Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Setter: Software Especializado em Gerenciamento de Times Amadores de Voleibol

Autor: Hugo Sobral de Lima Salomão e Micaella Lorraine Gouveia de Lima

Orientadora: Profª. Drª. Milene Serrano

Brasília, DF

2023

Hugo Sobral de Lima Salomão e Micaella Lorraine Gouveia de Lima

Setter: Software Especializado em Gerenciamento de Times Amadores de Voleibol

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Profª. Drª. Milene Serrano

Brasília, DF

2023

Hugo Sobral de Lima Salomão e Micaella Lorraine Gouveia de Lima

Setter: Software Especializado em Gerenciamento de Times Amadores de Voleibol/ Hugo Sobral de Lima Salomão e Micaella Lorraine Gouveia de Lima. – Brasília, DF, 2023-

165 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof[a]. Dr[a]. Milene Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Vôlei. 2. Software. I. Prof[a]. Dr[a]. Milene Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Setter: Software Especializado em Gerenciamento de Times Amadores de Voleibol

CDU 02:141:005.6

Hugo Sobral de Lima Salomão e Micaella Lorraine Gouveia de Lima

Setter: Software Especializado em Gerenciamento de Times Amadores de Voleibol

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 26 de Julho de 2023:

Profª. Drª. Milene Serrano
Orientadora

Profª. Drª. Carla Silva Rocha Aguiar
Examinador 1

Prof. Dr. Maurício Serrano
Examinador 2

Brasília, DF
2023

Este trabalho é dedicado a nossos entes queridos, que sempre nos forneceram apoio e forças durante nossa trajetória acadêmica.

Agradecimentos

Agradecemos às nossas famílias, que por vezes foram capazes de nos fornecer a base para nosso desenvolvimento intelectual e, eventualmente, para o desenvolvimento deste trabalho. Suas palavras de incentivo e estímulo foram importantes para a nossa motivação e determinação. De coração, agradecemos a vocês, familiares, por tudo o que fizeram por nós.

Agradecemos imensamente aos nossos amigos de curso, que estiveram ao nosso lado durante nossa trajetória na Universidade de Brasília. Sua amizade e apoio foram fundamentais para nossas motivação e dedicação, sempre nos fizeram acreditar que tudo seria possível. As noites de estudo em grupo, as mais diversas discussões e as risadas compartilhadas tornaram este processo muito mais agradável e enriquecedor. Além disso, as colaborações e disponibilidades em ajudar em momentos difíceis foram decisivas para o sucesso deste projeto. Desejamos, com um carinho especial, um muito obrigado(a) ao Leonardo, ao Durval, à Sofia, ao Ésio, ao Gabriel Davi, ao Iuri, à Thais, e a tantos outros que tivemos o prazer de conhecer em ambiente acadêmico!

Eu, Micaella, agradeço ainda ao meu parceiro de relacionamento, que foi fonte constante de motivação para o desenvolvimento deste trabalho. As horas dedicadas à leitura e á escrita foram intensas e, por muitas vezes, nos afastaram. No entanto, sua paciência e compreensão, mesmo diante de nossa ausência, permitiram que eu pudesse continuar a avançar e concluir este trabalho. Obrigada, Durval!

Gostaríamos de expressar nossa profunda gratidão à nossa orientadora, Profa. Dra. Milene Serrano. Durante todo o desenvolvimento do Trabalho de Conclusão de Curso, sua orientação, acompanhamento e paciência foram fundamentais para o sucesso deste trabalho. Seu vasto conhecimento e domínio em Engenharia de Software nos permitiram avançar com confiança e segurança, sempre nos encorajando a buscar novas perspectivas e soluções. Além disso, suas disponibilidade e dedicação em nos ajudar, mesmo diante de prazos apertados e situações desafiadoras, foram decisivos para a conclusão bem-sucedida deste projeto.

A todos que contribuiram para o sucesso da nossa formação, de forma direta ou indireta, deixamos o nosso muito obrigado(a).

"O que as vitórias têm de mau
é que não são definitivas.
O que as derrotas têm de bom
é que também não são definitivas."
(José Saramago)

Resumo

Devido ao cenário de notável prestígio e relevância do voleibol como esporte coletivo, no Brasil e no mundo, a adesão da prática amadora do vôlei apresenta um eminente crescimento. Desta forma, o número de times desportivos não profissionais focados no esporte aumenta de forma que soluções computacionais não têm conseguido suprir as necessidades intrínsecas do domínio. Ainda neste sentido, é somado o fato de que a sociedade busca, cada vez mais, automatizar processos repetitivos e exaustivos por meio de produtos de *software* especializados em gestão de produtividade. Neste contexto, o presente trabalho desenvolveu um sistema computacional que fornece, de forma semiautomatizada, funcionalidades que tangenciam as principais responsabilidades do gestor esportivo em uma aplicação *web*. O desenvolvimento da ferramenta baseia-se em um conjunto de técnicas pertencentes ao arcabouço da Engenharia de *Software*, que perpassam as diferentes etapas do ciclo de vida de um *software*. O sistema é de código aberto, licenciado pela *GNU General Public License v3.0*, com uma arquitetura Cliente-Servidor, estruturada em três camadas principais. O projeto orienta-se por metodologias de cunho investigativo (Levantamento Bibliográfico); de desenvolvimento (combinação adaptada de *Scrum*, *Kanban* e *Extreme Programming*), e de análise de resultados (Pesquisa-ação).

Palavras-chave: Voleibol. Gerenciamento de Times Esportivos Amadores. Engenharia de *Software*. Ciclo de Vida de *Software*. Código Aberto.

Abstract

Given the prestige and importance of volleyball as a team sport in Brazil and globally, the popularity of amateur volleyball is experiencing a significant growth. As a result, the number of non-professional volleyball teams is increasing, and existing computational solutions are struggling to keep up with the demands of the sport. Additionally, more and more people are seeking automation for repetitive and tedious processes through the use of productivity management software. With this in mind, this work developed a web application that automates the main responsibilities of a sports manager. The development of the tool will follow established software engineering techniques and progress through different stages of the software life cycle. The system will be an open-source project, licensed under the GNU General Public License V3.0, and designed using a three-tier client-server software architecture. The project is guided by investigative methodologies (Literature Review), development (an adapted combination of Scrum, Kanban, and Extreme Programming), and results analysis (Action-Research).

Key-words: Volleyball. Sports Teams Management. Software Engineering. Software Life Cycle. Open Source.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
CD	*Continuous Delivery*
CI	*Continuous Integration*
CLI	*Command Line Interface*
DE-R	Diagrama Entidade-Relacionamento
DevOps	*Development and Operations*
DLD	Diagrama Lógico de Dados
DVC	*Data Version Control*
GRASP	*General Responsibility Assignment Software Patterns*
GTD	*Getting Things Done*
GUI	*Graphical User Interface*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IDE	*Integrated Development Environment*
MVC	*Model View Controller*
MVP	*Minimum Viable Product*
REST	*Representational State Transfer*
SGBD	Sistema Gerenciador de Banco de Dados
SPA	*Single Page Application*
TCLE	Termo de Consentimento Livre Esclarecido
TCC	Trabalho de Conclusão de Curso
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TDD	*Test Driven Development*
WCAG	*Web Content Accessibility Guidelines*
WWW	*World Wide Web*

Sumário

1 Introdução

Este capítulo tem como objetivo apresentar a Contextualização desta pesquisa, na qual são abordados a importância dos esportes para a saúde e bem-estar populacional, a grande relevância do voleibol como modelo de atividade física para o Brasil contemporâneo, e, por fim, o atual cenário de times amadores. Somado a isto, tem-se ainda a Justificativa, que coloca em evidência os desafios da gestão esportiva, e apresenta como a problemática pode ser abordada a partir de técnicas da Engenharia de *Software* nos times amadores, além de enfatizar como aplicações de gestão auxiliam o dia a dia da sociedade moderna. Na sequência, são apresentadas as Questões de Pesquisa e Desenvolvimento, seguidas pelos Objetivos do presente trabalho, objetivos estes que são separados entre Objetivo Geral e Objetivos Específicos. Por fim, têm-se a Organização do Trabalho segmentada entre seus capítulos.

1.1 Contextualização

Existem numerosas evidências que afirmam que a prática de esportes é um ponto de aumento da qualidade de vida das pessoas. Organization et al. (2020) afirmam que a prática de atividades físicas é fundamental para prevenir e controlar doenças cardíacas, diabetes e câncer. Oliveira et al. (2011) afirmam que a atividade física atua na melhoria da autoestima, do autoconceito, da imagem corporal, das funções cognitivas e da socialização, assim como na diminuição do estresse e da ansiedade. Neste aspecto, o vôlei surge não só como um esporte recreativo com ampla notoriedade e reconhecimento popular (NASCIMENTO, 2017), mas também como uma modalidade que se consolidou como importante atividade econômica no Brasil, por meio de investimentos monetários em campeonatos profissionais ou amadores (NETO; OLIVEIRA, 2003).

Globalmente, o voleibol é o segundo esporte com mais praticantes, antecedido apenas pelo futebol (MEZZAROBA; PIRES, 2011). Neste cenário, então, o Brasil não é exceção e, de acordo com o Atlas Brasileiro do Esporte, esse comportamento foi observado há uma década, quando, em um *ranking* dos dez esportes mais populares nacionalmente, cerca de 15,3 milhões de pessoas praticavam o voleibol, atrás apenas dos 30,4 milhões de praticantes do futebol (DACOSTA et al., 2006).

Ainda neste sentido, devido ao número de medalhas, prêmios e reconhecimentos nacionais e internacionais, o voleibol é o esporte coletivo com o maior prestígio no Brasil. Seja como conteúdo teórico e prático de educação física, seja como atividade física recreativa, ou até mesmo como modalidade esportiva competitiva em âmbito profissional, o vôlei ocupa uma posição de extrema relevância para o Brasil (COUTINHO, 2017).

O cenário de times amadores do esporte acompanha o fenômeno da crescente popularidade e ascensão do vôlei no Brasil. O aumento no número de times amadores fez com que se tornasse inviável a manutenção do time sem uma gestão interna estruturada (BOJIKIAN; BOJIKIAN, 2008). No âmbito desportivo, o gestor é encarregado de múltiplas obrigações, que englobam aspectos gerais, mais associados ao esporte, e aspectos específicos, relacionados às particularidades de cada time. Em linhas gerais, o gestor desportivo é responsável pela adequação e pelo planejamento do time (CAPINUSSÚ, 2002).

1.2 Justificativa

Devido à natureza interdisciplinar do cargo de gestor desportivo, segundo Amaral e Bastos (2015), o profissional que atua como tal deve ser capaz de:

- Conjugar políticas;
- Definir objetivos estratégicos para o time;
- Possuir conhecimentos administrativos;
- Aplicar técnicas de *marketing*, imagem e comunicação, e
- Realizar o controle de atividades de recursos humanos.

O gestor desportivo pode, então, ser identificado por diferentes terminologias ao longo do país, como por exemplo, **presidente** ou **representante**. Além disto, o cargo pode ser assumido por diferentes perfis. O gestor pode ser um atleta do time ou então o técnico esportivo. Dentre as responsabilidades do gestor, Cárdenas e Feuerschütte (2014) elencaram os pontos descritos no Quadro 1.

Com o aumento dos diferentes aplicativos de controle de tempo e organização pessoal, é notório que cada vez mais as pessoas busquem automatizar processos repetitivos e exaustivos; manter suas tarefas e atividades em controle; e possuir ambientes dedicados para os diferentes segmentos pessoais, como o profissional e o financeiro. Para tal, existe uma gama de produtos de *software* especializados em gestão pessoal que foram projetados para atender diferentes demandas pessoais (GREGG, 2015).

Apesar da oferta de recursos, o domínio da gestão esportiva enfrenta uma escassez de produtos de *software* especializados no mercado. Em decorrência desse fato, seja pela falta de disseminação, seja pela baixa popularização, é comum soluções adaptadas e improvisadas, usando aplicações de produtividade conhecidas, tal como as aplicações *Getting Things Done* (GTD) (GREGG, 2015), em atendimento às necessidades da gestão esportiva. Não obstante, isto se torna um problema, uma vez que nem todas as especificidades do gerenciamento esportivo são atendidas em soluções de outros domínios e,

Quadro 1 – Atividades e Responsabilidades do Gestor Desportivo

Atividades e responsabilidades do Gestor Desportivo
- Responsabilidade geral pela organização esportiva e atividades relacionadas à mesma; - Manutenção e melhoria dos processos organizacionais e da estrutura física que a organização dispõe; - Gerir e lidar com pessoas; - Avaliar, corrigir e solucionar problemas; - Planejamento organizacional; - Gestão financeira; - Captação de recursos; - Gestão de marketing; - Relacionamento com o público-alvo e outros *stakeholders* da organização, e - Organização de eventos esportivos.

Fonte: (CÁRDENAS; FEUERSCHÜTTE, 2014)

como consequência, o gestor terá de fazer uso de mais de uma aplicação, ou ainda realizar trabalho manual para que as demandas sejam atendidas de maneira adequada.

Com o objetivo de atender este público alvo que possui uma demanda específica, este projeto procurou desenvolver uma aplicação de cunho gerencial para times amadores de voleibol.

1.3 Questões de Desenvolvimento

Ao final do presente trabalho, constam insumos que permitem responder os seguintes questionamentos, mitigando as inerentes preocupações associadas:

- Tendo em vista o público alvo da aplicação, como viabilizar seu desenvolvimento, considerando um *Minimum Viable Product* (MVP)?
- Quais boas práticas da Engenharia de *Software* podem ser aplicadas para o desenvolvimento do projeto, e como é possível integrá-las?

1.4 Objetivos

Tem-se como o Objetivo Geral o **Desenvolvimento de uma aplicação de gerenciamento de times amadores de voleibol**. A partir do Objetivo Geral, pôde-se definir os Objetivos Específicos, sendo:

- Uso de técnicas de elicitação e modelagem, visando elicitar e especificar os principais requisitos do MVP;

- Realização de validação e priorização dos requisitos elicitados e modelados;
- Modelagem de dados, uma vez que há necessidade de persistência de dados na solução;
- Aplicação de boas práticas da Engenharia de *Software* na solução, tal como o uso de um Ciclo de Vida de *Software* apropriado, Práticas Ágeis, Padrões Arquiteturais e de Projeto;
- Documentação de Referencial Teórico, Referencial Tecnológico e aspectos metodológicos, orientando-se pelos estudos realizados no cumprimento dos Objetivos Específicos anteriores;
- Viabilização de uma solução *Open Source*, e
- Apresentação dos resultados obtidos usando uma abordagem, preferencialmente, qualitativa.

1.5 Organização do Trabalho

Esta monografia está subdividida nos seguintes capítulos:

- Capítulo 2 - Referencial Teórico: mapeia os principais fundamentos teóricos utilizados para a confecção do trabalho, em especial os conceitos acerca da Engenharia de *Software* e as disciplinas que a compõem como área de conhecimento;
- Capítulo 3 - Suporte Tecnológico: define o aparato ferramental de tecnologias que foi utilizado durante o desenvolvimento do *software* e a gerência de configuração do trabalho;
- Capítulo 4 - Metodologia: expõe os aspectos metodológicos quanto às diferentes características da elaboração do trabalho, como o levantamento bibliográfico, o desenvolvimento de *software*, a análise dos resultados, e apresenta o cronograma de atividades;
- Capítulo 5 - Setter: diz respeito ao produto de *sotfware* Setter. Delibera nuances do produto ao evidenciar detalhes de desenvolvimento da aplicação *web* quanto ao contexto, às funcionalidades da aplicação, à organização arquitetural do sistema, à modelagem da aplicação e, por fim, à identidade visual;
- Capítulo 6 - Análise de Resultados: apresenta os resultados obtidos com o trabalho desenvolvido, e

- Conclusão: Compreende as principais considerações finais acerca do trabalho realizado, com destaque para os *status* das Questões de Pesquisa e Objetivos Concluídos, além de propostas para trabalhos futuros.

2 Referencial Teórico

Neste capítulo, são apresentados os principais referenciais que embasam conceitualmente o presente trabalho. É preciso lembrar que o principal objetivo do trabalho é o desenvolvimento de uma aplicação para um público alvo específico, portanto, há a necessidade de compreender sobre o Ciclo de Vida de um *Software*, uma vez que, ao longo do desenvolvimento da aplicação, várias etapas de um projeto de desenvolvimento de *software* foram contempladas, sendo estas:

- Engenharia de Requisitos;
- Projeto de Arquitetura de *Software*;
- Padrões de Projeto de *Software*;
- Implementação de *Software*;
- Testes de *Software*;
- Implantação de *Software*, e
- Manutenção de *Software*.

Adicionalmente, o trabalho demandou conhecer o público alvo, fazendo-se necessários os processos de elicitar, modelar e analisar os requisitos de *software*. Uma vez que a *baseline* de requisitos começou a ser conhecida, a solução computacional orientou-se não apenas por ela, mas também por boas práticas da Engenharia de *Software* como um todo, justificando estudos sobre Arquitetura de *Software*, Padrões de Projeto, Implementação, Testes, Implantação e Manutenção.

Em Arquitetura, o foco do texto encontra-se na definição conceitual do domínio e, em adição, na apresentação dos padrões de N-Camadas, MVC, Arquitetura Limpa, REST e Cliente-Servidor (SOMMERVILLE, 2016). Na sequência, é dedicada uma seção aos Padrões de Projeto de *software*, a qual confere uma revisão sobre os GRASP's Coesão e Acoplamento (DUNCAN, 2012) e os Princípios SOLID (MARTIN, 2019). A seção seguinte descreve os detalhes de Implementação relacionados às metodologias ágeis utilizadas no projeto, com destaque ao *Scrum* (SUTHERLAND, 2010), ao *XP* e ao *Kanban* (POPPENDIECK; POPPENDIECK, 2003). Por fim, são apresentadas colocações, de cunho geral, sobre Testes, Implantação e Manutenção de *Software*, que não foram etapas amplamente exploradas neste trabalho, mas que fazem parte do Ciclo de Vida de um *Software*.

2.1 Ciclos de Vida de *Software*

Ciclo de vida de *software* é o termo utilizado para definir o conjunto de etapas que ocorrem entre a concepção de um sistema até o instante em que ele é mantido ou então descontinuado pela equipe de desenvolvimento responsável. Tais etapas são realizadas por meio de processos bem definidos que podem ser descritos via modelos conceituais, que por sua vez podem ser utilizados de acordo com objetivos específicos.

Os modelos conceituais não são normas definitivas sobre como devem ser realizados os processos de *software*, mas sim abstrações e orientações gerais que podem ser adaptadas e personalizadas para abranger diferentes abordagens do desenvolvimento de *software* (SOMMERVILLE, 2016). Dentre os diversos modelos evidenciados pela literatura utilizada na pesquisa do presente trabalho, destacam-se os modelos Cascata, Espiral e, em especial, o modelo Iterativo-Incremental.

2.1.1 Modelo em Cascata

O modelo **Cascata** foi derivado de processos mais gerais da engenharia de sistemas, e ficou reconhecido por ser o primeiro modelo a ser publicado para a comunidade de desenvolvimento de *software* (SOMMERVILLE, 2016). O modelo é caracterizado pelo encadeamento sequencial entre suas fases descritas, isto é, trata-se de um modelo dirigido a planos que se faz necessária a aplicação inicial de uma robusta etapa de planejamento e programação de todas as atividades subsequentes do projeto (SOMMERVILLE, 2016).

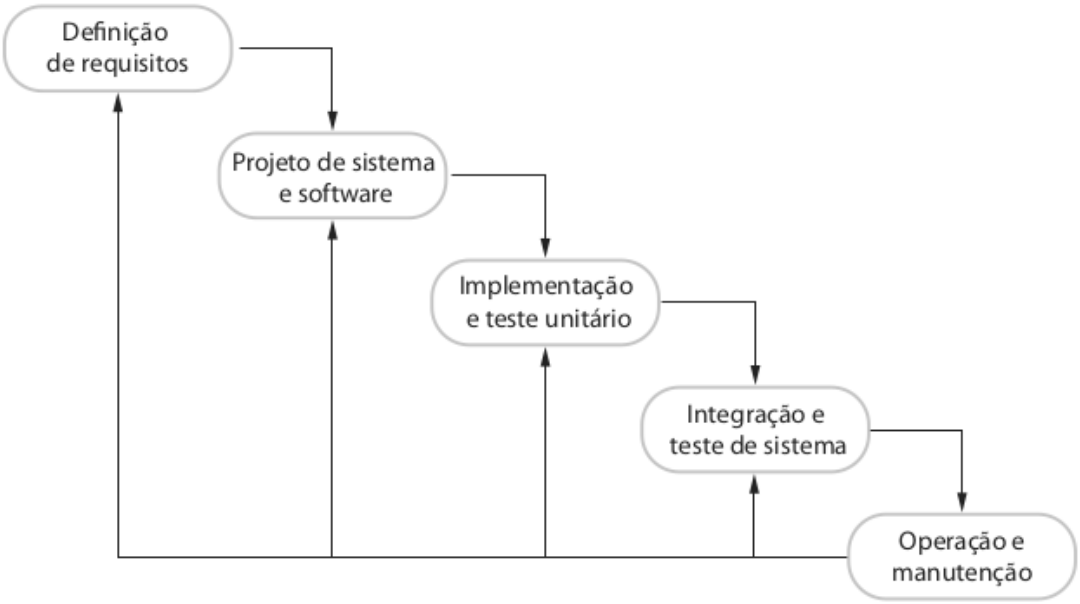
As etapas ocorrem em sequência, sendo assim, o modelo foi nomeado como Cascata, visto a notável dependência entre fases estruturadas em uma série lógica e bem definida. As fases do modelo e suas dependências diretas são ilustradas pela Figura 1.

2.1.2 Modelo Espiral

O modelo **Espiral** foi proposto em 1998 por Boehm (1988), e é caracterizado por representar o processo de *software* como fluxo de espiral, e não como uma sequência de atividades com início e fim definidos. Ele combina prevenção e tolerância a mudanças ao assumir que mudanças em si são um resultado de risco do projeto e, por isso, é necessária a inclusão de atividades explícitas de gerenciamento de riscos para sua respectiva redução e mitigação. Tal preocupação evidenciada pelo modelo é tida como um diferencial, já que o mesmo adota uma visão que posiciona os riscos como um elemento explícito do projeto que deve ser devidamente tratado (SOMMERVILLE, 2016).

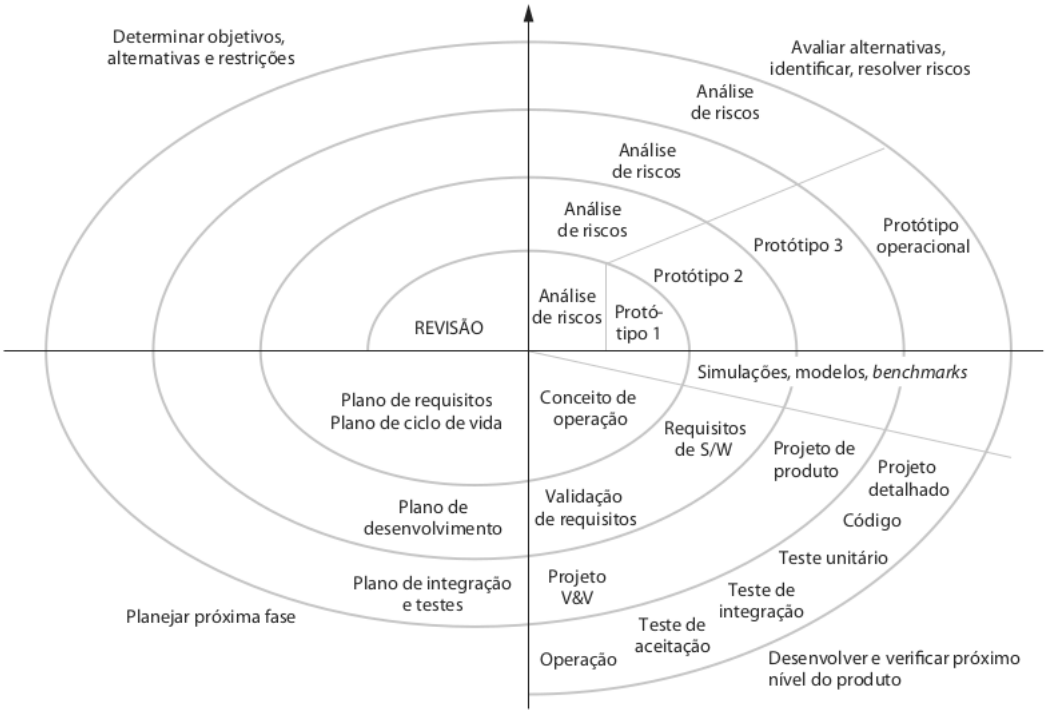
O modelo é representado graficamente pela Figura 2. Ao analisar a imagem, é possível visualizar um fluxo de atividades que evolui de forma gradual ao longo de uma espiral. Neste modelo, cada volta na espiral representa uma fase do processo de *software*,

Figura 1 – Etapas do Modelo Cascata



Fonte: (SOMMERVILLE, 2016)

Figura 2 – Etapas do Modelo Espiral



Fonte: (SOMMERVILLE, 2016)

de forma que a volta mais interna abrange responsabilidades acerca da viabilidade do sistema; em sequência, na próxima volta, são definidos os requisitos do projeto; mais adiante o projeto de implementação é abordado e, ao seguir esta sequência lógica, cada volta define uma etapa do desenvolvimento (SOMMERVILLE, 2016).

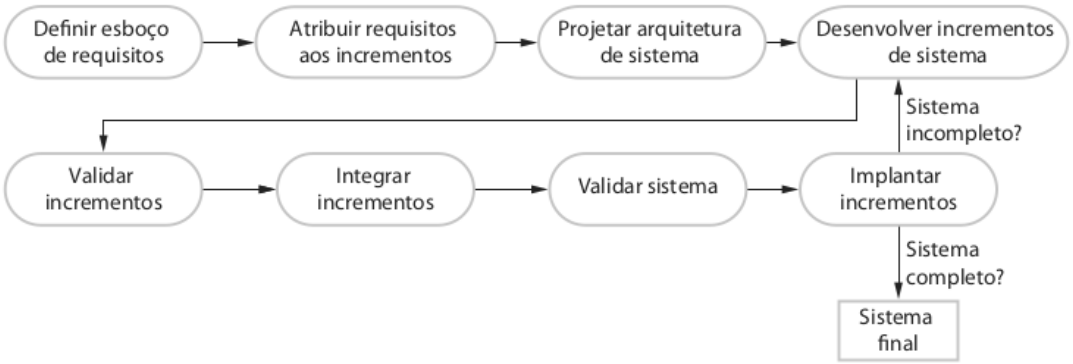
2.1.3 Modelo Iterativo-Incremental

O modelo iterativo incremental é uma abordagem para o desenvolvimento de *software*, na qual iterações de processos são realizados, e incrementos desenvolvidos são entregues ao cliente e implantados para uso em um ambiente operacional de forma contínua (SOMMERVILLE, 2016).

O incremento ocorre em pequenas entregas de *software* por iteração. A característica notável do modelo dá-se pela frequência a nível de iterações em preceitos como o avanço do conhecimento do projeto, novas elicitações de requisitos e, por fim, revisão da arquitetura. Este modelo possibilita aos desenvolvedores e *stakeholders* a capacidade de decisão conjunta para cada etapa do processo, o que fornece um maior apoio à prevenção de erros e aumento da qualidade do *software*.

Em um processo de entrega iterativo-incremental, os clientes identificam e priorizam os serviços que o produto deve possuir. Feita esta priorização, é possível entregar, continuamente, um subconjunto de funcionalidades, isto é, o incremento da iteração realizada. O fluxo do processo é descrito na Figura 3.

Figura 3 – Fluxo do Processo do Modelo Iterativo Incremental



Fonte: (SOMMERVILLE, 2016)

As etapas deste modelo podem ser definidas como: Etapa de Requisitos, Projeto de Arquitetura, Implementação, Teste, Implantação e, por fim, Manutenção.

1. **Etapa de Requisitos**: Diz respeito à toda Engenharia de Requisitos aplicada, conhecimento que será abordado na próxima seção.

2. **Projeto de Arquitetura**: Diz respeito à decisão de como o sistema será implementado a nível arquitetural, ou seja, nesta etapa, são definidos a arquitetura do sistema, os estilos e os padrões que serão adotados. O Projeto de Arquitetura é abordado na seção de Projeto de Arquitetura de *Software*.

3. **Implementação**: Diz respeito ao desenvolvimento das funcionalidades do sistema, respeitando as decisões arquiteturais e os requisitos elicitados.

4. **Teste**: Diz respeito aos testes realizados para garantir a qualidade do produto, além daqueles feitos com os *stakeholders* como forma de validação do que foi produzido.

5. **Manutenção**: Sendo a última fase do modelo, diz respeito ao processo de correções do *software* desenvolvido já em produção. Essa manutenção pode ser evolutiva, trazendo melhorias pontuais ao produto, como por exemplo, melhorias na usabilidade e na experiência do usuário.

A utilização deste modelo traz vantagens como:

- Os incrementos podem ser usados como protótipos para os *stakeholders*, que podem validar os requisitos e levantar *feedbacks*;
- Os *stakeholders* não precisam aguardar o desenvolvimento total do produto para obter ganhos de valor, já que, a partir da priorização realizada, os primeiros incrementos já são capazes de satisfazer os requisitos mais críticos do produto, e
- O processo permite a incorporação de mudanças no sistema, adaptadas aos pedidos de *stakeholders*.

Não obstante, o maior risco deste modelo concentra-se no atraso do prazo da entrega final do produto. Produtos que possuem muitas iterações correm o risco de aumentar o escopo definido previamente, ou até mesmo de mudar a ideia original do produto, demandando mais tempo de desenvolvimento e, consequentemente, pode acarretar no deficit da entrega dentro do prazo estipulado (BRAATZ; OLIVEIRA; ROCHA, 2018).

2.2 Engenharia de Requisitos

Para o desenvolvimento de um produto de *software*, é imprescindível conhecer e entender a fundo as necessidades do usuário final. Tal domínio está relacionado às funcionalidades pretendidas pelo usuário; a como este pode obter acesso à informação relevante produzida, e, por fim, à satisfação de uso que o mesmo terá ao ter contato com o produto desenvolvido, *i.e.*, aos seus requisitos. Sommerville (2016) define os requisitos como descrições formais do funcionamento de um sistema, seus serviços oferecidos e suas restrições de funcionamento.

Os **Requisitos de *Software*** refletem as necessidades dos clientes para um sistema que serve a finalidades específicas. Estes podem ser categorizados como **Requisitos Funcionais** e **Requisitos Não Funcionais**. Os Requisitos Funcionais são declarações de serviços que o sistema deve fornecer, *i.e.*, como o sistema deve reagir a entradas específicas e quais comportamentos o sistema deve apresentar em determinadas condições e estados.

Já os **Requisitos Não Funcionais** são restrições aos serviços ou funções oferecidas pelo sistema (SOMMERVILLE, 2016).

Para o entendimento de requisitos, faz-se necessária uma bagagem de processos de projeto, conhecida por **Engenharia de Requisitos**, em que as exigências do produto são coletadas, analisadas, documentadas, mapeadas e geridas para garantir o atendimento das expectativas do usuário final e da qualidade idealizada para todo o ciclo de vida do produto.

A **Engenharia de Requisitos** é definida como um processo de compreensão e definição dos serviços requisitados ao sistema, além da identificação de restrições relativas à operação e ao desenvolvimento deste (SOMMERVILLE, 2016). A Engenharia de Requisitos é um estágio particularmente crítico do processo de *software*, já que erros nesta fase invariavelmente acarretam em problemas no projeto e na implementação do sistema. Alguns autores afirmam tal criticidade ao apontar que a qualidade do produto final depende intrinsecamente da qualidade do levantamento de requisitos (FERGUSON; LAMI, 2006).

O processo de Engenharia de Requisitos é dividido por fases, que por sua vez variam de nomes e divisões entre diferentes autores. Tendo isto em vista, Jarke e Pohl (1994) destacam, como ciclo clássico, três fases: Elicitação, Modelagem e Análise.

2.2.1 Elicitação

Definido como o coração do desenvolvimento de requisitos, a **Elicitação** é o processo de identificação das necessidades e restrições das partes interessadas no sistema de *software* a ser desenvolvido (WIEGERS; BEATTY, 2013). Nesta fase, os engenheiros de *software* trabalham em contato com clientes e usuários finais do sistema para obter informações sobre o domínio da aplicação, os serviços que o sistema deve oferecer, o desempenho do sistema e as restrições de *hardware* (SOMMERVILLE, 2016).

Este processo é realizado por meio de **técnicas de Elicitação** que são utilizadas como forma de coleta de requisitos e que, posteriormente, podem funcionar como meios de validação dos requisitos em conjunto com os *stakeholders*. Os *stakeholders* incluem os usuários finais que hão de interagir com o sistema, ou então quaisquer outros indivíduos em organizações que serão afetadas pelo sistema (SOMMERVILLE, 2016). Essas técnicas podem ser usadas em conjunto para alcançar um objetivo específico, já que uma técnica é capaz de produzir insumos de entrada para outra técnica de elicitação com um maior nível de complexidade e robustez. Alflen e Prado (2021) elencam, por meio da publicação de um estudo, as dez técnicas mais citadas na literatura em projetos de desenvolvimento de *software*. Dentre elas, as técnicas descritas no Quadro 2 serão primordiais para o projeto.

Indo além das técnicas mencionadas no Quadro 2, e apesar de não ser umas das dez

Quadro 2 – Técnicas de Elicitação Mais Citadas na Literatura Utilizada no Desenvolvimento do Projeto

Técnica	**Descrição**	**Referência**
Entrevista	Técnica que necessita de um representante da equipe para realizar perguntas a um *stakeholder*. É a mais comum das técnicas de elicitação.	(ELMONIEM; NASR; GHEITH, 2017)
Questionário	É uma ferramenta simples, geralmente aplicada na fase inicial da Elicitação para coletar o máximo de requisitos de diferentes *stakeholders* que podem estar em lugares distintos.	(ELMONIEM; NASR; GHEITH, 2017)
Brainstorming	É uma reunião informacional, na qual cada participante pode expressar livremente os requisitos do sistema.	(YOUNAS et al., 2017)
Feedback	É o retorno dado pelos *stakeholders*. Propicia aos participantes a sensação de que suas ideias são importantes, entretanto, esta técnica pode levar à divergência de opiniões.	(HOSSEINI et al., 2015)

Fonte: Autores

técnicas mais utilizadas, a **Instrospecção** é uma técnica bem conhecida e amplamente utilizada, também merecendo menção neste trabalho. Ela exige que os participantes visualizem individualmente os requisitos do sistema em seu próprio pensamento. Esta técnica é mais bem utilizada quando o participante está muito familiarizado com o domínio cognitivo do sistema a ser desenvolvido (OKESOLA et al., 2018).

2.2.2 Modelagem

Com os requisitos elicitados, é preciso fornecê-los em uma estrutura bem definida para dar forma, processo este realizado durante a etapa de **Modelagem**. A Modelagem é o processo de desenvolvimento de modelos abstratos de um sistema, em que cada artefato produzido apresenta uma visão ou perspectiva diferente do sistema (SOMMERVILLE, 2016).

Os modelos são usados para auxiliar, extrair e priorizar os requisitos. Além do exposto, os modelos são utilizados para descrever o sistema que os engenheiros de *software* implementarão. Posteriormente, ainda é possível utilizar os artefatos para documentar a estrutura e a operação do sistema. Muitos modelos são, por padrão, baseados na notação *Unified Modeling Language* (UML), que podem ser categorizados em estruturais ou comportamentais. Não obstante, outros artefatos podem ser desenvolvidos durante esta fase a fim de auxiliar a modelagem dos dados.

Um importante artefato para a Modelagem é o *Backlog* de Produto, que é uma lista

organizada em sequência prioritária que concentra e descreve os itens de trabalho necessários ao desenvolvimento do sistema. Normalmente, um *Backlog* de produto é estratificado em histórias de usuário, que por sua vez são representações dos requisitos de *software* examinados pela ótica do usuário final, ou seja, são requisitos escritos em frases diretas em um alto nível de abstração de comunicação. A definição do *Product Backlog* abrange criação, refinamento, estimativa e priorização dos seus itens com base nos requisitos do usuário (SEDANO; RALPH; PÉRAIRE, 2019).

Outro artefato evolutivo produzido nesta fase é o protótipo. A prototipação possui o intuito de explorar aspectos centrais do sistema proposto por meio da implementação de funcionalidades a nível visual. Desta forma, possibilitando a interpretação do entendimento das funcionalidades sob a visão do usuário. Com a prototipação, é possível modelar os requisitos elicitados, considerando o resultado da navegabilidade do usuário (BOURQUE; FAIRLEY; SOCIETY, 2014).

2.2.3 Análise

Após a modelagem dos dados, é necessário realizar a **Análise** dos artefatos com o intuito final de validação com os *stakeholders*. Para isso, são necessárias que algumas etapas sejam devidamente cumpridas, como é o caso da priorização dos requisitos, da verificação e da validação dos mesmos.

A priorização dos requisitos consiste em uma atividade que busca definir a ordem prioritária dos requisitos, ou conjuntos destes, a serem executados ao longo do tempo de desenvolvimento do sistema. A sequência é ordenada a partir de prioridade ou importância em relação aos pontos de vista das partes interessadas (SOMMERVILLE, 2016). A priorização ocorre por meio de técnicas conceituais estruturadas, como por exemplo a técnica *MoSCoW*. Tal técnica possui um grande potencial de auxílio durante as tomadas de decisões. A priorização é um aspecto primordial no desenvolvimento de *software*, visto que a identificação e o mapeamento de quais requisitos geram mais valor ao cliente são processos fundamentais para o sucesso do sistema, e a decisão errônea acerca da ordenação destes requisitos pode afetar a qualidade global do sistema de maneira generalizada, e subsequentemente a aceitação por parte dos clientes.

O acrônimo *MoSCoW* foi proposto inicialmente por Clegg e Barker (1994), que definiram uma classificação de priorização dos requisitos a partir de níveis de valor: em *Must Have*, em *Should Have*, em *Could Have* e, por fim, em *Won't Have* (MIRANDA, 2022). Os requisitos classificados como *Must Have* são os prioritários, sendo os primeiros a serem desenvolvidos. Os classificados como *Should Have* são desenvolvidos em segundo lugar e, na sequência, são desenvolvidos os requisitos classificados como *Could Have*. Já os requisitos classificados como *Won't Have*, como já implica a tradução, são despriorizados, e consequentemente são retirados do escopo do projeto.

A verificação e a validação dos requisitos são estágios da análise que visam responder algumas questões. A verificação analisa se todo o processo de desenvolvimento foi realizado de acordo com o plano de garantia do produto (PAIVA; MACIEL; SILVA, 2020). Boehm (1984) define a **Verificação** como as atividades que respondem a pergunta:

"O sistema foi desenvolvido corretamente?".

Já a validação concentra-se em comprovar que o comportamento do sistema está correto, tanto em seu aspecto funcional quanto não funcional. Ela pode ser realizada por meio de um protótipo de alta fidelidade junto com os *stakeholders* (PAIVA; MACIEL; SILVA, 2020). O protótipo de alta fidelidade possui um nível de fidelidade com o produto final muito avançado, e permite que os *stakeholders* possam validar o produto a nível de *design* e a nível de usabilidade do produto, por permitir a interação do *stakeholder* com o protótipo, realizando testes dos fluxos da informação do produto a ser desenvolvido. Por fim, Boehm (1984) aponta que a **Validação** concentra as atividades que respondem a pergunta:

"Foi desenvolvido o sistema certo?"

2.2.4 Gerenciamento de Requisitos

Todas as etapas apresentadas anteriormente dizem respeito ao processo da Engenharia de Requisitos para alinhamento dos requisitos com os *stakeholders*, porém, requisitos para sistemas de grande porte tendem a mudar constantemente, pois o entendimento dos *stakeholders* a respeito do problema a ser resolvido está em constante mutação (SOMMERVILLE, 2016).

Para essa situação, é necessário possuir um gerenciamento de requisitos, que é o processo de compreensão e controle das mudanças dos requisitos do sistema (SOMMERVILLE, 2016). Ele fica responsável em formalizar a manutenção dos requisitos, perpassando por fases, como:

1. Análise de problema e especificação de mudanças;
2. Análise de mudanças e custos, e
3. Implementação de mudanças.

Porém, alguns métodos ágeis possuem uma forma diferente de lidar com o gerenciamento dos requisitos. O *Extreme Programming* (XP), por exemplo, não necessita de um processo formal, e lida com a priorização das mudanças. Em caso de alta prioridade, já haverá a tomada de decisão de quais serão os recursos necessários, e quais deverão

ser os recursos abandonados, na próxima iteração realizada (SOMMERVILLE, 2016). Os métodos ágeis serão mais bem explanados na seção 2.5.1.

2.3 Projeto de Arquitetura de *Software*

A **Arquitetura de *Software*** é um domínio da Engenharia de *Software* que busca estabelecer a conexão lógica entre a modelagem conceitual de um sistema computacional e sua implementação em código. Sommerville (2016) define que o projeto de arquitetura é um estágio do processo de projeto de *software* que se materializa como o vínculo fundamental entre o projeto e a Engenharia de Requisitos. Sendo assim, a Arquitetura de *Software* é capaz de identificar os principais componentes estruturais de um sistema e suas respectivas relações internas. Em síntese, a Arquitetura de *Software* visa compreender como um sistema computacional deve ser organizado e estruturado para então atender à *baseline* de requisitos previamente concebida.

Ainda sobre Arquitetura, esta disciplina inclui as decisões de projeto mais importantes em um sistema, isto é, uma vez tomadas, dificilmente poderão ser revertidas no futuro, já que a refatoração a nível estrutural é extremamente custosa. Isto ocorre, pois, provavelmente, a maior parte dos módulos deve ter de ser refatorada para acompanhar as mudanças arquiteturais (SOMMERVILLE, 2016). Portanto, é inviável resumir a Arquitetura como um agrupamento lógico de módulos, devendo-se levar em consideração que a disciplina inclui também um conjunto de decisões estruturais e, ao final do processo, é gerado um modelo conceitual capaz de descrever como o sistema está organizado em módulos e como estes se comunicam (VALENTE, 2020).

Conceitualmente, Sommerville (2016) define duas principais escalas de abstrações para descrições arquiteturais: a *arquitetura em pequena escala* e a *arquitetura em grande escala*. A arquitetura em pequena escala abrange a organização de um programa individual, isto é, como este se organiza em componentes e módulos internos. Já a arquitetura em grande escala busca descrever o relacionamento arquitetural entre diferentes programas de um sistema maior. Os padrões descritos neste capítulo são focados na visualização de arquitetura em pequena escala.

Antes de pontuar individualmente cada um dos padrões do capítulo, é necessário fornecer uma visão clara sobre o conceito em si de padrões arquiteturais. Um padrão arquitetural consiste em uma descrição formal de uma organização pré-definida para sistemas (GARLAN, 2000). A partir da definição concebida, os padrões de Arquitetura N-Camadas, MVC, Arquitetura Limpa, REST e Arquitetura Cliente-Servidor serão debatidos ao longo das próximas subseções.

2.3.1 Arquitetura N-Camadas

O padrão arquitetural em N-Camadas é um dos padrões arquiteturais mais bem consolidados e amplamente adotados para sistemas, pioneiramente estruturados pelos primeiros produtos de *software* de grande porte que foram desenvolvidos durante as décadas de 1960 e 1970 (VALENTE, 2020). Para a compreensão deste padrão, os conceitos de separação e de independência entre módulos são fundamentais, uma vez que as estruturas lógicas do contexto do sistema são organizadas em camadas que possuem responsabilidades e atribuições bem definidas e segregadas (SOMMERVILLE, 2016).

Sommerville (2016) defende que o estilo arquitetural em camadas promove uma propensão natural ao desenvolvimento no modelo Iterativo Incremental, previamente discutido nesta monografia. Tal afirmação dá-se pelo fato de que, ao seguir as restrições arquiteturais impostas pelo modelo, as funcionalidades do sistema podem ser entregues de maneira contínua: à medida que novas camadas são desenvolvidas, novos serviços podem ser disponibilizados aos usuários finais.

A arquitetura em N-Camadas particiona a complexidade do desenvolvimento em componentes menores que possuem fortes restrições de comunicação. As camadas estão restritas a criar ou estabelecer dependências apenas com camadas adjacentes, de forma que a comunicação deva, obrigatoriamente, obedecer definições hierárquicas de atribuição. Este rigor de comunicação auxilia o entendimento, a manutenção e a evolução de um sistema, de modo que a adaptabalidade e a reutilização de camadas se tornam possíveis (VALENTE, 2020). A Figura 4 fornece a visualização gráfica da estrutura lógica das camadas e restrições de comunicação entre cada uma delas.

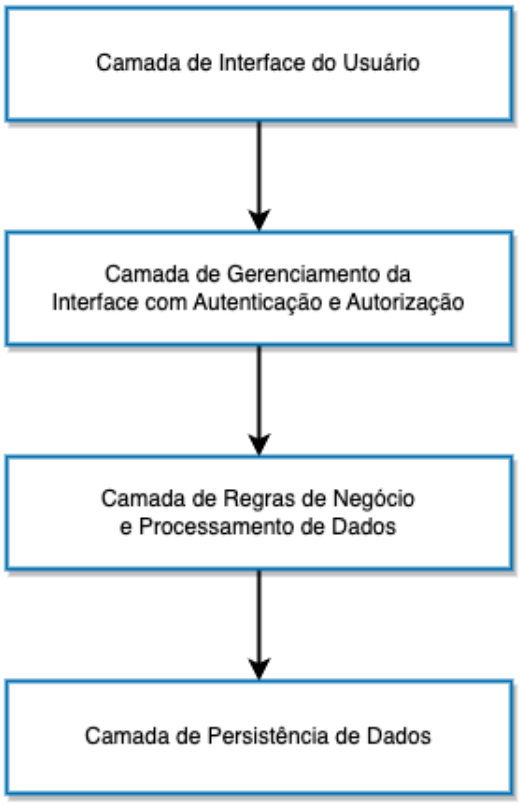
Vale ressaltar que, de acordo com o estilo arquitetural, camadas em níveis inferiores não podem criar dependências com camadas de níveis superiores. Para que a comunicação ocorra no sentido contrário à orientação hierárquica da arquitetura, é necessária a utilização de padrões de projeto e outros estilos arquiteturais, como é o caso do padrão *Observer* (SHVETS, 2018), por meio de notificações, e outros recursos provenientes do Arquitetura Orientada a Eventos (SOMMERVILLE, 2016).

2.3.2 Arquitetura MVC

O padrão de arquitetura *Model-View-Controller* (MVC) foi projetado inicialmente ao final da década de setenta por projetistas de *Smalltalk*[1], uma das linguagens de programação pioneiras do paradigma de Orientação a Objetos, a fim de substanciar a ideia de **separação de conceitos** (DEACON, 2009). O modelo MVC propõe uma organização interna, em que a apresentação, os dados e as regras de negócio de um determinado sistema são tratados de maneira segregada (SOMMERVILLE, 2016).

[1] Disponível em <http://www.smalltalk.org/>. Acesso em: 02 jan. 2023.

Figura 4 – Exemplificação da Arquitetura em N-Camadas



Fonte: Adaptação de (SOMMERVILLE, 2016)

Este padrão arquitetural defende que a arquitetura de um sistema deve ser decomposta em três principais módulos lógicos: o módulo de **Modelos**, o módulo de **Visão** e o módulo de **Controladores**. Cada um destes módulos possui responsabilidades específicas, as quais são ilustradas na Figura 5 e discutidas a seguir.

O módulo de **Modelos** é responsável por incorporar a essência imutável do sistema: as entidades. Isto é, a nível de implementação, esta camada trata-se das classes responsáveis por modelar as estruturas de dados que modelam as entidades do domínio (DEACON, 2009).

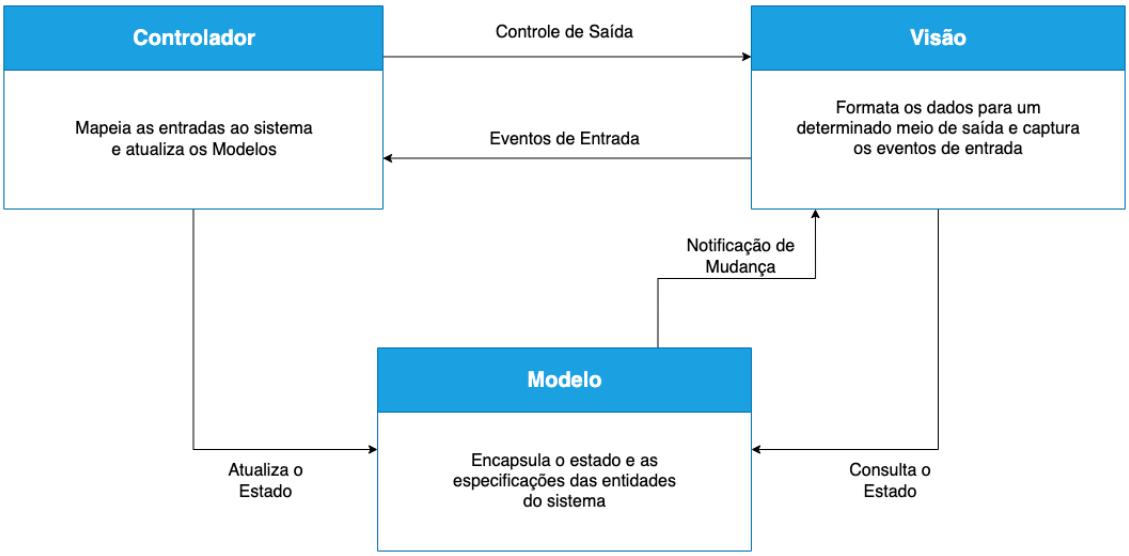
Ainda sobre o módulo de **Modelos**, vale ressaltar que as especificações das entidades do sistema dificilmente sofrem mutações ao longo do tempo. Portanto, a camada de Modelos deve ficar em níveis mais internos do *software*, de modo que os modelos não conheçam ou possuam dependências com quaisquer implementações externas (VALENTE, 2020).

Já o módulo de **Visão** é responsável por efetuar a disponibilização dos dados, estes armazenados na camada de Modelo, ao usuário. Comumente, o módulo de visão é identificado como a interface gráfica que deve ser interativa ao usuário. Entretanto, esta concepção é errônea, já que o módulo de Visão pode apresentar ao usuário os dados de

diferentes maneiras: através de uma *CLI* (Interface de Linha de Comando), de uma *API* (Interface de Programa de Aplicação) ou até mesmo por meio de uma *GUI* (Interface Gráfica do Utilizador) (DEACON, 2009).

Por fim, o módulo de **Controladores** interage com o módulo de Visão para interpretar e manipular os eventos gerados nos dispositivos de entrada ao sistema. O módulo de controladores é responsável por realizar modificações nas entidades do sistema, e então preparar o fluxo de dados para ser apresentado ao módulo de Visão (VALENTE, 2020).

Figura 5 – Representação Gráfica do Padrão Arquitetural MVC



Fonte: Adaptação de (SOMMERVILLE, 2016)

2.3.3 Arquitetura Limpa

O padrão arquitetural foi proposto, inicialmente, por Martin (2019), e reúne algumas das principais práticas de Projeto de Arquitetura de *Software*. O objetivo do padrão é impulsionar a criação de produtos de *softwares* bem organizados e estruturados a nível de contexto, de forma que a escalabilidade, a modificação e o entendimento do projeto sejam facilitados.

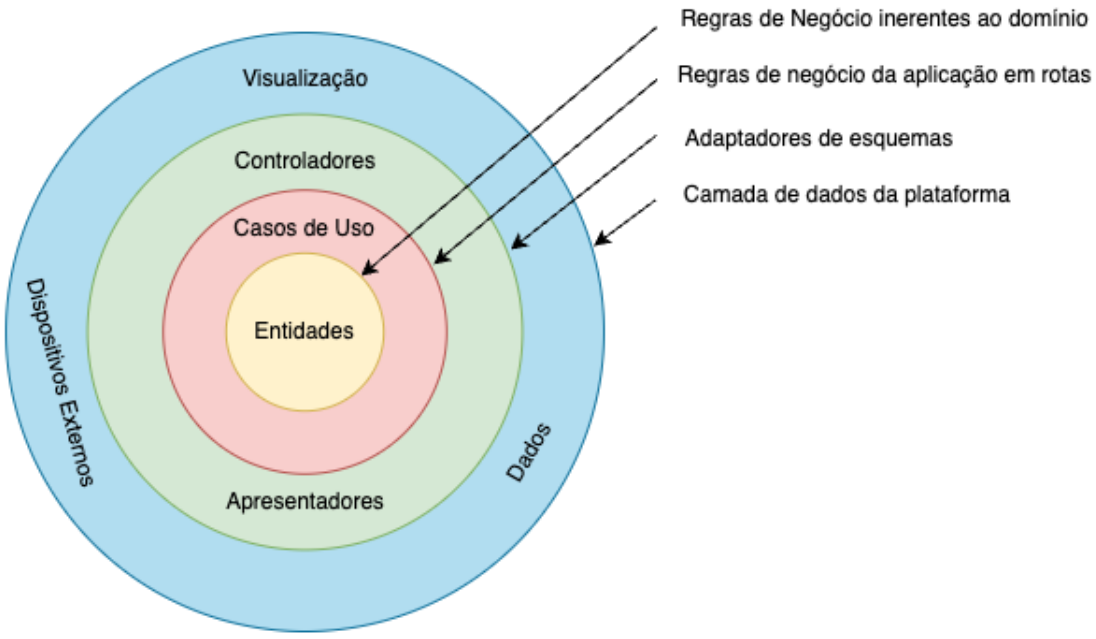
A Arquitetura Limpa fundamenta-se na divisão do projeto de *software* por camadas, conforme é ilustrado na Figura 6. Tal estilo segrega suas camadas de acordo com responsabilidades organizacionais. Valente (2020) aponta estas camadas como:

- Camada de Domínio: Diz respeito à camada mais interna, na qual são reunidas e modeladas as entidades que compõem a natureza do domínio cognitivo do *software*. Em via de regra, esta camada é a que menos sofre alterações ao longo do projeto,

uma vez que, em um ciclo de vida bem estruturado, alterações nas regras de negócio são mínimas;

- Camada de Aplicação: Diz respeito à camada responsável por armazenar as regras de negócio que compõem o sistema. Em suma, a camada da aplicação reúne os componentes necessários para compor a lógica funcional da aplicação;
- Camada de Comunicação: Diz respeito à camada responsável por mediar a interação entre as camadas internas e a camada externa do modelo, e
- Camada Externa: Por fim, tem-se a camada mais externa do padrão. Esta, por sua vez, é responsável por implementar os componentes lógicos necessários para a comunicação com dispositivos externos.

Figura 6 – Representação Gráfica do Padrão Arquitetura Limpa



Fonte: Adaptação de (MARTIN, 2019)

Além da organização por camadas, a Arquitetura Limpa propõe a **Regra da Dependência**. Isto é, as dependências de código devem apenas apontar para a camada imediatamente adjacente da qual a mesma se encontra. Desta forma, o modelo viabiliza o desenvolvimento de um *software* com o mínimo de acoplamento e com a capacidade de escalabilidade potencializada (MARTIN, 2019).

2.3.4 Arquitetura Cliente-Servidor

O padrão cliente-servidor é organizado como um conjunto de serviços e servidores associados, e de clientes que acessam e usam esses serviços via uma rede de computadores distribuídos e conectados por meio de protocolos de *Internet*. Sommerville (2016) defende que os principais componentes deste modelo são:

1. Conjunto de servidores que oferecem serviços, isto é, componentes de *software*, para outros componentes;
2. Conjunto de clientes que demandam os serviços oferecidos pelos servidores;
3. Uma rede que permite a comunicação entre clientes e servidores.

Em uma arquitetura cliente-servidor, o sistema é apresentado como um conjunto de serviços. Cada um destes serviços é fornecido por meio de um servidor apartado. Já os clientes são caracterizados por usuários desses serviços que usam os servidores para obter acesso. A Figura 7 apresenta uma organização em tempo de execução frequentemente utilizada em sistemas distribuídos que respeitam o modelo cliente-servidor.

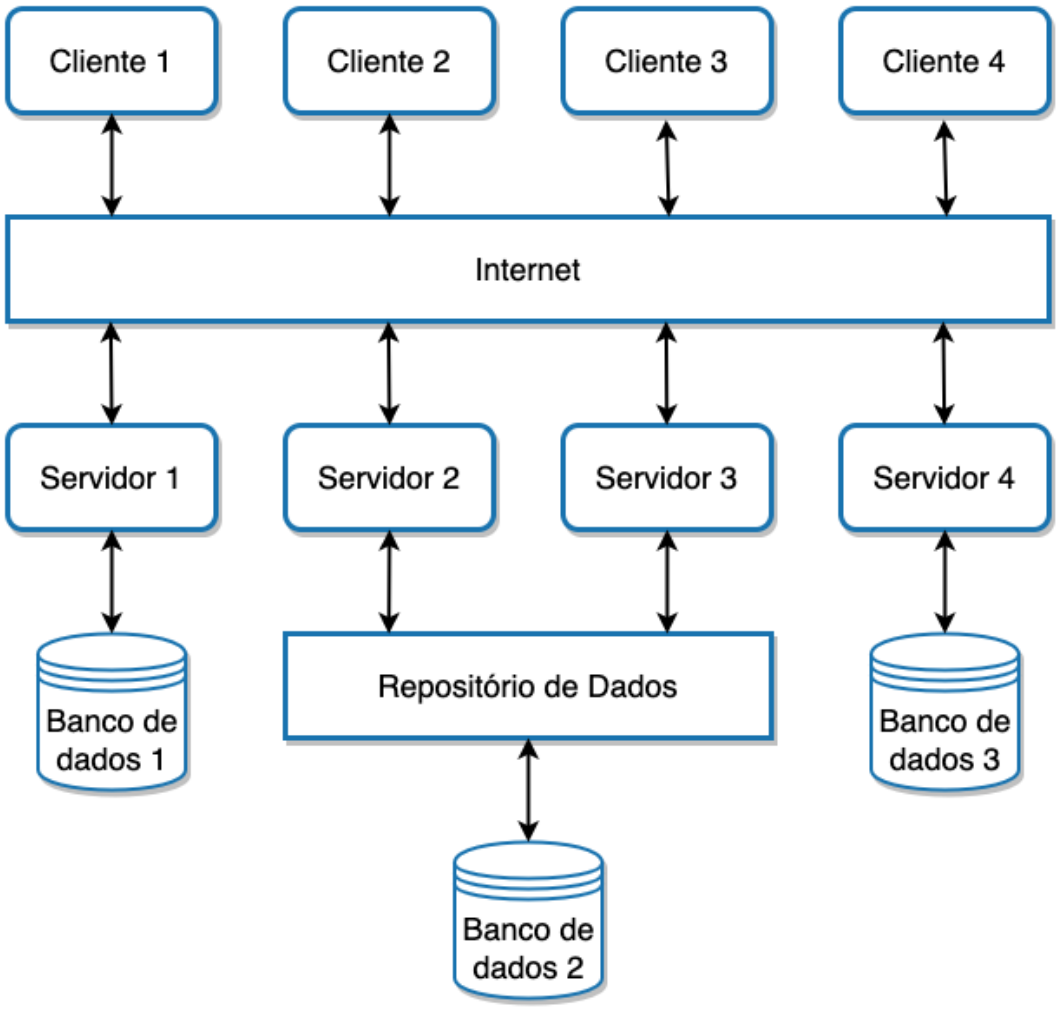
Sistemas computacionais geralmente implementam uma arquitetura Cliente-Servidor quando os recursos em um banco de dados são compartilhados e precisam ser acessados por diferentes origens, as quais podem estar ou não em diferentes locais. Devido à replicabilidade apresentada pelos componentes servidores, estes também podem ser utilizados em situações de cargas de dados variáveis dentro de um sistema (BERSON, 1996).

A principal vantagem deste modelo consiste na possibilidade de distribuição de serviços em rede, o que permite que a funcionalidade geral de um sistema seja disponibilizada para todos os clientes de modo simultâneo e sem levantar a necessidade de um modelo de implementação exaustivo de funcionalidades em todos os serviços do sistema. Entretanto, o modelo também apresenta prejuízos, que são identificados pelos serviços de um sistema individualmente: cada módulo é um ponto de falha por si. Cada serviço é suscetível a ataques ou a falhas de servidor, fenômenos que incluem um fator de imprevisibilidade ao comportamento do sistema devido à dependência direta da rede de distribuição (SOMMERVILLE, 2016).

2.3.5 Arquitetura REST

O padrão arquitetural *Representational State Transfer* (REST) é um modelo idealizado por Fielding (2000) para sistemas desenvolvidos em formato de *Application Programming Interface* API, sendo um formato focado em determinar princípios de comunicação. Desta forma, arquiteturas REST impõem condições acerca do detalhamento funcional e

Figura 7 – Exemplificação do Padrão Arquitetural Cliente-Servidor



Fonte: Adaptação de (SOMMERVILLE, 2016)

comportamental de um sistema, sobretudo quanto à transferência de informações entre diferentes módulos, componentes ou até mesmo sistemas por completo (MASSE, 2011).

Em suma, Fielding (2000) instituiu seis restrições que toda API REST deve respeitar para se adequar ao padrão arquitetural:

- Sistema Cliente-Servidor;
- Interface Uniforme;
- Sistema em Camadas;
- Capacidade de Armazenamento em *Cache*;
- Ausência de Estado, e
- Código sob Demanda.

A **Interface Uniforme** é um ponto fundamental para a estruturação de qualquer serviço REST. A Amazon (2022) define quatro restrições para o tópico da Interface Uniforme:

1. As solicitações à API devem reconhecer recursos do sistema por meio de identificadores únicos e uniformes;
2. Os clientes devem ter autonomia para modificar ou excluir recursos a partir das informações disponíveis e, consequentemente, o servidor atende a esta condição enviando metadados que descrevem o recurso;
3. As mensagens enviadas pelo servidor devem ser auto descritivas para que estas sejam devidamente processadas pelos clientes que recebem a representação, e
4. Os clientes devem receber todas as informações necessárias, referentes a recursos ou dependências relacionadas, para concluir uma determinada tarefa.

A arquitetura do **Sistema em Camadas** viabiliza que o cliente se conecte a outros intermediários, estes devidamente autorizados, entre o cliente e o servidor, de forma que o cliente ainda obtenha respostas do servidor. A asserção é válida para o sentido inverso: o servidor é capaz de repassar solicitações para outros servidores. Desta forma, API's REST podem ser executadas em vários servidores com diferentes camadas distribuídas para atender à demanda de solicitações (AMAZON, 2022).

Serviços REST apresentam a **Capacidade de Armazenamento em *Cache***, que é o processo de manter informações relevantes em memórias localizadas no próprio componente cliente ou em um componente intermediário para melhorar o tempo de resposta do servidor. Por muitas vezes, este ponto é útil para melhorar o tempo de acesso a recursos frequentemente utilizados ou requisitados pelo sistema. Sistemas REST gerenciam o armazenamento em *cache* usando respostas de API que se definem como armazenáveis ou não em *cache* (MASSE, 2011).

Um dos principais tópicos deste modelo é a **Ausência de Estado**, que se refere ao método de comunicação utilizado pelo servidor para completar as solicitações dos clientes de maneira independente de requisições passadas e paralelas. Desta forma, é viabilizado ao cliente que solicite os recursos oferecidos pela API isoladamente de outras requisições. Esta restrição ainda implica que o servidor deve ser capaz de atender às solicitações de maneira homogênea, regular e determinística (AMAZON, 2022).

O conceito de **Código sob Demanda** garante que os servidores possam estender ou personalizar temporariamente a funcionalidade do cliente ao transferir código fonte para o cliente em sistemas REST. Este princípio garante a extensibilidade de sistemas em relação à capacidade de entendimento e execução de código por diferentes módulos do sistema a partir de demandas específicas de utilização (MASSE, 2011).

2.4 Padrões de Projeto de *Software*

Muito conhecidos e difundidos na literatura, Padrões de Projeto não são apenas manuais de instrução que podem ser facilmente replicados de um cenário para outro. Tampouco podem ser definidos como trechos de código específicos para aplicações de termos gerais, mas sim como diretrizes para problemas comumente enfrentados em projetos de *software* (SHVETS, 2018). Pela natureza de exuberante especificidade dos Padrões de Projeto, esta seção traz como objetivo o levantamento teórico de alguns conceitos pertinentes ao domínio de conhecimento e que foram utilizados ao longo do desenvolvimento da aplicação de gerenciamento de times amadores de voleibol.

2.4.1 GRASP's Coesão e Acoplamento

GRASP é um conjunto de nove princípios e diretrizes de padrões de projeto de *software* utilizados para atribuir responsabilidades a classes e objetos durante o desenvolvimento de código (DUNCAN, 2012). Dentre estes conceitos, destacam-se os padrões avaliativos da **Alta Coesão** e **Baixo Acoplamento**, que comumente são analisados em conjunto para medir o nível de adequação, gerenciabilidade e compreensibilidade de códigos orientados a objeto. Esta subseção tem como objetivo apresentar as definições de **Coesão** em conjunto com a definição de **Acoplamento**. Em seguida, é apresentado como estes dois conceitos aplicados em conjunto foram utilizados no desenvolvimento da aplicação de interesse deste trabalho.

O conceito da Coesão define que toda e qualquer implementação de classe deve ser enxuta e objetiva, isto é, toda classe deve implementar um único objetivo para se manter coesa (EDER; KAPPEL; SCHREFL, 1994). Especificamente, todos os métodos e atributos de uma classe devem estar voltados para o mesmo domínio ao qual a classe foi projetada. Ao se definir a Coesão, invariavelmente é levantado o conceito de **Separação de Interesses** (WIN et al., 2002). A Separação de Interesses defende que uma classe deve implementar apenas um interesse, que pode ser interpretado como funcionalidade, requisito ou até mesmo responsabilidade.

Já o conceito de acoplamento abrange o nível de conexão entre classes ou módulos distintos dentro de um mesmo sistema (EDER; KAPPEL; SCHREFL, 1994). Apesar de parecer simples, o conceito possui nuances, as quais derivam complexos desafios encontrados comumente no cotidiano de desenvolvimento. Em essência, o grande impacto de um mau acoplamento é encontrado em dependências entre classes que não são mediadas por uma interface estável (VALENTE, 2020).

Em suma, a Coesão e o Acoplamento são tratados em conjunto para a construção de um Projeto de *Software* robusto. Valente (2020) define uma heurística para lidar com o assunto:

"Maximize a coesão das classes e minimize o acoplamento entre elas."

A citação descreve com maestria e simplicidade a correlação entre esses dois assuntos, visto que, quando uma classe possui várias dependências, é provável que esta esteja assumindo diversas responsabilidades, o que impacta negativamente tanto uma boa Coesão, quanto um bom Acoplamento. Em conformidade com o autor referenciado, os autores do presente trabalho procuraram construir uma solução de *software* que buscou, ao máximo, a aplicação dos conceitos descritos.

2.4.2 Princípios SOLID

Por definição, princípios de Projeto de *Software* são recomendações concretas acerca de decisões de desenvolvimento a nível operacional. Desta forma, Martin (2019) definiu um conjunto de princípios, oriundos de diferentes propriedades de projeto, com o objetivo de assegurar o desenvolvimento de uma solução de *software* com uma boa capacidade de manutenção e configuração. Martin cunhou esse conjunto de princípios como SOLID, acrônimo para as propriedades descritas no Quadro 3.

O *Single Responsibility Principle*, em português **Princípio da Responsabilidade Única**, é uma abstração direta da Propriedade da coesão. Este princípio propõe que toda classe deve possuir uma única e clara responsabilidade. Para este princípio, é acertada a separação das camadas de **apresentação** e das camadas de **regras de negócio**, isto é, devem existir módulos responsáveis pela implementação da interface de usuário e módulos responsáveis pelo processamento de dados que coexistem e se comunicam para a construção de um sistema maior (MARTIN, 2019).

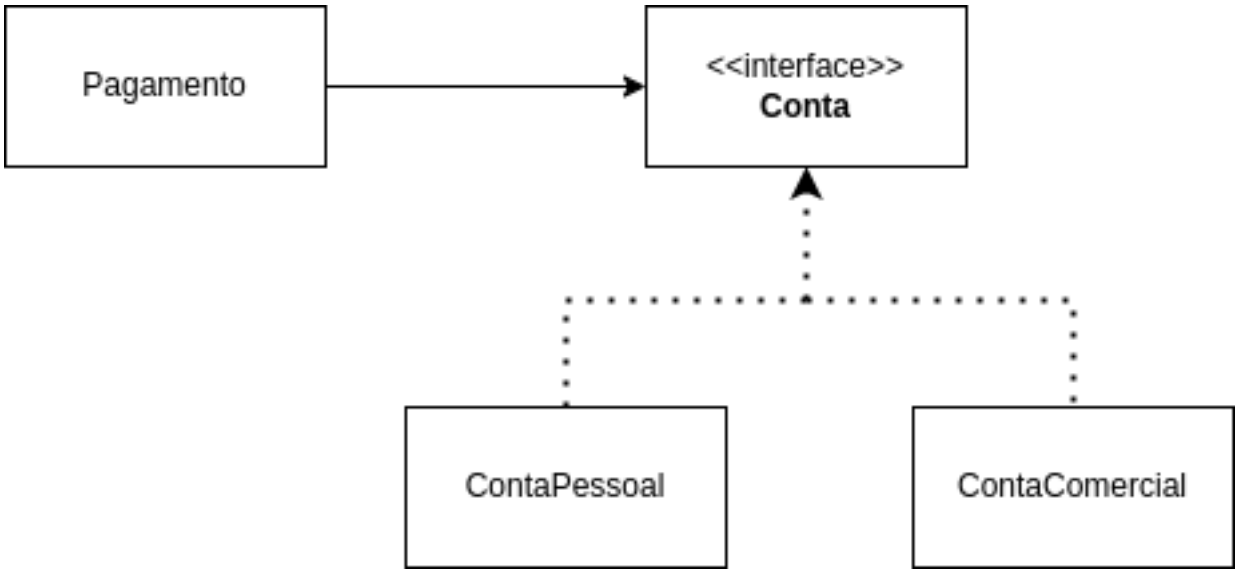
Já o *Open/Closed Principle*, em português **Princípio Aberto/Fechado**, defende que uma classe "deve estar fechada para modificações e aberta para extensões"(MEYER, 1997). Em síntese, a citação defende que o desenho das classes deve permitir extensões e customizações, que podem ser atingidas por meio da utilização de recursos como herança ou funções *lambda* (MARTIN, 2019). Ao aplicar este princípio, a construção de classes flexíveis e extensíveis é viabilizado, de forma que diversos casos de uso possam ser incorporados sem a necessidade de alteração de código-fonte já escrito, sendo apenas necessária a inclusão de novas customizações.

Por sua vez, o *Liskov Substitution Principle*, em português **Princípio da Substituição de Liskov**, define diretrizes para a sobrescrita de métodos de uma determinada classe pai em suas classes filhas. Nomeado após a professora Barbara Liskov, ganhadora do Prêmio Turing de 2008[2], este princípio defende que toda classe derivada a partir de uma determinada classe deve, invariavelmente, ser capaz de processar as mesmas requisições

[2] Disponível em <https://amturing.acm.org/award_winners/liskov_1108679.cfm>. Acesso em: 14 dez. 2022.

da classe pai e retornar respostas do mesmo formato. Tal princípio traz como objetivo reforçar a consistência em herança de classes a fim de evitar erros em dependências diretas (NOBACK, 2018b). Muitas vezes este princípio é mal compreendido. Portanto, a Figura 8 busca trazer uma representação gráfica de classes que respeitam o princípio.

Figura 8 – Modelagem a Nível de Classes que Contempla o Princípio da Substituição de Liskov



Fonte: Adaptação de (MARTIN, 2019)

O *Interface Segregation Principle*, em português **Princípio da Segregação de Interfaces**, também trata de uma extensão direta da aplicabilidade do conceito da **Coesão**. Este princípio pode ser interpretado como o Princípio da Responsabilidade Única particularizado para o contextos de interfaces. Com o objetivo de evitar dependências diretas a métodos sem utilidade, o Princípio da Segregação de Interfaces defende que classes abstratas devem ser enxutas e objetivas (THELMA, 2020).

Por fim, o *Dependency Inversion Principle*, em português **Princípio da Inversão de Dependências**, confere uma visão sobre como estruturar dependências diretas entre classes. O princípio defende que um cliente deve estabelecer, preferencialmente, dependências com abstrações e não com implementações, como por exemplo, utilizar-se do conceito de interfaces. O princípio surge com o propósito de melhorar a extensibilidade e as customizações possíveis de um código-fonte (NOBACK, 2018a).

Ao compilar estes cinco princípios de desenvolvimento em um projeto de *software*, tem-se maior possibilidade de construção de um sistema manutenível e escalável (VALENTE, 2020). Os Princípios SOLID foram utilizados em tempo de desenvolvimento da proposta de *software* do presente trabalho.

Quadro 3 – Princípios SOLID

Princípio de Projeto	**Propriedade de Projeto**
***S**ingle Responsibiliy Principle*	Coesão
***O**pen/Closed Principle*	Extensibilidade
***L**iskov Substitution Principle*	Extensibilidade
***I**nterface Segregation Principle*	Coesão
***D**ependency Inversion Principle*	Acoplamento

Fonte: (VALENTE, 2020)

2.5 Implementação de *Software*

O estágio mais crítico do ciclo de vida de *software* é identificado como a **Implementação**, já que nesta etapa é criada uma versão executável do *software*. Este estágio abrange o desenvolvimento de funcionalidades em si, respeitando restrições, padrões e estilos arquiteturais definidos na etapa de Projeto de Arquitetura, com o intuito de atender os requisitos previamente especificados (SOMMERVILLE, 2016).

A implementação tem o objetivo de construir a aplicação desejada e, com a popularização da *World Wide Web* (WWW), a decisão de criar um sistema *web* tem se tornado cada vez mais frequente. Tratando-se de uma implementação de uma aplicação *web*, é necessário considerar vários aspectos para que ela possa ter a qualidade esperada pelo usuário final. Miletto e Bertagnolli (2014) afirmam esses aspectos como:

- Quais tecnologias podem ser utilizadas?;
- Como é realizado o processamento do sistema?;
- Quais são o perfil do usuário e suas tarefas na interface gráfica da aplicação?, e
- Como será o armazenamento dos dados?.

Esses aspectos são fundamentais para a escolha de tecnologias de desenvolvimento, arquitetura, estilos e padrões a serem implementados, além do tipo do banco de dados que atende às especificações. Complementarmente à decisão do aparato ferramental que será utilizado, faz-se necessário o entendimento dos procedimentos que compõem este desenvolvimento, *i.e.*, sua metodologia de desenvolvimento.

2.5.1 Metodologias Ágeis

As metodologias ágeis são um conjunto de métodos de desenvolvimento iterativo e incremental em que os incrementos e as iterações são pequenos. Essas metodologias orientam o processo de desenvolvimento, como foco no cliente, permitindo obter *feedback* rápido sobre a evolução dos requisitos (SOMMERVILLE, 2016). Estas diferenciam-se de

metodologias tradicionais devido à aceitação das mudanças a partir da postura de assumir que elas existirão e que é necessário avaliar e responder a essas mudanças.

Essas metodologias variam em termos de práticas e ênfases, porém compartilham algumas características, que são inspiradas na filosofia do Manifesto Ágil (MANIFESTO..., 2022), documento criado pelos principais desenvolvedores dos métodos ágeis, em 2001. As principais características compartilhadas entre essas metodologias, como Scrum, XP e Kanban, são o desenvolvimento iterativo e incremental, comunicação e redução de produtos intermediários, como documentação extensiva. Desta forma, existem maiores possibilidades de atender aos requisitos do cliente que, normalmente, são mutáveis.

Elas trazem uma série de vantagens em relação aos métodos tradicionais de desenvolvimento. Uma pesquisa feita em 2021 por Russo, Silva e Larieira (2021) mostra as principais motivações para a adoção de práticas ágeis nas organizações, descritas no Quadro 4.

Quadro 4 – Principais Motivações para a Adoção de Práticas Ágeis em Organizações

Principais Motivações para a adoção de práticas ágeis nas organizações
- Acelerar as entregas dos produtos; - Atender melhor o cliente; - Aumentar a produtividade; - Melhorar a qualidade dos produtos; - Reduzir os riscos nos projetos, e - Melhorar a previsibilidade das entregas.

Fonte: (RUSSO; SILVA; LARIEIRA, 2021)

2.5.1.1 *Scrum*

O *Scrum* é um método ágil voltado para o gerenciamento de projetos e desenvolvimento iterativo de produtos proposto por Sutherland (2010). Esta metodologia inicialmente tinha o foco no desenvolvimento de *software*, porém, com sua popularidade, a mesma vem sendo aplicada ao desenvolvimento de produtos de maneira geral (CARVALHO; MELLO, 2012).

O método em questão não requer ou fornece qualquer técnica específica para a fase de desenvolvimento, apenas estabelece conjuntos de regras e práticas gerenciais que devem ser adotadas para o sucesso de um projeto.

Ele possui três fases, sendo a primeira o planejamento geral, onde ocorre o estabelecimento dos objetivos gerais do projeto e do produto. Em seguida, há uma série de ciclos de *Sprint*, em que a cada iteração há um incremento do sistema. Por fim, a última fase é o encerramento do projeto com a devida documentação (SOMMERVILLE, 2016).

Quadro 5 – Principais Conceitos do *Scrum*

Conceito	**Descrição**
Product Backlog	É uma lista priorizada do que é necessário, classificado em ordem de valor para o cliente ou a empresa, com os itens de maior valor no topo da lista. Será necessária para guiar o desenvolvimento das funcionalidades.
Sprint	Ciclos de trabalho que podem ter duração de 1 a 4 semanas. Possuem duração fixa, tendo o trabalho concluído ou não.
Sprint Planning	Planejamento de cada *sprint*, usando como base os itens priorizados no *Product Backlog*. Assim que os itens são selecionados, o time é dividido para o desenvolvimento.
Sprint Backlog	É uma lista priorizada de itens provenientes do *Product Backlog* que serão selecionados para desenvolvimento na *Sprint*.
Daily Meeting	Reunião diária curta para acompanhamento do andamento do desenvolvimento na *sprint*.
Sprint Review e *Retrospective*	Ao término da *sprint*, o time revisa o que foi feito durante o ciclo, e realiza também uma retrospectiva para levantar pontos de discussão dos pontos positivos e negativos no processo, além de trazer pontos de melhoria para a próxima iteração.

Fonte: Adaptação de (SUTHERLAND, 2010)

Um ciclo de *Sprint* é definido como uma unidade de planejamento, na qual o trabalho a ser feito é avaliado; os recursos para o desenvolvimento são selecionados, e o *software* é implementado. No fim do ciclo, há a entrega aos *stakeholders* (SOMMERVILLE, 2016).

Esse método possui conceitos e papéis próprios. Sutherland (2010) define os principais conceitos, conforme constam no Quadro 5, e elenca os papéis dos integrantes do time como:

1. ***Scrum Master***: É o responsável em aplicar as práticas do Scrum para obter valor de negócio. Tem o papel de servir à equipe, ser o intermediador entre o time e o *Product Owner*, guiando todos às práticas do Scrum;

2. ***Product Owner***: É o responsável pelo produto, identificando as características do produto, que são traduzidas em uma lista de recursos priorizados, o *Product Backlog*. Tem o poder de decidir, a cada iteração, quais itens da lista estarão no topo da lista, e

3. **Time**: Equipe responsável em construir o produto que o cliente irá usufruir. Essa equipe é multifuncional, e inclui toda a experiência necessária para entregar o produto a cada iteração de desenvolvimento. Além disso, ela é auto gerenciável, com um alto grau de autonomia e responsabilidade.

2.5.1.2 Extreme Programming (XP)

O *Extreme Programming* é um método ágil que foi desenvolvido por Beck (2004), com o intuito de impulsionar as práticas que já eram reconhecidamente consideradas boas (SOMMERVILLE, 2016). Ele possui a premissa que, por meio do *feedback* do *stakeholder* sobre o sistema, é possível reavaliar as necessidades e prioridades do sistema a cada iteração, que devem ser incorporadas ao *software* (TELES, 2017).

Ele possui uma organização voltada para um conjunto de valores e práticas que atua de forma harmônica e coesa para assegurar que o cliente receba um alto retorno do investimento do *software* (TELES, 2017). Os quatro valores fundamentais do XP são:

1. ***Feedback***: É o mecanismo que permite que o cliente conduza o desenvolvimento, e garante que a equipe direcione seus esforços para o que gerará mais valor;
2. **Comunicação**: Permite que todos os detalhes do projeto sejam tratados com atenção e agilidade, aproximando todos os envolvidos no projeto;
3. **Simplicidade**: Assume que é preciso implementar aquilo que é suficiente para atender as necessidades dos *stakeholders*. Garante a entrega mais rápida das funcionalidades, gerando mais *feedbacks* e possíveis alterações, e
4. **Coragem**: Necessária para assumir os riscos do projeto, inspirando os envolvidos a utilizarem as práticas e os valores do XP como formas de evoluir o *software* com segurança e agilidade.

As principais práticas do XP, segundo Teles (2017), estão apresentadas no Quadro 6, com suas devidas descrições.

2.5.1.3 Kanban

O Kanban é um método ágil mais moderno, baseado no desenvolvimento Lean (POPPENDIECK; POPPENDIECK, 2003), e tem se tornado uma extensão popular dos métodos ágeis tradicionais como Scrum e XP (BOEG, 2010). Este método tem como objetivo principal uma melhor visualização do trabalho em andamento. Com ele, é possível ter maior entendimento do fluxo de desenvolvimento do projeto, além de possibilitar o acompanhamento das tarefas selecionadas na iteração.

Nele, é apresentado o quadro de atividades (*Kanban Board*) como a principal ferramenta para visualização do fluxo do trabalho. É praticável identificar gargalos no processo, sendo possível estabelecer limites no fluxo, o que acelera as entregas e reduz os riscos técnicos e de negócios (ALEXANDRE; JUNIOR, 2020).

O Kanban possui elementos específicos e fundamentais para o seu funcionamento. Poppendieck e Poppendieck (2003) descrevem estes elementos de acordo com o Quadro 7.

Quadro 6 – Principais Práticas do *Extreme Programming*

Prática	**Descrição**
Pair Programming	Os desenvolvedores implementam as funcionalidades em pares, tendo um piloto, que escreve o código, e o copiloto, que auxilia na implementação. Permite que o código seja revisado enquanto é construído.
Refactoring	É a alteração do um código sem afetar a funcionalidade que ele implementa. Usado para tornar o *software* mais simples de ser manipulado.
Código coletivo	No XP, todos os desenvolvedores possuem acesso ao código, sendo possível realizar alterações sem autorizações, tornando o código coletivo.
Padronização	Para que o código seja desenvolvido por todos os integrantes, é necessário que haja um padrão de codificação, tornando o sistema mais homogêneo e fácil de se manter.
Releases curtas	A equipe produz um conjunto reduzido de funcionalidades, e já os coloca em produção para agilizar as pequenas entregas aos *stakeholders*.

Fonte: Adaptação de (TELES, 2017)

Quadro 7 – Elementos do *Kanban*

Elemento	**Descrição**
Quadro *Kanban*	Quadro físico ou digital que a equipe utiliza para apresentar o fluxo de trabalho. Possui colunas mostrando as etapas do processo e linhas indicando os vários itens que estão sendo produzidos.
Cartão *Kanban*	São as representações móveis das tarefas constantes no quadro. Cada cartão representa uma tarefa que deve ser alocada à coluna que representa seu estado no processo.
Software Kanban	Aplicações específicas em criação de quadros *Kanban*, permitindo o uso do quadro por equipes geograficamente distantes, além de possibilitar automatizações de mudança de cartões, de lembretes e coleta de métricas do processo.

Fonte: Adaptação de (POPPENDIECK; POPPENDIECK, 2003)

2.6 Testes de *Software*

Software é uma das construções humanas mais complexas da contemporaneidade. Portanto, é compreensível que sistemas de *software* estejam sujeitos aos mais variados tipos de erros e inconsistências. A fim de evitar que tais erros sejam entregues em conjunto com o sistema implementado, é fundamental introduzir atividades de testes em projetos de desenvolvimento. É um fato que os **Testes de *Software*** sejam uma das práticas de desenvolvimento mais valorizadas, independentemente do domínio cognitivo do sistema produzido (VALENTE, 2020).

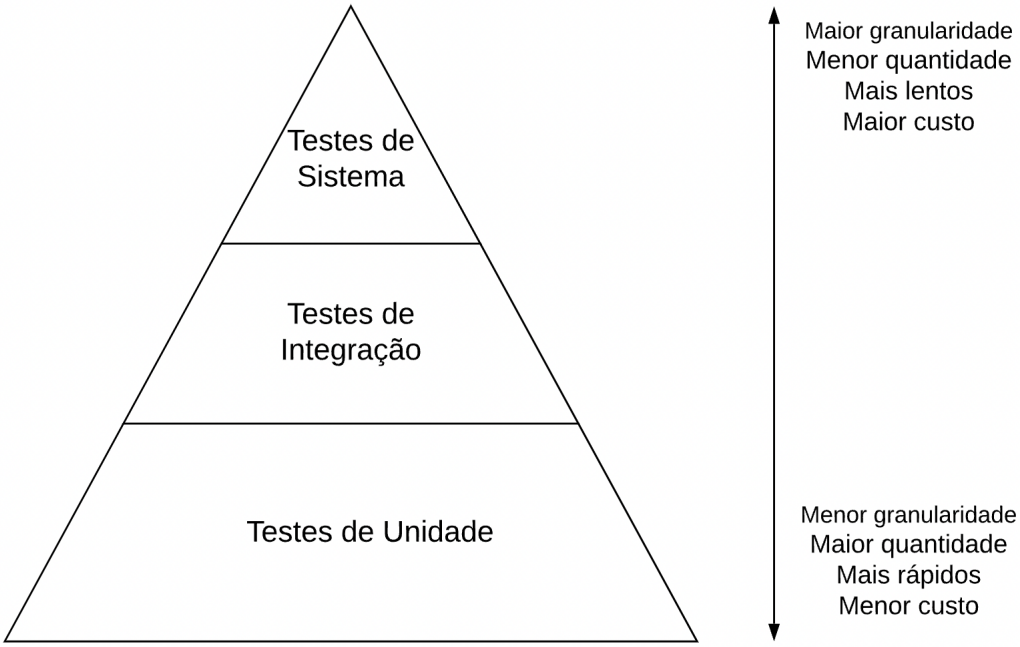
Testes tem como objetivo explicitar que o comportamento desempenhado pelo sistema satisfaz às condições levantadas no comportamento idealizado do sistema. Em um teste de *software*, o programa é executado com a utilização de dados artificiais a fim de identificar erros, anomalias ou informações sobre os atributos não funcionais do sistema. Sommerville (2016) define dois principais objetivos para a disciplina de testes de *software*:

1. Demonstrar ao desenvolvedor e ao cliente que o *software* atende aos seus requisitos, isto é, deve haver pelo menos um teste para cobrir cada requisito modelado, e
2. Identificar entradas nas quais o sistema apresenta um comportamento errôneo, indesejado, ou em desacordo com as especificações estabelecidas.

Com o advento de práticas automatizadas de testes introduzidas em conjunto com a ascensão das metodologias ágeis, Cohn (2010) propôs um modelo de categorização de testes de acordo com a granularidade de abstração. Este modelo ficou conhecido como **Pirâmide de Testes**, e pode ser visualizado na Figura 9.

Os **Testes de Unidade** verificam automaticamente pequenas partes de um código, normalmente compreendidas como unidades atômicas que compõem a base da pirâmide de Cohn (2010). O próximo nível da pirâmide abriga os **Testes de Integração**, que averíguam funcionalidades compreendidas como transações completas de um sistema, ou seja, são testes que encapsulam pacotes distintos e até mesmo componentes externos. Por fim, têm-se os **Testes de Sistema**, que verificam o comportamento do sistema como um todo por meio de simulações fidedignas de sessões de utilização de usuários reais. Neste trabalho, a partir da necessidade de testes automatizados, obteve-se um foco em Testes de Unidade. Sendo assim, cabe um maior aprofundamento sobre essa categoria de teste, conforme segue na próxima seção.

Figura 9 – Pirâmide de Testes



Fonte: (COHN, 2010)

2.6.1 Testes de Unidade

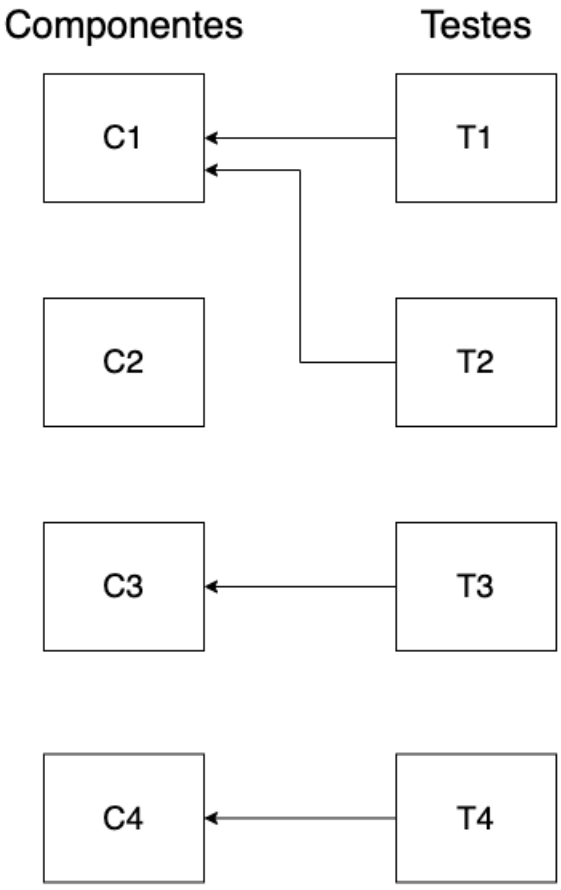
Testes de Unidade são testes automatizados de pequenas unidades de código, as quais são testadas de forma isolada do restante do sistema. Um teste de unidade é um programa que chama métodos de um componente específico, e verifica se os resultados retornados satisfazem os resultados esperados. Assim, quando se usa testes de unidades, o código de um sistema pode ser dividido em dois grupos: um conjunto de implementação e um conjunto de testes, como é ilustrado na Figura 10.

Para desenvolver casos de teste, é preciso adotar um processo pragmático de planejamento que, de acordo com Sommerville (2016), deve considerar:

- O componente testado apresenta uma comprovação factível de comportamento correto com a execução dos testes implementados, e
- Defeitos no componente testado devem ser explicitados por meio da suite de teste executada.

Portanto, devem ser projetados casos de teste que reflitam operações usuais e esperadas ao sistema para que o comportamento idealizado seja realizado e comprovado de maneira prática. De maneira análoga, testes que simulam situações extremas devem ser implementados para que problemas comuns, ou então problemas que possam vir a acontecer de acordo com o comportamento do componente em questão, possam ser evidenciados, identificados e eventualmente corrigidos (SOMMERVILLE, 2016).

Figura 10 – Organização de Testes de *Software*



Fonte: Adaptação de (VALENTE, 2020)

2.7 Implantação de *Software*

Após aplicar todas as etapas anteriormente discutidas no presente trabalho, o projeto de *software* está pronto para entrar no estágio de **Implantação**, isto é, a entrega do produto de *software* produzido por meio de técnicas de distribuição digital.

2.7.1 *DevOps*

O autor Valente (2020) destaca que, historicamente, o setor de tecnologia era organizado em dois departamentos no mercado:

- **Departamento de Desenvolvimento**: formado por desenvolvedores, analistas e arquitetos com foco na implementação de produtos de *software*, e
- **Departamento de Operações**: formado por administradores de redes, administradores de bancos de dados e técnicos de suporte e infraestrutura com foco em

atividades técnicas de sistemas operacionais, como por exemplo a disponibilização de produtos.

Tal modelo organizacional apresentava adversidades inerentes à natureza desta divisão. Majoritariamente, problemas de desenvolvimento só eram alinhados com o departamento de operações nas vésperas de implantações planejadas, o que estava diretamente relacionado com o frequente número de atrasos nas entregas, visto que uma gama de defeitos não eram identificados em várias fases do ciclo de vida do *software*. Entretanto, o problema citado anteriormente não era o único responsável pelos obstáculos da estrutura: a falta de *hardware* adequado para execução do sistema; desempenho aquém do razoável; incompatibilidades com bancos de dados de produção, e vulnerabilidades de segurança protagonizavam o limite para o cancelamento de implantações (VALENTE, 2020).

Para mitigar os problemas de implantação existentes, foi proposto o conceito de ***DevOps***, sigla para *Development and Operation*, que consiste na junção das culturas de desenvolvimento e de operação. A cultura *DevOps* visa viabilizar uma implantação ágil e simplificada de *software* por meio de técnicas e princípios difundidos durante o desenvolvimento.

2.7.2 Integração Contínua e Implantação Contínua

Integração Contínua é uma prática de desenvolvimento proposta em conjunto com o modelo de desenvolvimento *Extreme Programming*, com o objetivo de automatizar tarefas cumulativas e exaustivas do projeto de desenvolvimento a fim de fomentar a produtividade por meio de uma organização inteligente de tarefas e subtarefas (BECK, 2004).

A Integração Contínua garante que todo código novo seja absorvido de maneira frequente e organizada no ramo de desenvolvimento principal. Entretanto, vale ressaltar que isto não significa que o código integrado seja entregue para produção. A fim de aprofundar o nível de integração, a cultura *DevOps* propôs um novo estágio na cadeia de automação: a **Implantação Contínua**. A Implantação Contínua defende que todo código absorvido pela versão principal do código deve estar pronto para produção e, consequentemente, será distribuído e implantado de maneira automatizada (VALENTE, 2020).

A Integração Contínua e Implantação Contínua comumente são referidas por suas siglas em inglês, CI e CD, acrônimos para *Continuous Integration* e *Continuous Delivery*, respectivamente. Estas duas técnicas são usadas em conjunto por muitas organizações com o intuito de adicionar velocidade e modernização ao estágio de implantação de código.

O presente trabalho focou, durante seu período de execução, em um MVP, até ponderando o prazo de entrega estabelecido para realização do TCC. Portanto, não houve

uma preocupação direta com a etapa de Implantação de *Software*. Entretanto, como o intuito foi prover uma solução computacional direcionada a um público alvo específico, ou seja, equipes de voleibol, seja no escopo de atuação desse trabalho, seja na evolução do mesmo, foi imprescindível que essa etapa fosse implementada. Diante do exposto, foi conferido um olhar neste sentido, além de apontamentos sobre Manutenção de *Software*. Outros detalhes constam no Capítulo 5.

2.8 Manutenção de *Software*

A **Manutenção do *software*** é o processo geral de mudança de um sistema após sua liberação para uso (SOMMERVILLE, 2016). Essas mudanças podem escalar de simples correções de codificação até alterações extensas para acomodação de novos requisitos. Sommerville (2016) separa a manutenção de *software* em três tipos, sendo eles:

1. **Correção de defeitos**: Abrange erros de codificação, de projeto e requisitos. Os erros de codificação são mais baratos e rápidos para serem corrigidos. Os erros de projeto podem se tornar caros, pois podem afetar vários componentes do sistema. Já os erros de requisitos são os mais caros dentre os três, devido à necessidade de reprojeto do sistema;

2. **Adaptação ambiental**: Faz-se necessária uma manutenção adaptativa quando algum aspecto do ambiente do sistema, como por exemplo o *hardware* ou algum *software* de apoio, sofre alguma mudança, e

3. **Adição de funcionalidade**: Capaz de receber novos requisitos ao sistema, podendo ser uma mudança organizacional ou até mesmo de negócios.

Na prática, não há uma distinção clara entre os tipos de manutenção, pois eles podem ser realizados de forma conjunta para um mesmo objetivo. Todos eles demandam um custo, que varia de um domínio de aplicação para outro (SOMMERVILLE, 2016).

Novamente, devido ao prazo para realização do TCC, não houve um olhar aprofundado em Manutenção de *Software*. Entretanto, visando mitigar alguns problemas comumente encontrados em tempo de manutenção, a solução computacional do presente trabalho contou com atividades de validação junto ao público alvo, em tempo de Análise (LEITE, 2022), usando como base o protótipo de alta fidelidade. Essa prática permitiu acordar problemas desde a Engenharia de Requisitos, o que viabilizou a construção de um produto que satisfez mais adequadamente os usuários, demandando, consequentemente, menor manutenção, em curto e médio prazos.

2.9 Considerações Finais do Capítulo

Este capítulo buscou salientar sobre os principais conceitos que servem como base para o entendimento do contexto em que se inseriu a proposta deste Trabalho de Conclusão do Curso: o desenvolvimento de um produto de *software* como aplicação web para gerenciamento de times amadores de voleibol.

Para a apresentação desses conceitos, fez-se necessária uma breve explicação sobre a **Engenharia de *Software*** como um todo ao perpassar pelos tópicos de ciclo de vida do *software*, abordando em conjunto cada etapa do processo de Engenharia de Requisitos, Projeto de Arquitetura de *Software*, Padrões de Projeto de *Software*, Implementação, Testes, Implantação e Manutenção.

Cada tópico confere particularidades que foram abordadas ao longo do capítulo e foram primordiais para o desenvolvimento prático do projeto.

3 Suporte Tecnológico

Este capítulo apresenta as tecnologias utilizadas ao longo da realização desse trabalho. Nesse sentido, subdivide-se nos seguintes tópicos: API e Camada de Dados; Interface de Usuário e Prototipagem; Gerência e Configuração de *Software*, e Edição de Texto. Por fim, têm-se as Considerações Finais do Capítulo, seção na qual é apresentado o Quadro 8, com as principais tecnologias do trabalho, as respectivas descrições de uso, e as versões dos produtos de *software* selecionados.

3.1 API e Camada de Dados

Esta seção apresenta as principais tecnologias e ferramentas utilizadas para o desenvolvimento do *back-end* da aplicação. Nesta seção, são abrangidas as ferramentas para criação da API, em conjunto com as ferramentas utilizadas para compor a camada de dados. Uma API (*Application Programming Interface*) (MATHIJSSEN; OVEREEM; JANSEN, 2020) representa, basicamente, um conjunto de definições, diretrizes e protocolos que permitem desenvolvimento e integração de aplicações de forma padronizada. Quando reunidas e devidamente aplicadas, APIs e tecnologias associadas à camada de persistência propiciam o desenvolvimento efetivo de um sistema *back-end*.

3.1.1 FastAPI 0.88.0

O FastAPI (FASTAPI, 2022) é um *framework* Python focado no desenvolvimento de API's *web*. Ele tem como principais características sua rapidez, modernidade e simplicidade. Vale ressaltar que esta é uma tecnologia relativamente recente. Entretanto, cada vez mais, tem obtido destaque em grandes empresas do mercado de *software*, e aderência em comunidades Python, o que indica um considerável fator de crescimento para a tecnologia.

Por ser novo no mercado, possui algumas vantagens em relação a outros *frameworks web* para Python, como, por exemplo o *Django* ou o *Flask*. O FastAPI possui, por exemplo, um alto desempenho; uma baixa curva de aprendizado, e um bom grau de intuitividade.

3.1.2 SQLAlchemy 1.4.44

O SQLAlchemy é uma biblioteca de Mapeamento Objeto-Relacional (ORM) (SQLALCHEMY, 2022) desenvolvida para a linguagem Python. Uma grande vantagem da ferramenta é o auxílio fornecido na conversão de dados entre banco de dados relacionais e

linguagens de programação, isto é, a biblioteca reduz a programação necessária ao acesso ao banco de dados, aumentando, então, a produtividade no desenvolvimento de aplicações.

No presente trabalho, o SQLAlchemy foi utilizado em conjunto com o *framework* FastAPI, com o intuito de auxiliar na interação entre a API e o banco de dados relacional PostgreSQL. Sua utilização visou amenizar a necessidade do desenvolvedor em se preocupar com a linguagem SQL e com as conversões feitas entre os diferentes tipos de dados.

3.1.3 PostgreSQL 15.1

O PostgreSQL (POSTGRESQL, 2022) é um SGBD focado na implementação da linguagem SQL em estruturas, a fim de garantir um trabalho colaborativo entre os padrões relacionais e a aplicação da persistência de dados. Um dos principais pontos do PostgreSQL é a adequação com padrões de conformidade para construir bancos de dados otimizados.

O SGBD em questão lida bem com altos volumes de solicitações de consultas e com grandes cargas de trabalho, ou seja, o seu funcionamento é bastante recomendável para a construção de *sites* com uma grande intensidade de acesso, como por exemplo, *e-commerces* famosos que necessitam de uma estrutura organizada para obter um desempenho otimizado, devido ao alto número de acessos simultâneos. Apesar do produto construído nesse trabalho não demandar, em um primeiro momento, muitos acessos, a intenção é já prover suporte adequado para futuras demandas do *software*, com um sistema de gerenciamento de banco de dados condizente com vários acessos simultâneos.

3.1.4 Alembic 1.11.1

O Alembic (ALEMBIC, 2023) é uma ferramenta para o gerenciamento de migrações, e versionamento de bancos de dados desenvolvida pelos mesmos criadores do SQLAlchemy. Trata-se de uma ferramenta que sintetiza operações de alteração e criação de tabelas ou entidades do banco de dados, além de permitir que as operações sejam atômicas e reversíveis.

A ferramenta foi utilizada, no presente trabalho, com o intuito de prover uma interface de manutenção eficiente para a Camada de Dados do sistema. Desta forma, mudanças nos esquemas de dados foram viabilizadas de maneira transparente e simplificada, de forma que não houvesse a necessidade de execução de tarefas de alto custo para tal.

3.2 Interface de Usuário e Prototipagem

Esta seção apresenta as principais tecnologias e ferramentas utilizadas para a confecção de protótipos de alta fidelidade e para a criação de interfaces do usuário, também conhecidas como *front-end* de sistemas. Vale ressaltar que o trabalho compreende uma aplicação voltada para plataformas web, o que fez-se necessário o uso de ferramentas adequadas para a confecção e a prototipagem da finalidade em questão.

3.2.1 ReactJS 17.0.2

O ReactJS (REACT, 2022) é uma biblioteca *Javascript* de código aberto, que possui o objetivo de criar interfaces de usuário em sites *web*. Ele possui um conjunto de funções organizadas, que podem ser utilizadas para a construção de uma aplicação chamada de *Single Page Application* (SPA).

O ReactJS utiliza uma extensão de sintaxe para *Javascript* chamada JSX. Esta extensão é sintetizada em uma forma de criar elementos para serem utilizados como *templates* de aplicações React, assemelhando-se à sintaxe HTML. Por não ser interpretado pelo navegador, é necessária a utilização de um transpilador para a conversão de código para uma linguagem entendida por navegadores.

3.2.2 Figma

O Figma (FIGMA, 2022) é uma plataforma colaborativa para construção de *design* de interfaces e protótipos. Com ela, é possível criar protótipos de alta fidelidade já com toda a interação entre telas. Dentre outras vantagens, essa interação proporciona a realização de testes de validação do fluxo da informação da aplicação com usuários reais.

A plataforma é vista como uma adequada ferramenta de validação, auxiliando na melhor experiência do usuário na aplicação. Por ser colaborativa, mais de uma pessoa pode trabalhar no mesmo projeto de maneira simultânea, facilitando e agilizando sua conclusão.

3.3 Gerência e Configuração de *Software*

Nesta seção, são apresentadas as ferramentas utilizadas para fornecer o apoio ao desenvolvimento do projeto. As ferramentas listadas a seguir oferecem apoio aos processos de:

- Acompanhamento de mudanças;
- Integração contínua;

- Gerenciamento de dependências;
- Estruturação do ambiente de desenvolvimento, e
- Controle de versão.

3.3.1 Git 2.34.1

O Git (GIT, 2022) é um exemplo de *Data Version Control* (DVC), ou seja, um sistema de controle de versão, que possui uma arquitetura distribuída. Ele é gratuito, de código aberto, e projetado para possuir o histórico completo de alterações em forma de repositório, em que todo desenvolvedor possui sua cópia.

Trata-se de um recurso desenhado para ter velocidade e eficiência, pois quase todas as suas operações são executadas localmente, dando a ele uma enorme vantagem de velocidade em sistemas centralizados que constantemente precisam se comunicar com um servidor em algum lugar. Escrito em linguagem C, o Git reduz a sobrecarga de tempos de execução associados a linguagens de nível superior.

3.3.2 GitHub

O GitHub (GITHUB, 2022) é uma plataforma de hospedagem de código-fonte e arquivos com controle de versionamento utilizando o Git para isso. Com ele, é possível ter acesso aos mais diversos projetos de código aberto, além de poder contribuir significamente para estes projetos que ficam disponíveis.

O Github possui muitas funcionalidades voltadas para às comunidades de *software* livre, adotando muitas práticas ágeis. Ele possibilita a interação entre os usuários da aplicação, que pode ocorrer por meio de criação de *issues*, revisão de código, organização de tarefas em um determinado período de tempo, entre outros.

3.3.3 Yarn 1.22.19

O Yarn (YARN, 2022) é um gerenciador de pacotes de código aberto, que permite o uso de soluções de outros desenvolvedores para os mais diversos problemas de codificação. O código é compartilhado por meio de um pacote, que o desenvolvedor consegue instalar em seu projeto e utilizá-lo.

Por ser de código aberto, o Yarn possui inúmeros colaboradores para os mais diversos pacotes. Tal característica colaborativa torna a ferramenta uma solução adequada para agilizar o desenvolvimento de aplicações *web* que usam *Javascript* ou *Typescript* como base.

3.3.4 pip 22.3.1

O pip (PIP, 2022) é o gerenciador de pacotes padrão para a linguagem de programação *Python*. Adotado a partir da versão 2.7.9 da série *python2* e da versão 3.4 da série *python3*, o pip é distribuído como pacote pré-instalado da linguagem, o que faz com que seu uso seja amplamente difundido. A ferramenta possibilita ainda encontrar, adicionar, gerenciar e excluir dependências em projetos por meio de uma interface de linha de comando, tornando sua utilização simples e objetiva.

3.3.5 macOS Monterey 12.6.1

O macOS (MACOS. . . , 2022) é um sistema operacional proprietário, desenvolvido e distribuído pela empresa Apple, destinado para computadores da própria fabricante. O macOS é um sistema operacional tipo Unix, isto é, foi derivado e desenvolvido a partir do Unix, o que permite com que o sistema desempenhe uma série de comportamentos que auxiliam no desenvolvimento de *software*.

3.3.6 Ubuntu Jammy Jellyfish 22.04.1

O Ubuntu (UBUNTU. . . , 2022) é um sistema operacional de código aberto, construído a partir do núcleo Linux, baseado no Debian. Utiliza GNOME como ambiente de *desktop* de sua versão mais recente como suporte de longo prazo.

A principal proposta do Ubuntu é oferecer um sistema acessível, para qualquer finalidade do usuário final. Ele é amplamente utilizado em meio acadêmico por ser de fácil uso e instalação; atender as necessidades de alunos da tecnologia da informação, além de ser livre de taxas.

3.3.7 Docker 20.10.12

O Docker (DOCKER, 2022) é uma plataforma aberta para desenvolvimento, envio e execução de aplicações. Ele permite um certo isolamento do ambiente de desenvolvimento, resolvendo problemas de incompatibilidade de bibliotecas com o sistema operacional. Esse ambiente isolado é chamado de contêiner, sendo ele leve e possuindo tudo o que for necessário para a execução da aplicação.

O Docker simplifica o ciclo de vida do desenvolvimento, permitindo que os desenvolvedores trabalhem em ambientes padronizados usando contêineres locais que fornecem seus aplicativos e serviços.

3.3.8 Heroku 8.1.9

O Heroku (HEROKU, 2023) é uma plataforma do tipo PaaS (*Platform as a Service*) que possibilita a hospedagem, configuração, testagem e publicação de projetos virtuais na nuvem. Dentre suas funcionalidades, destaca-se a facilitação do trabalho dos desenvolvedores no que diz respeito à configuração da infraestrutura necessária para a implantação, ou seja, o *deploy*, das aplicações.

Ele é uma plataforma de computação em nuvem, na qual os usuários podem acessá-la de forma remota através de dispositivos conectados à internet. Essa tecnologia é de extrema importância, pois proporciona acessibilidade e segurança no armazenamento dos dados.

Os códigos das aplicações são armazenados em contêineres, os quais são comumente executados em um ambiente compartilhado, porém sempre de forma isolada e em ambientes leves. Nele, esses contêineres são denominados *dynos*. Cada ambiente de construção possui elementos computacionais, memória, sistema operacional e um sistema de arquivos efêmero.

3.4 Edição de Texto

Esta seção apresenta as principais tecnologias e ferramentas utilizadas para a confecção e a edição dos textos relacionados ao presente trabalho. Vale ressaltar que, para esta seção, o termo texto é utilizado para definir todo e qualquer conjunto de palavras relevantes para o trabalho, seja este um arquivo de código ou então a própria redação da monografia.

3.4.1 LaTeX

O LaTeX (LATEX. . . , 2022) é um sistema de composição tipográfica de alta qualidade. Nele, são inclusas funcionalidades destinadas à produção de documentação técnica e científica. O LaTeX é o padrão de fato para a comunicação e a publicação de documentos científicos, sendo disponível como *software* livre.

3.4.2 Overleaf

O Overleaf é um editor de texto *online* e colaborativo, utilizado para escrita, edição e publicação de documentos científicos. O site oferece como funcionalidade a criação de projetos baseados na linguagem de marcação LaTeX. Estes projetos podem abranger uma série de arquivos que, quando compilados em conjunto pela própria plataforma, geram um arquivo PDF devidamente formatado. Também é possível compartilhar projetos e

trabalhar simultaneamente com outros usuários de maneira gratuita (OVERLEAF..., 2022).

3.4.3 Visual Studio Code 1.73

O Visual Studio Code (VISUAL..., 2022) é um editor de código fonte leve, mas adequado, executado em sua área de trabalho, e disponível para Windows, macOS e Linux. Ele vem com suporte integrado para *JavaScript*, *TypeScript* e *Node.js*, possuindo um adequado ecossistema de extensões para outras linguagens e tempos de execução.

3.4.4 PyCharm 2022.2.4

O PyCharm é um ambiente de desenvolvimento integrado (IDE), dedicado à codificação de programas e sistemas baseados na linguagem de programação *Python* (PYCHARM, 2022). O editor oferece uma série de facilidades para o dia-a-dia de desenvolvimento, como por exemplo:

- Assistência de código;
- Ferramentas de refatoração;
- Analisador estático de guia de estilo;
- Suporte ao *framework* utilizado na pesquisa, e
- Integração com as ferramentas de versionamento de código utilizadas.

É importante salientar que a ferramenta possui três edições que são distribuídas ao público: *Community*, *Professional* e *Edu*. Dentre estas três, a única que possui livre acesso, entretanto limitado, é a *Community*. Para o desenvolvimento da aplicação, foi utilizada a edição *Edu*, que é fornecida a discentes e docentes de instituições educacionais licenciadas, como é o caso da Universidade de Brasília.

3.5 Considerações Finais do Capítulo

Este capítulo buscou salientar sobre as principais tecnologias utilizadas no decorrer deste trabalho. O Quadro 8 sintetiza de maneira objetiva as tecnologias utilizadas no desenvolvimento da aplicação, com as respectivas versões e *links* dos referenciais tecnológicos.

Ainda constam ferramentas que viabilizaram a comunicação entre os autores desse trabalho, bem como entre os autores e orientadora, tais como: Telegram (TELEGRAM, 2023), Slack (SLACK, 2023) e Microsoft Teams (MICROSOFT..., 2023). Por fim, foram

Quadro 8 – Resumo das Tecnologias Utilizadas no Projeto

Nome	**Descrição de Uso**	**Versão**	***Link***
FastAPI	Biblioteca para desenvolvimento da API	0.88.0	<https://fastapi.tiangolo.com/>
SQLAlchemy	ORM de integração do banco de dados	1.4.44	<https://www.sqlalchemy.org/>
Alembic	Ferramenta de gerenciamento de migrações	1.11.1	<https://alembic.sqlalchemy.org/en/latest/>
PostgreSQL	Banco de dados relacional para armazenar os dados da aplicação	15.1	<https://www.postgresql.org/>
ReactJS	Biblioteca de desenvolvimento da interface do usuário	17.0.2	<https://pt-br.reactjs.org/>
Figma	Ferramenta para prototipagem de alta fidelidade	-	<https://www.figma.com/>
Git	Ferramenta para controle de versão da aplicação	2.34.1	<https://git-scm.com>
Github	Plataforma para hospedagem do código fonte da aplicação	-	<https://github.com>
Yarn	Gerenciador de pacotes para ambiente JavaScript	1.11.19	<https://yarnpkg.com/>
pip	Gerenciador de pacotes para ambiente Python	22.3.1	<https://pypi.org/project/pip/>
macOS	Sistema Operacional *desktop* para desenvolvimento e testes da aplicação	12.6.1	<https://support.apple.com/pt-br/HT212585>
Ubuntu	Sistema Operacional *desktop* para desenvolvimento e testes da aplicação	22.04.1	<https://releases.ubuntu.com/22.04/>
Docker	Plataforma de isolamento de ambiente de desenvolvimento	20.10.12	<https://docs.docker.com/>
Heroku	Plataforma de hospedagem em nuvem	8.1.9	<https://www.heroku.com/>
Latex	Sistem de composição tipográfica de alta qualidade para documentação técnica	-	<https://www.latex-project.org>
Overleaf	Editor de texto *online* e colaborativo para escrita científica	-	<https://pt.overleaf.com/learn>
Visual Studio Code	Ambiente de desenvolvimento integrado para escrita e depuração do código fonte	1.73	<https://code.visualstudio.com>
Pycharm	IDE para ambiente Python	2022.2.4	<https://www.jetbrains.com/help/pycharm/>

Fonte: Autores.

utilizadas ferramentas que permitem especificar modelos, por exemplo, a ferramenta Heflo (HEFLO, 2023), notação BPMN, para modelagem dos processos especificados no Capítulo 4.

4 Metodologia

Este capítulo tem como objetivo apresentar as metodologias utilizadas para a escrita, exploração teórica e desenvolvimento prático do trabalho. A organização do capítulo dá-se, inicialmente, pela Classificação da Pesquisa, que procura categorizar o trabalho quanto à Abordagem, à Natureza, aos Objetivos e aos Procedimentos. Em seguida, é apresentada a Metodologia do trabalho em que são descritos: o Fluxo de Atividades; o Levantamento Bibliográfico; a Metodologia de Desenvolvimento, e a Metodologia de Análise de Resultados. Por fim, é definido o Cronograma de Atividades e expostas as Considerações Finais do Capítulo.

4.1 Classificação da Pesquisa

A Metodologia pode ser classificada como o estudo da organização, dos caminhos a serem seguidos, com o objetivo de realizar uma pesquisa ou estudo (GERHARDT; SILVEIRA, 2009). Esta seção visa categorizar a pesquisa deste trabalho quanto à abordagem, natureza, objetivos e aos procedimentos de acordo com as definições previstas por (GERHARDT; SILVEIRA, 2009).

4.1.1 Abordagem

A pesquisa utilizada neste trabalho pode ser classificada tendo abordagem Qualitativa. Nesse sentido, o objetivo é obter uma compreensão mais profunda de um grupo social, organização, entre outros. Com isso, preocupa-se com aspectos da realidade que não podem ser quantificados, centrando-se na compreensão e na explicação das dinâmicas das relações sociais (GERHARDT; SILVEIRA, 2009).

No caso do trabalho desenvolvido, o foco está no atendimento dos requisitos do público alvo definido, com a finalidade de conferir uma análise qualitativa, procurando orientar-se por critérios mais subjetivos, tais como satisfação e experiência dos usuários ao usar a aplicação construída.

4.1.2 Natureza

Quanto à natureza, a pesquisa deste trabalho pode ser classificada como Aplicada. A pesquisa aplicada visa construir conhecimentos para a aplicação prática, de forma a resolver problemas específicos. Diferentemente da pesquisa básica, que busca gerar conhecimentos novos, porém sem previsão de aplicação prática (GERHARDT; SILVEIRA, 2009).

O viés do presente trabalho acorda o desenvolvimento de uma aplicação, ou seja, um produto de *software*, especificamente pensado para um público alvo, visando mitigar problemas direcionados ao domínio de uma área de interesse, que é o gerenciamento de times amadores de voleibol.

4.1.3 Objetivos

Quanto aos objetivos, a pesquisa deste trabalho classifica-se como Exploratória. Ela consiste em proporcionar maior familiaridade com o problema, com a intenção de torná-lo mais explícito ou de formular hipóteses (GERHARDT; SILVEIRA, 2009).

Como dito anteriormente, a aplicação explora o domínio do gerenciamento de times amadores de voleibol, visando proporcionar facilidade na administração dos times. Para isso, o desenvolvimento da aplicação foi realizado consultando *stakeholders* próximos, que participaram ativamente de todo o processo envolvido no ciclo de vida do *software*.

4.1.4 Procedimentos

Quanto aos procedimentos, a pesquisa deste trabalho pode ser classificada como Bibliográfica e Pesquisa-Ação. A pesquisa bibliográfica é realizada via acervo de referências teóricas, analisadas e publicadas por meios escritos e tecnológicos, podendo ser em forma de livros, artigos científicos, sites, entre outros. Já a pesquisa-ação envolve a participação planejada do pesquisador na situação a ser investigada, com o objetivo de diagnosticar o problema, procurando obter um resultado prático (GERHARDT; SILVEIRA, 2009).

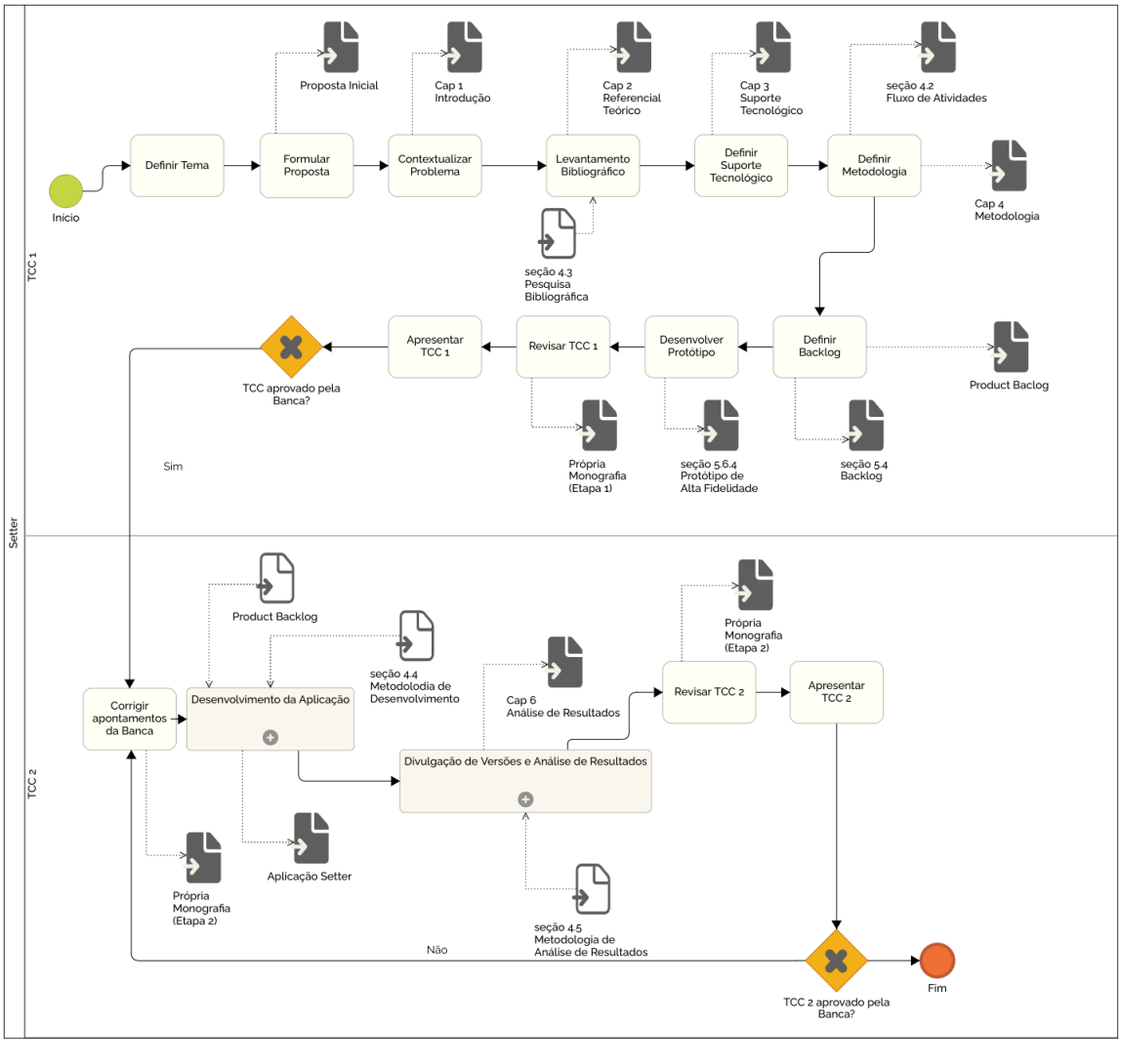
Neste trabalho, foi realizado um levantamento bibliográfico, que é mais bem apresentado na Seção 4.3, acordando todo o ciclo de vida do *software*, desde sua concepção até sua manutenção, com destaque às fases que são abordadas no trabalho com maior veemência. Nele, também fez-se uma seleção de um grupo de pessoas que acompanharam o desenvolvimento da aplicação, como *stakeholders*, e uma grande particularidade do trabalho é que um dos autores faz parte deste grupo, sendo a principal *stakeholder* do projeto. Além disso, no intuito de analisar os resultados obtidos, foi utilizado a pesquisa-ação, conforme acordado na seção 4.5.

4.2 Fluxo de Atividades

Para a condução deste trabalho, fez-se necessário o estabelecimento de algumas etapas. Essas etapas podem ser visualizadas no modelo, Figura 11, baseado na notação BPMN [1]. A descrição de cada etapa ocorre na sequência.

[1] Notação gráfica para representação de processos de negócios (BPMN, 2023)

Figura 11 – Fluxo de Atividades do TCC



Fonte: Autores

1. **Definir Tema**: atividade destinada à escolha do tema de desenvolvimento do trabalho. O tema surgiu como uma ideia de um dos autores, que foi totalmente aceita pelo outro autor, e validada com a orientadora. *Status*: Concluída;

2. **Formular Proposta**: atividade que visa o fechamento do escopo, bem como contextualizar, identificar questões de pesquisa, estabelecer objetivos e conferir justificativas. *Status*: Concluída; Resultado: Capítulo Introdução (1)

3. **Contextualizar Problema**: atividade de desenvolvimento de uma visão mais ampla da área de interesse, que no caso foi o gerenciamento de times amadores de voleibol. *Status*: Concluída; Resultado: Capítulo Introdução (1)

4. **Levantamento Bibliográfico**: subprocesso importante, que permitiu a consulta em bases científicas, afim de investigar a literatura especializada sobre o Ciclo de Vida do *Software* e seus desdobramentos. Outros detalhes relevantes sobre este sub-

processo constam na Seção 4.3. *Status*: Concluída; Resultado: Capítulo Referencial Teórico (2);

5. **Definir Suporte Tecnológico**: atividade que visa a definição dos apoios tecnológicos relevantes para a realização do trabalho em sua completude. *Status*: Concluída; Resultado: Capítulo Suporte Tecnológico (3);

6. **Definir Metodologia**: atividade relevante em que os autores se dedicaram ao desenho de um processo, afim de viabilizar a condução do trabalho em sua completude. *Status*: Concluída; Resultado: Fluxo de Atividades apresentado nesta seção (4.2);

7. **Definir *Backlog***: atividade relevante para iniciar o desenvolvimento da aplicação, desde o protótipo até o pleno desenvolvimento. Seu objetivo foi definir de forma objetiva e clara os principais requisitos da aplicação. *Status*: Concluída; Resultado: Capítulo Setter (5) - Seção 5.4;

8. **Desenvolver Protótipo da Aplicação**: atividade que visou a elaboração de um protótipo de alta fidelidade com base nos requisitos elencados no *Backlog*. O protótipo teve o objetivo de demonstrar de forma visual a proposta da aplicação, podendo ser visto como uma prova de conceito, em relação à proposta, uma vez que se orientou por e procurou atender vários detalhamentos em termos de funcionalidades e fluxos pretendidos na aplicação. *Status*: Concluída; Resultado: Capítulo Setter (5) - Seção 5.6.4;

9. **Revisar TCC1**: atividade de revisão de cada aspecto do TCC, incluíndo sua escrita e os artefatos produzidos. Ela foi feita por meio de iterações, tanto entre os autores, quanto dos autores com a orientadora. *Status*: Concluída; Resultado: Própria Monografia (Etapa 1) e artefatos em geral;

10. **Apresentar TCC1**: atividade destinada à aprovação da proposta, junto à Banca Avaliadora. *Status*: Concluída;

11. **Corrigir Apontamentos da Banca**: atividade que buscou, com base nos *feedbacks* realizados pela Banca Avaliadora, revisar a Monografia e seus artefatos, fazendo as devidas correções solicitadas. *Status*: Concluída; Resultados: Própria Monografia (Etapa 2) e artefatos em geral evoluídos;

12. **Desenvolvimento da Aplicação**: processo que consistiu no desenvolvimento do MVP da aplicação. O detalhamento deste processo consta na Figura 12. As subatividades desenvolvidas estarão explanadas na Seção 4.4, na qual é definida a Metodologia de Desenvolvimento. *Status*: Concluída; Resultado: Aplicação *Setter* (SETTER, 2023);

13. **Divulgação de Versões e Análise de Resultados**: subprocesso que consistiu em aplicar Pesquisa-Ação, conforme consta na Seção 4.1.4, com o intuito de coletar métricas qualitativas, e realizar evoluções na aplicação com base nessas métricas. Nesse processo, foi feita a validação da aplicação por parte dos *stakeholders* por meio de *feedbacks* que foram coletados na divulgação da versão do MVP da aplicação. *Status*: Concluída; Resultado: Capitulo Análise de Resultados (6);

14. **Revisar TCC2**: atividade que procurou revisar cada aspecto do TCC, incluindo escrita e elaboração de artefatos inerentes ao escopo de trabalho do TCC2. Foi utilizado o mesmo processo iterativo, conduzido para o escopo do TCC1. *Status*: Concluído; Resultados: Monografia (Etapa 2) e artefatos em geral, em suas versões finais;

15. **Apresentar TCC2**: atividade destinada à aprovação do trabalho na totalidade, junto à Banca Avaliadora. *Status*: Em andamento.

4.3 Levantamento Bibliográfico

O Levantamento Bibliográfico foi realizado de modo a entender o ciclo de vida do *software* em sua totalidade, abordando de forma mais profunda assuntos pertinentes para o desenvolvimento deste trabalho. Não foi elaborada uma *string* de busca, mas o objetivo foi encontrar fontes confiáveis para o embasamento teórico deste Trabalho de Conclusão de Curso.

Os principais autores e artigos usados tiveram como base referências usadas em disciplinas anteriores, cursadas durante a graduação, com grande enfoque na literatura de Sommerville (2016). As palavras-chave utilizadas para guiar as pesquisas foram as mesmas usadas para as definições contidas no Embasamento Teórico, que se encontra no Capítulo 2, tais como: Ciclo de Vida de *Software*, Engenharia de Requisitos, Metodologias Ágeis, Arquitetura de *Software*, entre outros.

Vale ressaltar que alguns termos correspondentes foram utilizados na língua inglesa para obter uma maior quantidade de referências no contexto proposto. Foram obtidos um total de 83 referências, incluindo artigos, livros e sítios que podem ser visualizados nas Referências ao final do trabalho.

A seleção das referências para a pesquisa bibliográfica levou em consideração a exclusão dos seguintes critérios:

- Textos em idiomas diferentes de inglês e português;
- Textos que não se aplicam ao contexto de *software*;

- Textos que abordam a parte de *hardware* de sistemas computacionais;
- Textos que abordam algoritmos específicos de *software*;
- Textos que definem arquitetura de *software* de maneira discrepante da defendida por Sommerville (2016), e
- Textos que definem as fases da Engenharia de Requisitos de *software* de maneira discrepante da defendida por Jarke e Pohl (1994).

Adicionalmente, levou-se em consideração a inclusão dos seguintes critérios:

- Textos que abordam de forma ampla todo o ciclo de vida do *software*;
- Textos que abordam sobre a Engenharia de Requisitos e seus processos;
- Textos que falam sobre métodos ágeis;
- Textos que abordam de forma ampla Arquitetura de *Software* e padrões de projeto, e
- Textos que explicam de forma geral os conceitos de Manutenção Evolutiva, Testes e Implantação.

Com o objetivo de fornecer uma visão mais clara e aprofundada sobre o processo de levantamento bibliográfico, foi elaborado um Documento de Referências. Este documento é responsável por centralizar e manipular a bibliografia deste trabalho. Ainda neste sentido, é importante frisar que este documento não representa a totalidade das referências, e sim um apanhado dos principais textos que fundamentaram a argumentação teórica do presente trabalho.

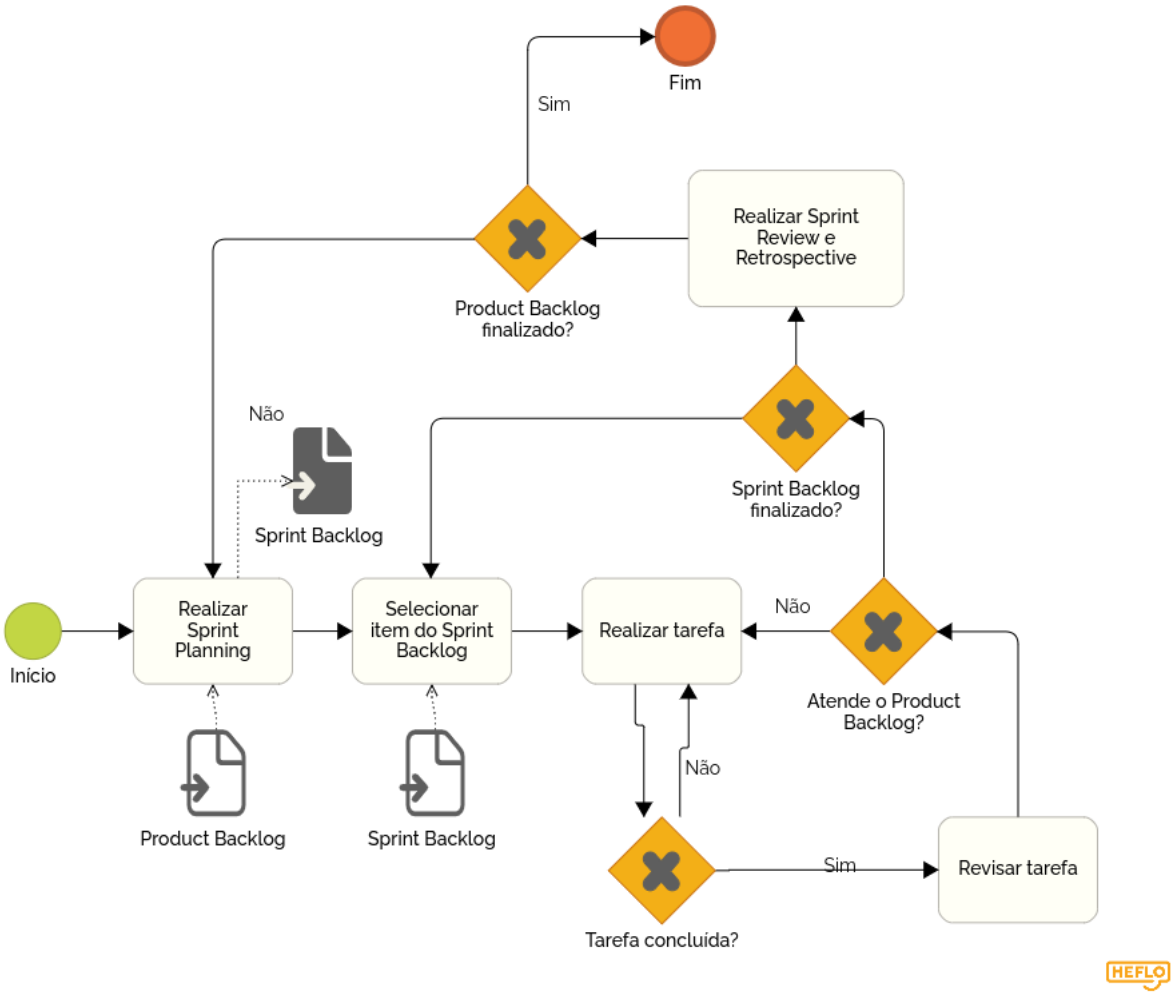
4.4 Metodologia de Desenvolvimento

A metodologia de desenvolvimento prático da aplicação foi definida, com critério de afinidade da dupla, e combinação de algumas práticas de três métodos ágeis, sendo eles: *Scrum*, *Kanban* e *Extreme Programing* (XP). Todas elas foram explicadas na Seção 2.5.1, e suas principais práticas foram adaptadas em função do perfil do trabalho.

4.4.1 Processo de Desenvolvimento

O processo de desenvolvimento escolhido está disponível na Figura 12, elaborado com base na notação BPMN, como um detalhamento do subprocesso Desenvolvimento da Aplicação, ilustrado na Figura 11, contendo as seguintes etapas:

Figura 12 – Fluxo de Atividades do Subprocesso Desenvolvimento da Aplicação *Setter*



Fonte: Autores

1. **Realizar *Sprint Planning***: a cada iteração, foi realizado um planejamento das tarefas a serem cumpridas durante a iteração. Essas tarefas foram retiradas do *Product Backlog*, e a seleção levou em consideração a priorização das histórias de usuário; a capacidade produtiva dos autores, e a gestão de riscos para a entrega do produto. Por fim, tem-se definido o *Sprint Backlog*;

2. **Selecionar item do *Sprint Backlog***: com o *Sprint Backlog* definido, os autores foram alocados nas tarefas disponíveis;

3. **Realizar a tarefa**: atividade de desenvolvimento das tarefas selecionadas durante a *Sprint*;

4. **Revisar tarefa**: atividade de revisão realizado pelo autor que não fez parte do desenvolvimento da tarefa, que fica responsável por analisar a entrega, entendendo se ela atendeu o que fora definido no *Product Backlog*, e

5. **Realizar *Sprint Review* e *Retrospective***: no final da *Sprint*, os autores realizaram essa atividade de revisão a fim de elencar os pontos positivos e negativos da

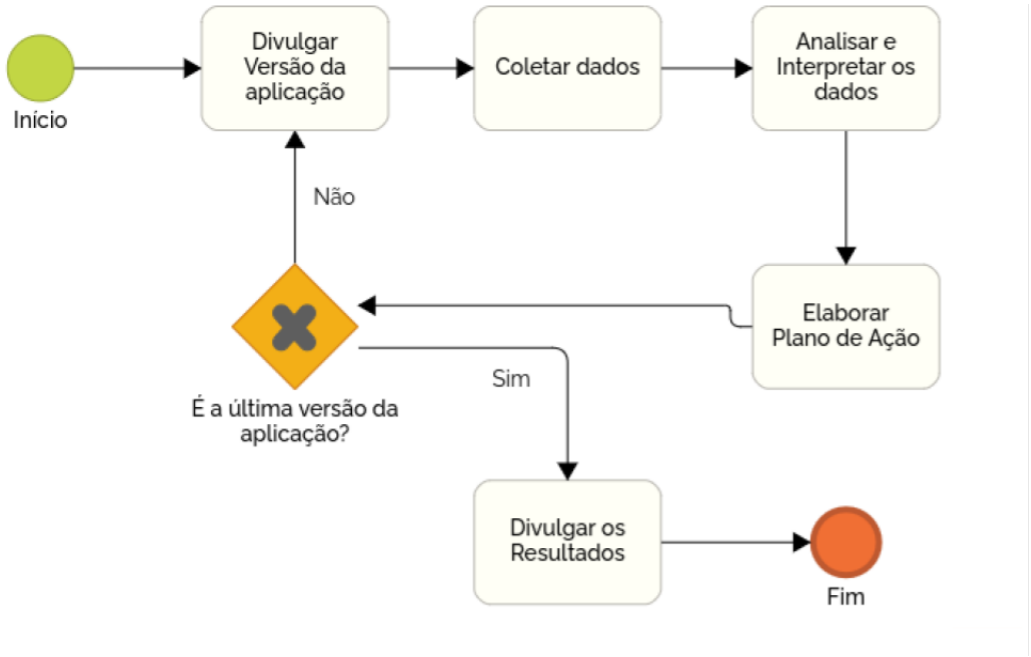
iteração, além de levantar pontos de melhoria para a próxima iteração.

4.5 Metodologia de Análise de Resultados

Como foi definido na seção 4.1.4, o método orientado a processo de análise de resultados do trabalho foi o Pesquisa-Ação. Este método possui nuances variadas entre as fases, que são determinadas pela dinâmica do grupo de pesquisadores com seu objeto de estudo, sendo flexível a adaptações (GERHARDT; SILVEIRA, 2009).

O protocolo utilizado neste trabalho foi adaptado, e pode ser visualizado na Figura 13, e a pesquisa-ação utilizada neste trabalho foi guiada de acordo com as seguintes etapas:

Figura 13 – Fluxo de Atividades do Subprocesso Versionamento e Análise da Aplicação *Setter*



Fonte: Autores

1. **Divulgar Versão da aplicação**: consiste na divulgação da última versão disponível para os *stakeholders* terem a possibilidade de usarem e testarem a aplicação;

2. **Coletar dados**: consiste na coleta dos dados qualitativos por meio de *feedbacks* feitos pela interação dos *stakeholders* com a aplicação;

3. **Analisar e interpretar os dados**: consiste na análise dos dados coletados e a realização de uma explanação dos resultados adquiridos;

4. **Elaborar plano de ação**: consiste na elaboração de um plano de ação que visou mitigar os problemas encontrados a partir dos dados analisados, e

5. **Divulgar resultados**: consiste na divulgação dos resultados obtidos e do plano de ação definido. Essas artefatos foram documentados e validados com os *stakeholders*, sendo apresentados à banca examinadora via essa monografia e apresentação oral.

4.6 Cronograma

Os Quadros 9 e 10 apresentam os cronogramas da primeira etapa e da segunda etapa do TCC, respectivamente. Nas tabelas, as atividades são descritas de acordo com suas datas de implementação, baseando-se no fluxo de atividades descrito na seção 4.2.

Quadro 9 – Cronograma de Atividades/ Subprocessos da Primeira Etapa do TCC

Atividade/ Subprocessos	**Out/22**	**Nov/22**	**Dez/22**	**Jan/23**	**Fev/23**
Definir Tema	X				
Formular Proposta		X			
Contextualizar Problema		X			
Levantamento Bibliográfico			X		
Definir Suporte Tecnológico			X		
Definir Metodologia			X		
Definir *Backlog*			X		
Desenvolver Protótipo da Aplicação				X	X
Revisar TCC1					X
Apresentar TCC1					X

Fonte: Autores

Quadro 10 – Cronograma de Atividades/ Subprocessos da Segunda Etapa do TCC

Atividade/ Subprocessos	**Mar/23**	**Abr/23**	**Mai/23**	**Jun/23**	**Jul/23**
Corrigir Apontamentos da Banca	X				
Desenvolvimento da Aplicação	X	X	X	X	
Divulgação das Versões e Análise de Resultados				X	
Revisar TCC2					X
Apresentar TCC2					X

Fonte: Autores

4.7 Considerações Finais do Capítulo

Este capítulo buscou salientar sobre as escolhas metodológicas definidas para o desenvolvimento deste Trabalho de Conclusão de Curso. Assim, foram evidenciadas as metodologias da fase de pesquisa, de desenvolvimento da proposta, e de análise de resultados, identificando e categorizando os pontos mais relevantes em cada uma delas. Como classificação, a pesquisa é de abordagem qualitativa, natureza aplicada, objetivos exploratórios, e tendo pesquisa bibliográfica e pesquisa-ação como procedimentos. Na sequência, foram apresentados os fluxos de atividades que orientaram o desenvolvimento do trabalho

com suas respectivas descrições. Por fim, foram apresentados o cronograma de realização do trabalho, separando-os de acordo com suas etapas, primeira e segunda etapas do TCC.

5 Setter

Este capítulo visa apresentar, com o apropriado nível de detalhamento, o *Setter*, que é uma ferramenta focada nos processos de gerenciamento de times amadores de voleibol. Neste aspecto, em um primeiro momento e dentro da seção de Contextualização, são retomadas diretrizes contextuais que incluem conceitos e características já abordados anteriormente no texto. Na sequência, com o intuito de apresentar adequadamente a aplicação, a seção que explica Sobre a Ferramenta *Setter* fornece uma visão detalhada acerca da aplicação ***Setter***, sucedida pela seção do *Backlog* do Produto, que expõe e elucida o *Backlog* produzido para modelar os requisitos de *software* da aplicação. Ademais, a seção de Arquitetura da Solução apresenta a descrição arquitetural da ferramenta em diferentes níveis, sendo estes a Visão Lógica e a Visão de Dados. Em adição à Arquitetura de *Software*, é disposta a seção responsável por apresentar a Modelagem da Solução. Posteriormente, a identidade visual da aplicação é apresentada pela seção Identidade Visual e a Implantação da Ferramenta *Setter* . Por fim, têm-se um breve detalhamento do Processo de Contribuição da Aplicação seguido das colocações conclusivas do capítulo na seção de Considerações Finais do Capítulo.

5.1 Contextualização

A principal justificativa para o desenvolvimento da ferramenta *Setter* diz respeito a contemplar os principais processos gerenciais em contextos de times amadores de voleibol. Motivada pela experiência empírica de um dos autores, que possui uma sólida participação competitiva em times desportivos fora do contexto profissional, a ideia da aplicação surgiu como uma proposta de solucionar gargalos administrativos substanciais deste contexto.

Neste sentido, a aplicação visa tirar proveito de aparatos tecnológicos amplamente difundidos entre a população para auxiliar os processos inerentes ao cargo de gestor desportivo. Ren et al. (2021) apontam que a utilização de aplicações *web* é uma prática que ganhou um notório prestígio ao longo do tempo, sendo capaz de promover a integração de várias pessoas. Portanto, é vantajoso que a ferramenta seja disponibilizada através de um *web site* especializado.

Conforme acordado anteriormente, o gestor desportivo deve ser capaz de desempenhar uma série de atividades intrínsecas ao cargo e que são de natureza multidisciplinar. Marcado pelo planejamento organizacional, gestão financeira e organização geral de recursos, o gestor desportivo encontra em suas atividades uma dificuldade em adaptá-las ao formato oferecido por aplicações GTD disponíveis no mercado (GREGG, 2015).

Por ser um domínio pouco explorado entre os aplicativos de gerenciamento, o ***Setter*** é uma ferramenta que viabiliza e semiautomatiza as principais responsabilidades do gestor por meio da aplicação de técnicas bem definidas da Engenharia de *Software*.

5.2 Sobre a Ferramenta *Setter*

A ferramenta ***Setter*** trata-se de uma aplicação *web* que possui o objetivo de facilitar a atuação do gestor esportivo no contexto de times amadores de voleibol, de forma que alguns dos principais obstáculos enfrentados por este público alvo sejam mitigados a partir de uma solução computacional disponibilizada por intermédio da *World Wide Web* (WWW).

O nome da aplicação deu-se a partir da tradução livre do termo em português **Levantador** para o inglês ***Setter***. O levantador é o nome de uma das posições fundamentais dentro de um time de voleibol. O idioma escolhido para o nome é o idioma oficial dos Estados Unidos da América, país berço[1] do voleibol como esporte.

O levantador assume um papel de pivô dentro de um time de voleibol, sendo o responsável por construir em estágios iniciais de jogadas e, consequentemente, coordenar o funcionamento do time dentro de quadra. Ao realizar um paralelo com este fato, a aplicação, que tem como objetivo coordenar e gerir o time fora de quadra, recebeu o nome em homenagem à posição.

Na aplicação *Setter*, é possível realizar atividades inerentes a cargos organizacionais de times desportivos, principalmente no contexto não profissional. As funcionalidades abrangidas pela ferramenta foram obtidas como decorrência da aplicação de técnicas descritas pela Engenharia de Requisitos. Em destaque, apresentam-se os resultados obtidos por meio da técnica de questionário, os quais são tratados de forma mais aprofundada no Apêndice A.

O sistema permite aos gestores de time criarem uma conta para administrar toda a organização de seu time por intermédio do *Setter*. Dentre as atividades organizacionais, destacam-se:

- **Gerenciamento de Atletas**: O sistema permite que os gestores administrem os dados do corpo de atletas do time, de modo que cada esportista tenha seus dados atualizados de maneira individual para se montar a visualização de dados coletiva;
- **Gerenciamento dos dados do time e técnico**: O sistema fornece apoio para armazenar os dados pertinentes do time, e do técnico, sendo estes pontos muito relevantes para controle do time desportivo, e

[1] Disponível em <https://origemdascoisas.com/a-origem-do-voleibol/>. Acesso em: 30 jan. 2022.

- **Registro de Caixa**: O sistema abrange as funcionalidades, inerentes a esse registro, no intuito de facilitar o controle financeiro sobre o caixa monetário do time;

Essas atividades são consideradas as mais relevantes para se realizar uma boa organização do time, pois abrangem gargalos que normalmente os times possuem ao gerirem os integrantes do time. Com isso, todas essas funcionalidades estão dentro do escopo do MVP da aplicação *Setter*.

Além disso, existem outras atividades que auxiliam no gerenciamento dos times amadores, sendo elas mapeadas para futuras versões do *Setter:*

- **Gerenciamento de Gestores**: O sistema irá permitir que os gestores atuem como membros administradores do time ao incluir ou remover outros administradores;
- **Controle de Uniformes**: O sistema irá abranger as funcionalidades, inerentes a esse controle, no intuito de facilitar a gestão de números de cada jogador e quais camisas poderão ser utilizadas em cada jogo;
- **Controle de Presença**: O sistema fornecerá apoio para realizar esses pontos, sendo os mesmos muito relevantes para controle do time desportivo;
- **Calendário de Jogos**: O sistema irá fornecer uma visualização temporal para o registro de jogos e, em adição, uma análise específica para os desempenhos de cada um dos jogos, e
- **Comunicação**: O sistema irá tratar um conjunto de funcionalidades responsável por atividades de lembrete e comunicação automatizada para os membros do time.

Para detalhar com mais profundidade cada um dos requisitos levantados para a aplicação, é apresentado o *Backlog* do Produto, artefato que fornece uma visão de alto nível da distribuição de requisitos, além de permitir a especificação da priorização destes. Entretanto, antes mesmo de comentar sobre o *Backlog*, há necessidade de esclarecer que o mesmo não foi elaborado sem conferir outras aplicações de cunho similar. Sendo assim, segue um *benchmarking*, que foi fundamental para a adequada especificação de épicos, *features*, histórias de usuários e priorização.

5.3 *Benchmarking*

O *benchmarking* pode ser definido como o processo contínuo e sistemático utilizado para investigar os resultados alcançados de unidades produtivas com processos e técnicas

similares em termos de gestão. Isto é, trata-se de um processo de identificação, compreensão e adaptação de práticas excelentes em organizações para auxiliar uma determinada organização a melhorar seu desempenho (OLIVEIRA et al., 2003).

Com isso, foi desenvolvido um *Benchmarking* com o objetivo de analisar e comparar os recursos das principais soluções encontradas para o domínio em questão.

Foram encontradas algumas aplicações voltadas ao público alvo, porém apenas duas delas possuíam o intuito de administrar um time esportivo (PLAYERPLUS, 2023; TEAMER, 2023), tendo as outras um cunho mais voltado para uma rede social de times.

Foram analisadas as seguintes *features* e particularidades:

- **Dados do Time**: *Feature* capaz de armazenar os principais dados relacionados ao time;
- **Dados dos atletas**: *Feature* capaz de armazenar os principais dados relacionados aos atletas que compõem o time;
- **Cadastro completo de atletas**: *Feature* capaz de armazenar dados completos dos atletas, podendo ser utilizados para inscrições em campeonatos, entre outros;
- **Dados do Técnico**: *Feature* capaz de armazenar os principais dados relacionados ao técnico do time;
- **Gestão de Administradores**: *Feature* capaz de gerir os administradores do time;
- **Gestão de Faltas**: *Feature* capaz de gerir as faltas em treinos, campeonatos e justificativas de ausências de cada atleta;
- **Gestão de Uniformes**: *Feature* capaz de gerir a logística de uniformes do time, podendo ser uniformes individuais ou coletivos;
- **Fluxo de Caixa**: *Feature* capaz de gerir as entradas e saídas de caixa do time;
- **Pagamento do Técnico**: *Feature* capaz de registrar pagamentos do técnico do time;
- **Lembretes**: *Feature* capaz de criar lembretes personalizados de acordo com as necessidades do time;
- **Histórico de Jogos**: *Feature* capaz de registrar os jogos realizados pelo time, assim como seu resultado, e
- **Aplicação *web***: Não se trata de uma funcionalidade, mas sim se a aplicação é *mobile* ou *web*.

A Figura 14 demonstra os resultados obtidos com a análise das duas aplicações em questão, *Teamer* (TEAMER, 2023) e *PlayerPlus* (PLAYERPLUS, 2023) e enfatiza a diferença que elas possuem para a aplicação Setter.

Figura 14 – *Benchmarking* das Funcionalidades das Aplicações

Funcionalidade	Setter	Teamer	PlayerPlus
Dados do Time			
Dados dos atletas			
Cadastro completo de atletas			
Dados do Técnico			
Gestão de Administradores			
Gestão de Faltas			
Gestão de Uniformes			
Fluxo de Caixa			
Pagamento do Técnico			
Lembretes			
Histórico de Jogos			
Aplicação *web*			

Legenda	
	Possui funcionalidade
	Não possui funcionalidade

Fonte: Autores

5.4 *Backlog* do Produto

O *Backlog* do Produto, conforme acordado anteriormente, define-se como um artefato que concentra a lista de tarefas, estas comumente modeladas como histórias de usuário necessárias para a compreensão das funcionalidades desejadas para o produto em questão (SUTHERLAND, 2010). É válido ressaltar que o **Backlog** é sequenciado a partir da prioridade de seus itens, analisados de maneira individual.

Para realizar a priorização, atividade esta que busca materializar a ordem prioritária de requisitos de *software*, foi utilizada a técnica ***MoSCoW***. O procedimento utilizado consiste no mapeamento dos requisitos em quatro níveis de abstração prioritários, destacados em ordem descendente de valor agregado ao longo do acrônimo da técnica. Ademais, tal técnica de priorização é usada em gerenciamento e análise de negócios, ou projetos de desenvolvimento de *software*, para convergir em um entendimento comum entre as diferentes partes interessadas sobre a importância atribuída à cada entrega de versão (MIRANDA, 2022).

A Figura 15 apresenta o *Backlog* do Produto priorizado para o aplicativo *Setter*. Além da priorização *MoSCoW*, o *Backlog* está segmentado em cores, as quais representam as *releases* de *software* planejadas para o produto, sendo estas explicadas pela legenda contida na Figura 16. As *releases* planejadas orientam-se por versões. Essas versões são idealizadas, considerando, cada qual, um conjunto de funcionalidades que agrega valor à aplicação. Seguem as principais versões planejadas:

- **MVP**: Versão, representada em laranja, que contém o conjunto de funcionalidades que compõem o mínimo produto viável do projeto;

- **V2 - Gerenciamento**: Segunda versão a ser distribuída, representada em verde, e que agrupa as funcionalidades que tratam os aspectos de gerenciamento de times desportivos amadores de voleibol;
- **V3 - Comunicação**: Terceira versão a ser distribuída, representada em lilás, e que abrange as funcionalidades que facilitam a comunicação entre gestores e atletas, e
- **V4 - Jogos**: Versão, representada em vermelho, que finaliza o escopo previsto pelo *Backlog* proposto para o presente trabalho. Esta versão concentra funcionalidades que lidam com o histórico e a análise de jogos dos times desportivos.

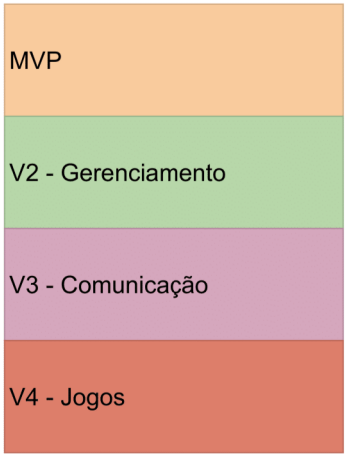
Com a finalidade de uma melhor visualização do *Backlog*, este artefato é reapresentado no Apêndice B com uma melhor qualidade de imagem, e mais espaço para apresentação.

Figura 15 – *Backlog* do Produto *Setter*

Épicos	Features	US	Priorização
Administrador	Conta	Eu como gestora de um time amador desejo criar uma conta no sistema para que eu possa criar meu time e administrá-lo.	MUST
		Eu como gestora de um time amador já cadastrado desejo entrar em minha conta com minhas credenciais para que eu possa ter acesso à organização do meu time.	MUST
	Gerenciamento de Admins	Eu como gestora de um time amador já cadastrado desejo enviar links de convite para novos usuários por meio do email para fornecer acesso à organização do time.	SHOULD
		Eu como gestora de um time amador já cadastrado desejo excluir outro administrador que tenha saído do time para que eu possa manter os dados seguros entre os administradores ativos do time. (OBS: uma vez que um admin é excluído, a conta também será).	SHOULD
		Eu como gestora de um time amador já cadastrado desejo visualizar todos os outros admistradores do time para que eu possa saber todos os que têm acesso às informações.	COULD
Time	Gerenciamento de atletas	Eu como gestora desejo incluir novas atletas na organização do meu time para manter um registro das atletas do time.	MUST
		Eu como gestora desejo alterar os dados de uma atleta do meu time em caso de alguma alteração em dado cadastral.	MUST
		Eu como gestora desejo visualizar uma lista com todas as atletas que fazem parte do atual corpo de atletas da equipe.	MUST
		Eu como gestora desejo desativar/reativar uma atleta da organização da minha equipe em caso de jogadoras que não fazem mais parte do corpo de atletas da equipe, ou caso queiram voltar a fazer parte.	MUST
	Dados do time	Eu como gestora de um time amador desejo cadastrar todos os dados do time para que eu possa ter acesso à essas informações quando necessário.	MUST
		Eu usuário desejo adicionar os dados do técnico do time para que eu possa ter acesso à essa informação quando necessário.	MUST
	Controle de presença	Eu como usuário desejo adicionar os dias da semana que possuem treino para que eu possa gerar tabelas de presenças em cada treino.	SHOULD
		Eu como usuário desejo gerar uma tabela de presença mensal para que eu possa fazer o controle de faltas do time.	COULD
		Eu como usuário desejo categorizar os tipos de falta para que eu possa entender os motivos das faltas de cada atleta.	SHOULD
	Uniforme	Eu como usuário desejo cadastrar todos os uniformes que o time possui para que eu possa saber a quantidade de uniformes e os números disponíveis.	COULD
		Eu como usuário desejo atualizar o status do uniforme para usado por algum atleta para que eu possa saber quais uniformes não estão guardados e com quem estão.	COULD
Financeiro	Mensalidade	Eu como usuário desejo adicionar o valor da mensalidade para que eu possa fazer o controle de pagamento.	MUST
		Eu como usuário desejo selecionar o atleta para o pagamento de mensalidade do mês para que eu possa fazer o controle das atletas que pagaram a mensalidade.	MUST
	Controle de Caixa	Eu como usuário desejo gerar tabelas de fluxo de caixa por mês para que eu possa adicionar as entradas e saídas daquele mês.	MUST
		Eu como usuário desejo adicionar entradas e saídas de caixa para que eu possa manter o valor total sempre atualizado.	MUST
		Eu como usuário deseja adicionar o pagamento de técnico ativo no mês para que eu possa manter o fluxo de caixa atualizado.	MUST
Comunicação	Lembretes	Eu como atleta do time gostaria de receber um lembrete acerca da data de pagamento da mensalidade para não atrasar o pagamento.	COULD
		Eu como atleta do time gostaria de receber um lembrete sobre o ocorrimento dos treinos para não perder nenhum dia de treinamento.	COULD
		Eu como atleta do time gostaria de receber um lembrete sobre os jogos que estão próximos de ocorrer para não faltar.	COULD
	Envio de Avisos	Eu como gestora desejo escrever e enviar mensagens à todas as atletas da minha equipe esportiva. (OBS: a mensagem deve ser propagadas por meio dos meios de comunicação cadastrados para atleta).	SHOULD
		Eu como gestora desejo escrever e enviar mensagens à algumas atletas específicas da minha equipe esportiva.	COULD
Jogos	Calendário do Time	Eu como gestora desejo cadastrar futuros jogos para manter um controle de quais partidas minha organização vai disputar.	COULD
		Eu como gestora desejo cadastrar os treinos da minha equipe para manter um controle de treinamentos. (OBS: é possível marcar opções de recorrência para os treinos, de modo que não seja necessário o cadastro de um novo treino para cada dia).	COULD
	Registro dos Jogos	Eu como gestora desejo incluir informações de um jogo já ocorrido para atualizar o registro dos jogos. (OBS: só é possível alterar as informações de uma partida recente, isto é, partidas que ocorreram nos últimos 7 dias contados a partir do momento da utilização do app).	COULD
		Eu como gestora desejo visualizar uma lista com o histórico de jogos disputados pela minha organização esportiva.	COULD
		Eu como gestora desejo visualizar o detalhamento de um jogo específico da minha equipe para obter mais detalhes da partida.	COULD

Fonte: Autores

Figura 16 – Legenda em Cores das Versões do *Setter*



Fonte: Autores

5.5 Arquitetura da Solução

A Arquitetura de *Software* do *Setter* é formada pela integração de três camadas, conforme observado no esquema arquitetural do sistema ilustrado pela Figura 17, as quais são:

- **Camada de Apresentação**: Camada responsável por implementar os leiautes definidos para a visualização do sistema pelos usuários finais;
- **Camada de Aplicação**: Camada que concentra e engloba o código da Interface de Programação da Aplicação, e
- **Camada de Dados**: Camada responsável pela persistência e pela integridade dos dados provenientes da execução do sistema.

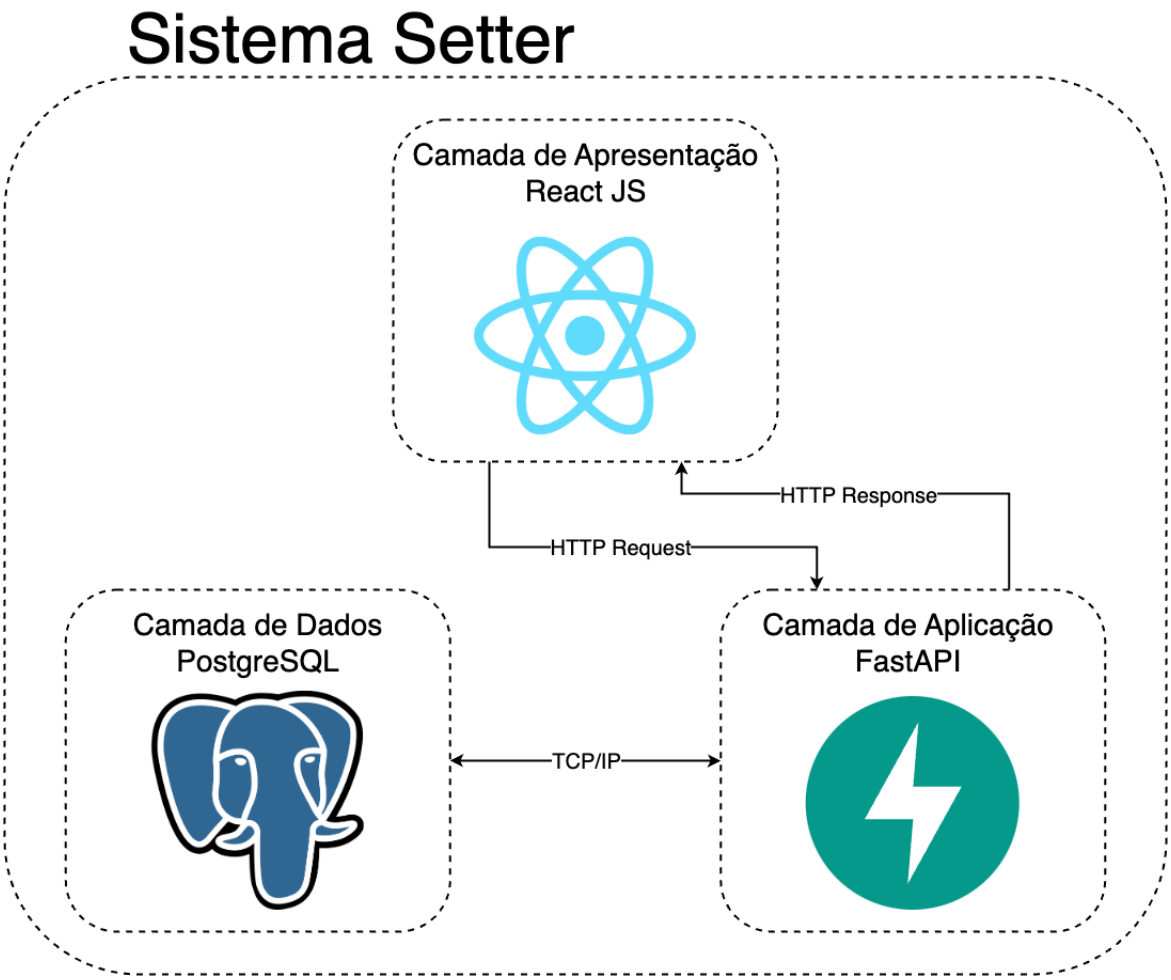
Devido ao uso do arcabouço tecnológico, com tecnologias como *FastAPI* e *ReactJS*, conforme descrito no Capítulo de Suporte Tecnológico, a arquitetura da ferramenta *Setter* compreende o uso de mais de um estilo arquitetural, com destaque à combinação dos estilos Cliente-Servidor, Arquitetura Limpa e N-Camadas, mais precisamente três camadas. Por exemplo, o cliente comunica-se com a *FastAPI* (*Backend*) via *REST Interface*, isto é, com protocolo *Hypertext Transfer Protocol* (HTTP), ou então com protocolo *Hypertext Transfer Protocol Secure* (HTTPS), e métodos baseados em verbos como *Post* e *Get*. Nesse sentido, já se evidencia a relação Cliente-Servidor.

O uso combinado desses estilos arquiteturais apresenta melhorias em termos de escalabilidade e portabilidade, quando comparado com outras propostas de desenvolvimento mais tradicionais. Nessas propostas tradicionais, mantém-se a maior parte da lógica do aplicativo centralizada no Cliente, sendo essa uma estratégia que comumente resulta em sobrecarga de operações e eventual exaustão de processamento. Já na arquitetura proposta para a ferramenta *Setter*, devido à exclusão da restrição de desempenho do lado do Cliente, tem-se maior possibilidade de escalabilidade (VALENTE, 2020).

5.5.1 Camada de Apresentação

O ReactJS representa a camada de apresentação para o aplicativo. A ferramenta consiste em uma biblioteca *JavaScript*, ou *TypeScript*, utilizada para desenvolver aplicações executáveis nos principais navegadores *web* disponíveis. A presente camada é responsável por lidar com todos os eventos de interação do usuário com o sistema, além de se comunicar com a Camada de Aplicação por meio de chamadas projetadas pelo protocolo *Hypertext Transfer Protocol* (HTTP).

Figura 17 – Arquitetura do Sistema



Fonte: Autores

5.5.2 Camada de Aplicação

Por sua vez, a Camada de Aplicação é construída a partir da ferramenta FastAPI, um *framework web* baseado na linguagem de programação *Python* e focado na construção de sistemas com adequado desempenho, escaláveis e robustos. Este *framework* é encarregado de prover toda a estrutura necessária para o desenvolvimento de aplicações *REST*, desde o roteamento até a implementação de métodos baseados em verbos *Hypertext Transfer Protocol* (HTTP). Devido à dependência direta do *framework* com ferramentas de tipagem para a linguagem dinâmica, a ferramenta *Setter* utiliza-se de diretrizes baseadas em padrões de desenvolvimento.

Desta forma, a Camada de Aplicação, que é composta pela *Application Programming Interface* (API), é responsável por duas linhas de comunicação estruturadas em diferentes protocolos: com a Camada de Apresentação, protocolada pelo *Hypertext Transfer Protocol* (HTTP), e com a Camada de Dados, procolada pelo *Transmission Control Protocol/Internet Protocol* (TCP/IP). A comunicação com a Camada de Apresentação é

baseada no processamento das regras de negócio. Processamento esse disparado por interações do usuário com a ferramenta *Setter*. Por sua vez, a comunicação com a Camada de Dados fica responsável por promover o armazenamento dos dados com a ferramenta *Setter*.

5.5.3 Camada de Dados

A tecnologia que compõe a Camada de Dados, por sua vez, é o sistema gerenciador de banco de dados relacionais PostgreSQL. Esse gerenciador fornece o devido suporte à integridade dos dados, e promove eficiência durante os processos de consulta e armazenamento dos dados, visto que o PostgreSQL implementa uma série de facilitadores descritos por modelos relacionais. Modelagens relacionadas à especificação dessa camada serão apresentadas adiante.

5.6 Identidade Visual

A identidade Visual é um conjunto de elementos gráficos e visuais que são utilizados para formular a estilização padrão dos leiautes de interface do sistema. Desta forma, este tipo de documentação define uma série de disposições normativas que guiam o desenvolvimento visual, como a Logotipo, a Tipografia e a Paleta de Cores do projeto. Cada um dos artefatos citados é destacado nas subseções seguintes.

5.6.1 Logotipo

Figura 18 – Logotipo da ferramenta *Setter*

Fonte: Autores

O logotipo do sistema, evidenciado pela Figura 18, é a captura do momento em que uma bola de voleibol é erguida ao ar em consequência ao movimento feito pela mão de um(a) atleta. A logotipo busca transmitir a ideia central proveniente do nome fornecido à aplicação.

5.6.2 Tipografia

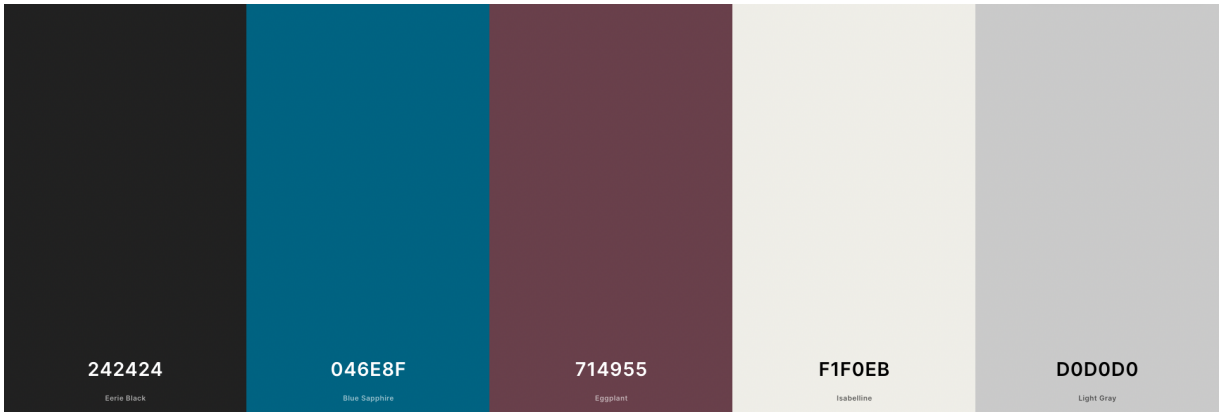
Figura 19 – Tipografia

Fonte: Autores

Para manter a integridade visual na formatação de textos localizados em diferentes partes da ferramenta *Setter*, foi utilizada a fonte *Inter*, que apresenta um visual moderno e objetivo, características que contribuem para a construção do cunho cognitivo da aplicação. Já para títulos e elementos de destaque, foi utilizada a fonte *Poppins*, para fornecer o realce necessário aos componentes. A tipografia do *Setter* é sintetizada na Figura 19.

5.6.3 Paleta de Cores

Figura 20 – Paleta de Cores da Aplicação



Fonte: Autores

A paleta de cores da ferramenta *Setter* é composta por cinco cores que são apresentadas na Figura 20. Foi definida como principal cor do sistema a cor *Eggplant*. Já as cores secundárias são definidas pelas cores *Eerie Black*, *Blue Sapphire*, *Isabelline* e *Light Gray*, que são usadas para complementar e realçar os diversos elementos que compõem os leiautes da aplicação. O contraste correto entre o fundo e o texto promove acessibilidade, incluindo esse critério qualitativo nos elementos gráficos da ferramenta *Setter*. O

contraste entre as cores foi testado com o apoio da ferramenta *Coolors Contrast Checker* [2], e apresentou um total de 13.60, resultado este que está alinhado com as normas de *Web Content Accessibility Guidelines* (WCAG) (CALDWELL et al., 2008).

5.6.4 Protótipo de Alta Fidelidade

O protótipo de alta fidelidade é uma representação detalhada e funcional de um *software*. Trata-se de uma etapa importante do processo de *design* de *software*, que permite que as funcionalidades do sistema sejam testadas antes de devidamente implementadas. O protótipo também permite aos *stakeholders* visualizar e interagir com a aparência e as funcionalidades do *software*, ajudando a validar ou ajustar suas expectativas e seus requisitos (RUDD; STERN; ISENSEE, 1996).

O protótipo de alta fidelidade do presente trabalho foi construído com o apoio do sistema Figma, uma ferramenta de *design* de sistemas, e inclui recursos interativos, leiautes e cores fidedignos à implementação. Em suma, o protótipo pode ser usado para simular o comportamento real do *software*. O protótipo da ferramenta *Setter* está disponível para utilização no endereço: <https://www.figma.com/proto/2DOjDecqtug6xg7cC7IF6K/Prot%C3%B3tipo-Setter?node-id=516%3A6995&scaling=scale-down&page-id=37%3A2&starting-point-node-id=516%3A6995>

Para um melhor aprofundamento e apresentação das telas, o Protótipo de Alta Fidelidade da ferramenta *Setter* é disponibilizado em imagens e textos que direcionam o entendimento visual no Apêndice C.

5.7 Modelagem da Solução

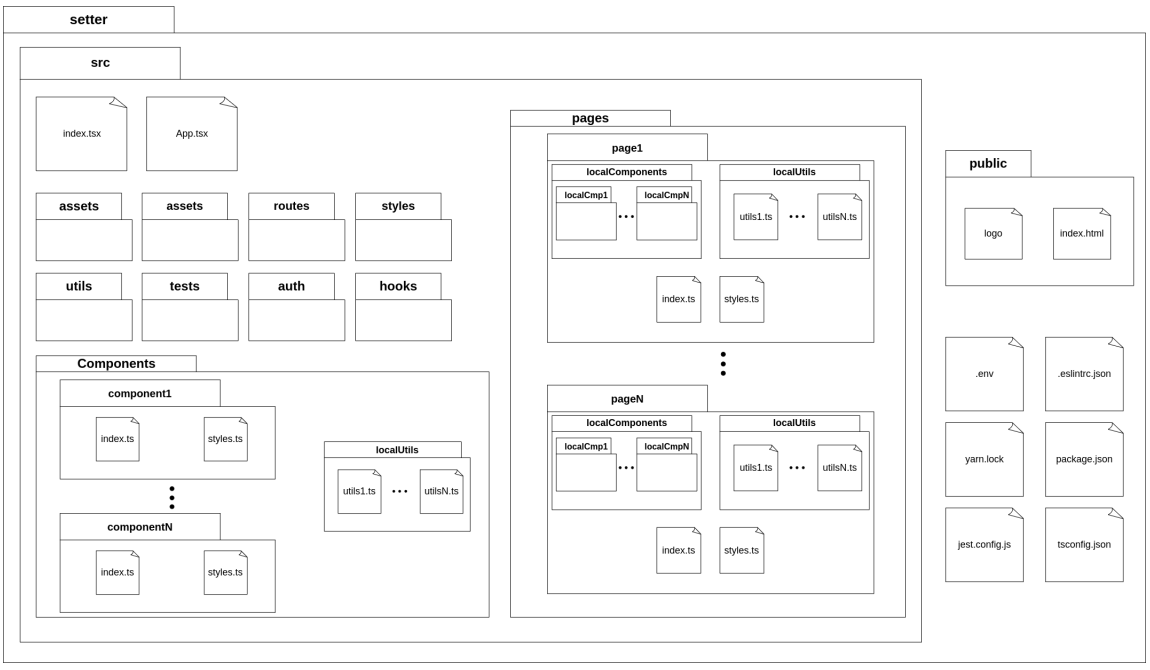
Um documento de arquitetura de uma solução computacional deve ser representado em visões, orientando-se pelo Modelo 5+1 (D'ANUNCIACAO, 2006), e compreendendo as visões: Lógica, Processo, Implementação, Implantação, Dados e Casos de Uso. A visão lógica é a única obrigatória no documento, sendo responsável por apresentar os diagramas e demais detalhamentos da lógica da solução. No caso dessa visão, é comum o uso de Diagramas de Classes e Diagramas de Pacotes. Adicionalmente, quando se tem camada de persistência, uma visão relevante é a visão de dados. Nela, devem constar diagramações como: Modelo de Entidade e Relacionamento e Diagrama Lógico de Dados. Seguem seções com esses detalhamentos para o caso da ferramenta *Setter*.

[2] Disponível em <https://coolors.co/contrast-checker/242424-f1f0eb>. Acesso em: 26 jan. 2023.

5.7.1 Visão Lógica

Na Visão Lógica, o Diagrama de Pacotes é representado como um esquema estático, que fornece a visualização da organização estrutural do projeto a nível de pacotes e componentes de *software*. Permite, dentre outras contribuições, compreender como a solução computacional está organizada. Por se tratar de uma visão coesa e objetiva, este modelo de representação fornece um melhor entendimento dos módulos, quando dispostos em camadas, as quais compõem a solução computacional. Portanto, foram confeccionados diagramas para representar tanto a Camada de Apresentação quanto a Camada de Aplicação, para o caso da ferramenta *Setter*, conforme constam ilustrados, respectivamente, na Figura 21 e na Figura 22. Dentre os pacotes destacados, constam: *pages*, *sections*, *services* e *assets*, que compreendem demais sub pacotes inerentes à Camada de Apresentação, ou seja, a mais próxima do usuário. Adicionalmente, ressalta-se sobre os pacotes *src*, *alembic* e *tests*, que compreendem outros sub pacotes inerentes à Camada de Aplicação, ou seja, a camada intermediária. Aqui, constam, por exemplo, as *models* dos sub pacotes *accounts*, *finances*, *games*, *teams* e *communication*.

Figura 21 – Diagrama de Pacotes da Camada de Apresentação

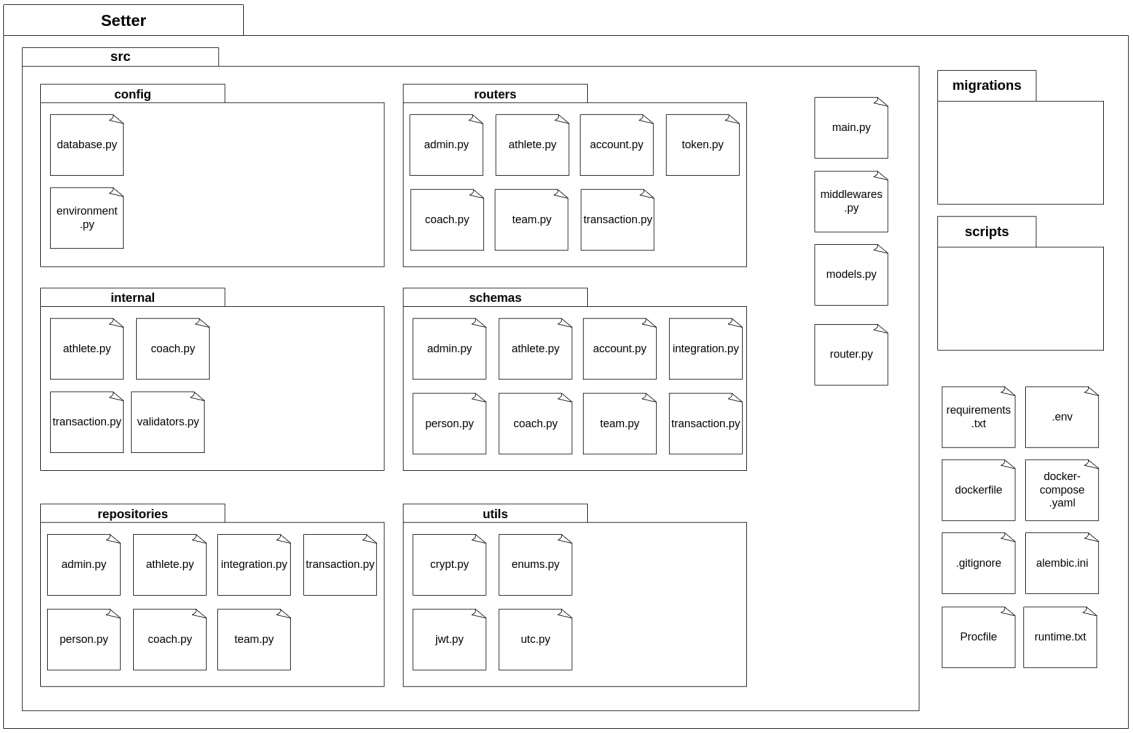


Fonte: Autores

5.7.2 Visão de Dados

A Visão de Banco de Dados compreende diagramas associados à camada de persistência. Como a ferramenta *Setter* terá um banco de dados relacional, cabem modelos orientados aos Bancos de Dados Relacionais. Diagramas de Banco de Dados Relacionais têm como objetivo modelar a estrutura, a qual deve seguir as diretrizes de um modelo

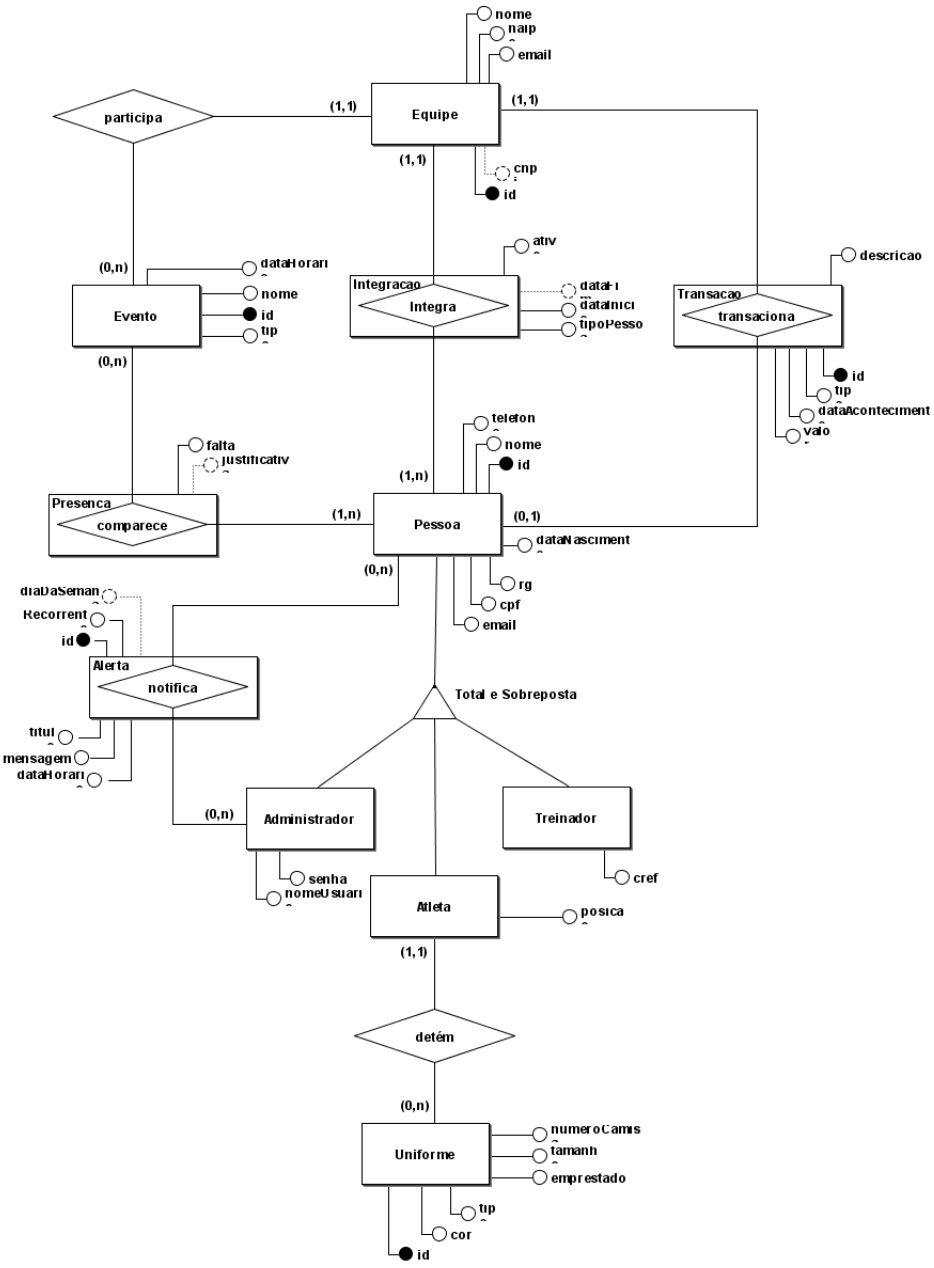
Figura 22 – Diagrama de Pacotes da Camada de Aplicação



Fonte: Autores

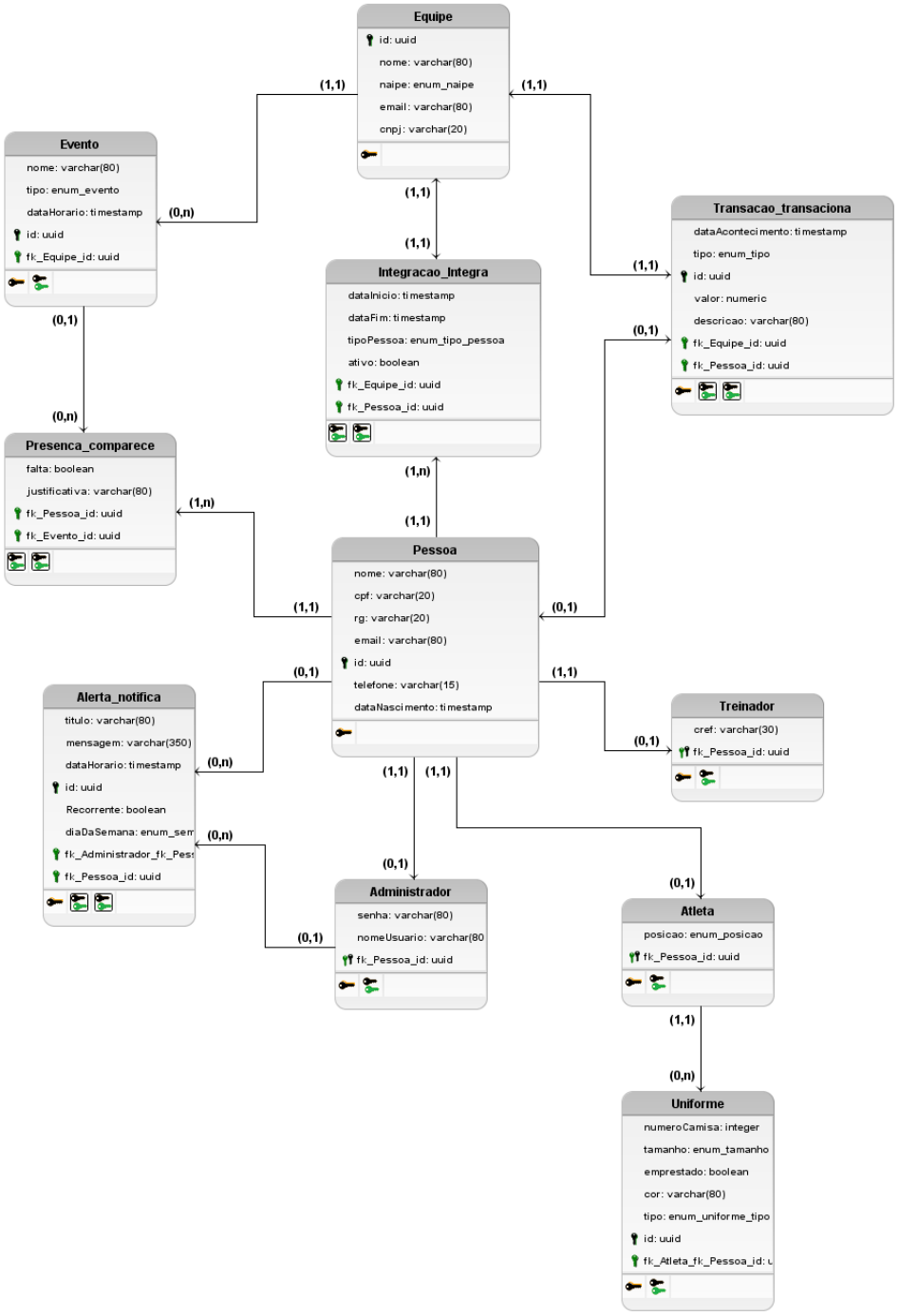
relacional, para armazenamento dos dados. O Diagrama Entidade-Relacionamento (DER) é apresentado na Figura 23, e pode ser conceituado como uma representação gráfica, sendo o principal nível de modelagem para visualização dos relacionamentos entre as diferentes entidades da ferramenta *Setter*. Já o Diagrama Lógico de Dados (DLD), ilustrado na Figura 24, descreve como os dados são armazenados nas estruturas físicas do Sistema de Gerenciamento de Banco de Dados, e como estes se relacionam.

Figura 23 – Diagrama de Entidade-Relacionamento da Aplicação



Fonte: Autores

Figura 24 – Diagrama Lógico de Dados da Aplicação



Fonte: Autores

5.8 Implantação da Ferramenta *Setter*

Conforme discutido previamente na seção de 2.7, a implantação de um *software* pode ser definida como o processo de configuração necessário para a operação do *software* (SOMMERVILLE, 2016). A implantação envolve, ainda, a hospedagem, execução e disponibilização da aplicação de *software*, em conjunto com suas nuances de implementação

em um ambiente produtivo, como, por exemplo, boas práticas de segurança.

Para o contexto da ferramenta *Setter*, a implantação foi realizada com o auxílio da ferramenta Heroku (2023). Esta, por sua vez, é explicada e detalhada na seção 3.3.8. A ferramenta *Heroku* permite hospedagem, configuração, testagem e publicação de projetos virtuais em núvem, tornando o processo de implantação conciso e objetivo.

A ferramenta *Heroku* possui integração direta com o *Github*, plataforma na qual se dispõe o código fonte da ferramenta *Setter*. Desta forma, uma vez implantado o sistema, foi gerado e disponibilizado um domínio próprio para a aplicação, que é evidenciado pelo seguinte *link* para o site *web*: <https://setter-front-d443799d9d71.herokuapp.com/>.

A disponibilização da ferramenta *Setter* foi de extrema importância para a metodologia de Análise de Resultados proposta, isto é, por meio da implantação realizada foi possível realizar a divulgação de versões da aplicação aos *stakeholders*. Desta forma, foi possível dar continuidade para o presente trabalho, de maneira a realizar todo o ciclo da Pesquisa-Ação, que, por sua vez, é abordada no próximo capítulo.

5.9 Processo de Contribuição

Esta seção tem como propósito deixar explícito os processos adotados para a metodologia de desenvolvimento de *software* adotados para o contexto do presente Trabalho de Conclusão de Curso. Isto é, apresentar as nuances do desenvolvimento em dupla, considerando o aparato tecnológico utilizado para a gerência e versionamento do código fonte, da ferramenta *Setter*.

Conforme elaborado anteriormente na seção 3.3.2, a plataforma para armazenamento dos repositórios de código foi o sítio digital Github (2022). A estruturação de repositórios se segmentou na estrutura hierárquica de uma Organização[3] interna que orientou os repositórios entre diferentes níveis:

- Repositório para a camada de apresentação[4];
- Repositório para a camada de aplicação[5], e
- Repositório de documentação[6].

Adicionalmente, o fluxo de atividades necessário para compor o processo de contribuição em código seguiu diretrizes formuladas por Driessen (2010) em seu artigo que

[3] Disponível em <https://github.com/Setter-TCC/>
[4] Disponível em <https://github.com/Setter-TCC/SetterFrontend/>
[5] Disponível em <https://github.com/Setter-TCC/SetterBackend/>
[6] Disponível em <https://github.com/Setter-TCC/SetterDocs>

canonizou a abordagem *Git Flow*. Em síntese, o fluxo de atividades consiste em contribuições segmentadas por requisitos, em que o código relacionado é apartado em uma ramificação de desenvolvimento, termo comumente substituído pela alcunha de *branch*, e adicionado de maneira gradual, atômica e documentada com mensagens relevantes que seguem uma estrutura bem definida.

Uma vez que as modificações relacionadas ao requisito desenvolvido perpassou por todo o ciclo da *Sprint*, processo este destacado pela subseção 4.4.1, o código fonte adjacente foi adicionado às ramificações principais por meio de *Pull Requests*. Por fim, quando o conjunto de requisitos fracionados em entregáveis alcançasse a completude da versão e, estivesse completo em código, este era organizado na estrutura de *Releases*.

Todo o processo de desenvolvimento foi de suma importância para a aplicabilidade de boas práticas da Engenharia de *Software* durante a etapa de construção da ferramenta *Setter*. Por consequência, destaca-se a grande importância das diretrizes de contribuições do projeto, que podem ser visualizadas no documento localizado no seguinte *link*: <https://github.com/Setter-TCC/SetterDocs/blob/main/CONTRIBUTING.md>.

5.10 Considerações Finais do Capítulo

Este capítulo adentrou nas especificidades da ferramenta *Setter*. De início, foi reapresentada a contextualização a respeito da atual situação e dos principais desafios encontrados por times desportivos amadores focadas na prática do voleibol. Na sequência, foi introduzida a solução para a problemática retratada, que é condensada em uma aplicação *web* centrada em processos gerenciais. Tais processos foram abordados detalhadamente, sendo identificados por meio de técnicas da Engenharia de Requisitos, mais precisamente um Questionário.

Adicionalmente, é apresentado um conjunto de artefatos que compõem a arquitetura da solução de *software*, com destaque para cada uma das camadas que são implementadas e como estas se comunicam entre si. Após as explicações sobre a arquitetura de *software*, são evidenciados os principais aspectos visuais da ferramenta *Setter*, por meio da Identidade Visual e do Protótipo de Alta Fidelidade. São fornecidas modelagens, revelando as visões lógica e de dados da ferramenta *Setter*, na sequência tem-se uma breve explicação do processo de implantação do *Setter* e sua importância para o trabalho, e, por fim, é pontuado o processo de contribuição que orientou o desenvolvimento da ferramenta.

6 Análise de Resultados

O presente capítulo tem o intuito de apresentar os resultados deste trabalho, bem como descrever as atividades de análise aplicadas sobre a ferramenta *Setter*, vistas no Capítulo 5, e acordar sobre as ações de melhorias encontradas no MVP, e as novas funcionalidades implementadas. A Análise de Resultados foi conduzida com base na Metodologia de Análise de Resultados, já apresentada anteriormente. Essa estabelece um protocolo de Pesquisa-Ação, o qual compreende as fases de Coleta de Dados, Análise e Interpretação de Dados, que estão expostas em conjunto na seção de Análise e Interpretação dos Dados, seguido da Elaboração do Plano de Ação e da Divulgação dos Resultados, conforme apresentado na sequência. Por fim, são reveladas as Considerações Finais do Capitulo.

6.1 Fases da Pesquisa-Ação

Conforme descrito na Metodologia de Análise de Resultados (seção 4.5), este trabalho segue as fases da Pesquisa-Ação. Em um primeiro momento, quando há o lançamento do MVP da aplicação, consta a fase de Coleta de Dados, sendo essa breve. Logo em seguida, tem-se a fase de Análise e Interpretação dos Dados, onde os dados coletados são analisados de forma qualitativa. Seguindo para a Elaboração do Plano de Ação, é realizado um planejamento para solucionar/mitigar erros e imprecisões apontadas pela fase anterior. Com isso, há as divulgações de novas versões da aplicação, que seguem o mesmo fluxo de coleta e análise de dados. Por fim, tem-se a fase de Divulgação de Resultados, concluindo o protocolo de Pesquisa-Ação.

Vale ressaltar que para este trabalho foi possível a realização da Pesquisa-Ação apenas para o primeiro Ciclo de Desenvolvimento, sendo este representado pela Divulgação da Versão do MVP. Ou seja, os resultados divulgados dizem respeito a essa versão da aplicação.

6.2 Coleta de Dados

Nesta fase, inicia-se o processo da pesquisa, com a identificação do problema e a descrição do contexto em que esse se insere. Problema: tendo em vista o público alvo da aplicação, como viabilizar seu desenvolvimento, considerando um *Minimum Viable Product* (MVP). Contexto: apresentação dos resultados obtidos usando uma abordagem, preferencialmente, qualitativa, ou seja, voltada para a qualidade da interação do usuário com a ferramenta.

Para isso, a coleta de dados foi feita visando reunir, do público alvo, dados com relação aos aspectos de interação, facilidade e satisfação do uso da ferramenta *Setter*. Para tanto, os testes foram aplicados para os *stakeholders* que validaram o Protótipo de Alta Fidelidade criado, e acompanharam o desenvolvimento da aplicação em si. Essa testagem foi feita sem treinamento prévio, justamente para que os dados apresentassem uma maior precisão quanto à intuitividade da aplicação criada.

Nesse contexto, ocorreu, em um primeiro momento, a coleta de *feedbacks*, visando validar a ferramenta *Setter*. Os participantes são os próprios *stakeholders*, atuantes na área de gerenciamento de times de voleibol. Eles possuem experiência prévia dos conceitos inerentes ao escopo do domínio cognitivo desse trabalho, podendo validar o uso da ferramenta *Setter* com mais tranquilidade.

O formulário foi criado pela plataforma do *Google Forms*; ficou disponível para respostas durante 5 dias corridos, e obteve um total de 7 respostas. Vale ressaltar que todos os *stakeholders* que enviaram a pesquisa concordaram com a divulgação dos dados no contexto deste Trabalho de Conclusão de Curso, com termo de consentimento de conhecimento de todos, conforme ilustra a Figura 25. O Termo de Consentimento Livre e Esclarescido completo orientou-se pelo *template* proposto em (ARARAQUARA, 2023), cuja versão adaptada consta no Apêndice E.

Figura 25 – Consentimento dos Participantes do Questionário em Fornecer Dados para o Trabalho

Fonte: Autores

6.2.1 Questionário

Para validar a ferramenta *Setter*, foi criado um questionário com algumas perguntas, obrigatórias ou não, além de um teste de usabilidade, de modo a entender a

experiência dos usuários. O questionário foi dividido em seções, sendo elas:

1. Coleta dos Dados do Administrador;
2. Teste 1: Criação de Conta;
3. Teste 2: *Login* em Conta;
4. Teste 3: Gestão de Atletas;
5. Teste 4: Fluxo de Caixa, e
6. Teste 5: Configurações.

Nessa subseção será mostrada a coleta de dados do administrador, trazendo mais contexto do público alvo. Já nas subseções seguintes serão abordadas as motivações de cada teste realizado, assim como as perguntas que foram feitas para os *stakeholders*. Como são muitas as perguntas feitas, e para um melhor visualização das coletas que foram realizadas, as imagens correspondentes às respostas obtidas nos testes realizados encontram-se no Apêndice D.

6.2.1.1 Coleta dos Dados do Administrador

A seção de Coleta dos Dados do Administrador serviu para identificar quais são as atribuições que um determinado administrador possui no time, e qual o time relacionado a este administrador. Estes tópicos fazem-se necessários, visto que havia, no grupo de *stakeholders*, pessoas que possuíam mais de uma atribuição em seus times de voleibol.

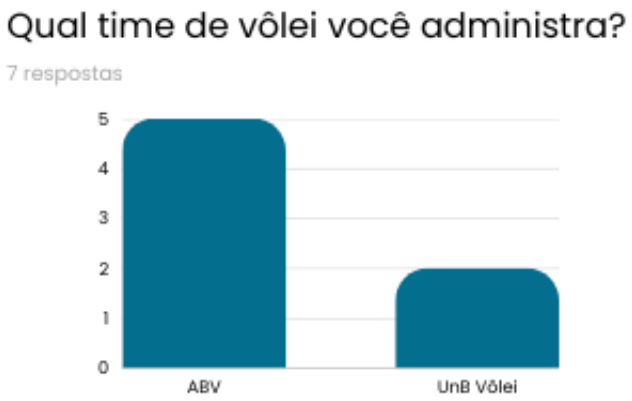
Nesta seção, no questionário, foram indagadas as seguintes perguntas:

1. Qual seu nome?
2. Qual time de vôlei você administra?
3. Qual o seu papel no gerenciamento do time?

A Figura 26 representa a quantidade de administradores por time que responderam ao questionário. Nela, é possível observar que o time ABV possui mais integrantes respondentes, e isso deve-se ao fato desse time ser uma Associação de Voleibol, possuindo times nos naipes feminino adulto, masculino adulto, além de times de base, nas duas categorias, enriquecendo os dados coletados na presente pesquisa. Já os representantes do time UnB Vôlei fazem parte apenas do time feminino da universidade.

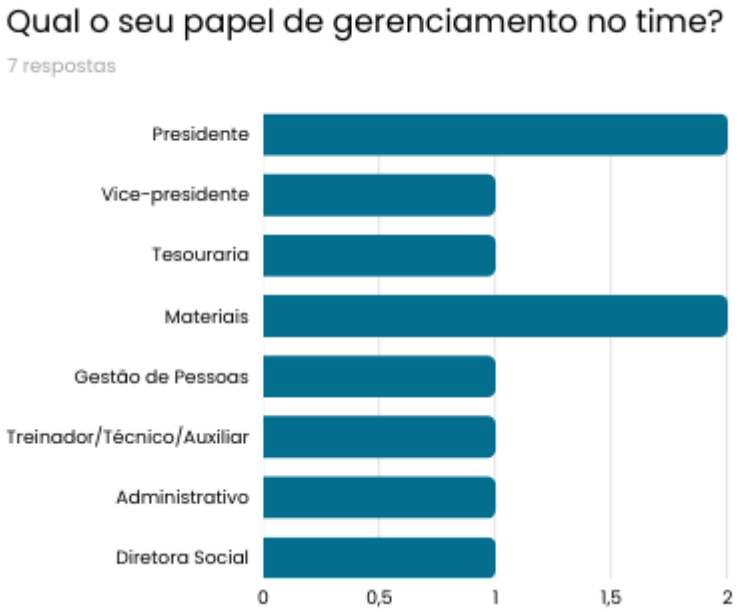
A Figura 27 distribui as atribuições dos *stakeholders* respondentes dentro de seus respectivos times. Com esse resultado, é possível notar que o grupo de *stakeholders* possui

Figura 26 – Quantidade de Administradores por Time Respondentes ao Questionário



Fonte: Autores

Figura 27 – Papel de Gerenciamento dos Administradores Respondentes ao Questionário



Fonte: Autores

papéis diversos, o que enriquece suas respostas, visto que cada *stakeholder* possui um ponto de vista distinto sobre o gerenciamento de seus respectivos times. Vale ressaltar que 70% das atribuições lidam com funcionalidades previstas no escopo do MVP da aplicação, que englobam todas as questões de presidência, tesouraria, gestão de pessoas e técnico, entre outros.

6.2.1.2 Teste 1: Criação de Conta

A seção de Teste sobre Criação de Conta ficou responsável por coletar os *feedbacks* dos *stakeholders* ao criarem suas contas no Setter. Nesta seção, no questionário, foram indagadas as seguintes perguntas:

1. Você encontrou a seção para criar um time?
2. Você teve alguma dificuldade ao criar um time? Se sim, quais?
3. Você sentiu falta de alguma informação? Se sim, quais?
4. Você tem alguma sugestão de melhoria?

6.2.1.3 Teste 2: *Login* em Conta

A seção de Teste sobre *Login* em Conta ficou responsável por coletar os *feedbacks* dos *stakeholders* ao entrarem em suas contas criadas no Setter, usando suas respectivas credenciais. Nesta seção, no questionário, foram indagadas as seguintes perguntas:

1. Você encontrou a seção para realizar o *login*?
2. Você teve alguma dificuldade ao realizar o *login*? Se sim, quais?
3. Você sentiu falta de alguma informação? Se sim, quais?
4. Você tem alguma sugestão de melhoria?

6.2.1.4 Teste 3: Gestão de Atletas

A seção de Gestão de Atletas ficou responsável por coletar os *feedbacks* dos *stakeholders* ao realizarem toda a administração dos atletas do time. Nesta seção, no questionário, foram indagadas as seguintes perguntas:

1. Você encontrou a seção de gestão de Atletas?
2. Você encontrou o botão de Adicionar Atleta?
3. Você teve alguma dificuldade ao adicionar um atleta? Se sim, quais?
4. Você encontrou o botão de Editar Atleta?
5. Você teve alguma dificuldade ao editar um atleta? Se sim, quais?
6. Você encontrou o botão de Desativar Atleta?
7. Você teve alguma dificuldade ao desativar um atleta? Se sim, quais?
8. Você encontrou o botão de Reativar Atleta?
9. Você teve alguma dificuldade ao reativar um atleta? Se sim, quais?
10. Você sentiu falta de alguma informação? Se sim, quais?
11. Você tem alguma sugestão de melhoria?

6.2.1.5 Teste 4: Fluxo de Caixa

A seção de Fluxo de Caixa ficou responsável por coletar os *feedbacks* dos *stakeholders* ao realizarem toda a administração das entradas e saídas de caixa do time, além de adicionar mensalidades pagas pelos atletas ativos, e realizar o pagamento do técnico em atividade e comando do time. Nesta seção, no questionário, foram indagadas as seguintes perguntas:

1. Você encontrou a seção de fluxo de caixa?
2. Você teve alguma dificuldade ao selecionar o mês de pagamento? Se sim, quais?
3. Você encontrou o botão de Adicionar Transação?
4. Você teve alguma dificuldade ao adicionar uma mensalidade? Se sim, quais?
5. Você teve alguma dificuldade ao adicionar pagamento de técnico? Se sim, quais?
6. Você teve alguma dificuldade ao adicionar uma entrada de caixa? Se sim, quais?
7. Você teve alguma dificuldade ao adicionar uma saída de caixa? Se sim, quais?
8. Você sentiu falta de alguma informação? Se sim, quais?
9. Você tem alguma sugestão de melhoria?

6.2.1.6 Teste 5: Configuração de Conta

A seção de Configuração de Conta ficou responsável por coletar os *feedbacks* dos *stakeholders* ao configurarem os dados das contas de Administrador, Time e os dados do Técnico. Nesta seção, no questionário, foram indagadas as seguintes perguntas:

1. Você encontrou a seção de Configurações?
2. Você teve alguma dificuldade em configurar os dados da conta do time? Se sim, quais?
3. Você teve alguma dificuldade em configurar os dados do seu perfil? Se sim, quais?
4. Você teve alguma dificuldade em configurar os dados do técnico? Se sim, quais?
5. Você sentiu falta de alguma informação? Se sim, quais?
6. Você tem alguma sugestão de melhoria?

6.3 Análise e Interpretação dos Dados

Nessa seção, serão abordadas, em conjunto, as etapas de Coleta de Dados (já descrita anteriormente), e de Análise e Interpretação dos Dados. A intenção é facilitar a compreensão de ambas as etapas por parte do leitor.

Para a análise e a interpretação dos dados, esta subseção será subdivida em:

1. Usabilidade do *Setter*;
2. Dificuldades e Sugestões de Melhorias, e
3. Pontos Positivos.

6.3.1 Usabilidade do *Setter*

Os testes de usabilidade da aplicação Setter, no escopo do MVP, foram feitos via questionário, como mencionado anteriormente. Para isso, ele foi dividido em seções de funcionalidades, como descrito previamente.

Todos os testes seguiram um mesmo padrão de perguntas, sendo ele composto das seguintes perguntas, que foram adaptadas para o contexto de cada teste realizado:

1. Você encontrou a seção específica do teste?
2. Você teve alguma dificuldade ao realizar a ação específica proposta? Se sim, quais?
3. Você sentiu falta de alguma informação? Se sim, quais?
4. Você tem alguma sugestão de melhoria?

O Quadro 11 condensa as respostas pelos testadores ao encontrarem seções específicas da aplicação. Essa ação era possível de se realizar por meio de cliques em botões de acesso.

É possível concluir que os usuários que testaram a aplicação obtiveram facilidade ao encontrar seções específicas, mantendo a taxa de acerto maior que 95% em sua totalidade. demonstrando que a aplicação possui uma boa usabilidade.

A taxa de 100% de acerto nos botões que ficam localizados na aba fixa lateral da aplicação, chamada de *sidebar*, que engloba os botões de Atletas, Fluxo de Caixa e Configurações, demonstra que sua implementação foi um acerto para a usabilidade da aplicação.

A proposta de utilizar as laterais da tela foi aproveitada em outros contextos da aplicação, principalmente como forma de preenchimento de informações. Como exemplo,

Quadro 11 – Porcentagem de Acertos e Erros dos Testadores ao clicarem em botões específicos

Botão	Quantidade de Testadores	% Acertos	% Erros
Criar um Time	7	100	0
Login	7	100	0
Atletas	7	100	0
Adicionar Atleta	7	100	0
Editar Atleta	7	85,7	14,3
Desativar Atleta	7	85,7	14,3
Reativar Atleta	7	85,7	14,3
Fluxo de Caixa	7	100	0
Adicionar Transação	7	100	0
Configurações	7	100	0

Fonte: Autores

tem-se o formulário para adicionar um novo atleta, ou então o formulário para adicionar uma transação ao fluxo de caixa, e entre outros. É possível observar que a proposta, mesmo que em outros contextos, foi muito bem aceita pelos *stakeholders*.

Fica evidente que os erros cometidos deram-se na seção de Gestão de Atletas e em alguns botões inseridos neste contexto da aplicação. Uma análise com maior nível de detalhamento acerca das dificuldades encontradas pelos *stakeholders* será apresentada posteriormente, ainda nas subseções deste capítulo.

6.3.2 Dificuldades e Sugestões de Melhorias

Esta subseção apresenta as principais coletas de dificuldades, encontradas pelos *stakeholders*, e reveladas via coleta de dados, além das sugestões de melhorias propostas em cada seção do teste realizado.

O Quadro 12 mostra a porcentagem de dificuldades encontradas em cada seção, e também a porcentagem de sugestões de melhorias. Pode-se observar que a taxa de dificuldade foi baixa, mantendo-se abaixo 5% em sua totalidade, ou seja, considerando todas as seções de teste. Porém, a taxa de sugestões já se mostra mais significativa, passando dos 28% em sua totalidade. Vale ressaltar ainda que foram feitas sugestões em todas as seções dos testes.

Diante do exposto, é possível afirmar sobre a facilidade no uso da aplicação pelos *stakeholders*, e o interesse que eles possuem em aprimorar a ferramenta, levantando pontos de melhorias e sugestões de novas funcionalidades que se aplicam ao contexto de gerenciamento de times de voleibol.

Um ponto de dificuldade bastante relevante que foi detectado foi a falha do sistema ao adicionar um atleta que possui os mesmos dados do administrador do time. A Figura 28

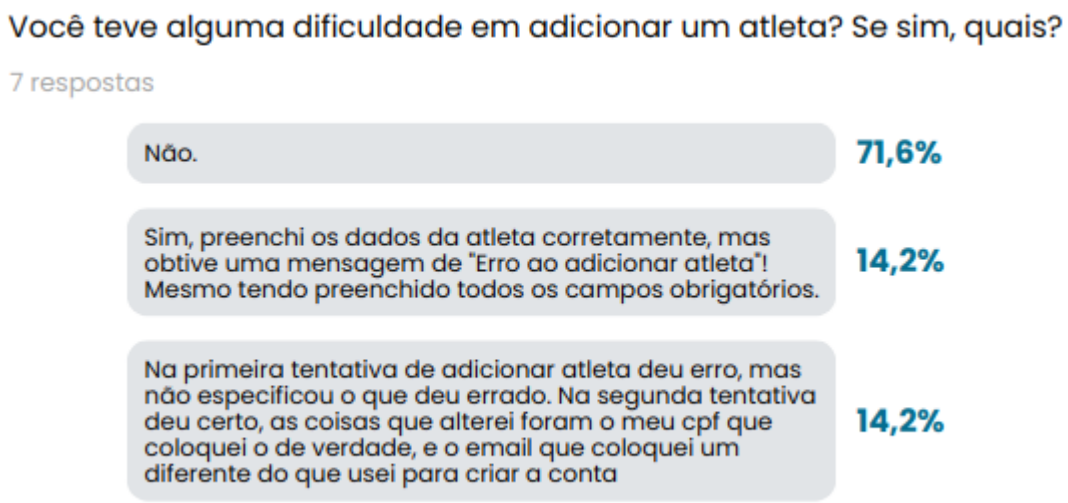
Quadro 12 – Porcentagem de Dificuldades e Sugestões dos Testadores em cada seção do Teste

Seção	Quantidade de Testadores	% Dificuldades Encontradas	% Sugestões de Melhorias Propostas
Criação de Conta	7	0	28,5
Login em Conta	7	0	28.5
Gestão de Atletas	7	17,8	28,5
Fluxo de Caixa	7	5,7	14,2
Configurações	7	0	42,6

Fonte: Autores

mostra que alguns tentaram realizar tal ação, e foram impedidos por um erro no sistema. Tal problema ocorreu por uma falha na implementação de regras de validação do banco de dados, impedindo que o Administrador relacionado a um time pudesse se cadastrar como Atleta.

Figura 28 – Dificuldades Encontradas pelos Administradores Respondentes ao Adicionar Atleta

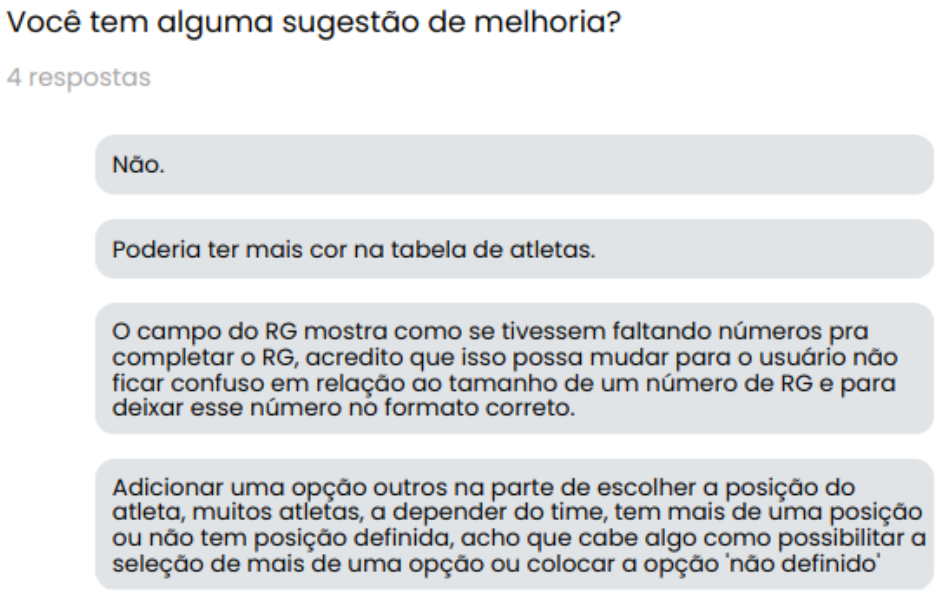


Fonte: Autores

A Figura 29 representa as respostas coletadas, nesta seção do questionário, relacionadas às sugestões de melhorias percebidas pelos *stakeholders* em relação à Gestão de Atletas como um todo. Uma observação pertinente, colocada pelos *stakeholders*, foi a máscara de texto aplicada ao campo de formulário responsável pela inserção do RG do atleta. Tal observação deu-se pela peculiaridade de variação do formato. Visto que esse número varia para cada estado brasileiro, ele deveria adaptar-se a essas possibilidades.

Outra observação realizada foi a falta de cor na tabela. Existem algumas possibilidades de melhoria para aprimorar a experiência do usuário, sendo uma delas a intercalação de cores nas linhas da tabela.

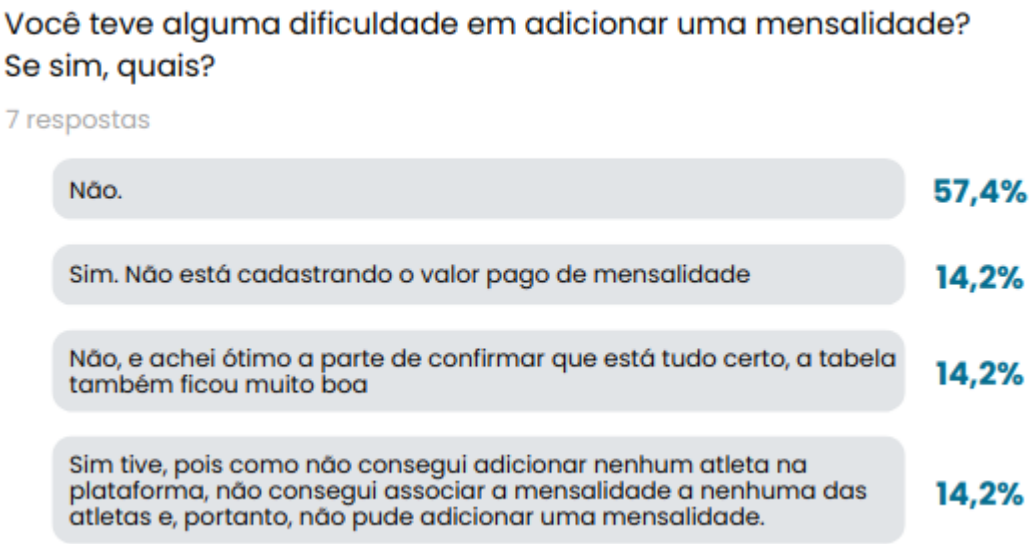
Figura 29 – Sugestões de Melhorias dadas pelos Administradores Respondentes ao realizar a Gestão de Atletas



Fonte: Autores

Já na seção de Fluxo de Caixa, uma dificuldade relatada trouxe outro ponto de atenção para a aplicação. A Figura 30 mostra que ocorreram algumas dificuldades ao realizar a adição de uma mensalidade.

Figura 30 – Dificuldades Encontradas pelos Administradores Respondentes ao Adicionar Mensalidade



Fonte: Autores

Um dos casos de erro deu-se pela situação em que um dos *stakeholders* adicionou uma data de pagamento de mensalidade maior que a data atual da realização do teste. Uma das limitações de regras de negócios definida para o Fluxo de Caixa é impedir

a adição de transações posteriores à data atual, para evitar inconsistências e falhas no sistema.

Entretanto, a validação feita na camada de apresentação não restringiu ao usuário a possibilidade de adicionar uma data inválida para a regra de negócio estabelecida, gerando um erro desnecessário para o usuário.

Todas as dificuldades encontradas e sugestões fizeram-se bastante pertinentes, agregando valor ao produto final, sendo consideradas e mapeadas para a versão de melhorias do MVP do *Setter*, que será mais bem detalhada em seções posteriores neste mesmo capítulo.

Vale relembrar que, na presente seção, constam explanadas as principais análises realizadas pela coleta dos dados por meio do questionário disponibilizado para os *stakeholders*. Entretanto, esse levantamento foi além, considerando a investigação de outros aspectos, sendo possível a visualização completa dessa coleta pelo Apêndice D.

6.3.3 Pontos Positivos

Apesar de algumas dificuldades relatadas pelos *stakeholders* nos testes realizados na aplicação *Setter*, o saldo foi bastante positivo. Como mencionado anteriormente, os *stakeholders* apresentaram pouquíssimas dificuldades ao usar a aplicação, e trouxeram muitas sugestões que agregam ao produto final.

Ao final do questionário, foi pedido para que os *stakeholders* pudessem escrever um pequeno depoimento quanto às impressões que eles possuíram ao utilizar a aplicação. A grande maioria deles revelou estar muito contente com o que foi apresentado, e muito otimista para utilizar e aplicar a ferramenta em seus respectivos times. Seguem amostras dos depoimentos feitos pela parcela dos *stakeholders* respondentes que se dispuseram para tal. Mantém-se o anonimato quanto aos nomes dos depoentes.

"A aplicação é ótima e não vejo a hora de disponibilizarem para uso. Faz anos que usamos planilhas e pastas para lidar com essas coisas do time, mas sempre foi um caos usá-las, sinto falta de algo mais intuitivo e esquematizado. Ao testar a aplicação ficou nítido o quanto ela nos ajudaria. Uma das minhas partes favoritas foi a página de controle de gastos, e sugiro adicionarem uma parte para controle de faltas também. Excelente trabalho!"

"Sobre a aplicação que eu testei, achei super interessante e queria parabenizar os integrantes do trabalho por esse lindo projeto. Sobre as funcionalidades dentro do site, eu achei super tranquilo é acessível, e não tive nenhuma dificuldade em cadastrar e manusear. Achei bem interessante algo criado diretamente para o vôlei pois é muito difícil de se achar hoje em dia, acredito que esse projeto será de grande proveito para o esporte!"

"O uso do aplicativo Setter demonstrou pontos positivos e práticos, facilitando o gerenciamento da equipe. O controle do financeiro supriu todas as necessidades, de forma clara e auto explicativa, além de mostrar todos os débitos e saldos negativos, algo que me surpreendeu positivamente. Quanto a configuração da conta e a gestão de atletas, não apresentam dificuldades, basta apenas seguir o passo a passo, como em todas as outras áreas. O Setter me auxiliou de forma satisfatória na organização do meu time, além de otimizar meu tempo que, sem ele faria um trabalho mais lento. Com certeza será um aplicativo que me seguirá e me ajudará."

"Achei a aplicação bastante objetiva e intuitiva também. Na minha opinião, quanto mais centralizado for o controle dos assuntos relacionados ao gerenciamento do time melhor e a aplicação faz isso muito bem ao colocar os principais assuntos de forma bastante fácil e simples de mexer. Ter algo como essa aplicação no dia a dia da organização do time de vôlei da UnB seria realmente muito útil e pouparia muitos esforços para achar dados e poder controlar gastos. Além de tudo isso, ainda acho que é uma aplicação escalável para outros tipos de esportes e até quem sabe outros tipos de gerenciamento sem ser o esportivo. Parabéns pelo ótimo trabalho!"

6.4 Elaboração do Plano de Ação

A partir dos *feedbacks* coletados com o questionário e com o teste realizado no MVP, e avaliando as análises descritas nas seções anteriores, foi possível levantar alguns pontos para melhorar a ferramenta *Setter*, resultando em um Plano de Ação. Dentre os pontos de melhorias, compreendidos no Plano de Ação, destacam-se:

1. Retirada da máscara de texto do RG para atender diferentes tipos de RGs pelo Brasil;
2. Correção na relação de Administrador e Atleta, possibilitando a criação de um administrador que seja atleta;
3. Correção na validação dos campos de data de pagamento para impedir que os usuários insiram datas inválidas no contexto, e
4. Melhoria da visibilidade da tabela de atletas, adicionando uma intercalação de cores nas linhas da mesma.

6.5 Divulgação de Resultados

A partir do plano de ação, foram implementadas melhorias na interface, no intuito de agregar maior experiência aos usuários no que compreende, principalmente, a facilidade

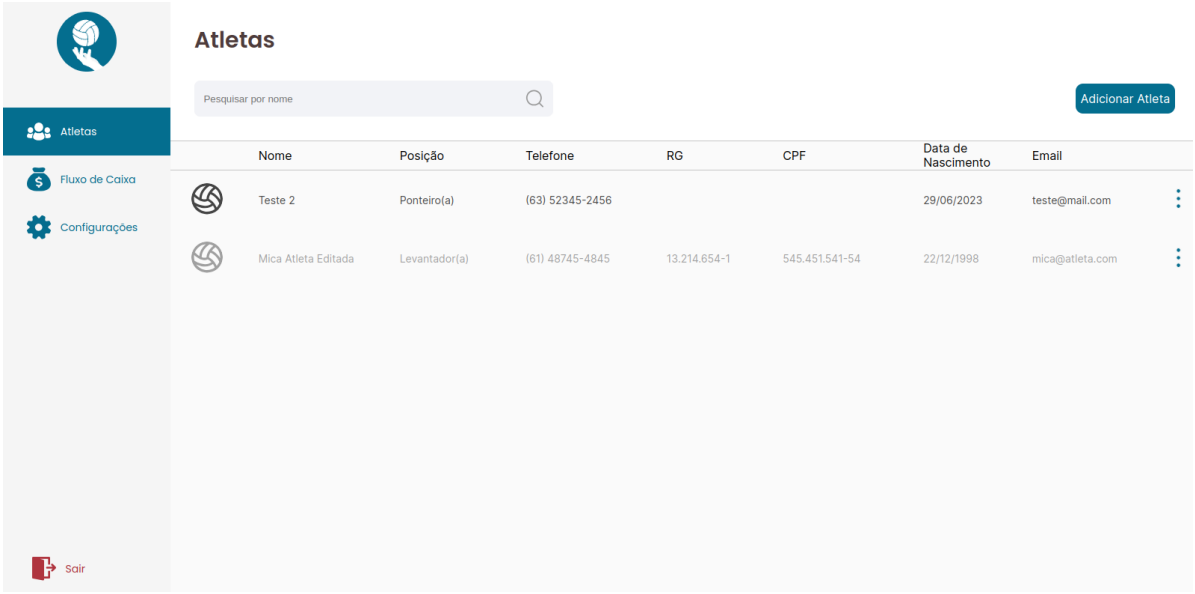
de uso. Ocorreram ainda correções de inconsistências previamente encontradas. Estas melhorias e correções permitiram obter uma nova versão do MVP desenvolvido, que já se encontra disponível para uso na aplicação *Setter*[1]. As especificações dessas melhorias estão disponíveis nas *releases* do *Frontend*[2] e *Backend*[3] lançadas no *Github*[4], local onde se encontra o código fonte da aplicação.

Uma das melhorias implementadas pode ser demonstrada pelas Figuras 31 e 32. A Figura 31 revela como a tela de gestão de atletas foi apresentada para os *stakeholders* no período de testes, ou seja, na primeira *release* do MVP lançada. Nota-se que ela era uma tabela de atletas com tons mais cinzas, com a máscara de RG aplicada com os padrões estabelecidos pelo RG do Distrito Federal. Já na Figura 32, tem-se a tela proposta na versão de melhoria do MVP, demonstrando ser uma tabela mais colorida, melhorando a legibilidade das linhas da tabela, além de retirar a máscara do RG previamente estabelecido.

Vale ressaltar que a disponibilização da ferramenta deu-se com o apoio de tecnologias de infraestrutura em nuvem. Para tal, foi realizado um estudo para a correta publicação da ferramenta *Setter* por meio do sistema de infraestrutura como *software* Heroku (2023).

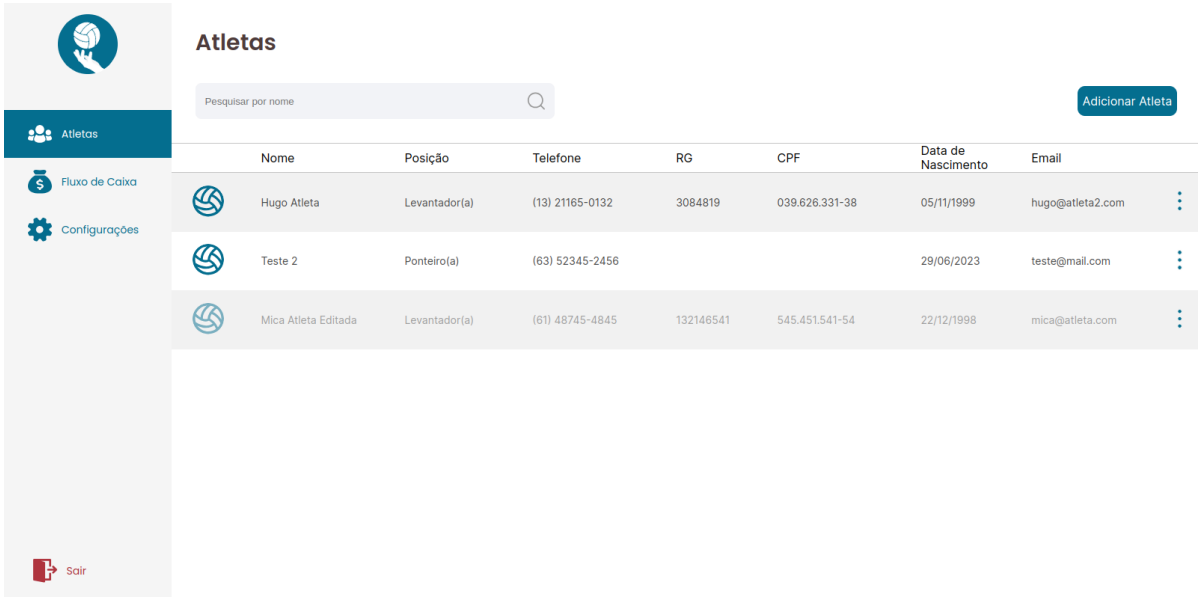
[1] Disponível em : <https://setter-front-d443799d9d71.herokuapp.com/>. Acesso em 05 jul. 2023.
[2] Disponível em <https://github.com/Setter-TCC/SetterFrontend/releases/tag/v1.1>. Acesso em 17 jul. 2023.
[3] Disponível em <https://github.com/Setter-TCC/SetterBackend/releases/tag/v1.1>. Acesso em 17 jul. 2023.
[4] Disponível em <https://github.com/Setter-TCC>. Acesso em 17 jul. 2023.

Figura 31 – Tela de Gestão de Atletas da Primeira *Release* do MVP



Fonte: Autores

Figura 32 – Tela de Gestão de Atletas da *Release* de Melhorias do MVP



Fonte: Autores

6.6 Considerações Finais do Capítulo

Neste capítulo, foram apresentados os resultados obtidos ao longo das fases de Pesquisa-Ação. A fase de Coleta de Dados tem como objetivo identificar o problema e o contexto em que este trabalho está inserido, com a finalidade de reunir informações para as etapas seguintes. Em seguida, a fase de Análise e Interpretação de Dados aborda o estudo realizado durante a Coleta de Dados, que envolveu a consulta aos *stakeholders* envolvidos no trabalho. Em seguida, foi elaborado um Plano de Ação com base nos dados

coletados, planejando-se ações futuras para atender às sugestões dos *stakeholders* que participaram do questionário e do teste de usabilidade. Por fim, os resultados obtidos a partir do Plano de Ação são divulgados, proporcionando oportunidades para as próximas versões do aplicativo desenvolvido.

7 Conclusão

Este capítulo compreende as considerações finais das atividades relacionadas ao Trabalho de Conclusão de Curso, bem como os resultados alcançados durante a condução, a elaboração e o desenvolvimento. Desta forma, em um primeiro momento, é apresentado o *Status* Geral do Trabalho, desde a atividade de Definir Tema à atividade de Apresentar TCC2 para a banca examinadora. Na sequência, constam as Questões de Pesquisa Respondidas. Há ainda um detalhamento sobre o cumprimento dos objetivos, em Objetivos Alcançados. Por fim, são elencadas as principais contribuições da Ferramenta *Setter*, procurando conferir um panorama das melhorias implementadas na versão atual, bem como ideias para trabalhos futuros, em Considerações da Ferramenta *Setter*.

7.1 *Status* Geral do Trabalho

Na primeira etapa deste trabalho, focou-se no desenvolvimento das atividades que fundamentavam e sistematizavam a proposta de projeto em relação à criação de uma ferramenta, capaz de conferir apoio e semiautomatizações às atividades relacionadas ao gerenciamento de times amadores de voleibol e suas peculiaridades, usando processos da Engenharia de *Software* como meio de atingir tal objetivo. Nesse sentido, a Quadro 13 mostra o *status* de atividades/ subprocessos da fase inicial do trabalho, desde à definição do tema até à apresentação aos membros da banca (TCC1), sendo todas tarefas concluídas.

Quadro 13 – *Status* de Atividades/ Subprocessos da Primeira Etapa do TCC

Atividade/ Subprocesso	***Status***
Definir Tema	Concluído
Formular Proposta	Concluído
Contextualizar Problema	Concluído
Levantamento Bibliográfico	Concluído
Definir Suporte Tecnológico	Concluído
Definir Metodologia	Concluído
Definir *Backlog*	Concluído
Desenvolver Protótipo da Aplicação	Concluído
Revisar TCC1	Concluído
Apresentar TCC1	Concluído

Fonte: Autores

Em um segundo momento, a condução do trabalho deu-se de forma mais prática, para desenvolver a ferramenta *Setter*; realizar testes de usabilidade com os *stakeholders*; analisar os resultados dos testes e mapear melhorias para a versão do MVP; aplicar as melhorias propostas pelo Plano de Ação proposto, e divulgar os resultados, documentando-os

Quadro 14 – *Status* de Atividades/ Subprocessos da Segunda Etapa do TCC

Atividade/ Subprocesso	***Status***
Corrigir Apontamentos da Banca	Concluído
Desenvolvimento da Aplicação	Concluído
Divulgação de Versões	Concluído
Análise de Resultados	Concluído
Revisar TCC2	Concluído
Apresentar TCC2	Concluído

Fonte: Autores

nesta monografia. Nesse sentido, o Quadro 14 mostra o status de todas as atividades/ subprocessos conduzidos nesse contexto, sendo os mesmos concluídos (exceto a apresentação à banca, que ocorrerá em breve).

7.2 Questões de Pesquisa Respondidas

Com o intuito de realizar a presente pesquisa, foram formuladas diversas questões que permearam as etapas metodológicas, e orientaram as decisões tomadas ao longo do trabalho. A partir da análise de todas as informações e evidências apresentadas neste Trabalho de Conclusão de Curso, pode-se inferir que:

1. **Tendo em vista o público alvo da aplicação, como viabilizar seu desenvolvimento, considerando um *Minimum Viable Product* (MVP)?**: diante dos resultados apresentados nos Capítulos 5 e 6, e das respostas obtidas pelo Teste de Usabilidade do MVP, pode-se concluir que a ferramenta *Setter*, em sua versão preliminar, aponta em direção de uma solução concisa para as principais dificuldades encontradas pelos administradores de times amadores de voleibol. Apesar dos problemas encontrados durante os testes, os usuários não apresentaram dificuldades de uso, e as inconsistências encontradas foram corrigidas em uma versão de melhoria do MVP.

2. **Quais boas práticas da Engenharia de *Software* podem ser aplicadas para o desenvolvimento do projeto, e como é possível integrá-las?**: a ferramenta *Setter* foi desenvolvida seguindo diretrizes estabelecidas, em todo o seu processo, de boas práticas da Engenharia de Software, que foram definidas previamente no Capítulo 5, abrangendo conceitos que foram explanados no Capitulo 2.

7.3 Objetivos Alcançados

Cabe ainda conferir uma visão geral sobre o cumprimento dos anseios do trabalho, retomando os objetivos específicos, apresentados no Capítulo 1, conforme segue:

- **Uso de técnicas de elicitação e modelagem, visando elicitar e especificar os principais requisitos do MVP**: cumprido, com uso da técnica de elicitação Questionário, junto ao público alvo, disponível no Apêndice A; e com a especificação do *Backlog* do Produto, cuja versão resumida consta apresentada na seção 5.4, e a versão ampliada consta no Apêndice B;

- **Realização de validação e priorização dos requisitos elicitados e modelados**: novamente, cumprido, sendo validado com o público alvo, via Questionário, disponível no Apêndice A; e priorizado com a técnica de priorização *MoSCoW*;

- **Modelagem de dados, uma vez que há necessidade de persistência de dados na solução**: cumprido, em uma visão preliminar, com as modelagens do Diagrama Entidade-Relacionamento e Diagrama Lógico de Dados, disponíveis, respectivamente, na Figura 23 e na Figura 24 dessa monografia;

- **Aplicação de boas práticas da Engenharia de *Software* na solução, tal como o uso de um Ciclo de Vida de *Software* apropriado, Práticas Ágeis, Padrões Arquiteturais e de Projeto**: cumprido, em termos conceituais e de embasamento, uma vez que consta um estudo detalhado sobre os principais aspectos de um Ciclo de Vida, conforme apresentado no Capítulo 2, e em termos práticos com a divulgação do MVP da aplicação *Setter*[1];

- **Documentação de Referencial Teórico, Referencial Tecnológico e aspectos metodológicos, orientando-se pelos estudos realizados no cumprimento dos Objetivos Específicos anteriores**: cumprido, com ênfase para os conteúdos disponíveis nos Capítulos 2, 3 e 4 dessa monografia;

- **Viabilização de uma solução *Open Source***: cumprido, com destaque para os conteúdos disponíveis no Capítulo 5 dessa monografia, e

- **Apresentação dos resultados obtidos usando uma abordagem, preferencialmente, qualitativa**: cumprido, considerando a primeira versão lançada, que engloba as funcionalidades propostas para o MVP da aplicação, disponível no Capítulo 6.

7.4 Lições Aprendidas

Esta seção tem como objetivo apresentar a percepção dos autores acerca do projeto desenvolvido, de modo que os tópicos que se mostraram relevantes durante a execução do presente Trabalho de Conclusão de Curso sejam evidenciados.

[1] Disponível em: <https://setter-front-d443799d9d71.herokuapp.com/>. Acesso em 10 jul. 2023.

Em um primeiro momento, é apresentado o papel das Metodologias de Desenvolvimento adotadas para o projeto, explicitando a importância dos artefatos, ritos e características das metodologias para a orientação de um desenvolvimento de *Software* em dupla. Na sequência, são apresentados os aspectos práticos que se mostraram pertinente ao projeto, pontuando considerações sobre decisões estratégicas de projeto assim como o fator de manutenibilidade de um produto de *software*.

7.4.1 Metodologias de Desenvolvimento

As metodologias de desenvolvimento de *software* são um importante pilar para a Engenharia de *Software* como um todo, portanto, a escolha de processos metodológicos que se adequem ao contexto de desenvolvimento são determinantes para o sucesso do projeto. De certo, as metodologias ágeis exercem um papel crucial no desenvolvimento de *software* efetuado por equipes, entretanto, o contexto do desenvolvimento em dupla demanda necessidades diferentes.

Como todo o ciclo de vida de um *software* foi mantido por uma dupla, as metodologias tiveram de sofrer adaptações para uma melhor adequação ao contexto. Vale o destaque para o processo de comunicação estabelecido, que foi necessário se manter constante e objetivo, devido à sobrecarga de atribuições dos autores, já que estes foram responsáveis por manter o sistema como um todo, desde as etapas mais conceituais até os aspectos mais práticos.

Destaca-se também a maturidade necessária para manter alguns ritos essenciais das metodologias, como foi o caso das reuniões de planejamento, revisão e retrospectiva das *sprints* de desenvolvimento. Por fim, vale ressaltar a relevância dos requisitos para o processo de estruturação do projeto, que, pelo número limitado de participantes, tiveram de compor uma sólida priorização, com a finalidade de incorporar o máximo de valor possível considerando o prazo e recursos para o MVP.

7.4.2 Decisões Estratégicas para Estruturação do Software

Uma vez que a *baseline* de requisitos foi elicitada, estruturada e priorizada, se faz necessário a execução da etapa do desenho de arquitetura de *software*. Durante esta etapa do ciclo de vida de um *software*, os requisitos são transformados em estruturas computacionais bem conhecidas e orientadas por padrões arquiteturais.

Durante o desenvolvimento do projeto, a arquitetura definida para a ferramenta *Setter* levou em consideração padrões sem muita complexidade e de fácil entendimento, como foi o caso do padrão arquitetural MVC. Entretanto, tal decisão não se mostrou muito acertada, visto que o estilo arquitetural proposto não se adequava ao domínio de produto em que se insere a ferramenta, isto é, gerenciamento de equipes de voleibol com

integrantes multidisciplinares.

Desta forma, uma decisão que se mostrou mais estratégica foi a adoção do estilo arquitetural da Arquitetura Limpa, visto que o mesmo apresenta um bom fator de manutenibilidade, assim como uma boa capacidade de representar o domínio da aplicação em código.

A decisão para alteração de estilos arquiteturais foi custosa e, ao mesmo tempo, desafiadora. Portanto, é possível concluir que a maturidade para tomada de decisões estratégicas, com base nos recursos disponíveis, contexto de desenvolvimento e domínio da aplicação, é um importante aspecto no desenvolvimento de projetos.

7.5 Considerações da Ferramenta *Setter*

Procurando apresentar as impressões dos autores sobre a ferramenta *Setter* em si, seguem as principais Contribuições e os Trabalhos Futuros.

7.5.1 Contribuições

O *Setter* é uma ferramenta que centraliza todos os processos de gerenciamento de times amadores de voleibol em apenas uma aplicação. Confere, ao administrador desportivo, facilidade e praticidade para lidar com as mais diversas peculiaridades do seu time.

Apesar do seu lançamento preliminar englobar apenas as funcionalidades previstas para o MVP da aplicação, a aceitação dos *stakeholders* foi ampla e contundente, a qual foi possível desdobrar elogios à usabilidade e à aparência da aplicação. Em adição, tem-se que as expectativas dos *stakeholders* foram devidamente atendidas para as funcionalidades que foram desenvolvidas. Além disso, novas versões já foram mapeadas e validadas para trabalhos futuros.

Mesmo com algumas dificuldades no uso da versão MVP, todas elas foram mapeadas no Plano de Ação mencionado na seção 6.4, e corrigidas em uma versão posterior.

7.5.2 Trabalhos Futuros

Devido à grande aceitação dos usuários do *Setter*, e aos anseios dos *stakholders* em efetivamente usar a aplicação, os próximos passos do produto seriam a implementação das versões mapeadas, sendo elas:

1. **V2 Gerenciamento**: Esta versão é um complemento da versão do MVP, que lida com todo o tratamento dos dados dos atletas. Já nesta versão, há funcionalidades

relacionadas ao controle interno do time, como Controle de Presença, Controle de Uniformes, e a possibilidade de ter mais administradores para gerir a aplicação;

2. **V3 Comunicação**: Esta versão lida com a comunicação interna do time, entre administradores e atletas, abrangendo o contexto de Lembretes e Envio de Avisos, e

3. **V4 Jogos**: Esta versão abrange o contexto do time como um time competitivo, englobando o contexto dos jogos disputados, com o Calendário do Time e o Registro dos Jogos.

Além das versões mapeadas para as próximas versões da ferramenta *Setter*, tem-se ainda a execução de próximas rodadas da metodologia da Pesquisa-Ação para que, em conjunto com os *stakeholders* do projeto, seja possível a realização da coleta, análise e disponibilização dos resultados de próximas melhorias e versões do *software*.

Referências

ALEMBIC. 2023. <https://alembic.sqlalchemy.org/en/latest/>. Acesso em: 25 jun. 2023. Citado na página 56.

ALEXANDRE, T. M. de; JUNIOR, J. R. de A. Gestão do desenvolvimento de software com o uso de quadro virtual kanban. *Brazilian Journal of Development*, v. 6, n. 12, p. 103726–103749, 2020. Citado na página 47.

ALFLEN, N. C.; PRADO, E. P. V. Técnicas de elicitação de requisitos no desenvolvimento de software: uma revisão sistemática da literatura. *AtoZ: novas práticas em informação e conhecimento*, v. 10, n. 1, p. 39–49, 2021. Citado na página 29.

AMARAL, C. M. d. S.; BASTOS, F. d. C. O gestor esportivo no brasil: Revisão de publicações no país. *Revista Intercontinental de Gestão Desportiva-Rigd*, v. 5, n. 1, p. 68–78, 2015. Citado na página 20.

AMAZON. *What Is RESTful API*. 2022. <https://aws.amazon.com/pt/what-is/restful-api/>. Acesso em: 03 jan. 2023. Citado na página 40.

ARARAQUARA, U. *TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO (TCLE)*. 2023. <https://www.uniara.com.br/comite-de-etica/termosobrigatorios/termo-de-consentimento-livre-e-esclarecido-tcle/>. Acesso em: 17 jul. 2023. Citado na página 93.

BECK, K. *Programação Extrema (XP) Explicada*. Bookman, 2004. ISBN 9788536303871. Disponível em: <https://books.google.com.br/books?id=xWWPkGLIuxMC>. Citado 2 vezes nas páginas 47 e 52.

BERSON, A. *Client/server architecture*. [S.l.]: McGraw-Hill, Inc., 1996. Citado na página 38.

BOEG, J. Kanban em 10 passos. *Tradução de Leonardo Campos, Marcelo Costa, Lúcio Camilo, Rafael Buzon, Paulo Rebelo, Eric Fer, Ivo La Puma, Leonardo Galvão, Thiago Vespa, Manoel Pimentel e Daniel Wildt. C4Media*, p. 27, 2010. Citado na página 47.

BOEHM, B. W. Verifying and validating software requirements and design specifications. *IEEE software*, IEEE Computer Society, v. 1, n. 1, p. 75, 1984. Citado na página 32.

BOEHM, B. W. A spiral model of software development and enhancement. *Computer*, v. 21, n. 5, p. 61–72, 1988. Citado na página 25.

BOJIKIAN, J. C. M.; BOJIKIAN, L. P. *Ensinando Voleibol*. 4th. ed. São Paulo: Phorte Editora, 2008. Citado na página 20.

BOURQUE, P.; FAIRLEY, R. E.; SOCIETY, I. C. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. 3rd. ed. Washington, DC, USA: IEEE Computer Society Press, 2014. ISBN 0769551661. Citado na página 31.

BPMN. 2023. <https://www.totvs.com/blog/gestao-industrial/bpmn/>. Acesso em: 14 jan. 2023. Citado na página 65.

BRAATZ, D.; OLIVEIRA, K. d. S. de; ROCHA, T. R. da. O modelo de ciclo de vida iterativo/incremental para desenvolvimento de software. In: *7ª MOEPEX*. [S.l.: s.n.], 2018. Citado na página 28.

CALDWELL, B. et al. Web content accessibility guidelines (wcag) 2.0. *WWW Consortium (W3C)*, v. 290, p. 1–34, 2008. Citado na página 85.

CAPINUSSÚ, J. M. *Administração desportiva moderna*. [S.l.]: Ibrasa, 2002. v. 27. Citado na página 20.

CÁRDENAS, A. R.; FEUERSCHÜTTE, S. G. A formação, relacionada à gestão, oferecida em cursos de graduação em educação física: um olhar qualitativo sobre currículos, disciplinas e ementas. *Pensar a Prática*, v. 17, n. 4, 2014. Citado 2 vezes nas páginas 20 e 21.

CARVALHO, B. V. d.; MELLO, C. H. P. Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequena empresa de base tecnológica. *Gestão & Produção*, SciELO Brasil, v. 19, p. 557–573, 2012. Citado na página 45.

CLEGG, D.; BARKER, R. *Case method fast-track: a RAD approach*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1994. Citado na página 31.

COHN, M. *Succeeding with agile: software development using Scrum*. [S.l.]: Pearson Education, 2010. Citado 2 vezes nas páginas 49 e 50.

COUTINHO, L. d. C. S. Gestão da tecnologia e inovação no esporte: Estudo de caso do voleibol brasileiro. 2017. Citado na página 19.

DACOSTA, L. P. et al. Cenário de tendências gerais dos esportes e atividades físicas no brasil. *Atlas do esporte no Brasil [Internet]. Rio de Janeiro: CONFEF*, p. 21–3, 2006. Citado na página 19.

DEACON, J. Model-view-controller (mvc) architecture. *Online][Citado em: 10 de março de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf*, v. 28, 2009. Citado 3 vezes nas páginas 34, 35 e 36.

DOCKER. 2022. <https://docs.docker.com/>. Acesso em: 01 dez. 2022. Citado na página 59.

DRIESSEN, V. *A successful Git branching model*. 2010. <https://nvie.com/posts/a-successful-git-branching-model/>. Acesso em: 18 jul. 2023. Citado na página 90.

DUNCAN, D. Grasp patterns. 2012. Citado 2 vezes nas páginas 24 e 41.

D'ANUNCIACAO, G. T. *Artefato: Documento de Arquitetura de Software*. 2006. <https://www.cin.ufpe.br/~gta/rup-vc/core.base_rup/workproducts/rup_software_architecture_document_C367485C.html>. Acesso em: 31 jan. 2023. Citado na página 85.

EDER, J.; KAPPEL, G.; SCHREFL, M. *Coupling and cohesion in object-oriented systems*. [S.l.], 1994. Citado na página 41.

ELMONIEM, M. A. A.; NASR, E. S.; GHEITH, M. H. A requirements elicitation tool for cloud-based erp software product line. *Proceedings of the 3rd Africa and Middle East Conference on Software Engineering*, 2017. Citado na página 30.

FASTAPI. 2022. <https://fastapi.tiangolo.com/>. Acesso em: 01 dez. 2022. Citado na página 55.

FERGUSON, R. W.; LAMI, G. An empirical study on the relationship between defective requirements and test failures. In: *2006 30th Annual IEEE/NASA Software Engineering Workshop*. [S.l.: s.n.], 2006. p. 7–10. Citado na página 29.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine, 2000. Citado 2 vezes nas páginas 38 e 39.

FIGMA. 2022. <https://www.figma.com/>. Acesso em: 01 dez. 2022. Citado na página 57.

GARLAN, D. Software architecture: a roadmap. In: *Proceedings of the Conference on the Future of Software Engineering*. [S.l.: s.n.], 2000. p. 91–101. Citado na página 33.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de pesquisa*. [S.l.]: Plageder, 2009. Citado 3 vezes nas páginas 64, 65 e 71.

GIT. 2022. <https://git-scm.com>. Acesso em: 22 nov. 2022. Citado na página 58.

GITHUB. 2022. <https://github.com>. Acesso em: 30 nov. 2022. Citado 2 vezes nas páginas 58 e 90.

GREGG, M. 12 getting things done: Productivity, self-management, and the order of things. *Networked affect*, MIT Press, p. 187, 2015. Citado 2 vezes nas páginas 20 e 74.

HEFLO. 2023. <https://www.heflo.com/pt-br/>. Acesso em: 23 jan. 2023. Citado na página 63.

HEROKU. 2023. <https://www.heroku.com/>. Acesso em: 9 jul. 2023. Citado 3 vezes nas páginas 60, 90 e 104.

HOSSEINI, M. et al. Configuring crowdsourcing for requirements elicitation. In: *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*. [S.l.: s.n.], 2015. p. 133–138. Citado na página 30.

JARKE, M.; POHL, K. Requirements engineering in 2001:(virtually) managing a changing reality. *Software Engineering Journal*, IET, v. 9, n. 6, p. 257–266, 1994. Citado 2 vezes nas páginas 29 e 69.

LATEX Project. 2022. <https://www.latex-project.org>. Acesso em: 22 nov. 2022. Citado na página 60.

LEITE, J. C. S. do P. *Livro Vivo*. 2022. <https://livrodeengenhariaderequisitos.blogspot.com/>. Acesso em: 14 fev. 2023. Citado na página 53.

MACOS Monterey. 2022. <https://support.apple.com/pt-br/HT212585>. Acesso em: 30 nov. 2022. Citado na página 59.

MANIFESTO Àgil. 2022. <http://agilemanifesto.org/>. Acesso em: 17 dez. 2022. Citado na página 45.

MARTIN, R. C. *Arquitetura Limpa: O guia do artesão para estrutura e design de software*. [S.l.]: Alta Books Editora, 2019. Citado 5 vezes nas páginas 24, 36, 37, 42 e 43.

MASSE, M. *REST API design rulebook: designing consistent RESTful web service interfaces*. [S.l.]: "O'Reilly Media, Inc.", 2011. Citado 2 vezes nas páginas 39 e 40.

MATHIJSSEN, M.; OVEREEM, M.; JANSEN, S. Identification of practices and capabilities in api management: a systematic literature review. *arXiv preprint arXiv:2006.10481*, 2020. Citado na página 55.

MEYER, B. *Object-oriented software construction*. [S.l.]: Prentice hall Englewood Cliffs, 1997. v. 2. Citado na página 42.

MEZZAROBA, C.; PIRES, G. D. L. Breve panorama histórico do voleibol: do seu surgimento à espetacularização esportiva. *Atividade Física, Lazer & Qualidade de Vida: Revista de Educação Física*, Curso de Educação Física da Universidade do Estado do Amazonas, 2011. Citado na página 19.

MICROSOFT Teams. 2023. <https://www.microsoft.com/pt-br/microsoft-teams/group-chat-software>. Acesso em: 23 jan. 2023. Citado na página 61.

MILETTO, E. M.; BERTAGNOLLI, S. de C. *Desenvolvimento de Software II: Introdução ao Desenvolvimento Web com HTML, CSS, JavaScript e PHP-Eixo: Informação e Comunicação-Série Tekne*. [S.l.]: Bookman Editora, 2014. Citado na página 44.

MIRANDA, E. Moscow rules: A quantitative exposé. In: SPRINGER. *International Conference on Agile Software Development*. [S.l.], 2022. p. 19–34. Citado 2 vezes nas páginas 31 e 78.

NASCIMENTO, L. L. Ascensão midiática do vôlei no brasil. *Encontros de Iniciação Científica UNIT*, v. 7, n. 1, 2017. Citado na página 19.

NETO, A.; OLIVEIRA, P. R. de. *Uma proposta de preparação de equipes jovens de Voleibol feminino*. Tese (Doutorado) — Dissertação de Mestrado. UNICAMP, 2003. Citado na página 19.

NOBACK, M. The dependency inversion principle. In: *Principles of Package Design*. [S.l.]: Springer, 2018. p. 67–104. Citado na página 43.

NOBACK, M. The liskov substitution principle. In: *Principles of Package Design*. [S.l.]: Springer, 2018. p. 31–53. Citado na página 43.

OKESOLA, O. et al. Qualitative assessment of systematic literatures in software engineering. *Journal of Theoretical and Applied Information Technology*, v. 96, p. 6018–6027, 09 2018. Citado na página 30.

OLIVEIRA, E. N. et al. Benefícios da atividade física para saúde mental. *Saúde Coletiva*, Editorial Bolina, v. 8, n. 50, p. 126–130, 2011. Citado na página 19.

OLIVEIRA, M. S. de et al. Benchmarking, análise envoltória de dados e análise de decisão multicritério: uma revisão da literatura. 2003. Citado na página 77.

ORGANIZATION, W. H. et al. Who guidelines on physical activity and sedentary behaviour: web annex: evidence profiles. World Health Organization, 2020. Citado na página 19.

OVERLEAF Documentation. 2022. <https://pt.overleaf.com/learn>. Acesso em: 29 nov. 2022. Citado na página 61.

PAIVA, A. C.; MACIEL, D.; SILVA, A. R. da. From requirements to automated acceptance tests with the rsl language. In: SPRINGER. *International Conference on Evaluation of Novel Approaches to Software Engineering*. [S.l.], 2020. p. 39–57. Citado na página 32.

PIP. 2022. <https://pypi.org/project/pip/>. Acesso em: 30 nov. 2022. Citado na página 59.

PLAYERPLUS. 2023. <https://play.google.com/store/apps/details?id=org.spielerplus.web&pli=1>. Acesso em: 27 mar. 2023. Citado 2 vezes nas páginas 77 e 78.

POPPENDIECK, M.; POPPENDIECK, T. *Lean software development: an agile toolkit*. [S.l.]: Addison-Wesley, 2003. Citado 3 vezes nas páginas 24, 47 e 48.

POSTGRESQL. 2022. <https://www.postgresql.org/>. Acesso em: 01 dez. 2022. Citado na página 56.

PYCHARM. 2022. <https://www.jetbrains.com/help/pycharm/quick-start-guide.html>. Acesso em: 29 nov. 2022. Citado na página 61.

REACT. 2022. <https://pt-br.reactjs.org/>. Acesso em: 01 dez. 2022. Citado na página 57.

REN, S. et al. Digitalization and energy: How does internet development affect china's energy consumption? *Energy Economics*, Elsevier, v. 98, p. 105220, 2021. Citado na página 74.

RUDD, J.; STERN, K.; ISENSEE, S. Low vs. high-fidelity prototyping debate. *interactions*, ACM New York, NY, USA, v. 3, n. 1, p. 76–85, 1996. Citado na página 85.

RUSSO, R. F. S. M.; SILVA, L. F. da; LARIEIRA, C. L. C. Do manifesto ágil à agilidade organizacional. *Gestão e Projetos: GeP*, Universidade Nove de Julho, v. 12, n. 1, p. 1–10, 2021. Citado na página 45.

SEDANO, T.; RALPH, P.; PÉRAIRE, C. The product backlog. In: IEEE. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. [S.l.], 2019. p. 200–211. Citado na página 31.

SETTER. 2023. <https://setter-front-d443799d9d71.herokuapp.com>. Acesso em: 04 jul. 2023. Citado na página 67.

SHVETS, A. Dive into design patterns. *Refactoring. Guru*, 2018. Citado 2 vezes nas páginas 34 e 41.

SLACK. 2023. <https://slack.com/>. Acesso em: 23 jan. 2023. Citado na página 61.

SOMMERVILLE, I. *Software Engineering*. Pearson, 2016. (Always learning). ISBN 9780133943030. Disponível em: <https://books.google.com.br/books?id=tW4VngEACAAJ>. Citado 25 vezes nas páginas 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 44, 45, 46, 47, 49, 50, 53, 68, 69 e 89.

SQLALCHEMY. 2022. <https://docs.sqlalchemy.org/en/14/>. Acesso em: 01 dez. 2022. Citado na página 55.

SUTHERLAND, J. *Jeff Sutherland's Scrum Handbook*. [S.l.: s.n.], 2010. Citado 4 vezes nas páginas 24, 45, 46 e 78.

TEAMER. 2023. <https://teamer.net/>. Acesso em: 27 mar. 2023. Citado 2 vezes nas páginas 77 e 78.

TELEGRAM. 2023. <https://web.telegram.org/>. Acesso em: 23 jan. 2023. Citado na página 61.

TELES, V. M. *Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade*. [S.l.]: Novatec Editora, 2017. Citado 2 vezes nas páginas 47 e 48.

THELMA, U. *The S.O.L.I.D Principles in Pictures*. 2020. <https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e898>. Acesso em: 14 dez. 2022. Citado na página 43.

UBUNTU Jammy Jellyfish. 2022. <https://releases.ubuntu.com/22.04/>. Acesso em: 01 dez. 2022. Citado na página 59.

VALENTE, M. T. Engenharia de software moderna. *Princípios e Práticas para Desenvolvimento de Software com Produtividade*, v. 1, p. 24, 2020. Citado 11 vezes nas páginas 33, 34, 35, 36, 41, 43, 44, 49, 51, 52 e 81.

VISUAL Studio Code. 2022. <https://code.visualstudio.com>. Acesso em: 22 nov. 2022. Citado na página 61.

WIEGERS, K.; BEATTY, J. *Software Requirements*. Microsoft Press, 2013. (Best practices). ISBN 9780735679665. Disponível em: <https://books.google.com.br/books?id=40lDmAEACAAJ>. Citado na página 29.

WIN, B. D. et al. On the importance of the separation-of-concerns principle in secure software engineering. In: *Workshop on the Application of Engineering Principles to System Security Design*. [S.l.: s.n.], 2002. p. 1–10. Citado na página 41.

YARN. 2022. <https://yarnpkg.com/>. Acesso em: 01 dez. 2022. Citado na página 58.

YOUNAS, M. et al. Non-functional requirements elicitation guideline for agile methods. *Journal of Telecommunication, Electronic and Computer Engineering*, v. 9, p. 137–142, 2017. Citado na página 30.

Apêndices

APÊNDICE A – Questionário de Levantamento de Requisitos

Conforme acordado anteriormente, no capítulo de Referencial Teórico, o Questionário é uma técnica de elicitação de requisitos de *software* com o objetivo de identificar e coletar informações sobre as necessidades dos *stakeholders* envolvidos com a aplicação.

O Questionário realizado para a proposta do presente trabalho de conclusão de curso foi distribuído entre representantes de times amadores de voleibol. Cada contato foi realizado pelo intermédio de um dos autores deste trabalho, visto o contexto de participação no cenário amador de voleibol em que o mesmo está inserido.

A estrutura das perguntas foi projetada, com o auxílio da ferramenta *Google Forms*[1], com o objetivo de identificar os principais focos de dificuldades empíricas no processo de gerir times esportivos. As imagens a seguir apresentam, de forma sintetizada, os resultados obtidos pela difusão do Questionário.

Em um primeiro momento, é garantido aos respondentes da pesquisa que os dados obtidos pela mesma serão analisados e utilizados no contexto deste trabalho de conclusão de curso. A Figura 33 expõe o consentimento dos respondentes da pesquisa em fornecer os dados de respostas das perguntas.

Na sequência, ilustrado pela Figura 34, é apresentada a segregação entre gestores e não gestores que responderam à pesquisa. Aqueles que não representam o público de interesse da pesquisa foram redirecionados para o encerramento do questionário, sem assinalar outras questões.

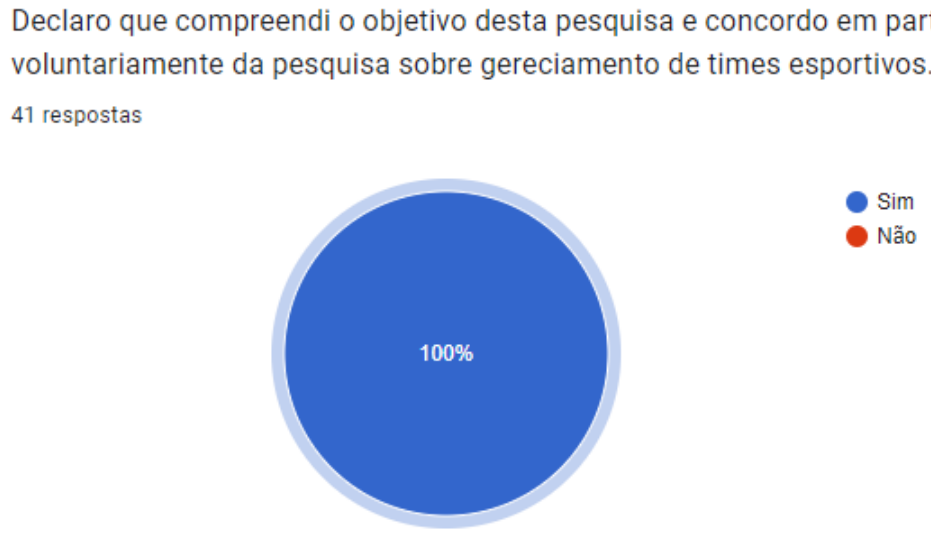
A Figura 35 demonstra os papéis de atuação dos respondentes dentro de times desportivos amadores. Percebe-se uma grande adesão de perfis de presidência e de treinamento ao questionário. Na sequência, são ilustradas, pela Figura 36, as principais atribuições dos respondentes nos times. A partir das atribuições observadas, foram elaboradas as funcionalidades do sistema que será desenvolvido ao decorrer do presente trabalho.

Para entender melhor sobre as dificuldades gerenciais em um âmbito geral, foi elaborada uma pergunta específica para tal. A Figura 37 traz alguns exemplos de respostas obtidas no Questionário.

Em seguida, com o intuito de entender o cenário de utilização de ferramentas digitais pelo público alvo, a Figura 38 elenca o aparato tecnológico disponível aos respon-

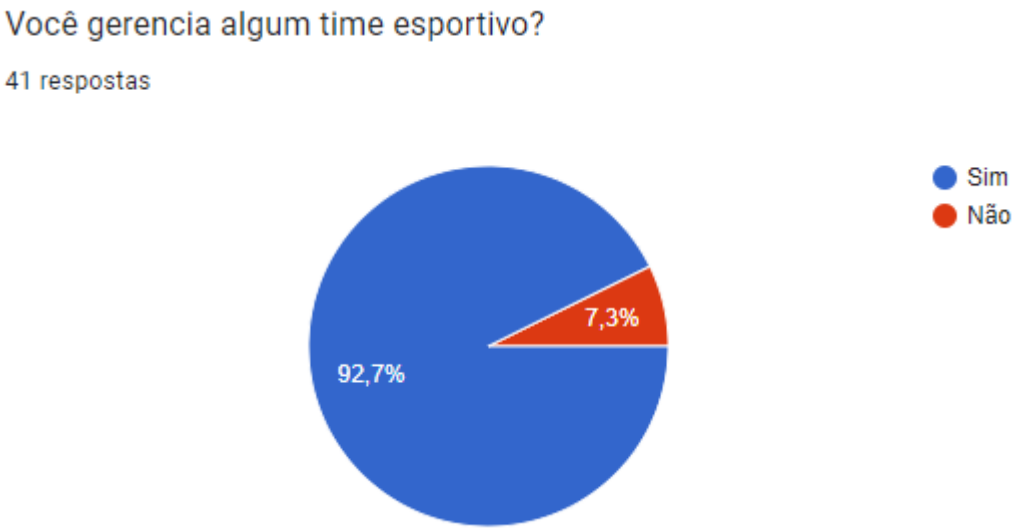
[1] Ferramenta de gerenciamento de pesquisas lançado pelo Google. Disponível em <https://docs.google.com/forms>.

Figura 33 – Consentimento dos Participantes do Questionário em Fornecer Dados para o Trabalho



Fonte: Autores

Figura 34 – Porcentagem de Gestores Respondentes à Pesquisa

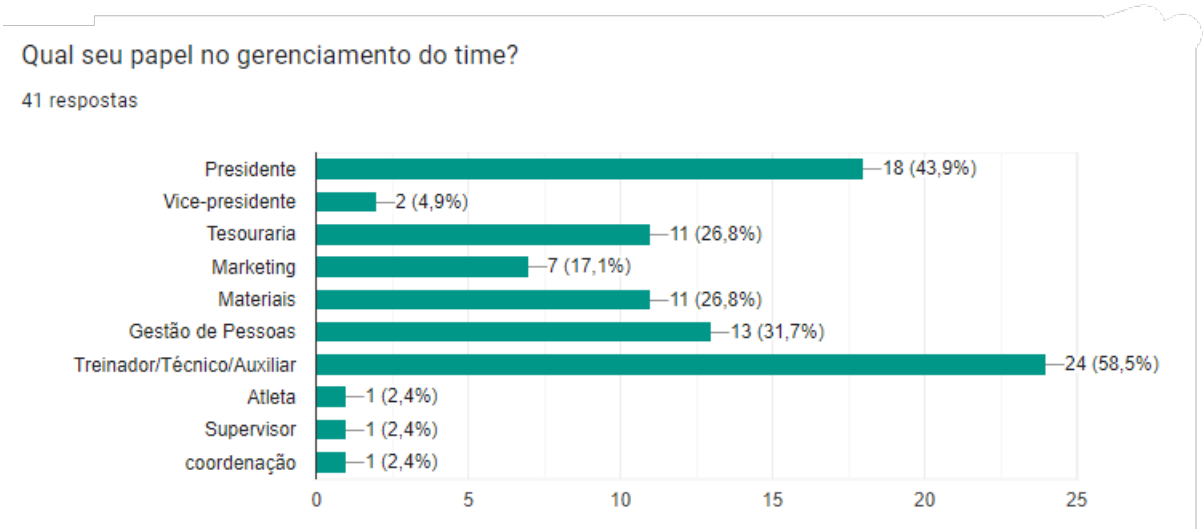


Fonte: Autores

dentes da pesquisa. Ainda neste contexto, a Figura 39 levanta o nível de conhecimento de aplicações especializadas para o gerenciamento esportivo. Vale ressaltar que, devido à característica de resposta de texto livre, as respostas com o mesmo rigor semântico estão separadas em diferentes barras, não obstante, observa-se uma notável predominância de desconhecimento.

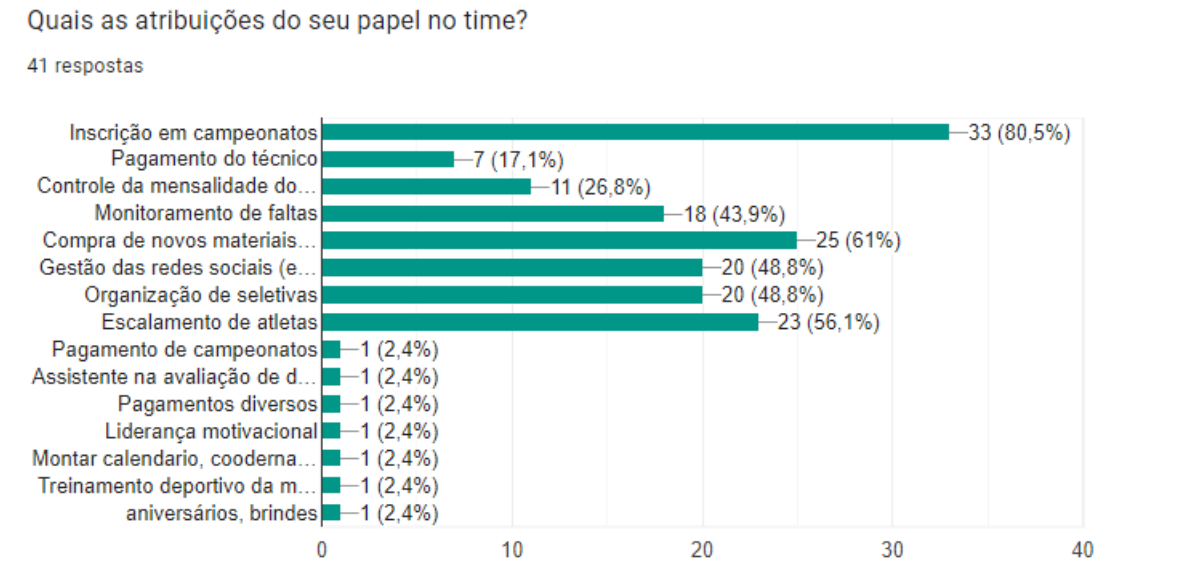
Por fim, a Figura 40 aponta alguns exemplos de funcionalidades relevantes, para o sistema especializado em gerenciamento desportivo amador, na ótica dos gestores respondentes.

Figura 35 – Perfil de Atuação dos Respondentes em Times Amadores



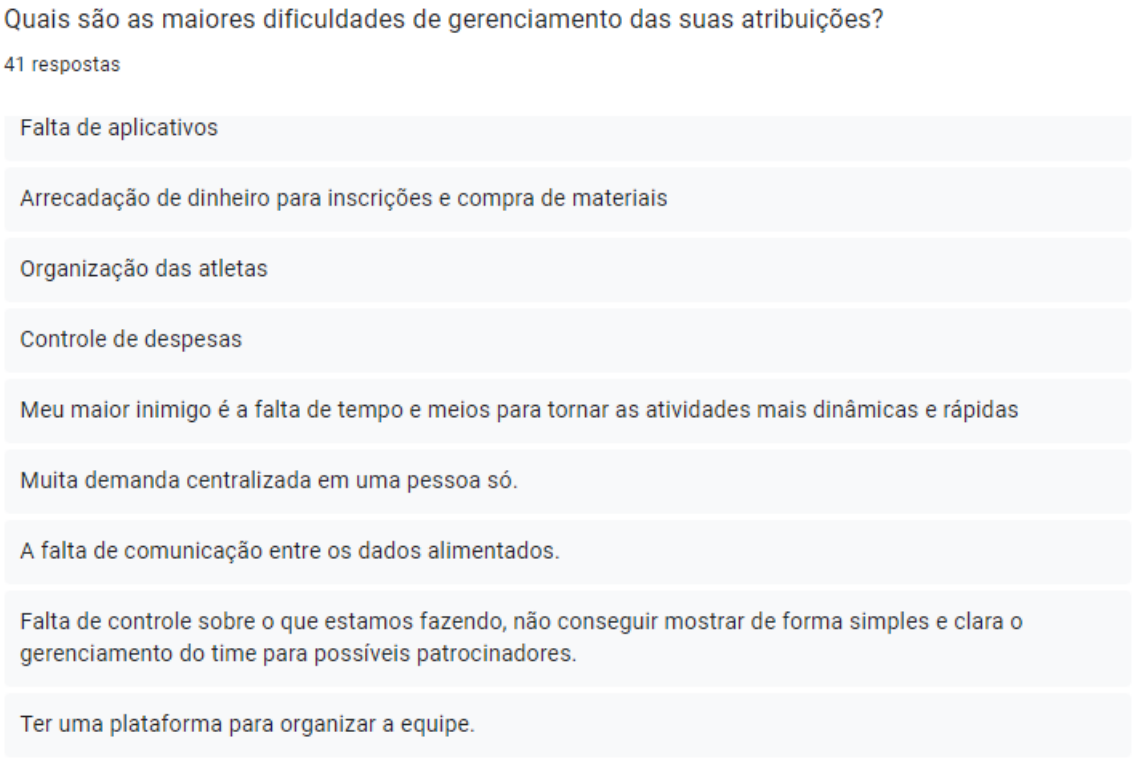
Fonte: Autores

Figura 36 – Principais Atribuições, Dentro de Times Amadores, do Público Alvo



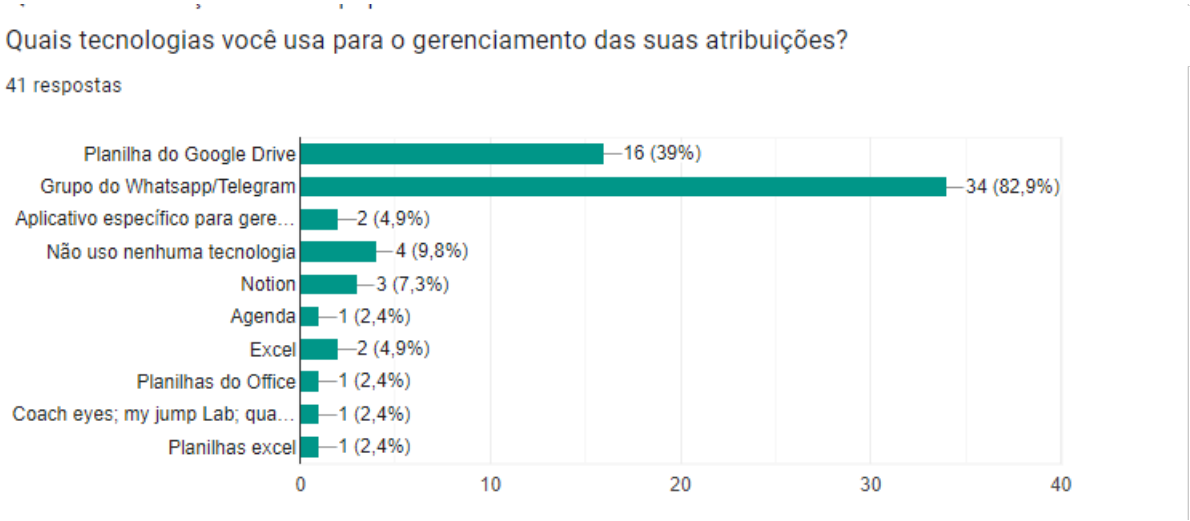
Fonte: Autores

Figura 37 – Dificuldades Gerenciais Evidenciadas pela Pesquisa



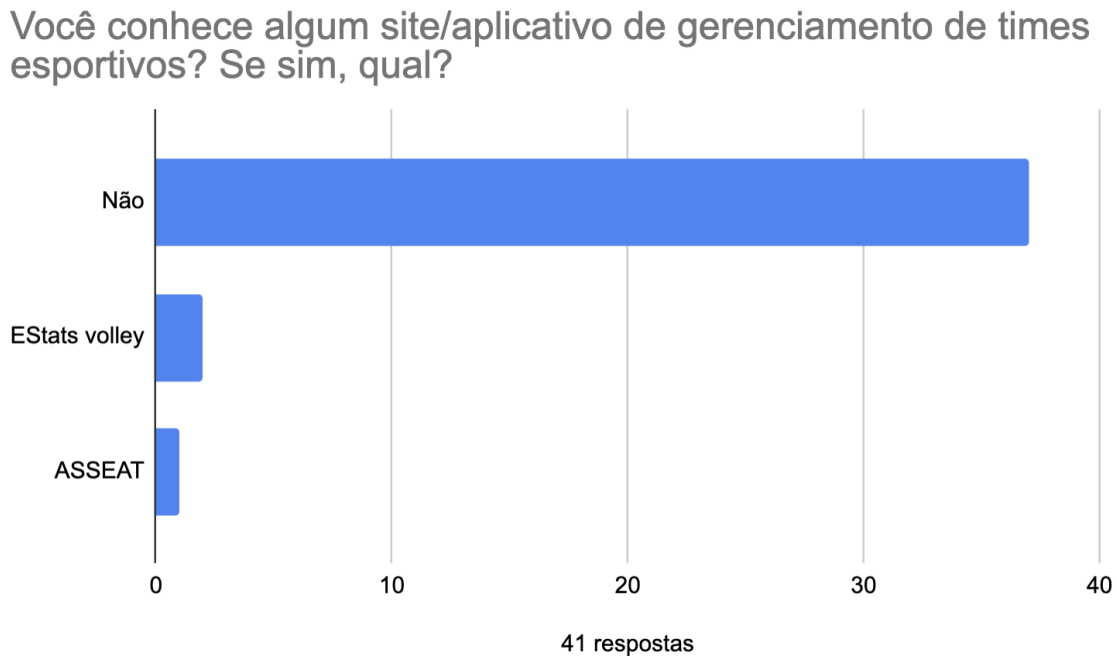
Fonte: Autores

Figura 38 – Aparato Tecnológico Utilizado pelo Público para o Gerenciamento de Times Amadores



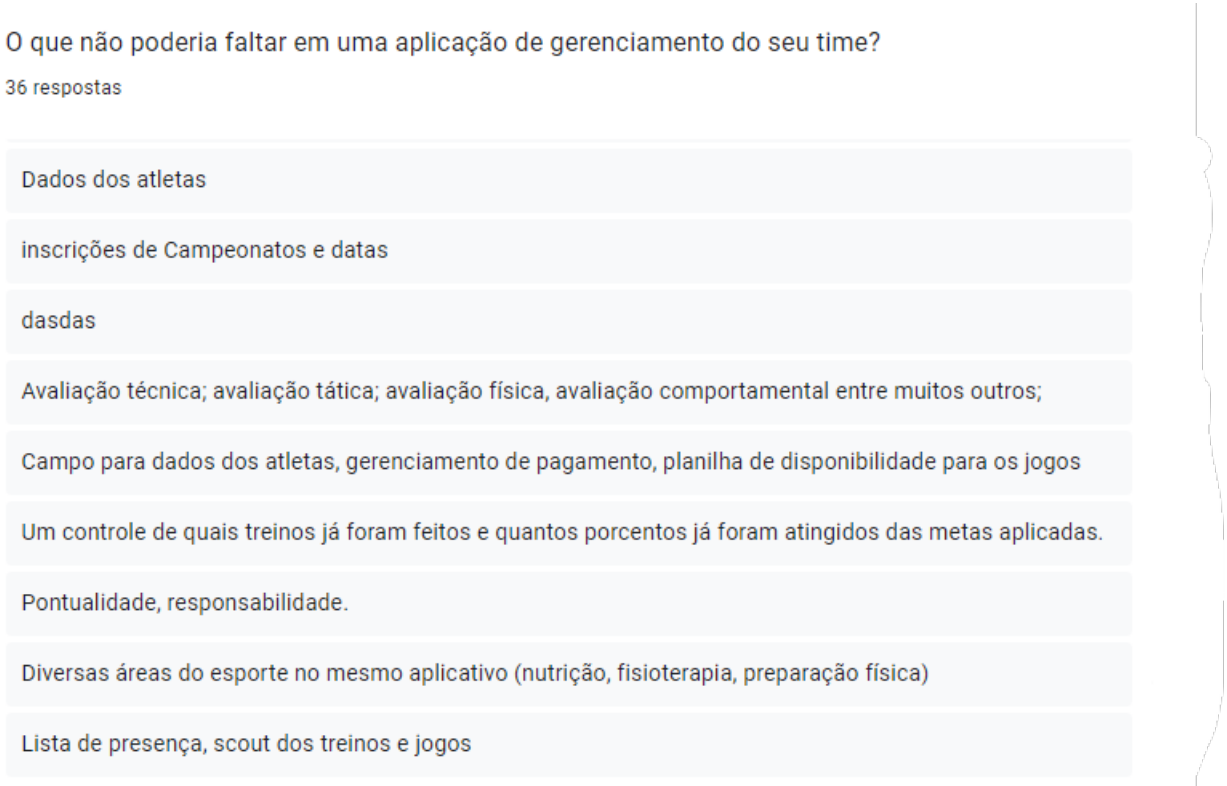
Fonte: Autores

Figura 39 – Conhecimento de Outras Ferramentas de Gerenciamento Focadas no Contexto Esportivo



Fonte: Autores

Figura 40 – Funcionalidades Apontadas como Essenciais Pelos Participantes



Fonte: Autores

APÊNDICE B – Ampla Visualização do *Backlog* do Produto *Setter*

Figura 41 – Primeira Visualização Parcial do *Backlog*

Épicos	Features	US	Priorização
Administrador	Conta	Eu como gestora de um time amador desejo criar uma conta no sistema para que eu possa criar meu time e administrá-lo.	MUST
		Eu como gestora de um time amador já cadastrado desejo entrar em minha conta com minhas credenciais para que eu possa ter acesso à organização do meu time.	MUST
	Gerenciamento de Admins	Eu como gestora de um time amador já cadastrado desejo enviar links de convite para novos usuários por meio do email para fornecer acesso à organização do time.	SHOULD
		Eu como gestora de um time amador já cadastrado desejo excluir outro administrador que tenha saído do time para que eu possa manter os dados seguros entre os administradores ativos do time. (OBS: uma vez que um admin é excluído, a conta também será).	SHOULD
		Eu como gestora de um time amador já cadastrado desejo visualizar todos os outros admistradores do time para que eu possa saber todos os que têm acesso às informações.	COULD
Time	Gerenciamento de atletas	Eu como gestora desejo incluir novas atletas na organização do meu time para manter um registro das atletas do time.	MUST
		Eu como gestora desejo alterar os dados de uma atleta do meu time em caso de alguma alteração em dado cadastral.	MUST
		Eu como gestora desejo visualizar uma lista com todas as atletas que fazem parte do atual corpo de atletas da equipe.	MUST
		Eu como gestora desejo desativar/reativar uma atleta da organização da minha equipe em caso de jogadoras que não fazem mais parte do corpo de atletas da equipe, ou caso queiram voltar a fazer parte.	MUST
	Dados do time	Eu como gestora de um time amador desejo cadastrar todos os dados do time para que eu possa ter acesso à essas informações quando necessário.	MUST
		Eu usuário desejo adicionar os dados do técnico do time para que eu possa ter acesso à essa informação quando necessário.	MUST
	Controle de presença	Eu como usuário desejo adicionar os dias da semana que possuem treino para que eu possa gerar tabelas de presenças em cada treino.	SHOULD
		Eu como usuário desejo gerar uma tabela de presença mensal para que eu possa fazer o controle de faltas do time.	COULD
		Eu como usuário desejo categorizar os tipos de falta para que eu possa entender os motivos das faltas de cada atleta.	SHOULD
	Uniforme	Eu como usuário desejo cadastrar todos os uniformes que o time possui para que eu possa saber a quantidade de uniformes e os números disponíveis.	COULD
		Eu como usuário desejo atualizar o status do uniforme para usado por algum atleta para que eu possa saber quais uniformes não estão guardados e com quem estão.	COULD

Fonte: Autores

Figura 42 – Segunda Visualização Parcial do *Backlog*

Épicos	Features	US	Priorização
Financeiro	Mensalidade	Eu como usuário desejo adicionar o valor da mensalidade para que eu possa fazer o controle de pagamento.	MUST
		Eu como usuário desejo selecionar o atleta para o pagamento de mensalidade do mês para que eu possa fazer o controle das atletas que pagaram a mensalidade.	MUST
	Controle de Caixa	Eu como usuário desejo gerar tabelas de fluxo de caixa por mês para que eu possa adicionar as entradas e saídas daquele mês.	MUST
		Eu como usuário desejo adicionar entradas e saídas de caixa para que eu possa manter o valor total sempre atualizado.	MUST
		Eu como usuário deseja adicionar o pagamento de técnico ativo no mês para que eu possa manter o fluxo de caixa atualizado.	MUST
Comunicação	Lembretes	Eu como atleta do time gostaria de receber um lembrete acerca da data de pagamento da mensalidade para não atrasar o pagamento.	COULD
		Eu como atleta do time gostaria de receber um lembrete sobre o ocorrimento dos treinos para não perder nenhum dia de treinamento.	COULD
		Eu como atleta do time gostaria de receber um lembrete sobre os jogos que estão próximos de ocorrer para não faltar.	COULD
	Envio de Avisos	Eu como gestora desejo escrever e enviar mensagens à todas as atletas da minha equipe esportiva. (OBS: a mensagem deve ser propagadas por meio dos meios de comunicação cadastrados para atleta).	SHOULD
		Eu como gestora desejo escrever e enviar mensagens à algumas atletas específicas da minha equipe esportiva.	COULD
Jogos	Calendário do Time	Eu como gestora desejo cadastrar futuros jogos para manter um controle de quais partidas minha organização vai disputar.	COULD
		Eu como gestora desejo cadastrar os treinos da minha equipe para manter um controle de treinamentos. (OBS: é possível marcar opções de recorrência para os treinos, de modo que não seja necessário o cadastro de um novo treino para cada dia).	COULD
	Registro dos Jogos	Eu como gestora desejo incluir informações de um jogo já ocorrido para atualizar o registro dos jogos. (OBS: só é possível alterar as informações de uma partida recente, isto é, partidas que ocorreram nos últimos 7 dias contados a partir do momento da utilização do app).	COULD
		Eu como gestora desejo visualizar uma lista com o histórico de jogos disputados pela minha organização esportiva.	COULD
		Eu como gestora desejo visualizar o detalhamento de um jogo específico da minha equipe para obter mais detalhes da partida.	COULD

Fonte: Autores

APÊNDICE C – Protótipo de Alta Fidelidade

Em um primeiro momento, por meio do protótipo, é apresentada ao usuário uma página de recepção, conhecida como *Landing Page*, que é ilustrada pela Figura 43. Nesta página, são dadas as opções de **Criar Time**, que pode ser interpretada como opção de criar conta, ou então **Entrar**, que referencia o ato de inserir credenciais já cadastradas para acessar as funcionalidades da ferramenta *Setter*.

A Figura 44 representa a tela de **Entrar**. Já as Figuras 45, 46 e 47 apresentam as consecutivas etapas do processo de **Criar Time**. Por fim, a Figura 48 demonstra a página que informa ao usuário o sucesso na criação de uma conta.

Uma vez que o usuário entra na ferramenta *Setter*, ele possui uma série de funcionalidades que são implementadas em diferentes páginas. A primeira a ser discutida neste apêndice será a listagem de atletas, evidenciada pela Figura 49. Esta página permite ao usuário adicionar, ou editar informações de atletas, funcionalidades que são representadas pelas Figuras 50 e 51, respectivamente.

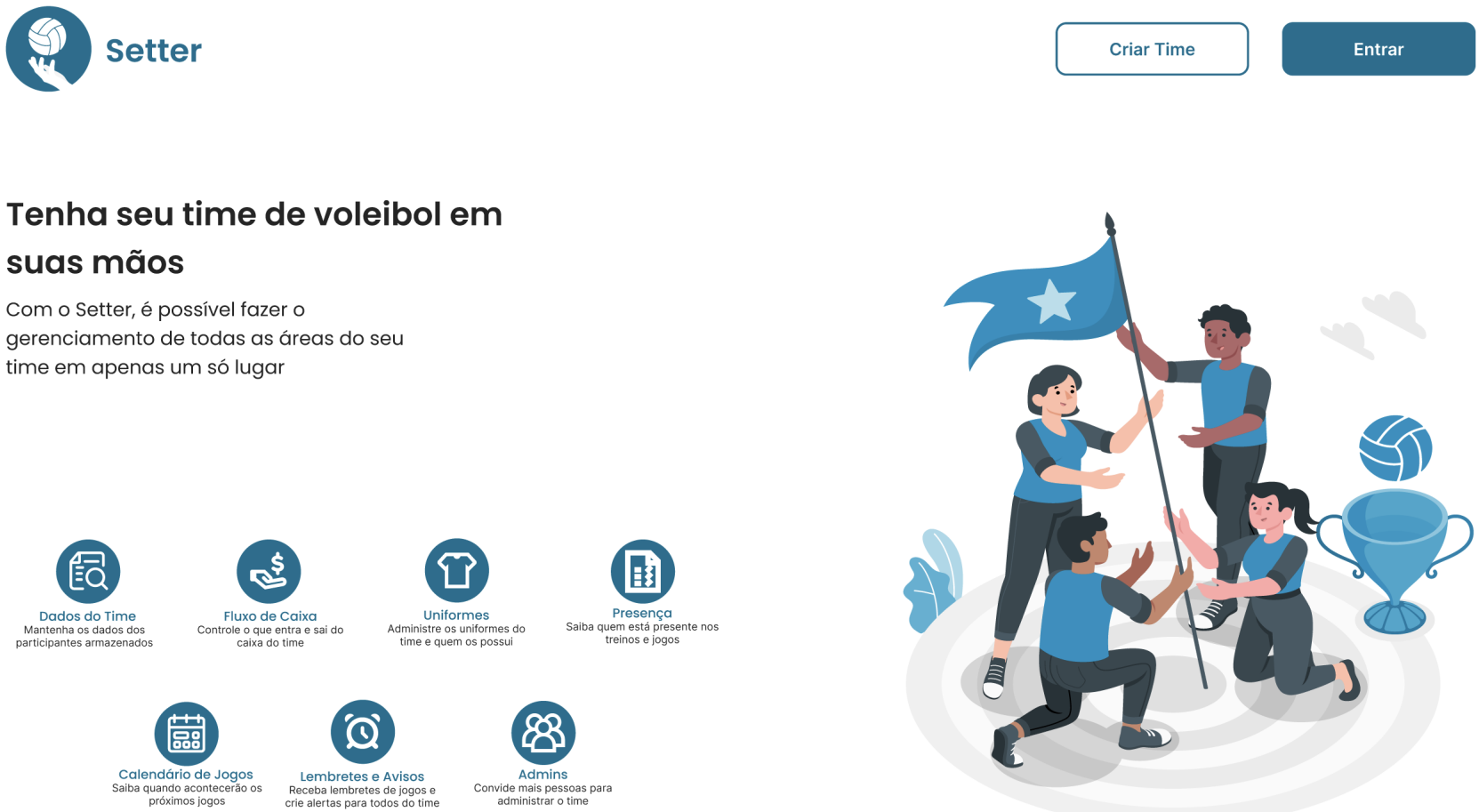
Na sequência, são apresentadas as opções de manipulação de uma atleta já cadastrada, que é possível visualizar na Figura 52. Caso um atleta seja desativado, este ainda estará visível na listagem, não obstante, com um filtro que dificulta a nitidez deste, como é possível notar na Figura 53. Um atleta desativado possui manipulações específicas, que são ilustradas na Figura 54.

Uma outra funcionalidade disponibilizada pela ferramenta *Setter*, é o **Fluxo de Caixa**. A página de Fluxo de Caixa pode ser vista na Figura 57. Ainda neste sentido, o usuário é capaz de adicionar diferentes transações para manter o controle do histórico financeiro do time. As diferentes tipos de transações e a adição de novas despesas estão disponíveis nas Figuras 58 e 59, respectivamente.

Por fim, a ferramenta *Setter* ainda permite ao usuário que o mesmo gerencie as informações submetidas ao sistema por meio das páginas de **Configuração**. As Figuras 60, 61 e 62 destacam as configurações do time, de perfil e de técnico esportivo. Já a Figura 63 destaca a adição de um técnico, caso o time não tenha nenhum técnico ativo.

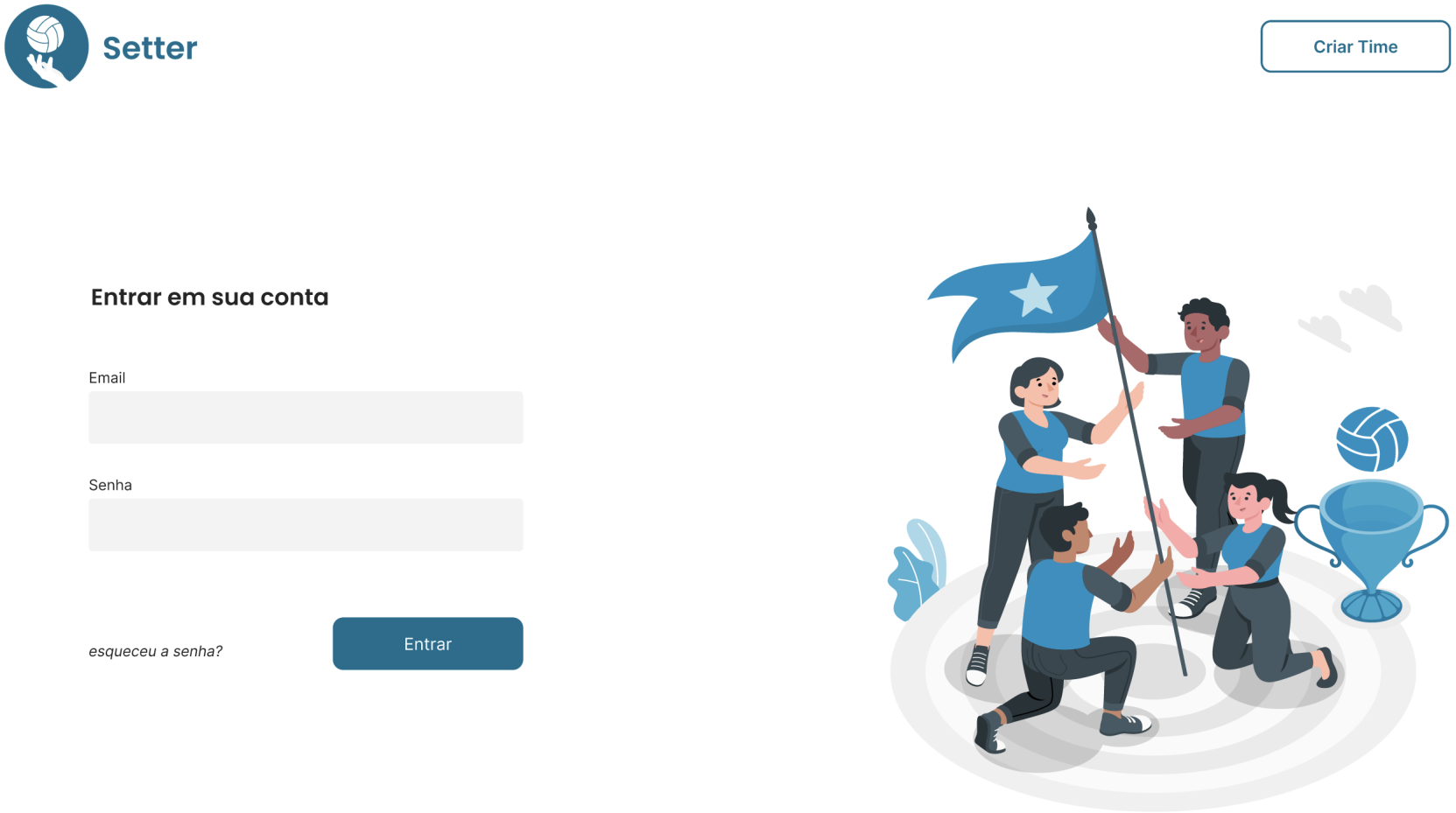
As páginas evidenciadas pelo protótipo compõem as principais funcionalidades para o MVP da ferramenta *Setter*, que é explorado mais detalhadamente na Seção de *Backlog* do Produto.

Figura 43 – *Landing Page* da Ferramenta *Setter*



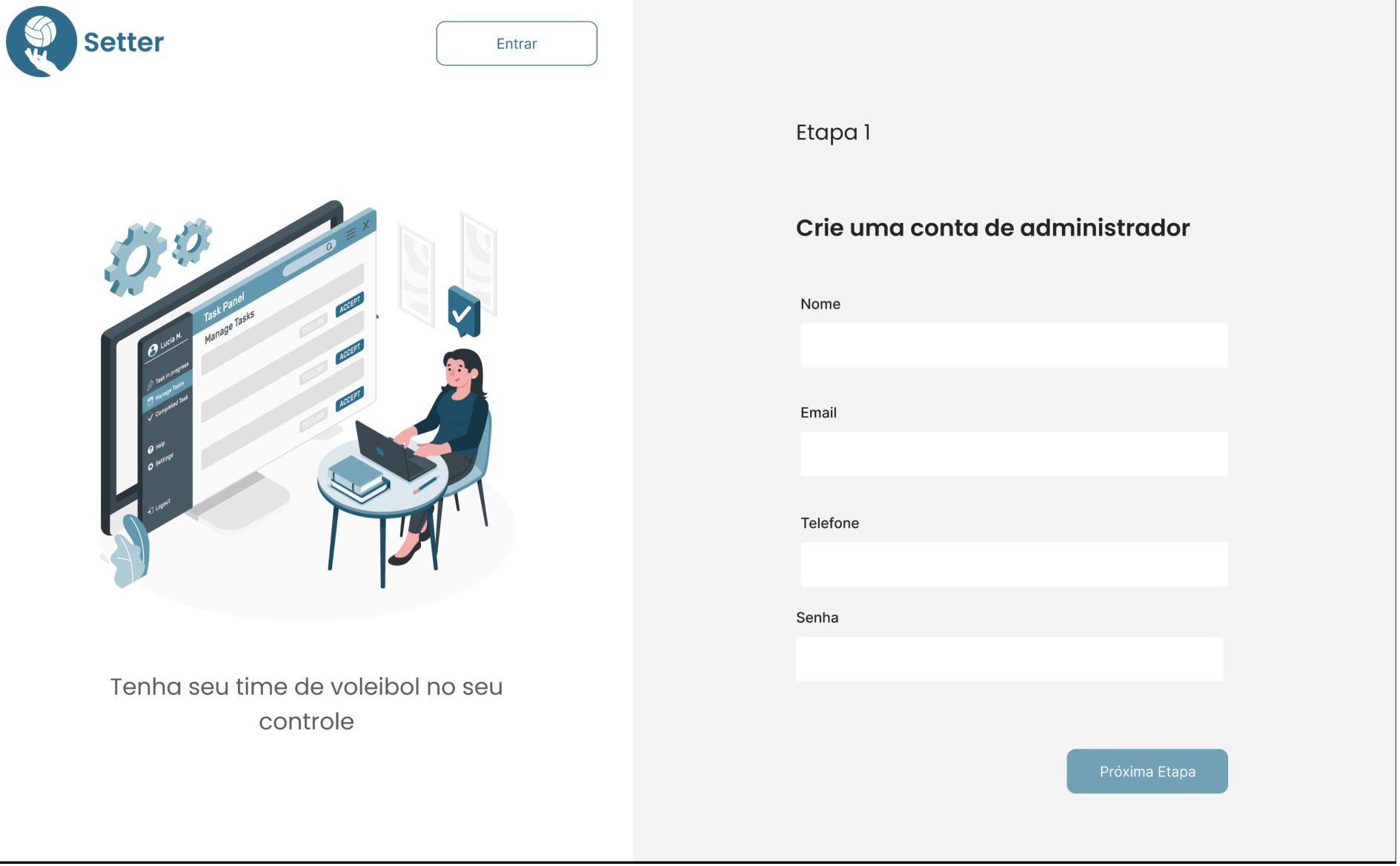
Fonte: Autores

Figura 44 – Página de *Login* da Ferramenta *Setter*



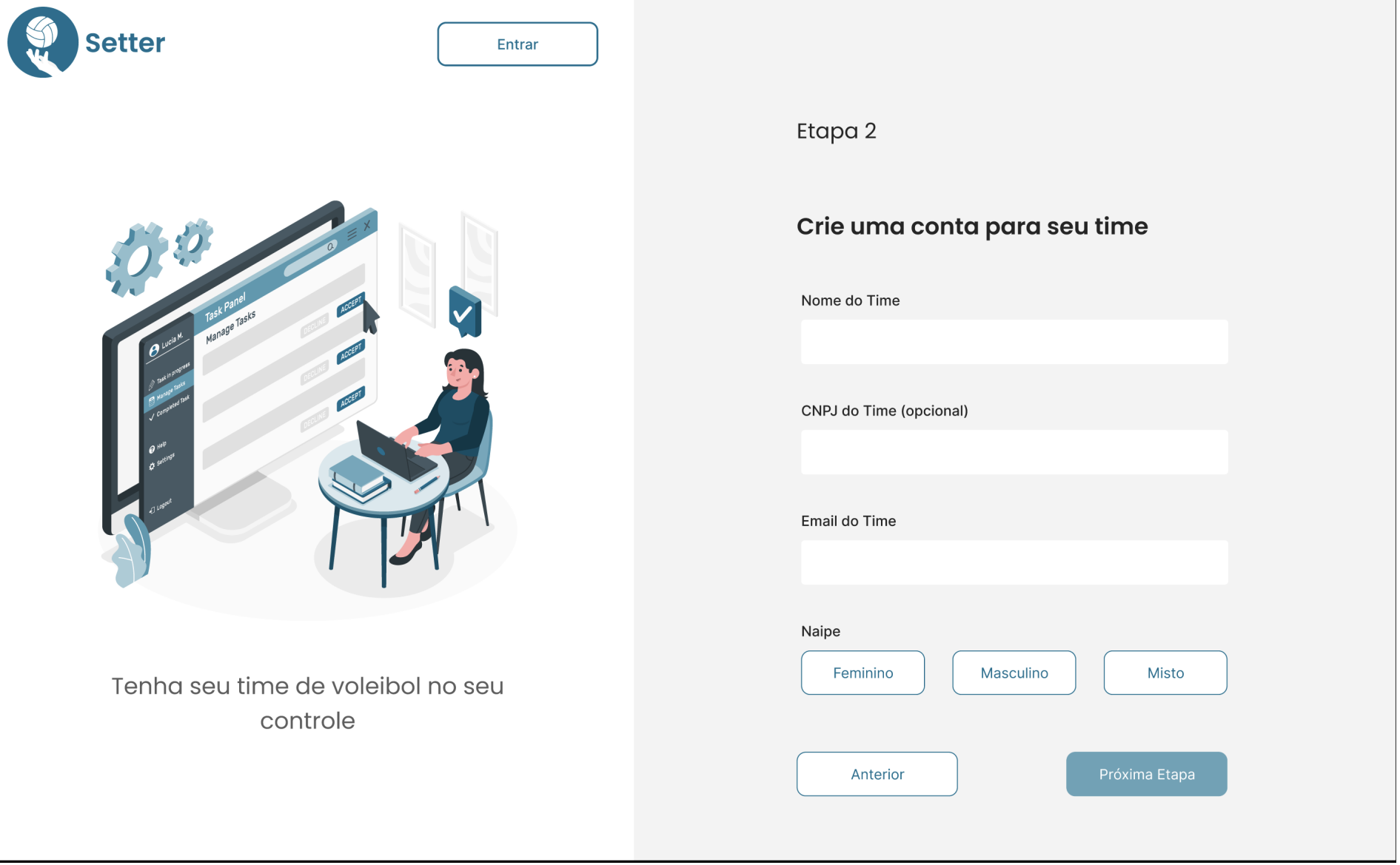
Fonte: Autores

Figura 45 – Primeira Página de Criação de Conta da Ferramenta *Setter*



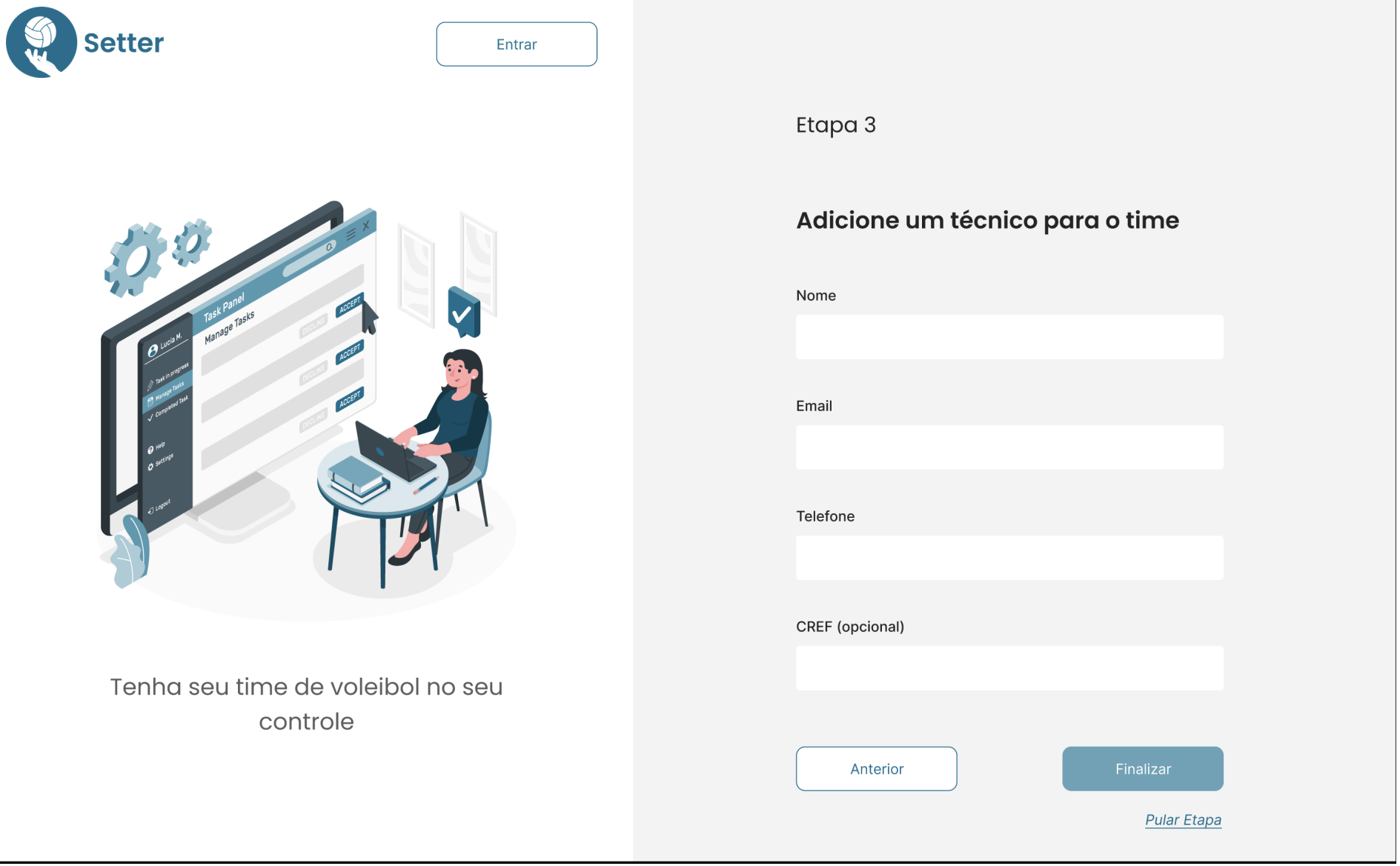
Fonte: Autores

Figura 46 – Segunda Página de Criação de Conta da Ferramenta *Setter*



Fonte: Autores

Figura 47 – Terceira Página de Criação de Conta da Ferramenta *Setter*

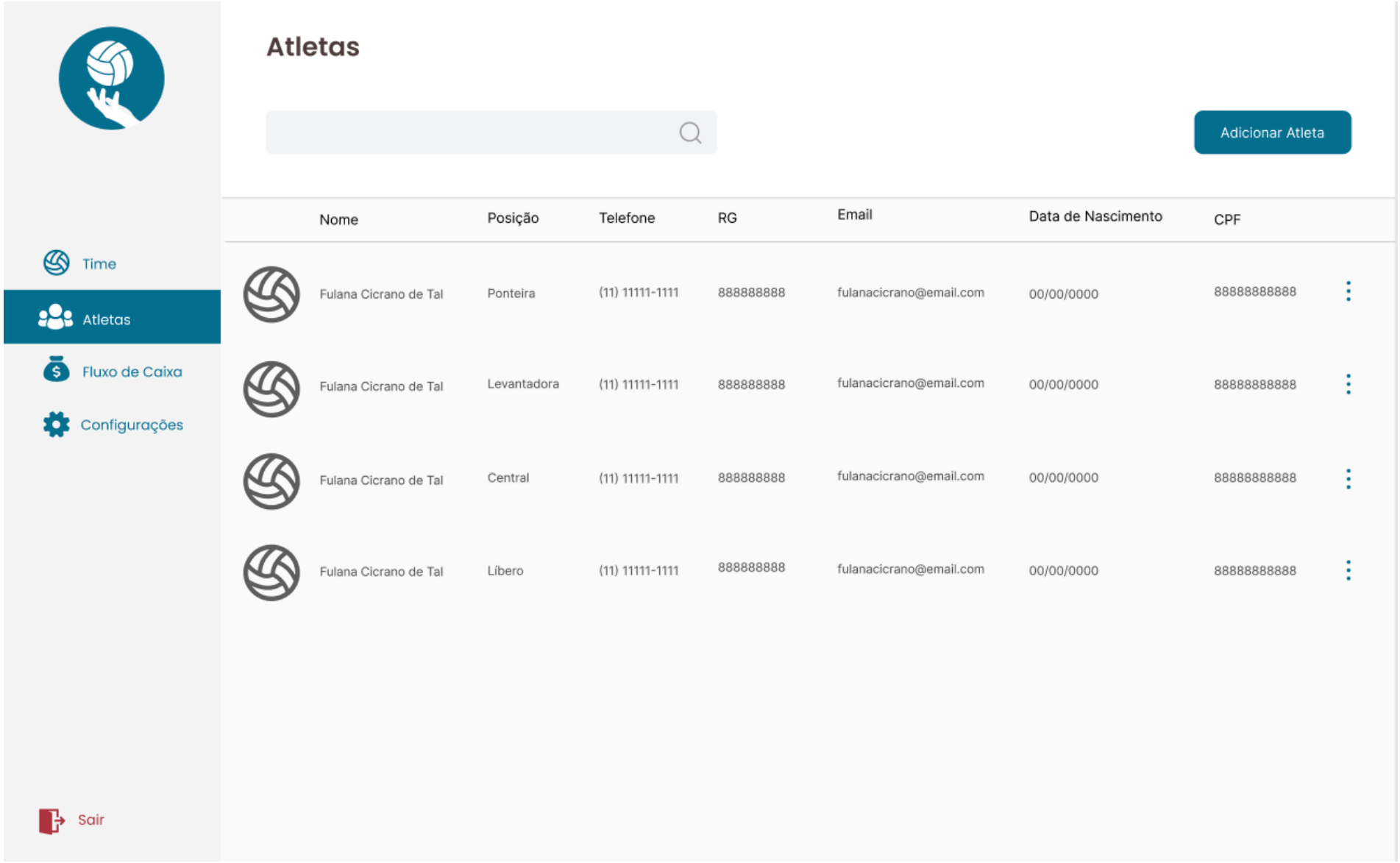


Fonte: Autores

Figura 48 – Página de Conta Criada da Ferramenta *Setter*

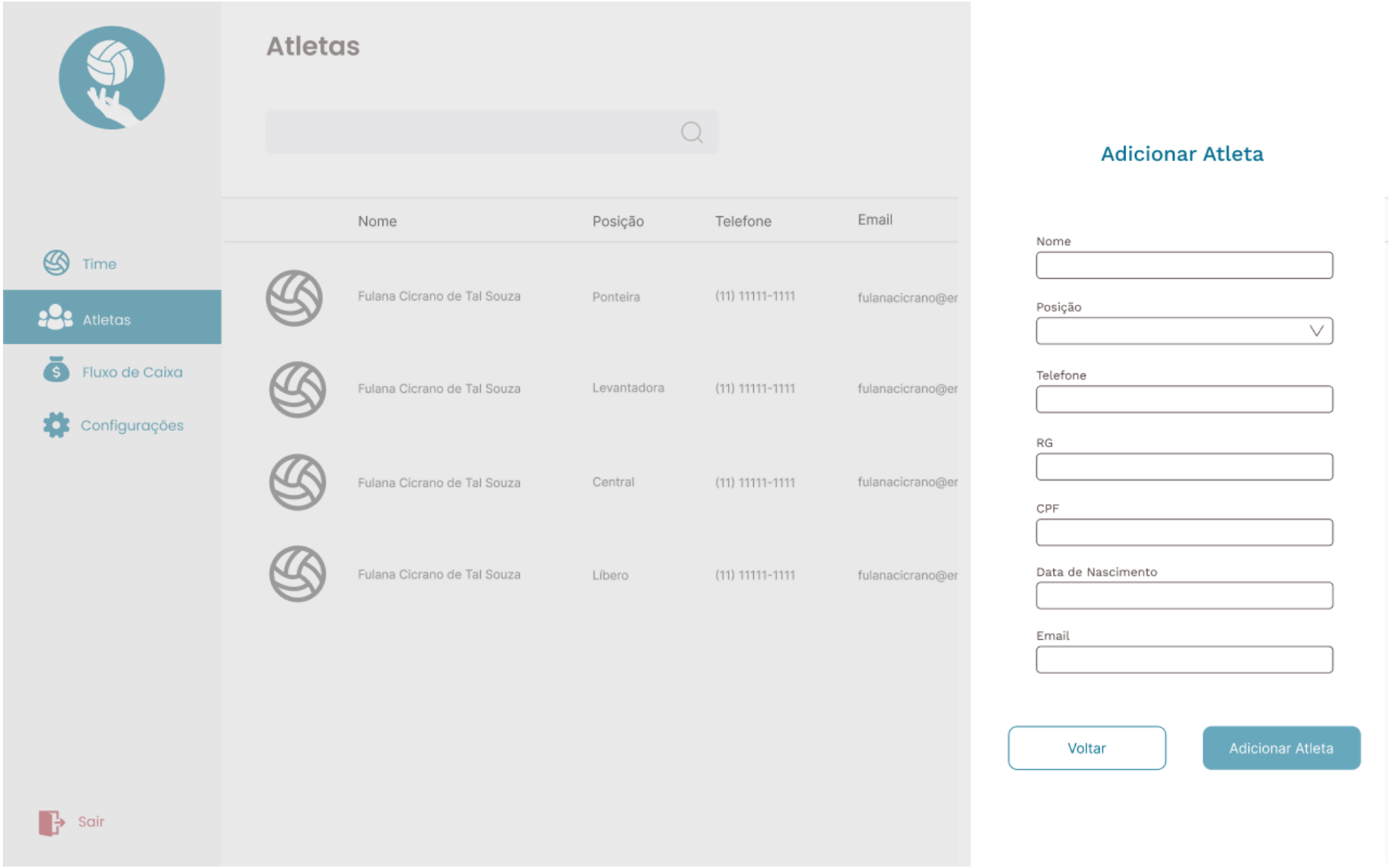
Fonte: Autores

Figura 49 – Página de Atletas da Ferramenta *Setter*



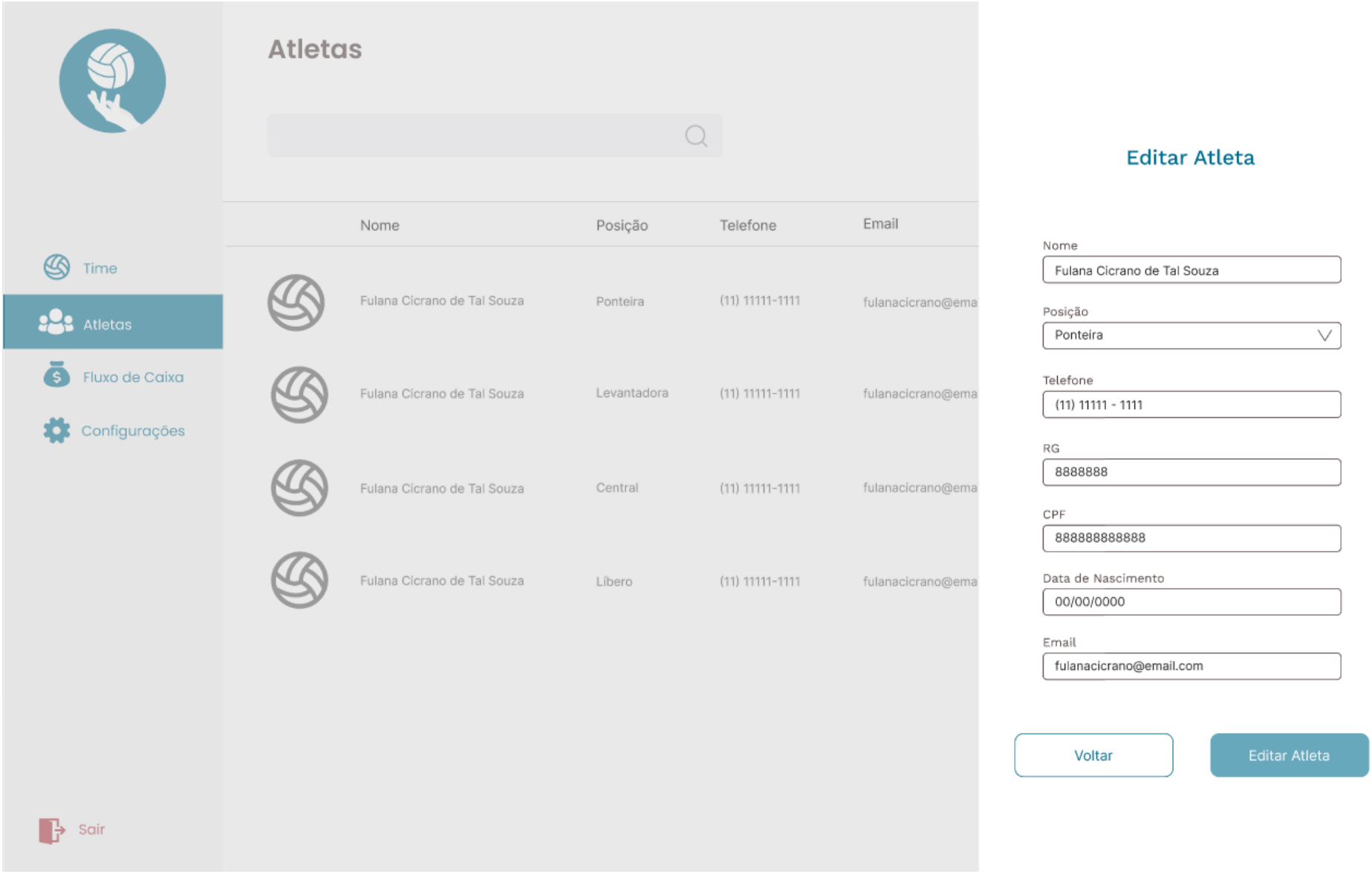
Fonte: Autores

Figura 50 – *Drawer* de Adição de Atleta da Ferramenta *Setter*



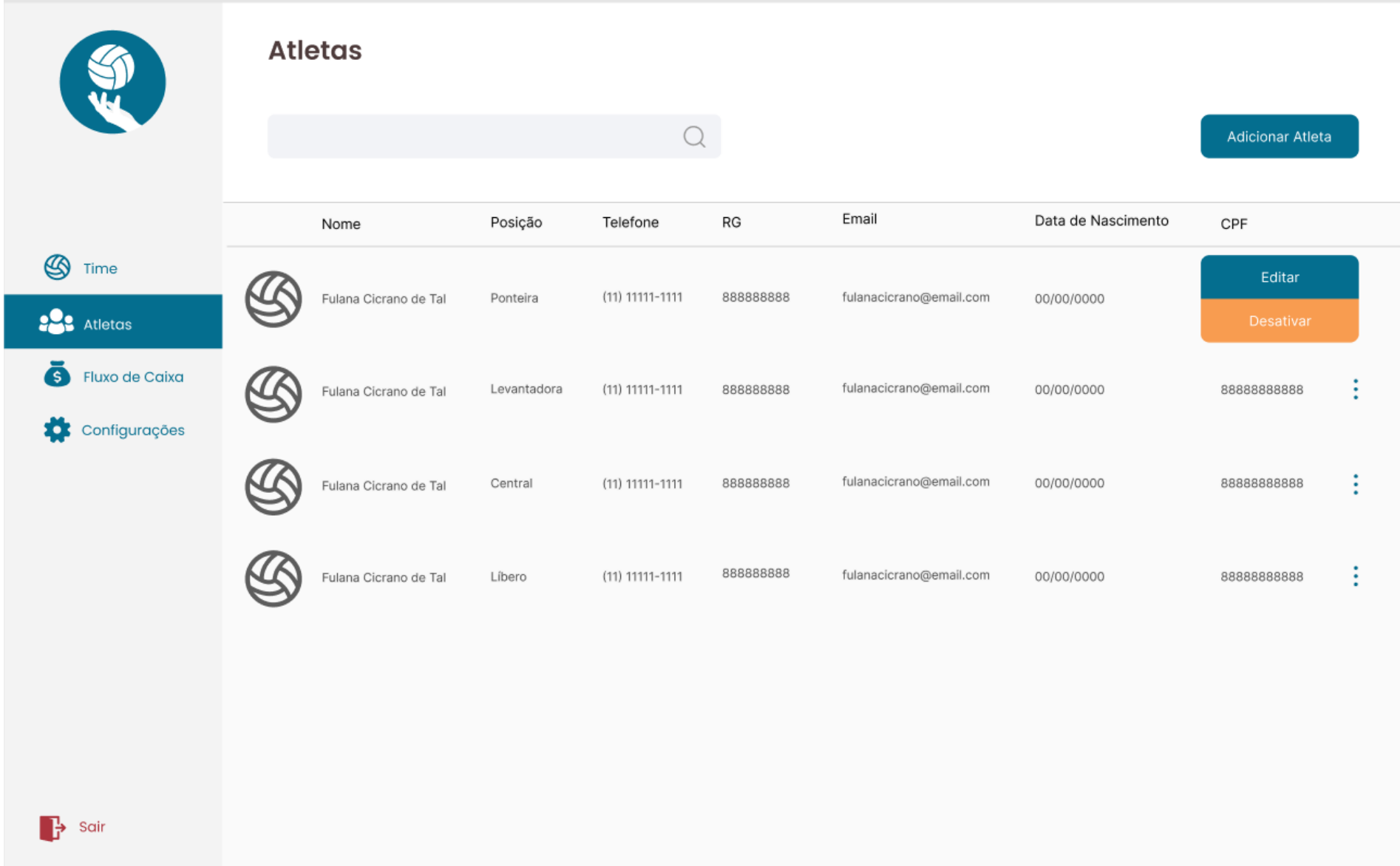
Fonte: Autores

Figura 51 – *Drawer* de Edição de Atleta da Ferramenta *Setter*



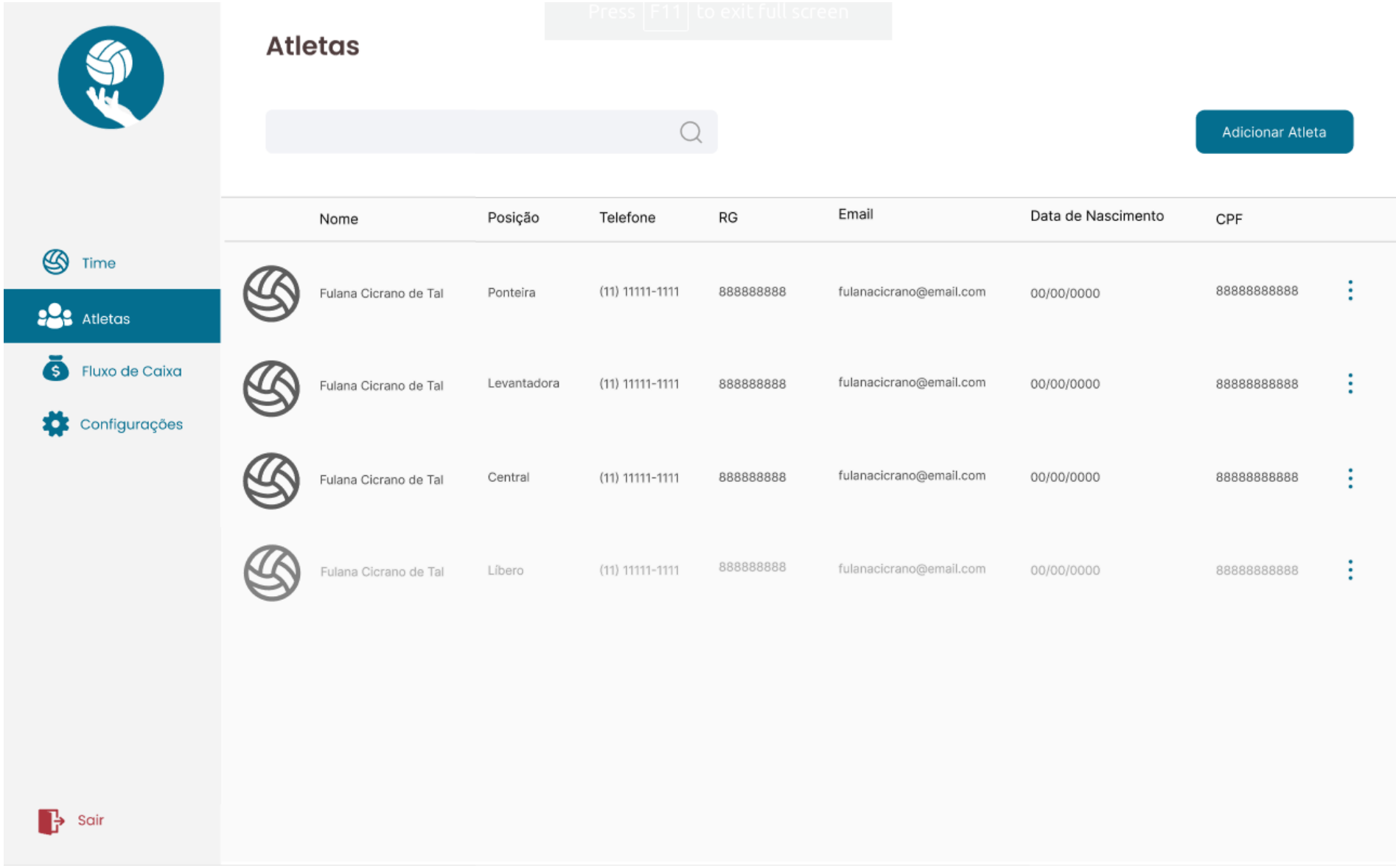
Fonte: Autores

Figura 52 – Opções de Atleta da Ferramenta *Setter*



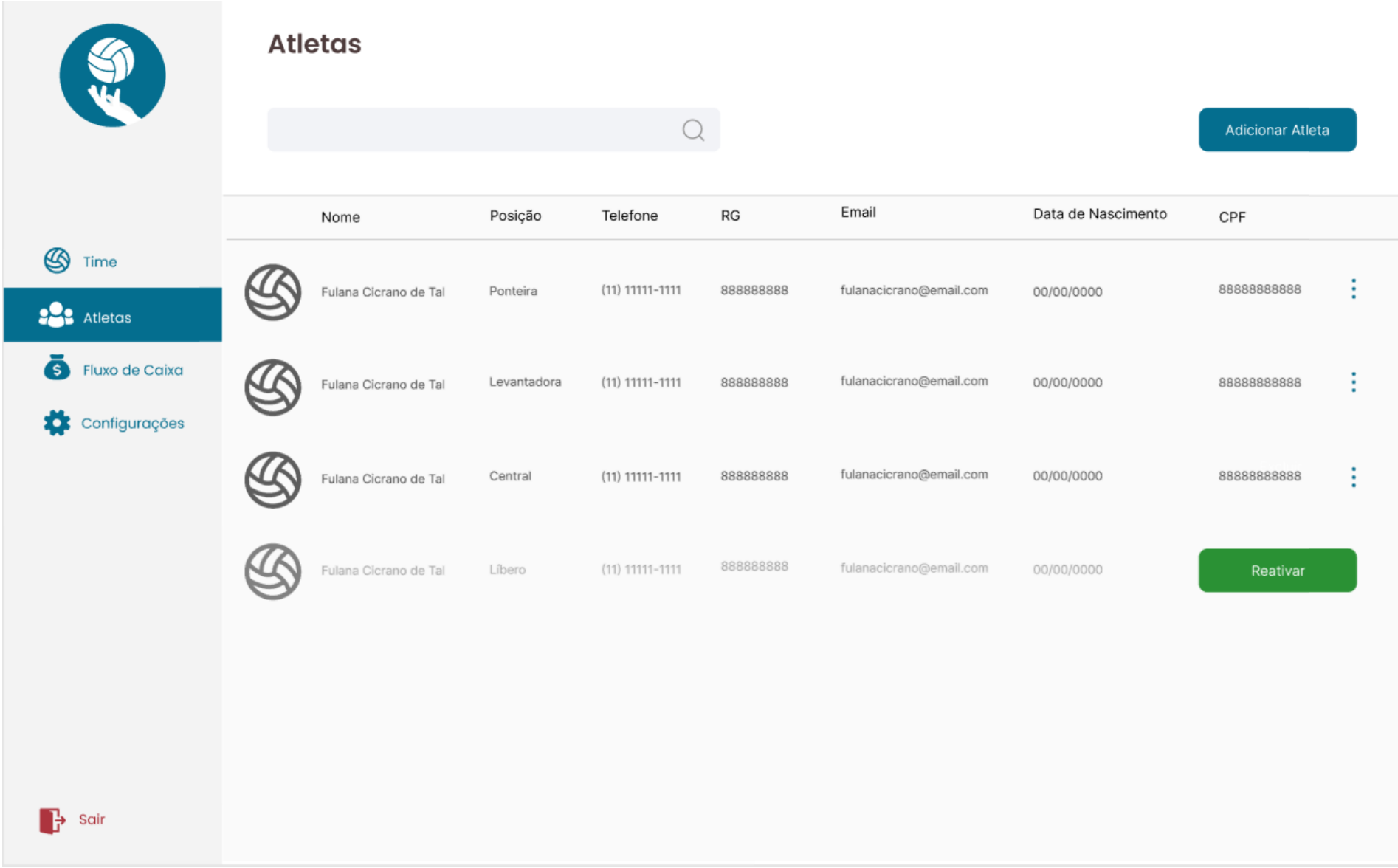
Fonte: Autores

Figura 53 – Visualização de Atleta Desativado da Ferramenta *Setter*



Fonte: Autores

Figura 54 – Opção de Reativar Atleta Desativado da Ferramenta *Setter*

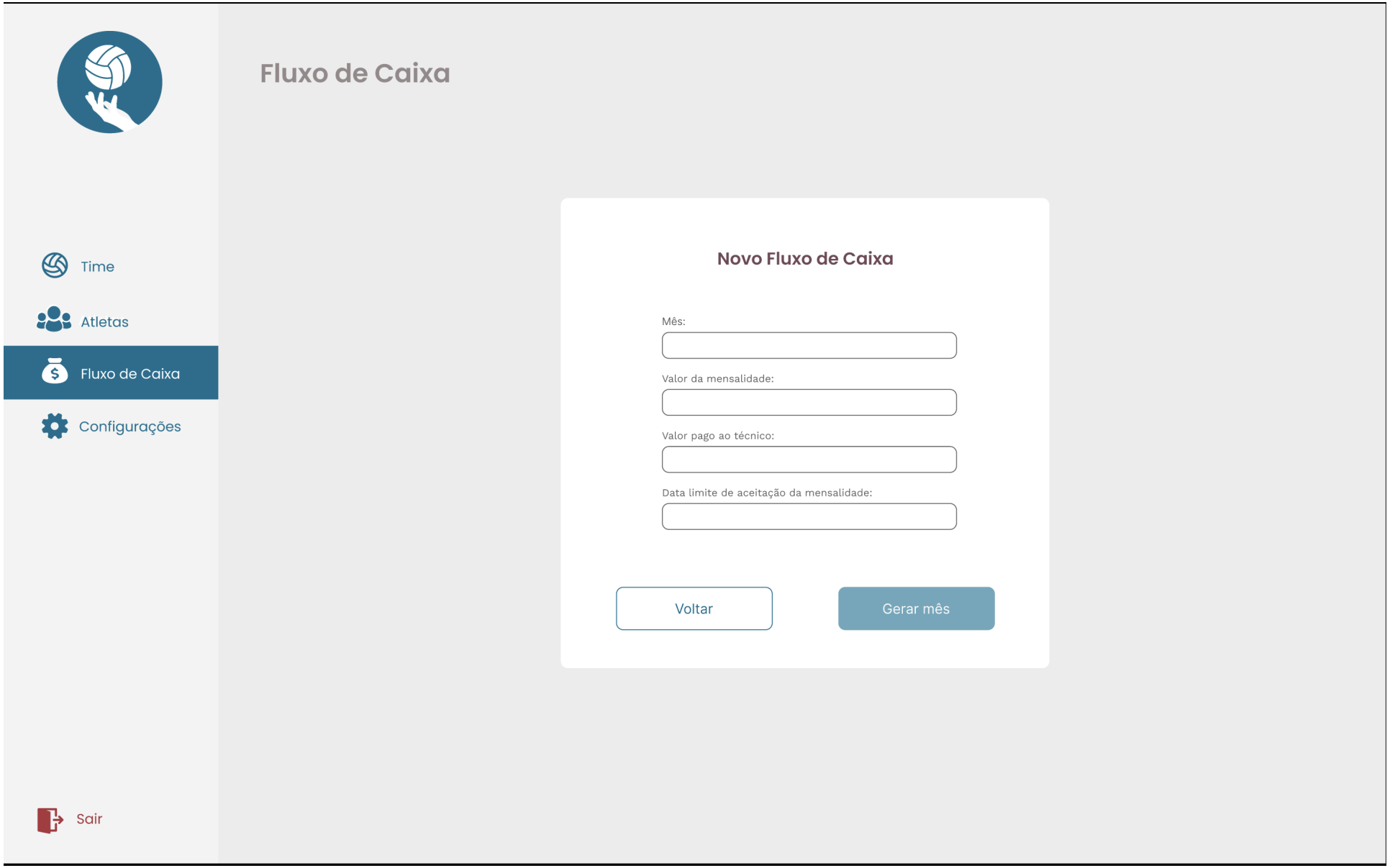


Fonte: Autores

Figura 55 – Página de Fluxo de Caixa Vazio da Ferramenta *Setter*

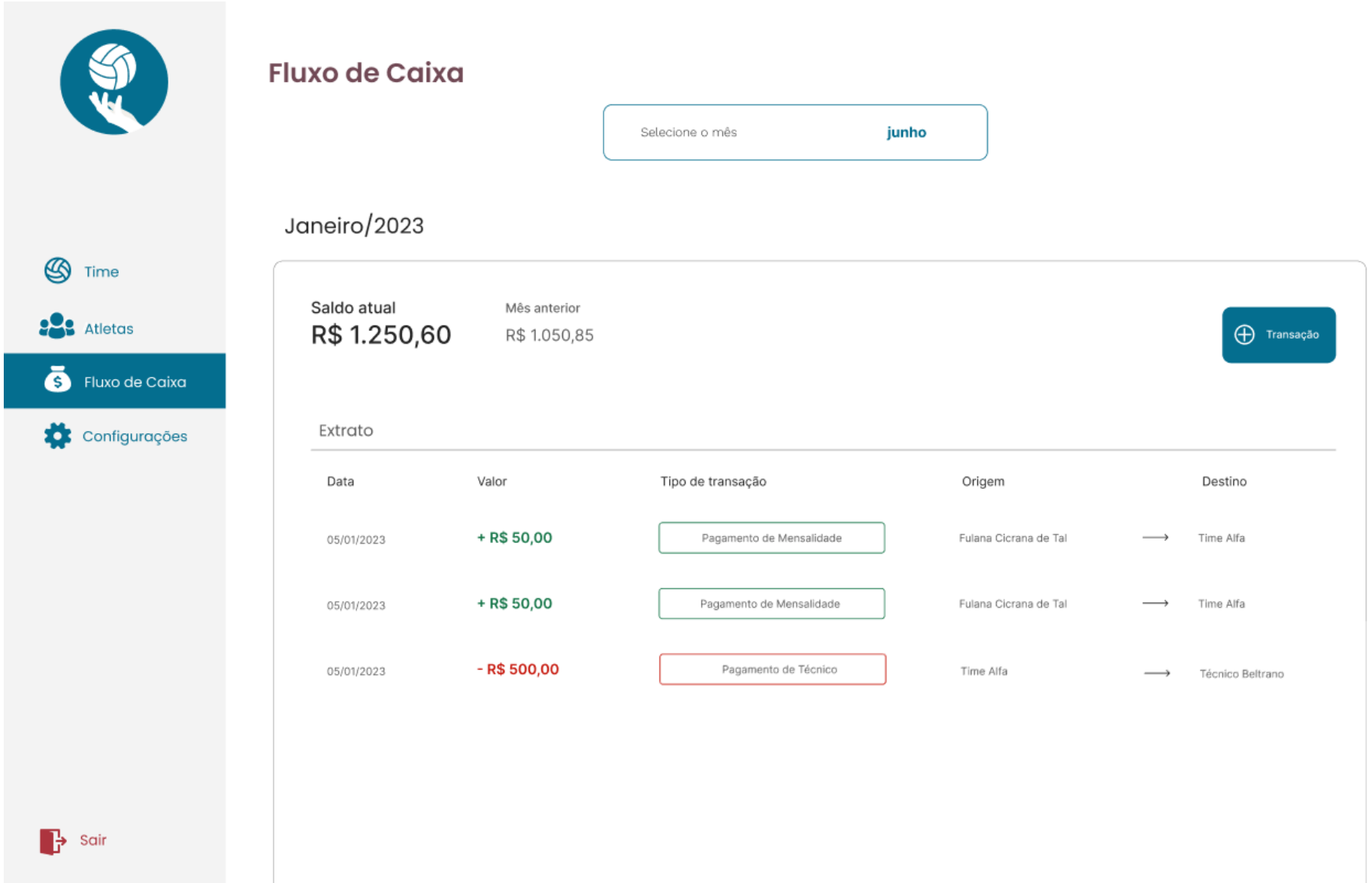
Fonte: Autores

Figura 56 – Página de Adição de Mês da Ferramenta *Setter*



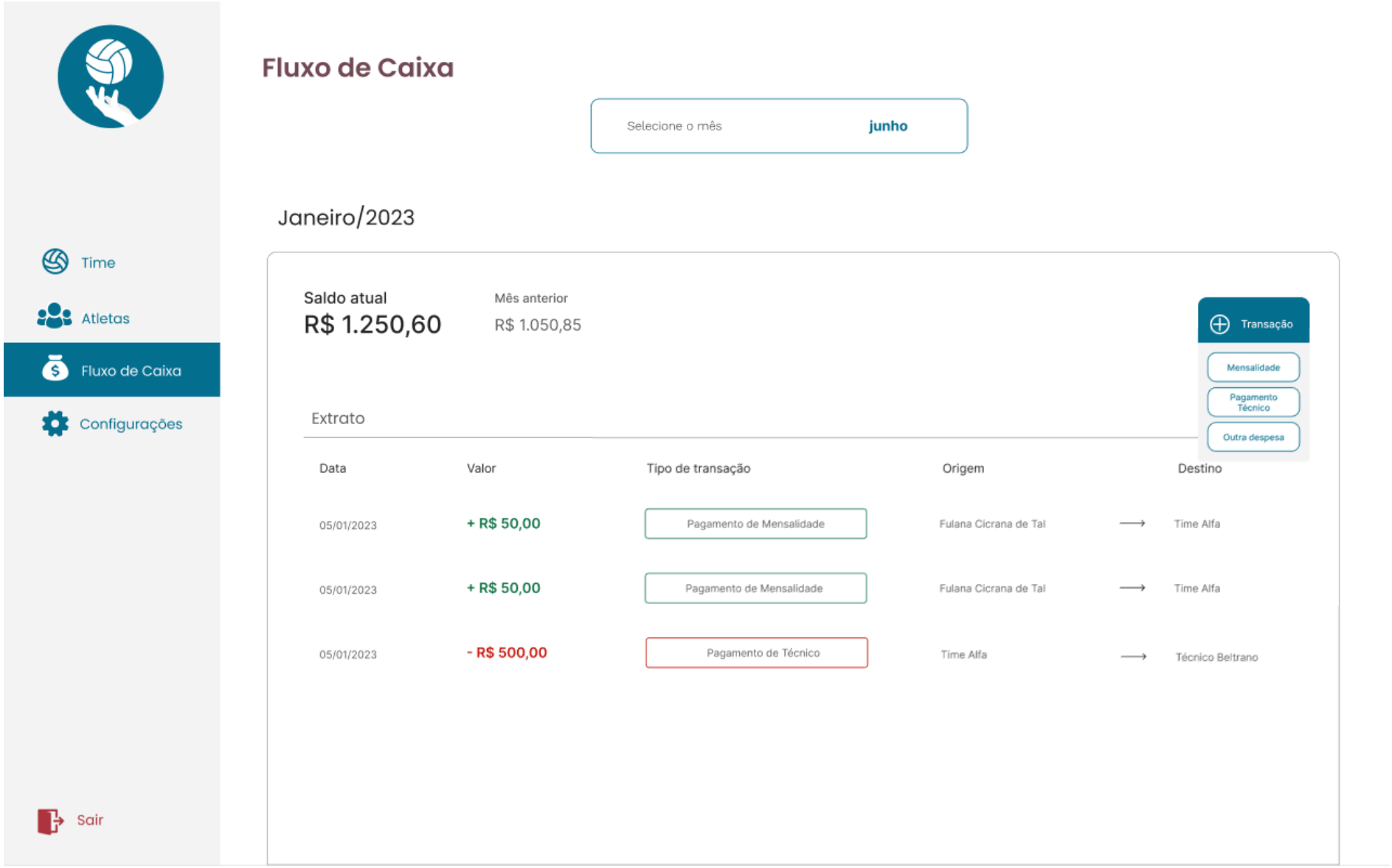
Fonte: Autores

Figura 57 – Página de Fluxo de Caixa da Ferramenta *Setter*



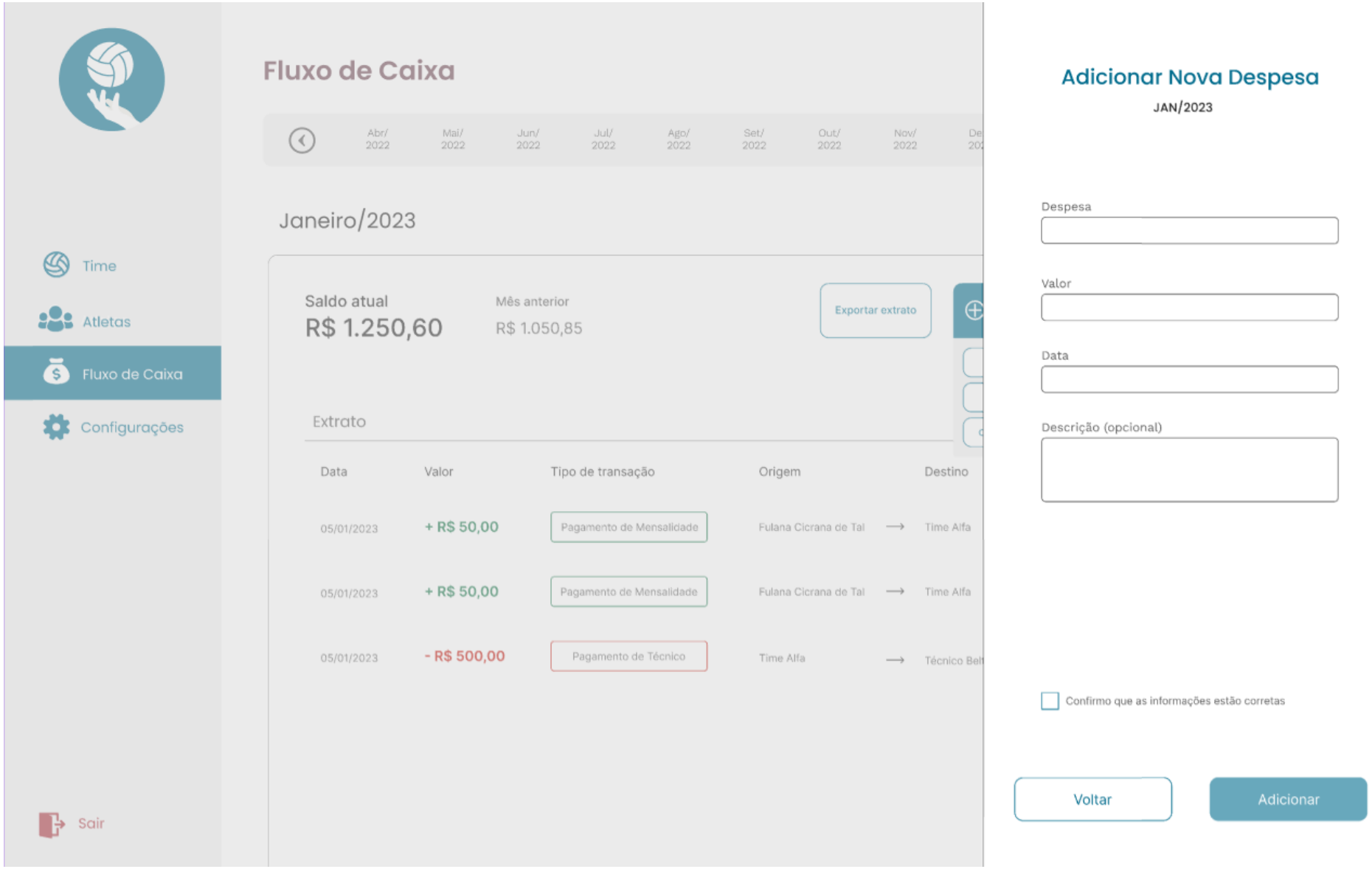
Fonte: Autores

Figura 58 – Opções de Despesa da Ferramenta *Setter*



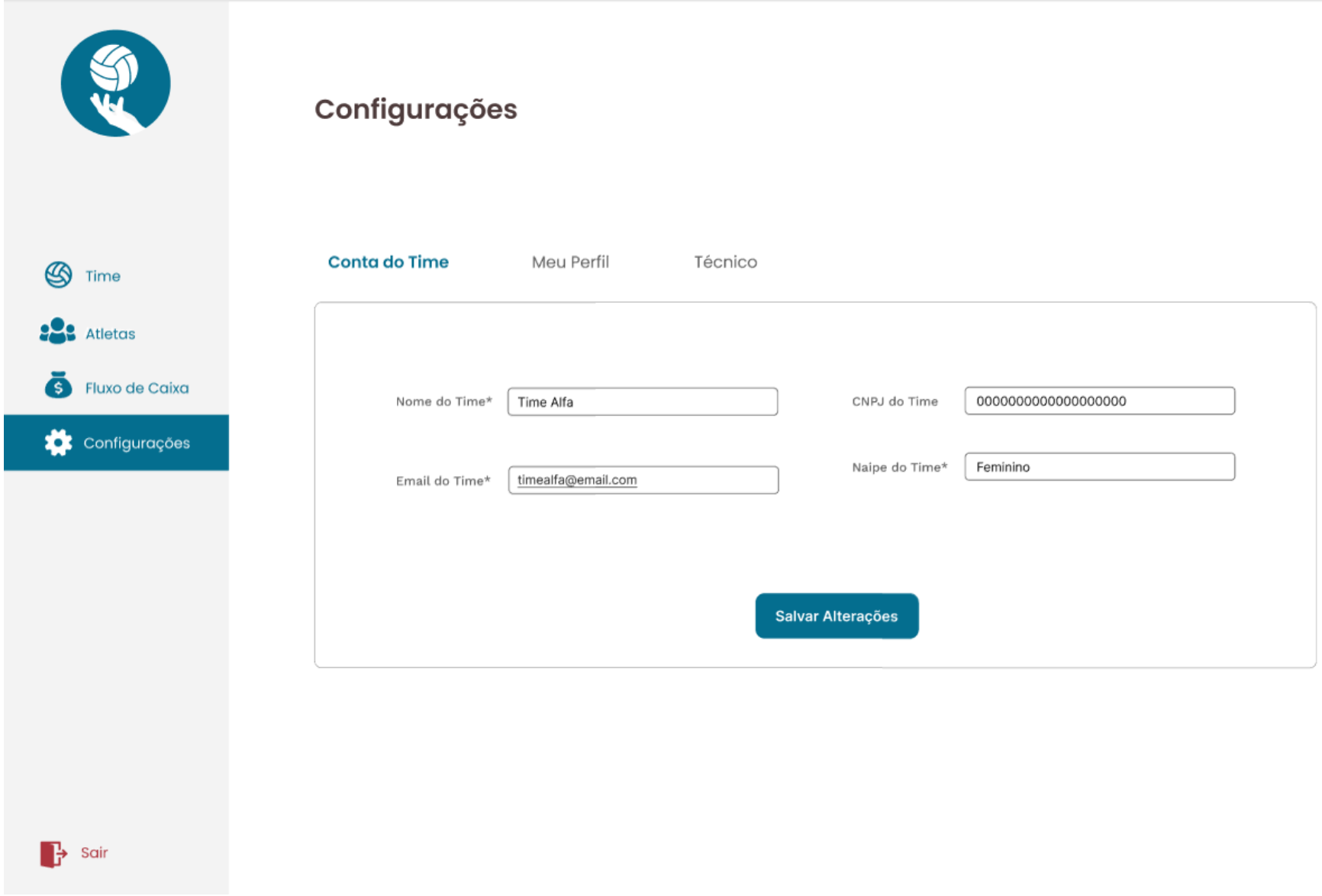
Fonte: Autores

Figura 59 – *Drawer* de Adição de Despesa da Ferramenta *Setter*



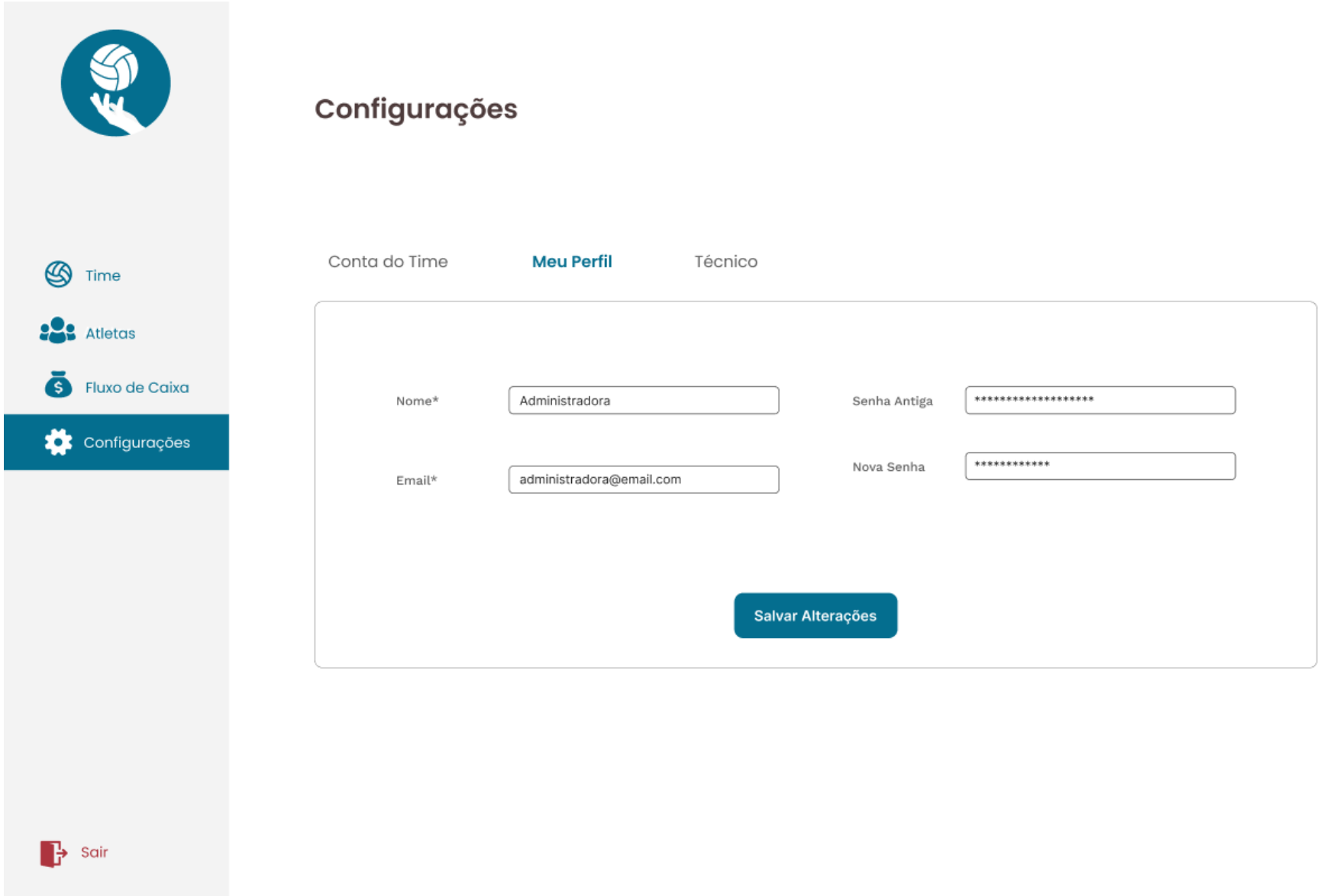
Fonte: Autores

Figura 60 – Página de Configuração do Time da Ferramenta *Setter*



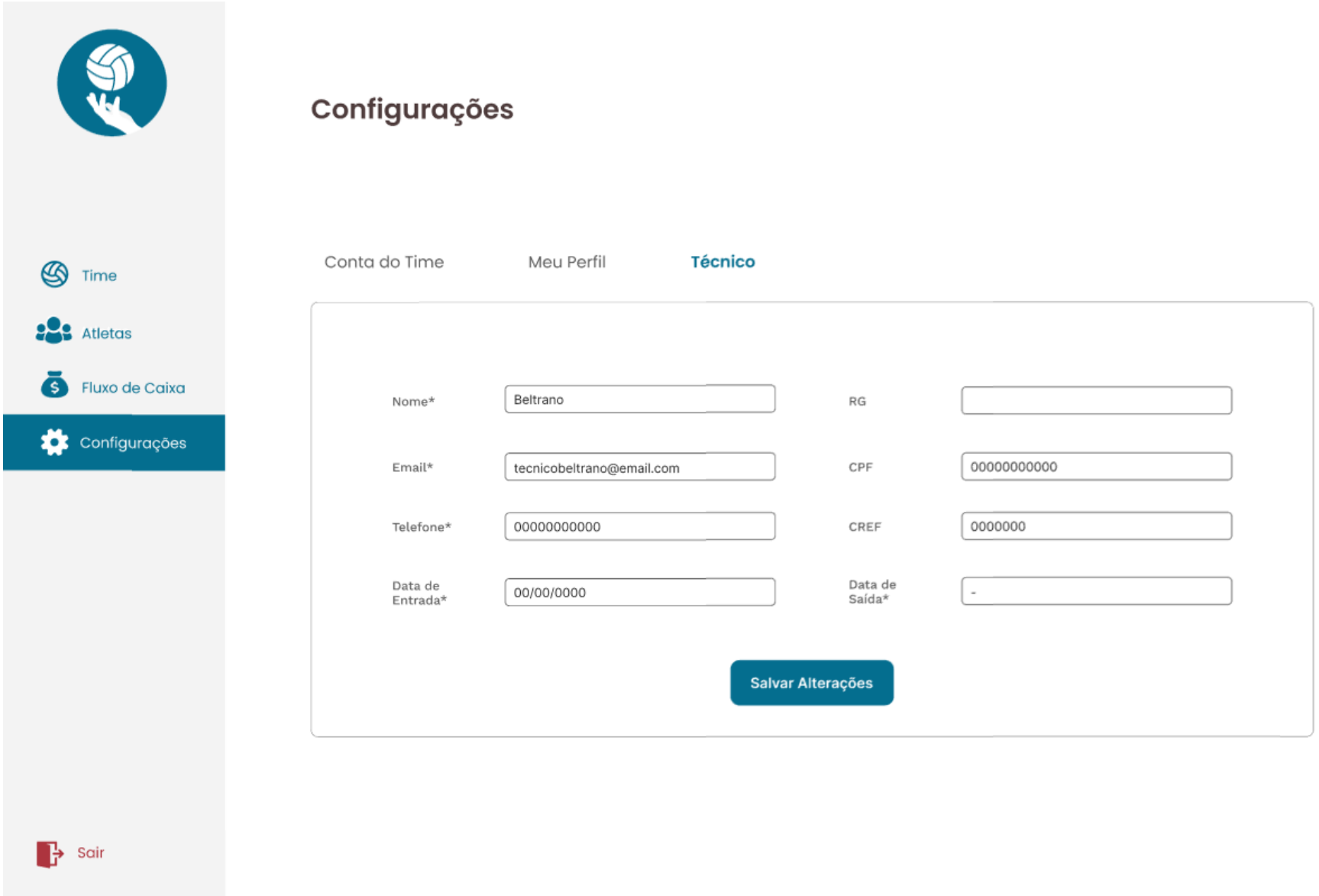
Fonte: Autores

Figura 61 – Página de Configuração de Conta da Ferramenta *Setter*



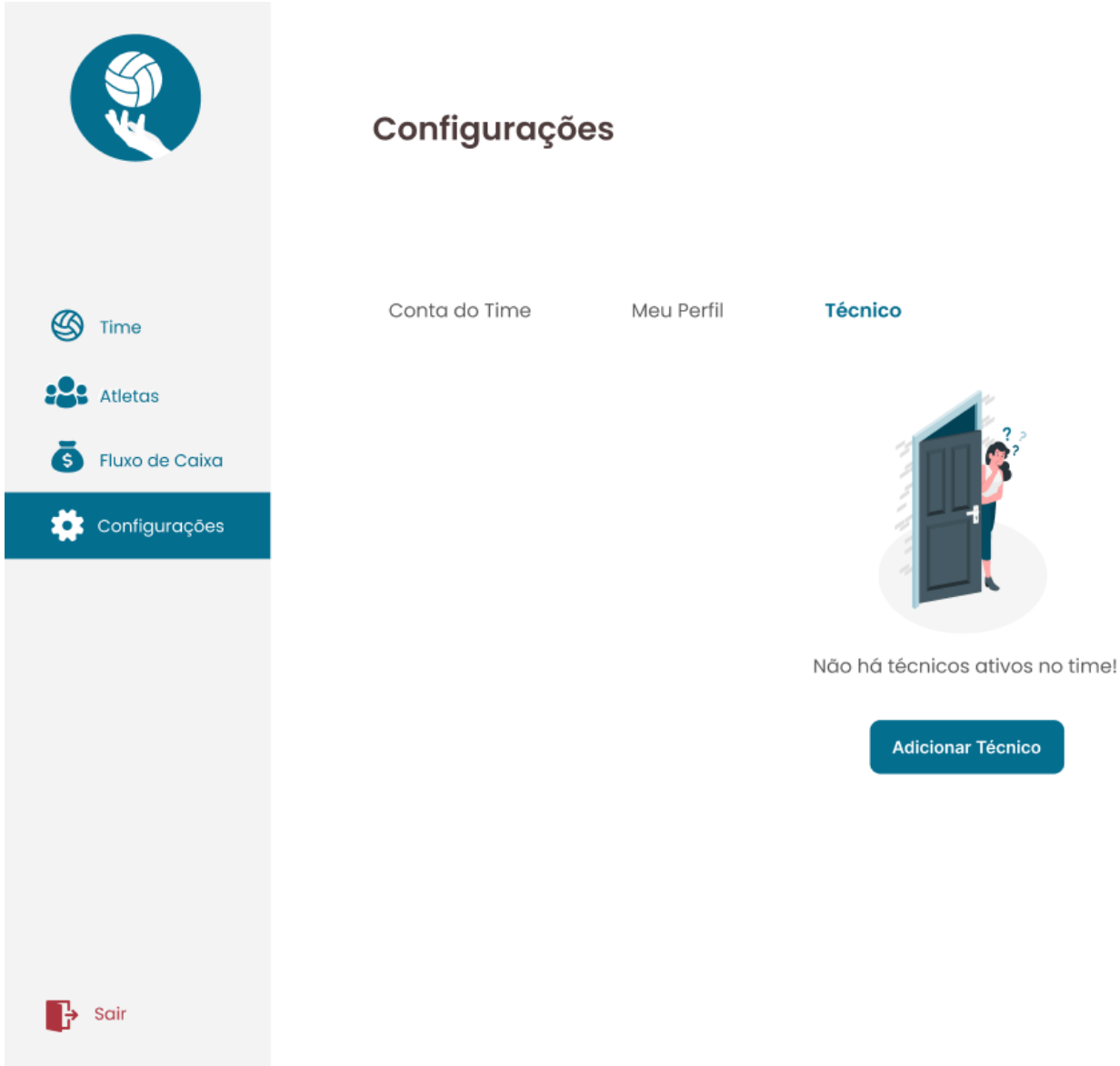
Fonte: Autores

Figura 62 – Página de Configuração de Técnico da Ferramenta *Setter*



Fonte: Autores

Figura 63 – Página de Adição de Técnico da Ferramenta *Setter*



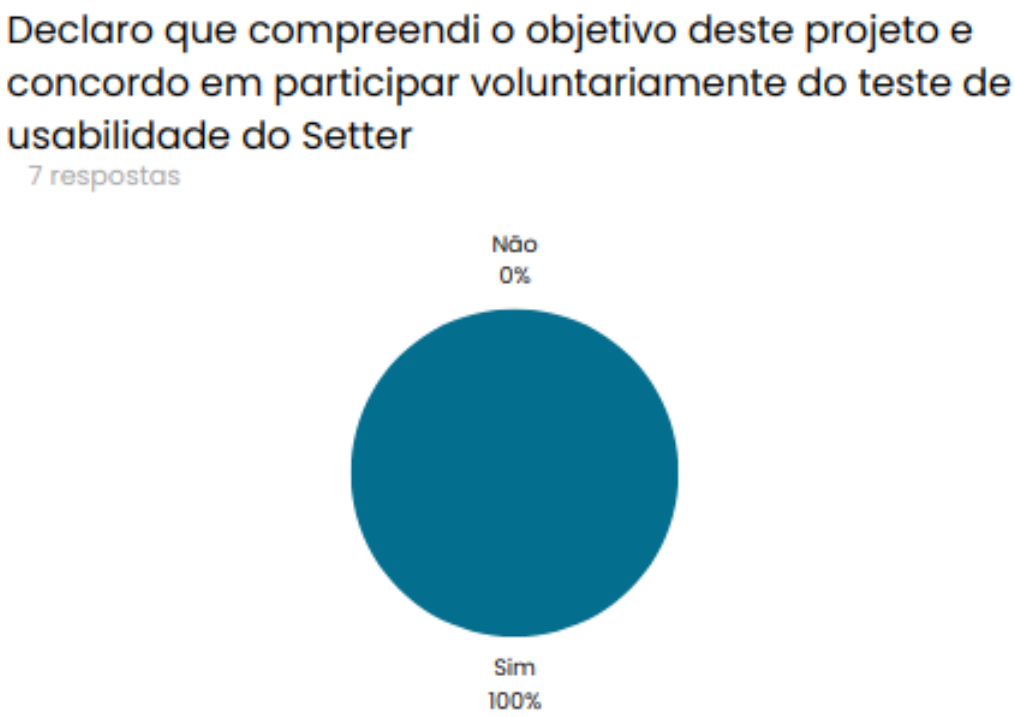
Fonte: Autores

APÊNDICE D – Teste de Usabilidade do MVP

Conforme mencionado no Capítulo 6, para a análise dos dados foi necessário a implementação de um teste de usabilidade da aplicação, no escopo da sua primeira versão, o MVP. A aplicação teve a primeira *release* lançada no dia 31 de junho de 2023, e conta com todo o escopo definido para o MVP. Esta release foi documentada no *Github* (Seção 3.3.2), local onde está armazenado o código fonte, e possui as especificações tanto do Frontend[1] quanto do Backend[2].

O Teste de Usabilidade foi realizado com os *stakeholders* que acompanharam todo o processo de desenvolvimento do presente trabalho. As imagens a seguir apresentam, em sua completude, todos os resultados obtidos pela aplicação do teste. As suas respectivas análises se encontram na Seção 6.3.

Figura 64 – Consentimento dos Participantes do Questionário em Fornecer Dados para o Trabalho

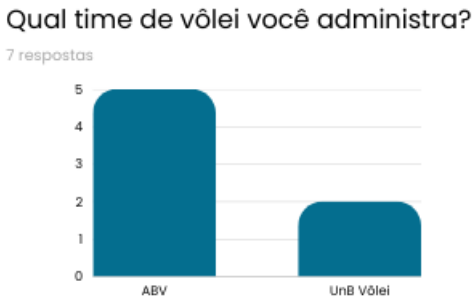


Fonte: Autores

[1] Disponível em: <https://github.com/Setter-TCC/SetterFrontend/releases/tag/v1.0>. Acesso em: 09 jul. 2023.

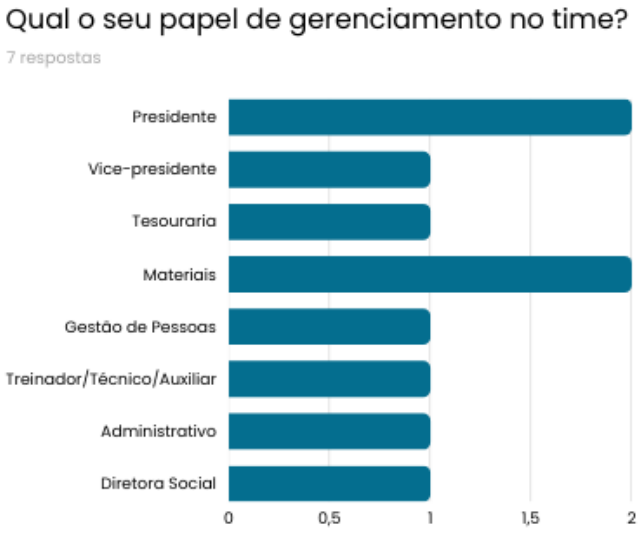
[2] Disponível em: <https://github.com/Setter-TCC/SetterBackend/releases/tag/v1.0>. Acesso em 09 jul. 2023.

Figura 65 – Quantidade de Administradores por Time Respondentes ao Questionário



Fonte: Autores

Figura 66 – Papel de Gerenciamento dos Administradores Respondentes ao Questionário

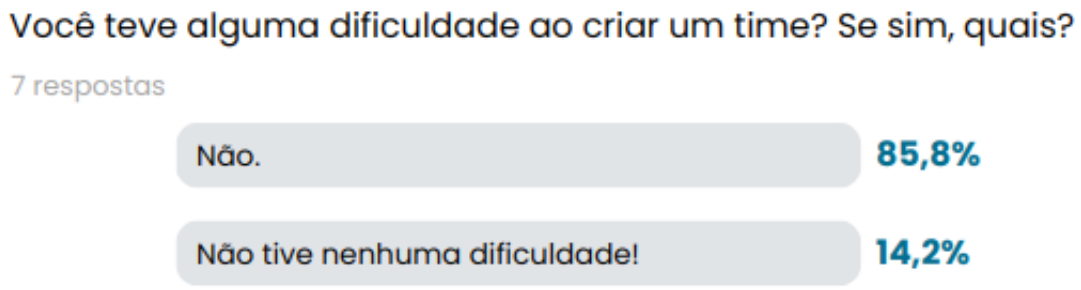


Fonte: Autores

Figura 67 – Porcentagem de Administradores Respondentes ao Questionário que encontraram a Seção para Criar um Time

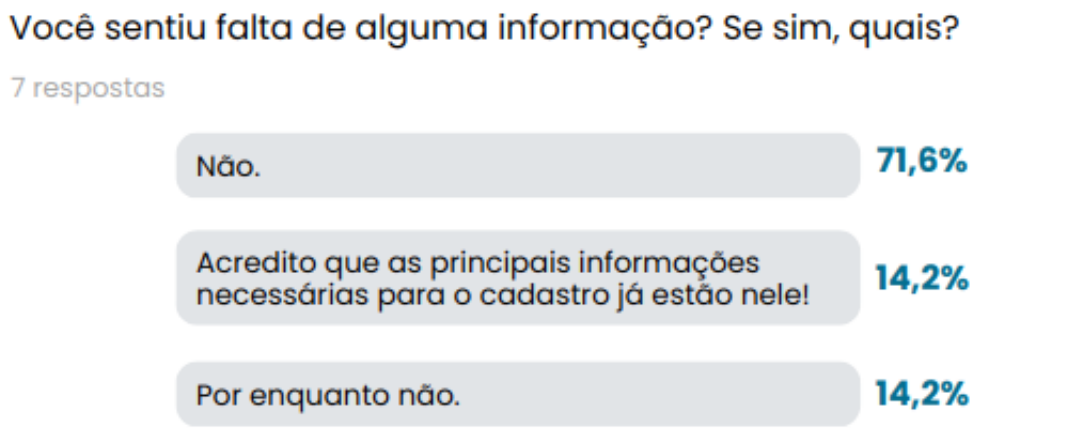
Fonte: Autores

Figura 68 – Dificuldades Encontradas pelos Administradores Respondentes ao Criar um Time



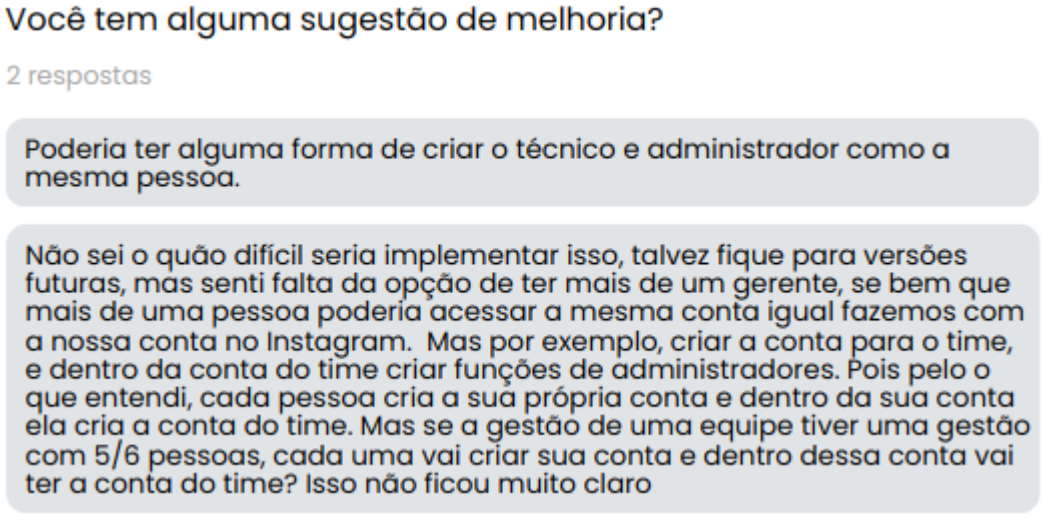
Fonte: Autores

Figura 69 – Falta de Informações encontradas pelos Administradores Respondentes ao Criar um Time



Fonte: Autores

Figura 70 – Sugestões de Melhorias dadas pelos Administradores Respondentes ao Criar um Time

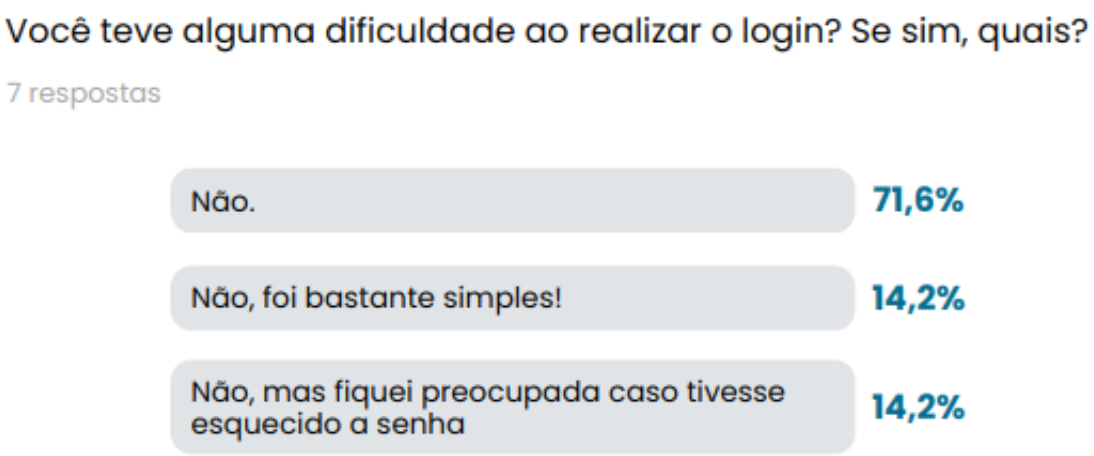


Fonte: Autores

Figura 71 – Porcentagem de Administradores Respondentes ao Questionário que encontraram a Seção para *Login*

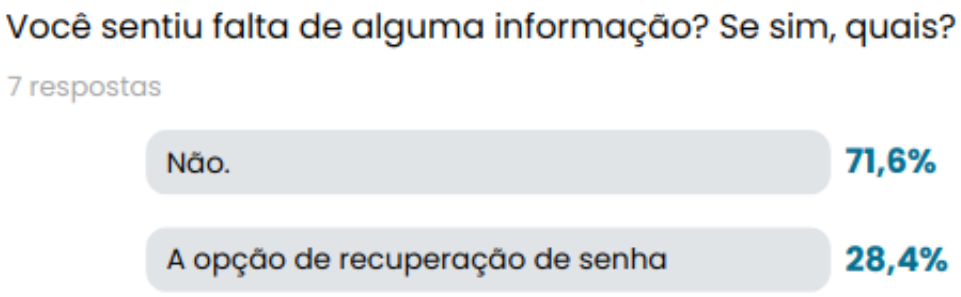
Fonte: Autores

Figura 72 – Dificuldades Encontradas pelos Administradores Respondentes ao Realizar *Login*



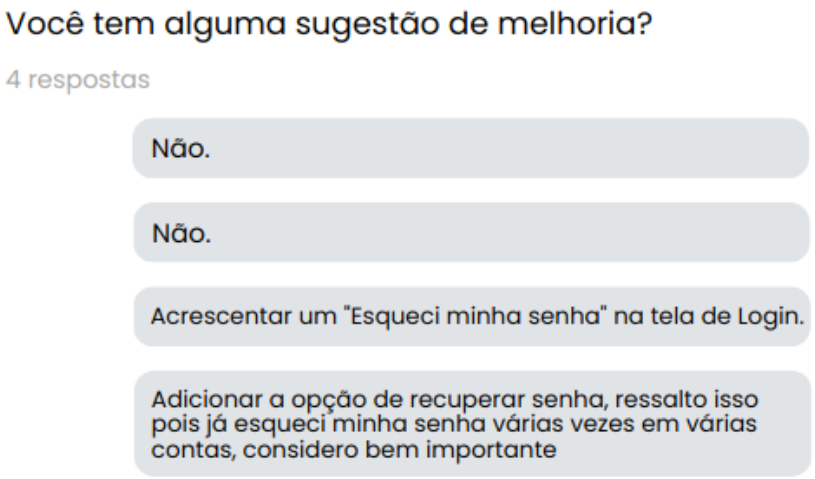
Fonte: Autores

Figura 73 – Falta de Informações encontradas pelos Administradores Respondentes ao Realizar *Login*



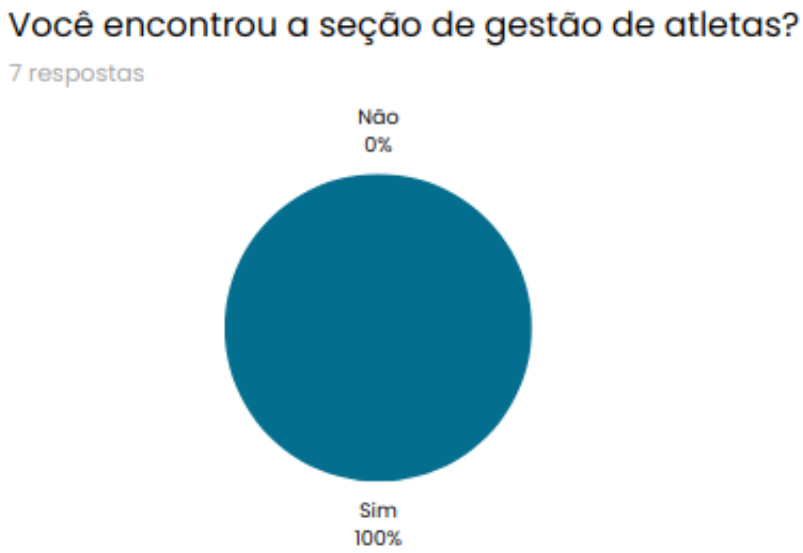
Fonte: Autores

Figura 74 – Sugestões de Melhorias dadas pelos Administradores Respondentes ao Realizar *Login*



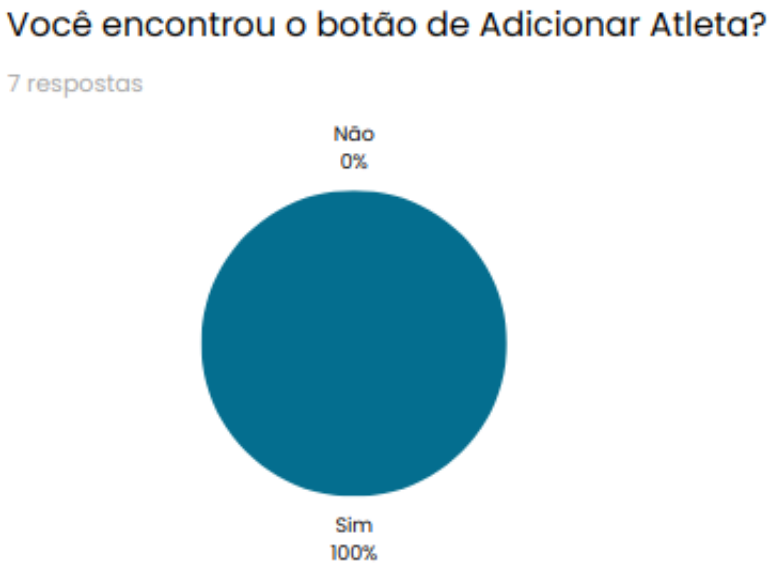
Fonte: Autores

Figura 75 – Porcentagem de Administradores Respondentes ao Questionário que encontraram a Seção para Gestão de Atletas



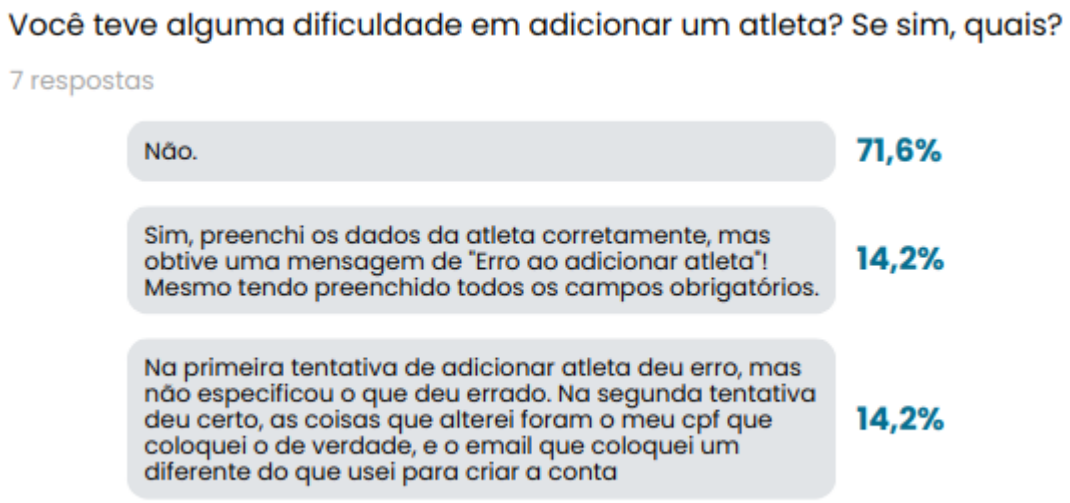
Fonte: Autores

Figura 76 – Porcentagem de Administradores Respondentes ao Questionário que encontraram o botão de Adicionar Atleta



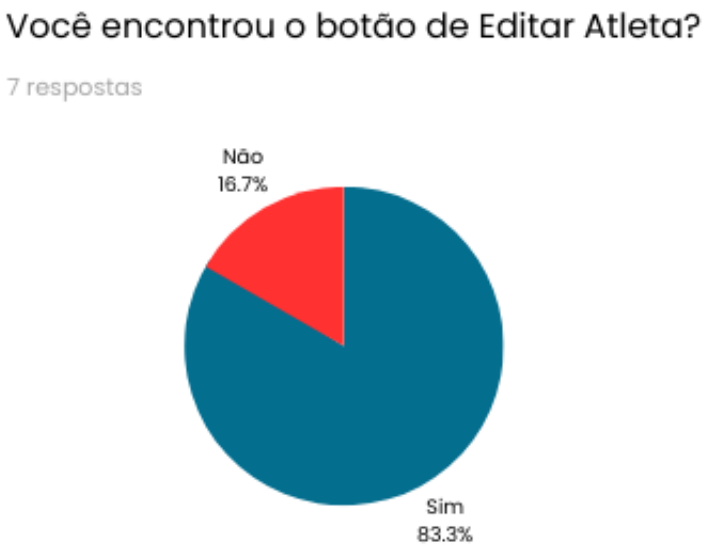
Fonte: Autores

Figura 77 – Dificuldades Encontradas pelos Administradores Respondentes ao Adicionar Atleta



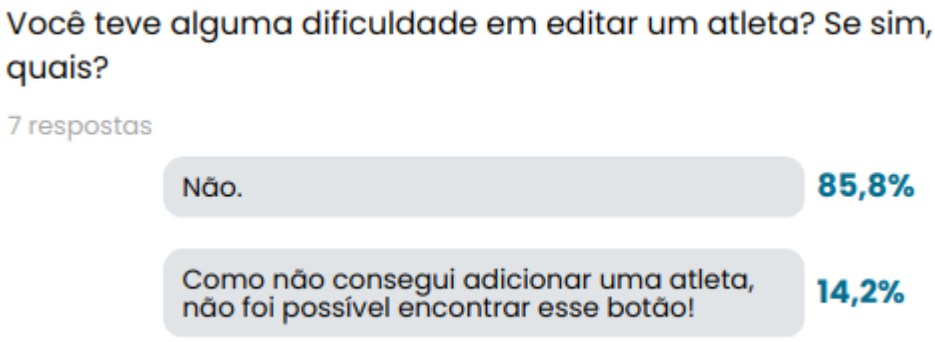
Fonte: Autores

Figura 78 – Porcentagem de Administradores Respondentes ao Questionário que encontraram o botão de Editar Atleta



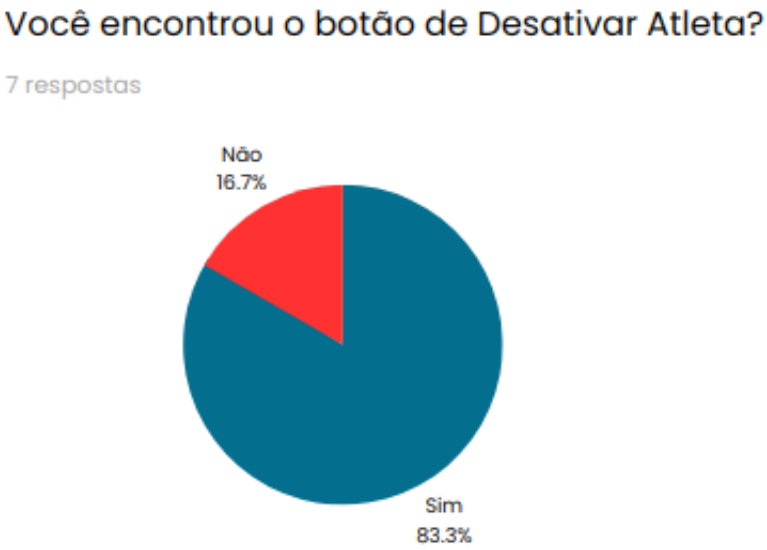
Fonte: Autores

Figura 79 – Dificuldades Encontradas pelos Administradores Respondentes ao Editar Atleta



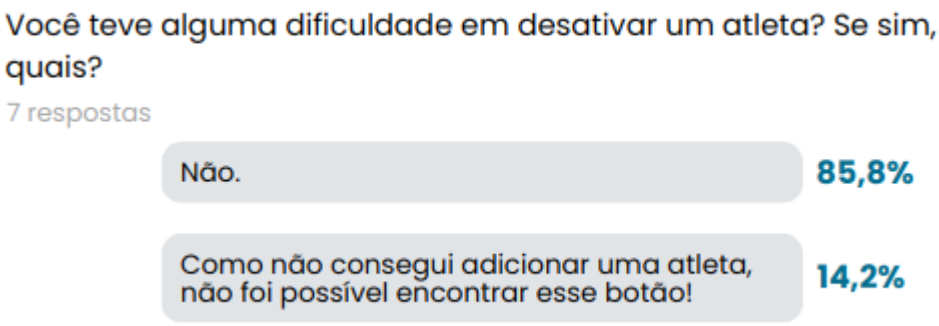
Fonte: Autores

Figura 80 – Porcentagem de Administradores Respondentes ao Questionário que encontraram o botão de Desativar Atleta



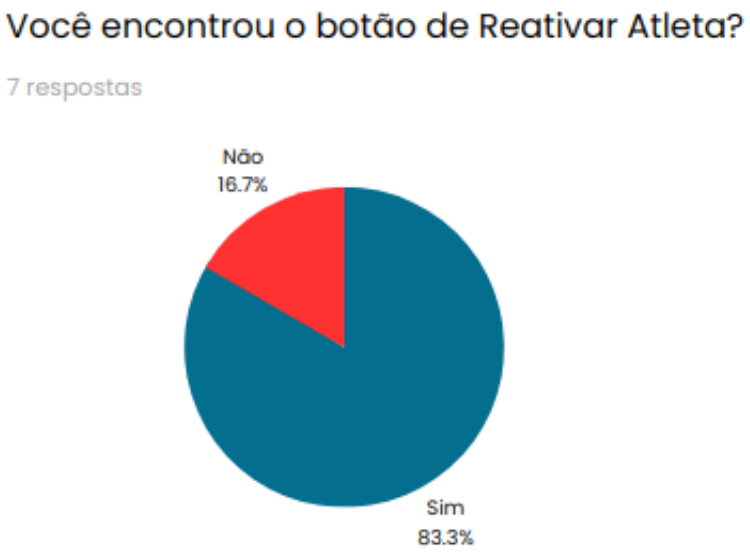
Fonte: Autores

Figura 81 – Dificuldades Encontradas pelos Administradores Respondentes ao Desativar Atleta



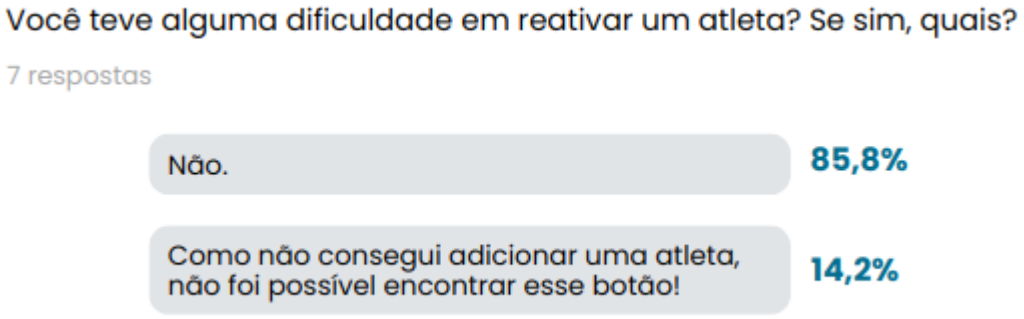
Fonte: Autores

Figura 82 – Porcentagem de Administradores Respondentes ao Questionário que encontraram o botão de Reativar Atleta



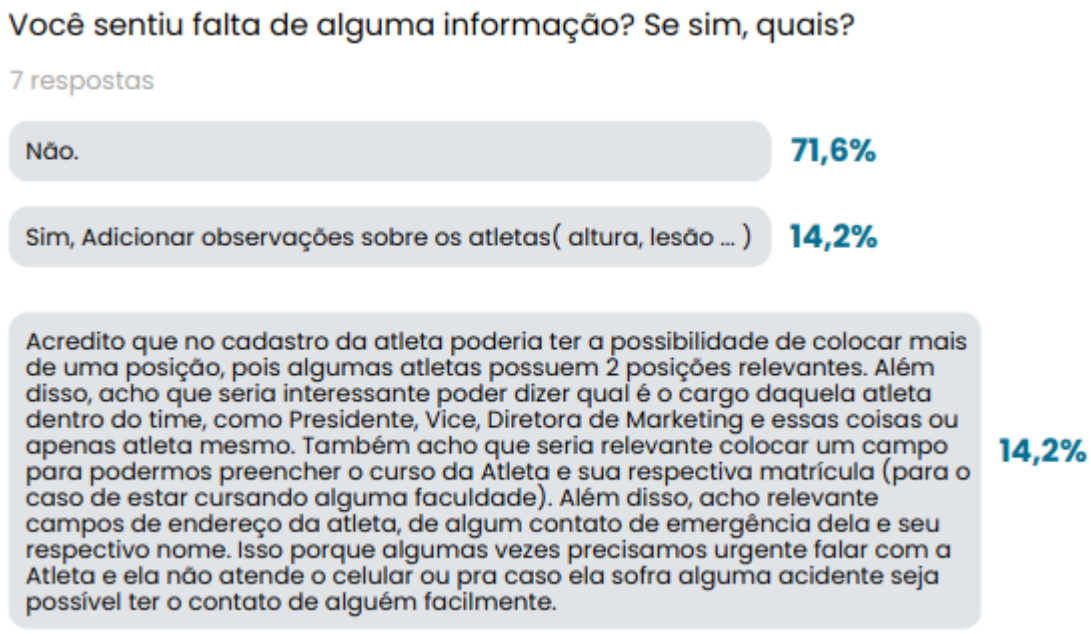
Fonte: Autores

Figura 83 – Dificuldades Encontradas pelos Administradores Respondentes ao Reativar Atleta



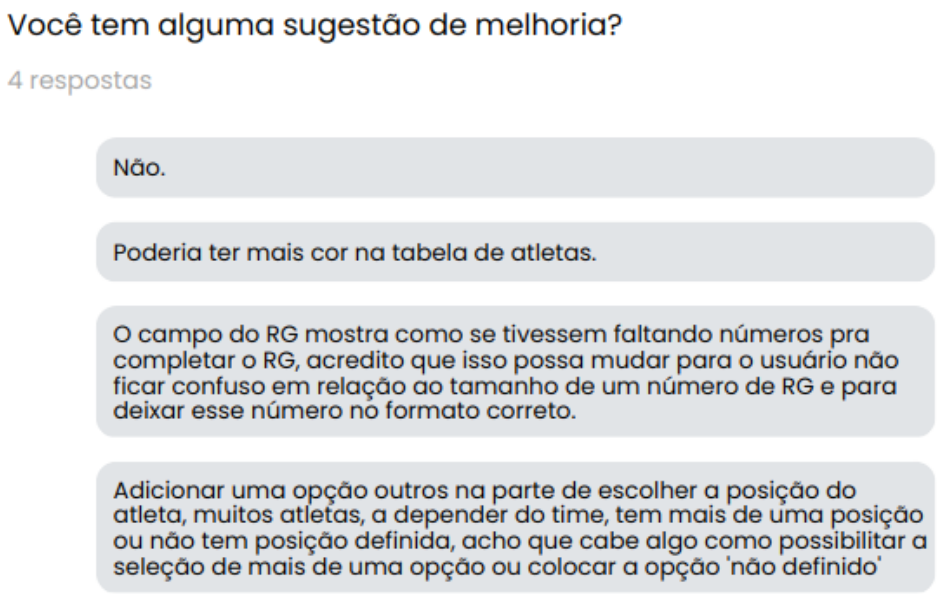
Fonte: Autores

Figura 84 – Falta de Informações encontradas pelos Administradores Respondentes ao realizar a Gestão de Atletas



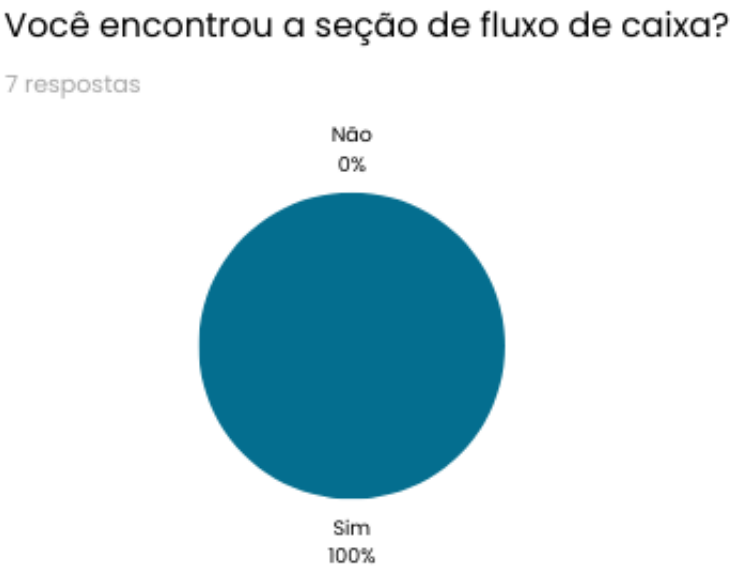
Fonte: Autores

Figura 85 – Sugestões de Melhorias dadas pelos Administradores Respondentes ao realizar a Gestão de Atletas



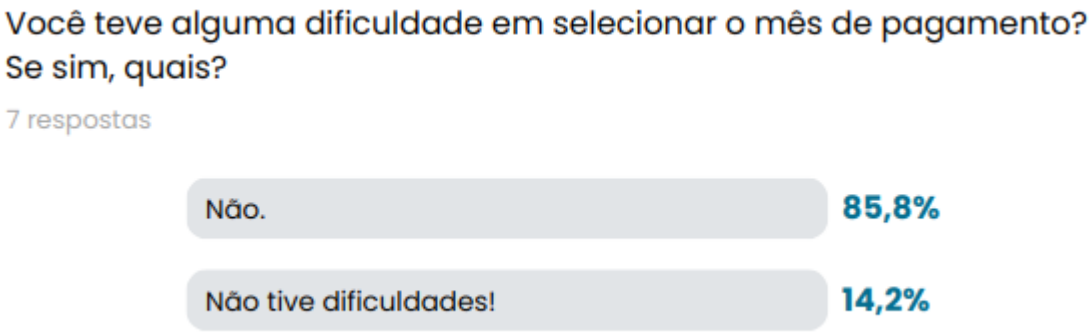
Fonte: Autores

Figura 86 – Porcentagem de Administradores Respondentes ao Questionário que encontraram a Seção de Fluxo de Caixa



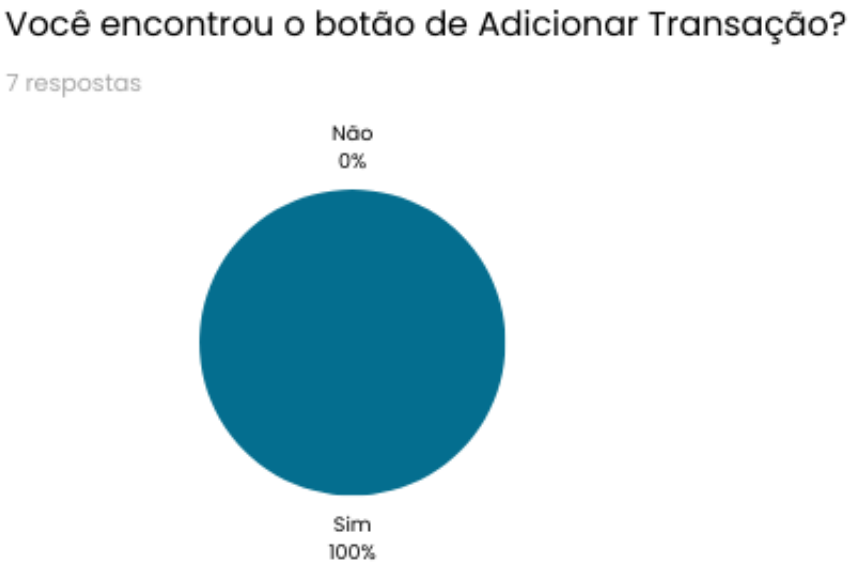
Fonte: Autores

Figura 87 – Dificuldades Encontradas pelos Administradores Respondentes ao Selecionar o mês do Fluxo de Caixa



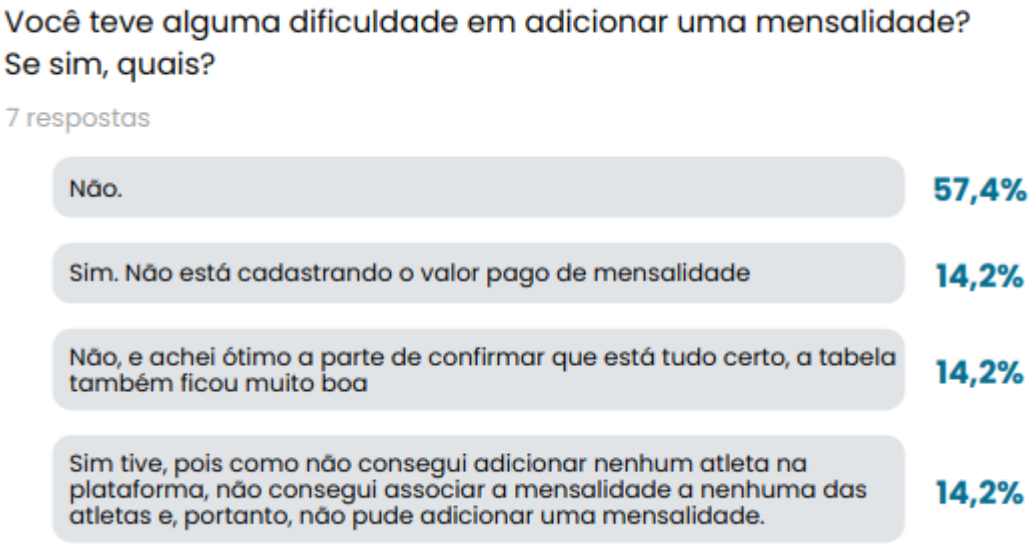
Fonte: Autores

Figura 88 – Porcentagem de Administradores Respondentes ao Questionário que encontraram o botão de Adicionar Transação



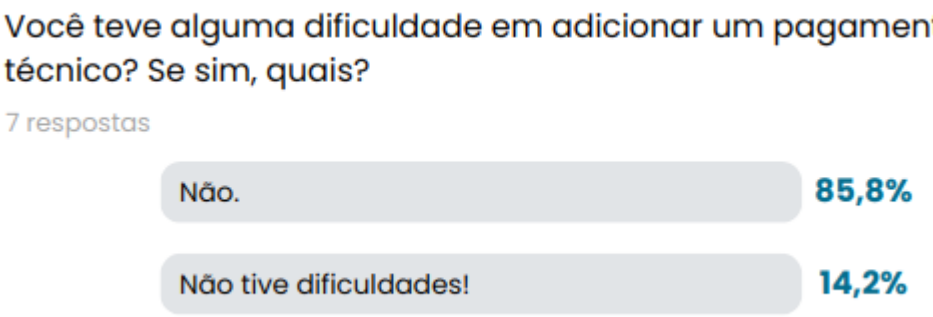
Fonte: Autores

Figura 89 – Dificuldades Encontradas pelos Administradores Respondentes ao Adicionar Mensalidade



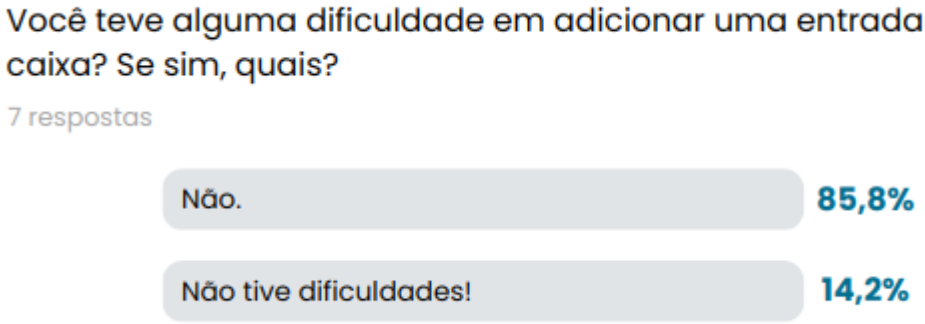
Fonte: Autores

Figura 90 – Dificuldades Encontradas pelos Administradores Respondentes ao Adicionar Pagamento do Técnico



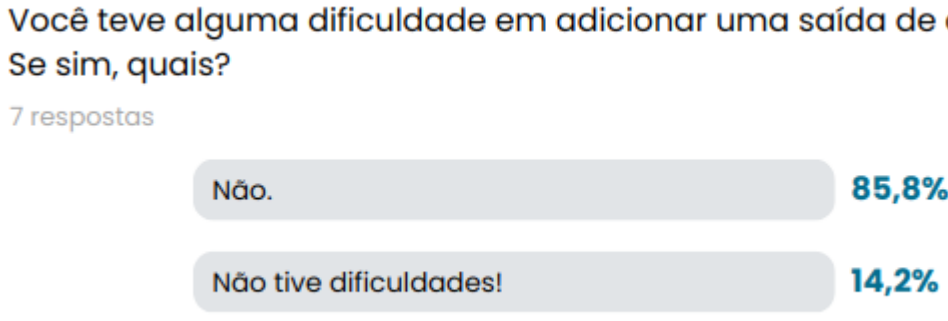
Fonte: Autores

Figura 91 – Dificuldades Encontradas pelos Administradores Respondentes ao Adicionar Entrada de Caixa



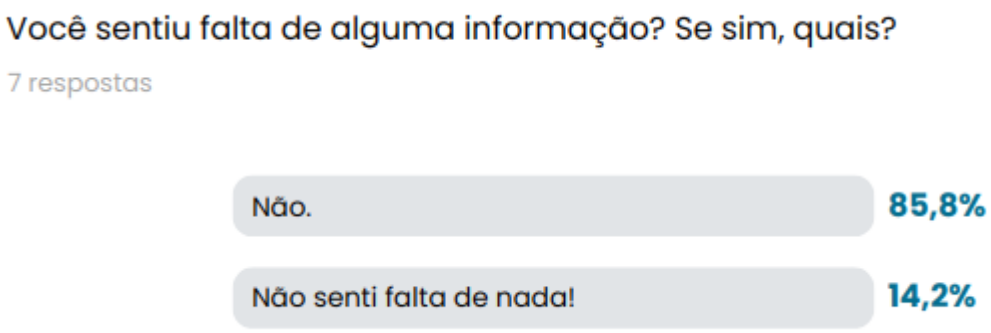
Fonte: Autores

Figura 92 – Dificuldades Encontradas pelos Administradores Respondentes ao Adicionar Saída de Caixa



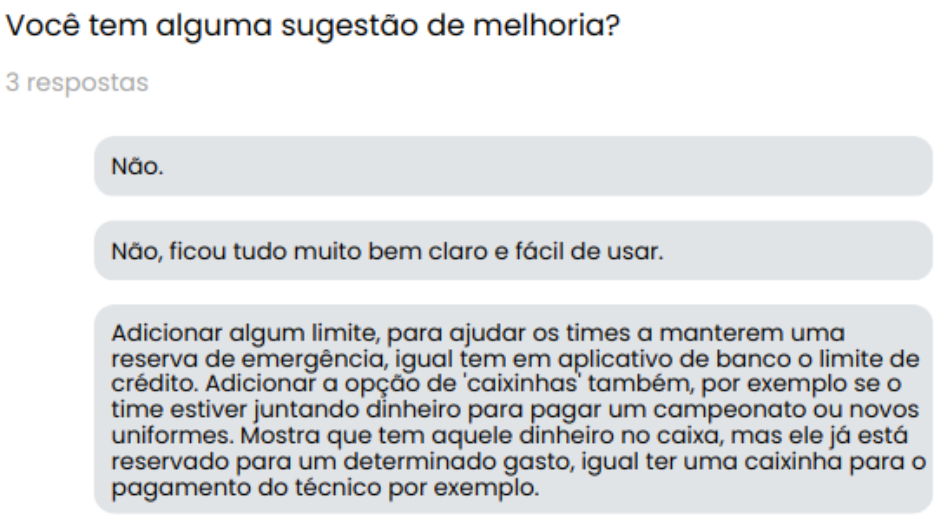
Fonte: Autores

Figura 93 – Falta de Informações encontradas pelos Administradores Respondentes ao interagir com o Fluxo de Caixa



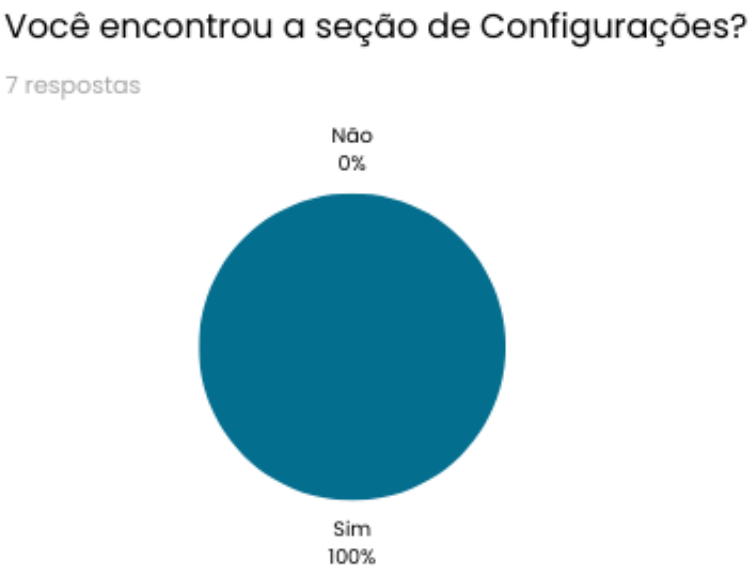
Fonte: Autores

Figura 94 – Sugestões de Melhorias dadas pelos Administradores Respondentes ao ao interagir com o Fluxo de Caixa



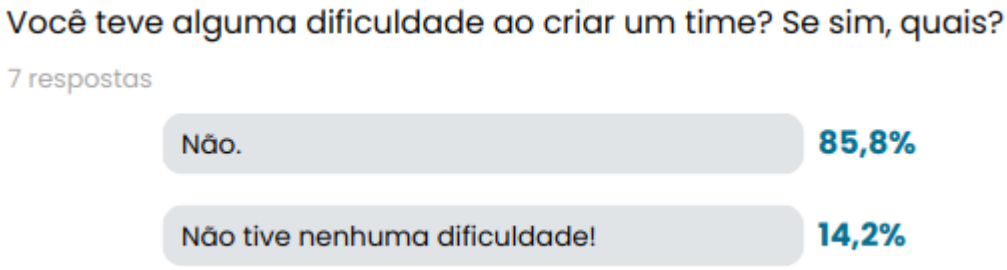
Fonte: Autores

Figura 95 – Porcentagem de Administradores Respondentes ao Questionário que encontraram a Seção de Configurações



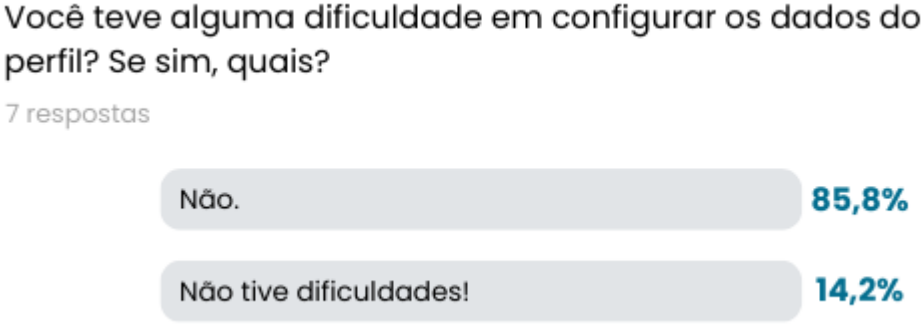
Fonte: Autores

Figura 96 – Dificuldades Encontradas pelos Administradores Respondentes ao Configurar os dados da Conta do Time



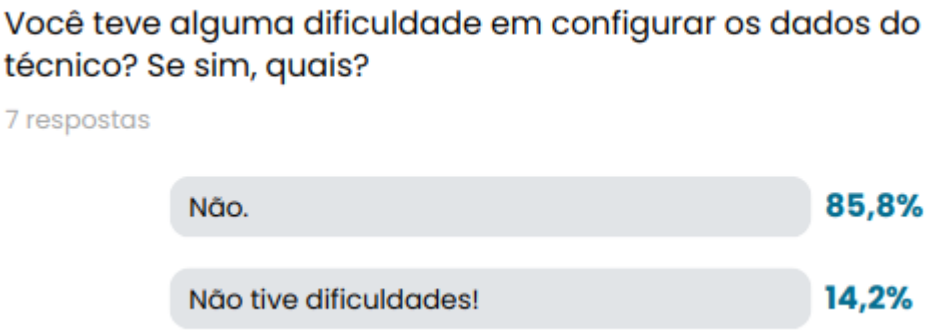
Fonte: Autores

Figura 97 – Dificuldades Encontradas pelos Administradores Respondentes ao Configurar os dados da Conta do Administrador



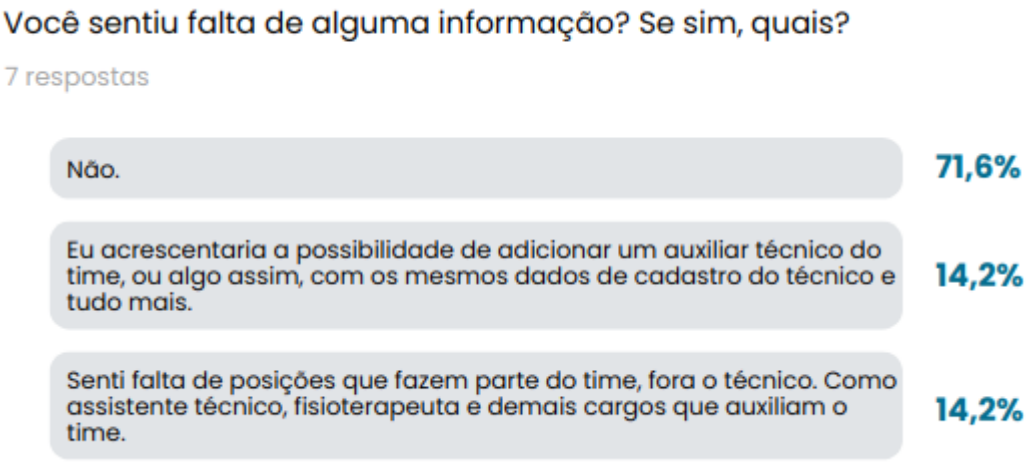
Fonte: Autores

Figura 98 – Dificuldades Encontradas pelos Administradores Respondentes ao Configurar os dados do Técnico



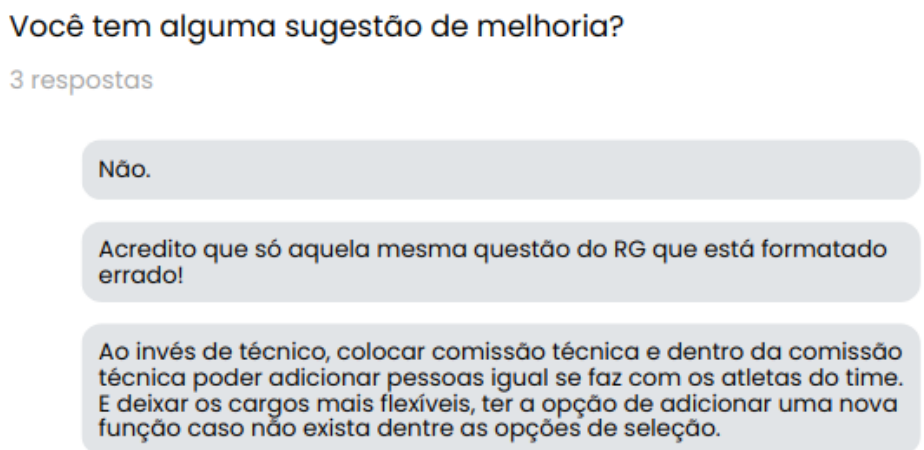
Fonte: Autores

Figura 99 – Falta de Informações encontradas pelos Administradores Respondentes ao interagir com o Fluxo de Caixa



Fonte: Autores

Figura 100 – Sugestões de Melhorias dadas pelos Administradores Respondentes ao ao interagir com o Fluxo de Caixa



Fonte: Autores

APÊNDICE E – TCLE

Figura 101 – Termo de Consentimento Livre Esclarecido - TCLE

Termo de Consentimento - Autorização

Você está sendo convidado(a) para ser participante do Projeto de pesquisa intitulado **Setter: Software Especializado em Gerenciamento de Times Amadores de Voleibol**, de responsabilidade dos pesquisadores Hugo Sobral de Lima Salomão e Micaella Lorraine Gouveia de Lima.

Leia cuidadosamente o que se segue e pergunte sobre qualquer dúvida que você tiver. Caso se sinta esclarecido(a) sobre as informações que estão neste Termo e aceite fazer parte do estudo, peço que assine ao final deste documento. Saiba que você tem total direito de não querer participar.

1. O trabalho tem por objetivo realizar um levantamento de dados sobre a experiência de usuário e usabilidade do aplicativo Setter.
2. A participação nesta pesquisa consistirá na utilização do aplicativo Setter com o objetivo de testar algumas funcionalidades específicas do aplicativo. A partir do teste realizado será passado um formulário para a realização da avaliação de aspectos relacionados a usabilidade e experiência de usuário do aplicativo.
3. Os benefícios da participação nesta pesquisa serão melhorias no aplicativo Setter, que visa o gerenciamento de times de voleibol.
4. Os participantes não terão nenhuma despesa ao participar da pesquisa e poderão retirar sua concordância na continuidade da pesquisa a qualquer momento.
5. Não há nenhum valor econômico a receber ou a pagar aos voluntários pela participação, no entanto, caso haja qualquer despesa decorrente desta participação haverá o seu ressarcimento pelos pesquisadores.
6. Caso ocorra algum dano comprovadamente decorrente da participação no estudo, os voluntários poderão pleitear indenização, segundo as determinações do Código Civil (Lei no 10.406 de 2002) e das Resoluções 466/12 e 510/16 do Conselho Nacional de Saúde. 7. O nome dos participantes será mantido em sigilo, assegurando assim a sua privacidade, e se desejarem terão livre acesso a todas as informações e esclarecimentos adicionais sobre o estudo e suas consequências, enfim, tudo o que queiram saber antes, durante e depois da sua participação.
8. Os dados coletados serão utilizados única e exclusivamente para fins desta pesquisa, e os resultados poderão ser publicados.

Qualquer dúvida, pedimos a gentileza de entrar em contato com Hugo Sobral de Lima Salomão, pesquisador responsável pela pesquisa, telefone: (61) 981468499, e-mail: hugosobral10@gmail.com, ou com Micaella Lorraine Gouveia de Lima, pesquisadora responsável pela pesquisa, telefone: (61) 986082374, e-mail: micaella2212@gmail.com.

Eu, _____, natural de _____, CPF _____, concordo em ser participante do Projeto de pesquisa acima descrito.

_____

Brasília, 28 de junho de 2023.

Fonte: Autores