Desenvolvimento de um Sistema Web para Cálculos Trabalhistas Utilizando *React* e *Firebase*

Bruno Ítalo Sampaio Ferreira[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO

sampaio.bruno9@gmai.com

Resumo. *A profissão jurídica passou por mudanças significativas nas últimas décadas, com ênfase no crescimento do empreendedorismo e na inovação tecnológica. Em particular, a utilização da tecnologia torna-se cada vez mais importante para os advogados, permitindo-lhes incorporar software às suas atividades profissionais. Isto não só agregou valor aos serviços prestados, mas também reduziu custos e liberou tempo para os profissionais se concentrarem em atividades estratégicas. Este artigo aborda o desenvolvimento de uma plataforma dedicada à realização de cálculos trabalhistas direcionados a escritórios de advocacia, que visou simplificar e otimizar a complexa tarefa de realizar os cálculos trabalhistas além de possibilitar o gerenciamento de clientes utilizando as tecnologias de desenvolvimento React e Firebase, detalhando as etapas seguidas para propor um produto tecnológico funcional e eficiente.*

1. Introdução

Nas últimas décadas, a advocacia passou por profundas transformações relacionadas ao empreendedorismo, refletindo as mudanças no mundo dos negócios e na sociedade como um todo. Tradicionalmente vista como uma profissão voltada principalmente para a prestação de serviços legais, a advocacia evoluiu para abraçar um espírito empreendedor que sustente e aprimore a entrega do serviço ao cliente [Carvalho, 2020]. Hoje, observa-se que essas transformações impactam o comportamento dos profissionais do setor em relação aos seus clientes e em suas estratégias de atuação.

Conforme a definição de Drucker [1980], o empreendedorismo é caracterizado pela ação, visão de mercado e adaptabilidade. Ele acrescenta que o papel fundamental do empreendedorismo em uma empresa é capacitar o presente para construir o futuro, promovendo transformações que tornem o negócio uma entidade diferente. Essa abordagem sublinha a importância da tecnologia no aprimoramento das atividades no âmbito da advocacia empreendedora.

O avanço tecnológico não se restringe apenas à adoção de novos produtos, mas também é uma atitude que se traduz em aprimoramento contínuo, adaptando os processos existentes e incorporando novos desenvolvimentos tecnológicos. Essa inovação pode levar a melhorias diretas, adicionando novas qualidades aos produtos, ou a melhorias indiretas, resultando na redução de custos por meio da otimização dos processos [Serio e Vasconcellos, 2009].

Assim, fica evidente o impacto da tecnologia no campo jurídico, uma vez que os avanços tecnológicos possibilitam que os advogados incorporem em suas práticas *softwares* de gestão de escritório e automação de tarefas burocráticas. Isso não apenas agrega

valor aos serviços prestados, mas também reduz custos, liberando tempo precioso para que os profissionais possam se concentrar em atividades estratégicas [Ferreira, 2020].

Esse cenário é particularmente relevante para os advogados da área trabalhista, que enfrentam desafios cada vez mais complexos, especialmente no que diz respeito aos cálculos trabalhistas. Tendo em vista que, a demanda por serviços jurídicos relacionados a questões trabalhistas é constante e crescente, considerando que o Brasil possui um sistema complexo de leis trabalhistas e regulamentos que protegem os direitos dos trabalhadores. Segundo o Relatório Geral da Justiça do Trabalho de 2022, ao longo do ano foram somados 3.161.287 processos recebidos pela Justiça do Trabalho [Tribunal Superior do Trabalho, 2022].

De acordo com um dos *stakeholders* participantes deste trabalho, a execução manual de calcular com precisão os valores a serem reivindicados em ações judiciais é árdua e demorada, dada a grande quantidade de dados e variáveis envolvidas. Essa complexidade dos cálculos trabalhistas representa um problema multifacetado. Além de exigir conhecimento jurídico, a abordagem necessária deve ser altamente organizada e precisa para evitar erros e garantir que os direitos dos trabalhadores sejam adequadamente protegidos.

Isso se torna ainda mais relevante ao considerar o Código de Processo Civil (CPC). A legislação atual estabelece que os advogados apresentem petições iniciais com pedido certo e líquido, seguindo as regras processuais tanto no âmbito civil como no trabalhista. Essa obrigação legal apresenta um desafio adicional, já que a apresentação de petições iniciais com cálculos precisos é fundamental para a validade do processo e para a adequada defesa dos direitos dos trabalhadores, conforme estabelecido no Artigo 319 da Lei nº 13.105, de 16 de março de 2015 [BRASIL, 2015] e no Art. 840 da Lei nº 5.452, de 01 de maio de 1943 [BRASIL, 1943].

Este trabalho aborda a possibilidade de simplificar e otimizar os cálculos trabalhistas, considerando a complexidade dos litígios, as obrigações legais de protocolar petições iniciais com pedidos líquidos e certos, e a crescente demanda por soluções tecnológicas eficazes no campo jurídico. Atualmente, os sistemas existentes que realizam cálculos trabalhistas são frequentemente complexos, onerosos e oferecem funcionalidades diversas. A automação proposta beneficia tanto advogados experientes quanto iniciantes, economizando tempo, reduzindo erros e eliminando a necessidade de contratar especialistas em cálculos, tornando os serviços jurídicos mais acessíveis. Visa-se criar um produto tecnológico capaz de simplificar a realização de cálculos trabalhistas, a gestão de informações dos clientes e a geração de relatórios.

1.1. Objetivos

A seguir são apresentados os objetivos do sistema proposto neste trabalho, sendo que o objetivo geral está relacionado com o resultado principal e os objetivos específicos ilustram funcionalidades e complementam o objetivo geral.

1.1.1. Objetivo Geral

Desenvolver um sistema web dedicado à realização de cálculos trabalhistas e à gestão de clientes, especialmente voltado para profissionais da advocacia.

1.1.2. Objetivos Específicos

Com vistas a alcançar o objetivo geral do trabalho, foram elencados os seguintes objetivos específicos:

- Realizar o levantamento e análise de requisitos do sistema;
- Criar protótipos da interface e funcionalidades do sistema; e
- Desenvolver o sistema com base nos requisitos estabelecidos.

A próxima seção detalha os conceitos e ferramentas utilizados para o desenvolvimento deste trabalho.

2. Fundamentação Teórica

Este capítulo aborda a base teórica necessária para compreender os elementos-chave deste trabalho. A seção 2.1 discute a relevância das leis trabalhistas no Brasil e a importância dos cálculos trabalhistas para garantir os direitos dos trabalhadores. A seção 2.2 explora a engenharia web, destacando seu papel no desenvolvimento de aplicativos web eficientes, e apresenta tecnologias client-side, como React e Tailwind CSS. Por fim, a seção 2.3 discute tecnologias server-side, incluindo o Node.js e Firebase, e sua aplicação no contexto deste projeto.

2.1. Leis e Cálculos Trabalhistas

As leis trabalhistas são fundamentais para regular as relações de trabalho no Brasil. Ao longo da história, várias leis foram criadas para proteger os direitos dos trabalhadores e garantir condições justas e dignas no emprego. Um marco importante foi a Consolidação das Leis do Trabalho (CLT), instituída em 1943 pelo governo de Getúlio Vargas. Ela reuniu várias normas trabalhistas existentes, estabelecendo direitos e responsabilidades tanto para empregadores quanto para empregados. A Constituição Federal de 1988 trouxe avanços significativos, conferiu status constitucional a diversas garantias, ampliando e incluindo direitos. Entre essas alterações, destacam-se a jornada de trabalho reduzida para oito horas diárias e 44 horas semanais, anteriormente 48 horas, o aviso-prévio proporcional, a extensão da licença-maternidade para 120 dias, a instituição da licença-paternidade e o reconhecimento do direito de greve. Em 2017, houve uma reforma trabalhista Lei nº 13.467/2017, que promoveu alterações significativas na legislação trabalhista brasileira, visando modernizá-la e estimular a criação de empregos [Santos, 2019].

Como foi visto, a CLT instituída em 1943 representa o alicerce das relações trabalhistas no Brasil. Ela estabelece direitos e obrigações para trabalhadores e empregadores, e dentre esses direitos, destacam-se os cálculos trabalhistas, que são essenciais para determinar os valores devidos ao trabalhador em diversas situações previstas pela legislação, a determinação dos valores que um empregador deve pagar a um empregado em caso de rescisão do contrato de trabalho. É crucial para as empresas cumprirem a legislação, evitando problemas legais e prejuízos futuros. Para realizar esses cálculos com precisão, é fundamental ter profissionais qualificados e conhecer bem os direitos dos trabalhadores [Galvão e Silva, 2020].

2.2. Engenharia Web e Tecnologias *Client-side*

Uma disciplina fundamental na criação e manutenção de aplicações e sistemas web é a engenharia web, como destacado no artigo "*Web Engineering: An introduction*" ela é essencial no cenário atual de rápida evolução das tecnologias web e da crescente demanda por aplicativos web eficientes [Ginige e Murugesan, 2001]. Com a internet integrada à vida cotidiana, a necessidade de desenvolver aplicativos web de alta qualidade e desempenho é indiscutível.

A engenharia web fornece ajuda no projeto, desenvolvimento e implementação de sistemas web, garantindo que eles atendam aos requisitos dos usuários e funcionem corretamente em diferentes plataformas e navegadores. Os processos de desenvolvimento web envolvem uma série de etapas organizadas e estruturadas que são essenciais para a criação de aplicativos web de sucesso. Este trabalho explora algumas dessas etapas, de acordo com Pastor [2004]:

1. Levantamento de Requisitos: nesta fase, identificam-se as necessidades e objetivos da aplicação web, garantindo uma compreensão clara dos requisitos do cliente e do público-alvo.
2. Design: aqui define-se a estrutura, arquitetura e *layout* da aplicação web. Isso inclui a criação de *wireframes*, protótipos e a definição da experiência do usuário.
3. Desenvolvimento: esta é a fase em que o código da aplicação web é criado, abrangendo desde a codificação das funcionalidades até a integração com banco de dados e outras tecnologias.
4. Testes: após o desenvolvimento, a aplicação web passa por rigorosos testes para identificar e corrigir erros, abrangendo aspectos de funcionalidade, usabilidade, desempenho e segurança.
5. Implantação: nesta etapa, a aplicação web é disponibilizada para os usuários finais após a configuração de servidores, publicação de código e garantia de funcionamento correto.
6. Manutenção: envolve a correção de *bugs*, atualizações de segurança, otimização de desempenho e adição de novas funcionalidades.

Em relação ao *client-side*, o termo se refere à parte de uma aplicação web que roda no navegador do usuário. Essas tecnologias são usadas para criar a interface do usuário e melhorar a experiência do usuário, garantindo que as páginas carreguem rapidamente e que as ações do usuário sejam processadas de forma eficaz. Elas trabalham em conjunto com o lado do servidor para fornecer uma experiência completa aos usuários da aplicação web [Maras et al., 2011].

React é uma biblioteca JavaScript baseada em componentes, criada com o objetivo de facilitar o desenvolvimento de interfaces para aplicações web. Uma das suas principais características é a capacidade de criar interfaces de usuário altamente reativas e interativas, permitindo aos desenvolvedores dividir o design da aplicação em componentes reutilizáveis. Isso torna a biblioteca React uma escolha popular para o desenvolvimento ágil e eficiente de interfaces de usuário dinâmicas e responsivas, interfaces que têm a capacidade de se adaptarem dinamicamente a diferentes tamanhos de tela) [Lou et al., 2013].

Tailwind CSS é um *framework* de manipulação de CSS altamente personalizável, que oferece uma abordagem única para o design de interfaces da web. Uma de suas

características distintivas é a capacidade de personalizar o estilo diretamente no HTML, por meio da aplicação de classes de utilitários. Em vez de criar folhas de estilo CSS separadas, o Tailwind permite que os desenvolvedores definam estilos de maneira ágil e modular, simplificando o processo de design e desenvolvimento [TAILWINDCSS, 2021].

2.3. Tecnologias Web *Server-side*

Server-side, ao contrário do termo *client-side* definido anteriormente, refere-se à parte de uma aplicação web que roda no servidor, não no navegador do usuário. Essa parte lida com a lógica de negócios, o armazenamento de dados e o processamento de informações. Ela se comunica com o *client-side* para fornecer os dados e as funcionalidades necessárias para a experiência do usuário [Iskandar et al., 2020].

O Node.js atua como um ambiente de execução JavaScript *server-side*. Ele pode ser definido como um *runtime* ou, em uma tradução livre, um interpretador de JavaScript assíncrono orientado a eventos, especialmente desenvolvido para lidar com aplicações de rede altamente escaláveis [Rubens, 2017]. Essa capacidade do Node.js de operar de forma assíncrona, em que fica em espera de eventos para executar ações, é de extrema importância para a eficiência e o desempenho do sistema como um todo.

O Firebase é uma plataforma desenvolvida pela Google que oferece uma variedade de serviços em nuvem para o desenvolvimento de aplicativos web e móveis. Essa plataforma inclui uma ampla gama de ferramentas, como autenticação, armazenamento em nuvem, banco de dados em tempo real e notificações por *push*, para citar algumas [FIREBASE, 2020]. O Firebase também é conhecido como um *Backend-as-a-service* (BaaS), que é definido por Batschinski [2016] como um serviço de computação em nuvem que serve como *middleware*, um serviço que atua como uma ponte entre diversas tecnologias. O mesmo fornece aos desenvolvedores uma forma para conectar suas aplicações *mobile* e web a serviços na nuvem a partir de interfaces de programação de aplicativos (APIs) e kits de desenvolvimento de *software* (SDKs).

A escolha do Firebase como parte integrante deste sistema se justifica pela abordagem moderna e eficiente para o desenvolvimento de aplicativos, aproveitando a infraestrutura escalável e os serviços prontos para uso que o Firebase oferece. No desenvolvimento do sistema fruto deste trabalho, usufruiu-se de dois produtos oferecidos pelo Firebase: o *Firebase Authentication* e o *Firebase Firestore*.

3. Metodologia

Para o desenvolvimento do produto tecnológico apresentado neste trabalho foi utilizado uma adaptação ao modelo sequencial linear, também conhecido como ciclo de vida clássico, que adota uma abordagem sequencial e sistemática no desenvolvimento de *software*, começando com a especificação dos requisitos do cliente e progredindo por fases de planejamento, modelagem, construção, disponibilização e suporte contínuo ao *software* finalizado [Pressman e Maxim, 2021].

O processo de desenvolvimento de *software*, iniciou-se com a fase de levantamento de requisitos, na qual coletou-se informações essenciais para compreender os objetivos e necessidades do cliente. Isso envolveu uma reunião com o *stakeholder*, onde foram discutidos objetivos, expectativas, funcionalidades e benefícios do projeto.

No planejamento, foram definidos os objetivos do projeto, que visam automatizar e simplificar os cálculos trabalhistas e o gerenciamento de clientes. O escopo inclui funcionalidades essenciais, como cálculos precisos e aspectos de design. As tecnologias escolhidas foram React e Firebase, com suporte do Adobe XD para prototipagem.

Na fase de modelagem, foram criadas representações visuais e abstratas do *software* em construção. Isso incluiu diagramas de casos de uso para definir a interação dos usuários com o sistema, um diagrama de classes para estruturar internamente o *software*, e a criação de interfaces de usuário no Adobe XD para validar o design antes da implementação. Essa fase estabeleceu as bases sólidas para o desenvolvimento futuro do sistema, garantindo conformidade com os requisitos e uma estrutura bem definida.

Na etapa de construção, seguindo a modelagem, utilizou-se as tecnologias React e Firebase para desenvolver a aplicação web. A biblioteca React foi utilizada para a criação da interface do usuário, proporcionando uma experiência dinâmica, enquanto o Firebase atuou como o *back-end*, gerenciando armazenamento de dados e autenticação. Durante a codificação, traduziu-se as funcionalidades dos modelos da fase de modelagem, como cálculos trabalhistas, gerenciamento de clientes e relatórios. Testes e revisões contínuos garantiram a conformidade com os requisitos e a usabilidade. A fase de codificação, por fim, resultou na transformação das ideias em um sistema funcional.

4. Desenvolvimento

Nesta seção, será abordado a modelagem e implementação do sistema proposto pelo presente trabalho: um sistema web denominado “Sistema Web de Cálculos Trabalhistas”. que tem como finalidade disponibilizar uma plataforma dedicada à realização de cálculos trabalhistas direcionada a escritórios de advocacia. Este trecho do trabalho detalha o processo de concepção e criação do produto, destacando alguns passos e considerações essenciais para tornar a solução funcional e eficaz.

Os usuários que se registrarem no aplicativo terão a capacidade de efetuar diversos tipos de cálculos trabalhistas e gerenciar seus clientes de maneira eficaz. Além disso, o sistema possibilitará a geração de relatórios em formato PDF, tanto dos cálculos elaborados, quanto dos clientes cadastrados. O objetivo é fornecer uma ferramenta de suporte aos profissionais de direito trabalhista em suas atividades diárias.

4.1. Modelagem do sistema

A Tabela 1 apresenta a lista de ferramentas e tecnologias que foram utilizadas no desenvolvimento proposto neste trabalho.

Ferramenta / Tecnologia	Versão	Finalidade
Node	18.17.0	Ambiente de execução de código
NPM	9.6.7	Gerenciador de pacotes
React	18.2.0	*Framework front-end*
Firebase	10.4.0	Banco de dados / Back-end
Tailwind CSS	3.3.3	*Framework* de CSS
Adobe XD	56.0.12.6	Ferramenta de design
Date-fns	2.30.0	Pacote para trabalhar com datas
React-pdf	7.4.0	Pacote para criar PDFs usando React
Visual Studio Code	1.83.1	Editor de código-fonte
Eslint	8.46.0	Ferramenta de análise estática
Prettier	3.0.1	Formatador de código

Tabela 1. Materiais utilizados no desenvolvimento deste trabalho

Na fase de modelagem do sistema, foram criados diagramas para uma compreensão mais profunda do projeto. A Figura 1 exibe o diagrama de caso de uso, destacando as funcionalidades do sistema executadas pelos atores, representados pelos advogados.

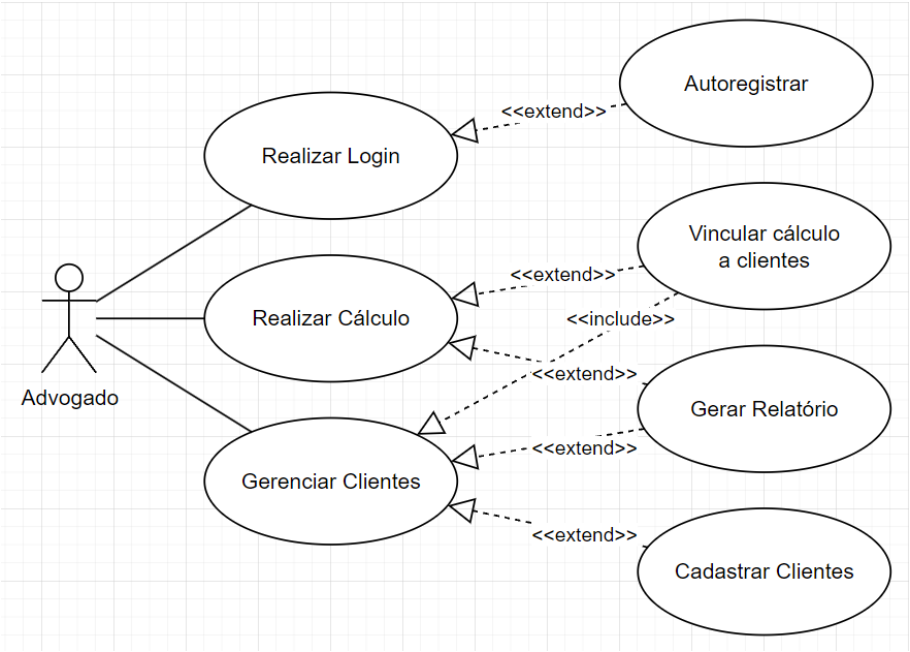

Figura 1. Diagrama de caso de uso

Além disso, a Figura 2 ilustra o diagrama de classes, que detalha os atributos e métodos das classes no sistema. Isso facilitou o processo de desenvolvimento e documentação do projeto.

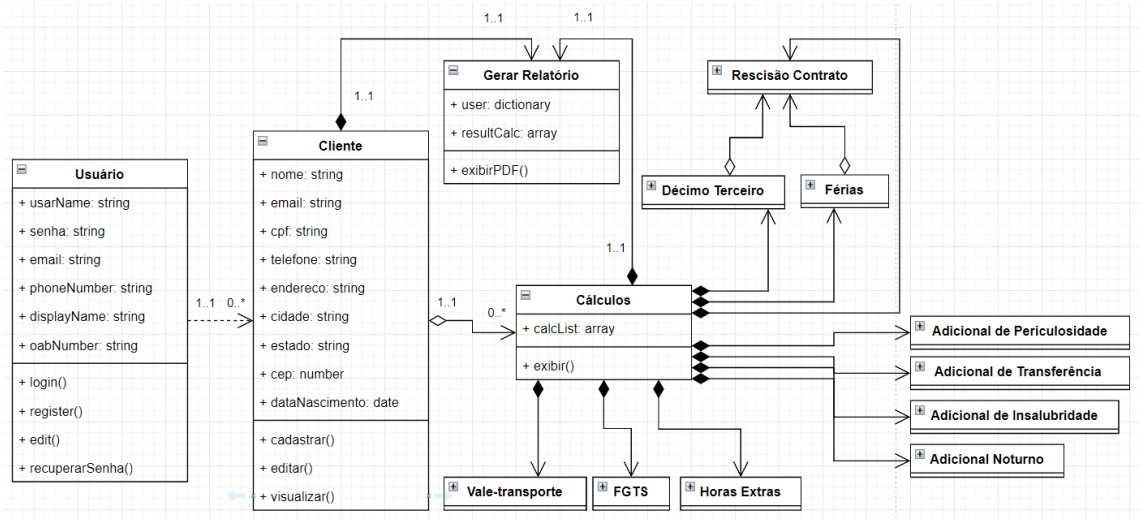

Figura 2. Diagrama de classes

Utilizando a ferramenta Adobe XD, foram projetadas as interfaces de usuário, algumas delas representadas na Figura 3, com atenção aos detalhes que ajudaram a visualizar como os usuários interagiriam com o sistema.

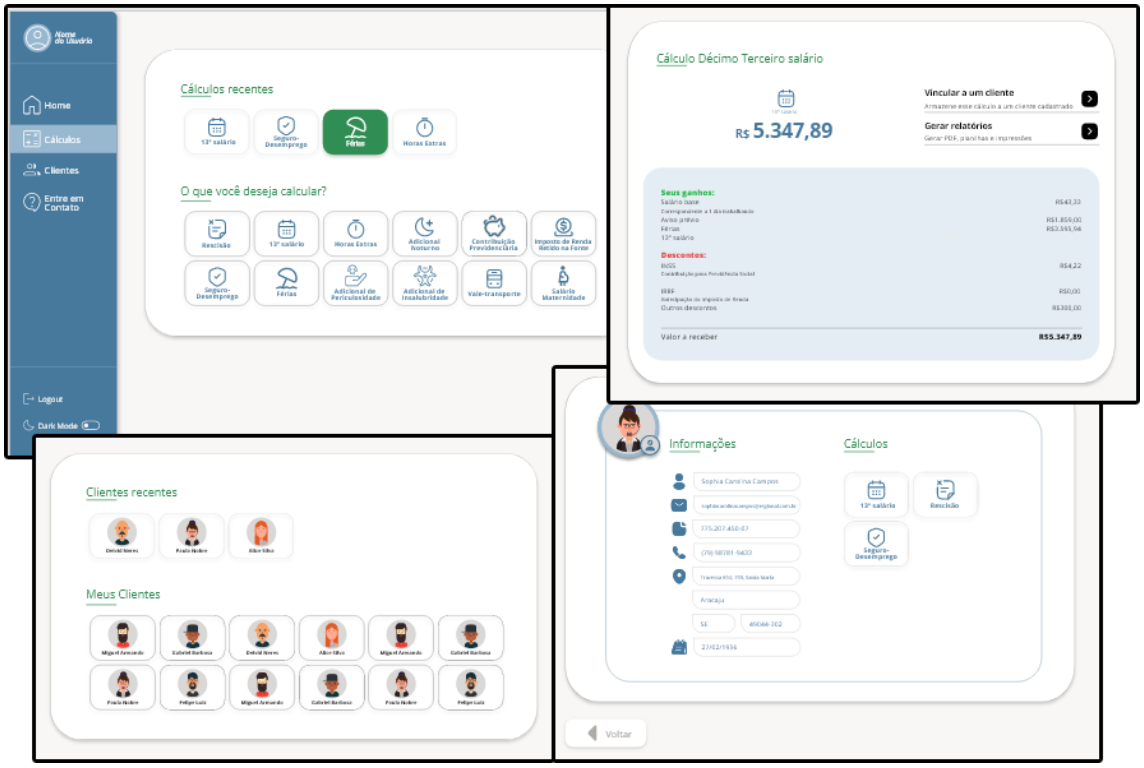

Figura 3. Protótipos do sistema

A prototipação das telas permitiu uma compreensão profunda dos componentes

a serem criados na biblioteca React, bem como das escolhas de estilos e cores para o sistema, além da disposição dos elementos em um design responsivo.

4.2. Implementação do Sistema

A seguir, serão apresentados alguns trechos de códigos do aplicativo, com o intuito de mostrar e exemplificar como foi realizada a implementação do sistema desenvolvido.

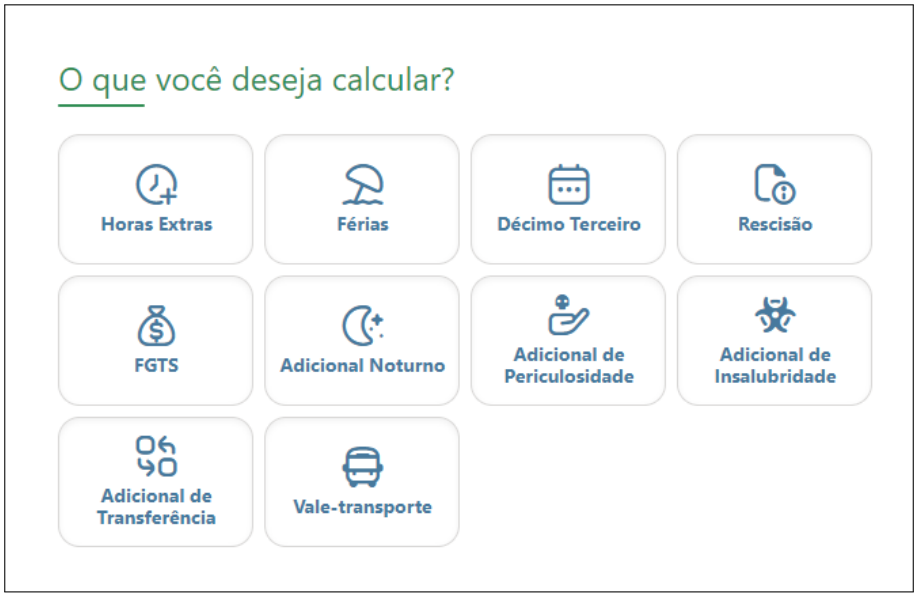

Figura 4. Tela de escolha dos Cálculos Trabalhistas

No que diz respeito aos cálculos trabalhistas, a Figura 4 ilustra as diversas categorias de cálculos trabalhistas incorporadas no sistema. A partir deste ponto, dois cálculos serão detalhadamente descritos, observando estritamente as normativas e legislações vigentes até a conclusão deste trabalho.

4.2.1. Férias Indenizatórias e Proporcionais

Conforme os Artigos 129 e 134 da CLT [BRASIL, 1977], todo funcionário tem direito a férias remuneradas após 12 meses de trabalho. Além disso, durante o período de férias, é adicionado $\frac{1}{3}$ (um terço) do valor do salário, chamado de terço constitucional. Esse adicional é estabelecido pelo artigo 7º, XVII, da Constituição Federal de 1988 [BRASIL, 1988]. Já as férias proporcionais, por sua vez, representam o período aquisitivo incompleto de férias, calculado na proporção de $\frac{1}{12}$ (um doze avos) por mês de serviço. Em casos de frações de mais de 14 dias trabalhados, considera-se um mês completo.

```
class BaseFeriasCalculator {
  constructor(inicioContrato, fimContrato, remuneracaoUltima, descontos, type = 'normal') {
    if (!inicioContrato || !fimContrato || remuneracaoUltima < 0 || descontos < 0) {
      throw new Error('Entradas inválidas.');
    }

    this.inicioContrato = this.createDateFromYYYYMMDD(inicioContrato);
    this.fimContrato = this.createDateFromYYYYMMDD(fimContrato);
    this.remuneracaoUltima = parseFloat(remuneracaoUltima);
    this.descontos = parseFloat(descontos);
    this.somaTotal = 0;
    this.type = type;
  }

  createDateFromYYYYMMDD(dateString) {
    const [year, month, day] = dateString.split('-').map(Number);
    return new Date(year, month - 1, day);
  }
```

Figura 5. Código da classe "BaseFeriasCalculator"

O código disposto na Figura 5 implementa o cálculo das férias de acordo com as regras mencionadas anteriormente. A classe "BaseFeriasCalculator" recebe como parâmetros para o construtor: (i) o início e fim do contrato; (ii) a última remuneração do trabalhador; (iii) os descontos; e (iv) um tipo de cálculo, que pode ter os valores "normal" ou "rescisao". A classe calcula o valor das férias devidas ao trabalhador com base no período e na remuneração, levando em consideração se o período de férias é menor ou igual a 12 meses.

```
calcularMesesTrabalhados(dataInicio, dataFim) {
  const diffMonths = differenceInCalendarMonths(dataFim, dataInicio) + 1;
  const daysInFirstMonth =
    differenceInDays(
      new Date(dataInicio.getFullYear(), dataInicio.getMonth() + 1, 0),
      dataInicio,
    ) + 1;
  const daysInLastMonth = dataFim.getDate();
  const adjustedDiffMonths =
    daysInFirstMonth >= 15
      ? daysInLastMonth >= 15
        ? diffMonths
        : diffMonths - 1
      : diffMonths - 1;
  return adjustedDiffMonths;
}
```

Figura 6. Método "calcularMesesTrabalhados"

O método "calcularMesesTrabalhados" (Figura 6) da mesma classe é responsável por calcular o número de meses de trabalho entre duas datas, levando em consideração a legislação que estabelece regras para o cálculo de férias proporcionais. O método recebe duas datas como entrada: "dataInicio" e "dataFim", que representam o início e o fim de

um período de trabalho. Esse método é essencial para o cálculo das férias proporcionais, onde o período de trabalho é considerado na proporção de $\frac{1}{12}$ (um doze avos) por mês de serviço, e frações de mais de 15 dias trabalhados são consideradas como meses completos. Dessa forma, o cálculo segue as regras estabelecidas pela legislação trabalhista.

```
calcularFeriasProporcionais(periodoInicial, dataFimPeriodo) {
  const mesesTrabalhados = this.calcularMesesTrabalhados(periodoInicial, dataFimPeriodo);

  if (mesesTrabalhados > 0) {
    const avos = this.remuneracaoUltima / 12;
    const ferias = avos * mesesTrabalhados;
    const tercoConstitucional = ferias / 3;
    const feriasProporcionais = ferias + tercoConstitucional;
    this.somaTotal += feriasProporcionais;

    return {
      'Última remuneração': this.remuneracaoUltima,
      'Valor das ferias': ferias,
      'Terço Constitucional': tercoConstitucional,
      'Férias Proporcionais': feriasProporcionais,
    };
  } else {
    return null;
  }
}
```

Figura 7. Cálculo de férias proporcionais

O cálculo do valor devido em relação às férias é executado pelo método "calcular()" da mesma classe. Ele itera sobre os períodos de férias, calculando as férias proporcionais ou indenizatórias, dependendo da situação. Para períodos menores que 12 meses, o cálculo é de férias proporcionais (Figura 7), enquanto para períodos iguais ou maiores que 12 meses, as férias são consideradas indenizatórias (Figura 8).

O método "calcularFeriasProporcionais" desta classe tem a finalidade de calcular o valor das férias proporcionais a que um trabalhador tem direito, com base no período de trabalho fornecido pelo usuário do sistema. Primeiramente, chama-se o método "calcularMesesTrabalhados" para determinar o número de meses de trabalho, levando em consideração as regras legais, como o arredondamento para meses completos quando frações de mais de 14 dias são trabalhadas. Em seguida, o método calcula o valor das férias, que corresponde a $\frac{1}{12}$ (um doze avos) da remuneração do último mês de trabalho multiplicada pelo número de meses trabalhados. Além disso, o método ainda calcula o terço constitucional, que equivale a $\frac{1}{3}$ (um terço) do valor das férias. Por fim, somam-se esses valores para obter o montante das férias proporcionais e o armazena na variável "somaTotal". O método retorna um objeto contendo informações sobre a última remuneração, o valor das férias, o terço constitucional e o total das férias proporcionais, seguindo as diretrizes estabelecidas pela legislação trabalhista para o cálculo das férias proporcionais.

```
calcularFeriasIndenizatorias(dataInicioPeriodo, dataFimPeriodo) {
  const diffMonths = differenceInCalendarMonths(dataFimPeriodo, dataInicioPeriodo);

  if (diffMonths >= 12) {
    const avos = this.remuneracaoUltima / 12;
    const mesesTrabalhados = this.calcularMesesTrabalhados(dataInicioPeriodo, dataFimPeriodo);
    const ferias = avos * mesesTrabalhados;
    const tercoConstitucional = ferias / 3;
    let feriasIndenizatorias = ferias + tercoConstitucional;

    const diffDays = Math.abs(differenceInDays(dataFimPeriodo, this.fimContrato));
    if (diffDays >= 365) {
      feriasIndenizatorias *= 2;
    }

    this.somaTotal += feriasIndenizatorias;

    const resultados = {
      'Última Remuneração': this.remuneracaoUltima,
      Férias: ferias,
      'Terço Constitucional': tercoConstitucional,
      'Férias Indenizatorias': feriasIndenizatorias,
    };
    return resultados;
  } else {
    return null;
  }
}
```

Figura 8. Cálculo de férias indenizatórias

O método “calcularFeriasIndenizatórias” da classe, é aplicado quando o trabalhador possui pelo menos 12 meses de serviço ininterruptos. Ele inicia calculando o número de meses trabalhados usando o método “calcularMesesTrabalhados” e, em seguida, calcula o valor das férias, que consiste em $\frac{1}{12}$ (um doze avos) da última remuneração multiplicada pelos meses trabalhados. Além disso, determina o terço constitucional, equivalente a $\frac{1}{3}$ (um terço) do valor das férias. A peculiaridade das férias indenizatórias ocorre quando o período de férias se estende por um ano em relação ao término do contrato, caso em que o valor das férias indenizatórias dobra. Após todos os cálculos, o método soma o valor das férias indenizatórias ao total armazenado na variável “somaTotal”. Os resultados são retornados em um objeto que inclui detalhes sobre a última remuneração, o valor das férias, o terço constitucional e o total das férias indenizatórias, atendendo às exigências legais para o cálculo das férias indenizatórias na legislação trabalhista.

4.2.2. Adicional Noturno

O adicional noturno é um direito garantido a todos os trabalhadores que atuam no período noturno, conforme estabelecido pela CLT. Este benefício é obrigatório por lei e implica um acréscimo de 20% (vinte por cento) para empregados urbanos ou 25% (vinte e cinco por cento) para empregados rurais na remuneração em relação ao período diurno, seja para profissionais contratados para trabalhar no turno noturno ou para aqueles que realizam

horas extras durante a noite de acordo com o Artigo 7º, Inciso IX, da Constituição Federal (CF) [BRASIL, 1988] e Artigo 73 da CLT [BRASIL, 1943].

O código apresentado na Figura 9 implementa o cálculo do adicional noturno de acordo com as regras mencionadas no texto. A classe "NoturnoCalculator" recebe como entrada o número de horas trabalhadas semanalmente, a remuneração, o tipo de empregado (que pode ser '20' ou '25' para representar o adicional de 20% ou 25%, respectivamente), e o número de horas trabalhadas no período noturno.

```
class NoturnoCalculator {
  constructor(horasTrabalhadasSem, remuneracao, tipoEmpregado, horasTrabalhadasNortuna) {
    if (horasTrabalhadasSem < 0 || remuneracao < 0 || !['20', '25'].includes(tipoEmpregado)) {
      throw new Error('Valores inválidos.');
    }

    this.horasTrabalhadasSem = parseFloat(horasTrabalhadasSem);
    this.remuneracao = parseFloat(remuneracao);
    this.ValorHoraTrabalho = calcularValorHoraTrabalho(this.horasTrabalhadasSem, this.remuneracao);
    this.tipoEmpregado = tipoEmpregado;
    this.horasTrabalhadasNortuna = parseFloat(horasTrabalhadasNortuna);
  }
```

Figura 9. Cálculo de Adicional Noturno

A função "calcularValorHoraTrabalho" (Figura 10) desempenha um papel fundamental na classe "NoturnoCalculator". Essa função é importada para a referida classe pois, dentro do sistema, ela se mostrou comum em outros cálculos que são realizados. A função é responsável por calcular o valor da hora de trabalho com base na carga horária semanal e na remuneração mensal do empregado. O cálculo é feito dividindo a remuneração mensal pelas horas mensais trabalhadas, onde as horas mensais são calculadas multiplicando as horas semanais de trabalho por 4,5, considerando que um mês tem em média 4 semanas e meia.

```
export function calcularValorHoraTrabalho(horasSemanais, remuneracao) {
  const horasMensais = horasSemanais * 4.5;
  const valorHora = remuneracao / horasMensais;

  return valorHora;
}
```

Figura 10. Função "calcularValorHoraTrabalho"

O cálculo do valor do adicional noturno é executado pelo método "calcularNoturno" (Figura 11) da classe. Ele determina o percentual de adicional com base no tipo de empregado e calcula o valor do adicional noturno. O resultado final inclui a última remuneração, o tipo de empregado, as horas noturnas trabalhadas, o valor do adicional noturno e o valor total a receber.

```
calcularNoturno() {
  const tipoEmpregadoText = {
    20: 'Empregado Urbano (20%)',
    25: 'Empregado Rural (25%)',
  };

  const percentuais = {
    20: 0.2,
    25: 0.25,
  };

  const percentual = percentuais[this.tipoEmpregado];

  const adicional = this.ValorHoraTrabalho * percentual;
  const valorHoraNoturna = this.ValorHoraTrabalho + adicional;
  const valorAdicionalNoturno = valorHoraNoturna * this.horasTrabalhadasNortuna;

  const resultado = {
    'Última Remuneração': this.remuneracao,
    'Tipo de Empregado': tipoEmpregadoText[this.tipoEmpregado],
    'Horas Noturnas Trabalhadas': `${this.horasTrabalhadasNortuna} Horas`,
    'Valor do Adicional Noturno': valorAdicionalNoturno,
  };

  const valorAReceber = {
    'Valor a Receber': valorAdicionalNoturno,
  };

  const resultados = [resultado, valorAReceber];

  return resultados;
}
```

Figura 11. Método "calcularNoturno()"

O código implementa as regras da legislação de forma a calcular o adicional noturno com base no número de horas trabalhadas no período noturno, a remuneração e o tipo de empregado, proporcionando um resultado que segue as especificações legais.

4.2.3. Clientes

O sistema possui um módulo dedicado ao gerenciamento de clientes, permitindo que os usuários do aplicativo mantenham um registro organizado de informações sobre seus clientes. Esse módulo é fundamental para profissionais que desejam manter um relacionamento eficaz com seus clientes e gerenciar suas informações de forma centralizada. Vamos explorar as funcionalidades e componentes relacionados ao gerenciamento de clientes que o sistema de cálculos trabalhistas oferece.

```
export const useClientes = () => {
  return useContext(ClientesContext);
};

export const ClientesProvider = ({ children }) => {
  const [clientes, setClientes] = useState([]);
  const [modoEdicao, setModoEdicao] = useState(false);
  const [clienteEditando, setClienteEditando] = useState(null);
```

Figura 12. "ClientesContext"

O contexto de clientes é uma parte central do sistema que fornece funcionalidades essenciais para a manipulação de dados dos clientes. Na biblioteca React, o "Contexto" é uma funcionalidade que permite compartilhar dados entre componentes sem a necessidade de passar esses dados explicitamente através das propriedades. Ele é especialmente útil quando existe uma árvore de componentes profundamente aninhada e vários componentes precisam acessar os mesmos dados. O contexto de clientes é definido no arquivo "ClientesContext" (Figura 12), e exporta as seguintes funcionalidades:

1. Acesso aos Dados dos Clientes: os dados dos clientes podem ser obtidos a partir do contexto, permitindo que outros componentes do aplicativo acessem e modifiquem essas informações.
2. Adicionar Cliente: o contexto oferece uma função "addCliente" para adicionar novos clientes ao sistema, armazenando suas informações no banco de dados.
3. Atualizar Cliente: os clientes existentes podem ser atualizados por meio da função "updateCliente", que permite que os dados do cliente sejam modificados e, em seguida, atualizados no banco de dados.
4. Excluir Cliente: a função "deleteCliente" permite a remoção de um cliente específico da base de dados.
5. Modo de Edição: o contexto gerencia um modo de edição que permite aos usuários editar os detalhes de um cliente. Isso é controlado por "modoEdicao" e "setModoEdicao".
6. Cliente Editando: o cliente que está atualmente em processo de edição é mantido pelo contexto em "clienteEditando" e "setClienteEditando".

Nesse contexto ainda, o Firebase é usado para realizar operações de leitura e escrita de dados tempo real. A função do Firebase é gerenciar a conexão com o Firestore (Figura 13) e executar operações como adicionar, atualizar e excluir clientes (dados) relacionados a um usuário específico, bem como manter os dados sincronizados em tempo real com o componente React. Isso permite que os dados dos clientes sejam buscados, atualizados e refletidos dinamicamente na interface do usuário em resposta a alterações no banco de dados, proporcionando uma experiência em tempo real para os usuários do aplicativo. Além disso, ele também lida com a autenticação do usuário para garantir que apenas usuários autenticados tenham acesso aos dados de seus clientes.

```
useEffect(() => {
  if (user) {
    const loadClientes = async (userId) => {
      try {
        const db = getFirestore(app);
        const userDocRef = doc(db, 'users', userId);
        const userClientesCollection = collection(userDocRef, 'clientes');

        const unsubscribe = onSnapshot(userClientesCollection, (querySnapshot) => {
          const clientesArray = [];
          querySnapshot.forEach((doc) => {
            clientesArray.push({ id: doc.id, ...doc.data() });
          });
          setClientes(clientesArray);
        });
        return () => unsubscribe();
      } catch (error) {
        console.error('Erro ao carregar clientes:', error);
      }
    };

    loadClientes(user.uid);
  }
}, [user]);
```

Figura 13. Conexão com o Firestore

O trecho de código disposto na Figura 13 representa uma função do react "useEffect" em um componente React, o qual é executado sempre que a variável *user* muda. A principal função desse código é carregar dados de clientes relacionados a um usuário específico. Primeiro, ele verifica se o objeto "user" está definido, o que implica que um usuário está autenticado. Em seguida, ele define uma função "loadClientes" que será executada para buscar os dados dos clientes desse usuário.

Ainda na função "useEffect", utiliza-se o Firebase Firestore para configurar uma referência ao documento do usuário e à coleção de clientes desse usuário. Em seguida, ele registra um ouvinte "onSnapshot" que observa as alterações nessa coleção em tempo real. Sempre que ocorrem alterações, o código transforma os dados de cada documento em um formato desejado (incluindo o ID do documento) e os armazena no vetor "clientesArray". Esse vetor de clientes é então utilizado para atualizar o estado do componente React, especificamente a variável de estado "clientes" que, quando atualizada, fará com que todo componente que utiliza dados dos clientes seja novamente renderizado exibindo os dados mais recentes na interface do usuário.

```
function CalculationResult({ title, results, icon }) {
  const { clientes } = useClientes();
  const [mostrarCaixaSelecao, setMostrarCaixaSelecao] = useState(false);
  const [clienteSelecionado, setClienteSelecionado] = useState(null);
  const [mostrarAviso, setMostrarAviso] = useState(false);
  const [userData, setUserData] = useState({});

  const { user } = useAuth();
```

Figura 14. Componente "CalculationResult"

No sistema, é possível vincular cálculos realizados a perfis de clientes previamente cadastrados. Isto é feito pelo componente “CalculationResult” (Figura 14), que exibe os resultados de cálculos específicos, permitindo ao usuário vinculá-los a um cliente no sistema e armazená-los no Firestore Database. Os resultados dos cálculos são apresentados de maneira legível na interface do usuário, incluindo valores formatados em reais brasileiros, e o usuário pode optar por vinculá-los a um cliente específico.

Para o vínculo aos perfis do cliente, o componente oferece um botão que permite ao usuário vincular os resultados a um cliente já cadastrado no sistema. Ao clicar nesse botão, uma caixa de seleção de clientes é exibida, na qual o usuário pode escolher um cliente específico a quem os resultados serão associados. Isso é útil para manter um registro organizado dos cálculos para cada cliente, facilitando o acesso posterior.

Para o armazenamento no Firestore Database, Após selecionar um cliente e clicar no botão de vinculação, os resultados dos cálculos são armazenados no banco de dados (Figura 15). Isso é realizado utilizando a API do Firebase Firestore. Os dados do usuário autenticado, como o ID do usuário, são usados para acessar a coleção de clientes no banco de dados. Os resultados são armazenados no documento do cliente correspondente, sob a chave “ResultadosCalculos”. O Firebase Firestore permite que os dados sejam atualizados em tempo real, garantindo que os cálculos mais recentes estejam sempre disponíveis.

```
useEffect(() => {
  if (results.length > 0 && clienteSelecionado !== null) {
    const saveResults = async () => {
      const db = getFirestore();
      const userDocRef = doc(db, 'users', user.uid);
      const userClientesCollection = collection(userDocRef, 'clientes');
      const clienteDoc = doc(userClientesCollection, clienteSelecionado);

      try {
        const clienteSnapshot = await getDoc(clienteDoc);
        const clienteData = clienteSnapshot.data();

        const novosResultadosCalculos = {
          ...clienteData.ResultadosCalculos,
          [title]: results,
        };

        await updateDoc(clienteDoc, {
          ResultadosCalculos: novosResultadosCalculos,
        });

        setMostrarAviso(true);
        console.log('Resultados salvos com sucesso no Firebase.');
      } catch (error) {
        console.error('Erro ao salvar resultados:', error);
      }
    };
    saveResults();
  }
}, [clienteSelecionado, results, title, user]);
```

Figura 15. Vincular cálculos a clientes

O trecho de código mostrado na Figura 15 utiliza um “useEffect” para monitorar mudanças em variáveis específicas e realizar ações quando essas mudanças ocorrem. Primeiro, ele verifica se existem resultados de cálculos disponíveis (armazenados no vetor “results”) e se um cliente foi selecionado (“clienteSelecionado” não é nulo). Se ambas as condições forem atendidas, a função “saveResults” é acionada. Dentro desta função, o código realiza as seguintes etapas:

1. Obtém uma referência ao banco de dados Firestore usando o Firebase SDK;
2. Define referências aos documentos necessários no banco de dados: o documento do usuário atual “userDocRef”, a coleção de clientes desse usuário “userClientesCollection”, e o documento específico do cliente que está selecionado “clienteDoc”;
3. Recupera os dados do cliente selecionado (como os resultados de cálculos existentes) por meio de uma consulta ao banco de dados;
4. Atualiza os resultados de cálculos armazenados no Firebase, adicionando os novos resultados sob a chave “title” no objeto “ResultadosCalculos”. Essa atualização é realizada usando “updateDoc”;
5. Define um estado “setMostrarAviso(true)” para exibir uma notificação de sucesso na interface do usuário; e
6. Registra no console uma mensagem indicando que os resultados foram salvos com sucesso ou, caso ocorra um erro, registra uma mensagem de erro.

O “useEffect” observa as variáveis “clienteSelecionado”, “results”, “title” e “user”, para que, sempre que houver uma alteração em qualquer uma delas, o código seja executado, garantindo que os resultados dos cálculos sejam armazenados no Firestore Database quando as condições apropriadas forem atendidas.

Além disso, o componente “CalculationResult” fornece a capacidade de gerar relatórios em PDF por meio do *component* “ReportPDF”. O fornecimento de relatórios em PDF é realizado por meio do código, representado na Figura 16, utilizando a biblioteca React-PDF.

```
const generatePdf = () => {
  return (
    <Document>
      <Page size='A4' style={styles.page}>
        <View>
          <Text style={styles.titulo}>{title}</Text>
        </View>

        <View style={styles.section}>
          <View style={styles.subtitle}>
            <Text>Informações do Usuário:</Text>
          </View>
          {renderNestedDataResults(userData, stylesResults)}
        </View>
```

Figura 16. Trecho do código que gera relatórios

O componente “ReportPDF” recebe várias informações, incluindo um título, resultados de cálculos, informações do usuário e resultados de cálculos específicos. O PDF é gerado como um documento composto por uma ou várias páginas, onde as informações são organizadas e formatadas de forma legível para o usuário. A página do PDF inclui o título do relatório, informações do usuário e detalhes sobre os resultados dos cálculos. Ao final, o PDF é renderizado na interface do usuário por meio do componente “PDFViewer” (Figura 17), permitindo ao usuário visualizar e baixar o relatório gerado em PDF.

```
return (
  <div>
    <PDFViewer width='100%;' height='100%;' margin='0'>
      {generatePdf()}
    </PDFViewer>
  </div>
);
```

Figura 17. Código responsável por renderizar o arquivo PDF

Essa funcionalidade é valiosa para a apresentação de resultados de cálculos de forma profissional e prontamente acessível no sistema.

4.2.4. Autenticação

A autenticação desempenha um papel fundamental em muitos sistemas, e na solução atual, é essencial para garantir a segurança e a identificação dos usuários. Nesta parte, aborda-se como a autenticação foi implementada no sistema utilizando o Firebase Authentication. O uso do Firebase Authentication simplifica esse processo e fornece uma camada de segurança robusta para proteger as contas dos usuários. Ao criar páginas de login, registro e redefinição de senha, o sistema oferece uma experiência completa de autenticação para os usuários, permitindo que eles acessem o sistema com facilidade e segurança.

Na página de login, fica-se responsável por permitir que os usuários façam login em suas contas. A autenticação é realizada chamando o comando “signInWithEmailAndPassword” (Figura 18), e em caso de sucesso, redireciona o usuário para a página principal do sistema. Em caso de erro, uma mensagem de erro é exibida.

```
useEffect(() => {
  if (auth.isAuthenticated) {
    history('/calculos');
  }
}, [auth.isAuthenticated, history]);

const handleLogin = async (e) => {
  e.preventDefault();

  try {
    await firebase.auth().signInWithEmailAndPassword(email, password);
    history('/calculos');
  } catch (error) {
    setErrorMessage('Email ou senha incorretos. Por favor, verifique suas credenciais.');
  }
};
```

Figura 18. Código responsável por verificar a autenticação do usuário

A página de registro permite que novos usuários se registrem no sistema. Após preencher o formulário de registro e passar pelas verificações necessárias, a função "createUserWithEmailAndPassword" é chamada para criar uma nova conta de usuário. O nome de exibição do usuário é configurado com "user.updateProfile" e, em seguida, um documento de cliente é criado no Firestore para associar informações adicionais ao usuário.

A página "esqueci a senha", por sua vez, lida com a redefinição de senhas. O usuário insere seu endereço de e-mail e, após a validação, uma mensagem de redefinição de senha é enviada com "sendPasswordResetEmail". A implementação também inclui verificações de validação de dados, como garantir que as senhas coincidam e que os campos estejam dentro dos limites permitidos (por exemplo, comprimento máximo de caracteres para o número de telefone).

Para melhorar a experiência do usuário, o sistema redireciona automaticamente os usuários autenticados para a página principal após o login bem-sucedido, utilizando o React Router. O arquivo "PrivateRoute" (Figura 19) contém um componente chamado "PrivateRoute", o qual é utilizado para proteger as rotas no sistema, permitindo que apenas usuários autenticados acessem as páginas protegidas. Caso o usuário não esteja autenticado, ele é redirecionado para a página de login.

```
const PrivateRoute = () => {
  const { isAuthenticated } = useContext(AuthContext);
  return isAuthenticated ? <Outlet /> : <Navigate to='/login' />;
};
```

Figura 19. Código "PrivateRoute" para verificar se um usuário está autenticado

A função deste código é criar um componente chamado "PrivateRoute", que age como um guardião de rotas em um sistema React usando React Router. Ele usa a função chamada de "useContext" para acessar informações de autenticação do contexto e verificar se um usuário está autenticado ou não. Se o usuário estiver autenticado (ou seja, "isAuthenticated" for verdadeiro), o componente "Outlet" é renderizado, permitindo que o usuário acesse as rotas protegidas. Caso contrário, o usuário é redirecionado para a página de login com "<Navigate to='/login' />". Isso garante que apenas usuários autenticados tenham acesso às páginas protegidas.

Esta seção apresentou a modelagem e implementação do sistema web de cálculos trabalhistas. A próxima seção de Conclusão destaca a realização do trabalho em resposta a uma demanda identificada, os propósitos do sistema, o atendimento aos requisitos, e sugere possíveis melhorias e expansões futuras.

5. Conclusão

O presente trabalho foi realizado em resposta a uma demanda identificada pelo desenvolvedor desta solução, originada por sua proximidade com uma advogada especializada em questões trabalhistas. Essa proximidade possibilitou uma comunicação direta com o *stakeholder* e, por conseguinte, direcionou o desenvolvimento do sistema de acordo com as necessidades por ela expressas.

O sistema criado tem o propósito de facilitar e automatizar os cálculos trabalhistas realizados por advogados na área trabalhista, e auxiliar na gestão de clientes deste tipo de demanda. A automação não apenas melhora a eficiência e a precisão dos cálculos, economizando tempo e recursos, mas também torna os serviços jurídicos mais acessíveis, tendo satisfeito todos os requisitos levantados.

Como trabalho futuro, pode-se expandir o sistema para abranger outros tipos de cálculos trabalhistas, incluir animações em componentes visuais do sistema para melhor experiência do usuário, promover o compartilhamento de informações e aprimorar os detalhamentos nos relatórios gerados, ainda a segurança do sistema ser constantemente reforçada. Além disso, pode-se considerar o desenvolvimento de um aplicativo móvel dedicado para atender às necessidades dos usuários que não possuem acesso a um computador de forma mais conveniente.

Referências

Batschinski, G. (2016). Backend as a service: Prós e contras. https://www.infoq.com/br/news/2016/07/backend-pros-e-contras/ [Acesso em 30/10/2023].

BRASIL (1943). Lei nº 5.452, de 1 de maio de 1943. aprova a consolidação das leis do trabalho. https://www.planalto.gov.br/ccivil_03/Decreto-Lei/Del5452.htm#art478%C2%A72 [Acesso em 30/10/2023].

BRASIL (1977). Lei nº 1.535, de 15 de abril de 1977. consolidação das leis do trabalho, relativo a férias, e dá outras providências. https://www.planalto.gov.br/ccivil_03/decreto-lei/del1535.htm [Acesso em 30/10/2023].

BRASIL (1988). Constituição da república federativa do brasil de 1988. https://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm [Acesso em 30/10/2023].

BRASIL (2015). Lei nº 13.105, de 16 de março de 2015. código de processo civil. https://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2015/Lei/L13105.htm#art212%C2%A72 [Acesso em 30/10/2023].

Carvalho, I. M. d. (2020). O empreendedorismo e o mundo jurídico-advocacia 4.0. *Trabalho de Conclusão de Curso*.

Drucker, P. F. (1980). O gerente eficaz. 9ª edição.

Ferreira, F. (2020). Inteligência artificial atua como juiz, muda estratégia de advogado e 'promove' estagiário. https://www.dianatech.com.br/inteligencia-artificial-atua-como-juiz-muda-estrategia-de-advogado-e-promove-estagiario/ [Acesso em 30/10/2023].

FIREBASE (2020). Documentação. https://firebase.google.com/docs/projects/learn-more?authuser=0 [Acesso em 30/10/2023].

Galvão and Silva (2020). Você sabe o que é cálculo trabalhista? entenda! https://www.jusbrasil.com.br/artigos/voce-sabe-o-que-e-calculo-trabalhista-entenda/843955428 [Acesso em 30/10/2023].

Ginige, A. and Murugesan, S. (2001). Web engineering.

Iskandar, T. F., Lubis, M., Kusumasari, T. F., and Lubis, A. R. (2020). Comparison between client-side and server-side rendering in the web development. *IOP Conference Series: Materials Science and Engineering*, 801(1):012136.

Lou, Y., Zhang, Y., and Yuan, Z. (2013). Website information extraction based on dom-model. In *Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation (ISCCCA 2013)*, pages 792–795. Atlantis Press.

Maras, J., Carlson, J., and Crnkovic, I. (2011). Client-side web application slicing.

Pastor, O. (2004). Fitting the pieces of the web engineering puzzle. *ACS Publications*.

Pressman, R. S. and Maxim, B. R. (2021). *Engenharia de software-9*. McGraw Hill Brasil.

Rubens, J. (2017). *Primeiros passos com Node.js*. Casa do Código.

Santos, L. d. C. (2019). Advocacia trabalhista no brasil: histórico da profissão no país e sua importância na construção de um estado democrático de direito.

Serio, L. and Vasconcellos, M. (2009). *Estratégia e Competitividade Empresarial: Inovação e Criação de Valor*. Editora Saraiva São Paulo.

TAILWINDCSS (2021). A utility-first css framework for rapidly building custom designs. https://v1.tailwindcss.com/ [Acesso em 30/10/2023].

Tribunal Superior do Trabalho (2022). https://www.tst.jus.br/web/estatistica/jt/relatorio-geral#:~:text=Ao%20iniciar%20o%20ano%20de,.497%20deles%2C%20Casos%20Novos. [Acesso em 30/10/2023].