SERVIÇO PÚBLICO FEDERAL

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS

PRÓ-REITORIA DE ENSINO

CÂMPUS GOIÂNIA

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GUILHERME HENRIQUE SOUZA NASCIMENTO

Dashboards como Ferramenta de Gestão Eficiente na Pecuária Leiteira: Um Estudo de Caso

Goiânia, 2022.

SERVIÇO PÚBLICO FEDERAL

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS

PRÓ-REITORIA DE ENSINO

CÂMPUS GOIÂNIA

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GUILHERME HENRIQUE SOUZA NASCIMENTO

Dashboards como Ferramenta de Gestão Eficiente na Pecuária Leiteira: Um Estudo de Caso

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Bacharelado em Sistemas de Informação como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Eduardo Noronha de Andrade Freitas

Goiânia, 2022.

N244d Nascimento, Guilherme Henrique Souza.
Dashboards como ferramenta de gestão eficiente na pecuária leiteira: um estudo de caso / Guilherme Henrique Souza Nascimento. – Goiânia: Instituto Federal de Educação, Ciência e Tecnologia de Goiás, 2022.
53f.

Orientação: Prof. Dr. Eduardo Noronha de Andrade Freitas.

TCC (Trabalho de Conclusão de Curso) – Curso de Bacharelado em Sistemas de Informação, Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

1. Dashboards. 2. Banco de dados relacional. 3. Pecuária leiteira. I. Freitas, Eduardo Noronha de Andrade (orientação). II. Instituto Federal de Educação, Ciência e Tecnologia de Goiás. III. Título.

CDD 005.74

Ficha catalográfica elaborada pela bibliotecária Lana Cristina Dias Oliveira CRB1/ 2.631
Biblioteca Professor Jorge Félix de Souza,
Instituto Federal de Educação, Ciência e Tecnologia de Goiás, Câmpus Goiânia.

INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
SISTEMA INTEGRADO DE BIBLIOTECAS

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO NO REPOSITÓRIO DIGITAL DO IFG - ReDi IFG

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia de Goiás, a disponibilizar gratuitamente o documento no Repositório Digital (ReDi IFG), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IFG.

Identificação da Produção Técnico-Científica

[ ] Tese	[ ] Artigo Científico
[ ] Dissertação	[ ] Capítulo de Livro
[ ] Monografia – Especialização	[ ] Livro
[X] TCC - Graduação	[ ] Trabalho Apresentado em Evento
[ ] Produto Técnico e Educacional - Tipo: _____	

Nome Completo do Autor: Guilherme Henrique Souza Nascimento
Matrícula: 20181011090091
Título do Trabalho: Dashboards como ferramenta de gestão eficiente na pecuária leiteira: Um estudo de caso.

Autorização - Marque uma das opções

1. (X) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso aberto);
2. ( ) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG somente após a data ___/___/_____ (Embargo);
3. ( ) Não autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso restrito).

Ao indicar a opção **2 ou 3**, marque a justificativa:

( ) O documento está sujeito a registro de patente.
( ) O documento pode vir a ser publicado como livro, capítulo de livro ou artigo.
( ) Outra justificativa: _____

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

i. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
ii. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
iii. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

Goiânia/Goiás , 03/01/2022.
Local Data

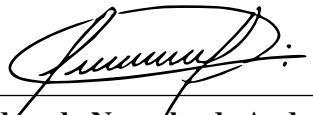
Assinatura do Autor e/ou Detentor dos Direitos Autorais

SERVIÇO PÚBLICO FEDERAL
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
PRÓ-REITORIA DE ENSINO
CÂMPUS GOIÂNIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

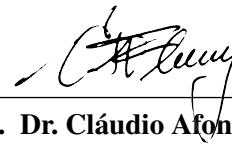
GUILHERME HENRIQUE SOUZA NASCIMENTO

Dashboards como Ferramenta de Gestão Eficiente na Pecuária Leiteira: Um Estudo de Caso

Trabalho de Conclusão de Curso defendido no Curso de Bacharelado em Sistemas de Informação como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação, aprovada em 07 de Fevereiro de 2022, pela Banca Examinadora constituída pelos professores:



Prof. Dr. Eduardo Noronha de Andrade Freitas
Departamento IV - IFG / Câmpus Goiânia
Presidente da Banca



Prof. Dr. Cláudio Afonso Fleury
Instituto Federal de Goiás - IFG

Msc. Edson da Silva Ramos
Universidade Federal de Goiás - UFG

RESUMO

Título: Dashboards como Ferramenta de Gestão Eficiente na Pecuária Leiteira: Um Estudo de Caso
Autor: Guilherme Henrique Souza Nascimento
Orientador: Dr. Eduardo Noronha de Andrade Freitas

O presente trabalho é um estudo de caso que visa a aplicação de técnicas e ferramentas de gestão da informação em uma fazenda. Nele é desenvolvido um *dashboard* em suporte ao monitoramento e na tomada de decisão nas atividades relacionadas a produção de leite da fazenda. O sistema desenvolvido é baseado na arquitetura de três camadas, sendo elas: camada de aplicativo, camada de dados e camada de apresentação. O escopo do trabalho abrange tanto na realização de transações no banco de dados, quanto na criação do *back-end* e *front-end* da aplicação. Para isso são utilizados diferentes tecnologias como os *frameworks* Spring, Hibernate, Angular e também diferentes conhecimentos a respeito de programação, programação web, interação homem-computador, banco de dados, etc. No trabalho são apresentados diferentes aspectos relacionados a implementação e também sobre as funcionalidades que foram desenvolvidas e como elas podem beneficiar a fazenda nos seus negócios.

Palavras–chave

Dashboard, Gestão eficiente, Pecuária de precisão, Produção de leite, Desenvolvimento web, Front-end, Back-end, Angular, Spring, Hibernate, Spring Boot, SQL, Web Services Rest, API.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Além de sua importância nutricional, o leite assume papel de destaque na economia brasileira, sendo o Brasil o país que ocupa o terceiro lugar no ranking dos maiores produtores de leite do mundo. A receita na produção primária desse produto foi cerca de R$ 35 bilhões, ocupando o sétimo lugar entre os produtos da agropecuária. Na indústria dos alimentos, o valor do faturamento líquido chegou a R$70,9 bilhões, ficando atrás somente dos setores derivados de carne e beneficiados de café, chá e cereais (DENIS; CARVALHO; RESENDE, 2020).

Apesar do papel de destaque na economia, existem muitos desafios que precisam ser enfrentados por quem decide trabalhar nesse ramo. Os desafios são em áreas diversas, estando presentes em aspectos de logística, saúde do animal, comercialização, entre outros. Na logística, cita-se o desafio de decidir quais animais devem ser mantidos como produtores de leite e quais devem ser abatidos para o consumo. Em questão da saúde do animal, pode-se citar o desafio de identificar e tratar diferentes doenças, como por exemplo a mastite, que além de diminuir a qualidade e quantidade do leite produzido, leva muitas vezes a necessidade de descartar o animal (MITTELBACH *et al.*, 2011).

Para lidar com os diferentes desafios, destaca-se a pecuária de precisão, que segundo a Embrapa é definida como : "Uma postura gerencial amparada em tecnologias da informação e comunicação que permite a melhoria do controle da fonte de variabilidade animal e espacial, otimizando o desempenho econômico, social e ambiental da fazenda leiteira" (EMBRAPA, 2018). Nessa abordagem, é utilizado de diferentes tecnologias para medir, armazenar e analisar indicadores de produtividade e saúde do animal. Dentre os benefícios dessa abordagem, pode-se citar a identificação e tratamento precoce de doenças, que facilita na recuperação do animal e evita perdas financeiras (EMBRAPA, 2018).

No contexto de monitoramento e tomada de decisão na pecuária de precisão, pode-se citar as ferramentas de *dashboard*. Estas permitem que diversas métricas e indicadores sejam representados visualmente de forma centralizada. Ao utilizar *dashboards*, é possível obter benefícios, como: aumento de produtividade, economia de tempo, economia de dinheiro, agilidade na tomada de decisão, melhora na tomada de decisão, etc (GOMES, 2017) (COSTA, 2020).

O presente trabalho consiste em um estudo de caso, no qual é desenvolvido uma ferramenta de *dashboard* para uma fazenda produtora de leite. Esta fazenda tem utilizado a abordagem da pecuária de precisão, possuindo diferentes tecnologias para auxiliar em seus processos, como a utilização de sensores para medir indicadores referentes às

ordenhas das vacas. O *dashboard* desenvolvido deve permitir o monitoramento, análise e tomada de decisão utilizando-se destes dados. É esperado que a ferramenta possa auxiliar na identificação de doenças e também ajude na análise de desempenho dos animais.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo trata de diferentes aspectos teóricos da implementação do *dashboard* e também a respeito das tecnologias que foram utilizadas para realizar essa tarefa. Na seção 2.1 será abordado sobre as ferramentas de *dashboards*, na seção 2.2 será abordado sobre a arquitetura em três camadas, na seção 2.3 será abordado sobre aplicações Web, na seção 2.4 será explicado diferentes aspectos sobre banco de dados, na seção 2.5 será abordado sobre o que são e quais as vantagens de utilizar *web services*, na seção 2.6 explicado as vantagens e funcionalidades do *framework* Spring, na seção 2.7 é tratado sobre a ferramenta Maven, na seção 2.8 é tratado sobre o Hibernate e também sobre o *JPA*, na seção 2.9 é tratado sobre o *framework* Angular e na seção 2.10 são abordados os diferentes padrões de projetos utilizados no presente trabalho.

2.1 Dashboards

Dashboards são ferramentas utilizadas no auxílio à tomada de decisão e no entendimento de um determinado sistema. Eles são capazes de disponibilizar diferentes informações em diferentes formatos, como: gráfico, tabelas, mapas entre outras. O objetivo de um *dashboard* é ter as informações relevantes em um mesmo local para o fácil entendimento de um sistema analisado, podendo esse sistema ser um site, empresa, peças, equipamentos, animais etc (GOMES, 2017).

As informações que estão presentes no *dashboard* e a maneira que são mostradas dependem muito da característica do problema analisado. No geral, deve-se conter somente informações que possam ser essenciais para o entendimento claro e rápido do sistema. Essas informações devem ser disponibilizadas da melhor maneira possível para que se possa entender o sistema, ter novas ideias, conseguir resolver problemas e tomar diferentes ações dependendo do objetivo do *dashboard* que foi desenvolvido (COSTA, 2020).

Os *dashboards* podem ser classificados de diferentes formas de acordo com seu objetivo, as três principais classificações são:

- Operacional: Ajudam a entender o que está acontecendo no momento (COSTA, 2020).
- Analítico: Ajudam a ter uma visão clara de possíveis problemas e tendências, ajudando a ter novas ideias e encontrar problemas (COSTA, 2020).

- Estratégico: Permite monitorar as principais estratégias de objetivos para o negócio, neles são mostrados diferentes indicadores chaves de desempenho (KPI) (COSTA, 2020).

2.2 Arquiteturas de três camadas

A arquitetura em três camadas é um tipo de arquitetura que se baseia na divisão de uma aplicação em três camadas, cada uma possuindo uma separação lógica e física. A vantagem dessa separação é que cada camada pode ser trabalhada separadamente por diferentes equipes e que as alterações em uma dessas camadas não interfiram nas demais camadas. Em outras palavras, não existe um acoplamento entre as camadas (IBM, 2020). A seguir são apresentadas e explicadas cada uma das três camadas que compõe esse modelo de arquitetura:

- Camada de apresentação: Essa camada é responsável pela comunicação direta com o usuário, recebendo informações do mesmo e também disponibilizando novas informações à medida que o mesmo interage com a aplicação (IBM, 2020).
- Camada de aplicativo: É a camada que faz a comunicação entre a camada de apresentação e a camada de dados. Essa camada possui as regras de negócio, e nela os dados são processados e validados (IBM, 2020).
- Camada de dados: É a camada que fica responsável por armazenar e acessar os dados, normalmente é representado por um sistema gerenciador de banco de dados (IBM, 2020).

2.3 Aplicações Web

As aplicações Web se baseiam na arquitetura cliente-servidor. Nessa arquitetura, o cliente requisita determinados serviços ou recursos ao servidor, o servidor por sua vez fica responsável por atender essas requisições ao cliente. Nas aplicações Web o cliente é a própria máquina do usuário e o servidor é uma máquina remota que fica responsável por processar essas requisições (PAUMEIRA, 2012).

Para que haja a comunicação entre o cliente e servidor nas aplicações Web é utilizado o protocolo *http*, que determina as diretrizes de como ocorre a troca de mensagens entre essas duas entidades. Ele determina tanto o formato das mensagens como também disponibiliza diferentes métodos que permitem ao usuário se comunicar com o servidor, sendo os principais: *GET*, *POST*, *PUT* e *DELETE* (MOZILLA, c2005).

Nas aplicações Web, o software desenvolvido para executar do lado do cliente é denominado *front-end*. Nele é determinado o visual da aplicação e também a maneira na

qual o usuário interage com a mesma. Ele também é responsável por realizar requisições ao servidor à medida que o usuário interage com o mesmo. O software que roda no cliente é passível de ser alterado pelo usuário e por isso se faz necessário do *back-end* para lidar com questões de segurança, validação, regras de negócio e acesso e manipulação dos dados (SOUTO, 2019).

2.4 Banco de dados relacional

A modelagem relacional é baseada na teoria dos conjuntos e considera que diferentes objetos possuem características próprias e se relacionam entre si. Dessa forma, ela utiliza-se de tabelas, linhas e colunas para representar diferentes tipos de objetos, suas características e os seus relacionamentos (FALEIRO, 2011).

- Tabela ou entidade: Armazenam os dados referentes a um determinado tipo de conjunto de elementos.
- Colunas: Determinam as propriedades dos elementos desses conjuntos.
- Linha ou tupla: Representam um elemento pertencente ao conjunto

Banco de dados relacionais utilizam da modelo relacional para separar a estrutura lógica dos dados da maneira que são armazenados fisicamente. Desse modo o usuário precisa conhecer somente a modelagem, não precisando lidar necessariamente com questões sobre onde e como estão armazenados os dados. Através de operações na estrutura lógica, os dados são criados, alterados, deletados e acessados fisicamente (ORACLE, c1995a).

Os bancos de dados relacionais, quando comparado com não relacionais, têm como uma de suas principais vantagens a capacidade de manter consistência dos dados entre diferentes aplicações e bancos de dados. Além disso, eles possuem a característica de manter a atomicidade das transações, dessa forma é possível que quando executado uma sequência de operações, as alterações só sejam efetivadas quando todas as operações sejam realizadas sem nenhum problema (ORACLE, c1995a).

2.4.1 Sistema gerenciador de banco de dados

Para conseguir lidar com um ou mais bancos de dados, é necessário a utilização de um sistema gerenciador de bancos de dados (SGBD). A partir delas é possível criar, alterar e eliminar base de dados, assim como acessar e manusear os dados presentes nos mesmos. Alguns exemplos de SGBD gratuitos que vem sendo utilizados são: *Postgresql* e *MySql*.

Para lidar com cada uma dessas operações, normalmente os SGBDs utilizam a linguagem SQL, *structured query language*. Essa linguagem é baseada na álgebra rela-

cional e vem sendo utilizado largamente ao longo dos anos. Também provém internamente uma linguagem consistente matematicamente, permitindo o melhor desempenho nas ações executadas. Apesar de utilizarem a mesma linguagem, o SQL pode possuir algumas diferenças entre os diferentes SGBDS existentes (SILVEIRA, 2019).

2.5 Web services

Os *web services* são tecnologias que permitem o envio e recebimento de dados entre aplicações através da internet. Através dela é possível prover diferentes serviços para aplicações com características distintas, ou seja, sem depender da tecnologia utilizada por cada uma. Para isso os *web services* utilizam de algum formato comum para a troca dos dados, como: *XML, Json, Csv* entre outros (OPENSOFT, c2001). A utilização de *web services* trazem diferentes benefícios, como:

- Reutilização: Podem ser reutilizados em diferentes aplicações (OPENSOFT, c2001).
- Baixo custo: A comunicação pode ser feita utilizando tecnologias existentes (OPENSOFT, c2001).
- Segurança: O acesso aos dados é não feito de maneira direta no banco dados (OPENSOFT, c2001).
- Rápido desenvolvimento: Utilização de *frameworks* para o desenvolvimento (OPENSOFT, c2001).
- Integração de sistemas: A comunicação independe da tecnologia dos sistemas (OPENSOFT, c2001).

Os *web services* que seguem um conjunto de princípios presentes na arquitetura REST, *representational state transfer*, são chamadas de *web services rest*. Através dos *web services rest* é possível conseguir benefícios como: performance, escalabilidade e manutenção (ORACLE, c1995b).

Nos *web services Rest*, tanto os dados quanto as funcionalidades são considerados como recursos. O acesso a esses recursos é feito utilizando-se dos métodos do protocolo *http*, sendo eles: *get, post, put ou delete*. Esses recursos são acessados através de um identificadores único de recurso (URI) chamados de *end-point*, como por exemplo as *URLs*. O acesso a esses recursos devem ser do tipo *stateless* (sem estado), ou seja, cada acesso é independente um do outro (ORACLE, c1995b).

2.6 Spring

O Spring é o *framework* mais popular na linguagem Java e tem o objetivo de tornar o desenvolvimento das aplicações simples e rápidas. O *framework* fornece a

infraestrutura necessária para o desenvolvimento, permitindo que o desenvolvedor foque nos aspectos principais da aplicação (SPRING, c2011a).

Os diversos recursos trazidos pelo *framework* Spring estão divididos entre vinte diferentes módulos, cada um deles agrupados em uma de seis categorias distintas: *Core Container, DataAcess/Integration, Web, AOP, instrumentation* e teste (SPRING, c2011b). Alguns exemplos de funcionalidades contidas nesses diferentes módulos são:

- Injeção de dependência e inversão de controle.
- Integração com a com as camadas de acesso aos dados.
- Teste unitário e teste de integração.
- Implementação de *WebServices Restful*.
- Desenvolvimento baseado no padrão de projeto MVC.
- Gerenciamento de dependência.
- Utilização de logs.

Além das funcionalidades presentes neste *framework*, existe a ferramenta o Spring Boot que simplifica o processo de criação e configuração de um projeto Spring. Nessa ferramenta é possível adicionar os módulos do Spring e de terceiros. Além disso, é possível ter um projeto pré configurado para que o servidor *tomcat, jetty* ou *Undertow* esteja adicionado no mesmo, bastando a execução do arquivo *jar* gerado na build para funcionar (SPRING, c2011c).

2.7 Maven

O processo de build consiste em diferentes etapas para gerar os arquivos necessários para a execução da aplicação. O Maven é uma ferramenta que permite gerenciar todo ciclo de vida de build do projeto trabalhado Maven. A partir dele é possível alterar cada uma das fases do processo de *build* no projeto, sendo elas: validação, compilação, empacotamento, verificação, instalação e implantação (MAVEN, c2002a). Todas essas configurações podem ser feitas em um arquivo de configuração denominado *pom*, que é escrito na linguagem de marcação *XML* (MAVEN, c2002b).

Outra funcionalidade útil dessa ferramenta é o gerenciamento de dependências. A partir dela é possível configurar diferentes dependências para o projeto sem a necessidade de baixar e configurar cada dependência manualmente. Para isso basta adicionar um trecho de código referente a dependência no arquivo pom. Esses trechos de códigos podem ser encontrados no site do fabricante ou em repositórios como o *MVNrepository* (MAVEN, c2002b).

2.8 Hibernate e JPA

Ao trabalhar com banco de dados em uma linguagem orientada a objetos, existe a possibilidade de utilizar a técnica de mapeamento objeto relacional (*object Relational Mapper* - ORM). Ela consiste em converter objetos em entidades do banco de dados e vice-versa. Isso permite que não haja a necessidade de trabalhar com detalhes do banco de dados na aplicação, toda a manipulação de dados é feita a partir de objetos (CADU, 2011).

O Hibernate consiste de um *framework* para mapeamento objeto relacional para a linguagem Java. Ele é capaz de abstrair a camada JDBC e gerar o código SQL em tempo de execução para o banco de dados que está sendo utilizado. Dessa forma, é possível que haja troca entre banco dados diferentes sem a necessidade de alterações no código criado. Outra vantagem desse *framework* é que ele implementa as especificações do JPA (CAELUM, c2004).

O JPA (Java persistence API) é uma especificação que determina como deve ser feito o ORM na linguagem Java. Ao utilizar dessa especificação é possível obter modularidade em relação ao *framework* que é utilizado para fazer o mapeamento objeto relacional (CAELUM, c2004).

O JPA utiliza como linguagem padrão o JPQL. Esta linguagem permite realizar consultas no formato da linguagem SQL. Porém, diferentemente de uma consulta realizada direto no banco de dados, para fazer a consulta utiliza-se valores referentes aos objetos e atributos que mapeiam as tabelas do banco trabalhado (ORACLE, c1995c).

2.9 Angular

As Aplicações SPA, *single page aplication*, baseiam-se em disponibilizar o conteúdo em uma única página, sendo a página atualizada de acordo com a interação com o usuário (ANASTASIA, 2018). Esse tipo de aplicações trazem diferentes tipos de benefícios, como:

- Melhor experiência do usuário.
- Menor tempo de carregamento.
- Consomem menos recursos do servidor.
- Utilizam menor largura de banda.

O *framework* Angular é destinado à criação do *front-end* de aplicações de página única (SPA) de maneira eficiente e moderna (ANGULAR, 2010a). O seu funcionamento é baseado na utilização de estruturas chamadas de componentes. Cada componente é descrito por três arquivos com responsabilidades distintas, sendo eles implementados em *typescript, html* e *css* (ANGULAR, 2010b).

- Typescript: Contém configurações do componente e implementa o dinamismo na página.
- Html: Determina a estrutura da página.
- Css: Determinam o visual dos elementos na página html.

Cada componente é identificado de maneira única através de seu seletor, configurado no arquivo com código *typescript*. Através do seletor um componente é identificado e pode ser renderizado ao carregar uma página. Dessa forma é possível que componentes sejam utilizados em outros componentes, sendo denominados componentes filhos aqueles que são utilizados e componentes país aqueles que utilizam.

Ao utilizar o Angular é possível usufruir de diferentes funcionalidades que facilitam o processo de desenvolvimento da aplicação, alguns exemplos são:

- Vincular valores de propriedades da classe typescript com o conteúdo html.
- Comunicação entre componentes pais e filhos.
- Fornece diretivas para lidar com condicionais e loops no conteúdo html.
- Possui diferentes componentes já criados que oferecem funcionalidades diversas.
- Injeção de dependência.

Além destas, possui diferentes bibliotecas para resolução de problemas diversos, como por exemplo o *http cliente*, que permite a comunicação com o servidor através dos métodos *http* (ANGULAR, 2010b).

O trabalho com projetos em Angular pode ser beneficiado pela utilização da ferramenta Angular CLI. Ela consiste em uma ferramenta na qual o usuário fornece comandos através da interface de linha de texto (CLI) para conseguir gerenciar o projeto Angular. A partir dela é possível criar o projeto, criar componentes, realizar a *build* no projeto e outras ferramentas para o gerenciamento do projeto. Além das funcionalidades citadas, ela vem com um servidor e pode ser utilizado para colocar a aplicação em execução (ANGULAR, 2010b).

2.10 Padrões de projeto

Os padrões de projeto tem o objetivo de fornecer soluções eficientes para problemas conhecidos e recorrentes na orientação a objetos. Cada padrão de projeto tem um nome, um problema que tenta resolver, a solução desse problema e as consequências da utilização desse padrão.

Nos subtópicos seguintes será explicado sobre alguns padrões de projetos que foram utilizados neste projeto.

2.10.1 DAO

O padrão de projetos DAO tem o objetivo de dividir as responsabilidades do acesso aos dados das demais partes da aplicação. Utilizando este padrão é possível alterar a forma que os dados são acessados sem a necessidade de alterar outras partes da aplicação. Isso beneficia o projeto deixando o software mais coeso e menos acoplado, facilitando a manutenção (DALEPIANE, 2014).

Esse padrão de projeto se baseia na criação de uma classe do tipo DAO, que permite criar objetos que ficam responsáveis em se comunicar com os SGBDs e disponibilizar diferentes métodos para realizar operações de CRUD (DALEPIANE, 2014).

Esse padrão de projeto pode ser implementado juntamente com a técnica de mapeamento objeto-relacional (ORM), que consiste em abstrair as tabelas do banco através de objetos na orientação a objetos. Para isso deve-se criar uma classe de domínio, uma interface DAO e uma classe DAO (BAELDUNG, c2020), cada um desses elementos possui as seguintes responsabilidades:

- Classe de domínio: Cada objeto dessa classe representa um registro de uma tabela no banco de dados. A classe é composta por atributos referentes às colunas da tabela e também de métodos para acessá-los.
- Interface DAO: Determina especificações de métodos de acessos aos dados. Ela permite desacoplar a lógica de acesso aos dados da implementação. Dessa forma, qualquer classe que implemente a interface pode ser utilizada para realizar o acesso.
- Classe DAO: Implementa os métodos determinados pela interface DAO.

2.10.2 Injeção de dependências

A injeção de dependência é um padrão de projeto que inverte o controle a respeito da criação dos objetos. Nesse padrão, quando um objeto necessita de outros, os objetos necessários são criados externamente e passados para ela de diferentes maneiras. Esse padrão de projeto torna o código menos acoplado e mais coeso, melhorando a manutenibilidade (GAMA, 2010).

As diferentes dependências de um objeto podem ser injetadas de três diferentes formas, são elas:

- Pelo construtor: As dependências são inseridas ao criar o objeto .
- Pela propriedade: As dependências são inseridas através dos setters do objeto.
- Por um *framework*: O *framework* injeta as depêndencias no objeto.

2.10.3 Template method

O *template method* é um padrão de projeto que permite que uma classe implemente métodos com um fluxo já pré-determinados de execução, porém, que seja necessário com que a classe filha determine como vai ser a implementação de uma ou mais etapa desse fluxo. A vantagem desse padrão é o reaproveitamento de código, menor acoplamento e maior coesão (REFACTORING.GURU, c2014).

Esse padrão se baseia em criar uma classe abstrata que implementa diferentes métodos, sendo que alguns deles utilizam métodos ainda não implementados. Esses métodos abstratos ficam para serem implementados pelas classes que herdam da classe abstrata.

3 DESENVOLVIMENTO DO DASHBOARD

Neste capítulo são tratados a respeito dos diferentes aspectos a respeito do desenvolvimento do *dashboard*. Na seção 3.1 é explicado sobre qual a arquitetura do sistema desenvolvido, nas seções 3.2, 3.3 e 3.4 são explicados sobre o que foi realizado em cada uma das camadas que compõem o *dashboard*, sendo elas a camada de dados, camada de aplicativo e camada de apresentação. Por fim, na seção 3.5 é explicado sobre como foi feita a implantação da aplicação para ser executada na fazenda.

3.1 Arquitetura do *dashboard*

O sistema desenvolvido utiliza de uma arquitetura em três camadas, permitindo manter a separação lógica e física entre as diferentes partes da aplicação. A camada de apresentação consiste na camada em que os dados são organizados no formato de gráficos/tabelas para o usuário e que permite com que o mesmo possa interagir com o sistema. A camada de aplicativo consiste na implementação das regras de negócio existentes, bem como na implementação do conjunto de padrões e tecnologia de acesso aos dados. A camada de dados é composta por um banco de dados, nele estão presentes os dados coletados e processados a respeito das ordenhas realizadas na fazenda.

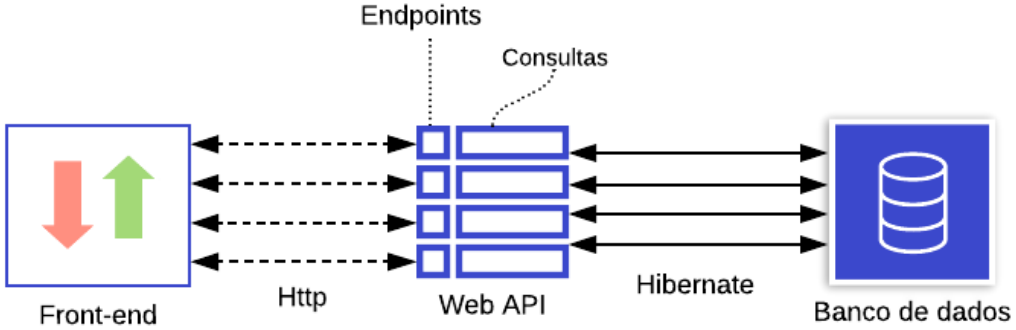


Figura 3.1: *Representação da arquitetura do sistema.*

A comunicação entre as camadas de visualização e de aplicativo ocorre através da utilização de *Web Services Rest*. Para isso, a camada de apresentação realiza consultas na camada de aplicativo através dos *end-points* disponibilizados.

3.2 Camada de dados

A camada de dados consiste em um banco de dados gerenciado através do SGBD *postgresql*. Neste trabalho os dados utilizados provêm de duas tabelas distintas, *producao* e *producaoOrdenha*, conforme mostrado na fig 3.2.

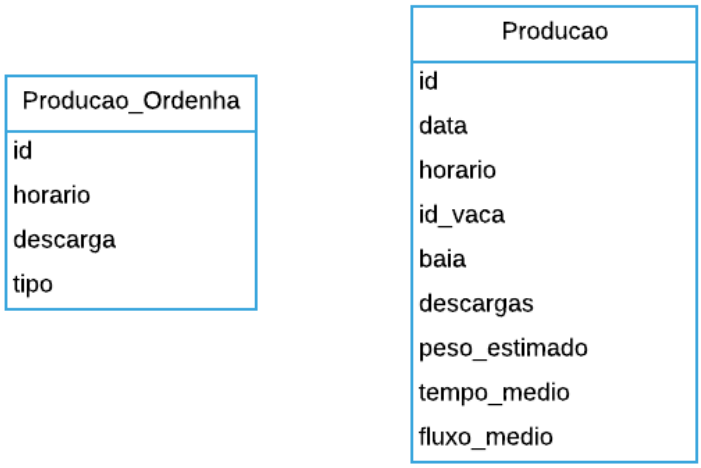


Figura 3.2: *Tabelas do banco de dados.*

A tabela *producaoOrdenha* é referente a cada balde de leite que foram retirados durante as diferentes ordenhas. Essas informações estão contidas nas colunas: *baia*, *horario*, *descarga* e *tipo*.

- baia: Local onde é tirado o leite da vaca.
- horario: Quando o balde de leite terminou de ser enchido.
- descarga: Representa o enésimo balde de leite enchido em uma ordenha.
- tipo: Indica se o registro na tabela é um balde de leite enchido ou a finalização da ordenha. Quando um, representa um balde de leite, quando dois, representa a finalização.

Durante as ordenhas os dados dessa tabela são adicionados em tempo real por equipamentos pertencentes a fazenda. Estes foram desenvolvidos por outra equipe de trabalho e seu funcionamento não será abordado nesse trabalho.

A tabela *producao* armazena informações sobre a ordenha como um todo de uma vaca, ou seja, leva em consideração todos os baldes de leite que foram enchidos em uma ordenha. Ela guarda informações como: data, horario, id_vaca, baia, descargas, peso_estimado, tempo_medio, fluxo_medio.

- data: Data da ordenha.
- horario: Horario da ordenha.
- id_vaca: Vaca ordenhada.

- baia: Onde ocorreu.
- descargas: Quantidade de baldes enchidos de leite.
- peso_estimado: Estimativa do peso.
- tempo_medio: Tempo médio para encher os baldes de leite.
- fluxo_medio: Fluxo médio para encher os baldes de leite.

3.2.1 População da tabela *producao*

Para preencher os dados da tabela *producao* em tempo real, foi criado uma *trigger* na linguagem *plpgsql* do *postgresql*. A *trigger* está associado a tabela *producaoOrdenha* e fica responsável por:

1. Identificar o momento que acaba a ordenha de uma vaca.
2. Identificar os baldes pertencentes a ordenha finalizada.
3. Realizar operações para calcular os valores que devem ser inseridos.
4. Inserir os resultados na tabela *producao*.

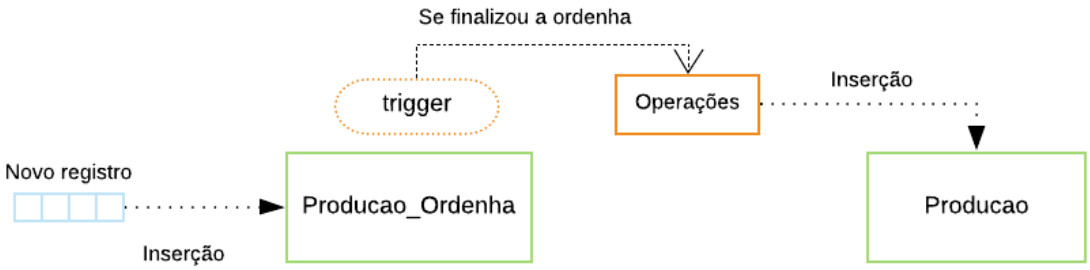


Figura 3.3: *Representação da trigger.*

A cada novo registro inserido em *producaoOrdenha*, a *trigger* verifica o valor da campo *tipo* com o objetivo de identificar a finalização de uma ordenha. Quando isso ocorre, a *trigger* leva em consideração os valores dos campos *descarga*, *horario* e *baia* do último registro inserido para realizar uma consulta em SQL que identifica todos os baldes de leite pertencente aquela ordenha. Através dos registros de cada balde, são realizadas operações para calcular os valores de *descargas*, *tempo_medio*, *peso_estimado* e *fluxo_medio*. Após essas etapas, é inserido na tabela *producao* um novo elemento referente a ordenha de uma vaca.

3.3 Camada de aplicativo

O *back-end* trata-se de uma *Web API*, ou seja, uma API que disponibiliza os dados para serem acessados através do protocolo *http*. Ele foi desenvolvido na *linguagem*

Java e que utiliza dos *framework* Spring e Hibernate no seu funcionamento. O projeto foi criado através da ferramenta Spring Boot e já possui um servidor configurado internamente. A *build* do projeto e as dependências são gerenciadas através da ferramenta Maven.

Nos próximos tópicos serão explicados diferentes aspectos do *back-end*. No tópico 3.3.1 é explicado sobre como é feito acesso aos dados e no tópico 3.3.2 é falado da disponibilização dos mesmos através de *Web services Rest*.

3.3.1 Acesso aos dados

Para lidar com o acesso aos dados, foi utilizado o padrão de projetos DAO com o objetivo de separar a lógica da consulta do resto da aplicação. Além disso, foi realizado o ORM seguindo as especificações do JPA através do *framework* Hibernate. Para isso foram criadas as classes, *Producao*, *ProducaoOrdenha*, *ProducaoDAOImpl*, *ProducaoOrdenhaDAOImpl* e as interfaces *ProducaoDAO*, *ProducaoOrdenhaDAO*, organizados de acordo com a figura 3.4.

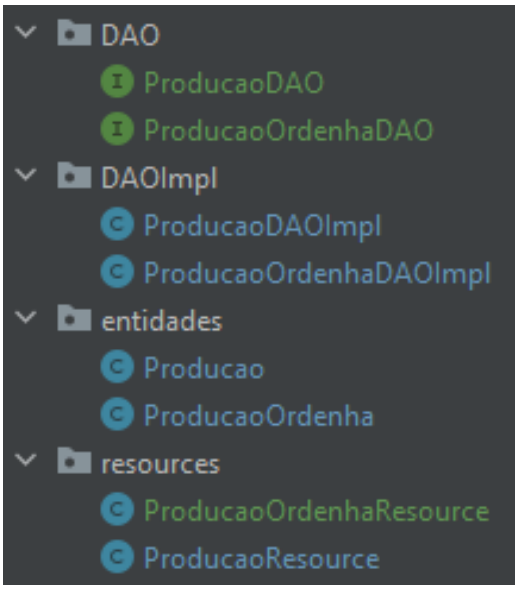


Figura 3.4: *Classes e interfacesses criadas*

As classes *Producao* e *ProducaoOrdenha* foram criadas com objetivo de fazer o mapeamento objeto-relacional e ficam responsáveis por representar as tabelas no banco de dados. Elas contêm como atributos os mesmos campos das tabelas que estão representando e também os métodos para acessá-los.

A interface *ProducaoDAO* determina as consultas que devem ser realizadas na tabela *producao*. Existem um total de nove métodos para realizar as diferentes consultas no banco. De maneira geral, as consultas devem permitir:

- Obter números de cada uma das baias e vacas presentes na fazenda.

- Obter dados a respeito de uma variável escolhida em função da passagem do tempo. Os dados podem ser referentes a uma baia ou a uma vaca escolhida.
- Obter dados a respeito de uma variável escolhida em função das vacas ou das baias.

A interface *ProducaoOrdenhaDAO* determina as consultas que devem ser realizadas na tabela *producaoOrdenha*. Elas determinam quatro métodos que devem permitir obter os dados a respeito de uma variável escolhida em função de cada balde de leite retirado. Os dados podem ser referentes a uma baia ou a uma vaca escolhida.

Em ambas as interfaces, os métodos possuem diferentes argumentos que determinam como vão ser realizadas as consultas. Os argumentos têm significados distintos e determinam:

- Se os dados são referentes a uma vaca ou a uma baia.
- Qual o número da baia ou da vaca analisada.
- Qual a variável analisada.
- Qual o nível de granularidade em relação ao tempo.
- Qual a operação utilizada no agrupamento dos dados.
- Qual o intervalo de tempo analisado.

Os métodos especificados pelas interfaces *producaoDAO* e *producaoOrdenhaDAO* são implementadas nas classes *producaoDAOImpl* e *producaoOrdenhaDAOImpl*. As consultas foram implementadas na linguagem de consulta JQPL da API JPA e utilizam das classes *Producao* e *ProducaoOrdenha* para fazer o mapeamento objeto relacional.

As consultas em JPQL são geradas dinamicamente através dos valores passados aos métodos que realizam as consultas, como exemplificado na figura 3.6 . Para isso é utilizado da concatenação de strings e também do método *setParameter* do JPA. O motivo de utilizar a concatenação de *strings* é para criar consultas genéricas e que abranjam a maior quantidade de situações possíveis. Sem a utilização da concatenação de *strings* seria necessário criar uma quantidade enorme de funções, tornando inviável o desenvolvimento e manutenção do sistema.

Para resolver o problema de *SQL injection* na concatenação de *strings*, os valores recebidos do usuário são mapeados para outros valores pré-determinados, ou seja, os valores das *strings* concatenadas não vem diretamente do usuário. Logo, caso alguém mal intencionado tente colocar um código em alguns dos campos, esse valor não será mapeado, gerando uma exceção. Para lidar com o mapeamento foi criado uma classe de nome *Mapeador* que contém métodos que fazem o mapeamento dos valores recebidos em valores possíveis, esse mapeamento é feito através da utilização da estrutura de condição *switch*, Figura 3.5.

```java
public static String converterOperacao(String operacao){
    String comando = "";
    switch(operacao) {
        case "soma":
            comando = "SUM";
            break;
        case "media":
            comando = "AVG";
            break;
        case "maximo":
            comando = "COUNT";
            break;
        case "minimo":
            comando = "MAX";
            break;
    }

    return comando;
}
```

Figura 3.5: *Exemplo de função que converte uma string passado pelo cliente em diferentes opções que irão ser usados na consulta JPQL*

```java
public List<variavelValor> findByIdVacaDia(String filtro, Integer id,
                                           String variavel, Date dataInicial,
                                           Date dataFinal, String operacao){

    String comando = Mapeador.converterOperacao(operacao);
    variavel = Mapeador.converterVariavel(variavel);
    filtro = Mapeador.converterFiltro(filtro);

    String jpql =
        "select new com.ifg.dashboard.variavelValor" +
        "(day(p.data),month(p.data),year(p.data),"+comando+"(p."+variavel+ ")) " +
        "from Producao p "+
            "where p."+filtro+ " = :id and p.data >= :dataInicial and p.data <= :dataFinal "+
                "group by p.data " +
                    "order by p.data";

    TypedQuery<variavelValor> query = em.createQuery(jpql, variavelValor.class);

    query.setParameter( name: "id", id);
    query.setParameter( name: "dataInicial", dataInicial);
    query.setParameter( name: "dataFinal", dataFinal);

    return query.getResultList();
}
```

Figura 3.6: *Exemplo de método presente na classe "ProducaoRepository".*

3.3.2 Web services

As classes *ProducaoResource* e *ProducaoOrdenhaResource* foram criadas para disponibilizar as consultas ao banco de dados através de *Web Services Rest*. Para isso, ambas as classes recebem a anotação *@RestController* do Spring, que identifica as

classes como controladores e determina que os retornos dos métodos sejam serializados e enviados no corpo da requisição *http* no formato *json*.

Nos métodos dessas classes, utiliza-se da anotação *@RequestMapping* para determinar o formato da URL que será utilizado para invocar cada um dos métodos. Através desse mesma anotação é possível informar os parâmetros que são passados na URL e que são utilizados nos métodos.

Para realizar a consulta ao banco de dados, os objetos DAO são injetados nos objetos das classes *ProducaoOrdenhaResource* e *ProducaoResource*. Os métodos presentes nessas classes, quando invocados, recebem os parâmetros passados pela URL e os utilizam para fazer a consulta através dos objetos DAO que foram injetados. O resultado das consultas são retornados ao requisitante no corpo da mensagem. A figura abaixo representa o formato de alguns dos métodos presentes nessas classes:

```
@RequestMapping
(value ="/vacaPeriodo/{filtro}/{id}/{variavel}/{dataInicial}/{dataFinal}/{turno}/{operacao}")
public ResponseEntity<List<variavelValor>> findByIdVacaPeriodo(
        @PathVariable("filtro") String filtro,
        @PathVariable("id") Integer id,
        @PathVariable("variavel") String variavel,
        @PathVariable("dataInicial") @DateTimeFormat(pattern = "yyyy-MM-dd") Date dataInicial,
        @PathVariable("dataFinal") @DateTimeFormat(pattern = "yyyy-MM-dd") Date dataFinal ,
        @PathVariable("turno") String turno,
        @PathVariable("operacao") String operacao){


    List<variavelValor> producao= service.findByIdVacaPeriodo(filtro, id, variavel,dataInicial,
                                                              dataFinal,turno,operacao);

    return ResponseEntity.ok().body(producao);
}
```

Figura 3.7: *Exemplo de método da classe ProducaoResouce.*

3.4 Camada de apresentação

O *front-end* da aplicação foi implementado utilizando o *framework* Angular e também diferentes bibliotecas, como: *ng2-charts*, *html2canvas*, *resizedEvent* e *maTable-ExporterModule*.

Nos próximos subtópicos são apresentados diferentes aspectos sobre a implementação da camada de apresentação. Na seção 3.4.1 será explicado como os dados são acessados no *back-end*, na seção 3.4.2 será explicado como esses diferentes dados consultados são estruturados para serem utilizados, na seção 3.4.3 é tratado sobre como são encontrado anomalias, na seção 3.4.4 será explicado como os dados consultados são representados no formato de gráficos e de tabelas, na seção 3.4.5 será explicado sobre diferentes aspectos a respeito da implementação dos painéis.

3.4.1 Acesso aos dados

Para a obtenção dos dados que são utilizados *front-end*, foi criada a classe de serviço *GetAPIService*, responsável por requisitar consultas ao banco de dados através dos *end-points* disponibilizados pelo *back-end*. Toda a comunicação entre o *back-end* e o *front-end* é realizada a partir de objetos dessa classe. Quando objetos de outras classes precisam realizar consultas, é injetado um objeto do tipo *GetAPIService* nos mesmos.

Essa classe possui o atributo *baseurl* do tipo *string*, que armazena a URL base do *back-end*. A partir da URL base, dos diferentes parâmetros presentes nos métodos e utilizando da interpolação de *strings* do *typescript*, é formada a URL responsável por realizar as consultas no *back-end*. Tendo as URLs, é realizado uma requisição *http* através do método *get*, para isso é utilizado o módulo *HttpClient* do Angular. Um exemplo de como é feita a consulta ao *back-end* é mostrado na Figura 3.8.

```
getValoresFiltros(filtro: string): Observable<any> {
  return this.http.get(`${this.baseurl}/distinct/${this.dicionario[filtro]}`);
}
```

Figura 3.8: *Exemplo de método presente em GetAPIService.*

3.4.2 Estruturação dos dados

No *front-end* existe a necessidade de trabalhar com uma grande quantidade de informações de diferentes tipos. Por esse motivo existe a necessidade de estruturar adequadamente os dados. Para isso foram criadas as classes: *conjuntoElemento*, *Elemento* e *Variavel*. A Figura 3.9 representa o relacionamento entre cada uma dessas classes.

- Variavel: Armazena os dados que foram consultados no banco de dados e outras informações sobre a variavel.
- Elemento: Armazenam diferentes objetos do tipo Variavel e diferentes informações sobre o Elemento. Os objetos dessa classe representam vacas ou baias.
- ConjuntoElemento: Armazenam múltiplos elementos. É utilizado para lidar com um conjunto de vacas ou um conjunto de baias.

Cada uma dessas classes possuem diferentes métodos que permitem adicionar, ordenar, deletar e atualizar os seus atributos.

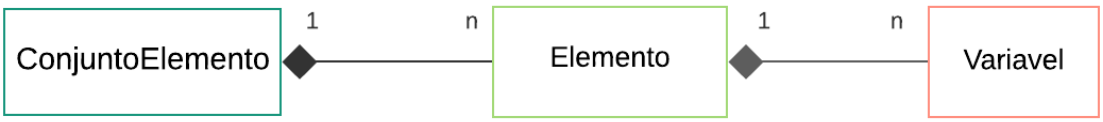


Figura 3.9: *Diagrama de classes representando as classes: Variavel, Elemento e ConjuntoElemento.*

3.4.3 Identificação de anomalias

Em diferentes partes do *dashboard* existe a necessidade de identificar anomalias de um conjunto de dados. Para realizar essa tarefa foi criado uma classe abstrata de nome *VerificadorAnomalias* e duas classes concretas de nome *VerificadorAnomaliasQuartis* e *VerificadorAnomaliasDesvioPadrao*, conforme a figura abaixo.

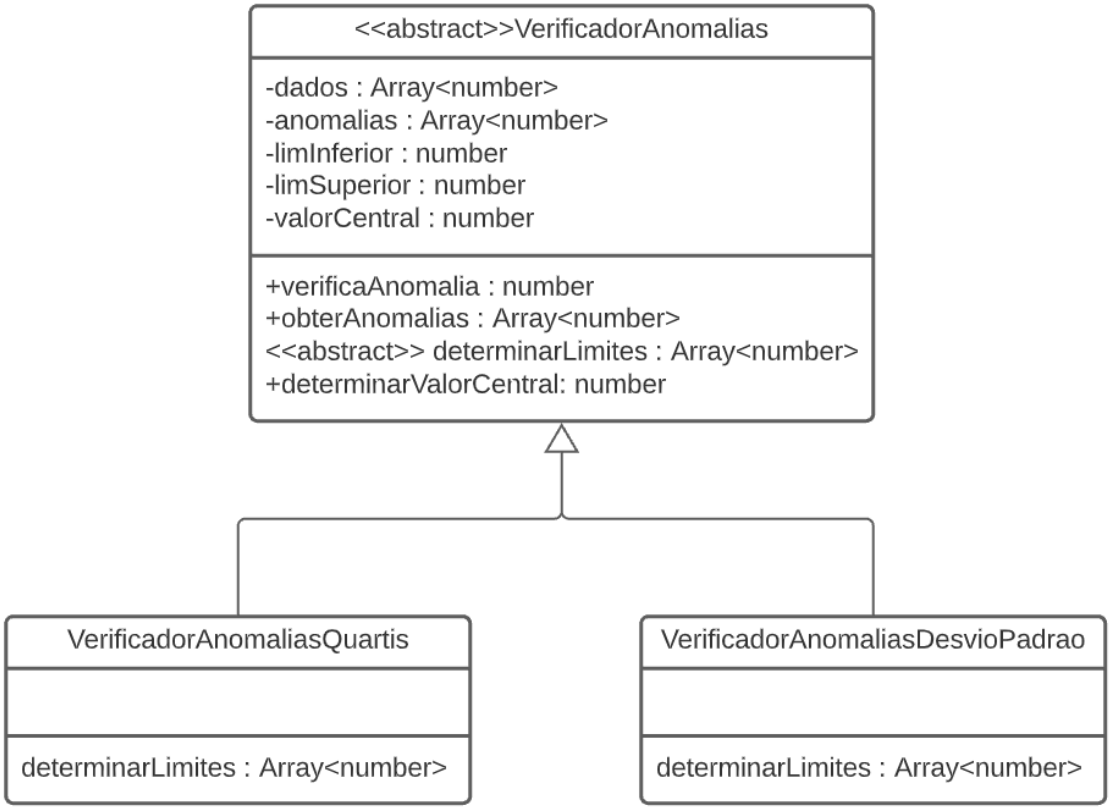


Figura 3.10: *Diagrama de classes representando as classes VerificadorAnomalias, VerificadorAnomaliasDesvioPadrao e VerificadorAnomaliasQuartis*

A implementação segue o padrão de projetos *template method*, onde a classe *VerificadorAnomalias* determina métodos comuns e também um fluxo geral para realizar as diferentes etapas na identificação de anomalias. Essa classe deixa um dos métodos para as classes filhas implementarem. Esse método a ser implementa é responsável por identificar o maior e o menor valor tolerável para os dados, denominados de limite superior e limite inferior, dados fora desses limites são considerados anomalias.

A classe *VerificadorAnomaliasQuartis* herda de *VerificadorAnomalias* e implementa o método abstrato da mesma. Essa classe determina se um dado é ou não anomalia utilizando-se das medidas estatísticas: primeiro quartil e terceiro quartil. O cálculo dos limites é realizado da seguinte maneira:

Q1 : Primeiro quartil

Q3 : Terceiro quartil

Q3 - Q1 : IQR

Q1 - 1.5 x IQR : Limite inferior

Q3 + 1.5 x IQR : Limite superior

A classe *VerificadorAnomaliasDesvioPadrao* também herda de *VerificadorAnomalias* e implementa o método abstrato da mesma. Ela utiliza das medidas estatísticas: média e desvio padrão. O cálculo utilizado para isso é apresentado a seguir.

$\bar{x}$: Média

α : Desvio padrão

$\alpha + \bar{x}$: Limite superior

α - $\bar{x}$: Limite inferior

O cálculo das medidas estatísticas utilizadas nas classes *VerificadorAnomaliasDesvioPadrao* e *VerificadorAnomaliasQuartis* é feito através de uma classe estática *MedidasEstatisticas*. Essa classe contém diferentes métodos responsáveis por calcular a média, variância, desvio padrão, mediana, primeiro quartil e terceiro quartil.

3.4.4 Elementos de visualização

No *dashboard* os dados podem ser representados graficamente ou no formato de tabelas. Para isso foram criados componentes específicos para lidar com essas duas situações. Esses componentes são utilizados pelos diferentes painéis para gerar diferentes formatos de gráficos e diferentes tabelas.

3.4.4.1 Gráficos

O componente *app-grafico* foi criado para lidar com todas as questões referentes à geração de gráficos. A sua implementação é feita utilizando da biblioteca *ng2-charts*, que permite trabalhar com representações gráficas de diferentes tipos. Através desse componente é possível gerar gráficos de barras ou linhas, sendo a escolha feita através do campo *tipoGrafico* que deve ser passado pelo componente pai ao utilizar o componente *app-grafico*.

Além do campo *tipoGrafico*, esse componente recebe os dados que devem ser disponibilizados através do campo *vetorComDados*. Dos dados recebidos, utiliza-se de

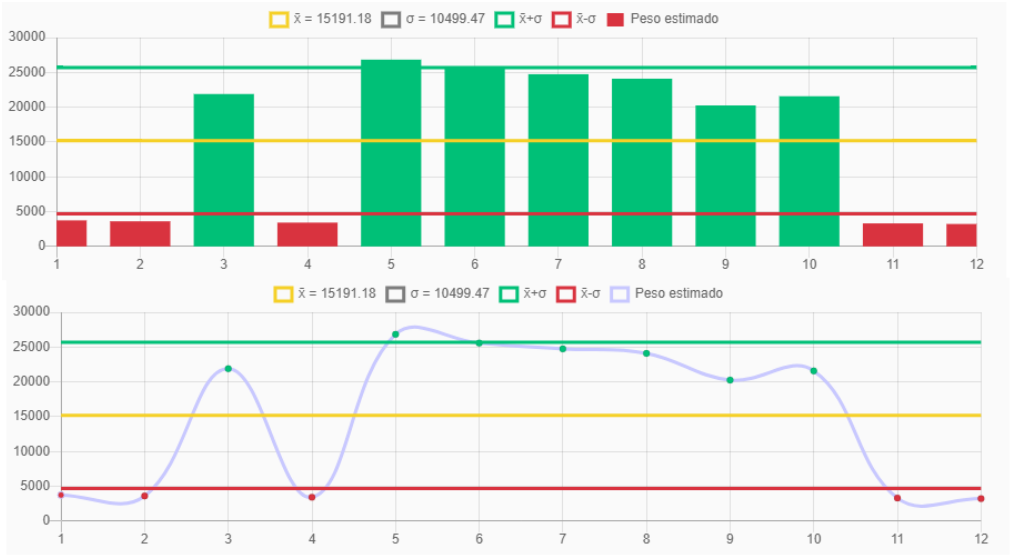


Figura 3.11: *Exemplo de gráfico de barras e de colunas gerado pelo componente app-grafico a partir de um mesmo conjunto de dados.*

um objeto do tipo *VerificadorAnomaliasDesvioPadrao* para encontrar os limites para um dado ser considerado anômalo ou não.

Esses limites são representados nos gráficos como retas e servem para classificar os dados. A classificação leva em conta se os dados beneficiam ou não a empresa à medida que crescem. Essa informação é passada para componente *app-grafico* através do atributo *maiorMelhor*, do tipo booleano.

- Se *maiorMelhor* for verdadeiro: Dados acima da média aparecem na cor verde, dados abaixo da média e acima do limite inferior, aparecem da cor amarela e dados abaixo do limite inferior aparecem da cor vermelha.
- Se *maiorMelhor* for falso : Dados abaixo da média aparecem da cor verde, acima da média e abaixo do limite superior, da cor amarela e acima do limite superior da cor vermelha.

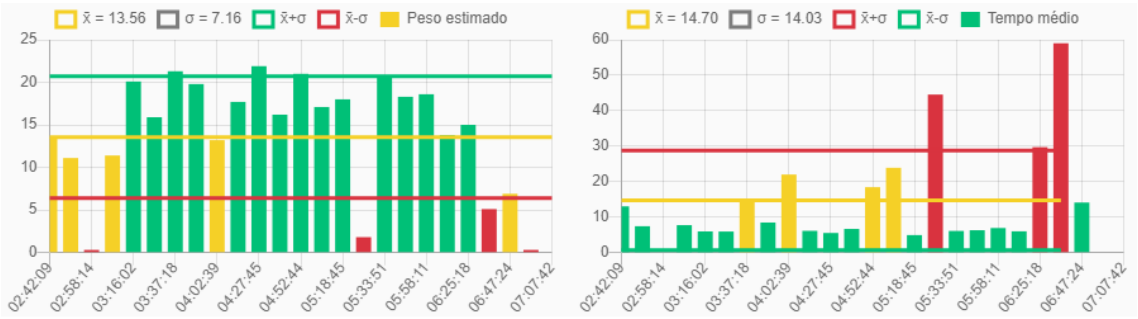


Figura 3.12: *Exemplicação de classificação através das cores. No gráfico da direita maiorMelhor é verdadeiro e no da direita maiorMelhor é falso.*

Essa classificação é útil à medida que ajuda a identificar visualmente possíveis anomalias nos dados, facilitando a indicação de problemas nos equipamentos ou doenças nos animais. Também ajuda a identificar quais os melhores animais presentes na fazenda e quais os piores.

A legenda do gráfico, presente na parte superior, indica a cor de cada elemento que está sendo representado no gráfico. Nela também é informado o valor numérico da média e do desvio padrão. A legenda pode ou não ser mostrada de acordo com o atributo *mostrarLegenda*, que é informado pelo componente pai ao componente app-grafico.

Figura 3.13: *Gráficos com e sem legenda*

3.4.4.2 Tabelas

O componente *app-tabela* é responsável pela disponibilização dos dados em tabelas e todas as funcionalidades presentes na mesma. Esse componente é implementado utilizando do componente *MatTableModule* do Angular Material. A imagem abaixo ilustra uma tabela gerada nesse componente.

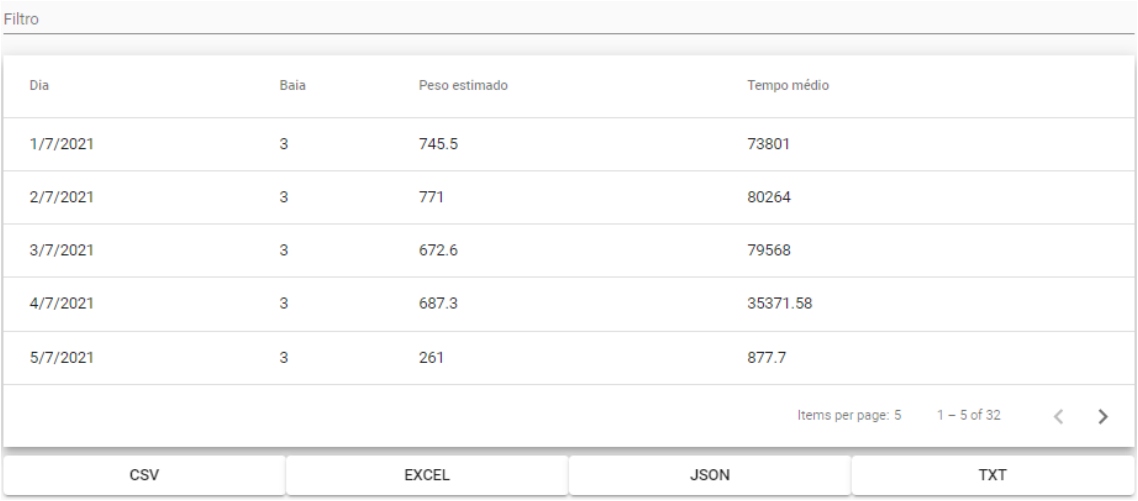


Figura 3.14: *Exemplo de tabela gerado pelo component app-tabela.*

Esse componente recebe os dados do componente pai a partir de um objeto do tipo *Elemento*. A partir dos dados presentes nesse objeto, é possível gerar nas tabelas diferentes colunas para cada uma das variáveis que estão sendo analisadas.

A tabela gerada possui a funcionalidade de filtragem, permitindo encontrar dados específicos de maneira rápida. Nela também está presente a funcionalidade de ordenação dos dados em ordem crescente ou decrescente. Essa funcionalidade é obtida através da utilização do módulo *MatSort* do Angular. A ordenação pode ser realizada para cada um dos campos presentes na tabela, dessa forma é possível encontrar os maiores e menores valores para cada uma das variáveis, auxiliando na tomada de decisão.

Para melhor organizar os dados no *dashboard*, as tabelas contam com o recurso de paginação, obtido através da utilização do módulo *mat-paginator*. A quantidade de linhas que é mostrado na tabela é informado pelo componente pai através do atributo *tamanhoPagina*.

As tabelas também possuem a funcionalidade de exportar os dados nos formatos csv, json, excel e txt. Para isso, existem quatro botões logo abaixo da tabela que permitem o usuário fazer o *download*. A exportação dos dados é feita através da utilização do módulo *MatTableExporterModule*.

3.4.5 Painéis

No *dashboard* existem três painéis: painel temporal, painel comparativo e painel de anomalias. Cada um destes são responsáveis por permitir aos usuários visualizar os dados de diferentes formas e dessa maneira realizar análises distintas. Todos os três são implementados em componentes separados e recebem o respectivos nomes, *app-painel-temporal*, *app-painel-comparativo* e *app-painel-anomalias*. Apesar de ter o funcionamento e implementação distintos, todos eles são responsáveis pelas seguintes tarefas:

- Receber entradas do usuário.
- Obter dados das vacas e baias através de objetos do tipo *GetAPIService*.
- Organizar os dados em objetos do tipo: *ConjuntoElemento*, *Elemento* e *Variavel* de maneira que fique adequado para a visualização.
- Organizar e mostrar os dados através dos componentes *app-grafico* e *app-tabela*.

3.4.5.1 Entradas dos usuários

Em cada um dos painéis estão contidos diferentes campos que permitem com que o usuário possa interagir com o *dashboard* e especificar o que ele deseja fazer. Na implementação desses campos foram utilizados diferentes componentes do Angular Material. Os componentes usados e suas funcionalidades são:

- *mat-autocomplete* : Campos de preenchimento automático.
- *mat-datapicker* : Campos para a seleção de datas.
- *mat-radio-button* : Campos para a seleção de opções pré-determinadas.
- *mat-button* : Permite a criação e utilização de botões.

3.4.5.2 Geração dos gráficos e das tabelas

Nas classes dos componentes *app-painel-temporal*, *app-painel-comparativo* e *app-painel-anomalias* estão contidos diferentes atributos e métodos relacionados ao visual e funcionamento do *dashboard*. À medida que o usuário interage com o *dashboard*, através dos campos disponibilizados, os diferentes métodos são ativados e alteram o estado dos atributos. Esses atributos que foram modificados são acessados pelas *templates* dos componentes através da funcionalidade de *data binding* fornecida pelo *framework* Angular.

Uma vez tendo acesso aos valores dessas diferentes variáveis nas *templates*, é utilizado das diretivas **ngFor* e **ngIf* para conseguir percorrer os dados presentes nos componentes e também decidir o que deve ou não ser disponibilizado de acordo com o estado atual do painel, Figura 3.15. Nesse processo são gerados instâncias dos componentes *app-grafico* e do *app-tabela*. Estas recebem dos componentes *app-painel-temporal*, *app-painel-comparativo* ou *app-painel-anomalias* todos os dados necessários para o seu funcionamento adequado através do *data-binding*, Figura 3.15.

```
<div *ngIf="tipoVisualizacao == 'graficos'" id="graficos">
  <div id="tudo">
    <div *ngFor="let elemento of this.dadosAtual.listaElemento; index as i"
         style="display: inline-block;"
         [style.width]=qtdGraficoExterno>
      <div [id]=elemento.id>
        <div class="graficoTitulo">
          {{agrupamento}} : {{elemento.id}}
        </div>
        <ng-container *ngFor="let variavel of elemento.variaveis">
          <div style="display: inline-block; "
               [style.width]=qtdGraficoInterno >
            <div class="graficoVariavelDiscretizacao" >
              {{variavel.nomeY}} - {{discretizacao}}
            </div>
            <div *ngIf="(discretizacao != 'Período' && discretizacao != 'Horário'
                         && discretizacao != 'Descarga');else elseBlock ">
              <app-grafico
                [escolherVariavel]=variavel.nomeY
                [titulo]=discretizacao
                [vetorComDados]=variavel.dados
                [tipoGrafico]=tipoGrafico
                [legenda]=legenda
                [maiorMelhor]=variavel.maiorMelhor>
              </app-grafico-agrupado>
            </div>
            <ng-template #elseBlock>
              <app-grafico-madrugada-manha-tarde
                [escolherVariavel]=variavel.nomeY
                titulo="Madrugada"
                [vetorComDados]=variavel.dados
                [tipoGrafico]=tipoGrafico
                [legenda]=legenda
                [maiorMelhor]=variavel.maiorMelhor>
              </app-grafico-madrugada-manha-tarde>
            </ng-template>
          </div>
        </ng-container>
      </div>
```

Figura 3.15: *Parte do código html presente no template do componente app-painel-temporal utilizado para gerar o visual dos gráficos*

3.4.5.3 *Download* dos gráficos e das tabelas

Em cada um dos painéis existem opções para fazer o *download* de imagens dos dados gráficos e das tabelas que foram gerados no *dashboard* no formato *png*. Essa funcionalidade foi implementada utilizando-se da biblioteca *html2canvas*, que permite obter uma captura de tela de algum elemento *html* que está sendo representado na página, bastando para isso passar para uma função o atributo *id* do elemento *html* que se deseja

fazer o *download*.

Dessa forma, para que fosse possível baixar o visual de cada um dos gráficos ou tabelas individualmente, foram setados os atributos *id* de cada um desses elementos no momento da sua criação através da funcionalidade de *data binding* do Angular. Além disso, para baixar o visual de todos os gráficos ou tabelas simultaneamente, é necessário somente capturar a imagem do elemento *html* que é pai de todos os gráficos ou de todas as tabelas.

3.4.5.4 Tamanho dos elementos

No *dashboard* existem campos que determinam o tamanho dos elementos que são mostrados na tela, permitindo obter diferentes configurações de como os dados são mostrados. Essa funcionalidade se baseia em alterar as propriedades do *css* dos diferentes elementos gerados no *dashboard*. Para isso são alterados os valores do atributo *style* das diretivas referentes aos componentes *app-tabela* e *app-graficos*, especificando a propriedade *width* com diferentes valores. Os valores são passados às diretivas através da funcionalidade de *data binding* fornecida pelo *framework* Angular, Figura 3.15.

3.4.5.5 Responsividade

A responsividade dos diferentes painéis é obtida através da utilização da ferramenta de *media queries* pertencente ao *css*, Figura 3.16. Essa ferramenta permite determinar novos valores para os atributos do *css* dos elementos para diferentes tamanhos de telas. Dessa forma é possível ter o *dashboard* apresentando o visual adequado para diferentes aparelhos.

Além da utilização das *media queries*, foi utilizado o módulo *ResizedEvent*, que permite identificar quando o tamanho de algum elemento *html* é alterado. A vantagem de utilizar essa ferramenta é a possibilidade de alterar o visual dos gráficos e tabelas não somente baseado no tamanho da tela, mas também em relação ao tamanho disponível para os elementos *html* ocuparem no *dashboard*. Essa funcionalidade se torna muito útil pelo fato do *dashboard* possuir campos que alteram o próprio tamanho dos gráficos e tabelas que estão sendo disponibilizados para visualização.

```
@media only screen and (max-width: 1000px) {
    .menu{
        padding-left:5px;
        padding-right:5px;
        height: 100%;
    }

    #menu-direita{
        width: 100px;
    }

    #menu-esquerda{
        width: 140px;
    }

    .tabelaPeriodo{
        width: 100%;
    }
}
```

Figura 3.16: *Exemplo de media query para adequar o visual dos painéis para o tamanho de tela de 1000px*

.

3.5 Implantação

O *dashboard* foi configurado para ser utilizado em ambiente de produção na fazenda. Para isso foi necessário servir as aplicações do *front-end* e do *back-end*. O *back-end*, por ser um projeto Spring Boot, já possui servidor interno, dessa forma foi necessário somente gerar o arquivo *jar* através do comando install do Maven, e executá-lo através do *Java JRE*. O projeto do *front-end* foi executado através da ferramenta Angular CLI.

Para inicializar ambas as aplicações de maneira simples, foi criado um *script* feito em *batch*. Esse *script* é executado sempre ao inicializar a máquina da fazenda, tornando o processo de execução do *dashboard* automático. A execução automática desse *script* foi feita através do agendador de tarefas do Windows. A figura abaixo mostra o *script* utilizado. Os campos entre colchetes representam o caminho dos diretórios do arquivo *jar* gerado pelo *back-end* e também o caminho do projeto do *front-end*

```
cd {{front-end-project-location}}
start /min ng serve --host 0.0.0.0 --disable-host-check
start /min java -jar {{back-end-jar-location}}
```

Figura 3.17: *Script para inicialização do dashboard*

.

4 DASHBOARD PROPOSTO

Neste capítulo é explicado sobre o funcionamento do *dashboard*. Nele são tratados sobre as diferentes funcionalidades que estão presentes no *dashboard* e como estas podem auxiliar na tomada de decisão pelos usuários do sistema.

Na seção 4.1 é explicado sobre a estrutura básica dos painéis que compõem o *dashboard*, nas seções 4.2, 4.3 e 4.4 é tratado sobre o funcionamento e utilidade de cada um dos três painéis desenvolvidos. Por fim, na seção 4.5 é tratado a respeito da responsividade do *dashboard*.

4.1 Estrutura dos painéis

No *dashboard* existem três painéis: painel temporal, painel comparativo e painel de anomalias. Cada um destes são responsáveis por permitir ao usuários visualizar os dados de diferentes formas e dessa maneira realizar análises distintas. Cada um dos painéis possuem a mesma estrutura básica: menu direito, menu esquerdo e parte central, como é possível de observar na Figura 4.1. O menu da esquerda é o menu dos dados, é a partir dele que se determina quais dados devem ser mostrados no *dashboard*. O menu à direita, é o menu de visualização, ele determina configurações de como devem ser apresentados os dados. Na parte central é onde é mostrado o conteúdo que é gerado através do que é configurado nos menus.

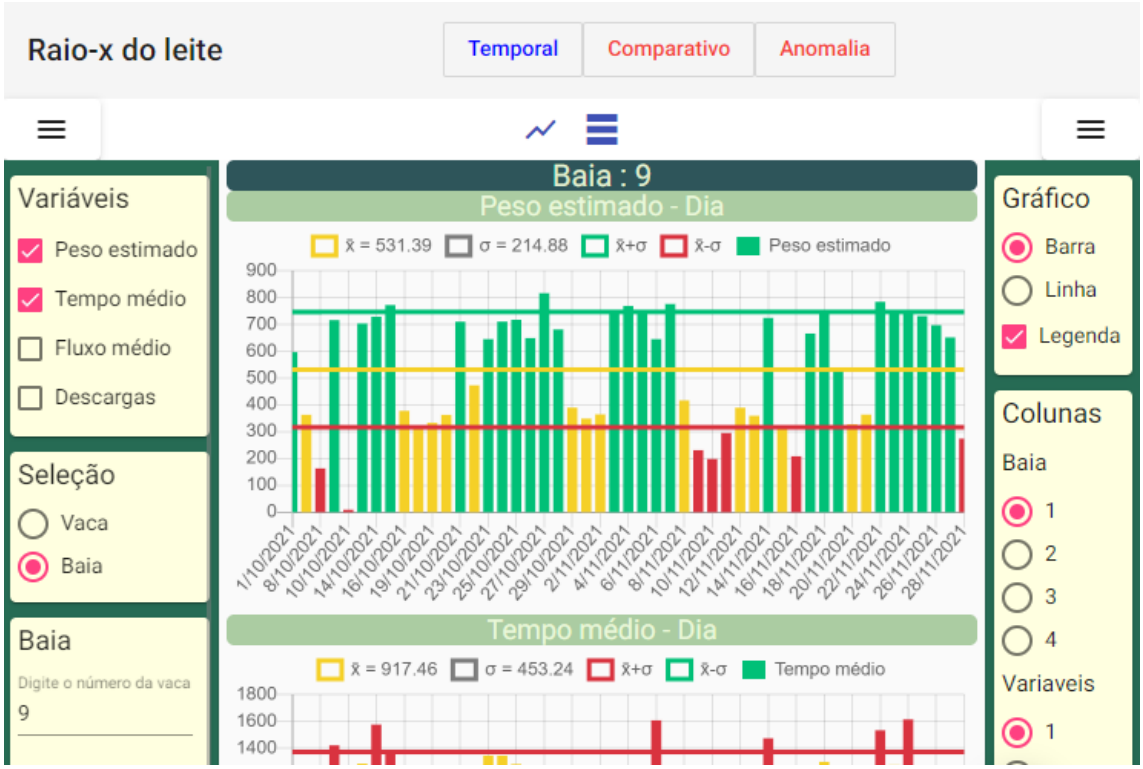


Figura 4.1: *Imagem do painel temporal*

No painel temporal e no painel comparativo estão presentes dois botões na cor azul que se encontram acima do local onde são disponibilizados os dados para serem visualizados. Esses dois botões servem para ser possível escolher o modo de como são representados os dados, como gráficos ou tabelas.

A transição entre os diferentes painéis é realizada através de três botões que se encontram no cabeçalho da página, Figura 4.1. Ao selecionar uma das opções, é ativado um dos painéis para serem visualizados.

4.2 Painel temporal

O painel temporal permite com que o usuário consiga analisar as produções de diferentes vacas ou que ocorreram em diferentes baias ao longo da passagem do tempo. Esse tipo de análise pode ser útil em diferentes circunstâncias, como por exemplo:

- Verificar quando uma vaca teve um desempenho superior ou inferior comparado aos períodos anteriores. Podendo por exemplo ser um indicativo de possíveis doenças.
- Monitorar como tem sido a recuperação de animais que estiveram doentes e como tem sido a eficiência do tratamento que o animal tem recebido.
- Verificar o desempenho das ordenhas que ocorreram nas diferentes baias, sendo que, valores muito divergentes podem indicar que os equipamentos presentes nas

baias estão com algum defeito ou então que algum fator ambiental possa estar interferindo no resultado das ordenhas.

- Verificar como a mudança na alimentação dos animais tem tido interferência no resultado da ordenha.

Nos tópicos a seguir serão apresentados e explicados o funcionamento das partes que compõem o painel temporal.

4.2.1 Adição de vacas ou baias

Figura 4.2: *Campos para seleção de vacas/baias*

No campo *Seleção*, do menu de dados, é possível determinar se serão adicionadas vacas ou baias para serem visualizadas. O submenu logo abaixo de seleção recebe o nome de "Vaca"ou de "Baia"dependendo da escolha feita em *Seleção*, conforme a Figura 4.2. Ao clicar em "Digite o número"é listado as baias/vacas que estão presentes na fazenda e a medida que o usuário preenche o campo ocorre a filtragem dos valores mostrados, Figura 4.3. Ao clicar em uma das opções listadas é adicionada uma nova vaca/baia para ser visualizada graficamente ou em tabelas.

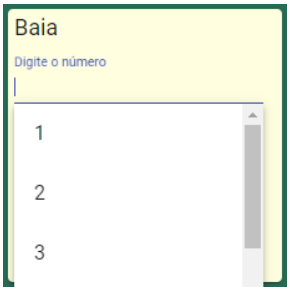


Figura 4.3: *Listagem das possíveis baias presentes na fazenda*

As vacas/baias já adicionadas pelo usuário são listadas logo abaixo do campo de preenchimento automático, Figura 4.2. Para cada vaca/baia listada, existem dois botões que permitem realizar diferentes ações em relação ao animal selecionado.

- O botão com o ícone da lixeira permite remover a vaca/baia já selecionada.

- O ícone da seta para baixo realiza o *download* de uma imagem no formato png dos gráficos ou tabelas da respectivas vacas.

Através desses campos, é possível que o usuário selecione uma ou mais vacas/-baias para serem visualizados simultaneamente, permitindo uma análise comparativa entre os diferentes gráficos ou tabelas gerados, conforme é mostrado na imagem abaixo:

Figura 4.4: *Exemplo de adição de múltiplas baias para visualização*

4.2.2 Selecão da discretização dos dados

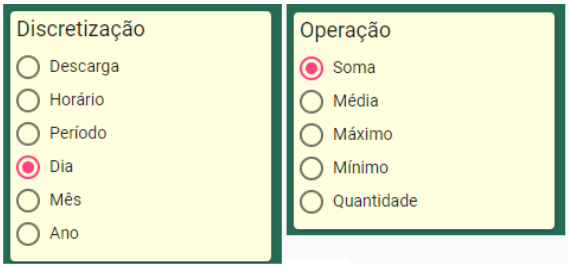


Figura 4.5: *Listagem das possíveis baias presentes na fazenda*

Através do campo *Discretização*, presente no menu dos dados, é possível determinar o nível de granularidade dos dados, em outras palavras, se os dados devem ser mostrados a cada descarga, a cada horário de retirada do leite ou serem agrupados por período do dia, a cada dia completo, a cada mês ou a cada ano.

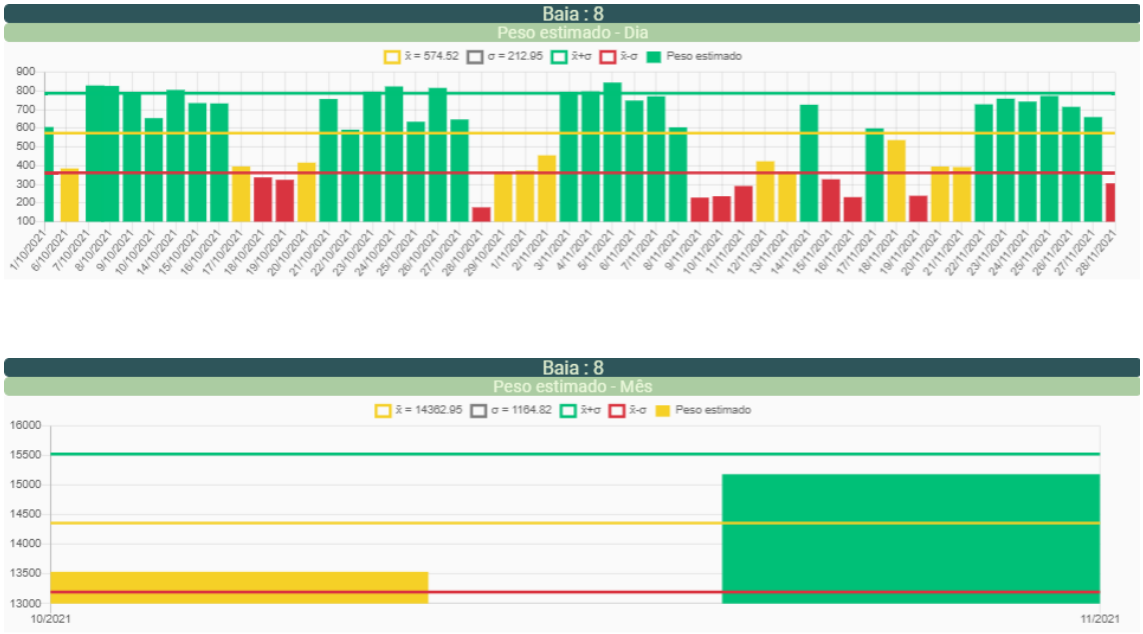


Figura 4.6: *Exemplos de gráficos com discretizações de dia e mês*

Ao selecionar a discretização como período, dia, mês ou ano, o painel se modifica e disponibiliza um novo submenu de nome *Operação*. Ele determina como são operados os dados para serem agrupados. Em *Operações* podemos escolher se os dados são somados, calcular a média dos dados, pegar o maior ou o menor valor ou obter a quantidade de dados.

Através da combinação desses dois campos, *Discretização* e *Operação*, pode-se obter diferentes resultados que podem ajudar na análise dos dados pelo usuário, alguns exemplos de casos de uso são:

- Determinar as somas diárias do peso de leite produzido pelas vacas.
- Determinar os valores médios anuais do tempo para realizar uma ordenha em uma baia.
- Determinar os menores resultados de uma vaca em cada um dos meses.

Diferente das outras opções de discretização, ao selecionar *Descarga*, *Horário* ou *Período*, são mostrados três gráficos ou três tabelas simultaneamente. Eles são disponibilizados lado a lado e representam os períodos das ordenhas na madrugada, na manhã e na tarde. Essa divisão em três gráficos facilita o entendimento e monitoramento pelos usuários.

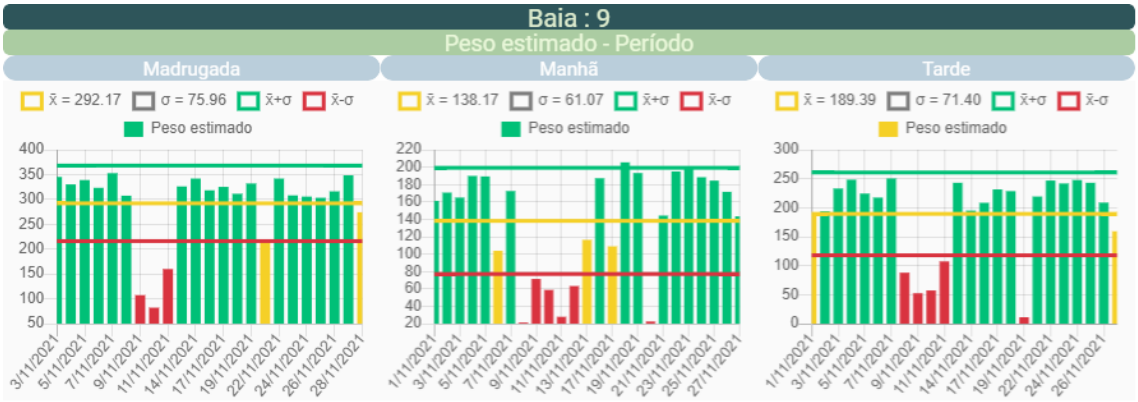


Figura 4.7: *Exemplos de gráficos com discretizações período*

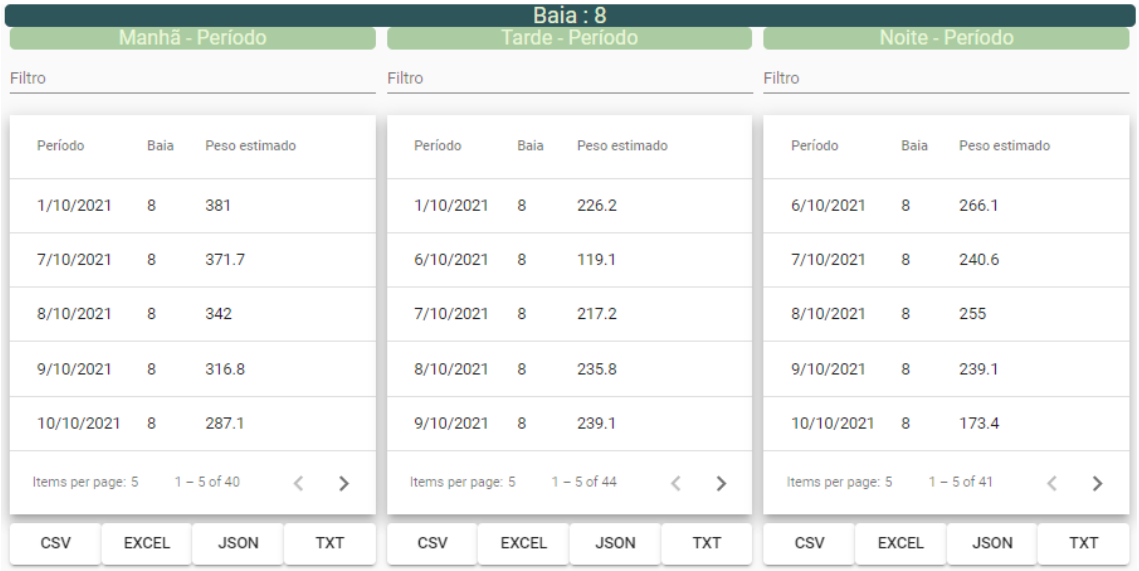


Figura 4.8: *Exemplos de tabelas com discretizações período*

4.2.3 Seleção de variáveis

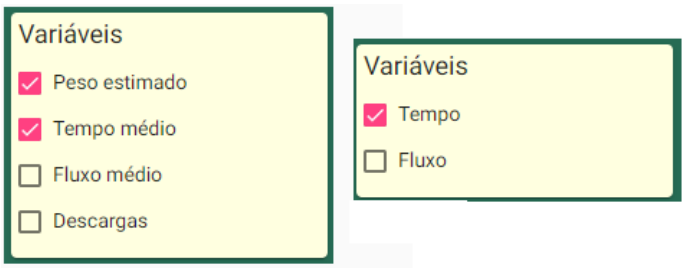


Figura 4.9: *Campo para seleção das variáveis. A esquerda encontram-se as opções disponíveis de variáveis quando no campo "Discretização"for selecionado "Descarga"*

Na campo *Váriaveis*, existe uma *check-box* para cada uma das opções de variáveis disponíveis. Através desse campo o usuário pode adicionar ou remover variáveis para serem visualizadas. Nesse campo pode-se escolher múltiplas variáveis para serem mostradas simultaneamente para cada baia/vaca, como mostrado na Figura 4.10. Isso facilita a análise do desempenho das vacas e baias sobre diferentes aspectos simultaneamente.

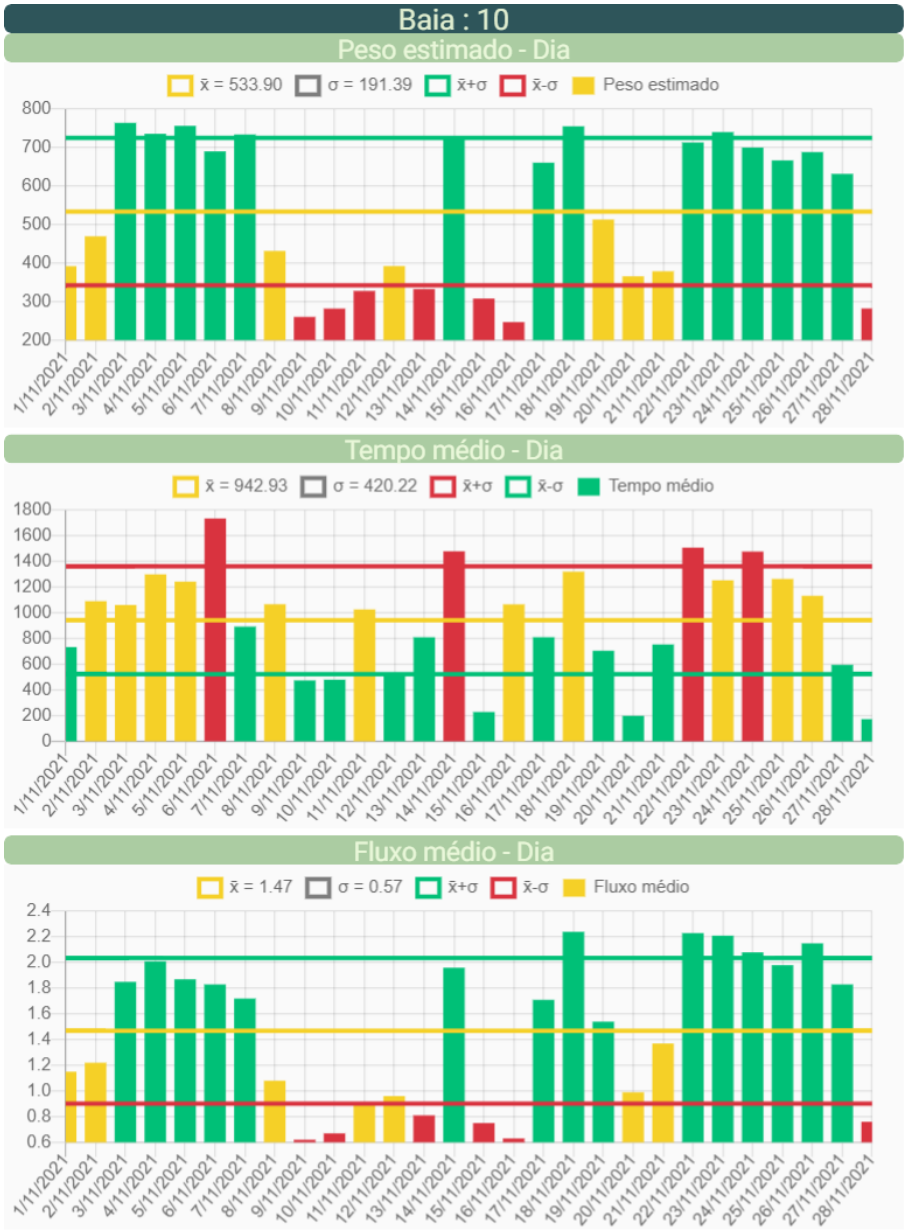


Figura 4.10: *Exemplo de visualização de diferentes variáveis a respeito da produção em uma mesma baia.*

Os valores que estão disponíveis para serem selecionados em *Variáveis* dependem da opção já selecionada no campo *Discretização*, Figura Figura 4.9. Quando esta selecionado a opçao *Descarga* no campo *discretização*, existem as opções de variáveis:

- Tempo: Tempo para encher cada balde.

- Fluxo: Fluxo de leite para encher cada balde.

Se a *Discretização* selecionada for diferente de *Descarga*, as opções são:

- Peso estimado: Peso estimado de cada ordenha.
- Tempo médio: Média do tempo necessário para encher os baldes.
- Fluxo médio: Média do fluxo de cada descarga individual em uma ordenha.
- Descargas: Quantidade de baldes enchidos.

4.2.4 Seleção do intervalo dos dados

Figura 4.11: *Campos para seleção do intervalo analisado*

Na parte de menu dos dados, existe um campo de nome *Intervalo*, no qual é possível escolher se é possível determinar quais as datas iniciais e finais a serem consideradas ao visualizar os dados. Dessa forma são levados em conta na análise somente dados pertencentes ao período que o usuário quer analisar. Na Figura 4.12 são mostrados alguns gráficos que foram gerados para diferentes intervalos de dados.

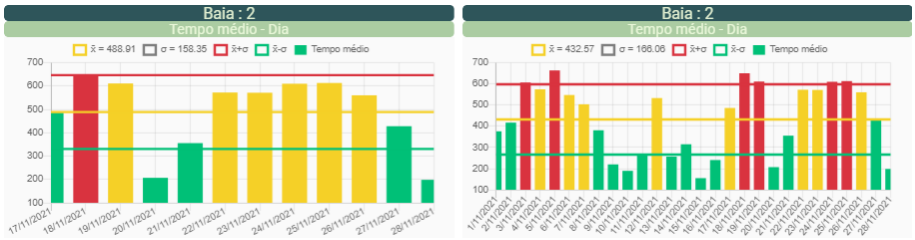


Figura 4.12: *Exemplo de visualização dos dados para dois intervalos diferentes*

4.2.5 Configurações de visualização dos gráficos e tabelas

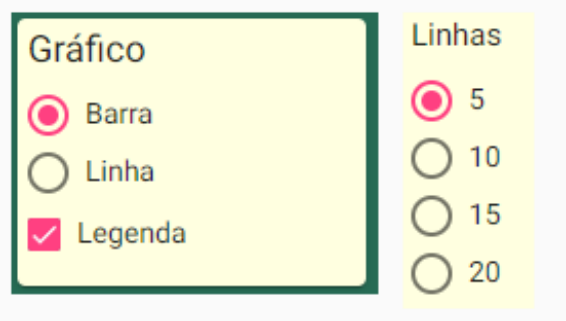


Figura 4.13: *Campos para selecionar opções a respeito dos gráficos e a respeito das tabelas*

No menu de visualização, existem opções para determinar qual o tipo de gráfico que deve ser mostrado no painel, pode-se escolher entre gráficos de linhas ou colunas. Além disso existe um campo do tipo *check-box* que permite escolher se os gráficos gerados possuem ou não legenda, Figura 4.13. Quando o modo de visualização dos dados está como tabelas, é possível alterar a quantidade de linhas que é mostrado nas tabelas geradas, Figura 4.13.

4.2.6 Organização visual dos dados em colunas

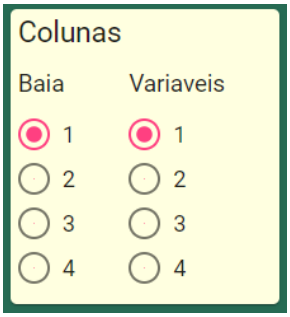


Figura 4.14: *Campos para determinar a quantidade de variáveis e baias/vacas que são disponibilizados lado a lado.*

No menu de visualização existem dois campos que permitem organizar os gráficos/tabelas em diferentes colunas. Através deles é possível determinar a quantidade de vacas/baias que são disponibilizadas lado a lado. Além disso, é possível determinar a quantidade de variáveis que são disponibilizados lado a lado para cada uma das vacas/baias. A combinação de diferentes valores para esses dois campos permite ao usuário obter diferentes formas de organizar os gráficos/tabelas no *dashboard*, como exemplificado nas imagens abaixo.

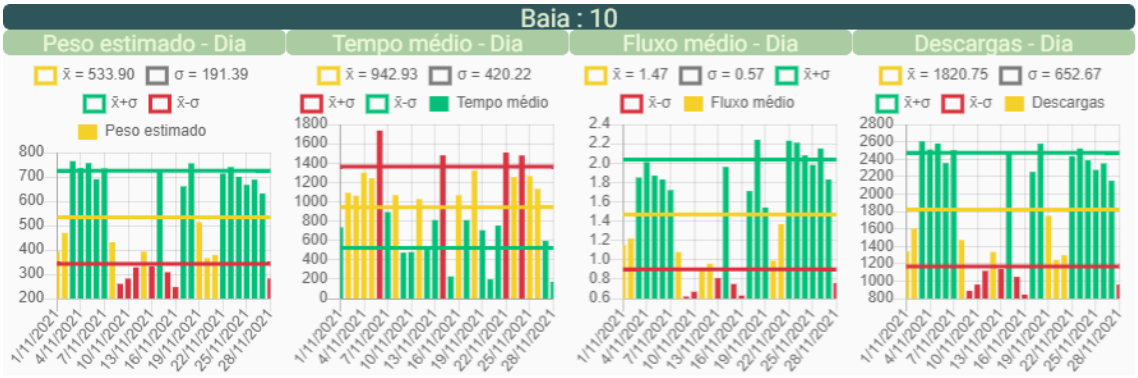


Figura 4.15: *Exemplo de possível visualizações quando a quantidade de colunas para variáveis for igual a quatro*

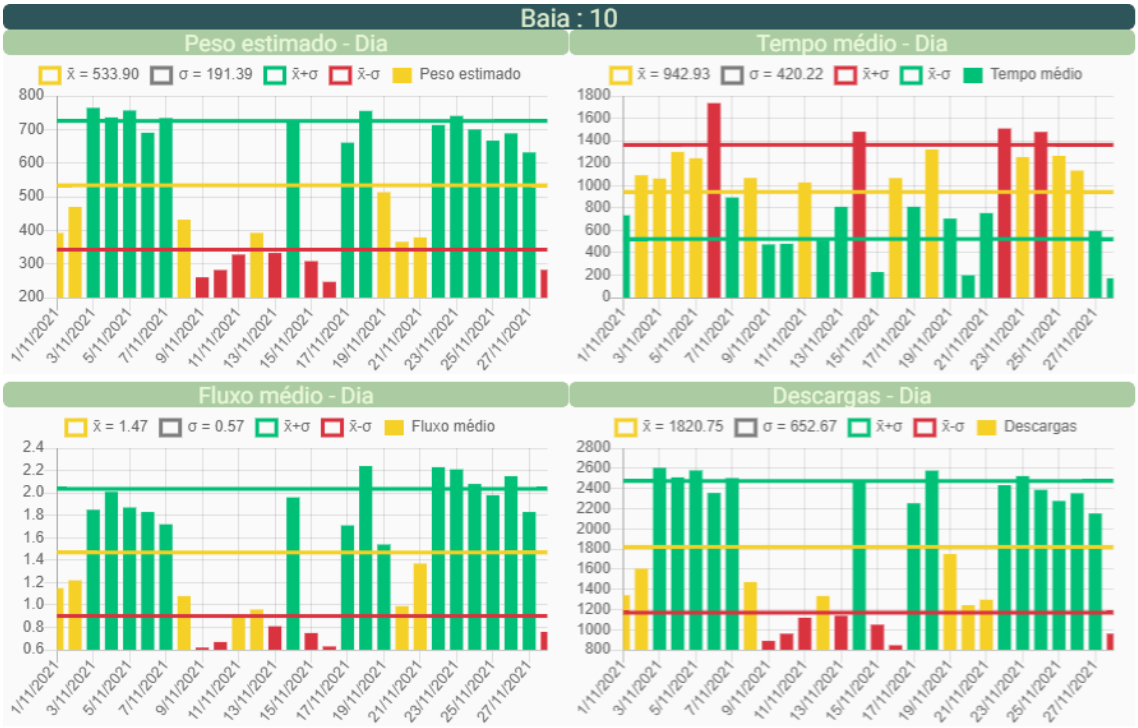


Figura 4.16: *Exemplo de possível visualizações quando a quantidade de colunas para variáveis for igual a dois*

4.3 Painel comparativo

Diferente do painel temporal que apresenta os dados em função do tempo, o painel comparativo permite apresentar os dados em função de cada uma das vacas ou baias. Dessa forma, em um mesmo gráfico/tabela são disponibilizados os dados de todas as vacas/baias que possuem dados no período especificado. Alguns exemplos resultados possíveis de se obter nesse painel são:

- Determinar a produção de leite total de cada uma das vacas no período especificado.

- Determinar o tempo médio da tiragem do leite de cada uma das vacas no período especificado.
- Determinar os menores valores da quantidade de descargas de cada uma das vacas no período especificado.

Esse painel é útil para realizar uma análise comparativa do desempenho das diferentes vacas/baias. Alguns exemplos de circunstâncias que esse painel pode ser útil na análise de dados são:

- Determinar quais os animais são mais rentáveis para a fazenda.
- Determinar os animais que podem estar doentes.
- Determinar animais pouco lucrativos e que poderiam ser abatidos para a venda de carne.
- Identificar baias que podem estar com equipamentos defeituosos.

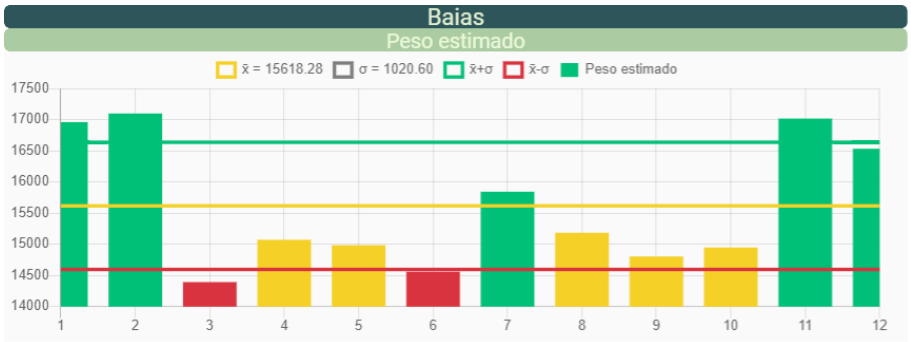


Figura 4.17: *Exemplo de gráfico gerado no painel comparativo. Ele mostra a quantidade total de leite produzido por cada uma das vacas em um intervalo de tempo de alguns dias*

O funcionamento e interação do usuário com esse painel é similar ao painel temporal, possuindo boa parte dos campos do menu de dados e também os mesmos campos do menu de visualização. Diferentemente do painel temporal, esse painel não possui o campo discretização e nem os campos necessários para a adição de vacas/baias, isso ocorre por que os dados deixam de ser analisados em função do tempo e passam ser analisados em função das vacas/baias.

4.4 Painel de anomalias

O painel de anomalias permite ao usuário identificar de maneira rápida e centralizada todas as anomalias que foram encontradas nas ordenhas das vacas em um período

pré-determinado. O principal caso de uso para esse painel é a identificação de doenças e problemas de desempenhos nos animais.

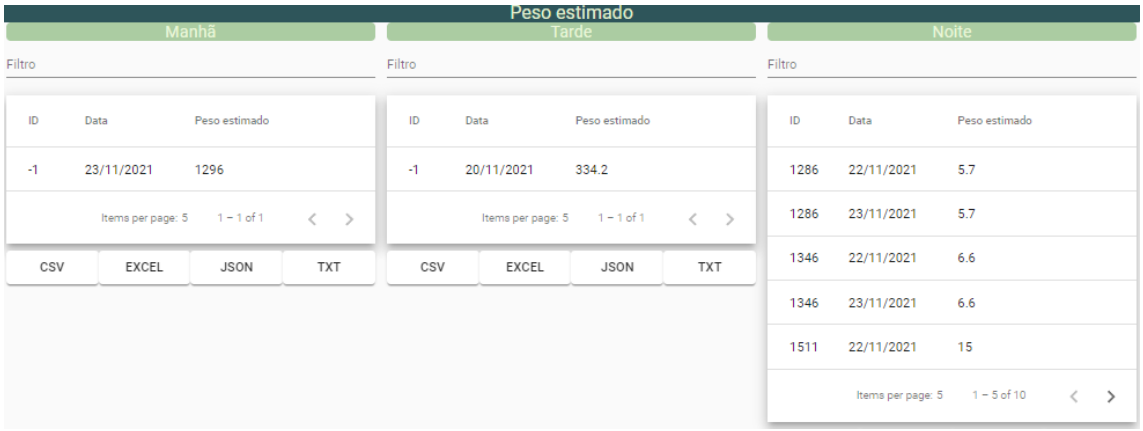


Figura 4.18: *Exemplo de tabelas gerada no painel de anomalias*

Nesse painel o usuário pode escolher qual o intervalo considerado dos dados e também quais as variáveis que irão ser analisadas. Para cada variável selecionada, o painel apresenta três tabelas contendo todas as anomalias que ocorreram no período da madrugada, manhã e da tarde, Figura 4.18. Em cada uma das tabelas geradas estão contidos três colunas, sendo estas responsáveis por identificar qual animal apresentou uma anomalia, quando ocorreu a anomalia e também qual o valor que está sendo considerado anômalo.

4.5 Reponsividade

No *dashboard* foram feitas implementações para que o mesmo seja utilizado em dispositivos mobile. Os menus esquerdo e direito têm seu tamanho reduzido e passam a ocupar o mínimo de espaço possível quando a largura da tela é menor que 1000px. Outra modificação feita em questão de responsividade é que os gráficos para os diferentes períodos do dia não são mostrados lado a lado quando o tamanho disponível para eles é menor que 700px, como mostrado na Figura 4.20.

Outra funcionalidade que ajuda a melhorar a responsividade para mobile é a opção de esconder o menu de dados ou o menu de visualização. Para isso existem dois botões no *dashboard*, cada um em cima de cada menu. Dessa forma o usuário tem a opção de obter maior espaço de tela para visualizar os dados gerados. Essa funcionalidade é principalmente útil para a visualização do *dashboard* com o dispositivo móvel na vertical. Na Figura 4.20 é possível observar essa funcionalidade, nela o menu da direita não está visível.

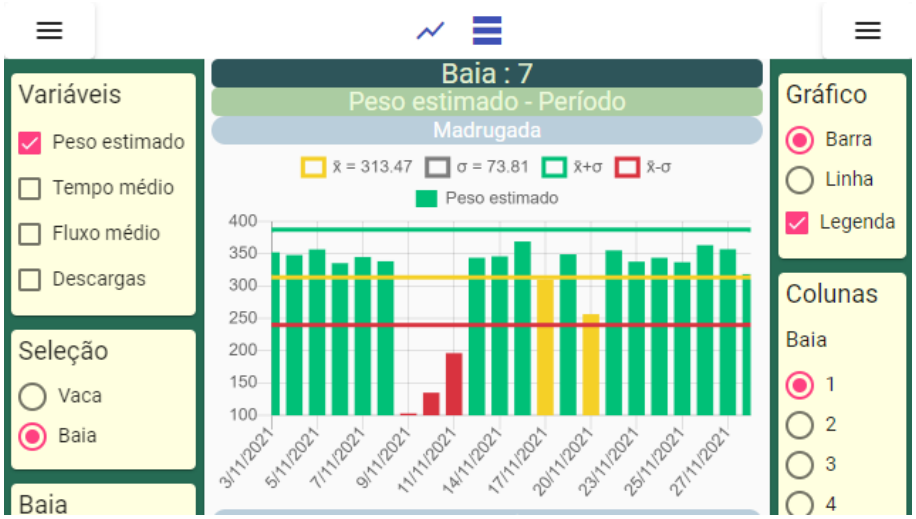


Figura 4.19: *Dashboard para resolução de 640 X 360*

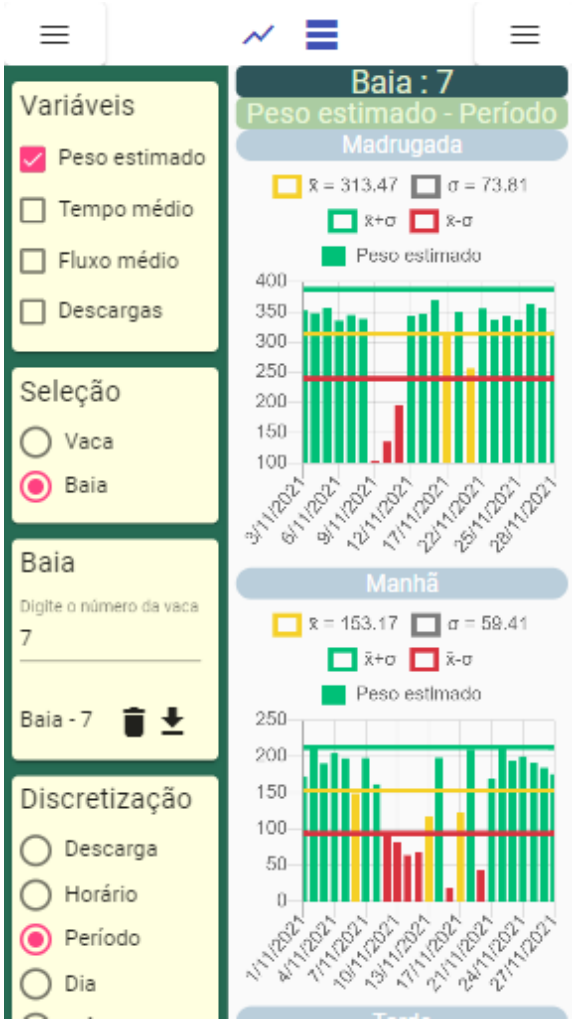


Figura 4.20: *Dashboard para resoluções de 340 X 640.*

5 CONCLUSÃO

A pecuária leiteira tem grande importância na economia do país, sendo que a mesma representa boa parte dos lucros vindos da pecuária como um todo. Nesse ramo existem grandes desafios que precisam ser enfrentados em diferentes quesitos, sendo necessário a utilização de métodos modernos para resolver os mesmos. A pecuária de precisão utiliza tecnologias para conseguir aumentar o desempenho na realização das atividades e na resolução de problemas. Dentre as diferentes tecnologias que podem ser utilizadas nesse ramo, os *dashboards* permitem o auxílio na visualização e manuseio de dados, possibilitando encontrar problemas, ter novas ideias e entender o estado atual da produção de leite.

Neste trabalho foi desenvolvido um *dashboard* com o objetivo de auxiliar no monitoramento e também na tomada de decisão. Nele foram trabalhados questões a respeito de persistência de dados, disponibilização dos mesmos para serem acessados através de *Web Services Rest* e também do manuseio, estruturação e da representação visual dos mesmos.

No *dashboard* foram trabalhados maneiras para visualizar os dados de diferentes formas, sendo possível encontrar anomalias nos dados, visualizar como cada animal tem seu desempenho ao longo do tempo e ter uma análise comparativa entre os diferentes animais. Além disso, estão presentes diferentes parâmetros que os usuários podem alterar, permitindo alterar tanto os dados que são mostrados e também a forma que os mesmos são apresentados.

A partir do trabalho desenvolvido foi possível desenvolver uma ferramenta que pode auxiliar na gestão eficiente da fazenda, podendo trazer possíveis benefícios como: encontrar animais doentes, determinar o desempenho total na produção de leite, determinar quais animais devem ser mantidos como produtores, quais devem ser abatidos, verificar como tem sido a recuperação dos animais que estiveram doentes, etc.

REFERÊNCIAS BIBLIOGRÁFICAS

ANASTASIA. **What's the Difference Between Single-Page and Multi-Page Apps**. 2018. Disponível em: https://rubygarage.org/blog/single-page-app-vs-multi-page-app. Acesso em: 01 dez 2021. 16

ANGULAR. **Introduction to the Angular Docs**. 2010a. Disponível em: https://angular.io/docs. Acesso em: 01 dez 2021. 16

____. **what is angular ?** 2010b. Disponível em: https://angular.io/guide/what-is-angular. Acesso em: 01 dez 2021. 16, 17

BAELDUNG. **The DAO Pattern in Java**. c2020. Disponível em: https://www.baeldung.com/java-dao-pattern. Acesso em: 07 dez 2022. 18

CADU. **ORM : Object Relational Mapper**. 2011. Disponível em: https://www.devmedia.com.br/orm-object-relational-mapper/19056. Acesso em: 01 dez 2021. 16

CAELUM. **UMA INTRODUÇÃO PRÁTICA AO JPA COM HIBERNATE**. c2004. Disponível em: https://www.caelum.com.br/apostila-java-web/uma-introducao-pratica-ao-jpa-com-hibernate. Acesso em: 01 dez 2021. 16

COSTA, Rebeca. **Dashboard Design: best practices and examples**. 2020. Disponível em: https://www.justinmind.com/blog/dashboard-design-best-practices-ux-ui/. Acesso em: 03 dez 2021. 9, 11, 12

DALEPIANE, Felipe. **DAO Pattern: Persistência de Dados utilizando o padrão DAO**. 2014. Disponível em: https://www.devmedia.com.br/dao-pattern-persistencia-de-dados-utilizando-o-padrao-dao/30999. Acesso em: 16 dez 2022. 18

DENIS, Rocha; CARVALHO, Glauco; RESENDE, João. **Cadeia produtiva do leite no Brasil: produção primária**. 2020. Disponível em: https://ainfo.cnptia.embrapa.br/digital/bitstream/item/215880/1/CT-123.pdf. Acesso em: 01 nov 2021. 9

EMBRAPA. **Pecuária leiteira de precisão: uso de sensores para monitoramento e detecção precoce de alterações na saúde de bovinos leiteiros**. 2018. Disponível em: https://ainfo.cnptia.embrapa.br/digital/bitstream/item/179729/1/DOC-227-Pec-Leit-Prec-Sensores.pdf. Acesso em: 01 nov 2021. 9

FALEIRO, Adail. **Modelagem Relacional**. 2011. Disponível em: https://www.devmedia.com.br/modelagem-relacional/19614. Acesso em: 03 dez 2021. 13

GAMA, Alexandre. **Inversão de Controle x Injeção de Dependência**. 2010. Disponível em: https://www.devmedia.com.br/inversao-de-controle-x-injecao-de-dependencia/18763. Acesso em: 01 dez 2022. 18

GOMES, Pedro. **O QUE É UM DASHBOARD? O GUIA COMPLETO E DEFINITIVO!** 2017. Disponível em: https://www.opservices.com.br/o-que-e-um-dashboard/. Acesso em: 03 nov 2021. 9, 11

IBM. **Arquitetura de três camadas (tiers)**. 2020. Disponível em: https://www.ibm.com/br-pt/cloud/learn/three-tier-architecture. Acesso em: 03 jan 2022. 12

MAVEN. **Introduction to the Build Lifecycle**. c2002a. Disponível em: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html. Acesso em: 12 dez 2021. 15

____. **What is maven ?** c2002b. Disponível em: https://maven.apache.org/what-is-maven.html. Acesso em: 12 dez 2021. 15

MITTELBACH, Frank; GOOSSENS, Michel; BRAAMS, Johannes; CARLISLE, David; ROWLEY, Chris. **MASTITE EM VACAS LEITEIRAS- REVISÃO DE LITERATURA**. 2011. Disponível em: http://faef.revista.inf.br/imagens_arquivos/arquivos_destaque/5birfPwQOBxdHFp_2013-6-26-11-19-44.pdf. Acesso em: 01 nov 2021. 9

MOZILLA. **Uma visão geral do HTTP**. c2005. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview. Acesso em: 03 jan 2022. 12

OPENSOFT. **Web service: o que é, como funciona, para que serve?** c2001. Disponível em: https://www.opensoft.pt/web-service/. Acesso em: 05 dez 2021. 14

ORACLE. **What is a Relational Database (RDBMS)?** c1995a. Disponível em: https://www.oracle.com/database/what-is-a-relational-database/. Acesso em: 05 dez 2021. 13

____. **What Are RESTful Web Services?** c1995b. Disponível em: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.h. Acesso em: 08 dez 2021. 14

____. **JPA Query**. c1995c. Disponível em: https://docs.oracle.com/html/E13946_04/ejb3_langref.html. Acesso em: 01 dez 2021. 16

PAUMEIRA, Thiago. **Como funcionam as aplicações web ?** 2012. Disponível em: https://www.devmedia.com.br/como-funcionam-as-aplicacoes-web/25888. Acesso em: 03 jan 2022. 12

REFACTORING.GURU. **Template Method**. c2014. Disponível em: https://refactoring.guru/pt-br/design-patterns/template-method. Acesso em: 02 jan 2022. 19

SILVEIRA, Paulo. **O que é SQL?** 2019. Disponível em: https://www.alura.com.br/artigos/o-que-e-sql?gclid=CjwKCAjwuvmHBhAxEiwAWAYj-HI2aWDCDYnmacjNnuP-6oXstHWPfr5OSHz_ttliuDs8UWtPC7m-vBoCUT4QAvD_BwEPostgresql. Acesso em: 05 dez 2021. 14

SOUTO, Mario. **O que é front-end e back-end?** 2019. Disponível em: https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end?gclid=Cj0KCQiA2sqOBhCGARIsAPuPK0icf-QcePjhi158m__sYcBMn7RyV57ClFxnYfsaZLU_UP_cOCqJ28YaAgM1EALw_wcB. Acesso em: 03 jan 2022. 13

SPRING. **Why spring ?** c2011a. Disponível em: https://spring.io/why-spring. Acesso em: 12 dez 2021. 15

____. **Introduction to Spring Framework**. c2011b. Disponível em: https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/overview.html. Acesso em: 12 dez 2021. 15

____. **Spring Boot**. c2011c. Disponível em: https://spring.io/projects/spring-boot. Acesso em: 09 dez 2021. 15