Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus “José Santilli Sobrinho”

JOSÉ ROBERTO VIEL BATISTA

PROGRAMAÇÃO FUNCIONAL: PARADIGMAS E APLICAÇÃO NO AGRONEGÓCIO

Assis/SP
2024

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

JOSÉ ROBERTO VIEL BATISTA

PROGRAMAÇÃO FUNCIONAL: PARADIGMAS E APLICAÇÃO NO AGRONEGÓCIO

Projeto de pesquisa apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Municipal de Ensino Superior de Assis – IMESA e à Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando(a): José Roberto Viel Batista
Orientador(a): Me. Guilherme de Cleva Farto

Assis/SP
2024

Batista, José Roberto Viel

B333p Programação funcional: paradigmas e aplicação no agronegócio / José Roberto Viel Batista. -- Assis, 2024.

40p.

Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) -- Fundação Educacional do Município de Assis (FEMA), Instituto Municipal de Ensino Superior de Assis (IMESA), 2024.

Orientador: Prof. Me. Guilherme de Cleva Farto

1. Linguagens de programação. 2. Processamento de dados. I Farto, Guilherme de Cleva. II Título.

CDD 004

Elaborada por Anna Carolina Antunes de Moraes – Bibliotecária – CRB-8/10982

DEDICATÓRIA

Dedico este trabalho aos meus familiares e em especial à minha mãe, que desde o início estiveram ao meu lado, apoiando e incentivando de todas as formas. Dedico também aos meus amigos e professores que estiveram sempre ao meu lado colaborando para o meu crescimento profissional e pessoal.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por me conceder saúde, sabedoria e força para enfrentar os desafios ao longo desta jornada. À minha mãe, Eliana Cristina Viel, por seu amor incondicional, paciência e apoio em todos os momentos. Sua presença foi uma fonte constante de inspiração e motivação para que eu seguisse adiante, apoiando em todos os quesitos, de forma financeira ou moral. À minha família, que sempre esteve ao meu lado, me incentivando e acreditando em meu potencial, e aos meus amigos, que fizeram parte deste processo, onde juntos superamos grandes desafios.

Ainda agradeço aos meus professores, que contribuíram de forma significativa para o meu desenvolvimento acadêmico e pessoal. Em especial, ao meu orientador e amigo, Guilherme de Cleva Farto, por sua paciência, dedicação e orientações fundamentais, que me guiaram para que esse trabalho fosse possível.

RESUMO

Atualmente, empresas e instituições buscam a praticidade, simplicidade e facilidade na tomada de decisões, para que sejam feitas ações de forma assertiva e com mais agilidade. Visto essa necessidade, a programação funcional vem para auxiliar em projetos, que manipulam grandes massas de dados, disponibilizando praticidade e rapidez. O objetivo deste trabalho é realizar um estudo exploratório relacionado ao paradigma funcional e suas vantagens, e na prática, implementar um *MVP* de um aplicativo de gestão agrícola, mostrando o que pode ser implementado e os recursos existentes para atender a esse cenário.

Palavras-chave: Paradigma funcional, Clojure; agronegócio;

ABSTRACT

Nowadays, companies and institutions seek practicality, simplicity, and ease in decision-making, enabling actions to be taken assertively and more quickly. Given this need, functional programming is emerging as a solution for projects that handle large amounts of data, providing both practicality and speed. The objective of this work is to conduct an exploratory study on the functional paradigm and its advantages, and, in practice, to implement an MVP of an agricultural management application, demonstrating what can be implemented and the available resources to meet this scenario.

Keywords: Paradigma funcional, Clojure, agribusiness;

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

A programação funcional ganhou foco no mundo do desenvolvimento há pouco tempo, sendo explorada cada vez mais com o objetivo de extrair um grande potencial existente neste modelo de desenvolvimento. O diferencial deste paradigma de desenvolvimento é a forma de criar soluções e manipular grandes volumes de dados e, assim, abriu-se uma lacuna ainda maior para a implementação e seguimento do paradigma funcional (HU, HUGHES, WANG, 2015).

O paradigma funcional é baseado no processo de construir softwares por meio da composição de funções puras, evitando o compartilhamento de estados, dados mutáveis e efeitos colaterais, assim como é explicitamente declarativa. Um modelo de desenvolvimento de software onde as funções são utilizadas para lidar com abstrações, sendo principalmente adotado em *softwares* de maior complexidade que precisam manter a organização para chegar no objetivo final (NASCIMENTO, 2023).

Em concordância com Sebesta (2018), mesmo que o paradigma funcional apresente um crescimento relativamente grande no mercado, ainda não se tem um impacto notório nos times de desenvolvimento de software. Este cenário permite vários caminhos possam ser explorados: como a parte de inteligência artificial, armazenamento em nuvem, streaming de dados, área da agricultura, serviços financeiros e ciência de dados. Neste trabalho, pretende-se apresentar um projeto e implementação de um sistema de gestão de fazendas, a ser aplicado na área da agricultura. Em particular, espera-se promover a manipulação, tratamento e monitoramento de dados, para tarefas relacionadas a atividades no campo (SEBESTA, 2018).

A necessidade de mudanças que incorporem rapidez ao software fez com que as linguagens funcionais, ocupassem mais espaço nos projetos, sendo inseridas efetivamente no mercado mundial do desenvolvimento. O paradigma funcional traz benefícios para aplicações baseadas em hardware e as que possuem chips multinucleares, pois permitem a execução de instruções de forma paralela, os princípios de imutabilidade e ausência de estado, auxiliam na resolução de problemas complexos, que são processados e simplificados pelo computador, facilitando a solução (SILVA, LEITE, OLIVEIRA, 2019).

A primeira linguagem de programação funcional surgiu com a necessidade da resolução de problemas na área da inteligência artificial, que desde 1950, vem sendo usada para um significativo avanço nessa área, sendo explorada e evoluindo cada vez mais, principalmente em projetos e aplicações que demandam de uma grande massa de dados (SEBESTA, 2018).

Seguindo o crescimento do paradigma funcional, a linguagem Clojure, obteve sucesso no mercado, de modo que, empresas de grande porte e de distintas áreas, adotaram a linguagem como parte de seu sistema, a exemplo da Netflix, que adotou o paradigma funcional, para o streaming de dados, devido a grande quantidade de eventos para a análise operacional. Outras que adotaram o modelo foram a *The Climate Corporation*, *Polis* e Nubank, para as vertentes de armazenamento de dados meteorológicos, pesquisas com tecnologia de inteligência artificial e serviços financeiros, respectivamente (HICKEY, 2020).

Este trabalho tem como objetivo principal realizar um estudo sobre as linguagens funcionais, em particular a linguagem Clojure. Como objetivo específico, implementar um sistema de gestão de fazendas utilizando a linguagem Clojure, que pertence ao paradigma funcional.

O presente trabalho está organizado em cinco capítulos. O Capítulo 1, Introdução, apresenta os objetivos e as justificativas para a execução do trabalho. O Capítulo 2 aborda o conceito de programação funcional. O Capítulo 3 aborda os objetivos e uma visão sobre o projeto desenvolvido. O Capítulo 4 descreve as linguagens e bibliotecas utilizadas no desenvolvimento. O Capítulo 5 detalha o desenvolvimento do projeto. Finalmente, no Capítulo 6, apresentam-se as conclusões obtidas a partir da realização deste trabalho e direciona para possíveis trabalhos futuros.

1.2 OBJETIVOS

Este trabalho tem como principal objetivo explorar os conceitos e paradigmas da programação funcional, e de suas linguagens, em particular a linguagem Clojure. Como objetivo específico, pretende-se propor, modelar e desenvolver uma aplicação Web de gestão voltada para o agronegócio, com a capacidade de fornecer relatórios sobre as atividades no campo, utilizando-se dos conceitos funcionais e da linguagem Clojure.

1.3 JUSTIFICATIVAS

O presente trabalho justifica-se pois, linguagens funcionais são eficientes para lidar com grandes volumes de dados, o que é crucial no agronegócio, para a análise e interpretação dos dados. Os dados imutáveis e a ausência de efeitos colaterais, são outros dois fatores que favorecem a exatidão e garantem a integridade na manipulação dos dados. Desta forma, são apresentados os paradigmas e benefícios para fomentar o uso das linguagens funcionais e auxiliar em problemáticas relacionadas a grandes massas de dados.

1.4 MOTIVAÇÃO

A motivação para a realização deste trabalho está em apresentar o paradigma funcional, a fim de trazer uma alternativa para melhorar o desempenho na implementação de softwares que manipulam e tratam grandes massas de dados. Desta forma, o presente trabalho de conclusão de curso segue na direção de desenvolver uma aplicação de gestão de fazendas, que pode servir de base para trabalhos futuros relacionados à programação funcional.

1.5 PERSPECTIVAS DE CONTRIBUIÇÃO

Em concordância com Sebesta (2018), mesmo que o paradigma funcional apresente um crescimento relativamente grande no mercado, ainda não se tem um impacto notório nos times de desenvolvimento de software. O estudo aqui gerado poderá contribuir para dar notoriedade e fomentar o uso das linguagens funcionais em projetos futuros, principalmente aqueles que lidam com grandes volumes de dados e procuram tratar dados sem efeitos colaterais.

1.6 METODOLOGIA

Para atender aos objetivos estabelecidos, foram conduzidos estudos exploratórios das principais referências em relação ao paradigma funcional e em específico a linguagem de programação Clojure. No desenvolvimento do projeto foi utilizada a metodologia UML, para apresentar o projeto e os conceitos vistos nos estudos exploratórios. Seguido da fase de implementação, que utilizou-se a linguagem Clojure para o back-end com a biblioteca *Pedestal.io* para facilitar o desenvolvimento com Clojure, Clojurescript para o front-end, com a biblioteca Reagent do React, que auxilia na criação de interfaces interativas para aplicações web e o *framework* Tailwind CSS, que utiliza classes pré-definidas para projetar interfaces web. Na camada de persistência e banco de dados, optou-se pelo PostgreSQL.

2. PROGRAMAÇÃO FUNCIONAL

2.1 CONTEXTUALIZAÇÃO

Sebesta (2018) explica que a primeira linguagem de programação funcional surgiu com a necessidade da resolução de problemas na área de inteligência artificial, na década de 1950, quando matemáticos precisaram processar cálculos complexos a partir de listas encadeadas com símbolos matemáticos em um computador.

Em 1959, surgiu a primeira linguagem funcional pura, até hoje conhecida como *List Process Language* (LISP), criada pelo Dr. John McCarthy do *Massachussetts Institute of Technology* (MIT). Inicialmente, essa linguagem foi criada para manipular símbolos com o intuito de representar palavras e símbolos inviáveis de serem representados pelas linguagens convencionais (SILVA, OLIVEIRA, OLIVEIRA, 2019).

No paradigma de programação funcional os programas são formados, exclusivamente,por funções, trata a computação como uma avaliação de funções matemáticas.O próprio programa principal é uma função que recebe dados de entrada como argumentos e produz um resultado. A função principal é definida em termos de outras funções, as quais, por sua vez, são definidas em termos de outras mais simples, até o nível em que as funções são primitivas da linguagem (PELLEGRINI, 2018).

2.2 FUNÇÕES PURAS

Na programação funcional se trabalha com funções e variáveis, esse tipo de programação é conhecida como *stateless* (as funções podem ser chamadas em diversas partes do código, mas a cada chamada será como se fosse a primeira, ou seja, ela não tem estado). Outra característica importante da programação funcional são as funções de alta ordem, que significa passar uma função como parâmetro de outra função.

Assim, a função se torna um parâmetro dentro da sua função. Por isso, é comum dizer que, para que uma função seja considerada uma função pura, ela não pode ter efeito colateral, ou seja, não pode alterar o estado de qualquer variável ou objeto que possa ser observado fora do escopo da função.

Outra premissa importante sobre as funções puras é que elas devem depender apenas de seus argumentos (SILVA, OLIVEIRA, OLIVEIRA, 2019).

2.3 RECURSIVIDADE

A programação funcional não possui loops e, para que seja possível a chamada das funções, uma de suas características é a recursão, definida pela capacidade da função de chamar a si mesma várias vezes, até que uma condição seja satisfeita (SILVA, OLIVEIRA, OLIVEIRA, 2019).

2.4 IMUTABILIDADE

A programação funcional lida basicamente com funções e, à medida que uma variável é alocada na memória com um determinado valor associado, esse valor permanece inalterado até o fim e em todas as partes do código nas quais for usado (SILVA, OLIVEIRA, OLIVEIRA, 2019).

2.5 EXPRESSÕES LAMBDA E FUNÇÕES ANÔNIMAS

Suporte para funções que não necessitam ser nomeadas e podem ser definidas em uma expressão. São úteis para operações que requerem funções simples que são usadas uma única vez (SILVA, OLIVEIRA, OLIVEIRA, 2019).

2.6 REPL

Um Clojure *REPL* (sigla para *Read-Eval-Print Loop*) é um ambiente de programação que permite ao programador interagir com um programa Clojure em execução e modificá-lo, avaliando uma expressão de código por vez. O Clojure REPL dá ao programador uma experiência de desenvolvimento interativa. Ao desenvolver uma nova funcionalidade, ele

permite que ele crie programas primeiro executando pequenas tarefas manualmente. Ele permite que o programador reproduza rapidamente o problema, observe seus sintomas de perto, então improvise experimentos para rapidamente restringir a causa do bug e iterar em direção a uma correção (HICKEY, 2024).

2.7 EXEMPLOS DE UTILIZAÇÃO

2.7.1 *THE CLIMATE CORPORATION*

The Climate Corporation é uma empresa americana focada em agricultura digital e tecnologia climática. Fundada em 2006 inicialmente como *WeatherBill*, a empresa começou a fornecer seguros agrícolas baseados em dados meteorológicos. Em 2011, ela mudou o nome para *The Climate Corporation* e se concentrou em ajudar agricultores a melhorar a produtividade agrícola utilizando dados climáticos, modelos preditivos e ferramentas de software.

Clojure desempenha um papel importante na infraestrutura tecnológica da The Climate Corporation. A empresa adotou Clojure devido à sua expressividade, concisão e capacidade de lidar com grandes quantidades de dados de forma eficiente, o que é essencial para os tipos de análises e modelagens preditivas que a empresa realiza.

A The Climate Corporation utiliza Clojure como uma parte essencial de sua infraestrutura técnica para construir soluções inovadoras que auxiliam agricultores a enfrentar os desafios do clima e aumentar a eficiência e produtividade agrícola. A escolha de Clojure reflete o compromisso da empresa em utilizar tecnologias avançadas e confiáveis para resolver problemas complexos e fornecer valor significativo à agricultura moderna.

2.7.2 *POLIS*

Polis é uma ferramenta de tecnologia cívica que utiliza inteligência artificial e análise de dados para facilitar a coleta, visualização e compreensão de opiniões em grandes grupos de pessoas. A plataforma foi projetada para ajudar organizações, governos e comunidades a tomar decisões mais informadas e inclusivas, promovendo diálogos e debates públicos de forma estruturada e eficiente.

Uma das características mais notáveis do Polis é sua capacidade de visualização de dados, que mostra como as opiniões dos participantes se agrupam em tempo real, fornecendo insights imediatos sobre a dinâmica de discussão em um grupo. Clojure desempenha um papel central na arquitetura da Polis, permitindo à plataforma oferecer uma ferramenta poderosa para a participação pública e a análise de opiniões.

2.7.3 NUBANK

Nubank é uma das maiores fintechs da América Latina, fundada em 2013 no Brasil. A empresa revolucionou o setor bancário ao oferecer uma alternativa digital aos bancos tradicionais, começando com um cartão de crédito sem anuidade e expandindo rapidamente para oferecer uma ampla gama de produtos financeiros, incluindo contas digitais, empréstimos, seguros e investimentos.

Clojure é usada em várias partes do sistema da Nubank, desde o backend que processa transações financeiras até os serviços que gerenciam as operações internas da empresa. Desempenha um papel fundamental no sucesso técnico da Nubank, permitindo que a fintech construa sistemas financeiros robustos, escaláveis e confiáveis. A linguagem ajudou a Nubank a manter a qualidade e a segurança de seus serviços, ao mesmo tempo em que se expande rapidamente para novos mercados e produtos. A adoção de Clojure é um dos pilares que sustenta a inovação contínua da Nubank, ajudando a fintech a se destacar como líder na transformação digital do setor financeiro.

3. PROPOSTA DE TRABALHO

Neste capítulo será apresentado o objetivo do trabalho e uma breve descrição do projeto a ser desenvolvido.

3.1 OBJETIVOS

O capítulo visa apresentar o objetivo do trabalho, sendo ele apresentar os conceitos de programação funcional e desenvolver um protótipo de um aplicativo para gestão de fazendas, com base nos conceitos apresentados.

3.2 ANÁLISE DO PROJETO

O projeto consiste em criar um protótipo de um aplicativo web de gestão agrícola, utilizando a linguagem Clojure e os conceitos da programação funcional, visando facilitar o processo de coleta de dados, análise e geração de relatórios para auxiliar na tomada de decisões em fazendas. Apresentando relatórios que auxiliem na leitura e visualização dos dados para cada usuário no contexto da sua fazenda, desta forma, tornando mais fácil a tomada de decisões com relação à produtividade e eficiência. Sendo possível fazer registros de equipamentos, colaboradores, campos, centros de custos, plantios, zonas, talhões, para que possam ser monitorados e gerados relatórios que possibilitem ver o andamento e panorama das atividades.

A ideia seria capturar dados *“fakes”* para ser possível a geração dos relatórios, simulando capturas de deslocamento dos equipamentos, atividades dos colaboradores, talhões que foram executadas as ações, tipo de atividade realizada e custo das operações.

A programação funcional será utilizada pois um dos princípios-chave da programação funcional é a imutabilidade dos dados. Isso significa que, uma vez criados, os dados não podem ser alterados. Essa característica reduz a complexidade e os erros relacionados à gestão de estado, o que é particularmente útil em aplicações que realizam muitas operações de leitura e análise de dados.

A gestão agrícola envolve coletar, analisar e reportar uma grande quantidade de dados. Clojure oferece uma rica coleção de estruturas de dados imutáveis e funções de alta ordem que tornam a manipulação desses dados muito eficiente e expressiva, adequando-se bem à natureza dos dados e às necessidades de análise.

Clojure, sendo uma linguagem da JVM (*Java Virtual Machine*), permite integrar facilmente com o vasto ecossistema de bibliotecas e ferramentas Java. Isso é particularmente útil para acessar bibliotecas de análise de dados, visualização e outras funcionalidades que possam ser necessárias.

3.3 PROTÓTIPOS

Figura 1: Protótipo de Tela 1 - Login

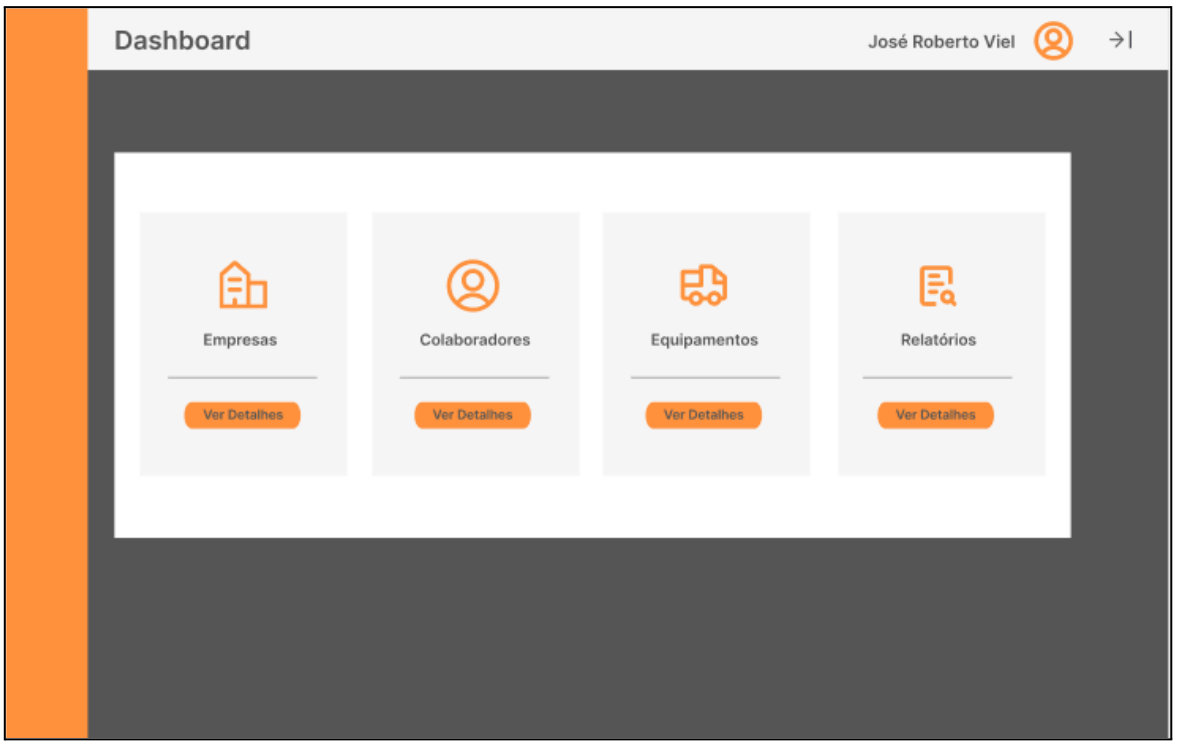

Figura 2: Protótipo de Tela 2 - Dashboard

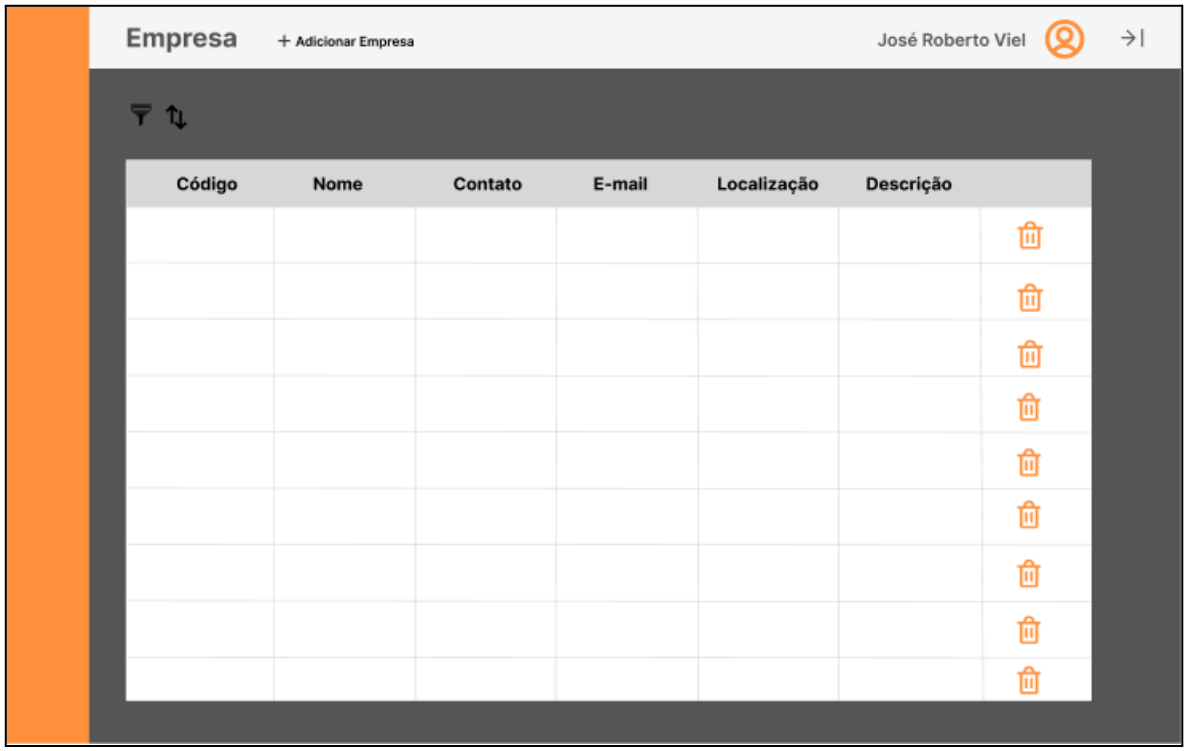

Figura 3: Protótipo de Tela 3 - Detalhes

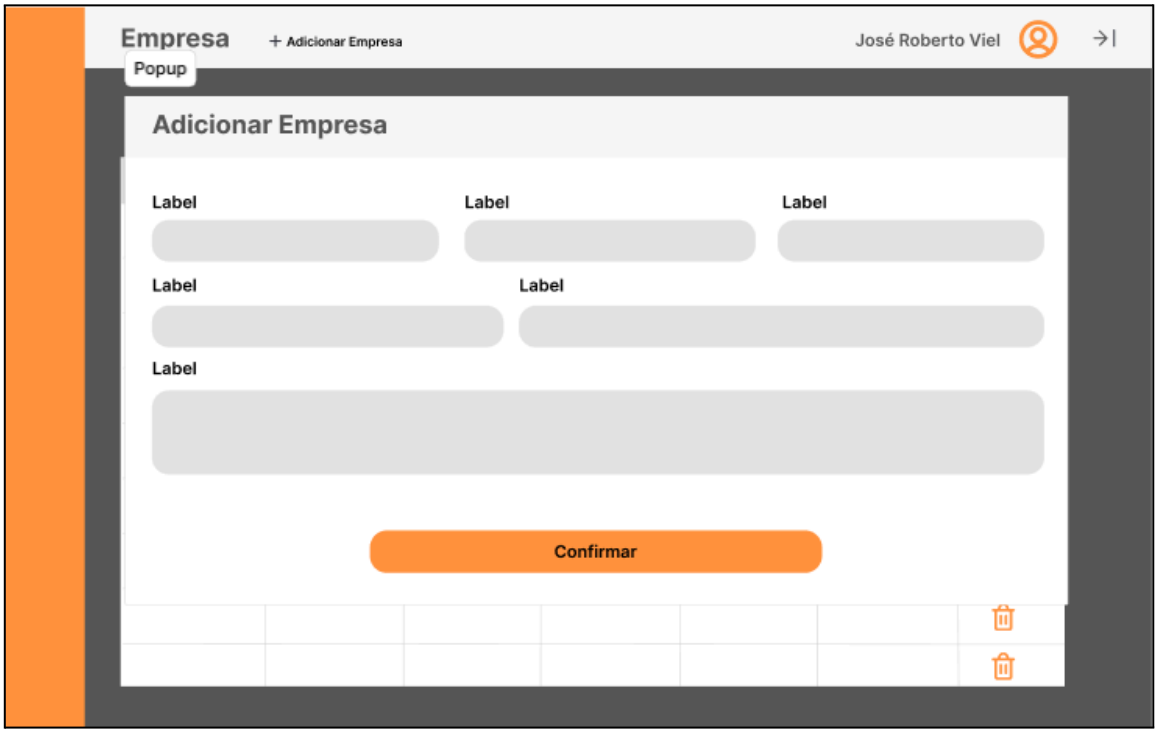

Figura 4: Protótipo de Tela 4 - Cadastro/Edição de Entidade

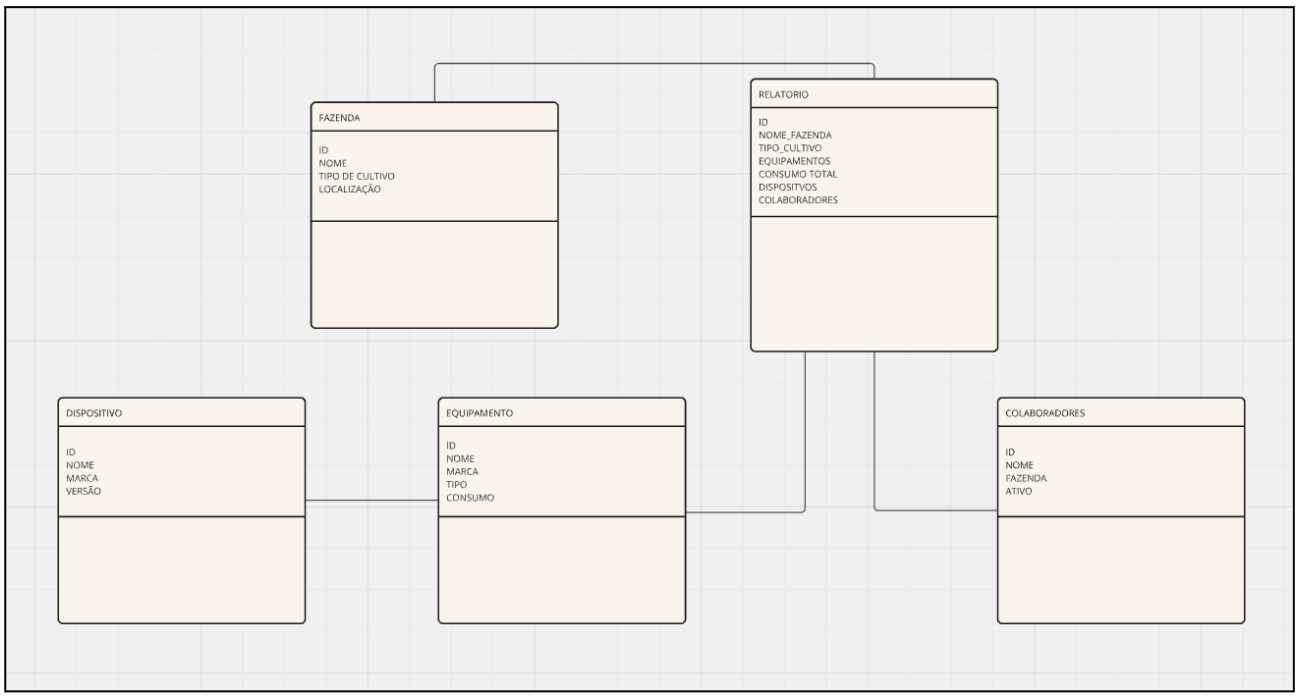

Figura 5: Diagrama de Classes

4. TECNOLOGIAS UTILIZADAS

Foi pensado na utilização do Clojure, já que é uma das linguagens que segue o conceito de programação funcional, permitindo lidar com grandes massas de dados e conceitos de imutabilidade, será útil ao realizar a manipulação dos dados, evitando que os dados sejam alterados por efeitos colaterais, trazendo maior exatidão e menor margem de erro.

Juntamente com o Clojure, vai ser utilizado o ClojureScript, que é uma tecnologia frontend, que se relaciona muito bem com Clojure, compartilhando de uma sintaxe bem parecida. Ainda para auxiliar no desenvolvimento frontend, a biblioteca Reagent será utilizada, pois permite mais eficiência na definição de componentes, interagindo muito bem com o ClojureScript. E por fim, o framework Tailwind CSS, para auxiliar e facilitar na utilização de classes na definição e estilização dos componentes.

Uma das vantagens na utilização do Clojure e Clojurescript juntos é que, enquanto o Clojure é executado na JVM e pode-se aproveitar o ecossistema JAVA no backend, o ClojureScript pode aproveitar do ecossistema JavaScript no frontend, já que é um compilador de Clojure voltado para o JavaScript, tendo maior abertura para utilizar bibliotecas que iterem com JavaScript, facilitando o desenvolvimento do código.

4.1 CLOJURE

Clojure foi projetada para ser uma linguagem funcional prática e de uso geral. Inicialmente sendo projetada em 2005 e lançada em 2007. É um dialeto LISP que contempla a programação com funções puras, dados imutáveis e gerenciamento de estado seguro. Clojure é hospedado intencionalmente, sendo assim compila e executa no tempo de execução de outra linguagem, como a *Java Virtual Machine*(JVM). Podendo assim, interoperar e aproveitar das bibliotecas da linguagem hospedeira de maneira direta e eficiente (HICKEY, 2020).

4.2 CLOJURESCRIPT

ClojureScript é uma linguagem de programação robusta, prática e rápida com um conjunto de recursos úteis que juntos formam uma ferramenta simples, coerente e poderosa. É um compilador para Clojure voltado para JavaScript. Ele emite código JavaScript compatível com o modo de compilação avançado do compilador otimizador

Google Closure, que foi um compilador desenvolvido pelo Google para uso em seus aplicativos Web, permitindo-se criar aplicativos web com JavaScript (HICKEY, 2024).

4.3 REAGENT

Reagent fornece uma interface minimalista entre ClojureScript e React. Ele permite que você defina componentes React eficientes usando nada além de funções e dados simples do ClojureScript. O objetivo do Reagent é tornar possível definir UIs arbitrariamente complexas usando apenas alguns conceitos básicos e ser rápido o suficiente por padrão para que você raramente precise pensar no desempenho.

4.4 TAILWIND CSS

O Tailwind CSS, é um framework CSS que visa a utilidade, fornecendo várias classes utilitárias, que você pode usar diretamente dentro de sua marcação de texto para projetar um elemento.

4.5 PEDESTAL.IO

Pedestal é um conjunto de bibliotecas que se usa para construir serviços e aplicativos da web. O Pedestal roda no back-end e pode manipular qualquer coisa, desde pequenos sites estáticos, a aplicativos tradicionais orientados a páginas, até aplicativos dinâmicos de página única utilizando eventos enviados pelo servidor e WebSockets. O pedestal se adapta às suas necessidades, é utilizado para trazer os principais atributos do Clojure, para o domínio do desenvolvimento web Clojure (NUBANK, 2024).

4.6 CURSIVE

O Cursive é construído no IntelliJ, traz funcionalidades como, gerenciamento de projetos até integrações de controle de versão em todas as plataformas. Cursive é escrito inteiramente em Clojure, permitindo-nos integrar facilmente todas as ferramentas fantásticas do ecossistema Clojure, Leiningen e REPL.

5. DESENVOLVIMENTO DO TRABALHO

Desenvolvimento projeto backend

Este projeto foi desenvolvido utilizando a linguagem Clojure, um dialeto de Lisp que roda na JVM (Java Virtual Machine), e foi configurado utilizando o ambiente de desenvolvimento integrado (IDE) IntelliJ IDEA com o plugin Cursive, foi estruturado utilizando o Leiningen, uma ferramenta de automação de projetos para Clojure. O projeto é um backend API que implementa os métodos HTTP padrão (GET, PUT, POST, DELETE) para manipulação de dados, utilizando um banco de dados PostgreSQL para persistência.

O banco de dados PostgreSQL foi escolhido por sua robustez e conformidade com o padrão SQL. A conexão com o banco de dados foi gerida pela biblioteca clojure.java.jdbc, que fornece uma interface simples para executar operações SQL diretamente a partir do código Clojure.

A comunicação de dados entre cliente e servidor segue o padrão JSON-RPC. Cada requisição JSON-RPC inclui um campo method para especificar a ação desejada e um campo params para os dados necessários.

Desenvolvimento projeto frontend

Este projeto frontend foi desenvolvido utilizando ClojureScript, uma versão de Clojure que compila para JavaScript, tornando-a adequada para o desenvolvimento de aplicações web. O projeto utiliza Reagent, uma biblioteca ClojureScript que oferece uma interface reativa e eficiente para a criação de componentes baseados em React, e re-frame, um framework que facilita o gerenciamento do estado da aplicação de forma unidirecional e previsível. A combinação dessas ferramentas proporciona uma base sólida para a criação de interfaces de usuário interativas e dinâmicas.

O re-frame foi utilizado para gerenciar o estado da aplicação de forma centralizada. A arquitetura re-frame segue o padrão unidirecional de fluxo de dados, garantindo que o estado da aplicação seja previsível e fácil de depurar. O estado é mantido em um átomo (app-db), e todas as interações com o estado passam por eventos e *subscriptions*.

A aplicação frontend interage com o backend por meio de requisições HTTP. O re-frame facilita essa interação por meio de efeitos (effects), que são funções que realizam operações assíncronas, como chamadas HTTP.

Telas do Projeto

A Figura 6 representa a tela de login da aplicação, formada por uma interface minimalista e simples, apenas para compor o sistema.

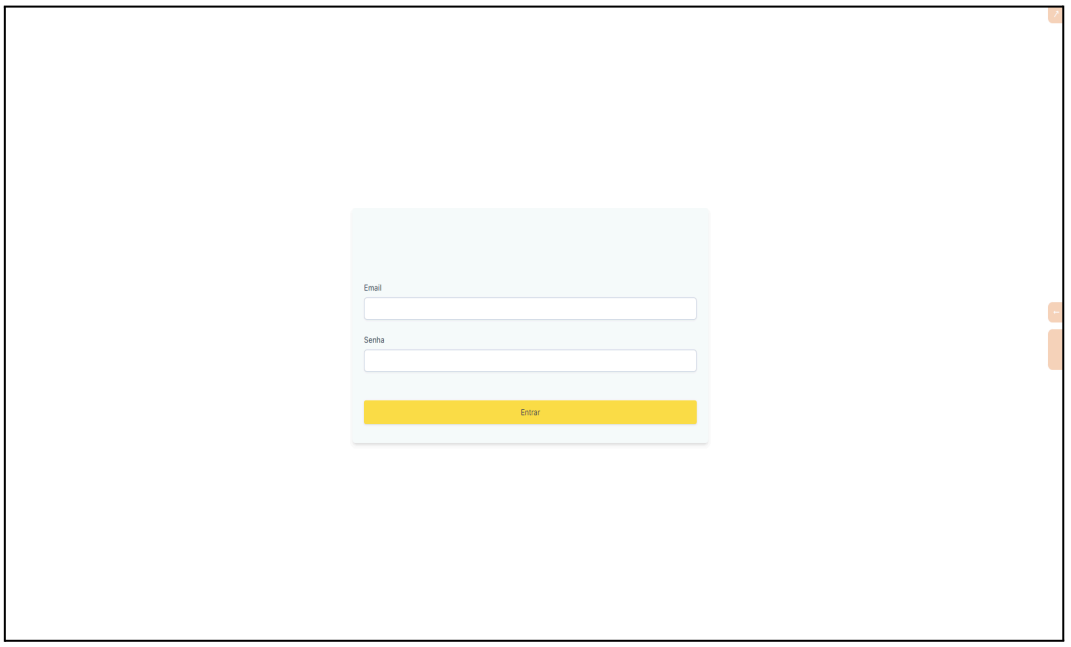

Figura 6: Tela de Login da Aplicação

A Figura 7, representa a tela de detalhes das entidades, todas se assemelham a esta, que está representada. Nela estão contidos alguns filtros que podem ser feitos de acordo com as informações que estão nas tabelas, disponibiliza de uma tabela que contém as informações da entidade cadastrada com seus atributos. Ainda podendo excluir de forma simples com o botão disponibilizado em cada linha da tabela.

Ainda nesta tela é disponibilizado uma paginação, para o caso de muitas informações, exibindo o número total de registros e podendo alterar o número de registros disponibilizados na tela.

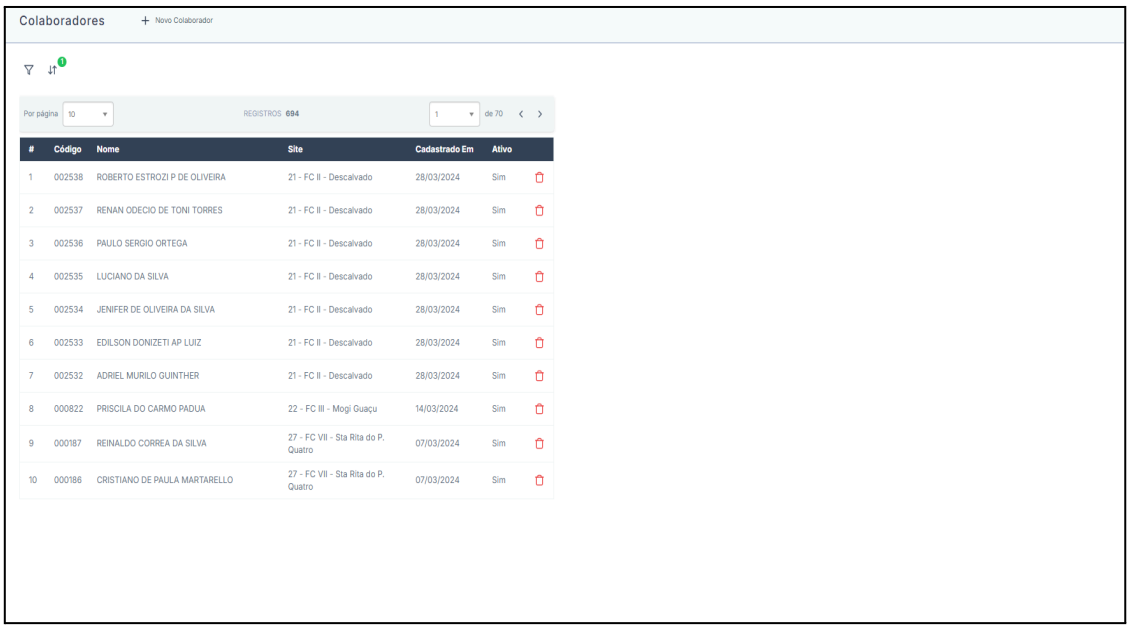

Figura 7: Detalhes da Entidade

Ainda na tela de detalhes da entidade é possível clicar no botão existente no cabeçalho, onde é mostrado um modal, para a inclusão de um novo registro, onde se insere as informações e se confirma no botão amarelo, para gerar então um novo registro.

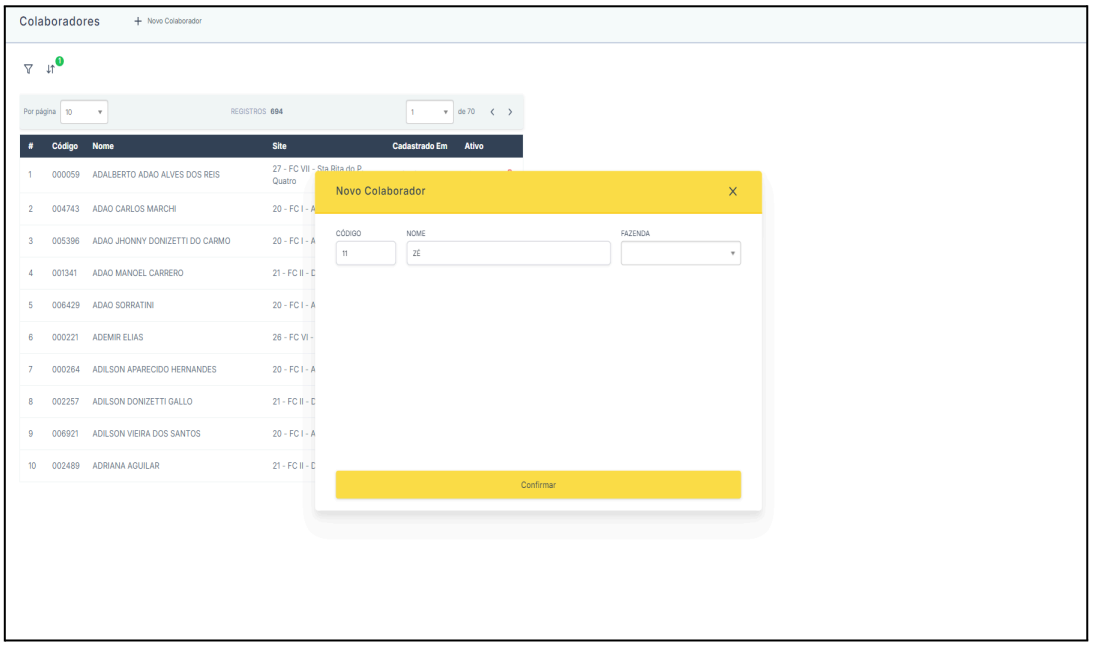

Figura 8: Tela de Cadastro de Entidades

Na Figura 9, é mostrado o exemplo, de quando já há um registro cadastrado, e então, o usuário pode clicar na linha da tabela e é disponibilizado um modal, com as informações do registro já criado, possibilitando a edição do mesmo.

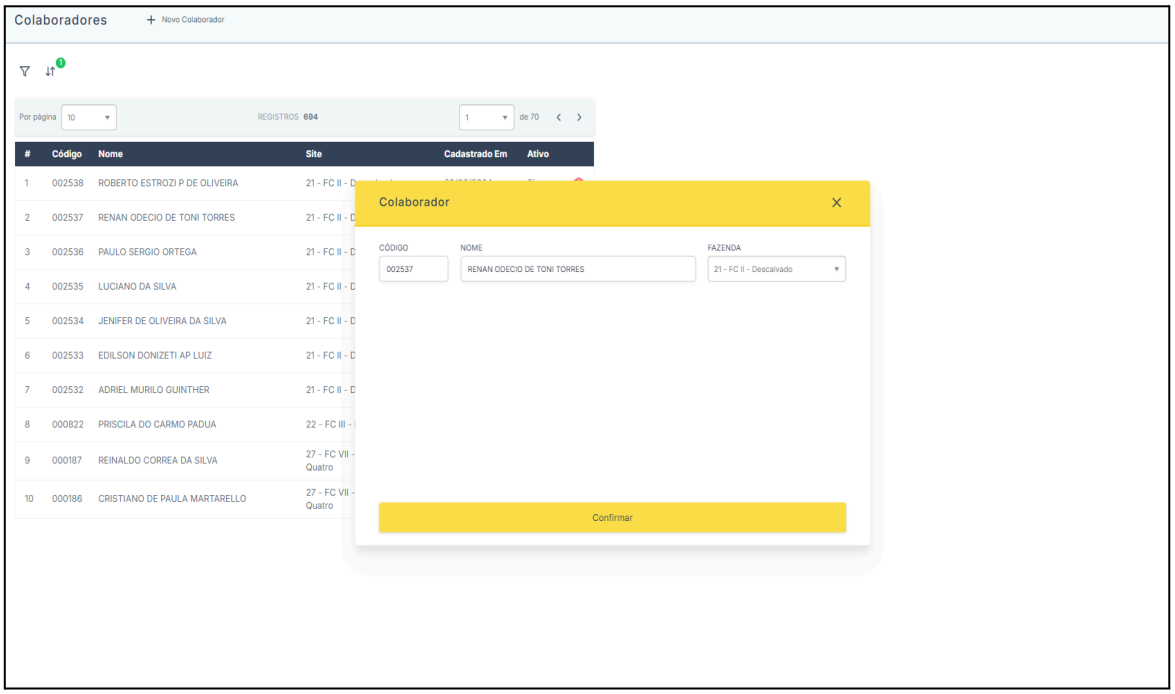

Figura 9: Tela de Edição de Entidades

Códigos da Aplicação

O trecho de código exemplificado na Figura 9 demonstra um exemplo de um método GET que compõe a API de colaboradores, neste trecho é definida a função para a requisição, construindo um contexto local com a função let, para a execução da requisição. Onde, nesse contexto, é utilizado *jdbc*, um namespace que geralmente contém funções para interagir com um banco de dados e o *execute!* que é uma função que executa uma instrução SQL no banco de dados. Ainda está sendo usado o *:builder-fn*, este é um parâmetro adicional que configura como os resultados da query serão formatados, neste caso, estão sendo formatados em mapas e letras minúsculas.

```
no usages  Jose Roberto Viel *
(defn all-operators
      "Retorna todos os Operadores."
      [request]
      (let [{:keys [datasource]} request]
           {:status 200
            :body
            (jdbc/execute! datasource
                           ["SELECT code, name, farm FROM operator"]
                           {:builder-fn rs/as-unqualified-lower-maps})}))
```

Figura 10: Método GET

O método de criação, *create-operator*, é responsável por criar um novo operador. Ele recebe um *request*, do qual extrai informações como a conexão com o banco de dados (datasource) e os parâmetros JSON fornecidos na requisição (json-params). A partir desses parâmetros, ele obtém os valores de code, name e farm, que representam o código do operador, seu nome e a fazenda a que ele está associado, respectivamente. Em seguida, usa esses valores para executar um comando SQL de inserção no banco de dados, adicionando um novo registro na tabela *operator*. Por fim, retorna uma resposta indicando que o operador foi criado com sucesso, com o status HTTP 201.

```
no usages  Jose Roberto Viel *
(defn create-operator
      "Cria um novo Operador."
      [request]
      (let [{:keys [datasource json-params]} request
            {:keys [code name farm]} json-params]
           (jdbc/execute! datasource
                          ["INSERT INTO operator (code, name, farm) VALUES (?, ?, ?)"
                           code name farm])
           {:status 201
            :body "Operador criado com sucesso"}))
```

Figura 11: Método POST

O método de atualização, *update-operator*, é usado para atualizar um operador existente. Ele também recebe um *request* e, semelhante ao método anterior, extrai o datasource e os parâmetros da requisição. No entanto, ele também precisa do id do operador a ser atualizado, que é extraído dos parâmetros de caminho (path-params). A partir dos parâmetros JSON, ele obtém os valores e usa essas informações para executar um comando SQL de atualização, alterando os campos correspondentes na tabela operator para o operador identificado pelo id fornecido. Após a execução bem-sucedida do comando, o método retorna uma resposta com o status HTTP 200, indicando que o operador foi atualizado com sucesso.

```
no usages  Jose Roberto Viel *
(defn update-operator
      "Atualiza um Operador existente."
      [request]
      (let [{:keys [datasource path-params json-params]} request
            {:keys [id]} path-params
            {:keys [code name farm]} json-params]
           (jdbc/execute! datasource
                          ["UPDATE operator SET code = ?, name = ?, farm = ? WHERE id = ?"
                           code name office work id])
           {:status 200
            :body "Operador atualizado com sucesso"}))
```

Figura 12: Método PUT

O método de exclusão, *delete-operator*, trata da exclusão de um operador. Ele recebe um *request* e, assim como no método de atualização, obtém o datasource e o id do operador a ser excluído dos parâmetros de caminho. Com o *id* em mãos, ele executa um comando SQL de exclusão, removendo o operador correspondente da tabela operator. Se a operação for bem-sucedida, o método retorna uma resposta com o status HTTP 200, indicando que o operador foi excluído com sucesso.

```
no usages  Jose Roberto Viel
(defn delete-operator
      "Exclui um Operador."
      [request]
      (let [{:keys [datasource path-params]} request
            {:keys [id]} path-params]
           (jdbc/execute! datasource
                          ["DELETE FROM operator WHERE id = ?"
                           id])
           {:status 200
            :body "Operador excluído com sucesso"}))
```

Figura 13: Método DELETE

Os métodos citados acima, são a base da aplicação, todas as entidades que são utilizadas na aplicação contém esses métodos.

6. CONCLUSÕES

A capacidade da programação funcional de facilitar a manipulação de grandes volumes de dados, com eficiência e menor propensão a erros, estimula ainda mais seu uso para o desenvolvimento de projetos. Os sistemas desenvolvidos com base nesses princípios não apenas aumentam a precisão das previsões e análises, mas também oferecem uma abordagem mais modular e escalável para o desenvolvimento de um software.

O desenvolvimento deste aplicativo de gestão agrícola utilizando programação funcional destaca o potencial dessa abordagem para a inovação no agronegócio. Este projeto não só atesta a viabilidade da programação funcional em contextos práticos e desafiadores, como também abre caminhos para futuras pesquisas e desenvolvimentos na interseção entre tecnologia e agricultura.

Foi possível pontuar durante o desenvolvimento do projeto, alguns tópicos importantes que foram essenciais para alcançar os objetivos propostos na parte prática:

Imutabilidade: A imutabilidade e a ausência de efeitos colaterais, que são características da programação funcional, garantiram um comportamento mais previsível, reduzindo a incidência de bugs e problemas relacionados a estados inconsistentes.

Facilidade na manipulação de grandes volumes de dados: Com o uso de funções puras e bibliotecas otimizadas, foi possível processar dados em larga escala de forma eficiente e organizada.

Reuso de código: O código desenvolvido permitiu que partes do sistema pudessem ser reutilizadas em outros contextos ou projetos futuros, além de facilitar a manutenção.

Facilidade no entendimento: A programação funcional e a linguagem Clojure, acabaram sendo utilizadas também, por sua facilidade e didática na escrita dos códigos, o que facilita a leitura, além do entendimento, para desenvolvimentos futuros.

6.1 TRABALHOS FUTUROS

Para o desenvolvimento e expansão da aplicação, propõe-se a implementação de um sistema de rastreamento em tempo real dos equipamentos por meio de um mapa interativo. Esse sistema seria alimentado por dados provenientes de dispositivos embarcados nos equipamentos, permitindo a localização e o monitoramento contínuo desses ativos. O objetivo é fornecer uma visão clara e atualizada da posição de cada equipamento, o que pode facilitar a coordenação das operações e a gestão dos recursos no campo.

Além do rastreamento geográfico, a aplicação será ampliada para incluir diversas funcionalidades que apoiarão a gestão operacional e a tomada de decisão. Será possível registrar e gerenciar diferentes frentes de trabalho, associando cada atividade ao equipamento utilizado e ao estado atual desses ativos. A integração de alertas automatizados será uma característica importante, fornecendo notificações em tempo real sobre o estado dos equipamentos, como o nível de consumo de combustível e eventuais paradas ou falhas dos colaboradores. Esses alertas permitirão uma resposta rápida e eficiente a problemas que possam surgir, otimizando a operação e reduzindo o tempo de inatividade dos equipamentos.

Para complementar essa solução é oferecer uma experiência de usuário mais fluida e acessível, considera-se o desenvolvimento de um aplicativo móvel. Este aplicativo será baseado em um paradigma funcional e utilizará uma linguagem adequada para o contexto, como o Elixir. A escolha do Elixir, com seu suporte robusto à concorrência e suas capacidades para construir sistemas distribuídos, permitirá uma integração eficiente entre a aplicação web e o aplicativo móvel. Assim, a tomada de decisão no campo será facilitada, com dados e alertas acessíveis diretamente nos dispositivos móveis dos usuários. A integração multiplataforma proporcionada por essa abordagem garantirá uma experiência consistente e eficiente, independentemente do dispositivo utilizado.

Com essas implementações, busca-se criar um sistema coeso que não apenas melhora a visibilidade e o controle sobre os equipamentos, mas também proporciona ferramentas avançadas para a gestão e otimização das operações no campo.

REFERÊNCIAS

BONNAIRE-SERGEANT, AMBROSE; ROWAN, DAVIES; TOBIN-HOCHSTADT. **Practical Optional Types for Clojure**. 2016. 94p. Artigo - Springer-Verlag, Bade-Vurtemberga, Heidelberg, 2016.

BORÉM, ALUÍZIO; QUEIROZ, DANIEL MARÇAL; VALENTE, DOMINGOS SÁRVIO MAGALHÃES; PINTO, FRANCISCO DE ASSIS DE CARVALHO. **Agricultura Digital**. 2. ed. São Paulo: Oficina de Textos, 2021.

HICKEY, RICH. **A History of Clojure**. 2020. 46p. Artigo - Technische Universität Darmstadt, Hesse, Darmstadt, 2020.

HICKEY, RICH. **The Clojure Programming Language**. Disponível em <https://clojure.org/index> Acesso em: 22 nov.2023.

HICKEY, RICH. **ClojureScript**. Disponível em <https://clojurescript.org/#:~:text=ClojureScript%20is%20a%20compiler%20for,the%20Google%20Closure%20optimizing%20compiler> Acesso em: 08 mar.2024.

HICKEY, RICH. **Programming at the REPL: Introduction**. Disponível em <https://clojure.org/guides/repl/introduction> Acesso em: 10 ago. 2024.

HU, ZHENJIANG; HUGHES, JOHN; WANG, MENG. **How functional programming mattered**. 2015. 22p. Artigo - Department of Informatics - SOKENDAI, Tokyo, Japan, 2015.

HUDAK, PAUL. **Conception, Evolution, and Application of Functional Programming Languages**. 1989. 53p. Artigo - Department of Computer Science - Yale University, Connecticut, New Haven, 1989.

NASCIMENTO, FELIPE. **Programação Funcional: O que é?**. Disponível em <https://www.alura.com.br/artigos/programacao-funcional-o-que-e?gclid=CjwKCAjw-b-kBBEiwA4fvKrL8_oxiD88CWc4fRgPJokaYbnzi6fm_rOtmc3p3cvWlzP_LMSdTLOhoCqWIQAvD_BwE> Acesso em: 18 jun. 2023.

NUBANK. **What is Pedestal?**. Disponível em <http://pedestal.io/pedestal/0.7/index.html> Acesso em: 24 ago. 2024.

PELLEGRINI, C. JERÔNIMO. **Programação Funcional e Concorrente com Scheme**. Versão 134. UFABC - Universidade Federal do ABC, Santo André, 2018.

ROY, SOHAM DE. **O que é Tailwind CSS? Um guia para iniciantes**. Disponível em <https://www.freecodecamp.org/portuguese/news/o-que-e-tailwind-css-um-guia-para-iniciantes/> Acesso em: 08 mar.2024.

SEBESTA, ROBERT W. **Conceitos de Linguagens de Programação**. 11. ed. Mariana Belloli Cunha. Tradução de João Eduardo Nóbrega Tortello. Porto Alegre: Editora Bookman, 2018.

SILVA, FABRÍCIO M.; LEITE, MÁRCIA C. D.; OLIVEIRA, DIEGO B. **Paradigmas de Programação**. 1. ed. Mirela Favaretto. Porto Alegre: Editora SAGAH, 2019.