UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATHEUS PACHECO DA SILVA

CONTROLE DE PRODUÇÃO DE GADO LEITEIRO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2018

MATHEUS PACHECO DA SILVA

CONTROLE DE PRODUÇÃO DE GADO LEITEIRO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

CONTROLE DE PRODUÇÃO DE GADO LEITEIRO

POR

MATHEUS PACHECO DA SILVA

Este trabalho de conclusão de curso foi apresentado no dia 12 de dezembro de 2018, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Profª Drª Beatriz Terezinha Borsoi
Orientador

Profª MSc. Andreia Scariot Beulke

Prof. Esp. João Guilherme Brasil Pichetti

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

SILVA, Matheus Pacheco da. Controle de produção de gado leiteiro. 2018. 50f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2018.

De maneira bastante simplificada, a lucratividade na produção de gado leiteiro está no resultado positivo da relação entre o custo de alimentação e a quantidade de leite produzida. É claro que, agregado ao custo de alimentação, estão os custos de compra das matrizes, de tratamento de doenças, de inseminações, as instalações, o pessoal e outros. Porém, a relação entre o custo de alimentação e o leite produzido pelo respectivo animal é que determina a viabilidade de manter o exemplar no plantel. O controle da produção individual por animal pode ser difícil quando o leite produzido em cada ordenha não é individualizado por animal. Contudo, a média dessa produção pode auxiliar a determinar a viabilidade do lote e definir a lucratividade do produtor. Visando auxiliar no controle dos custos e das despesas de produção de gado leiteiro, como resultado deste trabalho foi desenvolvido um sistema *web* para auxiliar no gerenciamento de uma propriedade rural que lida com gado leiteiro. O sistema possibilita o armazenamento de dados do plantel, produtividade, despesas e receitas, entre outros, obtidos com a atividade.

Palavras-chave: Rich Internet Application. Controle de produção leiteira. Java para web.

ABSTRACT

SILVA, Matheus Pacheco da. Management of dairy cattle production. 2018. 50f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2018.

In a very simplified way, the profitability of dairy catlle production is in the positive result of the relation between the cost of feeding and the quantity and quality of milk produced. In addition to the cost of food, are the costs of purchasing the herd, treatment of diseases, insemination, among others. However, the relationship between the cost of feeding and the milk produced by each animal determines the viability of maintaining each milky cow. The control of the production by animal can be difficult when the milk produced is not individualized per animal. However, the average of this production can help determine the viability of the lot and define the profitability of the producer. Aiming to help costs and expenses of dairy cattle production, as a result of this work, a web system was developed to assist in the management of milk catlle. The system allows the storage data of expenses and dairy milk production.

Keywords: Rich Internet Application. Milk cattle. Web Java.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API	*Application Programming Interface*
CRUD	*Create, Read, Update e Delete*
CSS	*Cascading Style Sheets*
HTML	*HyperText Markup Language*
PDF	*Portable Document Format*
REST	*Representational State Transfer*
RIA	*Rich Internet Application*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa de realização do trabalho. Por fim está a organização do texto com a apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

O agronegócio desempenha um papel bastante relevante na matriz econômica atual com pequenas, médias e grandes propriedades sustentando a balança comercial do País (ECOAGRO, 2016). O agronegócio é entendido como a soma de quatro segmentos (CENTRO…, 2016): (a) insumos para a agropecuária, (b) produção agropecuária básica, (c) agroindústria (processamento) e (d) serviços. Esses segmentos envolvem o setor agrícola (vegetal) e o pecuário (animal). No setor pecuário está a produção de leite, ou bovinocultura de leite. Dados de 2014 da Secretaria de Estado da Agricultura e do Abastecimento colocam o Brasil como o quinto maior produtor mundial de leite. Em 2014 foram produzidos no país 3,3 bilhões de litros de leite (SECRETARIA…, 2016).

Para aumentar a produtividade e o lucro e oferecer bem-estar aos animais, os produtores podem, cada vez mais, beneficiar-se de suporte tecnológico. Esse suporte vai das tecnologias robóticas para ordenha, por exemplo, até o uso de sistemas de informação para registro de dados de produção. Esses dados podem ser informativos de custo e de lucro e utilizados para a tomada de decisão.

Considerando esse contexto, o sistema proposto como resultado da realização deste trabalho foi desenvolvido visando auxiliar o produtor no gerenciamento de dados provenientes da atividade com gado leiteiro. Esses dados auxiliam a identificar os exemplares mais produtivos, o gerenciamento do plantel e o controle de receitas e despesas. O acompanhamento dos animais é a base para um rebanho produtivo.

O sistema visa oferecer a possibilidade de armazenar informações diárias. Nesse sistema os exemplares do plantel, destacando-se as vacas, são previamente cadastrados. Elas formam a base dos dados coletados. Esses dados se referem, basicamente, à produção leiteira diária e aos tratamentos utilizados com o animal

(vacinas, medicamentos, inseminações, ração e etc.). Assim, o produtor tem acesso a dados da produção leiteira e dos gastos com cada animal. Esses dados permitem identificar as vacas que são essenciais para o plantel, em termos da melhor relação entre custo e benefício, e quais é aconselhável que sejam descartadas ou vendidas.

1.2 OBJETIVOS

A seguir são apresentados os objetivos pretendidos com a realização deste trabalho.

1.2.1 Objetivo Geral

Desenvolver um sistema para controlar a produtividade de um plantel de gado leiteiro.

1.2.2 Objetivos Específicos

- Facilitar o armazenamento de dados gerais de um plantel de gado leiteiro, enfatizando dados das vacas e de produção do rebanho.
- Manter dados para inseminação.
- Armazenar dados de produção leiteira por animal ou por média de produção.
- Armazenar dados de receitas e despesas.

1.3 JUSTIFICATIVA

O sistema desenvolvido é para o ambiente *web*. Nas propriedades rurais é, ainda, relativamente comum não haver acesso à Internet. Contudo, isso não impede que um sistema *web* seja executado como *localhost*. A opção pelo desenvolvimento

web decorre das facilidades que tecnologias para esse tipo de desenvolvimento oferecem. E se houver acesso à internet ou quando na propriedade houver esse acesso, o sistema oferecerá ao produtor acesso aos dados a partir de outros computadores.

O controle da produção de cada animal é útil para o processo de tomada de decisão sobre alimentação e se o animal deve ser mantido ou descartado. O controle da quantidade geral de leite produzido, o valor obtido dessa produção e o respectivo registro de todos os custos envolvidos nessa produção, auxiliam a definir o lucro e, além disso, a estabelecer estratégias para minimizar custos.

1.4 ESTRUTURA DO TRABALHO

O texto a seguir está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico sobre aplicações Internet caracterizadas como ricas, por ser esse o tipo de aplicação desenvolvido como resultado deste trabalho. No Capítulo 3 estão as ferramentas e as tecnologias utilizadas na modelagem e na implementação do sistema. No Capítulo 4 é apresentado o resultado da realização do trabalho que é a modelagem e o desenvolvimento de um sistema para controlar a produtividade de um plantel de gado leiteiro. Por fim, estão as considerações finais seguidas pelas referências utilizadas na composição do texto.

2 APLICAÇÕES INTERNET RICAS

Com a evolução da Internet e seus recursos, as aplicações *web* baseadas em *HyperText Markup Language* (HTML) têm mostrado suas limitações, especialmente quando essas aplicações passaram a ser utilizadas para atividades mais complexas realizadas por meio de interface gráfica com o usuário. As *Rich Internet Applications* (RIA) são vistas como uma solução para essas limitações (ROUBI; ERRAMDANI; MBARKI, 2016).

Tendências no desenvolvimento *web* permanecem vinculadas ao paradigma de páginas *web*. Contudo, novos usos das tecnologias disponíveis e desenvolvimentos recentes em termos de conceitos, como comunicação assíncrona, tem produzido uma nova geração de aplicações *web*, as denominadas RIAs (MARTÍNEZ-RUIZ, 2010).

Uma *Rich Internet Application* é caracterizada pela combinação das vantagens do modelo distribuído da *web* com a interatividade e a riqueza da interface das aplicações *desktop* (ROUBI; ERRAMDANI; MBARKI, 2016). Em termos de funcionamento para o usuário, as RIAs são aplicações *web* que emulam as funcionalidades das aplicações *desktop* tradicionais em um navegador *web* (VELASCO et al., 2008).

Embora a concepção desse tipo de aplicação tenha surgido por volta dos anos 1990 com os *plugins* proprietários e algumas funcionalidades implementadas por meio de *scripts*, ainda não foi nos primeiros anos dessa década que o termo tornou-se amplamente divulgado e suas características definidas (ALLAIRE, 2002, DUHL, 2003).

As RIAs foram inicialmente planejadas para serem executadas em *sandboxes* proprietárias em diferentes agentes no usuário, mas o desenvolvimento de interfaces baseadas em HTML atingiu o seu limite. A inserção de objetos XMLHttpRequest no início dos anos 2000 e sua rápida adoção pela maioria dos navegadores, permitiu uma sobrevida para as aplicações *web* denominadas tradicionais (VELASCO et al., 2008).

Objetos XMLHttpRequest definem uma *Application Programming Interface* (API) que provê funcionalidades no cliente na forma de *scripts* para transferir dados entre cliente e servidor. Usando objetos XMLHttpRequest somente parte da página

web precisa ser recarregada (redesenhada), aumentando a responsividade da aplicação e melhorando a experiência do usuário. A natureza assíncrona da transferência aproxima as aplicações *web* das aplicações *desktop* em termos de interface com o usuário (VELASCO et al., 2008). Roubi, Erramdani e Mbarki (2015), ressaltam que nos anos recentes têm sido verificado um rápido crescimento das aplicações *web* com comportamento sofisticado em termos de interface com o usuário.

As RIAs estão mostrando enorme potencial de melhorar a interação do usuário alavancando condutores tecnológicos fundamentais, incluindo (PANDUINO et al., 2010):

a) distribuição de processamento entre cliente e servidor – permitindo ao cliente gerenciar dados distribuídos e a lógica de negócio e a realização de comunicação síncrona/assíncrona com o servidor;
b) atualização (*refresh*) local ao invés de atualização da página – trazendo para os usuários a sensação de fluxo contínuo e transação suave entre estados de interação, como se o usuário estivesse trabalhando com aplicações locais;
c) aumento significativo de *widgets* de interface disponíveis em navegadores *web* que são ativados por paradigmas de interação típicos de ambiente *desktop*.

As RIAs têm se tornado um novo padrão para aplicações *web*. Elas quebram o paradigma conceitual tradicional de uma aplicação *web* que executa exclusivamente no servidor. Uma aplicação *web* consiste de um software cliente (navegador, *browser*, *web*) que permite ao usuário enviar uma requisição para um servidor *web* que providencia o conteúdo como resposta. Em aplicações *web* tradicionais, o usuário navega para outra página da aplicação seguindo as *Uniform Resource Locators* (URL) incorporadas na página atual. Seguir uma URL gera uma nova requisição para o servidor e o servidor responde com uma nova página HTML que sobrescreve completamente a página atual. Inicialmente, aplicações residiam unicamente no servidor implementando a lógica da aplicação e a comunicação entre o cliente e o servidor era assíncrona. Essas aplicações não eram responsivas e faltava a elas a sensação *desktop* (*desktop feel*) de aplicações não *web* (DINCTURK, 2014).

As RIAs mudaram essa situação por meio da implementação de duas

melhorias (DINCTURK, 2014):

a) a computação nas RIAs pode ser realizada no lado cliente por meio de *scripts* como JavaScript. Assim, a página pode ser modificada parcialmente ou completamente pelos *scripts* no lado cliente, sem necessidade de comunicação com o servidor;
b) a comunicação assíncrona entre cliente e servidor. Isso permite que o usuário possa usar a aplicação sem ter que aguardar a resposta de uma requisição anterior.

As RIAs procuram atender as expectativas dos usuários em termos de usabilidade, confiabilidade, qualidade, manutenibilidade e desempenho (MARTÍNEZ-RUIZ, 2010).

3 MATERIAIS E MÉTODO

A seguir estão os materiais e o método utilizados para a modelagem e a implementação do sistema obtido como resultado da realização deste trabalho.

3.1 MATERIAIS

O Quadro 1 apresenta as tecnologias e as ferramentas utilizadas no desenvolvimento do trabalho.

Ferramenta / Tecnologia	Versão	Finalidade
IntelliJ IDEA	2016.2	Ambiente de programação Java.
PGAdmin		Administrador de banco de dados
PostgreSQL	9.6	Banco de dados.
Java		Linguagem de programação.
Astah	Community	Modelagem dos casos de uso e diagrama de classes.
Case Studio	2.23.1	Modelagem do banco de dados.
Sublime Text	3	Edição de arquivos HTML, AngularJS, JavaScript e *Cascading Style Sheets* (CSS).
TypeScript		Linguagem de programação.
CSS	3	Linguagem de estilização para HTML ou *eXtensible Markup Language* (XML).
HTML	5	Linguagem de marcação utilizada para a construção das páginas web.
Angular	6	Framework JavaScript.
Node.JS	6.2.2	Interpretador de código JavaScript que funciona ao lado do aervidor.
Spring boot	2.1.1	Utilizado no *back-end* para facilitar o processode configuração e publicação.
PrimeNG	6.0.0	Biblioteca de componentes para Angular.
Angular CLI	7.0.0	Facilita a criação de um aplicativo em Angular.

Quadro 1 – Tecnologias e ferramentas utilizadas

Para o *fron-end* como principal framework foi utilizado o Angular 6 e para o

back-end o Spring Boot.

3.2 MÉTODO

O método consiste nas atividades de levantamento de requisitos, análise, projeto e desenvolvimento e testes. A seguir estão descritas, de forma sucinta, as etapas desenvolvidas para a realização deste trabalho.

Levantamento de requisitos

Os requisitos para o sistema foram definidos a partir das necessidades e das atividades realizadas em uma propriedade que trabalha com gado leiteiro. A rotina diária, os controles e os dados que a aplicação deveria armazena foram identificados a partir da observação das atividades realizadas nessa propriedade.

Inicialmente foram listados os requisitos funcionais e não funcionais do sistema que posteriormente foram organizados na forma de casos de uso. Foram identificados dois atores. O administrador e o produtor.

Análise e projeto

Os requisitos organizados sob a forma de casos de uso foram expandidos e o diagrama de entidades e relacionamentos do banco de dados foi elaborado.

Desenvolvimento e testes

O desenvolvimento foi feito utilizando as tecnologias e as ferramentas constantes no Quadro 1. Os testes foram informais e visam identificar erros de codificação e o atendimento aos requisitos.

4 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho que é o desenvolvimento de um sistema para controle de dados de gado leiteiro.

4.1 ESCOPO DO SISTEMA

O sistema visa auxiliar no controle da produtividade de um plantel de gado leiteiro. Para que esse controle possa ser realizado é necessário haver o cadastro de cada uma das vacas do rebanho, com dados relacionados à numeração do brinco, raça e filiação. Esses dados permitem um controle geral do rebanho.

O controle do estoque e dos gastos é realizado por meio de cadastro de vacinas adquiridas pelo produtor, com dados de tipo, quantidade e valor unitário. Com as vacinas previamente cadastradas, o produtor poderá registrar as vacinações realizadas no rebanho, permitindo identificar as vacas que foram vacinadas e qual tipo de vacina aplicada. O tipo de vacina permite identificar o histórico dos problemas de saúde do animal e o período de carência para uso do leite em decorrência de medicamentos ministrados, por exemplo. O sistema conterá, também, um registro de doenças, sendo armazenado o tipo (descrição) da doença e o tempo estimado para o seu tratamento. Esse cadastro será utilizado para o registro das doenças de cada animal e permite o controle de carência de uso do leite, entre outros.

Um aspecto importante para o adequado gerenciamento do plantel pelo produtor é o controle das inseminações realizadas. Visando isso, o sistema contará com cadastro de inseminações realizadas com informações sobre data e tipo de inseminação (natural ou artificial) facilitando identificar quais inseminações trouxeram mais benefícios para a produção mensal e auxiliar na decisão sobre qual método utilizar em inseminações futuras. Exemplo: com o touro ‘x’ e a vaca ‘y’, a cria gerada obteve uma produção média de 10 litros diários, mas com o touro ‘n’ e a mesma vaca ‘y’, a cria gerada obteve uma produção média de 20 litros diários.

Para uma visualização ampla de suas vacas em lactação, ou seja, em estado de produtividade, o sistema permitirá a visualização do *status* de cada vaca, sendo eles: em lactação, período pré-parto, período pós-parto e em tratamento. Cada tipo de *status* será cadastrado previamente no sistema pelo produtor. Cada animal é

mantido no respectivo *status* pelo período de tempo que o produtor considerar necessário e em decorrência de tratamentos, doenças e outros, já que esse período pode variar por diversos fatores, tais como: falta de vacinação, tipo do rebanho e hábitos do produtor. Uma lactação tem duração normal de 305 dias, porém, em alguns rebanhos mestiços a lactação pode ser menor, algo entre 270 a 290 dias.

4.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta a descrição dos requisitos levantados para o sistema. Os campos código indicados nos cadastros são gerados internamente pelo sistema.

Identificação	Nome	Descrição
RF01	Manter cadastro de vacas	Cadastrar cada uma das vacas contendo os seguintes dados: código, nome, número do brinco, origem, data de nascimento, raça, filiação, observações.
RF02	Manter vacinas	Cadastrar as vacinas em estoque adquiridas pelo produtor contendo: código, descrição, valor, validade e quantidade.
RF03	Manter vacinações	O sistema deverá conter um registro de vacinações para histórico, armazenando o animal que foi vacinado, tipo da vacina, data de vacinação e o numero de doses aplicadas.
RF04	Manter inseminações	As inseminações realizadas serão cadastradas para controle de prenhez. Nesse cadastro deve ser informado se foi realizada inseminação artificial ou natural, o qual serão informados através de um enum.
RF05	Manter touros	Cadastro de touros disponíveis no plantel, informando os dados: nome, brinco, raça.
RF06	Manter sêmen	Cadastro de doses sêmen adquiridas pelo produtor e sua disponibilidade. Será informado no cadastro a raça do touro, valor unitário, data da compra e quantidade.
RF07	Manter ciclo de produção	O sistema controlará o período que uma determinada vaca está “seca”, ou seja, sem produzir e outras situações que possa apresentar. O ciclo de produção é controlado por status como: seca, em lactação, doente, período pós-parto, período pré-parto.
RF08	Manter períodos férteis	Obter o controle de datas em que uma novilha entrou no período fértil e se foi inseminada para o controle de inseminações.
RF09	Manter despesas	O controle dos gastos com o plantel. Gastos como em: ração, silagem, horas de máquinas, ferramentas, funcionários.
RF10	Manter registro de abates/vendas	Registrar abates e vendas do plantel. Os abates são realizados, geralmente, em situações nas quais o animal precisa ser sacrificado em decorrência de acidentes ou doenças, por exemplo,
RF11	Manter doenças	O registro dos tipos de doença.
RF12	Registrar produção	O produtor poderá registrar a produção média de cada ordenha do seu plantel, obtendo-se a produção por período por vaca (se for de seu interesse) ou do plantel como um todo.
RF13	Manter receitas	Cadastrar receitas obtidas com a atividade principal que é a venda do leite ou com atividades secundárias como

		venda de animais.

Quadro 2 – Requisitos do sistema

O diagrama de casos de uso apresentado na Figura 1 contém as funcionalidades essenciais do sistema realizadas pelos seus atores: administrador e produtor. O produtor é responsável pelos cadastros do sistema e registro de despesas e entradas (receitas) e o administrador por manter usuários. Nesses casos de uso manter refere-se a manter os registros de animais, vacinas, vacinações e inseminações realizadas, touos, sêmen, ciclos de produção e períodos férteis, doenças, despesas e receitas, produção, abates e vendas e usuários.

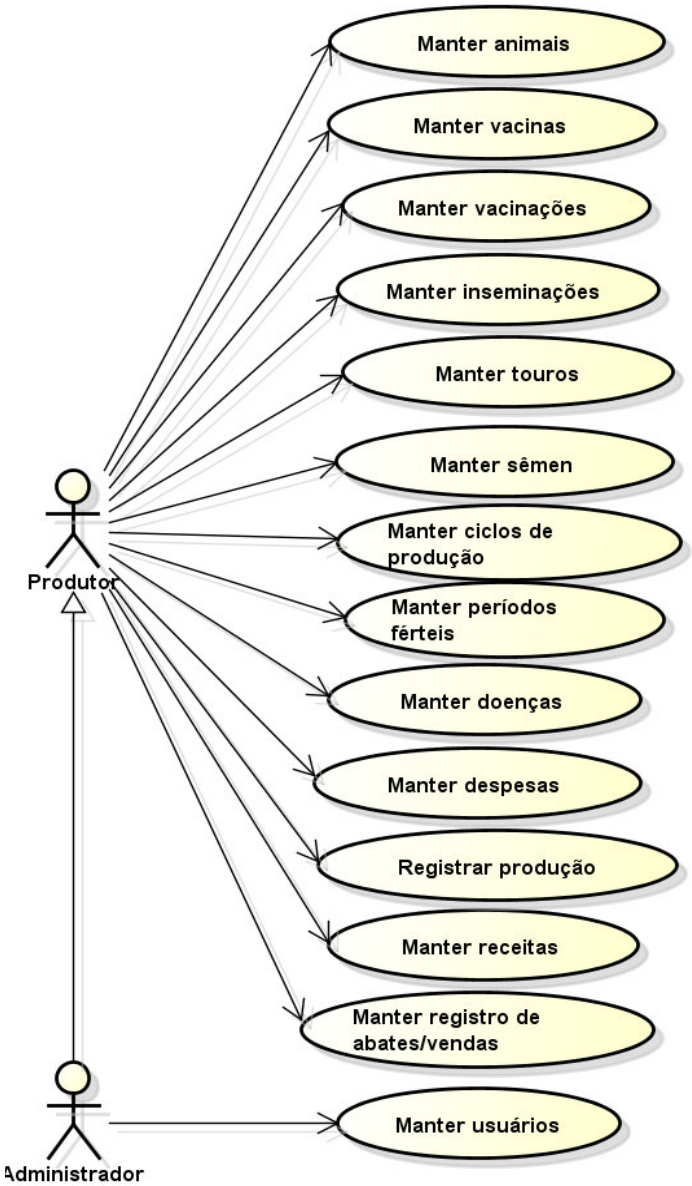

Figura 1 – Diagrama de casos de uso

Os Quadros 3 a 6 apresentam a descrição das ações de inclusão, alteração, exclusão e consulta dos casos de uso manter. No Quadro 3 está a expansão da operação incluir dos casos de uso manter.

<table>
<tr><td colspan="2">Caso de uso:
Incluir (refere-se à operação de inclusão de todos os casos de usos identificados como “manter”).
Descrição:
Inclusão dos dados cadastrais no sistema.
Evento Iniciador:
Ator solicita inclusão de um registro no sistema.
Atores:
Administrador ou produtor.
Pré-condição:
O usuário deve estar autenticado no sistema.
Sequência de Eventos:
1. Ator acessa a tela para realizar o cadastro inserindo as informações necessárias.
2. O sistema insere as informações no banco de dados e informa ao usuário o status do procedimento.
Pós-Condição:
Registro inserido no banco de dados.
Extensões:
Campos obrigatórios faltantes e campos no formato incorreto.</td></tr>
<tr><td>Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td>1. Campos obrigatórios não informados.</td><td>1.1. O usuário deixa de informar dados obrigatórios e clica em salvar.
1.2. O sistema valida que não foram informados todos os campos obrigatórios e exibe mensagem ao usuário sem salvar o registro.
1.3. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.</td></tr>
<tr><td>2. Campos informados em formato incorreto.</td><td>2.1. O usuário informa dados em um formato incorreto e clica em salvar.
2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro.
2.3. O sistema permanece na tela de inclusão mantendo os dados informados anteriormente.</td></tr>
</table>

Quadro 3 - Operação “incluir” dos casos de uso de cadastro

A descrição da operação alterar dos casos de uso manter é apresentada no Quadro 4.

<table>
<tr><td>Caso de uso:
Alterar (refere-se à operação de alteração de todos os casos de usos identificados como “manter”).
Descrição:
Alteração dos dados cadastrais no sistema.
Evento Iniciador:
Ator solicita alteração de um registro no sistema.
Atores:</td></tr>
</table>

<table>
<tr><td colspan="2">Administrador ou produtor.
Pré-condição:
Registro estar incluso no sistema.
Sequência de Eventos:
1. Ator acessa a tela para visualização dos dados do registro.
2. O sistema apresenta o registro selecionado para alteração.
3. Ator altera os dados do registro.
4. O sistema altera as informações no banco de dados e informa ao usuário o status do procedimento.
Pós-Condição:
Registro alterado no banco de dados.
Extensões:
Campos obrigatórios faltantes e campos no formato incorreto.</td></tr>
<tr><td>Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
<tr><td>1. Campos obrigatórios não informados.</td><td>1.1. O usuário exclui dados obrigatórios e clica em salvar.
1.2. O sistema valida que não foram informados todos os campos obrigatórios e exibe mensagem ao usuário sem salvar o registro.
1.3. O sistema permanece na tela de edição mantendo as alterações realizadas.</td></tr>
<tr><td>2. Campos informados em formato incorreto.</td><td>2.1. O usuário altera os dados deixando-os em um formato incorreto e clica em salvar.
2.2. O sistema valida que os dados não estão no formato esperado e exibe mensagem ao usuário sem salvar o registro.
2.3. O sistema permanece na tela de edição mantendo as alterações realizadas.</td></tr>
</table>

Quadro 4 - Operação "alterar" dos casos de uso de cadastro

O Quadro 5 apresenta a descrição da operação excluir dos casos de uso manter.

<table>
<tr><td colspan="2">Caso de uso:
Excluir (refere-se à operação de exclusão de todos os casos de usos identificados como "manter").
Descrição:
Exclusão dos dados cadastrais no sistema.
Evento Iniciador:
Ator solicita exclusão de um registro no sistema.
Atores:
Administrador ou produtor.
Pré-condição:
Registro estar incluso no sistema.
Sequência de Eventos:
1. Ator acessa a tela para exclusão do registro.
2. O sistema exclui as informações no banco de dados e informa ao usuário o status do procedimento.
Pós-Condição:
Registro excluído no banco de dados.
Extensões:
Registro possui vínculo com outros cadastros.</td></tr>
<tr><td>Nome do fluxo alternativo (extensão)</td><td>Descrição</td></tr>
</table>

1. Exclusão de registro com vínculos no sistema	1.1. O usuário clica em excluir um registro que possui vínculos no sistema. 1.2. O sistema verifica que o registro tem vínculos, não o exclui e exibe mensagem de alerta ao usuário.

Quadro 5 - Operação "excluir" dos casos de uso de cadastro

No Quadro 6 está a descrição da operação consultar referente aos casos de uso manter.

Caso de uso:
Consultar (refere-se à operação de consulta de todos os casos de usos identificados como "manter").
Descrição:
Consulta dos dados cadastrais dos registros do sistema.
Evento Iniciador:
Ator solicita consulta de um registro no sistema.
Atores:
Administrador ou produtor.
Pré-condição:
Registro estar incluso no sistema.
Sequência de Eventos:
1. Ator acessa a tela para visualização dos dados do registro.
2. O ator indica os filtros desejados para consulta.
3. O sistema apresenta os dados da consulta ao usuário.
Pós-Condição:
Dados da consulta apresentados ao usuário.

Quadro 6 - Operação "consultar" dos casos de uso de cadastro

O caso de uso registrar produção está detalhado no Quadro 7.

Caso de uso:
Registrar produção
Descrição:
Nesse caso de uso será realizado o controle de produção leiteira. Para isso o produtor deverá informar a quantidade de leite produzida, diariamente no sistema para cada vaca ou a média por animal. Assim, além de se obter a produção mensal total do rebanho, o produtor terá a informação de qual vaca está produzindo menos e provavelmente deverá ser descartada.
Evento Iniciador:
Coleta da quantidade produzida em litros de cada vaca.
Atores:
Produtor.
Pré-condição:
Dados necessários disponíveis.
Sequência de Eventos:
1. Ator seleciona formulário no qual deseja realizar a operação.
2. Sistema apresenta o formulário.
3. Ator realiza a operação desejada: incluir, excluir, consultar, alterar.
4. Sistema verifica se os dados para a operação estão consistentes e realiza a operação.
Pós-Condição:
.Operação de inclusão, exclusão, consulta, alteração.

Nome do fluxo alternativo (extensão)	Descrição
Linha 4: Dados não são válidos.	4.1 No momento de salvar, o sistema faz a verificação e constata que há dados inválidos. É emitida mensagem e retornado para o formulário de avaliação em estado de edição. Retorna ao passo 3 e o caso de uso prossegue com a sequência normal.
Linha 4: Exclusão de registro com dados de tabelas vinculadas ativos.	4.1 Se solicita exclusão de registro que possui tabelas vinculadas com dados ativos, é informada mensagem que não é possível realizar a exclusão. Sistema informa o usuário e permanece em estado anterior à realização da operação na Linha 4.

Quadro 7 – Caso de uso Controle de Produções

A Figura 2 apresenta o diagrama de entidades e relacionamentos que representam o banco de dados da aplicação.

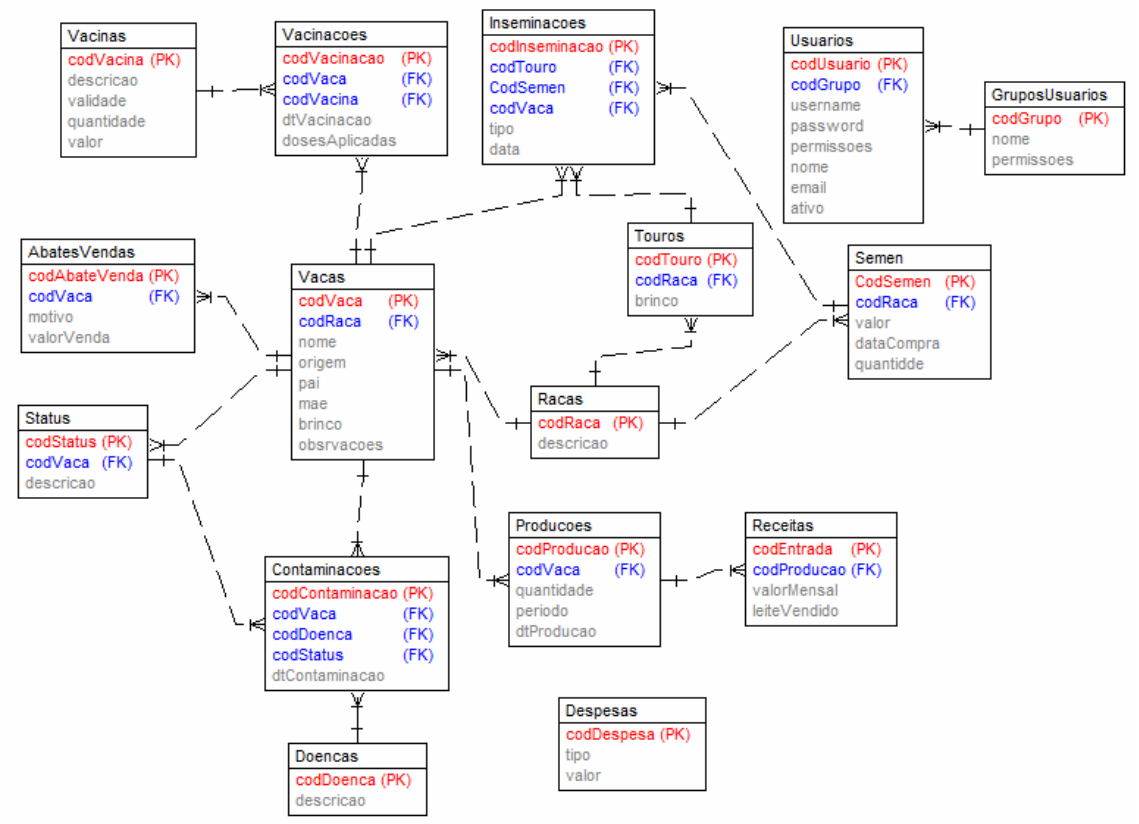

Figura 2 – Diagrama de entidades e relacionamentos do banco de dados

No Quadro 8 estão os campos da tabela vacinações. Para vacinação são utilizadas vacinas e uma vacinação pode ser aplicada em uma vaca.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codVacinacao	Numérico	Não	Sim	Não	
codVaca	Numérico	Não	Não	Sim	Da tabela Vacas
codVacina	Numérico	Não	Não	Sim	Da tabela Vacinas
dtVacinacao	Data	Sim	Não	Não	
dosesAplicadas	Numérico	Sim	Não	Não	

Quadro 8 – Campos da tabela vacinações

O Quadro 9 apresenta as informações da tabela vacinas.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codVacina	Numérico	Não	Sim	Não	
descricao	Texto	Não	Não	Não	
validade	Data	Não	Não	Não	
quantidade	Numérico	Sim	Não	Não	
valor	Numérico	Sim	Não	Não	

Quadro 9 – Campos da tabela vacinas

O Quadro 10 mostra os dados da tabela vacas.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codVaca	Numérico	Não	Não	Sim	Da tabela Vacas
codRaca	Numérico	Não	Não	Sim	Da tabela Racas
Tipo	Texto	Não	Não	Não	
nome	Texto	Não	Não	Não	
origem	Texto	Não	Não	Não	
pai	Texto	Sim	Não	Não	
mae	Texto	Sim	Não	Não	
brinco	Numérico	Não	Não	Não	
observacoes	Texto	Sim	Não	Não	

Quadro 10 – Campos da tabela vacas

O Quadro 11 apresenta a tabela status, que contém informações sobre o período que a vaca se encontra em seu ciclo produtivo.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codStatus	Numérico	Não	Sim	Não	
codVaca	Numérico	Não	Não	Sim	Da tabela Vacas
descricao	Texto	Não	Não	Não	

Quadro 11 – Campos da tabela status

O Quadro 12 apresenta os campos da tabela abates/vendas. Essa tabela possui ligação com a tabela vacas.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codAbateVnda	Numérico	Não	Não	Sim	
codVaca	Numérico	Não	Não	Sim	Da tabela Vacas
valorVenda	Numérico	Sim	Não	Não	
motivo	Texto	Sim	Não	Não	

Quadro 12 – Campos da tabela abates/vendas

O Quadro 13 apresenta os campos da tabela inseminações. Uma vaca pode ser inseminada diversas vezes. Uma inseminação pode ser realizada por sêmen (artificial) ou touro (natural).

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codInseminacao	Numérico	Não	Sim	Não	
codtouro	Numérico	Sim	Não	Sim	Da tabela Touros
codVaca	Numérico	Não	Não	Sim	Da tabela Vacas
codSemen	Numérico	Sim	Não	Sim	Da tabela Semen
tipo	Texto	Não	Não	Não	
data	Data	Não	Não	Sim	

Quadro 13 – Campos da tabela inseminações

O Quadro 14 apresenta os campos da tabela raças.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codRaca	Numérico	Não	Sim	Não	
descricao	Texto	Não	Não	Não	

Quadro 14 – Campos da tabela Raça

O Quadro 15 apresenta os campos da tabela sêmen, que possui ligação com

as tabelas de raças e inseminações.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codSemen	Numérico	Não	Sim	Não	
valor	Numérico	Não	Não	Não	
dataCompra	Data	Não	Não	Não	
quantidade	Numérico	Sim	Não	Não	
codRaca	Numérico	Sim	Não	Sim	Da tabela Racas

Quadro 15 – Campos da tabela sêmen

O Quadro 16 apresenta os campos da tabela doenças.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codDoenca	Numérico	Não	Sim	Não	
descricao	Texto	Não	Não	Não	

Quadro 16 – Campos da tabela doenças

O Quadro 17 apresenta as informações da tabela contaminações, que possuirá ligação com as tabelas vacas, status e doenças.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codContaminacao	Numérico	Não	Sim	Não	
codDoenca	Numérico	Não	Não	Sim	Da tabela Doencas
codVaca	Numérico	Não	Não	Sim	Da tabela Vacas
codStatus	Numérico	Não	Não	Sim	Da tabela Status
dtContaminacao	Data	Sim	Não	Não	

Quadro 17 – Campos da tabela contaminações

Quadro 18 apresenta as informações da tabela produções, que possui ligação com a tabela vacas.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codProducao	Numérico	Não	Sim	Não	
codVaca	Numérico	Não	Não	Sim	Da tabela Vacas
quantidade	Numérico	Não	Não	Não	
periodo	Texto	Não	Não	Não	
dtProducao	Date	Não	Não	Não	

Quadro 18 – Campos da tabela produções

O Quadro 19 apresenta os dados da tabela receitas. Por meio dos dados dessa tabela poderá ser calculada a renda do produtor com a venda do leite. Essa tabela possui ligação com a tabela produções.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codReceita	Numérico	Não	Sim	Não	
codProducao	Numérico	Sim	Não	Sim	Da tabela Producoes
valorMensal	Numérico	Não	Não	Não	
leiteVendido	Numérico	Não	Não	Não	

Quadro 19 – Campos da tabela receitas

O Quadro 20 apresenta as informações da tabela despesas.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codDespesa	Numérico	Não	Sim	Não	
tipo	Texto	Não	Não	Não	
valor	Numérico	Não	Não	Não	

Quadro 20 – Campos da tabela despesas

O Quadro 21 apresenta as informações da tabela usuários.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
codUsuario	Numérico	Não	Sim	Não	
username	Texto	Não	Não	Não	
password	Texto	Não	Não	Não	
nome	Texto	Não	Não	Não	
email	Texto	Não	Não	Não	
ativo	Boolean	Não	Não	Não	
grupo de usuario	Grupo de Usuario	Não	Não		Tabela Grupo de Usuario.

Quadro 21 – Campos da tabela usuario

O Quadro 22 a presenta as informações da tabela grupos de usuarios.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
Id	Numérico	Não	Sim	Não	
nome	Texto	Não	Não	Não	
permissoes	Texto	Não	Não	Não	

Quadro 22 – Campos da tabela grupo de usuários

4.3 APRESENTAÇÃO DO SISTEMA

O sistema possui um leiaute clássico *web*, dividido em três setores. No setor superior da tela é apresentado o usuário que está autenticado no sistema, possibilitando o acesso às configurações da conta, informações do perfil e fazer *logout*. O setor lateral da página é composto pelo menu, no qual o usuário pode acessar os cadastros do sistema, bem como *dashboards* e relatórios. Por fim, no centro são apresentadas as páginas do sistema que o usuário pode acessar.

Na Figura 3 é apresentada a página principal do sistema. Após o usuário autenticar-se ele é redirecionado para a página *home*, que terá as informações atualizadas de sua propriedade. Na parte superior da página, o usuário poderá verificar a produção do mês atual, o número de vacas em seu plantel, o valor de entradas (partindo do cálculo de produção em litros x preço do litro de leite) e o valor de despesas (cálculo de todos os gastos cadastrados pelo usuário, vacinas, despesas gerais, inseminações etc.). Logo abaixo é apresentado um gráfico, com os picos de produção no mês.

Figura 3 - Tela inicial do sistema

Na Figura 4 é apresentado o cadastro de produções. Nessa tela, o usuário poderá visualizar as produções já cadastradas no sistema, podendo realizar a inclusão, exclusão e alteração dos registros.

Para realizar a inclusão de uma nova produção, o usuário deverá clicar no botão "Criar novo". Para alterar um registro de produção, deverá clicar no botão com

o ícone de um lápis e, para excluir um registro, é necessário clicar no botão com o ícone de lixeira. Os referidos ícones estão circulados na Figura 4.

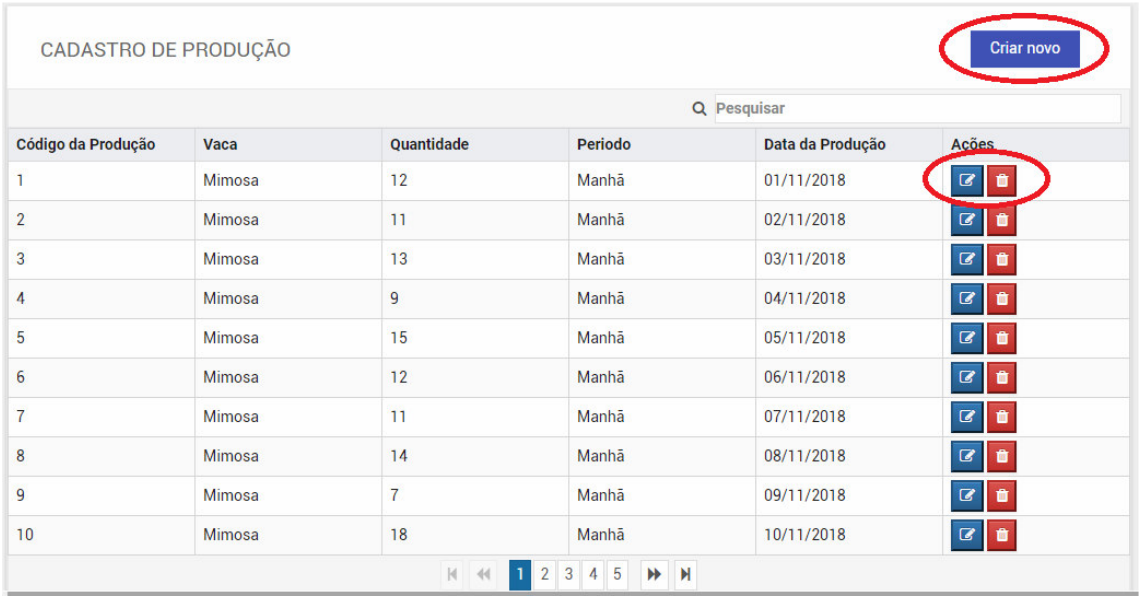

Figura 4 - Tela de cadastro de produção

Ao clicar no botão "Criar novo" é aberto um modal para o usuário. Nesse modal é apresentado um formulário para a inclusão de uma nova produção, juntamente com os botões de salvar e cancelar, como mostra a Figura 5.

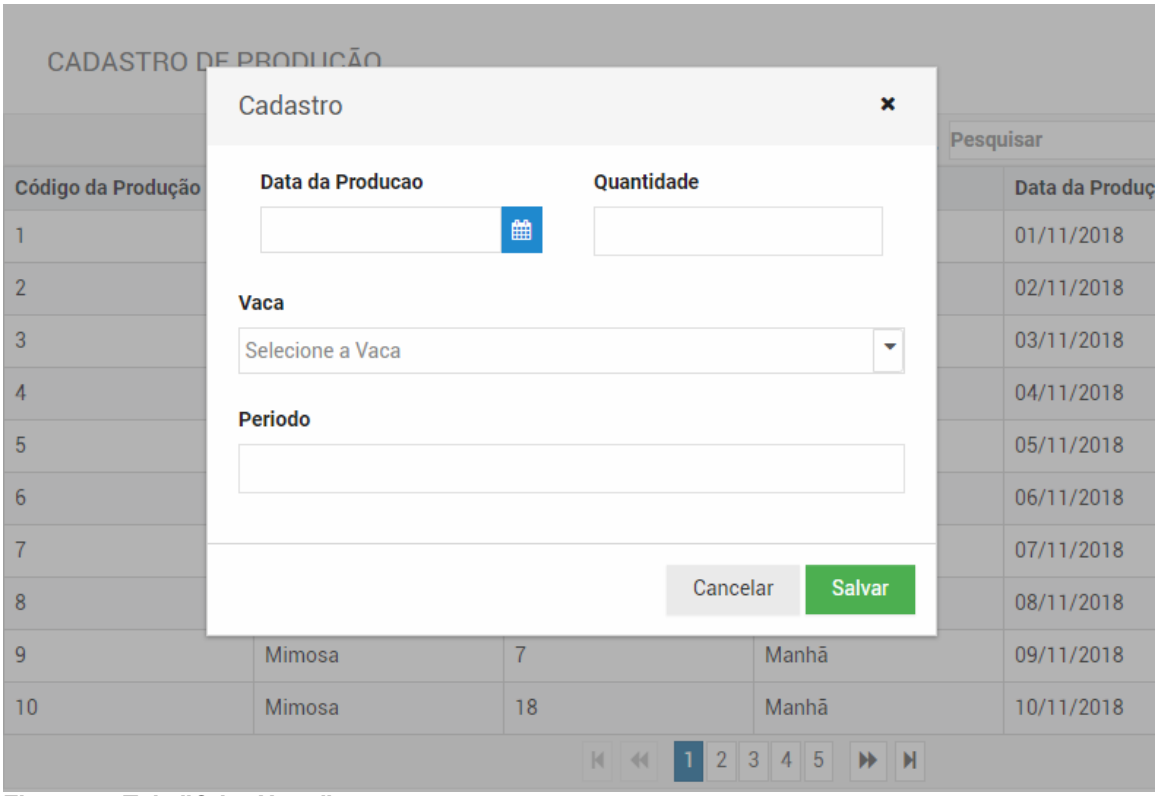

Figura 5 - Tela "Criar Novo"

Se o usuário clicar no botão de "Salvar" sem informar os campos que estão marcados como obrigatórios pelo asterisco (*), o sistema impedirá que ele clique em salvar, desabilitando o botão automaticamente e deixando sublinhado em vermelho os campos que devem ser preenchidos para habilitar o botão de salvar, como pode ser visto na Figura 6. Essa validação é realizada pelo módulo *ReactiveForms* disponibilizado pelo @angular/forms.

Figura 6 - Tela de validação de campos

Para alteração, o usuário deverá clicar no botão com o lápis, que abrirá um *dialog* retornando a tela semelhante à de inclusão, retornando as informações do registro selecionado que estão no banco de dados já preenchidas.

As regras de validação dos campos permanecem as mesmas da inclusão, como ilustra a Figura 7.

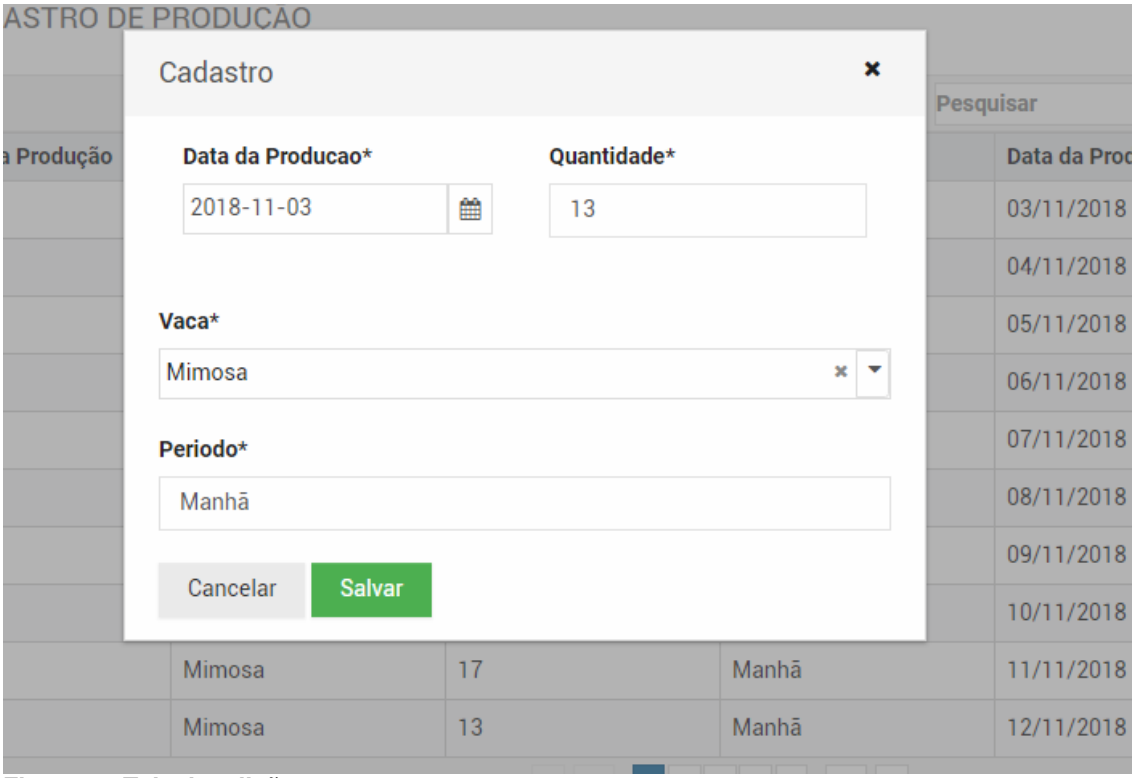

Figura 7 - Tela de edição

Para realizar a exclusão de um registro, basta clicar no botão com o ícone da lixeira e que será apresentado um *dialog*, com uma mensagem de confirmação (Figura 8). Caso confirmada a exclusão, a tabela com os dados cadastrados será automaticamente carregada com a nova listagem e o registro selecionado será excluído do banco de dados.

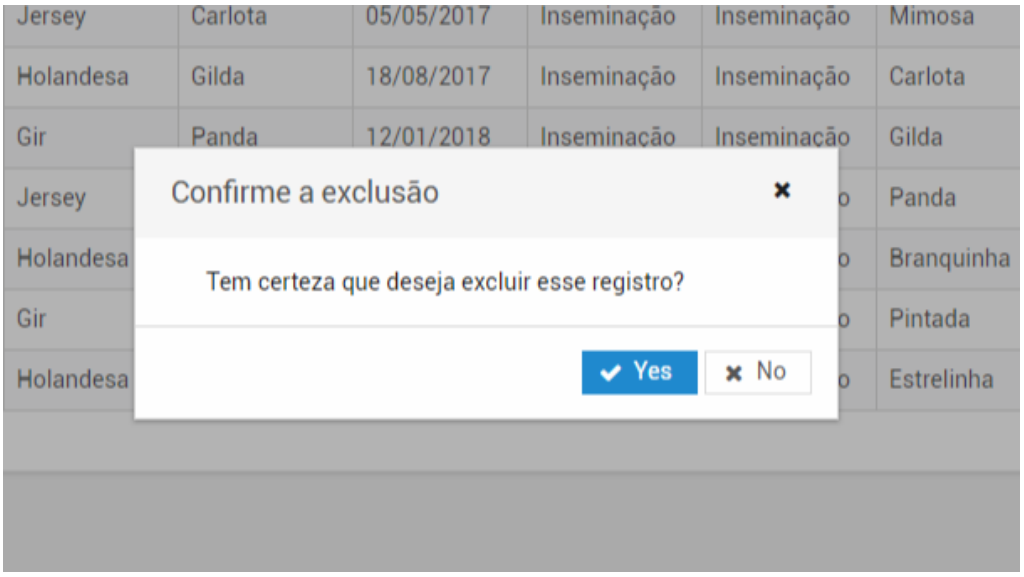

Figura 8 - dialog de confirmação de exclusão

Na Figura 9 é possível verificar a tela de cadastro de usuários. Previamente na instalação do sistema, é disponibilizado o *login* de um usuário Administrador para a pessoa responsável para a inserção dos demais usuários, bem como suas permissões de acessos.

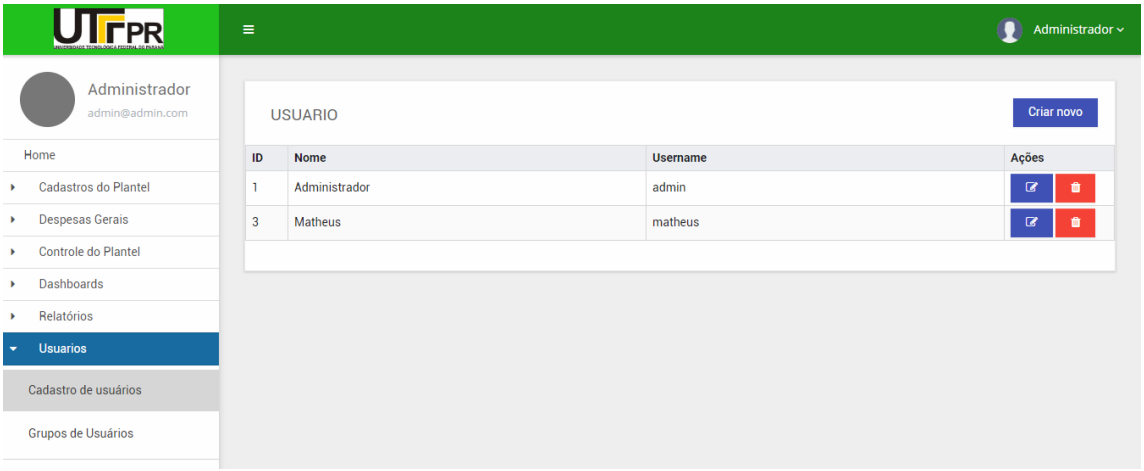

Figura 9 - Tela de cadastros de usuarios

Ao clicar em "Criar Novo", o sistema apresentará um *dialog*, conforme mostra a Figura 10.

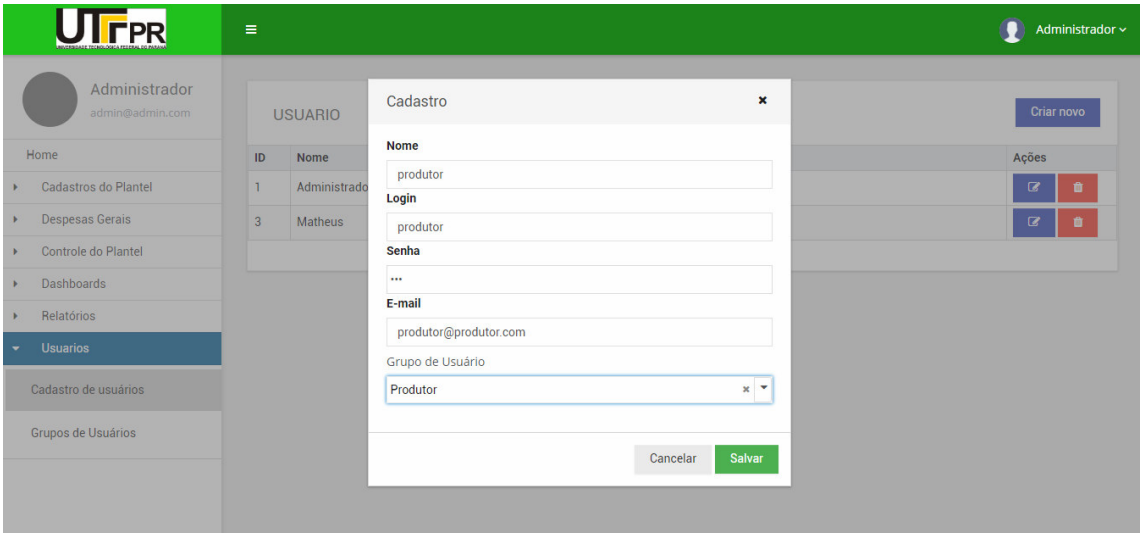

Figura 10 - Cadastrando um novo usuario

Nesse *dialog*, o usuário deverá preencher com o nome, *login*, senha, *email* e grupo de usuário ao qual o usuário pertence.

Importante destacar, que o grupo de usuário é o responsável por definir o que o usuário terá permissão para acessar no sistema. Portanto, para se cadastrar um usuário é necessário ter cadastrado um grupo de usuário previamente.

A Figura 11 mostra a tela de cadastro de Grupo, na qual o administrador do sistema definirá os direitos de acessos aos demais usuários.

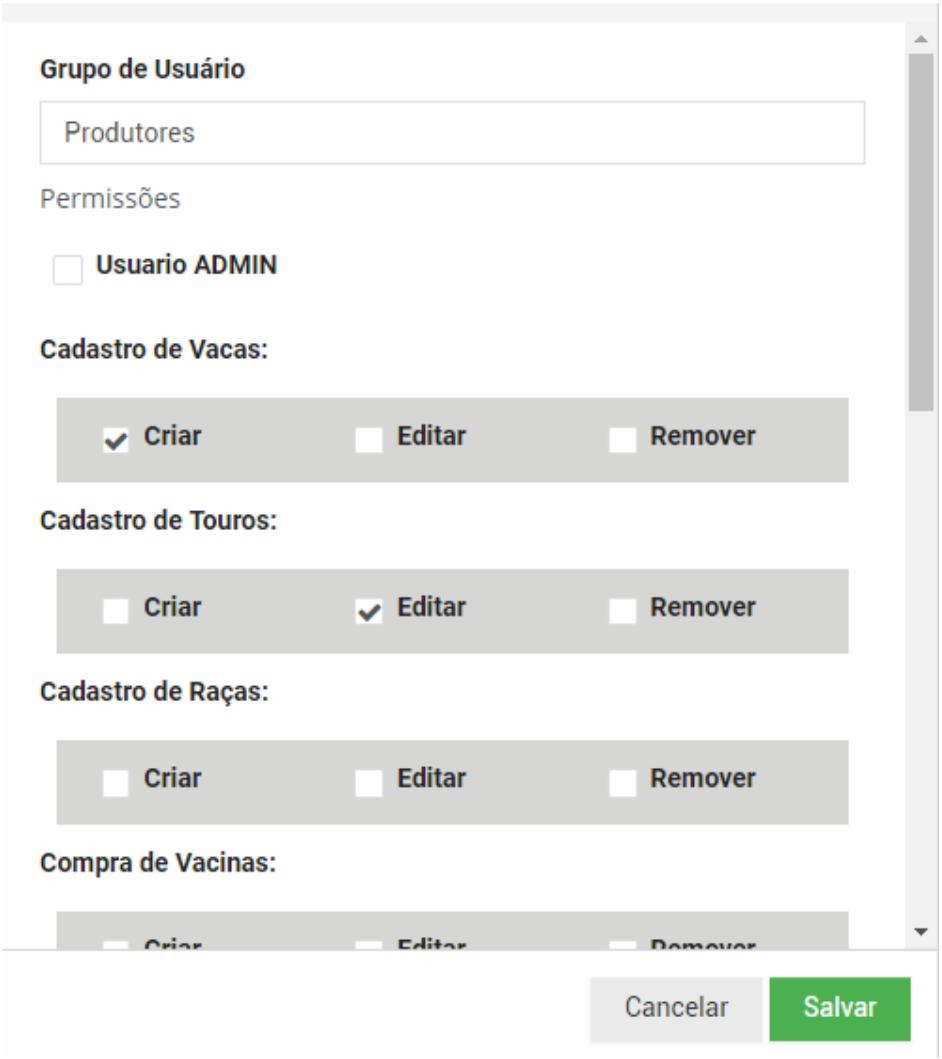

Figura 11 - Permissões do grupo

Nessa tela o administrador definirá quais rotinas do sistema o usuário pertencente aquele grupo poderá acessar e realizar as operações de: criar, editar ou remover.

4.4 IMPLEMENTAÇÃO DO SISTEMA

Na Figura 12 é apresentada a organização do projeto no cliente.

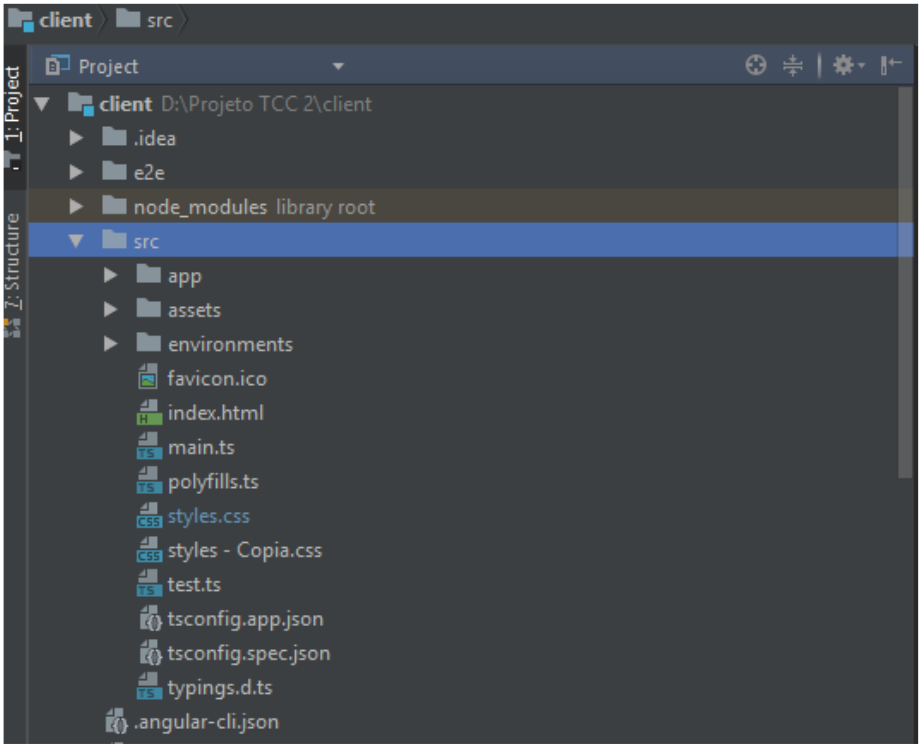

Figura 12 - Projeto Front-End

A Figura 12 apresenta a estrutura padrão de organização do Angular CLI ao criar o projeto. Na pasta *app* ficam todas as classes (*model*, *service* e *controller*).

Na pasta *assets* ficam as imagens e os arquivos JavaScript utilizados. Já na pasta *environments* ficam as configurações da API, como porta na qual está sendo executado o servidor. Fora dessas pastas estão os arquivos de configurações e o index.html, que deverá conter dados de cada importação realizada no projeto.

A Figura 13 mostra como foi organizado o projeto no *back-end*, sendo utilizada a linguagem Java com Spring Boot para desenvolvimento.

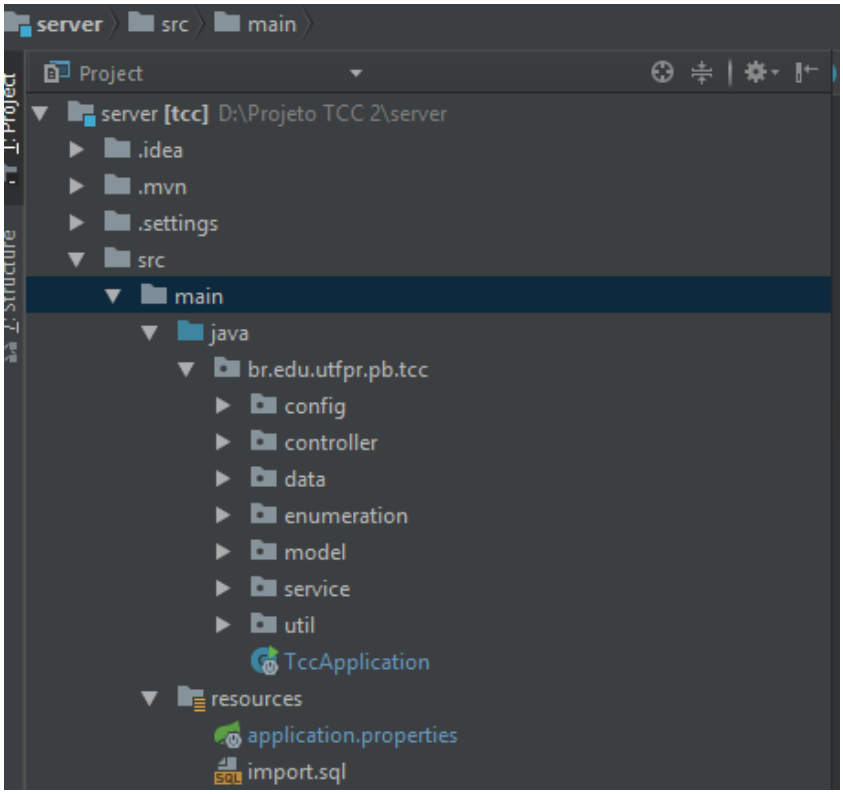

Figura 13 - Estrutura do servidor

As pastas principais da estrutura apresentada na Figura 13 são:

a) Model: todas as classes model do projeto.

b) Data: as classes que estendem do JPA, para realizar a persistência dos dados.

c) Controller: o *controller* de cada classe, que é o responsável por recuperar as requisições do *front* e utilizar as demais classes para salvar, editar e excluir arquivos.

Na Listagem 1 é apresentada a forma de implementação da tela de cadastro de produção.

```
15        <link rel="stylesheet" type="text/css" class=".../styles.css">
 <div _ngcontent-c2="" class="col-lg-12">
    <section _ngcontent-c2="" class="box ">
      <header _ngcontent-c2="" class="panel_header">
        <div class="row">
          <div class="col-sm-10">
            <h4 _ngcontent-c2="" class="title pull-left title1">Cadastro
de Produção</h4>
          </div>
          <div class="col-sm-2">
            <button style="margin-top: 15%" class="btn btn-primary"
(click)="novo()">Criar novo</button>
          </div>
        </div>
```

```
        <p-table #dt [globalFilterFields]="['codProducao', 'dtProducao',
'periodo', 'quantidade', 'codVaca.nome']" [paginator]="true" [rows]="10"
[value]="producoes">
          <ng-template pTemplate="caption">
            <div style="text-align: right">
              <i class="fa fa-search" style="margin:4px 4px 0 0"></i>
              <input type="text" pInputText size="50"
placeholder="Pesquisar" (input)="dt.filterGlobal($event.target.value,
'contains')" style="width:auto">
            </div>
          </ng-template>

          <ng-template pTemplate="header">
            <tr>
              <th>Código da Produção</th>
              <th>Vaca</th>
              <th>Quantidade</th>
              <th>Periodo</th>
              <th>Data da Produção</th>
              <th>Ações</th>
            </tr>
          </ng-template>
          <ng-template pTemplate="body" let-producao>
            <tr>
              <td>{{producao?.codProducao}}</td>
              <td>{{producao?.codVaca?.nome}}</td>
              <td>{{producao?.quantidade}}</td>
              <td>{{producao?.periodo}}</td>
              <td>{{producao?.dtProducao | date:'dd/MM/yyyy'}}</td>
              <td>
                <button  class="btn-primary" (click)="editar(producao)">
                  <i class="fa fa-edit"></i>
                </button>
                <button  class="btn-danger" (click)="remover(producao)">
                  <i class="fa fa-trash"></i>
                </button>
              </td>
            </tr>
          </ng-template>
        </p-table>

      </header>
      <div _ngcontent-c2="" class="content-body">
      </div>
    </section>
 </div>

<p-dialog header="Cadastro"
          [(visible)]="showDialog"
          modal="modal"
          [responsive]="true"
          [width]="500">
  <form [formGroup]="formularioDeProducao" (ngSubmit)="enviarDados()">
  <div clas="row">
    <div class="col-sm-6">
      <div class="form-group" [ngClass]="aplicaCssErro('dtProducao')">
      <label>Data da Producao*</label>
      <p-calendar [(ngModel)]="producaoEdit.dtProducao" dateFormat="yy-mm-
dd" formControlName="dtProducao" [showIcon]="true"></p-calendar> <span
style="margin-left:35px"></span>
```

```
      </div>
    </div>
    <div class="col-sm-6">
      <div class="form-group" [ngClass]="aplicaCssErro('quantidade')">
        <label for="quantidade">Quantidade*</label>
        <input type="number"
               required="required"
               id="quantidade"
               name="quantidade"
               formControlName="quantidade"
               class="form-control"
               [(ngModel)]="producaoEdit.quantidade">
      </div>
    </div>
  </div>

  <div class="row">
    <div class="col-sm-12">
      <div class="form-group" [ngClass]="aplicaCssErro('codVaca')">
        <label for="nome">Vaca*</label>
        <p-dropdown styleClass="width: 100%" formControlName="codVaca"
[options]="vacas" [(ngModel)]="producaoEdit.codVaca"
placeholder="Selecione a Vaca" optionLabel="nome" [showClear]="true"></p-
dropdown>
      </div>
    </div>
  </div>

  <div class="row">
    <div class="col-sm-12">
      <div class="form-group" [ngClass]="aplicaCssErro('periodo')">
        <label for="periodo">Periodo*</label>
        <input type="text"
               required="required"
               id="periodo"
               name="periodo"
               class="form-control"
               formControlName="periodo"
               [(ngModel)]="producaoEdit.periodo">
      </div>
    </div>
  </div>
  <p-footer>

    <button type="button" class="btn btn-default" (click)="showDialog =
false">Cancelar</button>
    <button  type="submit" [disabled]="!formularioDeProducao.valid"
class="btn btn-success">Salvar</button>
  </p-footer>
  </form>
</p-dialog>
```

Listagem 1 – Código do cadastro de produção

Conforme a Listagem 1 foi desenvolvida uma página HTML que receberá as informações do *controller*. Para exibir a tabela de cadastros realizados é utilizada a *tag <p-table>* do PrimeNG. Nessa *tag,* foi setado o *[globalFilterFields]* que é um atributo próprio da *tag <p-table>.* Essa *tag* faz com que os elementos do

controller se tornem visíveis e possam ser filtrados dentro da tabela. Para isso, foi adicionada uma *tag* <ng-template> com o atributo *pTemplate* recebendo "caption" que faz com que a tabela crie um filtro de pesquisa. Nesse filtro, o usuário poderá digitar o texto a ser procurado pelo método *dt.filterGlobal($event.target.value, 'contains')* no qual "dt" é identificado declarado ainda no p-table.

A tabela é composta por três divisões, sendo que o atributo *pTemplate* define cada divisão. Na primeira divisão é passado o valor de "*caption*" fazendo a tabela receber um filtro de pesquisa.

Em seguida foi criada uma nova *tag* <ng-template> que receberá como atributo um pTemplate com o valor de "header" que montará o cabeçalho da tabela de cadastros, nela os valores de cada *<th>* são passados fixamente.

Em seguida foi criada uma nova *tag <ng-template>* recebendo o atributo pTemplate com o valor de "body". Nela por meio do *let-producao* a tabela receberá os valores do *controller* dinamicamente para cada <tr> e <td>.

Para setar os valores, o Angular possui desde sua primeira versão uma sintaxe por meio da qual é passada uma variável entre chaves, por exemplo {{producao?.codProducao}}. Essa variável receberá os valores do *array* que foi definido no *let*.

Logo abaixo da tabela que apresenta os dados cadastrados, ainda na Listagem 1 é apresentado como foi desenvolvido o *dialog* que será responsável pelas operações de inclusão e alteração em conjunto com o *controller*.

Para o desenvolvimento do *dialog* foi utilizada a *tag* <p-dialog> também do PrimeNG. Dessa *tag* é interessante destacar os seguintes atributos:

a) *header* - recebe como valor o título do *dialog*.

b) *[(visible)]* - por meio dessa *tag* será controlado, utilizando o *controller*, quando o *modal* será visivel ou não, trabalhando o showDialog com *true* e *false*.

Após criar o *dialog*, dentro dele foi utilizado um *<form>* passando como atributo um *[formgroup]="formularioDeProducao".* Por meio desse atributo é possível trabalhar com o formulário pelo *controller*, utilizando a biblioteca *ReactiveFormsModule* do *@angular/forms.*

Dentro do formulário foram utilizadas algumas outras *tags* do PrimeNG que são interessantes destacar, como:

a) <p-dropdown> - responsável por criar um *input* do tipo *select*, que possibilitará o usuário selecionar a opção desejada no cadastro. Para isso, é

necessário informar no atributo [options] o *array* com todos os itens que deverão ser disponibilizados para seleção e o atributo [NgModel] que receberá como valor, a variável que deverá ser salva no *controller*.

b) <p-calendar> - essa *tag*, retornará um *input* do tipo calendário para o usuário selecionar a data que deverá ser enviada para o *controller*. Da mesma forma que o *dropdown*, essa *tag* possui um [NgModel] que fará esse controle por meio de uma variável.

Para trabalhar em conjunto com a parte visual do HTML é necessário um *controller*, que é apresentado na Listagem 2.

```
@Component({
  templateUrl: './producao.component.html',
  styleUrls: ['./producao.component.css']
})
export class ProducaoComponent implements OnInit {

  producoes: Producao[];
  vacas: Vaca[];
  formularioDeProducao: FormGroup;

  showDialog = false;
  producaoEdit = new Producao();
  msgs: Message[] = [];

  constructor(private producaoService: ProducaoService,private
loginService: LoginService,
              private vacaService: VacaService,
              private fb: FormBuilder) {
  }
```

Listagem 2 – Controller do cadastro de produção

A partir do Angular 2, para definir que um *controller* alimentará uma determinada página HTML, basta utilizar a anotação *@Component* na classe que será o *controller*, no caso do exemplo a *ProducaoComponet.* Nessa classe é definido tudo o que será utilizado na tela, como variáveis, métodos, *services* e outros.

Na Listagem 3 são apresentadas todas as funções utilizadas na tela de cadastro de produção.

```
ngOnInit(): void {
  console.log(this.producoes);
  this.producaoService.findAll().subscribe(e => {
    this.producoes = e;
  });
  console.log(this.producoes);

  this.vacaService.findAll().subscribe(e => this.vacas = e);
  this.criarFormularioDeProducao();
}
```

```
findAll() {
  this.producaoService.findAll().subscribe(e => this.producoes = e);
  console.log(this.producoes);
}

criarFormularioDeProducao() {
  this.formularioDeProducao = this.fb.group({
    dtProducao: ['', Validators.compose([Validators.required])],
    quantidade: ['', Validators.compose([Validators.required])],
    codVaca: ['', Validators.compose([Validators.required])],
    periodo: ['', Validators.compose([Validators.required])]
  });
}

novo() {
  this.showDialog = true;
  this.producaoEdit = new Producao();
}

enviarDados() {
  if(this.formularioDeProducao.valid){
    this.producaoService.save(this.producaoEdit).subscribe(e => {
        this.producaoEdit = new Producao();
        this.findAll();
        this.showDialog = false;
        this.msgs = [{severity:'sucess', summary:'Confirmado',
detail:'Registro salvo com sucesso'}];
      },
      error => {
        error.error.errors.forEach(error => {
          this.msgs = [{severity:'error', summary:'Erro', detail:'O campo
' + error.field.toUpperCase() + ' ' + error.defaultMessage}];
        });
      }
    );
  } else {
      console.log('Formulario Invalido');
      Object.keys(this.formularioDeProducao.controls).forEach(campo => {
       console.log(campo);
       const controle = this.formularioDeProducao.get(campo);
       controle.markAsDirty();
      });
  }
}

verificaValidTouched(campo){
  return !this.formularioDeProducao.get(campo).valid &&
this.formularioDeProducao.get(campo).touched;
}

aplicaCssErro(campo){
  return{
    'has-error': this.verificaValidTouched(campo),
    'has-feedback': this.verificaValidTouched(campo)
  }
}

editar(producao: Producao) {
  this.producaoEdit = producao;
  this.showDialog = true;
```

```
  this.msgs = [{severity:'sucess', summary:'Confirmado', detail:'Registro
alterado com sucesso'}];
}

remover(producao: Producao) {
  this.producaoService.delete(producao.codProducao).subscribe(() => {
    this.findAll();
  });
}
```

Listagem 3 – Funções da tela de cadastro de produção

Conforme é apresentado na Listagem 3, a seguir a descrição da finalidade de cada função utilizada:

a) ngOnInit() - essa função é uma implementação da *interface* OnInit, nela foi definido o que será processado quando realizada a requisição da página. No caso do exemplo, foi realizada uma requisição por meio do método *findAll()* da *producaoService* e *vacaService*, retornando a lista a ser apresentada na tela.

b) findAll() - nessa função é realizada uma requisição por meio do producaoService que foi instanciado no *constructor*.

c) *criarFormularioDeProducao()* - essa função é utilizada para definição dos campos do formulário, por meio da instrução *fb.group()* (sendo "fb" uma instância chamada no *constructor* do FormBuilder), que também fará o controle da validação do formulário, por meio do "Validators.required".

d) *novo()* - utilizado para criar um novo *model* de produção.

e) *enviarDados()* - essa função é chamada no onSubmit do formulário de cadastro, nela é utilizado um *if(this.formularioDeProducao.valid)* que verifica se o formulário é válido. Se sim, ele é salvo no banco, se não ele apresenta uma mensagem de erro a usuário e não salva no banco.

f) *verifcaValidTouched()* - essa função verifica se um campo do formulário foi alterado, o qual a função AplicaCssErro() fará uso para alertar o usuário que o campo precisa ser preenchido corretamente.

g) *editar()* - cria um novo *model* de produção para edição e seta o showDialog para *true*, fazendo com que o *dialog* seja apresentado em tela.

h) *remover()* - exclui do banco o registro selecionado, em seguida no *callback* é realizado um findAll() para atualizar a lista apresentada em tela.

Além do *controller*, é necessário também em uma aplicação Angular, o *service* (responsável pela instanciação das requisições ao *back-end*) e o *model*, que deve corresponder a mesma classe criada no *back-end* pelo Spring.

Nas Listagens 4 e 5 são apresentados o *service* e o *model* do cadastro de produção.

```
@Injectable()
export class ProducaoService extends CrudService<Producao, number> {

  constructor(http: HttpClient) {
    super(environment.api + '/producao', http);
  }
}
```

Listagem 4 – Service do cadastro de produção

```
import {Vaca} from '../vaca/vaca';

export class Producao {
  codProducao: number;
  codVaca: Vaca;
  periodo: string;
  quantidade: number;
  dtProducao: Date;
}
```

Listagem 5 – Model do cadastro de produção

Para que seja possível a inserção dos dados, exclusão e alteração no banco foi utilizado o *framework* Spring Boot que funciona de uma forma bem semelhante ao *front-end*.

Na Listagem 6 é apresentado como funciona a classe *controller* pelo Spring, no cadastro de produção.

```
@RestController
@RequestMapping("producao")
public class ProducaoController extends CrudController<Producao, Long> {

    @Autowired
    private ProducaoService producaoService;

    @Override
    protected CrudService<Producao, Long> getService() {
        return producaoService;
    }
}
```

Listagem 6 – Classe controller

Com a anotação @RestController é definido para o Spring que essa classe será o *controller* da aplicação *Representational State Transfer* (REST), e com a @RequestMapping é passado com qual URL será realizado a requisição.

Para facilitar o desenvolvimento foram criadas duas classes como padrão de *Create, Read, Update e Delete* (CRUD), uma para o *controller* que é a CrudController, sendo passado como parâmetro a classe *model* e o tipo do seu Id e outra para o *service*, que é a CrudService, na qual é passada a classe *model* e o tipo

da variável Id.

Para o controle de permissões de usuarios, foi utilizado o método *hasRole()* em cada classe do projeto, na qual sempre irá carregar as permissões do usuário autenticado.

No servidor foi criado um *enumeration* com cada permissão possível no sistema, conforme a Listagem 7.

```
public enum Role {

    ADMIN,

    USUARIO_CRIAR,
    USUARIO_EDITAR,
    USUARIO_REMOVER,

    VACA_CRIAR,
    VACA_EDITAR,
    VACA_REMOVER,

    TOURO_CRIAR,
    TOURO_EDITAR,
    TOURO_REMOVER,

    RACA_CRIAR,
    RACA_EDITAR,
    RACA_REMOVER,

    VACINAS_CRIAR,
    VACINAS_EDITAR,
    VACINAS_REMOVER,

    SEMEN_CRIAR,
    SEMEN_EDITAR,
    SEMEN_REMOVER,

    DESPESAS_CRIAR,
    DESPESAS_EDITAR,
    DESPESAS_REMOVER,

    PRODUCAO_CRIAR,
    PRODUCAO_EDITAR,
    PRODUCAO_REMOVER,

    DOENCAS_CRIAR,
    DOENCAS_EDITAR,
    DOENCAS_REMOVER,

    CONTAMINADA_CRIAR,
    CONTAMINADA_EDITAR,
    CONTAMINADA_REMOVER,

    VACINACOES_CRIAR,
    VACINACOES_EDITAR,
    VACINACOES_REMOVER
}
```

Listagem 7 – Enumeration das permissões

A definição das *enumerations* simplificou o processo de desenvolvimento, ficando a classe que controla os acessos do perfil do usuário da forma apresentada na Listagem 8.

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Perfil {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    //mapear a lista
    @ElementCollection(fetch = FetchType.EAGER)
    //gravar as pemissoes como string
    @Enumerated(EnumType.STRING)
    private Set<Role> permissoes;
}
```

Listagem 8 – Permissões do grupo

Feito isso, o servidor está pronto para receber do cliente as definições cadastradas de acesso por usuário.

Na Listagem 9 é apresentado um trecho do código HTML de como foi desenvolvido a tela de cadastro de grupo de usuário, que utiliza como *value* dos *checkbox*, cada *enumeration* esperado pelo servidor.

```
<p-dialog header="Cadastro"
          [(visible)]="showDialog"
          modal="modal"
          [responsive]="true"
          [width]="500">

  <div class="row">

    <div class="col-sm-12">
      <div class="form-group">
        <label for="nome">Grupo de Usuário</label>
        <input type="text"
               id="nome"
               name="nome"
               class="form-control"
               [(ngModel)]="perfilEdit.nome" required>

        <h5>Permissões</h5>
        <div class="ui-g" style="width:300px;margin-bottom:10px">
          <div class="ui-g-8"><p-checkbox name="admin" value="ADMIN"
label="Usuario ADMIN" [(ngModel)]="perfilEdit.permissoes"></p-
checkbox></div>
        </div>

        <label>Cadastro de Vacas:</label>
        <div class="row" style="margin: 2%; padding:2%; background-color:
```

```
#d6d6d5">
          <div class="col-sm-4"><p-checkbox name="vaca" value="VACA_CRIAR"
label="Criar" [(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
          <div class="col-sm-4"><p-checkbox name="vaca"
value="VACA_EDITAR" label="Editar"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
          <div class="col-sm-4"><p-checkbox name="vaca"
value="VACA_REMOVER" label="Remover"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
        </div>

        <label>Cadastro de Touros:</label>
        <div class="row" style="margin: 2%; padding:2%; background-color:
#d6d6d5">
          <div class="col-sm-4"><p-checkbox name="touro"
value="TOURO_CRIAR" label="Criar" [(ngModel)]="perfilEdit.permissoes"></p-
checkbox></div>
          <div class="col-sm-4"><p-checkbox name="touro"
value="TOURO_EDITAR" label="Editar"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
          <div class="col-sm-4"><p-checkbox name="touro"
value="TOURO_REMOVER" label="Remover"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
        </div>

        <label>Cadastro de Raças:</label>
        <div class="row" style="margin: 2%; padding:2%; background-color:
#d6d6d5">
          <div class="col-sm-4"><p-checkbox name="raca" value="RACA_CRIAR"
label="Criar" [(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
          <div class="col-sm-4"><p-checkbox name="raca"
value="RACA_EDITAR" label="Editar"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
          <div class="col-sm-4"><p-checkbox name="raca"
value="RACA_REMOVER" label="Remover"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
        </div>

        <label>Compra de Vacinas:</label>
        <div class="row" style="margin: 2%; padding:2%; background-color:
#d6d6d5" >
          <div class="col-sm-4"><p-checkbox name="raca"
value="VACINAS_CRIAR" label="Criar"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
          <div class="col-sm-4"><p-checkbox name="raca"
value="VACINAS_EDITAR" label="Editar"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
          <div class="col-sm-4"><p-checkbox name="raca"
value="VACINAS_REMOVER" label="Remover"
[(ngModel)]="perfilEdit.permissoes"></p-checkbox></div>
        </div>
```

Listagem 9 – Trecho HTML da tela de cadastro

Como mostra a Listagem 9, ao usuário salvar o cadastro será enviado pelo ngModel um objeto chamado "perfilEdit.permissoes" ao *controller*, definindo quais campos foram preenchidos e devem ser salvos no banco.

Após salvar no banco os direitos de acesso de cada usuário, o sistema já está

pronto para receber essas configurações. Para isso, como relatado anteriormente, é utilizado o método *hasRole()* do *service* de *login* em cada *controller*, como mostra a Listagem 10.

```
hasRole(role: string): boolean {
  return this.loginService.hasRole(role);
}
```

Listagem 10 – Service do login

Recebendo essas configurações do usuário, basta definir com um NgIf, as permissões que o usuário deve possuir para visualizar determinado elemento da tela, conforme é demonstrado na Listagem 11, o código que foi utilizado para controlar os acessos na tela de cadastro de vacas.

```
<header _ngcontent-c2="" class="panel_header">
  <div class="row">
    <div class="col-sm-10">
    <h4 _ngcontent-c2="" class="title pull-left title1">Cadastro de
Vacas</h4>
    </div>
    <div class="col-sm-2">
     <button style="margin-top: 15%" class="btn btn-primary"
*ngIf="hasRole('VACA_CRIAR') || hasRole('ADMIN')"
(click)="novo()">Criar novo</button>
    </div>
  </div>
  <p-table [value]="vacas">
    <ng-template pTemplate="header">
      <tr>
        <th>Código da Vaca</th>
        <th>Raça</th>
        <th>Nome</th>
        <th>Data de Nascimento</th>
        <th>Origem</th>
        <th>Pai</th>
        <th>Mae</th>
        <th>Brinco</th>
        <th>Observacoes</th>
        <th>Ações</th>
      </tr>
    </ng-template>
    <ng-template pTemplate="body" let-vaca>
      <tr>
        <td>{{vaca?.codVaca}}</td>
        <td>{{vaca?.codRaca?.descricao}}</td>
        <td>{{vaca?.nome}}</td>
        <td>{{vaca?.dtNascimento | date:'dd/MM/yyyy'}}</td>
        <td>{{vaca?.origem}}</td>
        <td>{{vaca?.pai}}</td>
        <td>{{vaca?.mae}}</td>
        <td>{{vaca?.brinco}}</td>
        <td>{{vaca?.observacoes}}</td>
        <td>
            <button *ngIf="hasRole('VACA_EDITAR') ||
hasRole('ADMIN')" class="btn-info" (click)="editar(vaca)">
              <i class="fa fa-edit"></i>
            </button>
            <button  *ngIf="hasRole('VACA_REMOVER') ||
hasRole('ADMIN')" class="btn-danger" (click)="remover(vaca)">
              <i class="fa fa-trash"></i>
            </button>
        </td>
      </tr>
    </ng-template>
  </p-table>
```

Listagem 11 – Permissões do usuário para acessar permissões da tela

Utilizando o **Ngif="hasRole(VACA_CRIAR)" || hasRole('ADMIN')"* no botão de "Criar novo", por exemplo, somente usuários pertencentes à um grupo que possua esse privilégio poderão visualizar esse botão ou que esteja marcado como ADMIN, visto que se a condição for *false* no ngIf, o componente nem é carregado no HTML.

5 CONCLUSÃO

A modelagem do sistema *web* para gerenciamento de um plantel de gado leiteiro, bem como a implementação do sistema foram realizados de acordo com o que foi planejado, atendendo os objetivos definidos para o trabalho. No desenvolvimento do sistema foram utilizados *frameworks* como o *Angular 6* e *Spring Boot* que possibilitaram a caracterização de interface rica.

O objetivo principal de apresentar ao produtor quais cabeças de seu plantel se destacam em produção para auxiliá-lo a definir estratégias em diferentes épocas do ano para sempre estar com uma produção de qualidade, foi alcançado. O sistema também permite que o produtor cadastre e visualize os gastos que ele está tendo em vacinas, inseminações e despesas em geral e compara com a entrada em dinheiro que ele conseguiu com a produção realizada, permitindo ao produtor tomar ações e definir metas para aumentar seu plantel.

Como o sistema é para ambiente *web*, além de possuir recursos que facilitam o produtor, como apresentar as cotações do dia, o sistema pode ser executado em *locahost*, sem conexão com a internet, não interferindo na utilização do sistema.

Por já ter trabalhado com o *Spring Boot* o desenvolvimento do servidor foi de fácil implementação e tornou essa etapa muito produtiva. Além de que, a ideia desse *framework* é justamente agilizar o processo de desenvolvimento, visto que reaproveita tecnologias já utilizadas.

Já para a parte de *front-end* foi utilizado um *framework* com o qual o autor deste trabalho não possuía grande afinidade, visto que sempre utilizou o AngularJS, uma versão totalmente diferente ao seus sucessores. Pelo fato de o AngularJS ser uma versão descontinuada foi optado pelo Angular 6.e. Apesar do desafio de adequar-se ao Angular, o desenvolvimento foi muito prático e foi possível obter suporte em fóruns e *blogs* especializados.

Também para o *front-end*, foi optado a utilizar o PrimeNG para a criação dos formulários e menus. É uma biblioteca limpa e rica de componentes com entendimento relativamente simples e com códigos bem organizados. Foi identificado que ela não supriu as necessidades em alguns momentos, como, por exemplo, em suas *tables,* foi necessário pesquisar em fóruns para verificar como implementar essa funcionalidade, pois sua documentação não contempla essa

explicação.

Por fim, o projeto atingiu o seu objetivo, as tecnologias e as ferramentas utilizadas desde a sua modelagem até a implementação do sistema, sofreram algumas mudanças, mas foram bem aplicadas. A ideia de utilizar algo novo foi bem interessante, visto que a comunidade está disponibilizando material que auxiliam no desenvolvimento, deixando o processo menos complexo e muito mais ágil.

Futuramente para desenvolvimento pode ser acrescentadas melhorias no projeto como relatórios em Excel e em *Portable Document Format* (PDF), bem como relatórios em modo de gráficos, com *widgets* personalizados de acordo com as necessidades do produtor.

REFERÊNCIAS

ALLAIRE, Jeremy. Macromedia Flash MX—A next-generation rich client. Tech. rep., **Macromedia**, March 2002. Disponível em: <http://underpop.online.fr/f/flash/richclient1.pdf>. Acesso em: 20 nov. 2016.

CENTRO DE ESTUDOS AVANÇADOS EM ECONOMIA APLICADA. **Relatório PIB Agro-Brasil**. Disponível em: <http://www.cepea.esalq.usp.br/comunicacao/Cepea_PIB_BR_maio16.pdf>. Acesso em: 06 set. 2016.

DINCTURK, Mustafa Emre; JOURDAN, Guy-Vincent; BOCHMANN, Gregor V.; ONUT, Iosif Viorel. A Model-Based Approach for Crawling Rich Internet Applications. **ACM Transactions on the Web**, v. 8, n. 3, June 2014, p.19:2-19:39.

DUHL, Joshua. White Paper: Rich Internet Applications. Tech. rep., **IDC**, November 2003. Disponível em: <Duhl, Joshua. White Paper: Rich Internet Applications>. Acesso em: 20 nov. 2016.

ECOAGRO. **O Agronegócio no Brasil**. Disponível em: <http://www.ecoagro.agr.br/agronegocio-brasil/>. Acesso em: 15 nov. 2016.

MARTÍNEZ-RUIZ, Francisco J. The Triad-based Design of Rich User Interfaces for Internet Applications. **ACM**, 2010, p. 345-348.

PANDURINO, Andrea; BOLCHINI; Davide; MAINETTI, Luca; PAIANO, Roberto. Ri**ch-IDM: Extending IDM to Model Rich Internet Applications. In:** iiWAS2010, 2010, p. 147-154.

ROUBI, Sarra; ERRAMDANI, Mohammed; MBARKI, Samir. Modeling and Generating Graphical User Interface for MVC Rich Internet Application using a Model Driven Approach. **Computer and Information Science**, v. 9.n. 2, 2016, p. 1-6.

SECRETARIA DE ESTADO DA AGRICULTURA E DO ABASTECIMENTO. **Análise da conjuntura agropecuária**: leite – ano 2014. Disponível em: <http://www.agricultura.pr.gov.br/arquivos/File/deral/Prognosticos/bovinocultura_leite_14_15.pdf>. Acesso em: 06 set. 2016.

VELASCO, Carlos A.; DENEV, Dimitar; STEGEMANN, Dirk; MOHAMAD, Yehya. **A Web Compliance Engineering framework to support the development of accessible Rich Internet Applications. In:** 17th International World Wide Web Conference, 2008, p. 45-49.