quando direcionado ao dispositivo localizado na largada, ou sensor fotoelétrico, quando direcionado ao dispositivo localizado na chegada)

O valor referente a chave "*status*" é preenchido pelos embarcados para informar se a operação solicitada deu certo ou não. Nesse caso, são possíveis apenas dois valores:

- 0 - Erro
- 1 - Ok

O valor do referente a chave "*message*" é preenchido pelos embarcados para fornecer informações adicionais sobre os eventos do sistema em forma de texto. Por exemplo, caso o *software* envie uma solicitação para o embarcado em um formato diferente de JSON, o embarcado deve retornar o valor da chave "*status*" igual a 0 e o campo "*message*" deve informar "JSON inválido.". Outro exemplo é quando há um

teste de conexão bem sucedido, em que a mensagem pode ser "Conexão realizada com sucesso.". A mensagem enviada pelo embarcado é apresentada ao usuário do *software* para computador.

Durante a corrida é possível acompanhar em tempo real a comunicação dos embarcados com o *software* do computador e, para tornar mais fácil a identificação dos *logs* dos competidores, é possível utilizar uma *tag* de substituição no campo "*message*" chamada "`[NOME_COMPETIDOR]`". Então, quando um competidor é identificado pelo sensor RFID localizado na largada, ele pode enviar a seguinte mensagem: "O competidor `[NOME_COMPETIDOR]` iniciou a prova.". Quando o *software* do computador receber a mensagem, ele substituirá "`[NOME_COMPETIDOR]`" pelo nome do competidor cadastrado com o RFID informado no campo "*data*".

Já o campo "*data*", por sua vez, serve para armazenar dados. Como dito anteriormente, um dado que pode ser gravado no campo "*data*" é o "rfid", que é o número de identificação da *tag* do piloto durante a corrida. Outro dado que pode ser gravado em "*data*" é o "*dateTime*", que pode ser utilizado em dois momentos diferentes: durante a corrida, para informar o tempo em que o competidor cruzou o *start gate* ou sensor fotoelétrico, ou ao realizar a operação de atualização de data e hora do módulo RTC, em que o *software* do computador envia a data e hora atual e o embarcado atualiza os dados e retorna a data e hora obtida do módulo RTC.

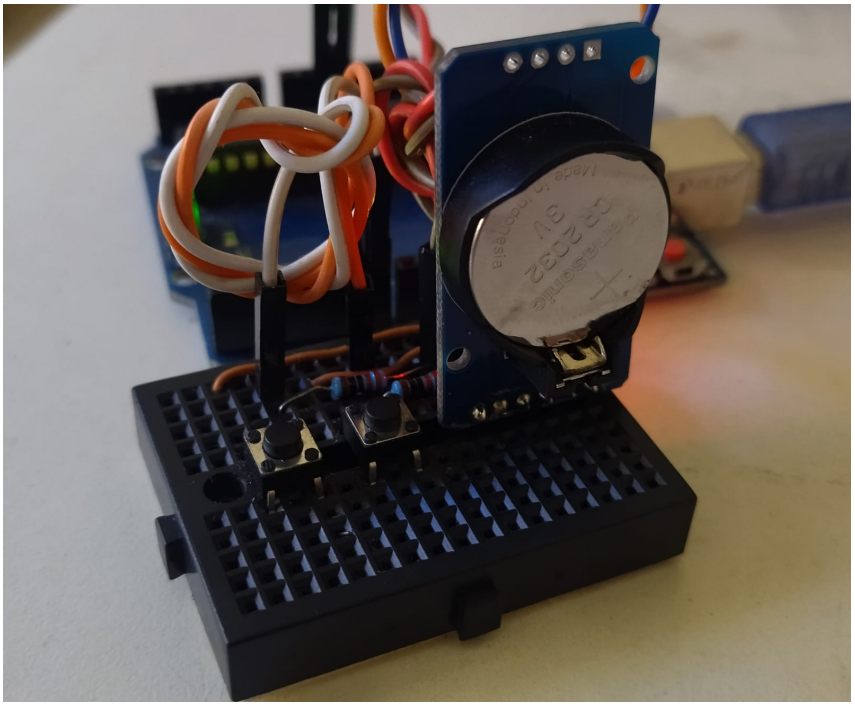
Vale ressaltar que a obrigatoriedade dos campos depende da operação realizada e de quem está fazendo a requisição. Nos momentos em que a requisição parte do *software* do computador, são obrigatórios os campos "*device*" e "*operation*". Já para requisições que partem dos embarcados, são obrigatórios os campos "*device*", "*operation*", "*status*" e "*message*". Para operações do tipo 2, é obrigatório que o embarcado informe os valores de "*dateTime*" e "rfid", dentro do campo "*data*". Já para operações do tipo 3, é obrigatório que os dispositivos comuniquem somente o valor de "*dateTime*" dentro do campo "*data*".

## 4.5 SISTEMA EMBARCADO DE TESTE DA API

Neste projeto, a placa de prototipagem Arduino Uno foi utilizada no projeto para simular o comportamento esperado por um sistema embarcado que tenha como

objetivo se conectar ao *software* proposto. Também foram utilizados dois botões, para simular a largada e chegada dos ciclistas, módulo RTC, para trabalhar com data e hora e *jumpers*, *protoboard* e resistores, para conexão dos componentes.

FIGURA 7 – SISTEMA EMBARCADO



FONTE: o Autor (2023)

Para simular uma competição, os códigos de RFID definidos para os competidores cadastrados no *software* do computador devem ser definidos manualmente e armazenados em uma constante no código do Arduino, como é possível observar na FIGURA 8.

FIGURA 8 – CONSTANTE QUE ARMAZENA OS CÓDIGOS RFID UTILIZADOS NO TESTE

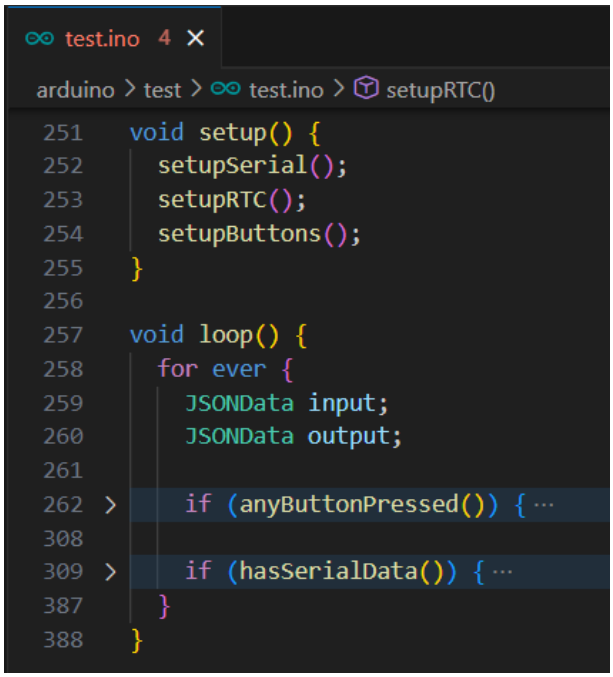
```
const String rfids[] = {
  "123456", "234567", "345678", "456789",
  "789012", "890123", "901234", "101234"
};
```

FONTE: o Autor (2023)

Quando o botão à esquerda é pressionado, ele envia a informação de que um dos códigos RFID listados no array da FIGURA 8 cruzou a linha de largada.

Isso continua até que todos os códigos RFID tenham sido enviados. Já o botão à direita, quando pressionado, envia a informação de que um dos códigos RFID, que anteriormente cruzou a linha de largada, agora cruzou a linha de chegada. Esse processo se repete até que todos os códigos RFID que anteriormente cruzaram a linha de largada sejam enviados.

FIGURA 9 – TRECHO DO CÓDIGO DO SISTEMA EMBARCADO




FONTE: o Autor (2023)

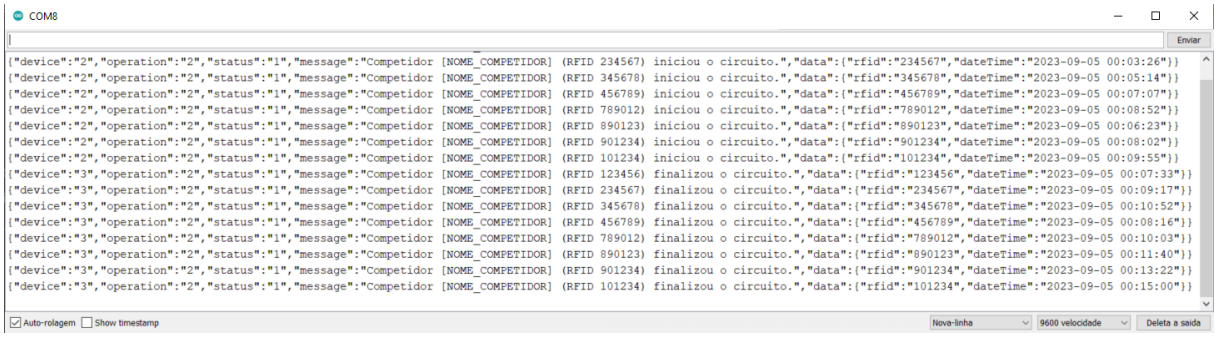
Conforme é possível observar na FIGURA 9, o trecho de código contém as funções "*setup*" e "*loop*" que são obrigatórias para programas escritos em Arduino. A função "*setup*" é executada uma vez assim que o dispositivo é ligado e é usada para implementar todos os códigos de configuração de funcionalidades, sensores e atuadores. Após a função "*setup*" ser executada, a função "*loop*" é executada repetidas vezes até que o dispositivo seja desligado e é nela que deve ser inseridas a lógica e as regras de negócio do *software*.

No trecho de código fornecido, a função "*setup*" é responsável por configurar a conexão serial, inicializar o módulo RTC e configurar os botões usados para simular o início e o término da corrida. Na função "*loop*", existem duas operações principais. A primeira verifica se o botão de largada ou chegada foi pressionado e, em caso afirmativo, envia essa informação por meio da interface serial. A segunda operação

verifica se há dados recebidos pela porta serial e gera uma resposta simulada com base na solicitação recebida.

Na FIGURA 10 é possível observar os dados enviados através da API serial ao pressionar os botões para simular a competição.

FIGURA 10 – DADOS ENVIADOS PARA SIMULAR PROVA DE *DOWNHILL*




FONTE: o Autor (2023)

## 4.6 TELAS DO SISTEMA

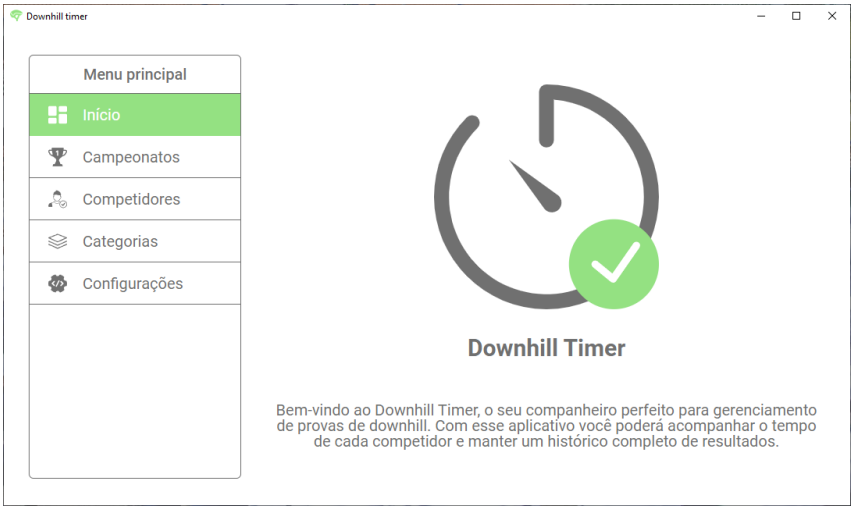
As telas de um sistema desempenham um papel fundamental na eficácia da interação com a aplicação, então devem ser bem trabalhadas para permitir uma boa experiência aos usuários. Segundo Cooper et al. (2014, p. 271), é importante que os usuários alcancem os seus objetivos sem que sejam submetidos a interações pesadas e trabalhos desnecessários.

Todos os recursos visuais do sistema passaram antes pela prototipagem, conforme apresentado na seção 4.2.2, e foram pensados de forma a atingir o objetivo de facilitar a utilização do produto. Abaixo será apresentado o resultado final de cada tela do sistema e uma breve descrição de suas funcionalidades.

### 4.6.1 Início

Após executar o *Downhill Timer*, nome escolhido para o *software* em questão, a tela inicial é exibida para o usuário. Conforme é possível observar na FIGURA 11, esta tela apresenta o nome do sistema, uma breve introdução sobre ele e, do lado esquerdo, o Menu Principal, que permite o acesso aos outros recursos.

FIGURA 11 – TELA INICIAL

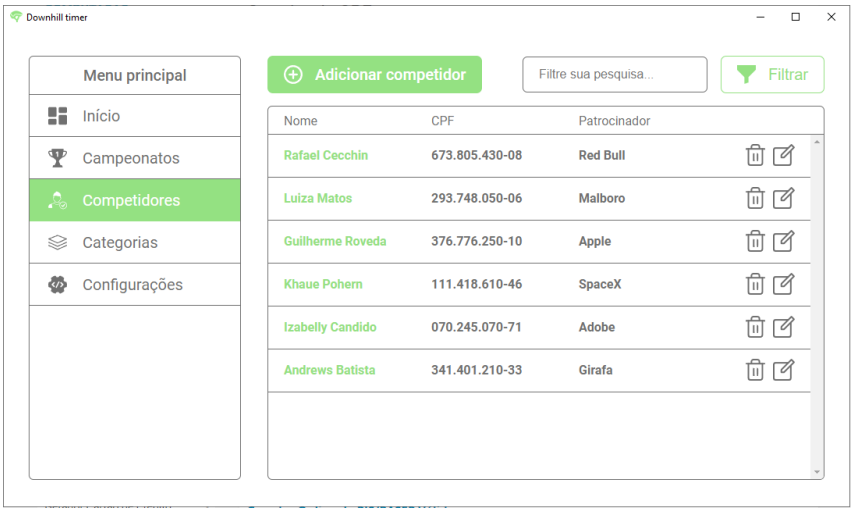



FONTE: o Autor (2023)

### 4.6.2 Apresentar competidores

Após clicar em "Competidores", no Menu Principal, a tela de apresentação dos competidores é exibida. Conforme é possível observar na FIGURA 12, esta tela apresenta um botão para adicionar um novo competidor, um filtro de pesquisa para facilitar a localização do competidor desejado e uma listagem dos competidores existentes com nome, CPF e patrocinador.

FIGURA 12 – TELA DE APRESENTAÇÃO DE COMPETIDORES

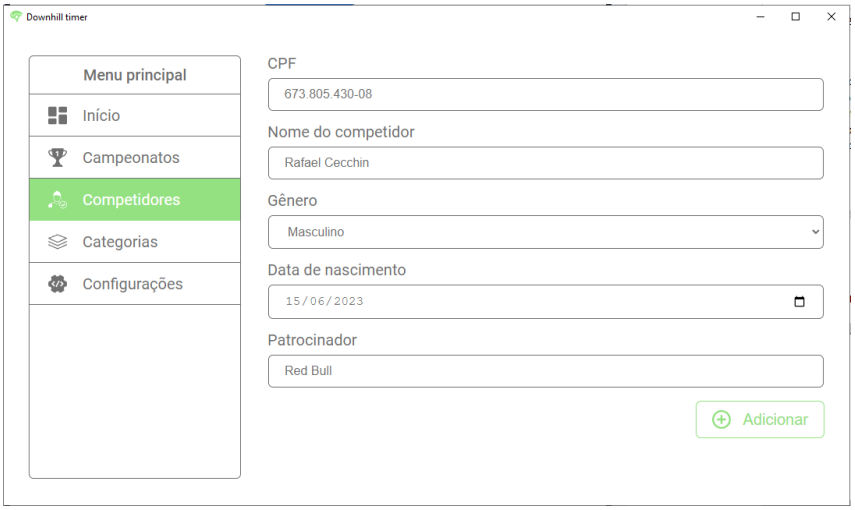



FONTE: o Autor (2023)

#### 4.6.2.1 Adicionar competidor

Após clicar em "Adicionar competidor", a tela de cadastro de competidores é exibida. Conforme é possível observar na FIGURA 13, há um botão para concretizar o cadastro e campos para que o usuário informe o CPF, nome, gênero, data de nascimento e patrocinador.

FIGURA 13 – TELA DE CADASTRO DE COMPETIDOR




FONTE: o Autor (2023)

#### 4.6.2.2 Apresentar competidor

Após criar ou acessar um competidor, a tela de apresentação é exibida e o CPF, nome, gênero, data de nascimento e patrocinador do competidor são apresentados. Conforme é possível observar na FIGURA 14, há um botão "Salvar", que deve ser utilizado caso algum dado do competidor seja alterado, e um botão "Excluir", para excluir o competidor.

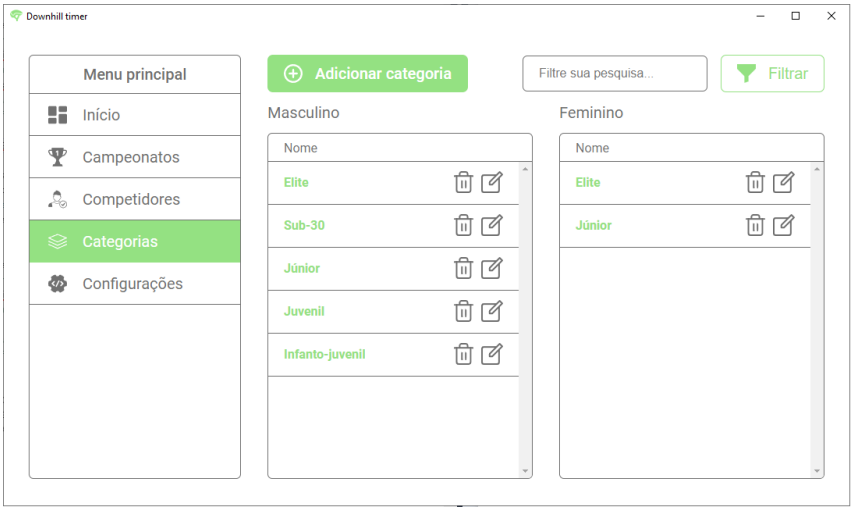
FIGURA 14 – TELA DE APRESENTAÇÃO DE COMPETIDOR


FONTE: o Autor (2023)

### 4.6.3 Apresentar categorias

Após clicar em "Categorias", no Menu Principal, a tela de apresentação de categorias é exibida. Conforme é possível observar na FIGURA 15, esta tela apresenta um botão para adicionar uma nova categoria, um filtro de pesquisa para facilitar a localização da categoria desejada e duas listagens das categorias existentes, separadas por gênero.

FIGURA 15 – TELA DE APRESENTAÇÃO DE CATEGORIAS

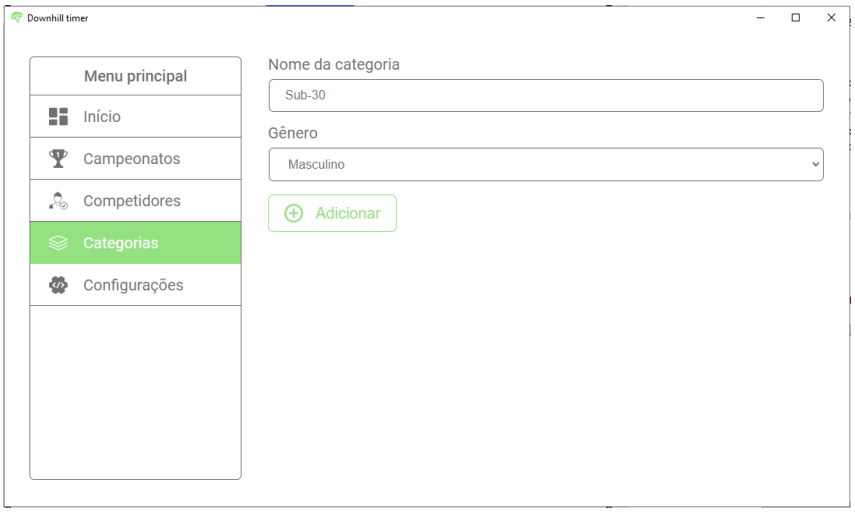



FONTE: o Autor (2023)

#### 4.6.3.1 Adicionar categoria

Após clicar em "Adicionar categoria", a tela de cadastro de categoria é exibida. Conforme é possível observar na FIGURA 16, há um botão para concretizar o cadastro e campos para que o usuário informe o nome e gênero da categoria.

FIGURA 16 – TELA DE CADASTRO DE CATEGORIA

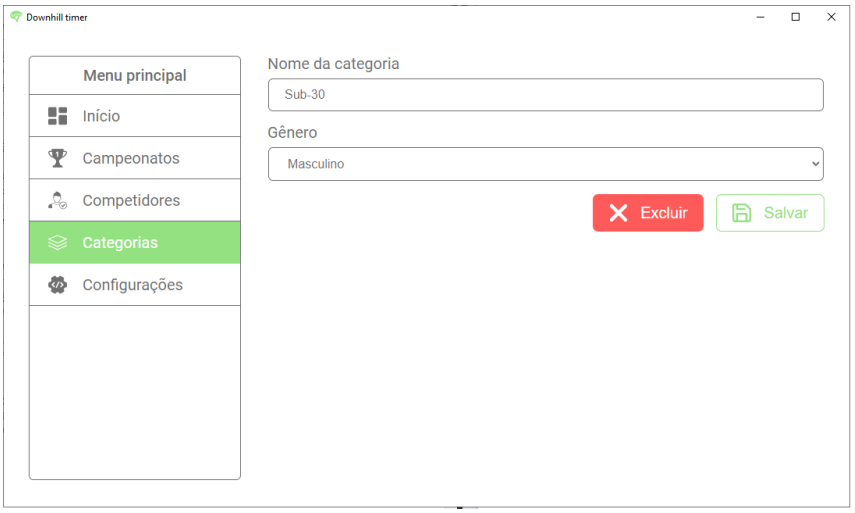



FONTE: o Autor (2023)

#### 4.6.3.2 Apresentar categoria

Após criar ou acessar uma categoria, a tela de apresentação é exibida e nome e gênero são apresentados. Conforme é possível observar na FIGURA 17, há um botão "Salvar", que deve ser utilizado caso algum dado da categoria seja alterado, e um botão "Excluir", para excluir a categoria.

FIGURA 17 – TELA DE APRESENTAÇÃO DE CATEGORIA

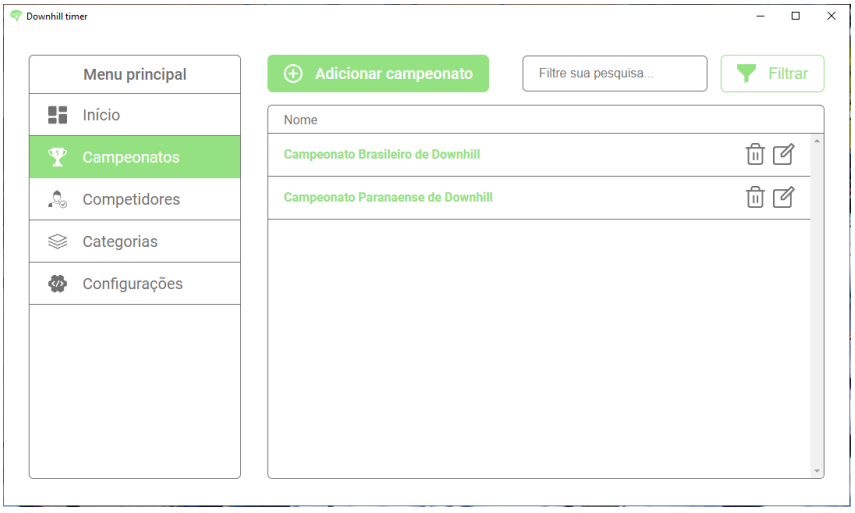



FONTE: o Autor (2023)

### 4.6.4 Apresentar campeonatos

Após clicar em "Campeonatos", no menu principal, a tela de apresentação de campeonatos é exibida. Conforme é possível observar na FIGURA 18, esta tela apresenta um botão para adicionar um novo campeonato, uma listagem dos campeonatos existentes e um filtro de pesquisa para facilitar a localização do campeonato desejado.

FIGURA 18 – TELA DE APRESENTAÇÃO DE CAMPEONATOS

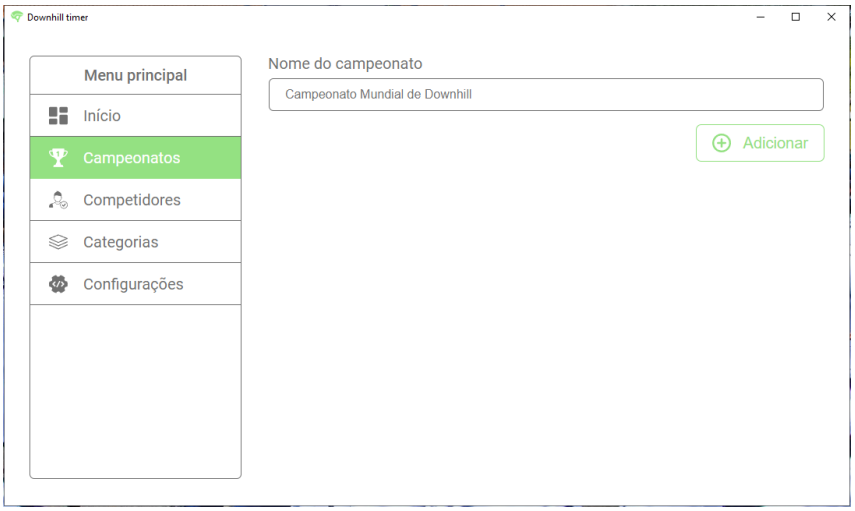



FONTE: o Autor (2023)

#### 4.6.4.1 Adicionar campeonato

Após clicar em "Adicionar campeonato", a tela de cadastro de campeonatos é exibida. Conforme é possível observar na FIGURA 19, há um campo para que o usuário informe o nome do campeonato e um botão para concretizar o cadastro.

FIGURA 19 – TELA DE CADASTRO DE CAMPEONATOS




FONTE: o Autor (2023)

#### 4.6.4.2 Apresentar campeonato

Após criar ou acessar um campeonato, a tela de apresentação de campeonato é exibida. Conforme é possível observar na FIGURA 20, há um botão "Salvar", que deve ser utilizado caso o nome do campeonato seja alterado, um botão "Excluir", para excluir o campeonato e todas as suas etapas, uma tabela que permite acessar todas as etapas do campeonato e um botão "Adicionar etapa", para adicionar uma nova etapa no campeonato.

FIGURA 20 – TELA DE APRESENTAÇÃO DE CAMPEONATO


FONTE: o Autor (2023)

#### 4.6.4.3 Adicionar etapa

Após clicar em "Adicionar etapa", a tela de cadastro de etapas é exibida. Conforme é possível observar na FIGURA 21, há um campo de seleção de campeonato, que é previamente preenchido com o campeonato utilizado para acessar esta tela, um campo para informar o número da etapa, um campo para informar a data da etapa e um botão para concretizar o cadastro.

FIGURA 21 – TELA DE CADASTRO DE ETAPA


FONTE: o Autor (2023)

#### 4.6.4.4 Apresentar etapa

Após criar ou acessar uma etapa, a tela de apresentação de etapa é exibida. O conteúdo dessa tela muda de acordo com o status da etapa, que pode ser "Aguardando", "Descida classificatória finalizada" ou "Prova finalizada". Entretanto, os campos principais são mantidos, como o status da etapa, a tabela que apresenta os competidores separados por gênero e categoria, o botão "Salvar", para aplicar as modificações realizadas, e o botão "Excluir", para excluir a etapa.

A tabela de exibição dos dos competidores apresenta o nome do competidor, código RFID, número da placa, tempo de início da descida classificatória (DCI), tempo de finalização da descida classificatória (DCF), tempo total da descida classificatória (DCT), tempo de início da prova (PI), tempo de finalização da prova (PF) e tempo total da prova (PT).

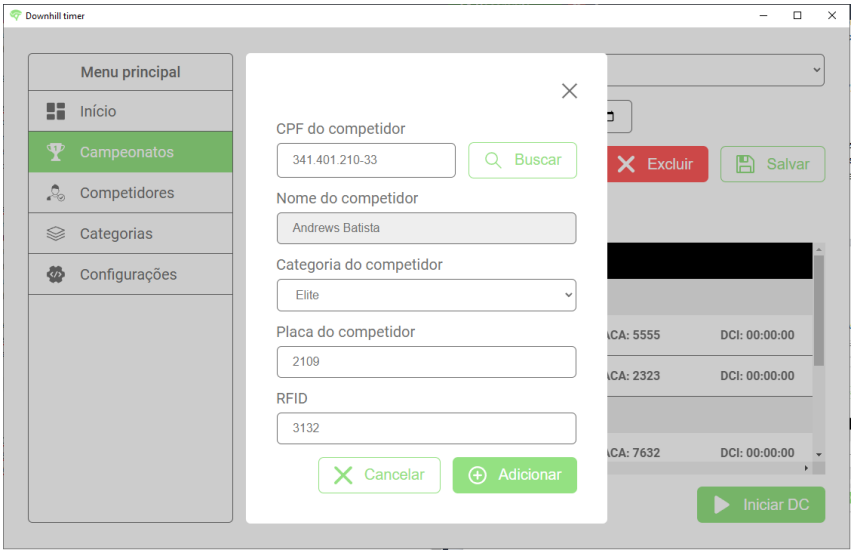
##### 4.6.4.4.1 Etapa com status "Aguardando"

FIGURA 22 – TELA DE APRESENTAÇÃO DE ETAPA COM STATUS "AGUARDANDO"


FONTE: o Autor (2023)

Conforme é possível observar na FIGURA 22, quando o status da etapa é "Aguardando" é possível adicionar novos competidores. Ao clicar em "Adicionar competidor", um modal com campos para informar o CPF do competidor, categoria, placa e RFID é exibido, junto com dois botões, um "Cancelar" e outro "Adicionar".

FIGURA 23 – MODAL PARA ADICIONAR COMPETIDOR NA ETAPA




FONTE: o Autor (2023)

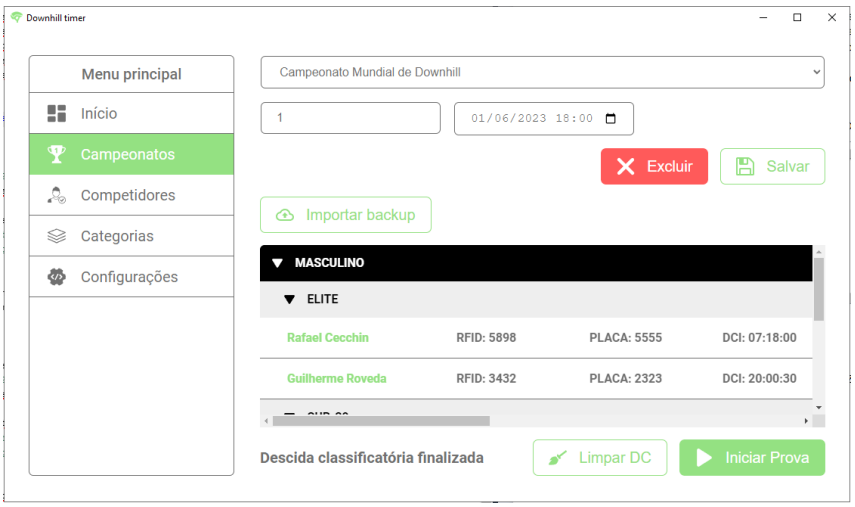
Quando o usuário seleciona a opção "Iniciar DC", isso indica o início da descida classificatória. A partir desse momento, o sistema começa a escutar os dados recebidos através da porta serial e esses registros são apresentados em um modal com dois botões. Se o usuário optar por "Finalizar", os registros recebidos são tratados, armazenados e a etapa passa a ter o status "Descida Classificatória Finalizada". No entanto, caso o usuário escolha "Cancelar", os registros recebidos serão descartados.

FIGURA 24 – MODAL DE APRESENTAÇÃO DO LOG DE REGISTROS DA DESCIDA CLASSIFICATÓRIA


FONTE: o Autor (2023)

#### 4.6.4.4.2 Etapa com status "Descida classificatória finalizada"

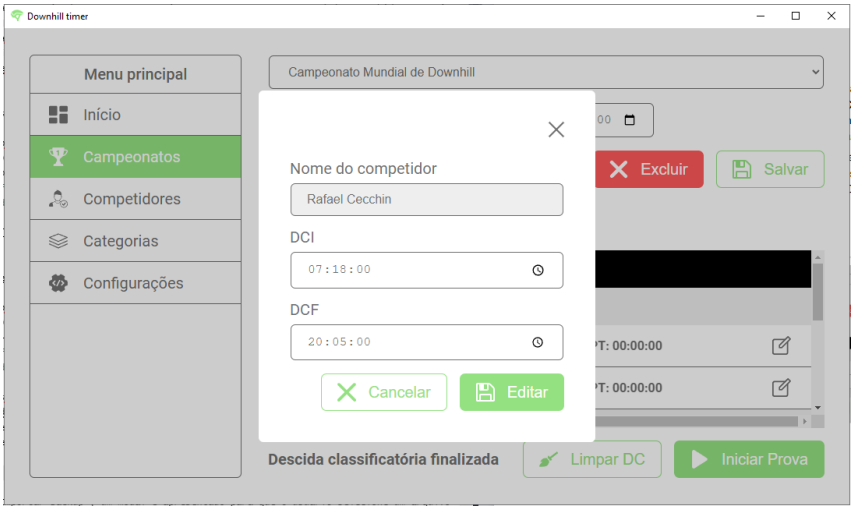
FIGURA 25 – TELA DE APRESENTAÇÃO DE ETAPA COM STATUS "DESCIDA CLASSIFICATÓRIA FINALIZADA"




FONTE: o Autor (2023)

Conforme é possível observar na FIGURA 25, quando o status da etapa é "Descida Classificatória Finalizada", os competidores são ordenados por melhor tempo dentro de suas categorias. São apresentados os botões "Importar *Backup*", "Iniciar Prova" e "Limpar DC", utilizado para limpar o tempo dos competidores e voltar a etapa para o status "Aguardando". Também é possível ajustar o tempo dos competidores manualmente, caso necessário.

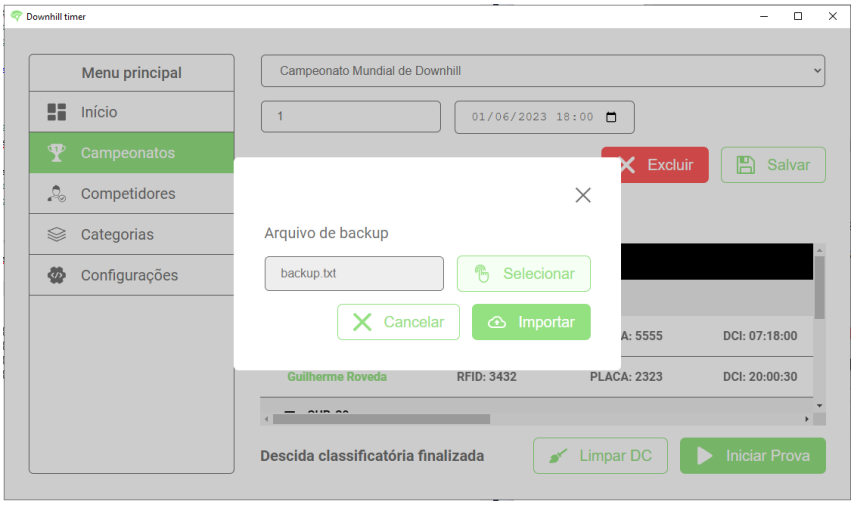
FIGURA 26 – MODAL DE EDIÇÃO DE TEMPO DO COMPETIDOR NA DESCIDA CLASSIFICATÓRIA




FONTE: o Autor (2023)

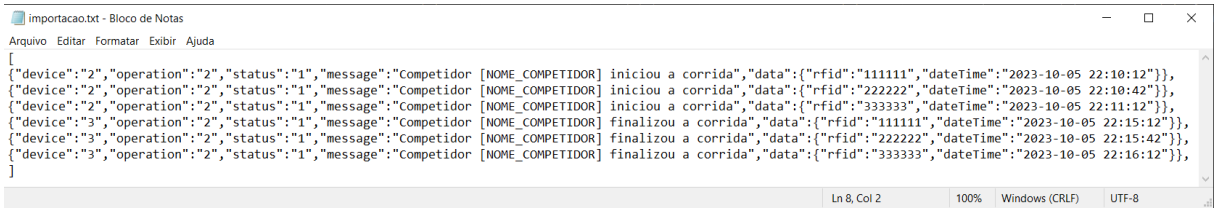
Ao clicar em "Importar *Backup*", um modal é apresentado para que o usuário selecione um arquivo com a extensão "TXT", conforme apresentado na FIGURA 27. O arquivo de backup deve estar em formato de uma lista JSON, deve conter todos os registros da corrida e cada elemento deve seguir os padrões estabelecidos pela API.

FIGURA 27 – MODAL DE IMPORTAÇÃO DE BACKUP




FONTE: o Autor (2023)

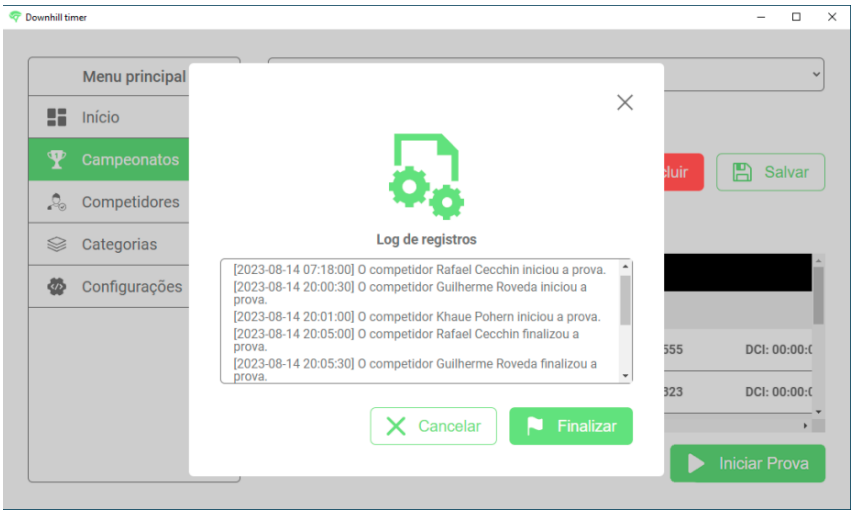
FIGURA 28 – EXEMPLO DE ARQUIVO DE BACKUP




FONTE: o Autor (2023)

Quando a opção "Iniciar Prova" é selecionada, o sistema começa a escutar os dados recebidos através da porta serial e esses registros são apresentados para o usuário através de um modal. Se o usuário optar por "Finalizar", os registros recebidos são tratados, armazenados e a etapa passa a ter o status "Prova finalizada". No entanto, caso o usuário escolha "Cancelar", os registros recebidos serão descartados.

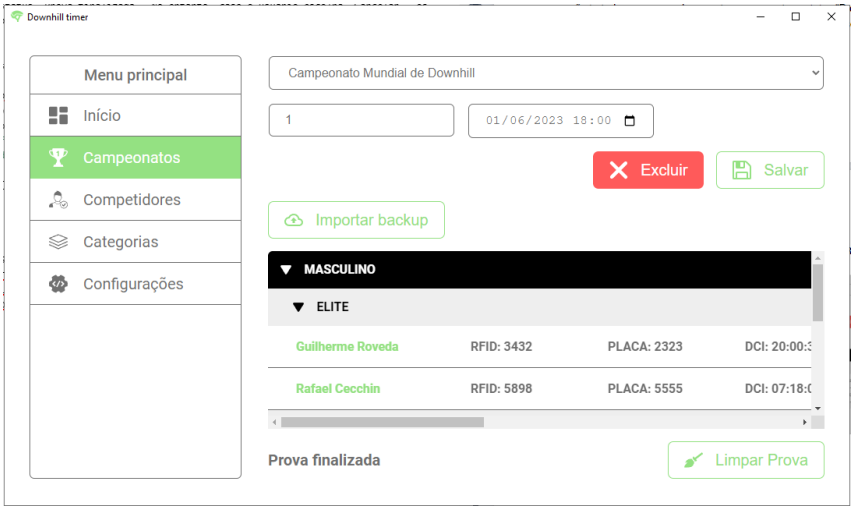
FIGURA 29 – MODAL DE APRESENTAÇÃO DO LOG DE REGISTROS DA PROVA




FONTE: o Autor (2023)

#### 4.6.4.4.3 Etapa com status "Prova Finalizada"

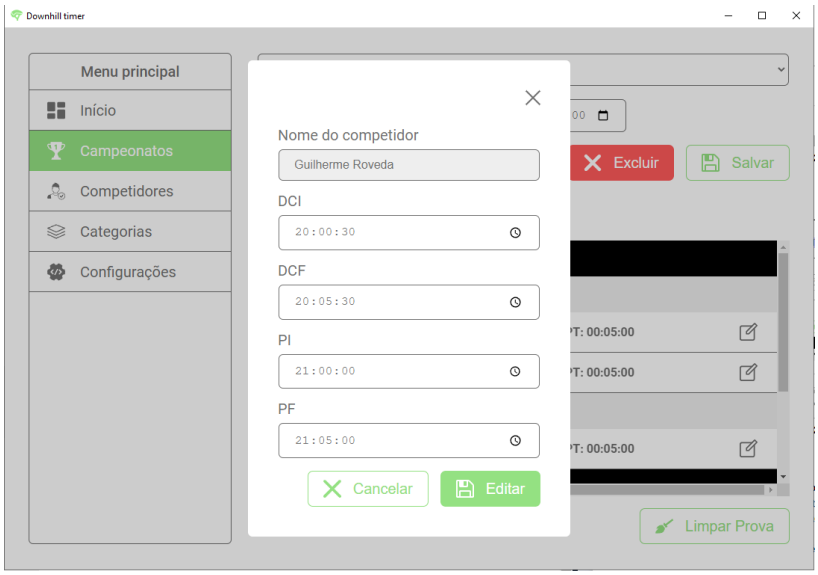
FIGURA 30 – TELA DE APRESENTAÇÃO DE ETAPA COM STATUS "PROVA FINALIZADA"




FONTE: o Autor (2023)

Conforme é possível observar na FIGURA 30, quando o status da etapa é "Prova Finalizada", os competidores são ordenados por melhor tempo dentro de suas categorias. São apresentados os botões "Importar *Backup*" e "Limpar Prova", utilizado para limpar o tempo dos competidores e voltar a etapa para o status "Descida Classificatória Finalizada". Também é possível ajustar o tempo dos competidores manualmente, caso necessário.

FIGURA 31 – MODAL DE EDIÇÃO DE TEMPO DO COMPETIDOR NA PROVA




FONTE: o Autor (2023)

### 4.6.5 Apresentar configurações

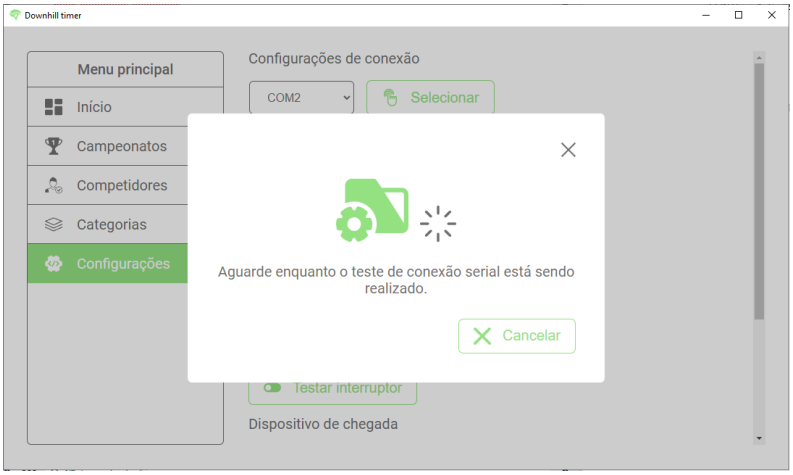
Após clicar em "Configurações", no Menu Principal, a tela de apresentação de configurações é exibida. Conforme é possível observar na FIGURA 32, esta tela apresenta uma entrada para informar a porta serial utilizada na comunicação e diversos botões para testar os componentes externos.

FIGURA 32 – TELA DE APRESENTAÇÃO DE CONFIGURAÇÕES


FONTE: o Autor (2023)

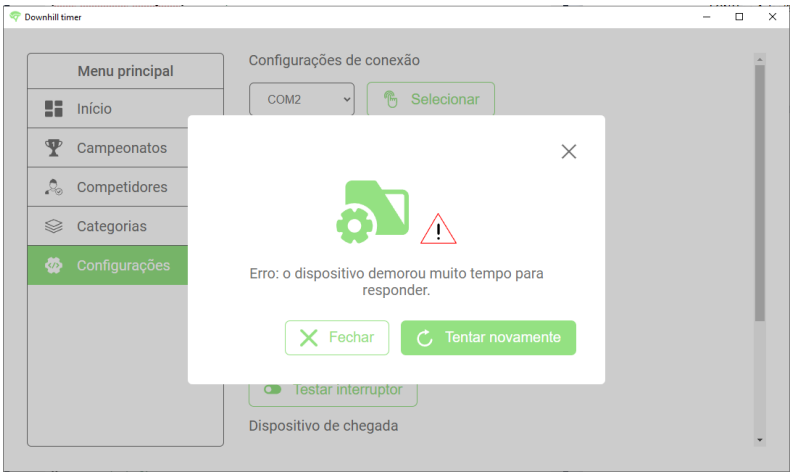
Ao clicar para realizar algum teste, é apresentado um modal que é alterado conforme o status do teste, que pode ser "aguardando", "erro" ou "sucesso".

FIGURA 33 – MODAL DE TESTE COM STATUS "AGUARDANDO"




FONTE: o Autor (2023)

FIGURA 34 – MODAL DE TESTE COM STATUS "ERRO"




FONTE: o Autor (2023)

FIGURA 35 – MODAL DE TESTE COM STATUS "SUCESSO"


FONTE: o Autor (2023)

# 5 TESTE

Para validar o funcionamento do *software*, foi projetado um cenário que se assemelha a uma competição de *downhill*. Nessa projeção, imagina-se que o responsável por organizar a competição recebeu uma lista com os dados dos competidores cadastrados, designou suas identificações e dispôs os dados na TABELA 3. Agora o organizador deve cadastrá-los no *software* e gerir a competição que está prestes a acontecer. Exposta a situação problema, serão apresentados os passos que o organizador deve realizar no *software* para atingir seu objetivo.

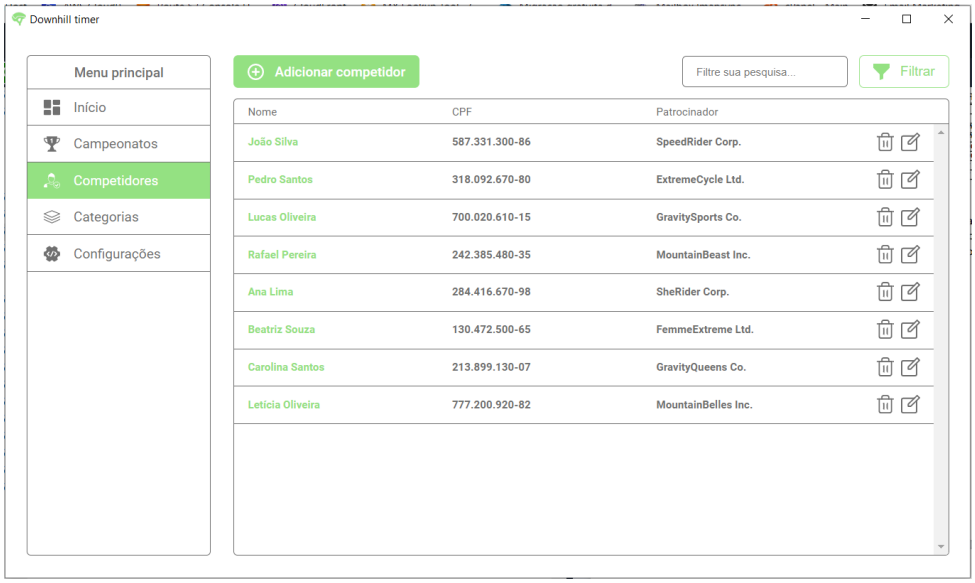
TABELA 3 – COMPETIDORES CADASTRADOS NA COMPETIÇÃO

| CPF | Nome | Gênero | Nascimento | Patrocinador | Categoria | Placa | RFID |
|---|---|---|---|---|---|---|---|
| 58733130086 | João Silva | M | 15/03/1990 | SpeedRider Corp. | Elite | ABC123 | 123456 |
| 31809267080 | Pedro Santos | M | 28/06/1995 | ExtremeCycle Ltd. | Elite | DEF456 | 234567 |
| 70002061015 | Lucas Oliveira | M | 10/11/1998 | GravitySports Co. | Master A1 | GHI789 | 345678 |
| 24238548035 | Rafael Pereira | M | 02/04/1992 | MountainBeast Inc. | Master A1 | JKL012 | 456789 |
| 28441667098 | Ana Lima | F | 25/07/1998 | SheRider Corp. | Amador | ABC987 | 789012 |
| 13047250065 | Beatriz Souza | F | 08/12/1991 | FemmeExtreme Ltd. | Amador | DEF876 | 890123 |
| 21389913007 | Carolina Santos | F | 20/09/2001 | GravityQueens Co. | Júnior | GHI765 | 901234 |
| 77720092082 | Letícia Oliveira | F | 08/10/2002 | MountainBelles Inc. | Júnior | JKL654 | 101234 |

Fonte: Autor (2023)

Antes de tudo, é necessário instalar o *software* a partir do arquivo de instalação disponibilizado pelo desenvolvedor. Depois disso, o usuário deve executar o sistema através do atalho localizado na área de trabalho do seu dispositivo e acessar "Competidores", para fazer o cadastro dos competidores da tabela. No teste realizado, os cadastros foram realizados com sucesso, conforme é possível observar na FIGURA 36.

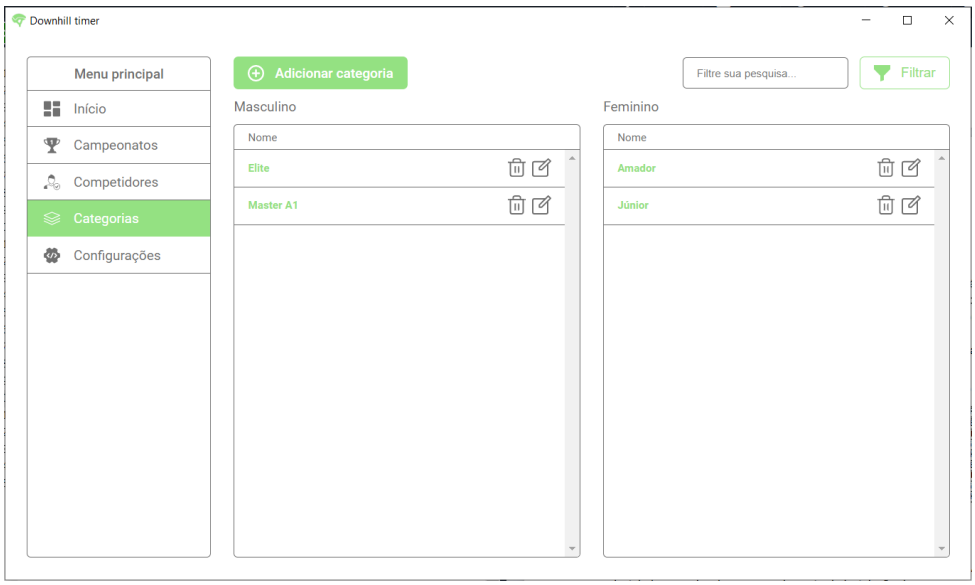
FIGURA 36 – COMPETIDORES CADASTRADOS NO SISTEMA




FONTE: o Autor (2023)

Após cadastrar os competidores, o usuário deve acessar "Categorias", no menu lateral esquerdo, e realizar o cadastro das categorias presentes na TABELA 3. Conforme é possível observar na FIGURA 37, todas as categorias foram cadastradas com sucesso.

FIGURA 37 – CATEGORIAS CADASTRADAS NO SISTEMA



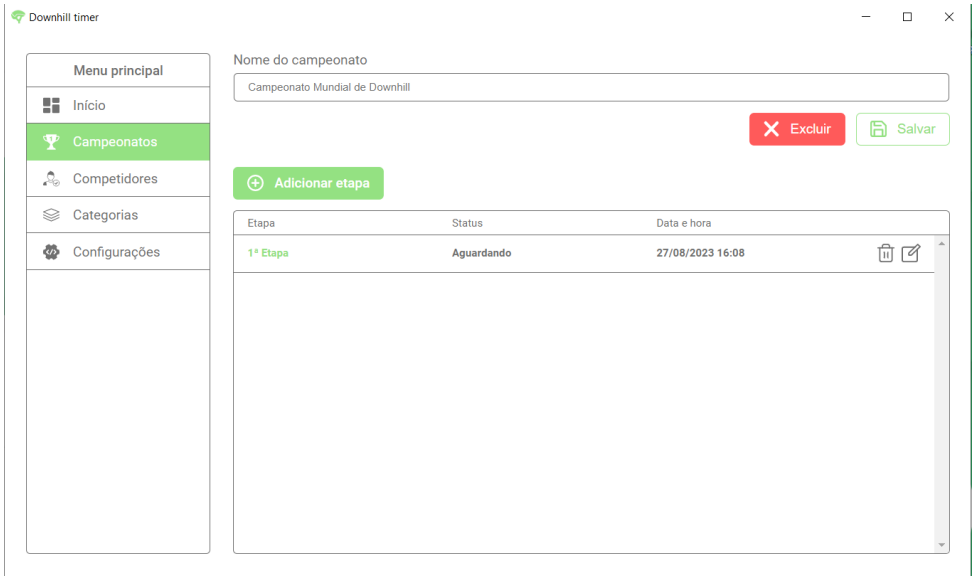

FONTE: o Autor (2023)

Outro passo que o usuário deve realizar é conectar o *hardware* embarcado ao computador, acessar "Configurações" e selecionar a porta em que o dispositivo está

conectado. Feito isso, o usuário pode realizar todos os testes para inferir se a conexão está respondendo como esperado. Como apresentado na seção 4.5, o embarcado utilizado no projeto foi programado para atender o respectivo teste, comunicando dados no formato esperado pela API. Nos testes realizados, o dispositivo respondeu como esperado.

Agora o usuário deve se direcionar até "Campeonatos", no menu lateral esquerdo, cadastrar o campeonato de nome "Campeonato Mundial de *Downhill*" e, após isso, cadastrar a primeira etapa do campeonato. Como é possível observar na FIGURA 38, os cadastros foram realizados com sucesso.

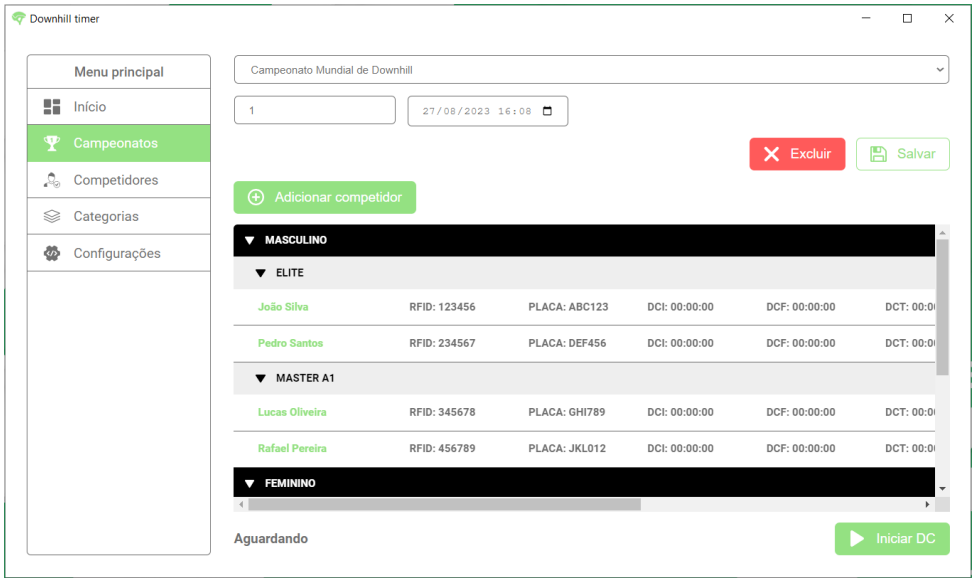
FIGURA 38 – ETAPAS CADASTRADAS NO SISTEMA




FONTE: o Autor (2023)

Após cadastrar a etapa é possível inserir os competidores da tabela e, para isso, o usuário deve acessar a etapa, pressionar "Adicionar competidor", informar o CPF do competidor cadastrado e clicar em "Buscar". Ao encontrar o competidor desejado, é possível informar sua placa de identificação, seu RFID e sua categoria. Concluído o preenchimento, o usuário deve clicar em "Adicionar". Todos os cadastros foram realizados com sucesso, como é possível observar na figura FIGURA 39.

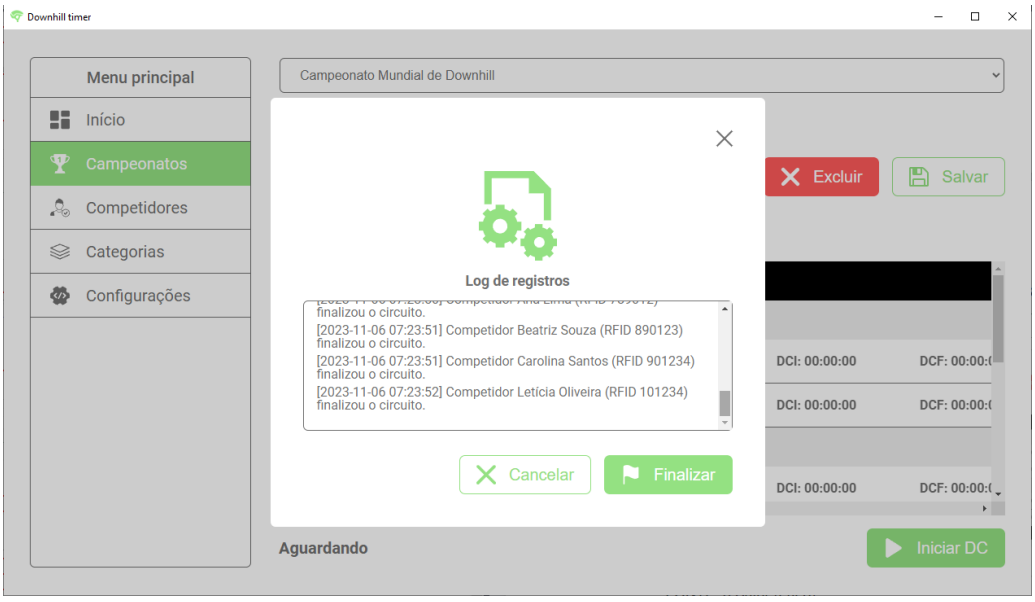
FIGURA 39 – COMPETIDORES CADASTRADOS COM SUCESSO




FONTE: o Autor (2023)

Ao clicar em "Iniciar DC" e pressionar repetidas vezes os botões de largada e chegada do sistema embarcado conectado ao computador, foi possível obter o tempo de início e fim dos competidores na descida classificatória. Ao clicar em "Finalizar", os dados foram salvos com sucesso no sistema. Também foi possível importar o *backup* com sucesso. Na FIGURA 40 é possível observar o *log* de registros.

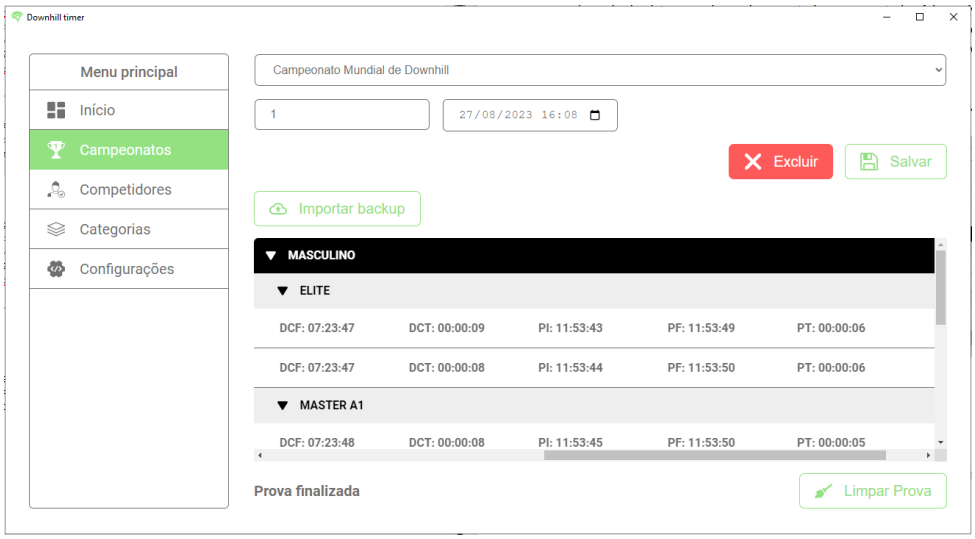
FIGURA 40 – LOG DE REGISTROS DA DESCIDA CLASSIFICATÓRIA NO TESTE




FONTE: o Autor (2023)

Também foi possível fazer o registro e importação dos dados de *backup* da prova final. Na FIGURA 41 é possível observar os dados de tempo dos competidores preenchidos.

FIGURA 41 – DADOS DA PROVA NO TESTE




FONTE: o Autor (2023)

No teste realizado foi possível observar que todas as etapas, desde o cadastro de competidores e categorias até a execução das competições, foram concluídas com sucesso. O *software* mostrou-se eficaz na interação com o usuário e na integração com o *hardware* embarcado.

# 6 RESULTADOS

No contexto das competições de *downhill*, onde a precisão e eficiência são essenciais, a necessidade de um sistema de registro digitalizado se torna evidente. Organizar eventos complexos e dinâmicos requer um mecanismo que vá além dos métodos tradicionais de papel e caneta. Um sistema informatizado além de simplificar o processo de inscrição e gerenciamento de competidores, também proporciona uma visão aprofundada do desempenho dos atletas e facilita a análise de dados em tempo real.

Uma funcionalidade fundamental do sistema é o registro completo dos competidores, incluindo informações como nome, CPF, data de nascimento, patrocinador e categoria. Além disso, esses dados não apenas simplificam o processo de inscrição, mas também ajudam os organizadores a manterem um controle detalhado dos participantes. Ao eliminar a papelada e facilitar a organização, o sistema torna a gestão das inscrições mais simples.

A capacidade de criar campeonatos e suas etapas é uma característica essencial do sistema. Ao possibilitar o planejamento das datas de cada etapa, o sistema oferece uma visão clara do calendário de corridas, tornando a experiência de todos ainda mais profissional e bem organizada.

O sistema também oferece uma funcionalidade essencial ao permitir o cadastro de diversas categorias, possibilitando que os organizadores classifiquem os competidores de acordo com idade e habilidade. Com isso, essa flexibilidade garante uma competição justa e emocionante, onde cada participante compete em um ambiente adequado ao seu nível.

O aspecto mais inovador do sistema é sua integração com *hardware* embarcado. Juntamente com isso, a capacidade de capturar dados em tempo real, como tempos de largada e chegada, é incrivelmente valiosa para os organizadores e competidores. Esses dados ajudam a monitorar o desempenho dos competidores e podem ser utilizados para análises detalhadas, identificando padrões e áreas de melhoria.

Em resumo, o sistema integra várias funcionalidades cruciais para uma com-

petição de *downhill*. Isso inclui o registro detalhado dos competidores, a organização clara de campeonatos e etapas, a definição de categorias e integração com *hardware* embarcado. Ao oferecer uma plataforma centralizada e automatizada para gerenciar esses aspectos, o sistema aumenta a qualidade e o profissionalismo das competições, proporcionando uma experiência excepcional para todos os envolvidos.

# 7 CONSIDERAÇÕES FINAIS

O gerenciamento de um campeonato de *downhill* demanda considerável esforço por parte dos organizadores, uma vez que uma variedade de tarefas precisa ser coordenada para assegurar o bom andamento do evento. Essas responsabilidades incluem a elaboração do calendário de provas, o processo de inscrição dos competidores, a divisão destes em suas respectivas etapas e a cronometragem do tempo durante as descidas classificatórias e provas de cada fase do campeonato.

O objetivo central deste projeto é simplificar o gerenciamento de competições de *downhill* através do desenvolvimento de um *software* para computador. Este *software*, devidamente elaborado, permite a centralização de todas as informações relacionadas a competidores, campeonatos e etapas em um único local.

Antes de iniciar o desenvolvimento, foi necessário compreender o funcionamento das competições de *downhill*, para o qual foram realizadas pesquisas bibliográficas sobre o tema. Além disso, durante o levantamento de requisitos, reuniões foram realizadas junto a praticantes do esporte para o esclarecimento de dúvidas.

Posteriormente, a infraestrutura completa do sistema foi projetada, o que permitiu a definição precisa dos padrões de comunicação entre o *software* para computador e um *hardware* embarcado que venha a se comunicar com ele.

Após isso, o *software* foi desenvolvido de acordo com a infraestrutura projetada. Também foi desenvolvido um programa para o dispositivo embarcado que possibilitou a realização de testes que confirmaram que é possível a integração com o *software* para computador. Nos testes realizados, o sistema revelou-se promissor na assistência ao gerenciamento das provas de *downhill*.

Após a conclusão dos testes, a visibilidade do repositório do projeto no GitHub foi ajustada para possibilitar que outras pessoas tenham acesso ao código-fonte do sistema. Essa abertura proporciona uma oportunidade valiosa para outros desenvolvedores explorarem, aprenderem e, se desejarem, construírem sobre o que foi desenvolvido até o momento. O repositório pode ser acessado através do seguinte link: https://github.com/RafaelCecchin/downhill-timer.

Através desse sistema, tornou-se possível cadastrar, editar e excluir campeonatos, etapas e competidores. Além disso, a conexão com dispositivos embarcados desenvolvidos de acordo com o padrão estabelecido pela API permite o registro dos dados de cronometragem de cada competidor em tempo real.

# 8 TRABALHOS FUTUROS

Uma área promissora para o aprimoramento do sistema de registro de competições de *downhill* é a integração mais profunda com dispositivos embarcados. Atualmente, o sistema já dispõe de uma API por meio de porta serial para a comunicação com dispositivos embarcados, possibilitando a interação entre o *software* e os dispositivos físicos. No entanto, para uma implementação prática e eficiente, é crucial desenvolver e utilizar dispositivos embarcados reais capazes de capturar dados precisos e confiáveis durante as corridas.

Na arquitetura atual do sistema, o registro do patrocinador é apenas uma coluna na tabela de competidores do banco de dados. Para um registro mais preciso e em conformidade com as regras de normalização, é necessário criar uma nova tabela para os patrocinadores, estabelecendo uma relação entre o patrocinador e o competidor.

Outra melhoria importante consiste em permitir que o *software* armazene o registro de alterações manuais realizadas nos tempos dos competidores. Ao efetuar uma modificação no tempo de um competidor, o sistema também deve solicitar ao usuário o motivo da alteração. Essa abordagem não apenas proporciona uma trilha de auditoria, mas também contribui para a transparência do processo.

Em resumo, aprimorar a integração com dispositivos embarcados, otimizar a distribuição dos dados e implementar o registro de alterações nos tempos dos competidores são direções de pesquisa para melhorar o sistema de competições de *downhill*. Ao explorar essas possibilidades, o sistema poderá oferecer uma solução mais avançada e precisa para o universo das corridas de *downhill*, beneficiando tanto os competidores quanto os entusiastas do esporte.

# REFERÊNCIAS

AMAZON WEB SERVICES. **O que é IDE (Ambiente de Desenvolvimento Integrado)?** [S.l.: s.n.], 2023. <https://aws.amazon.com/pt/what-is/ide/>. Acesso em: 30 de setembro de 2023.

ARDUINO. **Documentação de Referência da Linguagem Arduino**. [S.l.: s.n.], 2023. <https://www.arduino.cc/reference/pt/>. Acesso em: 30 de setembro de 2023.

AUGUSTIN, Aloÿs et al. A study of LoRa: Long range & low power networks for the internet of things. **Sensors**, MDPI, v. 16, n. 9, p. 1466, 2016.

BANZI, Massimo; SHILOH, Michael. **Getting started with Arduino**. [S.l.]: Maker Media, Inc., 2014.

BARROS, João Paulo. Casos de uso e respectivos diagramas. **Escola Superior de Tecnologia e Gestão Instituto Politécnico de Beja**, 2009.

CHACON, Scott; STRAUB, Ben. **Pro git**. [S.l.]: Springer Nature, 2014.

CHASE, Otavio; ALMEIDA, F. Sistemas embarcados. **Mıdia Eletrônica. Página na internet:< www. sbajovem. org/chase>, capturado em**, v. 10, n. 11, p. 13, 2007.

COOPER, Alan et al. **About face: the essentials of interaction design**. [S.l.]: John Wiley & Sons, 2014.

FLATSCHART, Fábio. **HTML 5-Embarque Imediato**. [S.l.]: Brasport, 2011.

HEUSER, Carlos Alberto. Projeto de Banco de Dados, 6ª edição. **Instituto de Informática. Editora Sagra**, 1998.

INTERNATIONALE, Union Cycliste. UCI cycling regulations: Part I - General organisation of cycling as a sport. **Union Cycliste Internationale, Switzerland**, p. 1–93, 2023.

_____. UCI cycling regulations: Part IV - Mountain bike races. **Union Cycliste Internationale, Switzerland**, p. 1–98, 2023.

JAYAKODY, Chaminda et al. Efficient API Migration across Environments. **International Journal of Computer Applications**, v. 975, p. 8887.

MENEZES, Matheus Henrique Dinato; SATO, Victor Yuji. **Desenvolvimento de interface de comunicação baseada no protocolo Modbus para conexão de um computador (PC) a um inversor de Frequência**. 2019. F. 57. Trabalho de Conclusão de Curso – Universidade de Brasília, Brasília.

NOGUEIRA FILHO, Cıcero Casemiro da Costa. Tecnologia RFID aplicada à Logıstica. **Rio de Janeiro**, 2005.

OVERFLOW, Stack. **Developer Survey 2023**. [S.l.: s.n.], 2023. <https://survey.stackoverflow.co/2023/>. Acesso em: 07 de maio de 2023.

PASIC, Roberto; KUZMANOV, Ivo; ATANASOVSKI, Kokan. ESP-NOW communication protocol with ESP32. **Journal of Universal Excellence**, Faculty of Organizational Sciences-Slovenia, v. 6, n. 1, p. 53–60, 2021.

SASS. **Sass: Syntactically Awesome Style Sheets**. [S.l.: s.n.], 2023. <https://sass-lang.com/>. Acesso em: 15 de setembro de 2023.

SEQUELIZE. **Sequelize Documentation**. [S.l.: s.n.], 2023. <https://sequelize.org/docs/v6/>. Acesso em: 15 de maio de 2023.

SHARMA, Vatsal; TIWARI, A Kumar. A Study on User Interface and User Experience Designs and its Tools. **World Journal of Research and Review (WJRR)**, v. 12, n. 6, 2021.

SILVA DIAS, Ariel da. **Bancos de dados não relacionais**. [S.l.]: Editora Senac São Paulo, 2023.

STROUSTRUP, Bjarne. An overview of the C++ programming language. **Handbook of object technology**, CRC Press, p. 72, 1999.

SUMATHI, Sai; ESAKKIRAJAN, S. **Fundamentals of relational database management systems**. [S.l.]: Springer, 2007. v. 47.

TANENBAUM, Andrew S; ZUCCHI, Wagner Luiz. **Organização estruturada de computadores**. [S.l.]: Pearson Prentice Hall, 2013.

TEIXEIRA, Pedro. **Hands-on Node. js**. [S.l.]: Retrieved on May, 2011.

TEIXEIRA, Tiago. Controle de Fluxo de Pessoas Usando RFId. **Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina-Campus São José, São José-SC**, 2011.

TSAI, Wei-Feng. **A Low Cost RFID Tracking and Timing System for Bike Races**. 2011. Tese (Doutorado) – The Ohio State University.

USABILITY.GOV. **Prototyping**. [S.l.: s.n.], 2013. <https://www.usability.gov/how-to-and-tools/methods/prototyping.html>. Acesso em: 07 de maio de 2023.