CENTRO UNIVERSITÁRIO DE ANÁPOLIS – UniEVANGÉLICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

JOÃO MAURO CARDOSO LOPES

SISTEMA WEB PARA O AUXÍLIO NA CRIAÇÃO DE FRANGOS CAIPIRAS

ANÁPOLIS
2018-01

CENTRO UNIVERSITÁRIO DE ANÁPOLIS – UniEVANGÉLICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

JOÃO MAURO CARDOSO LOPES

SISTEMA WEB PARA O AUXÍLIO NA CRIAÇÃO DE FRANGOS CAIPIRAS

Trabalho de Conclusão de Curso II apresentado ao Curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Orientador: Prof. Kléber Silvestre Diogo

ANÁPOLIS
2018-01

RESUMO

O proprietário da chácara NC possui uma criação de frango e carece de dados relativos ao custo de sua produção. Seus registros são feitos em cadernos e muitas vezes esses dados são perdidos, dificultando a possibilidade de investigação de informações importantes para tomadas de decisões assertivas em relação ao custo de seus produtos e o andamento da sua criação de frangos. Este estudo teve como objetivo geral desenvolver uma aplicação web para registro dos gastos e recursos referente a cada lote de produção de frango. Para que o objetivo dessa pesquisa fosse alcançado o primeiro passo realizado foi a coleta de informações referente ao problema com base em entrevista e estudo bibliográfico. Posteriormente a modelagem dos dados com diagramas da UML e prototipagem do sistema. A próxima etapa consistiu em codificar e implantar o software. Para produção dos artefatos foi utilizado o modelo iterativo e incremental, onde foi possível obter retorno do cliente em todas as etapas do desenvolvimento. A linguagem de programação Java foi utilizada para o desenvolvimento do back-end com apoio do framework Spring. Já para o desenvolvimento do front-end foi utilizado o framework angular. O resultado foi desse trabalho foi o desenvolvimento de um software capaz de gerir os custos lotes de produção de frango utilizando técnicas de engenharia de software. O cliente verificou uma melhora significativa na sua gestão já que o sistema facilitou a rastreabilidade dos custos e despesas de cada lote produzido além de cálculos automatizados que viabilizaram a rapidez na obtenção das informações.

ABSTRACT

The owner of the NC farm has a chicken breeding and miss data on the cost of his production. Their records are made in notebooks and often this data is lost, making it difficult to investigate important information for assertive decision making about the cost of their products and the progress of their broiler breeding. This study had as general objective to develop a web application to record the expenses and resources related to each batch of chicken production. In order to achieve the objective of this research, the first step was to collect information about the problem based on interviews and bibliographic studies. Later the modeling of the data with diagrams of the UML and prototyping of the system. The next step was to code and deploy the software. For the production of the artifacts it was used the iterative and incremental model, where it was possible to obtain customer feedback at all stages of development. The Java programming language was used for the development of the backend with support of the Spring framework. Already for the development of the front end was used the Angular framework. The result of this work was the development of a software capable of managing costs chicken production batches using software engineering techniques. The customer noticed a significant improvement in its management since the system facilitated the traceability of costs and expenses of each batch produced in addition to automated calculations that enabled the speed in obtaining the information.

LISTA DE ILUSTRAÇÕES

Figura 1 - Modelo Incremental 23
Figura 2 - Diagrama de Caso de Uso do Sistema 28
Figura 3 - Diagrama de Classe do Sistema 30
Figura 4 - Representação da Arquitetura do Projeto 31
Figura 5 - Tela de criação do projeto Spring no Eclipse 32
Figura 6 - Configuração do application.properties 33
Figura 7 - Arquivo pom.xml com algumas dependências 33
Figura 8 - Classe entidade Lote 34
Figura 9 - Classe LoteController 35
Figura 10 - Classe LoteService 36
Figura 11 - Classe LoteRepository 37
Figura 12 - Enum Status 38
Figura 13 - Classe LotesModule 39
Figura 14 - Lote Routes Module 40
Figura 15 - Classe entidade Lote Front End 41
Figura 16 - Component FormLoteComponent 42
Figura 17 - Service LoteService 43
Figura 18 - Tela para exportação do arquivo war 46
Figura 19 - Configuração do Banco de Dados no heroku 47
Figura 20 - Arquivo application-prod.properties 47
Figura 21 - Arquivo Procfile 48
Figura 22 - Tela de Login do Sistema 49
Figura 23 - Tela de Listagem dos Lotes 49
Figura 24 - Tela de Cadastro de Lotes 50
Figura 25 - Tela de Visualização do Relatório do Lote 51

TABELAS

Tabela 1- Descrição dos Requisitos Não Funcionais..... 27
Tabela 2 - Descrição do Caso de Uso 28

SUMÁRIO

1. Introdução 10
2. Referencial Teórico 13
   2.1 Criação de Frango 13
   2.2 Software no Agronegócio 14
   2.3 Engenharia de Software 15
      2.3.1 Processo de Software 15
      2.3.2 Modelo de Processo de Software 15
      2.3.3 Modelo de Processo Incremental 15
   2.4 Engenharia de Requisitos 16
      2.4.1 Levantamento de Requisitos 16
      2.4.2 Requisitos Funcionais 16
      2.4.3 Requisitos Não Funcionais 16
   2.5 UML 17
      2.5.1 Diagrama de Caso de Uso 17
      2.5.2 Diagrama de Classe 17
   2.6 Projeto de Arquitetura de Software 17
      2.6.1 Arquitetura MVC 18
   2.7 Java 18
   2.8 JPA e Hibernate 19
   2.9 Spring Framework 19
   2.10 Angular 20
   2.11 PostgreSQL 21
   2.12 Heroku 21
3. Metodologia 22
4. Desenvolvimento 25
   4.1 Diagrama de Caso de Uso do Sistema 27
   4.2 Diagrama de Classe do Sistema 30
   4.3 Arquitetura do projeto 31
   4.4 Criação do Back End Spring Rest 32
   4.4.1 Criação do projeto Spring boot no Eclipse 32
   4.4.2 Configuração do projeto 33

4.4.3 Model 34
4.4.4 Controller 35
4.4.5 Service 36
4.4.6 Repositoy 37
4.4.7 Security 37
4.4.8 Constantes 38
4.5 Criação do Front End Angular 38
4.5.1 Criação do projeto Angular 38
4.5.2 Module 39
4.5.3 Model 41
4.5.4 Component 42
4.5.5 Service 43
4.6 Deploy 46
4.6.1 Build do projeto Front End Angular 46
4.6.2 Criação do arquico war 46
4.6.3 Deploy no Heroku 47
4.7 O sistema avisoftware Apresentação do Sistema 48
4.7.1 Tela de Login 49
4.7.2 Tela de Listagem de Lotes 49
4.7.3 Tela Cadastro de Lotes 50
4.7.4 Tela Relatório do Lote 51
5. Considerações finais 52
6. Referências Bibliográficas 53
APÊNDICE A – Protótipos do Sistema 56
1. Protótipo – Listagem dos Lotes 56
2. Protótipo – Cadastro dos Lotes 56
3. Protótipo – Menu do Lotes 57
4. Protótipo – Listagem das Alimentações 57
5. Protótipo – Cadastro das Alimentações 58
6. Protótipo – Listagem das Vacinas 58
7. Protótipo – Cadastro das Vacinas 59
8. Protótipo – Listagem das Medicações 59
9. Protótipo – Cadastro das Medicações 60
10. Protótipo – Listagem dos Clientes 60
11. Protótipo – Cadastro dos Clientes 61

12. Protótipo – Listagem dos Fornecedores 61
13. Protótipo - Cadastro dos Fornecedores 62
14. Protótipo – Tela Administrador 63
15. Protótipo – Tela Cadastro Venda 64
16. Protótipo – Tela Cadastro Despesa 64

1. Introdução

O proprietário da fazenda NC tem como uma de suas principais rendas da propriedade uma criação de frango caipira. Muitas vezes fica difícil obter o controle preciso dos gastos de sua produção. A sua criação se inicia com a aquisição das aves com aproximadamente 5 dias. A partir de então inicia o processo de contabilização de seus gastos com rações, medicamentos, vacinas e outros.

Todos esses fatores vão interferir no valor com que os frangos serão vendidos. Os gastos vão aumentar com o tempo até que seja a hora certa para iniciar a venda de seu produto. Será que o produtor sabe realmente quanto está tendo de lucro e qual é o preço que seu produto deve ser vendido para suprir seus investimentos? Com uma entrevista com o proprietário da fazenda, descobriu-se que há poucos registros em relação seus gastos durante o processo de criação de frangos.

A falta de precisão sobre os custos das aves e recursos gastos durante o processo de criação podem comprometer o cálculo da rentabilidade de cada lote produzido, dificultando também a capacidade do produtor de tomar futuras decisões devido à falta de informações coletadas. Seus dados estão sendo armazenados em cadernos o que dificulta a rastreabilidade das informações que muitas vezes são perdidas.

Um dos segmentos que se tem mostrado promissor da avicultura alternativa é a criação de frangos caipiras, pois, além de agregar valor ao produto e utilizar um sistema de criação que valoriza o bem-estar animal, serve tanto para pequenos, médios e grandes produtores (MORAIS, Jonas et al., 2015).

Dados sobre a criação de frangos no brasil é que cerca de 91,1% das criações não fazem nenhum registro zootécnico, 73% não possuem instalações adequadas, 71,42% não utilizam incubadora artificial, 64,28% possuem incidência de aparecimento de doença e mortes, 37,5% possuem baixo índice de vacinação, 17,85% não se preocupam com o vazio sanitário e 8,92% apenas se preocupam em receber orientações técnicas para a criação de aves (ROCHA, L .et al., 2016).

O Brasil é um dos maiores produtores agropecuários do mundo e conforme a ONU, será o maior exportador de alimentos do mundo na próxima década. Para que os empreendimentos obtenham maior lucro e qualidade no produto final faz necessário o uso de tecnologias que possam apoiar a gestão dos seus negócios para liquidar gastos e aumentar a produtividade (PORTAL, 2016).

Os empresários rurais necessitam de melhor gerenciamento de suas atividades, precisando de cada vez mais embasamento teórico, técnico e de informações sobre o seu

negócio para tomadas de decisões. Com a evolução dos cenários da economia, o produtor deve buscar ferramentas que lhe forneça resultados rápidos, práticos e com alto teor de assertividade para gerir sua propriedade rural (SOUZA, S. 2013).

A tecnologia de informação tem papel relevante na competividade do agronegócio brasileiro pois cada vez mais, esse setor se desloca para a inovação de gestão, tecnológica e organizacional. A utilização de tecnologia de informação influencia as atividades de gestão e produção ao facilitar o acesso, o armazenamento e a disseminação de informações que favorecem a tomada de decisões (MENDES, C .et al., 2011).

O setor do agronegócio em relação a adoção de tecnologias de informação é um pouco mais lento e menos intensivo do que outros setores como o de serviço, comércio e indústria, porém já se iniciou-se o seu uso e disseminação nessa nova sociedade (MENDES, C .et al., 2011).

A adoção de TI no agronegócio foi impulsionada por dois fatores. O primeiro foi a demanda, pois as organizações do setor necessitaram de tornarem mais eficaz seus processos e atividades. O segundo foi a oferta já que empresas de tecnologia da informação tanto de hardware, software microeletrônica, automação, internet e telecomunicação identificaram o setor como possível mercado em expansão (MENDES, C .et al., 2011).

As tecnologias de administração e gestão torna os produtores melhores administradores e gestores pois proporciona organização, controle, redução de custo, melhoria da qualidade de processos e produtos, e potencializa a competitividade no mercado. (MENDES, C .et al., 2011).

O gerenciamento dos custos proporciona ao empresário rural uma visão geral sobre o seu negócio, como gastos, quanto está ganhando em cada criação, qual seu real valor de negócio esperado e alcançado, qual o melhor momento de venda e compra de sua produção, ou seja, propicia um cálculo mais preciso do custo de sua produção (SOUZA, S. 2013).

As atividades dos produtores se tornam um risco quando não há o controle do que se gasta ou o que se recebe. Podendo os mesmos fazer dívidas desnecessárias em momentos críticos e acabar perdendo seus ganhos de produtividade (SOUZA, S. 2013).

Para uma gestão eficiente é muito importante fazer registros das atividades realizadas pois servirão de indicadores podendo assim evitar dívidas desnecessárias, aumento do custo do produto, redução de lucros e perda do capital investido. Esse registro proporcionará ao produtor informações para que ele realize sua análise financeira e identifique seus prejuízos e lucros (SOUZA, S. 2013).

Onde existe processos bem definidos, há também o uso de TI. A gestão e o desenvolvimento dos negócios recebem o apoio da internet e de outras tecnologias. Os dados

armazenados devem ser confiáveis e dar sentido a algo do interesse de alguém. A informações são a base para o processo de tomada de decisões (SOUZA, S. 2013).

Com base nesse contexto, o foco desse trabalho foi construir um produto de software que atende-se a gestão da criação de frangos o qual tem como objetivo registrar os lucros, os gastos, as vendas, os dados dos lotes produzidos, os fornecedores e os clientes para facilitar a vida do produtor na rastreabilidade das informações de seu processo de produção auxiliando nas tomada de decisões importantes para a manutenção e crescimento do seu negócio.

O cliente decidiu por uma aplicação Web, para que as informações possam ser registradas usando internet 3g do celular ou a partir de qualquer lugar que tenha internet. A propriedade rural está em fase de aquisição internet via satélite.

O principal objetivo desse trabalho foi desenvolver uma aplicação web para apoio ao controle de gastos e recursos de uma criação de frangos caipiras. Para que isso fosse possível fez se necessário alguns objetivos específicos como o levantamento de requisitos relacionados a produção de frangos, visando obter informações sobre as principais atividades do processo de criação das aves, a utilização do modelo iterativo e incremental para o apoio ao processo de desenvolvimento de software, a análise e projeto de software utilizado os recursos da UML, a elaboração de protótipos e a codificação do sistema utilizando tecnologia web Java, Spring e Angular.

2. Referencial Teórico

2.1 Criação de Frango

A inovação tecnológica é a principal ferramenta tanto para os grandes quanto para os pequenos produtores agropecuários sobreviverem no mercado. Porém muitas vezes para o pequeno produtor isso torna assustador, pois para se adaptar a esse novo modelos de mercado é necessário investimento em capital financeiro e em conhecimento para aquisição e absorção das novas tecnologias (BUAINAIN et al., 2014).

A cadeia produtiva avícola é um exemplo de como a inovação tecnológica pode contribuir para o sucesso de uma atividade e seu contínuo desenvolvimento. A interdependência dos agentes da cadeia produtiva contribui para o funcionamento dos mecanismos dos vários segmentos propiciando fornecimento de carne de frango à população de forma barata e de qualidade (BUAINAIN et al.,2014).

Uma cadeia produtiva que representa bem a lógica da modernização no agronegócio como forma de sobrevivência em mercados altamente competitivos é a cadeia avícola. O mercado brasileiro e o mercado mundial de frangos têm evoluído de forma acentuada. Uma das cadeias produtivas de destaque no Brasil é a de frangos em função da utilização de sistemas modernos de planejamento, coordenação da cadeia produtiva e do uso de tecnologias (SARTIN, 2016).

Ao se analisar a escala de produção mundial da carne de frango entre os anos 2000 e 2014 verifica-se um aumento de aproximadamente 40%, passando de 68,03 para 102,48 milhões de toneladas. Em relação ao crescimento percentual ano a ano nota-se um crescimento médio anual de 3,8% (SARTIN, 2016).

Atualmente os Estados Unidos é a maior produtor de frango, em 2015 este país produziu 17.966 mil toneladas de carne de frango. O segundo maior produtor é o Brasil, produzindo 13.146 mil toneladas, a China ocupou o terceiro lugar no ranking de produção, com 13.025 mil toneladas. Ao se comparar a produção de 2015 em relação à produção de 2014 destes três países, têm-se que houve um crescimento de 4,13% e 3,59% para Estados Unidos e Brasil respectivamente (SARTIN, 2016).

Conforme dados IBGE, dentre os Estados produtores do Brasil destacam-se o Paraná, Santa Catarina, Rio Grande do Sul e São Paulo. Sendo estes responsáveis por mais de 70% da produção nacional. O Estado de Goiás ocupa a sexta posição no ranking da produção de carne de frango e com taxa de crescimento inferior aos Estados da região Sul. Em relação às exportações, a ABPA informa que o Estado de Goiás é o quinto colocado no ranking, na ordem, Paraná, Santa Catarina, Rio Grande do Sul e São Paulo (SARTIN, 2016).

2.2 Software no Agronegócio

A informática, a microeletrônica e as telecomunicações são a base da tecnologia de informação, sendo responsáveis pelas transformações nos modelos de produção e acumulação atuais. Software, portais para agricultura, dispositivos eletrônicos para armazenamento de informações, canais de televisão e estrutura de telecomunicações fazem parte da Tecnologia de Informação e Comunicação (OLIVEIRA et al., 2011).

A nova geração de trabalhadores e/ou proprietários rurais se sentem mais confortáveis com a utilização da informática e estão buscando informatizar suas propriedades e processos produtivos (OLIVEIRA et al., 2011).

As empresas de pequeno porte ou microempresas são os principais ofertantes de soluções e softwares para o agronegócio. Nelas incluem universidades, instituições de produção rural, empresas produtoras de insumos e software-houses. Os produtores rurais, empresários da agroindústria, técnicos, cooperativas agrícolas, universidades, instituições de pesquisa e desenvolvimento, serviços de apoio a micro e pequenas empresas, governo e órgãos de extensão rural são os principais demandantes de software agropecuário (OLIVEIRA et al., 2011).

A maior parte da demanda de softwares para agronegócio são soluções simples, a quais, necessitam de suporte e constante evolução do produto e proporcionam um perfil de software específico. As tecnologias de gestão servem de apoio as atividades administrativas, de acompanhamento e gerenciamento de atividades produtivas, tais como, sistema contábeis e financeiros, de controle de estoque, manutenção de equipamentos, logística, sistema de suporte a decisão, modelagem e otimização de produção, sistema de controle e gestão de rebanhos (ZAMBALDE, et al., 2011).

A rastreabilidade é um tema de grande interesse para os produtores rurais, pois permite que tenham um controle efetivo sobre sua produção, podendo fornecer certificados que garantam a origem e a segurança da carne, bem como uma melhor gestão e conhecimento de sua propriedade. A quantidade de dados, o conjunto de atividades, o controle, a necessidade de cálculos e processamento rápidos, o risco, torna se necessário o uso de aplicações direcionadas a fazendas, cooperativas, agroindústria e envolvem praticamente todos os níveis e setores organizacionais (ZAMBALDE, 2011).

Administradores e gestores que participam diretamente ou indiretamente de processo produtivo se beneficiam com o uso de aplicações voltadas a administração e gestão pois estão em busca de controle, redução de custos, agregação de valor a processos e produtos e aumento da competitividade. Dentro do contexto administrativo e gerencial coletar, armazenar, recuperar

e distribuir dados exige que os sistemas sejam projetados adequadamente para objetivos e necessidades organizacionais sejam atendidos (ZAMBALDE, et al2011).

2.3 Engenharia de Software

A engenharia de software permite que profissionais desenvolvam softwares de qualidade com apoio de ferramentas, métodos e processos bem definidos (PRESSMAN, 2011).

O desenvolvimento de softwares profissionais exige informações adicionais além do código do programa e a engenharia de software fornece técnicas que apoiam a etapa de especificação, projeto e até mesmo a etapa de evolução de programas. Caso algum outro engenheiro queira alterar ou até mesmo entender melhor o funcionamento de um programa a documentação do sistema servirá de guia, pois conterá partes da estrutura e de como o software foi desenvolvido (SOMMERVILLE, 2013).

2.3.1 Processo de Software

O processo de software é o conjunto de atividades, ações e tarefas necessária para desenvolver um software de qualidade. É importante seguir passos previsíveis na elaboração de um sistema ou produto que garanta um resultado de alta qualidade. Esses passos que devem ser seguidos é denominando de processo de software (PRESSMAN, 2011).

Devido à complexidade de um modelo de processo de software ideal, cada organização geralmente desenvolve seu próprio processo de software e sempre há espaço para a constante melhorias desses processos. Dependendo do escopo e das regras do sistema define-se o processo de software mais adequado para a sua construção (SOMMERVILLE, 2013).

2.3.2 Modelo de Processo de Software

As principais atividades metodológicas contidas no modelo de processo de software são: comunicação, planejamento, modelagem, construção e entrega. O acompanhamento de controle do projeto, a administração de riscos, a garantia da qualidade, o gerenciamento das configurações, revisões técnicas e outros são um conjunto de atividades de apoio ao processo de software (PRESSMAN, 2011).

2.3.3 Modelo de Processo Incremental

Quando há a necessidade de fornecimento de algumas funcionalidades iniciais para o cliente e após esse fornecimento expandir e melhorar essas funcionalidades em novas versões do software, utiliza-se o modelo incremental. Na utilização do método incremental, geralmente,

a primeira entrega é um produto essencial. Isto significa que os requisitos básicos foram atendidos, porém ainda existem outras funcionalidades complementares algumas conhecidas e outras desconhecidas que ainda não foram entregues (PRESSMAN, 2011).

O produto essencial quando entregue é testado pelo cliente passando por uma avaliação detalhada. Após a avaliação e o feedback do cliente é desenvolvido o planejamento do próximo incremento. Esse planejamento já leva em consideração as modificações encontradas no produto essencial para adequar as necessidades do cliente e a adição de novas de funcionalidades. Após cada incremento esse processo é repetido até que o produto completo seja produzido (PRESSMAN, 2011).

2.4 Engenharia de Requisitos

Para entender a necessidade do cliente a engenharia de requisitos fornece mecanismos apropriados para analisar a viabilidade, negociar uma solução razoável, especificar uma solução sem ambiguidade, validar a especificação e gerenciar essas necessidades à medida que são transformadas um software (PRESSMAN, 2011).

2.4.1 Levantamento de Requisitos

A resolução de problemas, elaboração, negociação e especificação são os elementos os elementos do levantamento de requisitos. Os interessados identificam o problema, propõe elementos da solução, negociam diferentes abordagens e especificam um conjunto de requisitos preliminares da solução (PRESSMAN, 2011).

O alicerce de todo desenvolvimento de software é o levantamento de requisitos. Para seguir um padrão corretamente documentado qualquer alteração ou implantação de uma nova funcionalidade deve ser registrada na documentação para facilitar a detecção de erros e a manutenção do sistema (SOMMERVILLER, 2011).

2.4.2 Requisitos Funcionais

Define as funcionalidades que o sistema terá considerando as entradas e saídas de dados, comportamento com entradas específicas e muitas vezes descreve o que o sistema não irá disponibilizar. Os requisitos funcionais vão ser construídos de acordo com o tipo de software que está sendo desenvolvido, quem são os usuários do sistema e qual a política de negócio adotada pela empresa (SOMMERVILE. 2011).

2.4.3 Requisitos Não Funcionais

Define as limitações em que o sistema irá desempenhar considerando o sistema com um todo. Algumas dessas limitações podem ser restrições de tempo de resposta, no processo de

desenvolvimento do produto, normas impostas pela empresa, limitações de orçamento, integração com outros sistemas, legislação de privacidade, etc (SOMMERVILE, 2011).

2.5 UML

UML (Unified Modeling Language – linguagem de modelagem unificada) é uma linguagem padrão utilizada para descrever ou documentar projeto de software. Artefatos de um sistema de software podem ser visualizados, especificados, construídos e documentados através dos diagramas da UML Esses diagramas facilitam o trabalho dos desenvolvedores pois ajudam no entendimento e na especificação do sistema (PRESSMAN, 2011)

O principal objetivo da UML é fornecer diferentes visões do sistema, modelando o sistema em diferentes aspectos, através dos recursos oferecidos pelos complementos dos diagramas. Quando se utiliza diversos diagramas é possível detectar falhas, o que diminui a ocorrência de erros futuros (GUEDES, 2011).

2.5.1 Diagrama de Caso de Uso

Define uma visão externa geral das funcionalidades que o sistema irá apresentar, sem se preocupar em como essas funcionalidades serão implementadas, servindo de auxílio para a compreensão e identificação dos requisitos do sistema. O diagrama de caso de uso ajuda a documentar, visualizar e especificar as funcionalidades desejadas pelo usuário (GUEDES, 2011).

Esse diagrama costuma ser utilizado em reuniões iniciais com o cliente como forma de apresentar o comportamento do sistema, facilitar a compreensão dos usuários, detectar possíveis falhas na especificação e verificar se os requisitos foram entendidos (GUEDES, 2011).

2.5.2 Diagrama de Classe

O diagrama de classe fornece uma visão estática da organização das classes definindo sua estrutura lógica. O foco desse diagrama é permitir que sejam visualizadas as classes do sistema com seus atributos, métodos, relações e como elas compartilham suas informações (GUEDES, 2011).

2.6 Projeto de Arquitetura de Software

O projeto da arquitetura trata de tomada de decisões da estrutura do programa com a responsabilidade de abstrair a representação de informação e de sequências de processamento. Define os componentes e a estrutura de dados necessários para construir o software. A arquitetura permite analisar se o projeto está atendendo os requisitos, considerar uma

arquitetura alternativa caso haja uma alteração no projeto na fase inicial e reduzir riscos durante a construção do sistema. (PRESSMAN, 2011).

2.6.1 Arquitetura MVC

A arquitetura Modelo-Visão-Controlador (MVC, model-view-controller) é utilizada em aplicações web que separa a funcionalidade interface do usuário da e do conteúdo de informações. A camada de modelo contém o conteúdo e a lógica de processamento específicos à aplicação, inclusive todos os objetos de conteúdo e a lógica de processamento específicos à aplicação, todos os objetos de conteúdo, acesso a fontes de dados e toda funcionalidade de processamento específica para a aplicação (PRESSMAN, 2016).

A visão contém todas as funções específicas à interface e possibilita a apresentação do conteúdo e lógica de processamento e toda a funcionalidade de processamento exigida pelo usuário. O controlador gerencia o acesso ao modelo e a visão e coordena o fluxo de dados entre eles. Em uma aplicação web a visão é atualizada pelo controlador com dados do modelo baseado nas informações fornecidas pelos usuários (PRESSMAN, 2016).

Em aplicações complexas, que enviam uma série de dados para o usuário, o desenvolvedor frequentemente necessita separar os dados (model) da interface (view). Utilizando MVC, alterações feitas na interface não afetarão a manipulação dos dados e estes poderão ser reorganizados sem alterar a interface do usuário. O MVC resolve esse problema por meio da separação das tarefas de acesso aos dados e a lógica do negócio da apresentação e da interação com o usuário. Isso é feito por meio da inserção de um componente entre model e wiew. Esse componente é o controller. (KIST, 2012).

2.7 Java

A Sun Microsystems financiou um projeto de pesquisa corporativa interna no qual resultou em uma linguagem baseada em C++, inicialmente chamada de Oak em homenagem a uma árvore de carvalho. Então, em um encontro da equipe da Sun em uma cafeteria sugeriu-se o nome Java, pois era o nome da cidade de origem de um tipo de café importado consumido por eles (DEITEL, 2010).

A linguagem Java é considerada simples porque permite o desenvolvimento de aplicativos para diferentes sistemas operacionais e arquiteturas de hardware, sem que o programador tenha que se preocupar com detalhes de infra-estrutura. Dessa forma, o programador consegue desempenhar seu trabalho de uma forma mais produtiva e eficiente. (MENDES, 2009).

Java é uma linguagem orientada a objetos, sendo assim, a maior parte dos elementos de um programa Java são objetos. Os programas em Java consistem em partes chamadas classes. As classes possuem atributos e métodos que realizam tarefas e retornam informações quando as tarefas são concluídas. É possível criar o programa em partes. A linguagem Java possui ricas coleções de classes contidas nas bibliotecas de classe Java, que são conhecidas como Java API (Application Programming Interfaces) (DEITEL, 2010).

Java é utilizada para desenvolvimento de sistemas de grande porte, programas Web, aplicativos voltados celulares pagers e PDA e para muitos outros propósitos. O compilador Java converte o código-fonte Java em bytecodes que são executados pela Java Virtual Machine. Então independente do sistema operacional se o mesmo tiver a maquina virtual do Java o programa poderá ser executado (DEITEL, 2010).

2.8 JPA e Hibernate

O JPA (Java Persistence) permite a persistência objeto-relacional com banco de dados relacionais. O Hibernate utilizado para a implementação do JPA, transforma as requisições de feitas por meio das classes em comandos SQL que são enviados para o banco de dados. Alguns recursos oferecidos são gerenciamento de entidades para a realização de operações de salvar, ler, atualizar e excluir dados e consulta de dados via linguagem JPQL (BRAVESCO, 2015).

JPA ou Java Persistence API é uma API (Application Programming Interface) em linguagem java para persistência de dados, esta API facilita o uso do Hibernate através de anotações nos objetos, poupando grande parte da configuração e mapeamento destes objetos nos arquivos de configuração do Hibernate (MEDEIROS, 2014)

Os diversos sistemas de gerenciamento de banco de dados, usados hoje, possuem interfaces distintas para uso de SQL, nesse sentido, o provedor de persistência Hibernate foi criado, a partir de uma implementação de código aberto, para mediar a interação entre a aplicação e o banco de dados. Ele é um dos provedores de persistência disponíveis para uso com a JPA (BARAVESCO, 2015).

2.9 Spring Framework

Spring é um framework de código aberto que foi criado por Rod Johnson para lidar com a complexidade de desenvolvimento de aplicativos corporativos. O Spring torna possível usar simples JavaBeans para conseguir coisas que antes só eram possíveis com EJBs. Porém, a utilidade do Spring não é limitada ao desenvolvimento do lado do servidor. Qualquer aplicativo pode se beneficiar do Spring provendo simplicidade, aumentando a produtividade de

desenvolvimento, o desempenho em tempo de execução e ao mesmo tempo prove uma cobertura para testes. (BRAGA, 2011).

O Spring é um framework e um container leve orientado a aspectos, com injeção de dependência, que ajuda no desenvolvimento de código de aplicativo com baixo acoplamento. O Spring é extremamente modular, assim é possível usar somente as partes necessárias, os módulos são construídos a partir da base de injeção de dependência e Programação Orientada a Apectos (AspectOriented Programming – AOP), criando uma plataforma cheia de recursos, sobre a qual se constroem os aplicativos (BRAGA, 2011).

Spring Boot Trata-se de um framework que tem como objetivo reaproveitar as tecnologias já existentes no Spring Framework e aumentar a produtividade do desenvolvedor. Introduzido na versão 4.0 do Spring, surgiu como necessidade em vista das críticas sobre a dificuldade e o tempo necessário para se iniciar o desenvolvimento de novos projetos. A principal diferença se dá no modo de configurar, organizar o código e executar a aplicação. (BIANCHI,2015)

O Spring Boot facilita a criação de aplicativos baseados em Spring, como ele abstrai a complexidade de configuração, basta "executar" o projeto. A maioria dos aplicativos Spring Boot precisa de uma configuração de Spring muito pequena, o que facilita para o desenvolvedor (SPRINGBOOT, 2018).

O conceito de convenção sobre configuração é o grande motor por trás do ganho de produtividade do Spring Boot, ou seja, a maior parte das configurações que o desenvolvedor precisa escrever no início de um projeto são sempre as mesmas, dessa forma, o Spring Boot já inicia novo projeto com todas estas configurações definidas podendo ser alteradas a qualquer momento (BIANCHI,2015).

2.10 Angular

O Angular é uma plataforma que facilita a criação de aplicativos para web. Angular combina templates declarativos, injeção de dependência, ferramentas de teste ponta a ponta e práticas recomendadas integradas para resolver desafios de desenvolvimento. O framework permite que os desenvolvedores criem aplicativos Web, dispositivos móveis ou aplicativos desktop de forma rápida e de fácil manutenção (ANGULAR, 2018).

O Angular CLI (command line interface) é uma ferramenta de interface de linha de comando que pode criar um projeto, adicionar arquivos e executar várias tarefas de desenvolvimento, como teste, empacotamento e implantação. Por exemplo o comando ng serve

sobe o projeto para o servidor, verifica seus arquivos e recria o aplicativo conforme você faz alterações nesses arquivos (ANGULAR, 2018).

A CLI cria o primeiro componente Angular automaticamente para você. Este é a componente raiz e é chamado de raiz do aplicativo. Você pode encontrá-lo em ./src/app/app.component.ts. A pasta src é apenas um dos itens dentro da pasta raiz do projeto. Outros arquivos ajudam você a criar, testar, manter, documentar e implantar o aplicativo. Esses arquivos vão na pasta raiz ao lado de src (ANGULAR, 2018).

A primeira grande ideia é que um aplicativo Angular 2 é composto de componentes. Uma maneira de pensar em Componentes é ensinando novas tags do navegador. Uma das grandes coisas sobre os componentes é que eles podem ser combinados. Isso significa que podemos construir componentes maiores a partir de componentes menores. O aplicativo é simplesmente um componente que renderiza outros componentes (ANGULAR, 2018).

2.11 PostgreSQL

PostgreSQL é um poderoso sistema gerenciador de banco de dados objeto-relacional de código aberto. Pode ser considerada como uma ferramenta madura, pois tem mais de 15 anos de desenvolvimento ativo e uma arquitetura que comprovadamente ganhou forte reputação de confiabilidade, integridade de dados e conformidade a padrões (SOAREZ, 2014).

O PostgreSQL roda em todos os grandes sistemas operacionais, incluindo GNU Linux, Unix e Microsoft Windows. É totalmente compatível com ACID (Acrônimo de Atomicidade, consistência, Isolamento e Durabilidade), tem suporte completo a chaves estrangeiras, junções joins, visões, gatilhos e procedimentos armazenados. Suporta também o armazenamento de objetos binários, incluindo figuras, sons ou vídeos. Possui interfaces nativas de programação para C/C++, Java, .Net, Perl, Python, entre outros, e uma excepcional documentação (SOAREZ, 2014).

Pela riqueza de recursos e conformidade com os padrões, o PostgreSQL é um SGBD (Sistema Gerenciador de Banco de Dados) muito adequado para o estudo universitário do modelo relacional, além de ser uma ótima opção para empresas implementarem soluções de alta confiabilidade sem altos custos de licenciamento (SOAREZ, 2014).

2.12 Heroku

Heroku é uma plataforma de nuvem como serviço onde os desenvolvedores podem realizar o deploy, gerenciar e escalar seus aplicativos. Essa plataforma dá suporte a várias linguagens de programação como Java, Node.js, Scala, Ruby, Python, Php, Go, Scala e Clojure.

O heroku fornece ao desenvolvedor liberdade para se concentrar no produto que está criando sem se preocupar com servidores, hardware ou infraestrutura (HEROKU, 2018).

Muitas abstrações de software são necessárias para que os desenvolvedores aumentem a sua produtividade. O heroku abstrai o fardo de ter que gerenciar hardware ou máquinas virtuais. Ao realizar o deploy no heroku ele empacota o código e as dependências do aplicativo em containers que são ambientes leves e isolados que fornecem processamento, memória, sistema operacional e um sistema de arquivos (HEROKU, 2018).

3. Metodologia

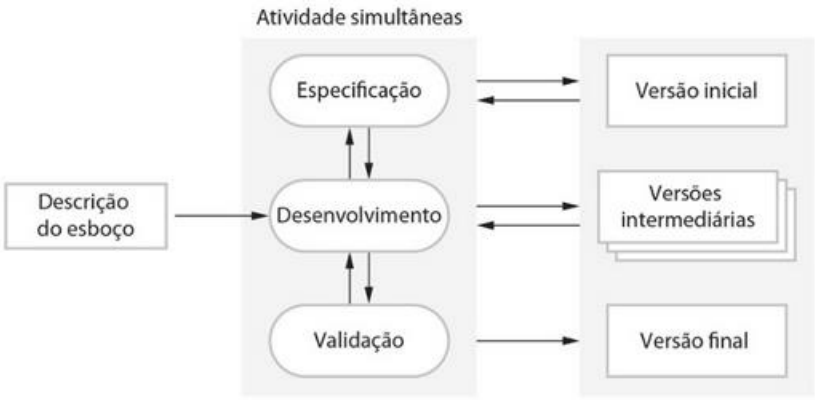
Inicialmente foi realizado um estudo sobre dados estatísticos do mercado agropecuário no estado de goiás e no Brasil como forma de contextualizar a motivação da criação do sistema descrito. Em seguida foi realizada uma revisão da literatura mediante a aplicação de métodos explícitos e sistematizados de busca, apreciação crítica e síntese de informação selecionada. A busca das fontes (artigos científicos, dissertações, teses e outros) foi realizada no *Google Academics* e Biblioteca da SciELO, a partir das palavras-chave: Sistema-WEB, Java, UML, Frango Caipira, Arquitetura REST, gestão de criação.

As atividades que envolveram o desenvolvimento do sistema:

- Levantamento dos requisitos com base em entrevistas/reuniões com o cliente;
- Definição das funcionalidades do sistema, diagrama de caso de uso e diagrama de classe;
- Estudo de técnicas e ferramentas a serem utilizadas para o desenvolvimento;
- Elaboração da arquitetura do sistema;
- Prototipagem das telas do sistema
- Codificação do sistema
- Implantação do sistema

O modelo de processo de software utilizado nesse trabalho foi o Incremental. O desenvolvimento incremental é baseado na idéia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido. As atividades de especificação, desenvolvimento e validação são intercaladas, e não separadas, com rápido feedback entre todas as atividades (SOMMERVILLE, 2011) A figura 1 exemplifica o modelo incremental.

Figura 1 - Modelo Incremental



Fonte: SOMMERVILLE (2011)

Desenvolvimento incremental reflete a maneira como resolvemos os problemas. Raramente elaboramos uma completa solução do problema com antecedência, geralmente movemo-nos passo a passo em direção a uma solução, recuando como percebemos que cometemos um erro. Ao desenvolver um software de forma incremental, é mais barato e mais fácil fazer mudanças no software durante seu desenvolvimento (SOMMERVILLE, 2011).

Cada incremento ou versão do sistema incorpora alguma funcionalidade necessária para o cliente. Frequentemente, os incrementos iniciais incluem a funcionalidade mais importante ou mais urgente. Isso significa que o cliente pode avaliar o sistema em um estágio relativamente inicial do desenvolvimento para ver se ele oferece o que foi requisitado. Em caso negativo, só o incremento que estiver em desenvolvimento no momento precisará ser alterado e, possivelmente, nova funcionalidade deverá ser definida para incrementos posteriores (SOMMERVILLE, 2011).

A validação ao decorrer do projeto se deu pela a validação dos requisitos do sistema, validação dos dados do sistema, validação dos protótipos do sistema, validação da usabilidade do sistema e validação final do sistema. Ao decorrer do projeto toda solicitação de mudança foi avaliada e os artefatos desse projeto já estão de acordo com as últimas especificações do cliente.

Através dos diagramas de caso de uso e uma breve descrição do mesmo o cliente pôde verificar quais seriam as funcionalidades principais do sistema. O cliente aprovou as funcionalidades e o diagrama de caso de uso foi modificado ao decorrer do projeto de acordo com o cliente ou de acordo com adaptações técnicas.

Por meio do diagrama de classe o cliente pôde verificar quais seriam os dados principais do sistema que seriam registrados no banco de dados. O cliente aprovou os dados e o diagrama de classe de uso foi modificado ao decorrer do projeto de acordo com o cliente ou de acordo

com adaptações técnicas.

Os protótipos permitiram ao cliente verificar como a tela e os dados estariam dispostos no sistema. O cliente aprovou os protótipos e foram modificados ao decorrer do projeto de acordo com novas perspectivas do cliente. As validações de usabilidade também foram verificadas nos protótipos e aprovadas.

A validação final do projeto se deu através de simulação em ambiente de produção fornecido pela plataforma Heroku. O cliente pode testar tanto pelo celular quanto pelo seu computador avaliando e aprovando a usabilidade. A simulação com dados reais o cliente validou e verificou a integridade dos dados e aprovou a aplicação como um todo, pois atendeu a necessidade do mesmo e todas as suas sugestões de adaptações ou mudanças foram implementadas.

4. Desenvolvimento

Esse capítulo apresentará os passos realizados para o desenvolvimento do software avisoftware utilizando como base o modelo iterativo e incremental.

Inicialmente será apresentado como foi coletado os requisitos do sistema relacionados a produção de frangos, depois será apresentado os artefatos que foram gerados após a definição e aprovação dos requisitos, ou seja, os artefatos da modelagem do projeto. E por fim será demonstrado como foi realizada a etapa de codificação e deploy do sistema.

Os requisitos foram levantados através de reuniões com o dono na fazenda NC. O proprietário da fazenda informou que os registros dos gastos são feitos em cadernos e que muitas vezes perde a rastreabilidade de sua produção. Com a necessidades de saber seus verdadeiros gastos foram coletados os dados que são considerados essenciais para o controle de suas despesas.

O proprietário informou que seus gastos realizados em cada lote geralmente são com alimentação, vacina, medicamento, mortalidade e despesa em geral de manutenção do local. O mesmo também informou que as vacinas são aplicadas em datas específicas após o início de cada produção.

Notou-se então que o sistema deveria de alguma maneira contabilizar todos esses gastos e como resultado da entrevista foi realizado o primeiro incremento do processo que foi o diagrama de caso de uso e uma breve descrição do mesmo. Com o caso de uso pode-se perceber as principais funcionalidades que o sistema deveria oferecer. Após o primeiro esboço desses artefatos o cliente validou as especificações apresentadas.

A próxima iteração se deu após a definição do diagrama de classe das entidades. Os atributos das classes foram desenvolvidos a partir do caso de uso, descrição do caso de uso e perante ao cliente que pode contribuir para adição ou remoção de dados e validação dos mesmos.

Com os dados necessários em mãos a próxima etapa foi o desenvolvimento dos protótipos das principais telas do sistema. Com o protótipo o cliente ficou mais familiarizado de como o sistema iria funcionar e como os dados estariam dispostos parar a visualização e cadastro. O cliente solicitou algumas mudanças e posteriormente validou o artefato. Os protótipos do sistema estão no APÊNDICE A deste trabalho.

O diagrama de caso de uso e o diagrama de classe foram criados utilizando a ferramenta Astah. Que fornece recursos para modelagem de sistema utilizando UML. Facilitando o entendimento de como o sistema deveria funcionar.

A codificação se iniciou com a criação da estrutura da arquitetura. Foi desenvolvido dois projetos separados um para o front end que é a parte do software onde usuário estará interagindo

diretamente, também podendo ser chamada de cliente e outro para o back end, que é a parte responsável por realizar as regras de negócio e comunicar com o banco de dados, podendo ser chamada de servidor.

A arquitetura do back end foi desenvolvida com a linguagem de programação Java. Para a comunicação com banco de dados e recebimento e envio de requisições do front end, que foi utilizado o framework spring para sua construção. Como banco de dados da aplicação foi utilizado o Postgres. A aplicação apresenta segurança com spring security e utiliza autenticação via token jwt.

O front end foi desenvolvido com o framework Angular. O framework permite a manipulação de arquivos html, css e typescript que são essenciais para o desenvolvimento das telas. Foi implementado nesse framework uma camada de serviço que ficou responsável por fazer e receber as requisições do back end.

Cada caso de uso teve aproximadamente uma semana para ser desenvolvido, após a implementação dos 4 primeiros casos de uso, houve o primeiro retorno do cliente que aprovou a implementação e sugeriu alguns ajustes. Os ajustes e os outros casos de uso foram implementados e o sistema foi apresentado novamente ao cliente. O cliente fez suas primeiras simulações com todas as funcionalidades juntas e aprovou o software.

Com a validação do sistema foi necessário fazer o deploy do sistema para que o mesmo fosse testado em ambiente de produção. Foi utilizado a plataforma Heroku para realização do deploy do software. A aplicação está rodando no endereço https://nocsoftware.herokuapp.com/ e contém todas as funcionalidades propostas neste trabalho.

Com a aplicação em produção o cliente testou o software e verificou que houve otimização dos dados e relatou que o sistema facilitou a visualização do histórico de suas produções, possibilitou a observação do preço atual das aves com o controle de todos os gastos para cada lote e aumentou a rastreabilidade dos lucros tanto do proprietário quanto dos colaboradores.

Os dados utilizados pelo cliente para o teste no sistema foi uma simulação trimestral com dados reais de suas vendas e despesas. O proprietário observou que os dados fornecidos pelo sistema estavam de acordo com seus cálculos previstos. O cliente verificou que é possível acompanhar os lucros e as despesas ao longo de todo ano e que é possível extrair desses dados quais meses são mais rentáveis e variação de custos de recursos durante o período de cada produção.

A seguir será apresentado os principais artefatos produzidos para que o processo de desenvolvimento do software fosse alcançado. Fez necessário definir as limitações do sistema

ou seja os requisitos não funcionais. Os principais requisitos não funcionais do sistema estão descritos na tabela 1.

Tabela 1- Descrição dos Requisitos Não Funcionais

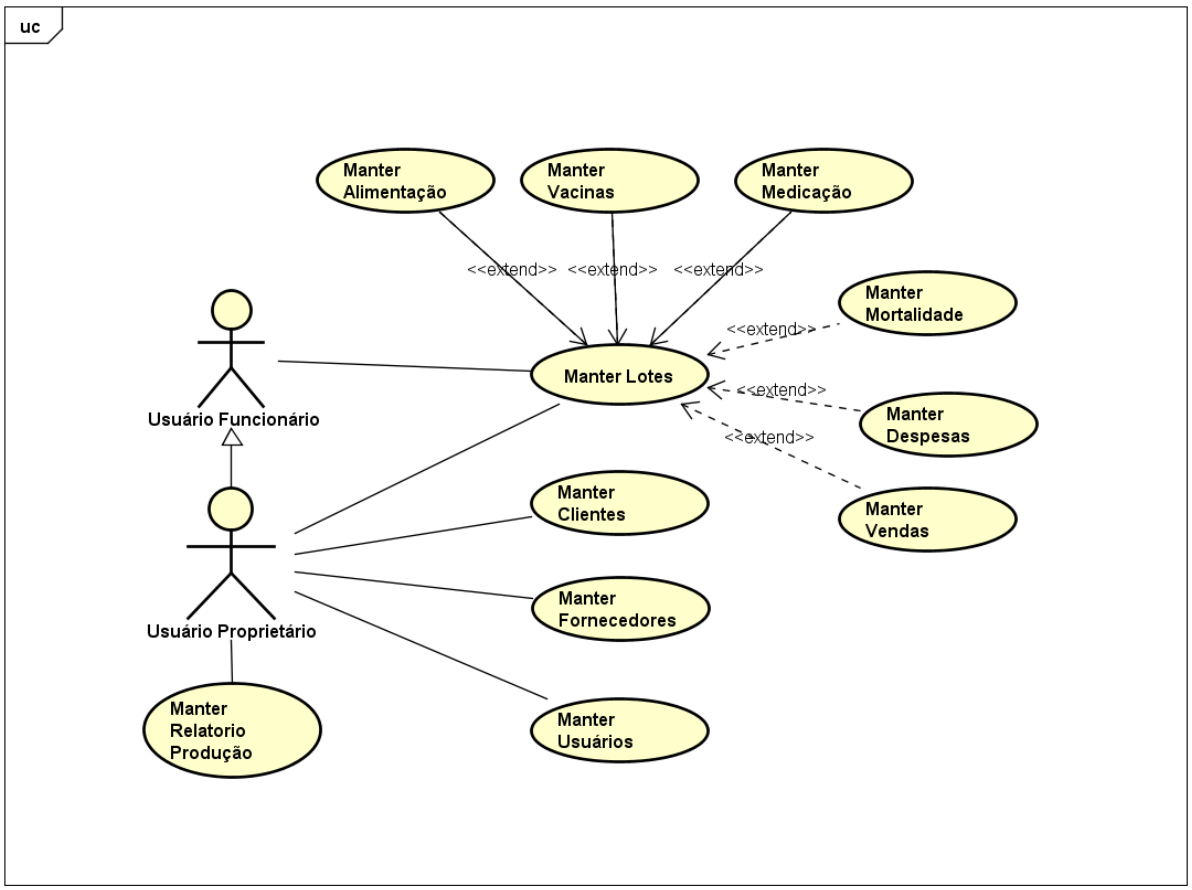
Requisito Não Funcional 01 – Acesso ao Sistema
O sistema deve oferecer tela de login para usuários de diferentes níveis de acesso. As senhas armazenadas no banco de dados devem estar criptografadas.
Requisito Não Funcional 02 – Utilização 3g
O sistema deve fornecer condições para que o usuário acesse o sistema via 3g do seu celular. Logo o sistema deve ser web e responsivo ou seja se adaptar a tela do celular.
Requisito Não Funcional 03 – Usabilidade
O sistema deve oferecer facilidade para a utilização e memorização dos recursos oferecidos. A tela deve ser objetiva o oferecer poucos cliques para que o usuário realize alguma operação.
Requisito Não Funcional 04 – Fácil Manutenção e Escalabilidade
O sistema deve ser de fácil manutenção e escalabilidade. O sistema deve ter recursos para crescer ou ser modificado facilmente conforme as necessidades do cliente.

Fonte: Autor da Pesquisa

4.1 Diagrama de Caso de Uso do Sistema

O diagrama de caso de uso do sistema representado na figura 2 reflete bem as principais funcionalidades do sistema. O termo “manter” significa as operações básicas de cadastro, edição, pesquisa e exclusão. O extend no diagrama significa que o usuário após o cadastro de um lote pode ou não realizar o cadastro de alimentação, vacina, medicação mortalidade, despesa e venda. A descrição do caso de uso está na tabela 2.

Figura 2 - Diagrama de Caso de Uso do Sistema



Fonte: Autor da Pesquisa

Tabela 2 - Descrição do Caso de Uso

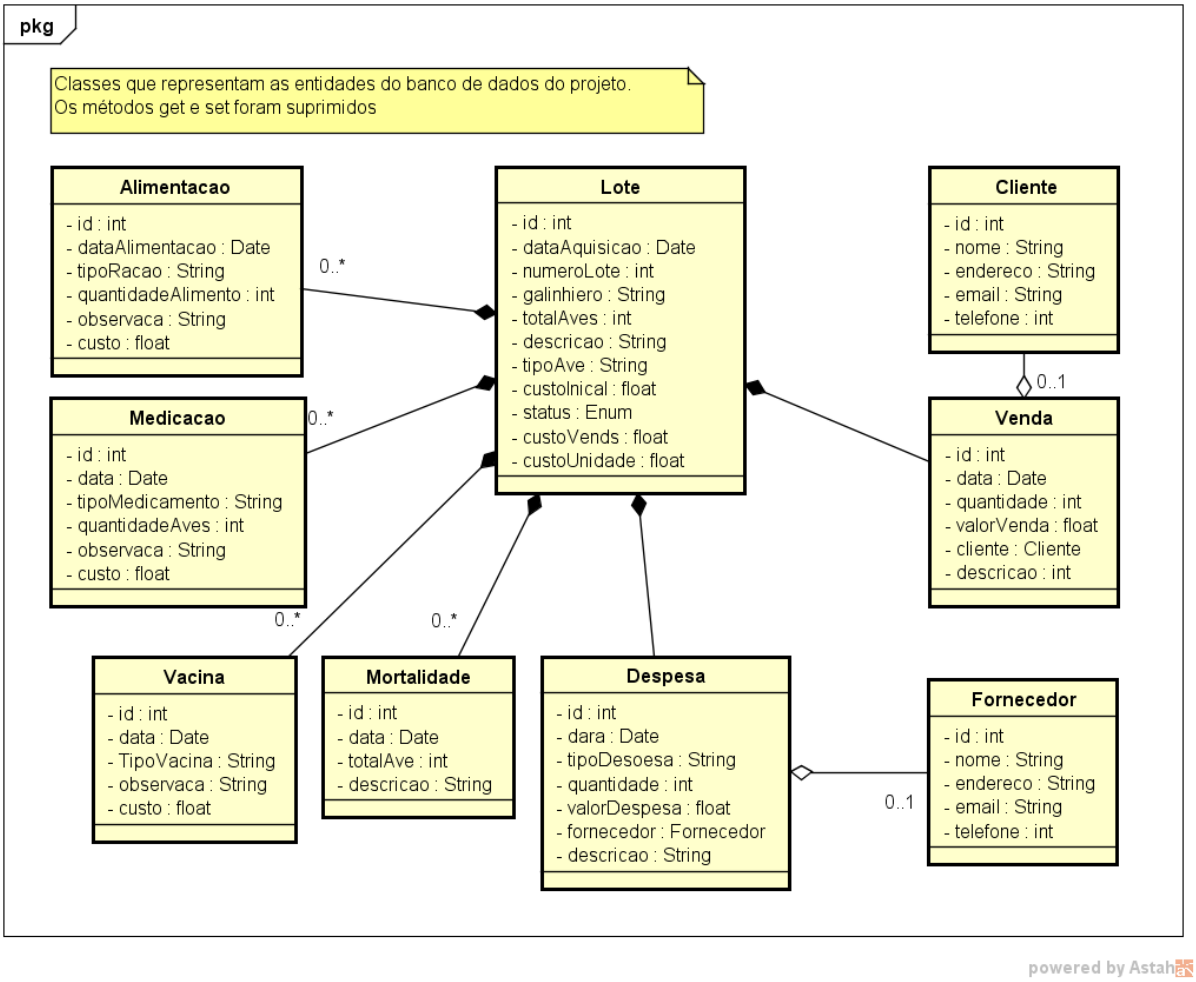
Caso de Uso 01 – Manter Clientes (descrição informal)
O administrador do sistema, terá a permissão de realizar os cadastros dos clientes no sistema. O sistema disponibilizará um formulário contendo informações relacionadas aos clientes relacionados ao negócio da empresa. Alguns dados dos clientes são: nome, endereço, e-mail telefone. O usuário poderá realizar as operações de inclusão, exclusão, alteração e busca.
Caso de Uso 02 – Manter Fornecedores (descrição informal)
O administrador do sistema, nesse caso o usuário proprietário, terá a permissão de realizar os cadastros dos fornecedores no sistema. O sistema disponibilizará um formulário contendo informações relacionadas aos fornecedores relacionados ao negócio da empresa. Alguns dados dos fornecedores são: nome, endereço, e-mail telefone. O usuário poderá realizar as operações de inclusão, exclusão, alteração e busca.
Caso de Uso 03 – Manter Lotes (descrição informal)
O administrador do sistema, nesse caso o usuário proprietário, terá a permissão de realizar os cadastros dos lotes no sistema. O sistema disponibilizará um formulário contendo informações relacionadas aos lotes da

empresa. Alguns dados dos lotes são: número do lote, data de aquisição, status (iniciado/finalizado). O usuário poderá realizar as operações de inclusão, exclusão, alteração e busca.
Caso de Uso 04 – Manter Medicação (descrição informal)
O administrador do sistema, nesse caso o usuário proprietário ou funcionário, terá a permissão de realizar os cadastros das medicações dos lotes no sistema. O sistema disponibilizará um formulário contendo informações relacionadas as medicações dos lotes. Alguns dados das medicações são: tipo da medicação, data e número de aves vacinadas. O usuário proprietário poderá realizar as operações de inclusão, exclusão, alteração e busca. O usuário funcionário poderá apenas incluir e buscar dados da medicação.
Caso de Uso 05 – Manter Mortalidade (descrição informal)
O administrador do sistema, nesse caso o usuário proprietário ou funcionário, terá a permissão de realizar os cadastros das mortalidades dos lotes no sistema. O sistema disponibilizará um formulário contendo informações relacionadas as mortalidades dos lotes. Alguns dados das mortalidades são: lote, data, , quantidade de aves, descrição. O usuário proprietário poderá realizar as operações de inclusão, exclusão, alteração e busca. O usuário funcionário poderá apenas incluir e buscar dados das mortalidades.
Caso de Uso 06 – Manter Ração (descrição informal)
O administrador do sistema, nesse caso o usuário proprietário ou funcionário, terá a permissão de realizar os cadastros dos dados das rações no sistema. O sistema disponibilizará um formulário contendo informações relacionadas as rações dos lotes. Alguns dados das rações são: tipos de ração, quantidade, data do preparo. O usuário proprietário poderá realizar as operações de inclusão, exclusão, alteração e busca. O usuário funcionário poderá apenas incluir e buscar dados da ração.
Caso de Uso 07 – Manter Vacinas (descrição informal)
O administrador do sistema, nesse caso o usuário proprietário ou funcionário, terá a permissão de realizar os cadastros dos dados das vacinas no sistema. O sistema disponibilizará um formulário contendo informações relacionadas as vacinas. Alguns dados das vacinas são: data, tipo de vacina, custo.
Caso de Uso 08 – Manter Despesas (descrição informal)
O administrador do sistema, nesse caso o usuário proprietário, terá a permissão de realizar os cadastros das despesas no sistema. O sistema disponibilizará um formulário contendo informações relacionadas as despesas dos lotes. Alguns dados das despesas são: gasto, item, fornecedor, valor, descrição. O usuário proprietário poderá realizar as operações de inclusão, exclusão, alteração e busca.
Caso de Uso 09 – Manter Vendas (descrição informal)
O administrador do sistema, nesse caso o usuário proprietário, terá a permissão de realizar os cadastros das vendas no sistema. O sistema disponibilizará um formulário contendo informações relacionadas as vendas dos lotes. Alguns dados das vendas são: data, quantidade, valor, descrição. O usuário proprietário poderá realizar as operações de inclusão, exclusão, alteração e busca.
Caso de Uso 10 – Manter Usuários (descrição informal)
O administrador do sistema, inicialmente cadastrará e definirá as permissões dos usuários do sistema diretamente no banco de dados.

Fonte: Autor da Pesquisa

4.2 Diagrama de Classe do Sistema

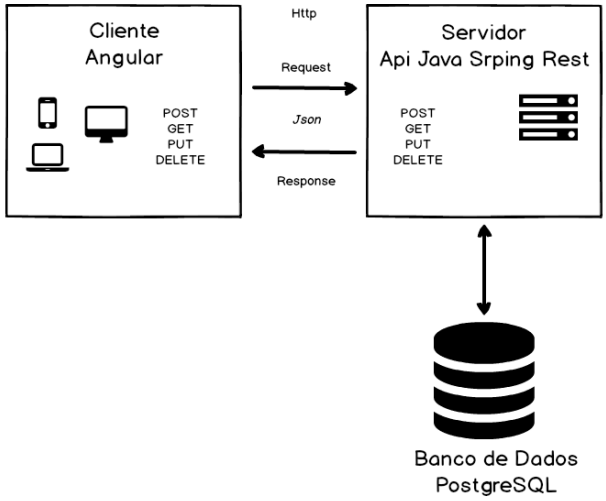
Figura 3 - Diagrama de Classe do Sistema



Fonte: Autor da Pesquisa

4.3 Arquitetura do projeto

Figura 4 - Representação da Arquitetura do Projeto



Fonte: Autor da Pesquisa

A arquitetura funciona através de trocas de requisições http enviadas/recebidas do front end que foi desenvolvido em angular para o back end. Os objetos com as informações das entidades são enviados via json para a api que foi desenvolvida em java spring que recebe essas requisições e efetua a ação conforme requisitado (POST, GET, PUT, DELETE).

As requisições POST por exemplo são enviadas para o armazenamento dos dados. As requisições GET recuperam os dados da api. A api é responsável por receber os dados enviados do front end e armazena-los ou recupera-los do banco de dados.

A Api através da interface do pacote Repository herda JpaRepository que é capaz de realizar todas as operações no banco de dados. As classes do pacote Service são responsáveis por realizar as regras de negócios dos dados, comunicando entre o controller e o repository. As requisições http são recebidas e enviadas através do controller que o mesmo fica responsável por comunicar as classes entidades do pacote modelo com a as classes de serviço.

As requisições realizadas pelo front end angular são fornecidas pela classe service que realiza a comunicação via http com o back end. Toda as requisições solicitadas pelo o usuário são realizadas nessa camada. Uma classe model com as mesmas entidades do back end são enviados e recebidos da api via json pelo service.

O Angular é responsável por toda a parte de interação com o usuário, ele define como será os formulários e modelos de disponibilização de dados na tela. Sendo que esses dados são recebidos ou enviados da api. O método get da classe de serviço do angular solicita ao método

get do controller da api, que retorna os dados via json e são apresentados na tela.

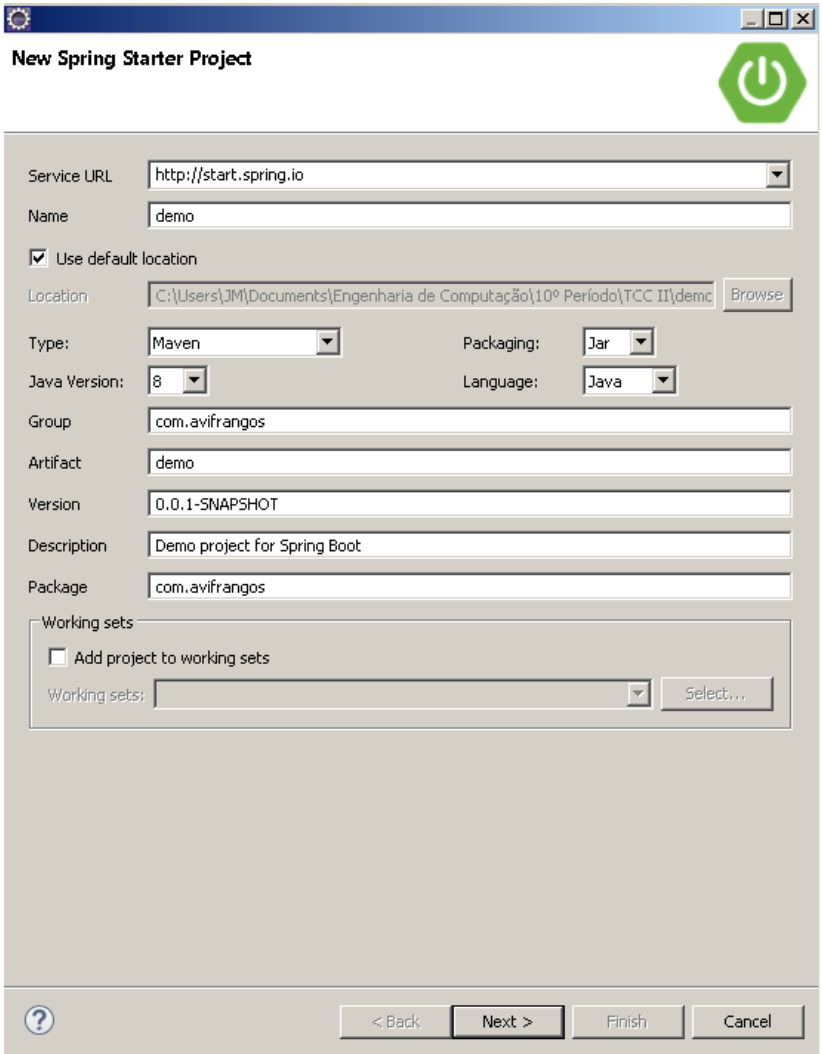
4.4 Criação do Back End Spring Rest

A api foi desenvolvida utilizando a linguagem de programação Java, framework spring e o banco de dados postgreSql. A Ide para a codificação foi Eclipse mars.2.

4.4.1 Criação do projeto Spring boot no Eclipse

O projeto foi criado utilizando a opção Spring starter Project que está no menu no eclipse dentro de file-> new. Essa opção permite algumas configurações iniciais importantes para a codificação da api como a definição do projeto tipo Maven, a versão do java e o tipo de pacote (war) e a nomenclatura que será utilizada nos pacotes do projeto e nomenclatura do projeto.Essa opção de criação do projeto pode ser observada na figura 5.

Figura 5 - Tela de criação do projeto Spring no Eclipse



Fonte: Print screen da opção Spring Start Project da Ide Eclipse

4.4.2 Configuração do projeto

O arquivo application.properties que foi gerado automaticamente com criação de um novo projeto spring contêm algumas configurações importantes, o projeto foi configurado conforme a figura 6. Nesse arquivo podemos definir o número da porta que o projeto está sendo executado e toda a configuração necessária para acessar o banco de dados. As informações da url, nome de usuário, senha do banco e se o hibernate será responsável por criar automaticamente as tabelas no banco de dados são adicionadas nesse arquivo.

Figura 6 - Configuração do application.properties

```
server.port=${port:8082}
spring.datasource.url= jdbc:postgresql://localhost:5432/avifrangos
spring.datasource.username = postgres
spring.datasource.password= root
spring.jpa.hibernate.ddl-auto=update
```

Fonte: Repositório do Autor da Pesquisa

O arquivo pom.xml representado na figura 7 fica responsável por fazer conter todas as dependências do projeto. Nesse arquivo fica por exemplo as dependências do spring boot, banco de dados postgresql, spring security, e outras. Essas dependências são baixadas automaticamente pelo Maven.

Figura 7 - Arquivo pom.xml com algumas dependências

```
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
…
    </dependencies>
```

Fonte: Repositório do Autor da Pesquisa

4.4.3 Model

As classes contidas no pacote model são entidades do projeto que foram armazenadas no banco de dados e os atributos dessas classes foram abstraídos através do diagrama de classes. A anotação @Entity identifica essa classe como uma entidade de modelo. A anotação @Id fez a identificação da chave primária da classe e a anotação @GeneratedValue permitiu a criação de um id diferente para cada instancia da classe. Essas anotações podem ser observadas na figura 8 que contêm um exemplo da classe entidade Lote utilizada no projeto.

Figura 8 - Classe entidade Lote

```
@Entity
public class Lote {

        @Id
        @GeneratedValue
        private int id;
        private Date dataAquisicao;
        private int numeroLote;
        private String galinheiro;
        private int totalAves;
        private String descricao;
        private String tipoAve;
        private Double custoInicial;
        private Double custoVenda = 0.0;
        private Double custoUnidade;
        private Status status = Status.INICIADO;

       .....
```

Fonte: Repositório do Autor da Pesquisa

4.4.4 Controller

As classes do pacote controller foram responsáveis por receber as requisições http e isso é permitido pela definição da classe com a anotação @RestController conforme a figura 9. A anotação @CrossOrigin definiu em qual porta essa classes receberá as requisições. Para a utilização do serviço utilizou-se a anotação @Autowired e através da anotação @RequestMapping é onde recebe o tipo de requisição (POST,GET,PUT ou DELETE) e também recebe como parâmetro que informa que a aplicação deverá consumir um JSON através do parâmetro consumes=MediaType.APPLICATION_JSON_VALUE.

Figura 9 - Classe LoteController

```
@RestController
@CrossOrigin(origins="http://localhost:4200", allowedHeaders="*")
public class LoteController {

	@Autowired
	LoteService loteService;

	@RequestMapping (method=RequestMethod.POST, value="/lotes",
consumes=MediaType.APPLICATION_JSON_VALUE)
	public ResponseEntity<Lote> cadastrarLote(@RequestBody Lote lote){
		Lote loteCadastrado = loteService.cadastrar(lote);
		return new ResponseEntity<>(loteCadastrado, HttpStatus.CREATED);
	}

	@RequestMapping (method=RequestMethod.GET, value="/lotes",
produces=MediaType.APPLICATION_JSON_VALUE)
	public ResponseEntity<Collection<Lote>> buscarTodosLotes(){
		Collection<Lote> lotesBuscados = loteService.buscarTodos();
		return new ResponseEntity<>(lotesBuscados, HttpStatus.OK);
	}

	@RequestMapping (method=RequestMethod.DELETE, value="/lotes/{id}")
	public boolean excluirLote(@PathVariable int id) throws CustomException{
		Lote loteEncontrado = loteService.buscarPorId(id);
		loteService.excluir(loteEncontrado);
		return true;
	}

	@RequestMapping (method=RequestMethod.PUT, value="/lotes",
consumes=MediaType.APPLICATION_JSON_VALUE)
	public ResponseEntity<Lote> alterarLote(@RequestBody Lote lote){
		Lote loteAlterado = loteService.alterar(lote);
		return new ResponseEntity<>(loteAlterado, HttpStatus.CREATED);
	}
	…
}
```

Fonte: Repositório do Autor da Pesquisa

4.4.5 Service

A definição da classe de serviço se deu pela anotação @Service e essa classe foi responsável pelas regras de negócio do projeto. Nela foram efetuadas validações ou operações nos dados quando necessário. A figura 10 tem alguns métodos utilizados na classe LoteService como cadastrar, buscarTodos, excluir, buscarPorId e alterar.

Figura 10 - Classe LoteService

```
@Service
public class LoteService {

	@Autowired
	LoteRepository loteRepository;

	...

	public Lote cadastrar(Lote lote){
		lote.setCustoUnidade((lote.getCustoInicial()/lote.getTotalAves()));
		lote.setCustoInicial(lote.getCustoInicial());
		return loteRepository.save(lote);
	}

	public Collection<Lote> buscarTodos() {
		return loteRepository.findAll();
	}

	public void excluir (Lote lote) throws CustomException{
		verificaVinculo(lote);
		loteRepository.delete(lote);
	}

	public Lote buscarPorId(int id){
		return loteRepository.findOne(id);
	}

	public Lote alterar(Lote lote){
		return loteRepository.save(lote);
	}
 ...
}
```

Fonte: Repositório do Autor da Pesquisa

4.4.6 Repositoy

A interface repository foi responsável por comunicar diretamente com o banco de dados como ela herda do JpaRepository todos os métodos como de salvar, alterar, excluir, buscar e buscar por id já estão contidos nessa interface. Caso alguma operação especifica no banco de dados fosse necessária através da anotação @Query foi possível fazer consultas ou operações customizadas no banco de dados conforme a figura 11.

Figura 11 - Classe LoteRepository

```
public interface LoteRepository extends JpaRepository <Lote, Integer>  {

        @Query("SELECT SUM(lote.custoInicial) from Lote lote")
        Double getSomaCustoInicial();

        @Query("SELECT SUM(lote.totalAves) from Lote lote")
        Integer getTotalAveLote();
}
```

Fonte: Repositório do Autor da Pesquisa

4.4.7 Security

As classes contidas no pacote security foram utilizadas para realização do login do usuário e autenticação via token. O usuário ao acessar ao sistema se ele estiver cadastrado na base de dados com as devidas permissões receberá um token e poderá realizar as operações destinadas ao seu perfil de acesso.

4.4.8 Constantes

O pacote constantes foi utilizado para armazenar os enums do projeto. A figura 12 contém o exemplo de enum status que contêm o status do lote que pode ser iniciado ou finalizado. Um enum importante também nesse projeto é o enum de mensagens que foi responsável por conter todas as mensagens contidas na api.

Figura 12 - Enum Status

```
public enum Status {

        /* Iniciado */
   INICIADO("Iniciado"),

   /* Finalizado */
   FINALIZADO("Finalizado");

   public String nome;

   Status(String nome) {
      this.nome = nome;
   }

   public String getNome() {
      return nome;
   }

   public static List<Status> listarStatus(){
      return Arrays.asList(Status.INICIADO, Status. FINALIZADO);
   }
```

Fonte: Repositório do Autor da Pesquisa

4.5 Criação do Front End Angular

O front end foi criado utilizando como ide o visual studio e o angular como framework. Nesse projeto o angular cli utilizado foi a versão 1.6.6. e a versão do node 8.9.4. O node js foi instalado previamente a instalação do angular cli.

4.5.1 Criação do projeto Angular

Inicialmente o angular foi instalado no computador através do terminal do visual studio através do código “npm install -g @angular/cli”. O projeto front end foi construído através do comando “ng new avifrangos” onde os principais arquivos da aplicação angular foram gerados.

4.5.2 Module

Os módulos são definidos através da anotação @NgModule conforme a figura 13. Estes foram utilizados para declarar quais componentes, diretivas e pipes que pertencem a esse módulo. Como por exemplo nessa classe da tabela foi definido como importação o RouterModule que permite a esse módulo o acesso de rotas e o FormModule permite o acesso ao módulo de formulário. O BsDatePickerModule.forRoot() foi permitido ao módulo o uso de formatos de data. Os componentes utilizados nesse módulo foram definidos no array declaration e no array exports foi declarado o componente ListarLoteComponet para que o mesmo fosse utilizados por outros módulos do sistema.

O modulo LoteRoutes permitiu a definição de rotas que nesse exemplo conforme a figura 14, o path com valor lotes tem a reponsabilidade de direcionar o sistema para o path:lotes/listar que permite a listagem do lote através do componente ListarLoteComponent. Cada componente é carregado de acordo com o path definido. No caso o path lotes/formulário irá direcionar o sistema para a página de formulário que está contida no componente LoteFormComponent. O código para a criação do modulo de lote foi “ng g module lotes”.

Figura 13 - Classe LotesModule

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';
import { FormsModule } from '@angular/forms';
import { BsDatepickerModule }  from 'ngx-bootstrap/datepicker';

import { LoteService } from './shared';
import { ListarLoteComponent } from './listar';
import { FormLoteComponent } from './form';
import { VisualizarLoteComponent } from './visualizar';

@NgModule({
  imports: [
    CommonModule,
    RouterModule,
    FormsModule,
    BsDatepickerModule.forRoot()

  ],
  declarations: [ListarLoteComponent, FormLoteComponent,
VisualizarLoteComponent],
  providers:[
    LoteService,
```

```
    ListarLoteComponent
  ],
  exports: [ListarLoteComponent]
})
export class LotesModule { }
```

Fonte: Repositório do Autor da Pesquisa

Figura 14 - Lote Routes Module

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ListarLoteComponent } from './listar';
import { FormLoteComponent } from './form';
import { CanActivateAuthGuard } from '../can-activate.authguard';
import { VisualizarLoteComponent } from './visualizar';


export const LoteRoutes: Routes = [
        {
        path: 'lotes',
        redirectTo: 'lotes/listar'
        },
        {
        path: 'lotes/listar',
        component:ListarLoteComponent, canActivate: [CanActivateAuthGuard]
        },
         {
        path: 'lotes/formulario',
        component:FormLoteComponent, canActivate: [CanActivateAuthGuard]
        },
        {
        path: 'lotes/visualizar',
        component:VisualizarLoteComponent, canActivate:
[CanActivateAuthGuard]
        }
];
```

Fonte: Repositório do Autor da Pesquisa

4.5.3 Model

A classe *model* foi definida com os mesmos atributos contidos no *back end* da aplicação para disponibilizar e enviar os dados da tela, os modelos são enviados ou recebidos pelo componente que via *json* solicita as requisições do *service*. A figura 15 demonstra como foi criada a classe lote através dos recursos *TypeScript*.

Figura 15 - Classe entidade Lote Front End

```
export class Lote {

    constructor(
        public id?: number,
        public dataAquisicao?: string,
        public numeroLote?: number,
        public galinheiro?: string,
        public totalAves?: number,
        public descricao?: string,
        public tipoAve?: string,
        public custoInicial?: number,
        public status?: boolean){}
}
```

Fonte: Repositório do Autor da Pesquisa

4.5.4 Component

Os principais componentes criados em cada módulo foram os componentes de formulário e componentes de listagem. O componente e definido com a anotação @Component e o componente de formulário foi criado conforme a figura 16 O componente de formulário fica responsável por receber o objeto modelo que é preenchido pelo usuário na tela de formulário Html do componente. O componente faz a verificação se o objeto do formulário preenchido possui um id definido. Caso o id seja indefinido o objeto será enviado para o *service* utilizando o método *create* para a criação de um novo objeto. Caso o id já esteja definido o componente envia para *service* utilizando o método *update* para que os dados sejam alterados. O código "ng g componente lotes/form-lote" foi responsável para a criação do componente de formulário do lote.

Figura 16 - Component FormLoteComponent

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { LoteService, Lote } from '../shared';
import { Router } from '@angular/router';
import { NgForm } from '@angular/forms';

@Component({
  selector: 'app-form-lote',
  templateUrl: './form-lote.component.html',
  styleUrls: ['./form-lote.component.css']
})
export class FormLoteComponent implements OnInit {
  private selectUndefinedOptionValue:any;
  statusMessage: string;

  @ViewChild('formLote') formLote: NgForm;

  private lote:Lote;
  constructor(private _loteService:LoteService, private _router:Router) { }

  ngOnInit() {
    this.lote =this._loteService.getter();
}

  processForm(f){
    if(this.lote.id == undefined){
    this._loteService.createLote(this.lote).subscribe((lote)=>{
```

```
        console.log(lote);
        this._router.navigate(['/lotes']);
      },(error) =>{
        this.statusMessage = error.message;
        console.log(error);
      });
    }else {
      this._loteService.updateLote(this.lote).subscribe((lote)=>{
        console.log(lote);
        this._router.navigate(['/lotes']);
      },(error) =>{
        this.statusMessage = error.message;
        console.log(error);
      });
    }
  }
}
```

Fonte: Repositório do Autor da Pesquisa

4.5.5 Service

A classe de serviço ficou responsável para fazer a comunicação do *back end*. O componente solicita as classes do *service* enviando o modelo com dados preenchidos ou recebendo o modelo via *json*. O serviço via http envia requisições (POST,GET,PUT,DELETE) para a api com base na url. Para a criação do service o código foi "ng g service lotes/shared/lote". O *service* utiliza a anotação @Injectable e a implementação dessa classe pode ser observada na figura 17.

Figura 17 - Service LoteService

```
import { Injectable, EventEmitter, Output } from '@angular/core';
import { Http, Response, Headers, RequestOptions } from '@angular/http';
import { Observable } from 'rxjs/Observable';

import 'rxjs/add/operator/map'
import 'rxjs/add/operator/catch';
import 'rxjs/add/observable/throw';
import { Lote } from './';
import { AuthService } from '../../login/auth.service';


@Injectable()
export class LoteService {
  private baseUrl:string='http://localhost:8082';
```

```
  private headers = new Headers({'Content-Type':'application/json',
'Authorization': 'Bearer ' + this._authService.getToken()});
  private options = new RequestOptions({headers:this.headers});
  private lote:Lote;
  @Output() static emitirLote = new EventEmitter<Lote>();

  constructor(private _http:Http, private _authService: AuthService) { }

  getLotes(){

    return this._http.get(this.baseUrl+'/lotes',
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }

  getLote(id:Number){

    return this._http.get(this.baseUrl+'/lotes/'+id,
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }

  deleteLote(id:Number){

    return
this._http.delete(this.baseUrl+'/lotes/'+id,this.options).map((response:Resp
onse)=> response.json())
      .catch(this.errorHandler);
  }

  createLote(lote:Lote){

    return this._http.post(this.baseUrl+'/lotes', JSON.stringify(lote),
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);

  }

  updateLote(lote:Lote){

    return this._http.put(this.baseUrl+'/lotes', JSON.stringify(lote),
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }

  errorHandler(error:Response){
      return Observable.throw(error.json()||"SERVER ERROR");
  }
```

```
  setter(lote:Lote){
    this.lote=lote;
  }

  getter(){
    return this.lote;
  }

  menuLote(lote){
    LoteService.emitirLote.emit(lote);
   // this.emitirLote.emit(lote);
  }

   getAlimentacoesLote(id:Number){

    return this._http.get(this.baseUrl+'/lotes/alimentacaoLote/'+id,
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }

  getVacinasLote(id:Number){

    return this._http.get(this.baseUrl+'/lotes/vacinaLote/'+id,
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }

  getMedicacoesLote(id:Number){

    return this._http.get(this.baseUrl+'/lotes/medicacaoLote/'+id,
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }

  getMortalidadesLote(id:Number){

    return this._http.get(this.baseUrl+'/lotes/mortalidadeLote/'+id,
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }

  getDespesasLote(id:Number){

    return this._http.get(this.baseUrl+'/lotes/despesaLote/'+id,
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }

  getVendasLote(id:Number){
```

```
    return this._http.get(this.baseUrl+'/lotes/vendaLote/'+id,
this.options).map((response:Response)=> response.json())
      .catch(this.errorHandler);
  }
}
```

Fonte: Repositório do Autor da Pesquisa

4.6 Deploy

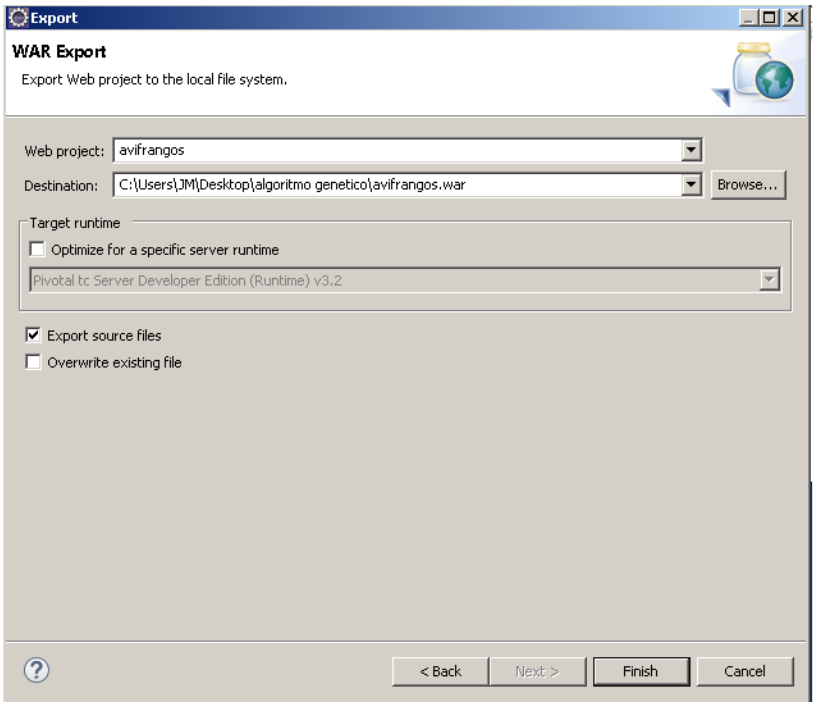
4.6.1 Build do projeto Front End Angular

Para realizar o deploy da aplicação front end junto ao back end foi realizado primeiramente o build do projeto angular que se deu pelo comando da tabela. Após o buid uma pasta dist foi criada com os principais arquivos da aplicação angular. Esses arquivos da pasta dist foram copiados e colados na pasta static do projeto back end. O build do projeto angular foi gerado através do código “ng g build --prod”.

4.6.2 Criação do arquico war

Após o arquivo contido na pasta dist armazenado na classe static foi necessário gerar um arquivo war da aplicação, através da opção export encontrada com um clique direito no mouse sobre o projeto. Dentro da opção export contém a opção war file conforme a figura 18.

Figura 18 - Tela para exportação do arquivo war



Fonte: Print screen da opção Export war file da Ide Eclipse

4.6.3 Deploy no Heroku

Para a realização deploy no heroku primeiramente foi criado uma conta no site www.heroku.com. Após a criação da conta foi necessário criar um novo projeto e realizar a configuração do banco de dados desejado. As configurações do banco de dados estão de acordo com na figura 19.

Figura 19 - Configuração do Banco de Dados no heroku

Fonte: Print screen das credenciais do banco de dados do Heroku.

As informações do banco de dados do heroku foram armazenadas em um novo arquivo de configuração do projeto chamado application-prod.properties apresentado na figura 20.

Figura 20 - Arquivo application-prod.properties

```
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=postgresql://ec2-54-204-46-236.compute-
1.amazonaws.com:5432/d3s15f3lbtkin1?sslmode=require&user=umbesmntorllus&password=eecf6d5affb35b
a25d5d22456760dbee3d0ee67b91339fe94d5e1a0ee99c25e8
spring.datasource.username = umbesmntorllus
spring.datasource.password= eecf6d5affb35ba25d5d22456760dbee3d0ee67b91339fe94d5e1a0ee99c25e8
spring.jpa.hibernate.ddl-auto=create

spring.jpa.properties.hibernate.show_sql=false
spring.jpa.properties.hibernate.use_sql_comments=true
spring.jpa.properties.hibernate.format_sql=false
spring.jpa.properties.hibernate.type=info
```

Fonte: Repositório do Autor da Pesquisa

E para que o heroku receba as informações contidas nesse arquivo mais um arquivo de configuração foi adicionado na raiz do projeto chamado Procfile que contém as informações de acordo com a figura 21.

Figura 21 - Arquivo Procfile

```
Web java $JAVA_OPTS -jar webapp-runner.jar ${WEBAPP_RUNNER_OPTS} --port $PORT
./avifrangos.war
```

Fonte: Repositório do Autor da Pesquisa

O deploy no heroku foi realizado a partir do código da tabela 16. Onde o heroku realiza o deploy do arquivo avifrangos.war para a aplicação nocsoftware que foi criada no heroku. O comando para a realização do deploy foi: heroku war: deploy avifrangos.war –app nocsoftware

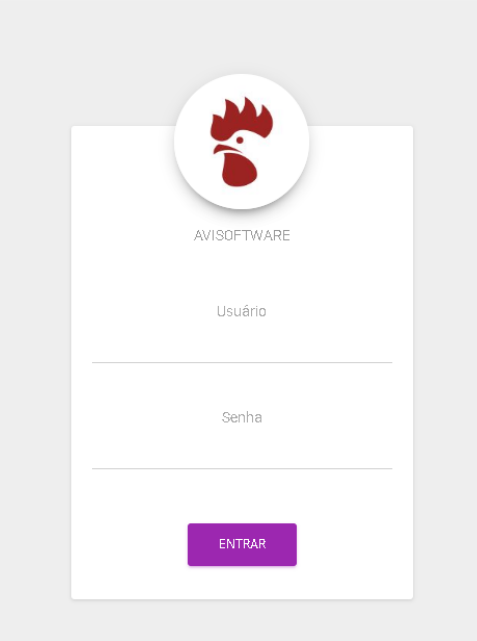
4.7 O sistema avisoftware Apresentação do Sistema

O sistema avisoftware foi projetado para auxiliar os gastos e vendas de uma criação de aves caipiras. Sua principal responsabilidade é registrar lotes de produção e os respectivos gastos para cada lote. O sistema permite o cadastro de alimentações, vacinas, medicações, mortalidades, despesas e vendas para cada lote. O sistema também permite o cadastro de fornecedores e clientes.

O usuário do sistema consegue obter facilmente um relatório de cada lote aumentado assim a sua rastreabilidade dos seus gastos. O usuário final consegue ver o balanço do custo e porcentagem de perda de aves de todos seus lotes produzidos. O sistema está disponível no link https://nocsoftware.herokuapp.com/.

4.7.1 Tela de Login

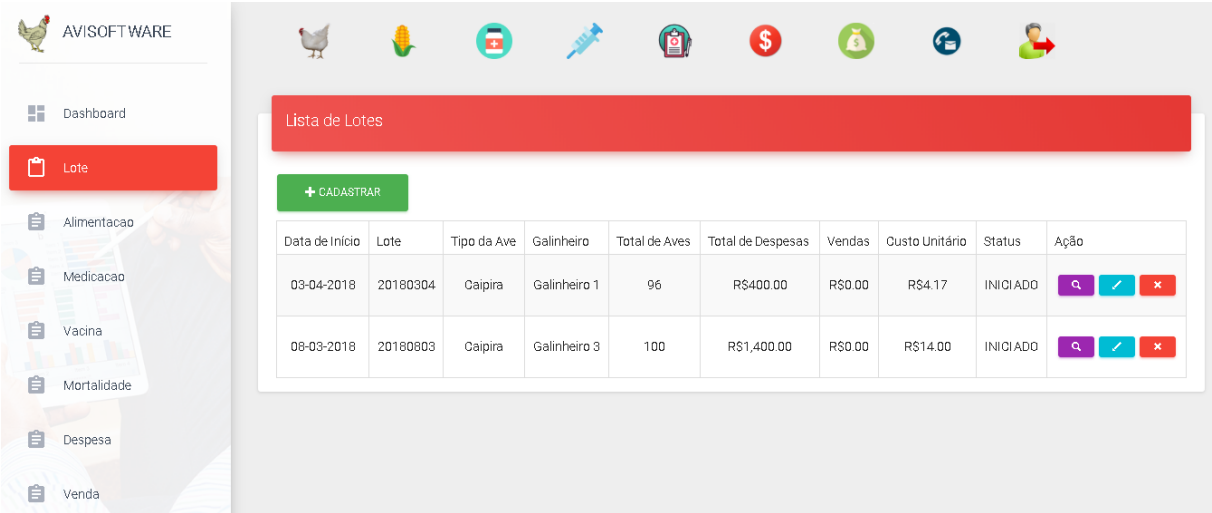
Figura 22 - Tela de Login do Sistema



Fonte: < https://nocsoftware.herokuapp.com/login>

4.7.2 Tela de Listagem de Lotes

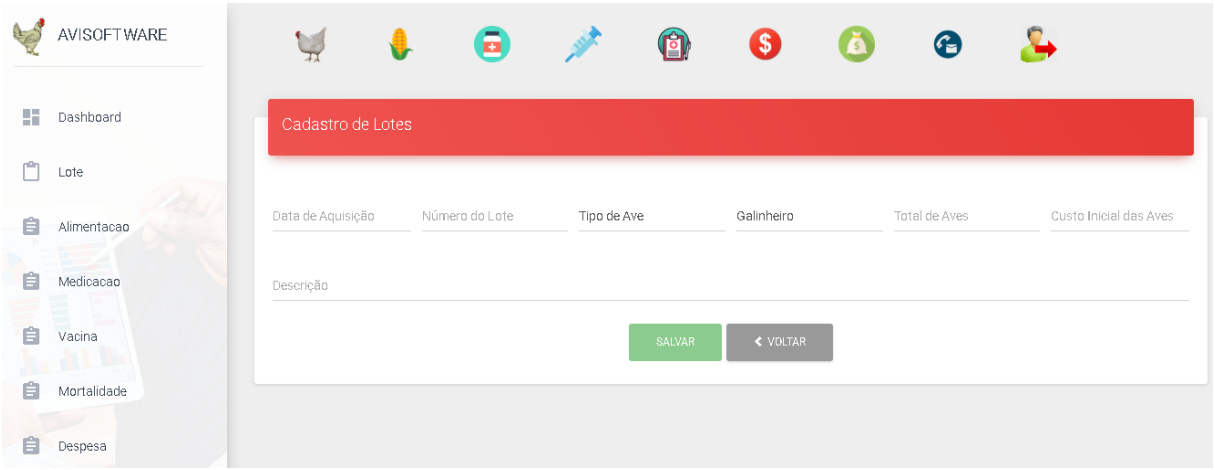
Figura 23 - Tela de Listagem dos Lotes



Fonte: <https://nocsoftware.herokuapp.com/lotes/listar>

4.7.3 Tela Cadastro de Lotes

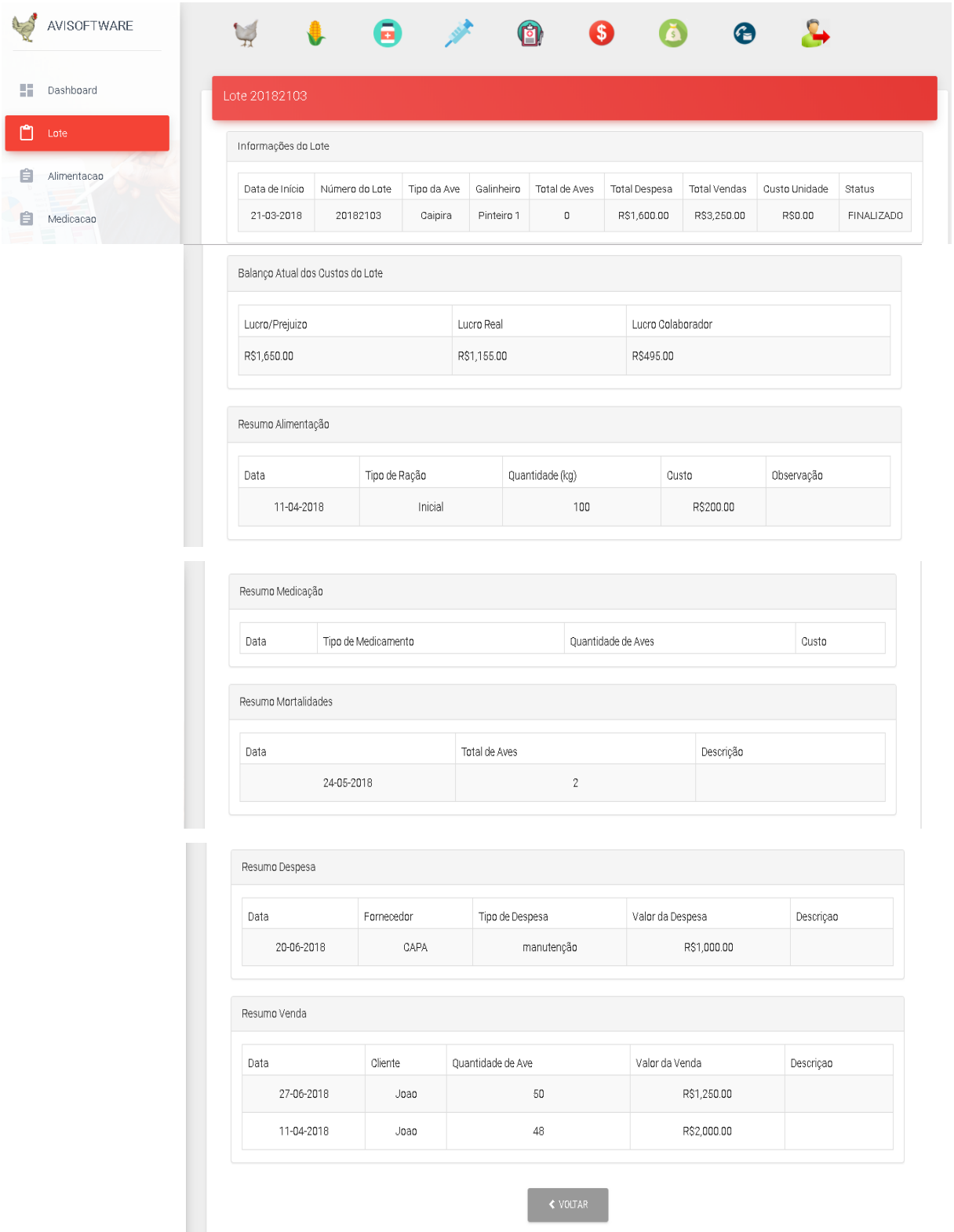
Figura 24 - Tela de Cadastro de Lotes



Fonte: <https://nocsoftware.herokuapp.com/lotes/formulario>

4.7.4 Tela Relatório do Lote

Figura 25 - Tela de Visualização do Relatório do Lote



Fonte: <https://nocsoftware.herokuapp.com/lotes/visualizar>

5. Considerações finais

Observa-se que o gerenciamento dos custos de uma produção é de extrema importância para o negócio e que a tecnologia da informação pode auxiliar o produtor a ter um maior controle sob a gestão dos seus recursos. O software desenvolvido nesse trabalho pôde mostrar ao produtor que todos seus registros de sua produção podem ser facilmente rastreados.

A utilização da tecnologia web proporcionou ao cliente acesso as suas informações de qualquer equipamento que tenha internet, ou seja também pode ser acessado pelo 3g do seu smartphone, que foi uma das suas necessidades especificadas para o projeto. O cliente verificou que o sistema permite a rastreabilidade das vendas e despesas de cada lote e que os cálculos automáticos aumentaram a rapidez na obtenção dos dados. O esforço para calcular tudo referente aos lotes que antes era feito tudo no papel ficou muito menor.

O processo de desenvolvimento de software proporcionou uma maior facilidade para o entendimento da necessidade do cliente. Desde da definição de um modelo baseados em etapas especificas para execução de atividades até a criação de artefatos da UML que amplificam a visão da construção do software como um todo.

O framework spring utilizado para a codificação do back end aumentaou muito a produtividade na hora da codificação já que o mesmo pode focar nas regras de negócios do projeto. O framework angular também proporcionou um aumento de produtividade e fácil escalabilidade do projeto. O heroku por oferecer toda uma infraestrutura para o deploy facilitou muito para desenvolvimento do software, pois proporcionou o foco na codificação do projeto.

6. Referências Bibliográficas

ANGULAR. 2018. Disponível em: <https://angular.io/>

BIANCHI, Evaldo. **Sistema de envio e recebimento de mensagens para plataforma android utilizando spring boot e google cloud messaging.** Pato Branco 2015 Universidade Tecnológica Federal do Paraná

BRAGA, Jackson. **Inversão de controle com injeção de dependência, aplicações EJB com spring framework**. Trabalho de diplomação. Universidade Tecnológica Federal do Paraná - UTFPR. 2011.

BUAINAIN, M. A. et al. **O Mundo Rural no Brasil do século 21: A formação de um novo padrão agrário e agrícola**. Embrapa Informação Tecnológica, 1º edição, 2014.

GUEDES, Gilleanes**. UML 2. UmaAbordagem Prática**. 2ª Edição. Novatec Editora. 2011.

HELLMEISTER FILHO, Paulo et al . **Efeito de genótipo e do sistema de criação sobre o desempenho de frangos tipo caipira**. R. Bras. Zoote**c.**, Viçosa , v. 32, n. 6, supl. 2, p. 1883-1889, Dec. 2003 . Available from <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1516-35982003000800012&lng=en&nrm=iso>. access on 15 August. 2017. http://dx.doi.org/10.1590/S1516-35982003000800012.

HEROKU. 2018. Disponível em: <https://www.heroku.com/>

KIST, Cássyo. **Sistema web para lojas de aluguel e venda de roupas**. Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. 2012

MENDES, Cássia, et al. **Estudo do mercado brasileiro para o agronegócio**. Campinas, SP: Embrapa Informática Agropecuária. 2011.

MORAIS, Jonas et al . Curva de crescimento de diferentes linhagens de frango de corte caipira. **Cienc. Rural**, Santa Maria , v. 45, n. 10, p. 1872-1878, Oct. 2015 . Available from

<http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-84782015001001872&lng=en&nrm=iso>. access on 15 Sept. 2017. Epub July 10, 2015. http://dx.doi.org/10.1590/0103-8478cr20130867.

PORTAL BRASIL. Disponível em: http://www.brasil.gov.br/economia-e-emprego/2015/07/brasil-sera-maior-exportador-de-alimentos-do-mundo-na-proxima-decada-aponta-onu. Acesso em: 20 agosto. 2017.

PRESMAN, Roger. **Engenharia de Software. Uma Abordagen Profissional**. 7ª Edição. AMGH Editora Ltda. 2011.

PRESSMAN, Roger, MAXIM, Bruce. **Engenharia de Software**, 8th ed. AMGH, 10/2016. VitalSource Bookshelf Online.

ROCHA, Laudécia et al. **Panorama da criação de aves e suínos caipiras em regiões periurbanas no município de Senador Canedo (GO),Brasil**. Investigação Qualitativa em Ciências Socias. Atas CIAIQ 2016.

SARTIN, Karka. **Escala de produção, tecnologia e desempenho da avicultura de corte em goiás.** Universidade Federal de Goiás. Goiania. 2016.

SILVEIRA, Valdemir. **Sistema para supermercados modularizado desenvolvido em Java**. Pato Branco. 2014

SOARES, Augusto. **Sistema Web para controle de matrículas e emissão de boletim escolar**. Trabalho de conclusão de curso. Universidade Tecnológica Federal do Paraná. Pato Branco. 2014.

SOMMERVILLE, Ian**. Engenharia de Software**. 9ed. São Paulo:Pearson Pretice Hall, 2011.

SOUZA, S. **Sistemas de Informações Gerenciais no Agronegócio: Estudo de Caso de Aplicação de Software em Administração Rural pelos Produtores de Grãos do Município de Rio Verde–GO**. 2013. Fundação

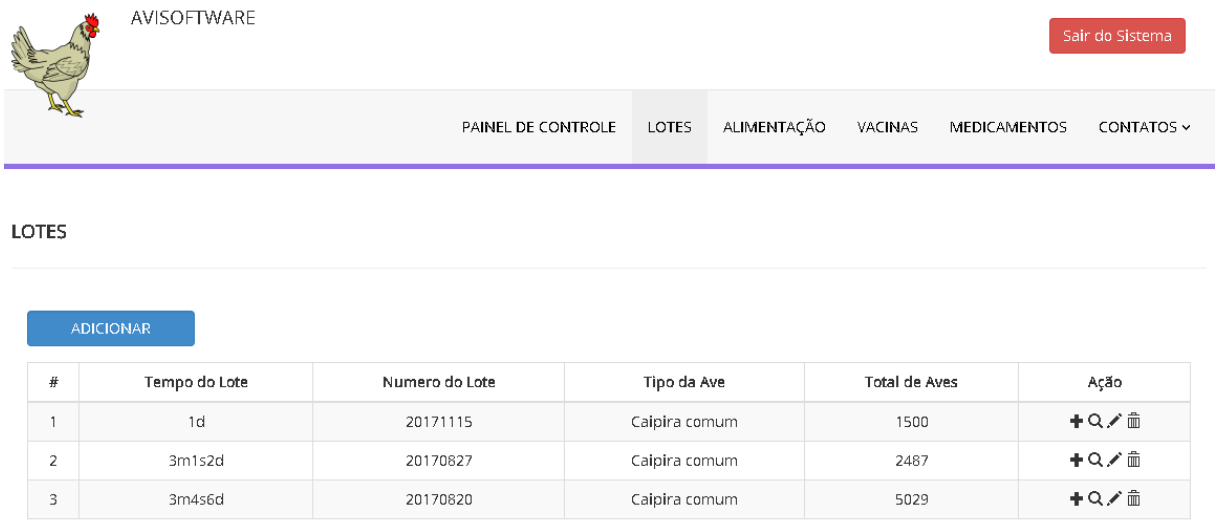
SPRINGBOOT. 2018. Disponível em: < https://spring.io/projects/spring-boot/>

VALE, Bernado. **Ambiente computacional para análise de dados spaciais medianeira**. Curso Superior de Tecnologia em Análise e esenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná. 2013

ZAMBALDE, Andre et al. **Tecnologia da Informação no agronegócio**. Embrapa. 2011

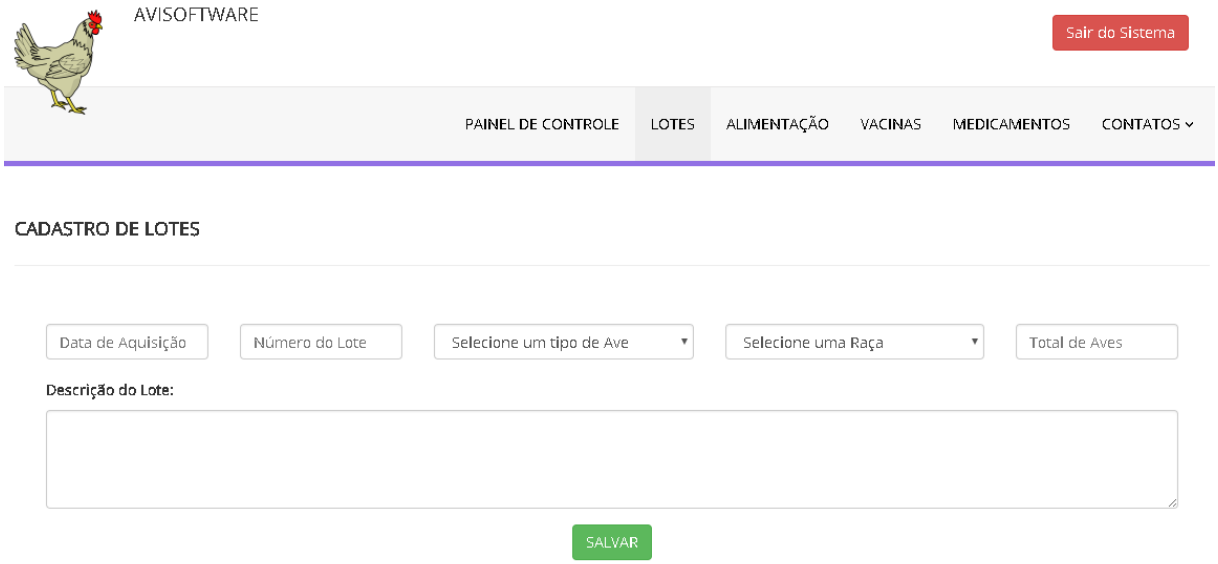
APÊNDICE A – Protótipos do Sistema

1. Protótipo – Listagem dos Lotes



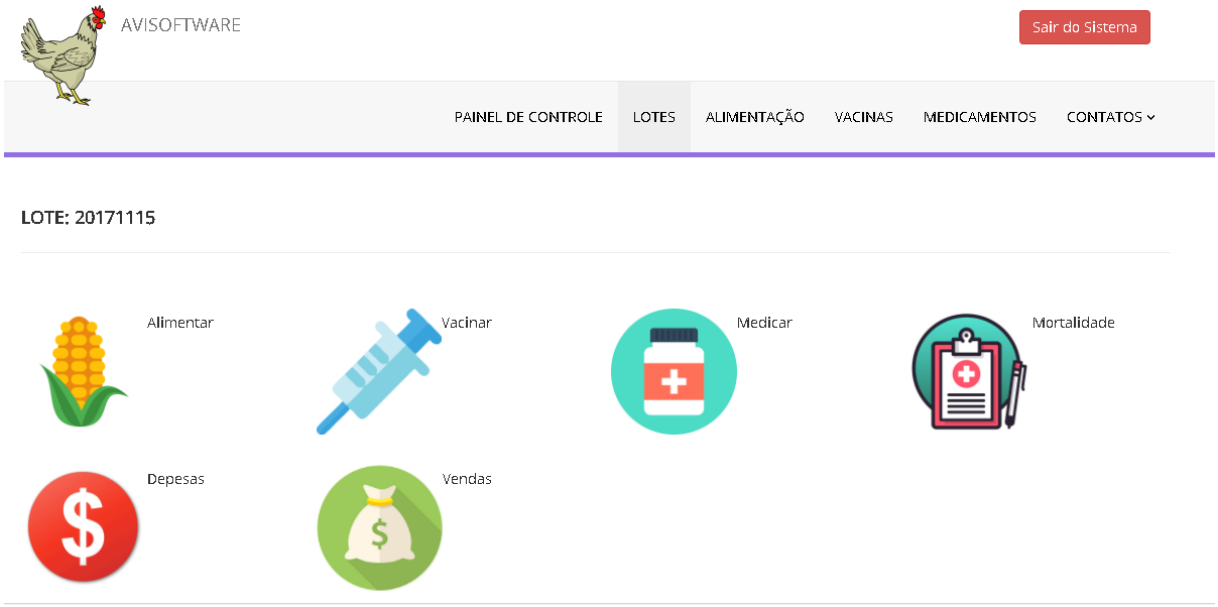
Fonte: Autor da Pesquisa

2. Protótipo – Cadastro dos Lotes



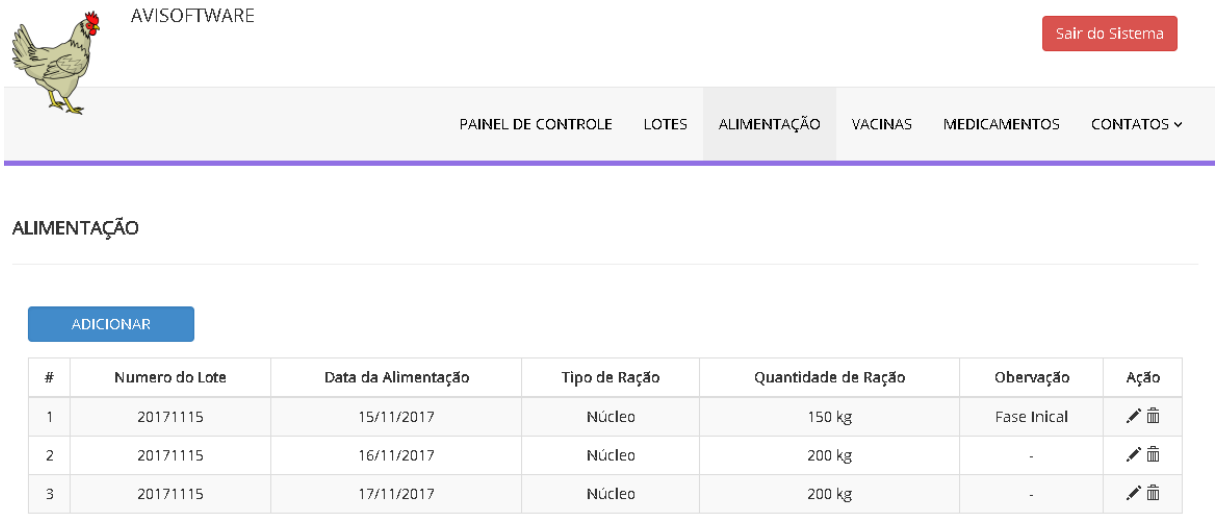
Fonte: Autor da Pesquisa

3. Protótipo – Menu do Lotes



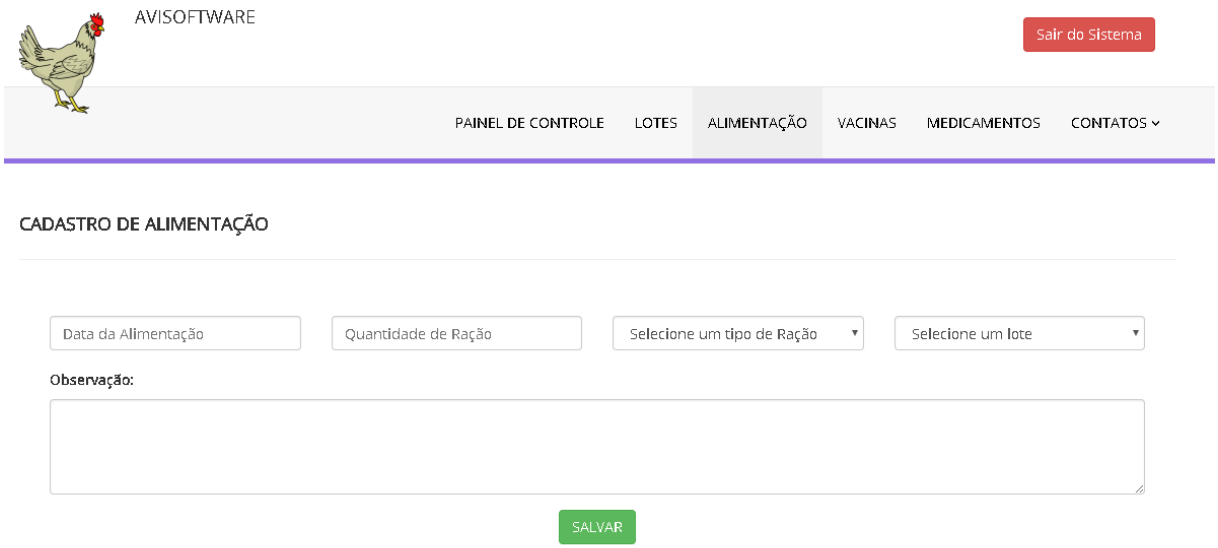
Fonte: Autor da Pesquisa

4. Protótipo – Listagem das Alimentações



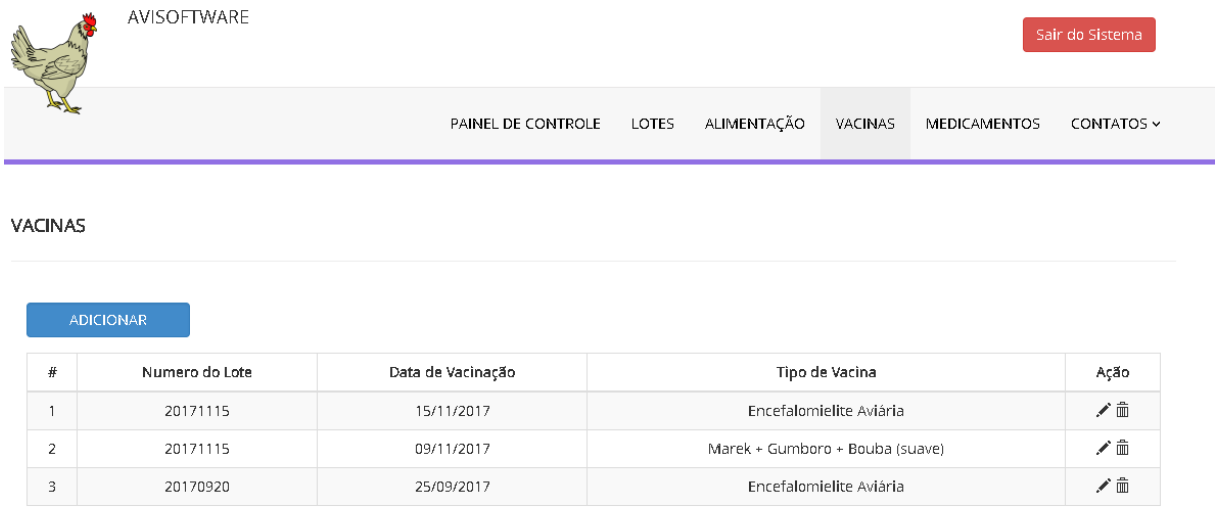
Fonte: Autor da Pesquisa

5. Protótipo – Cadastro das Alimentações



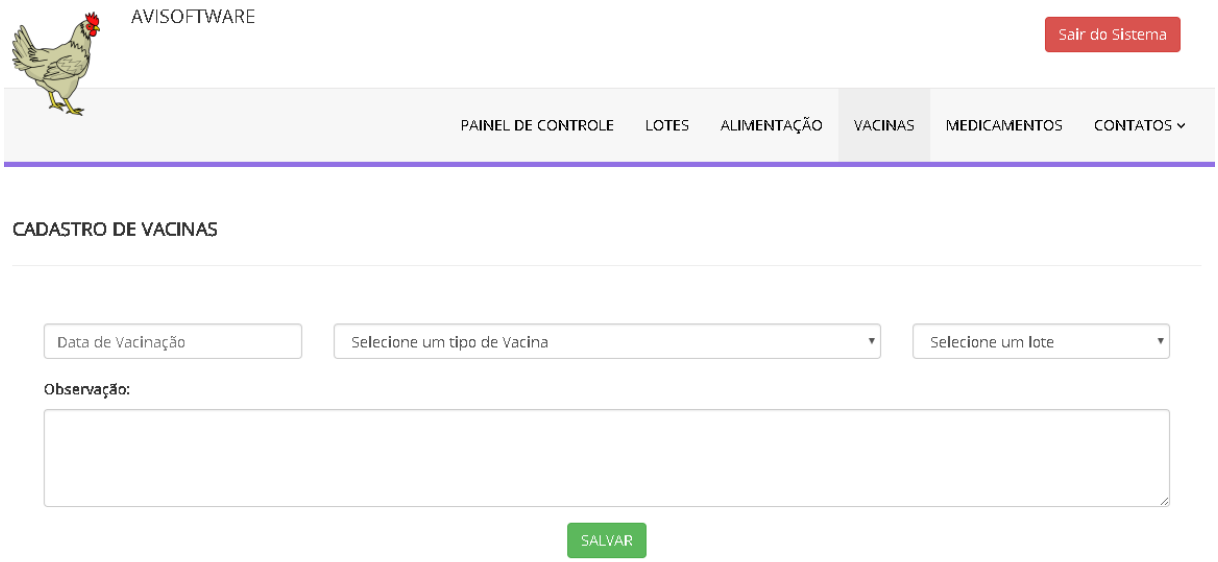
Fonte: Autor da Pesquisa

6. Protótipo – Listagem das Vacinas



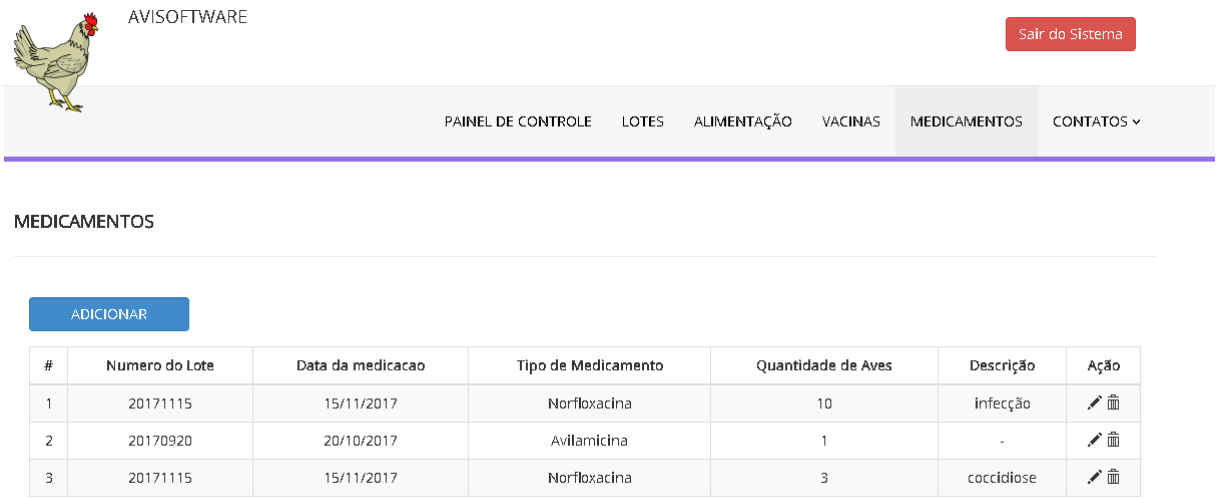
Fonte: Autor da Pesquisa

7. Protótipo – Cadastro das Vacinas



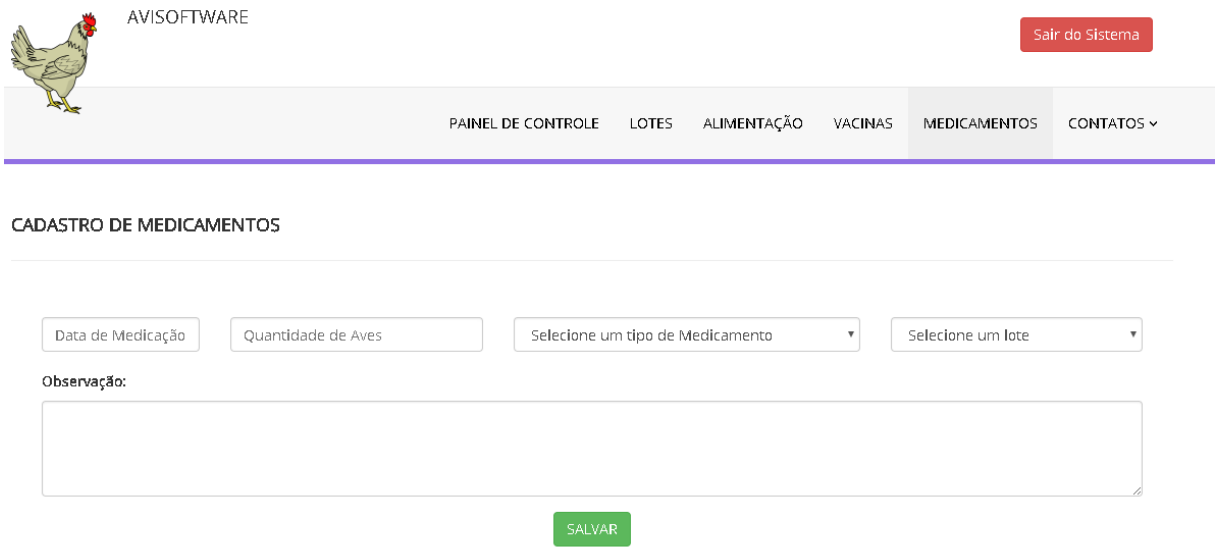
Fonte: Autor da Pesquisa

8. Protótipo – Listagem das Medicações



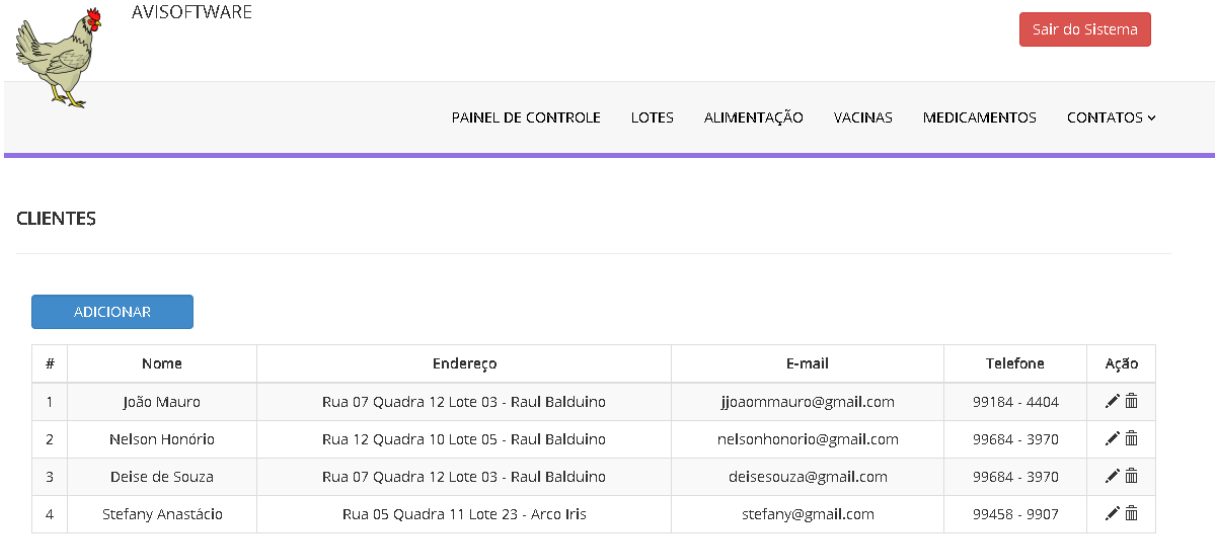
Fonte: Autor da Pesquisa

9. Protótipo – Cadastro das Medicações



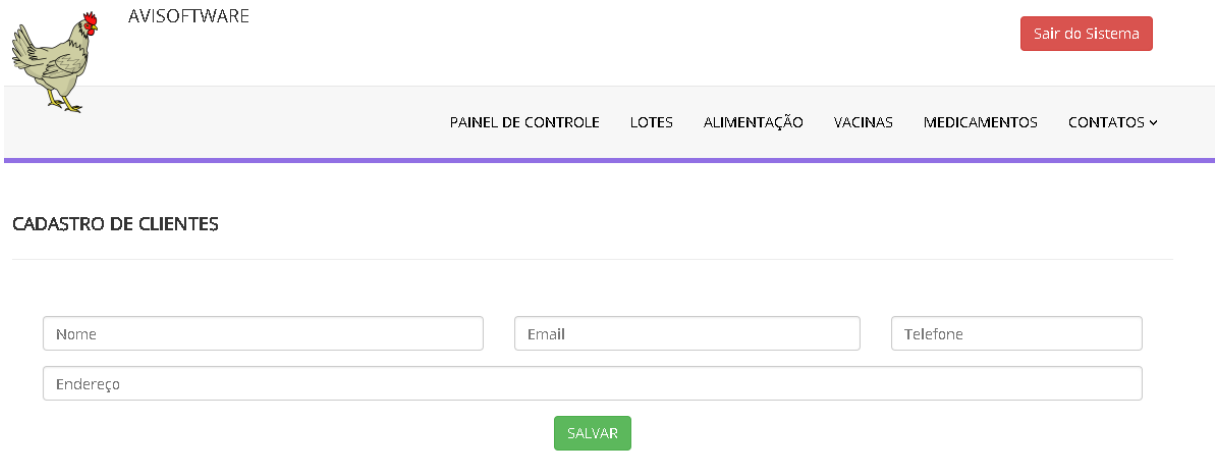
Fonte: Autor da Pesquisa

10. Protótipo – Listagem dos Clientes



Fonte: Autor da Pesquisa

11. Protótipo – Cadastro dos Clientes

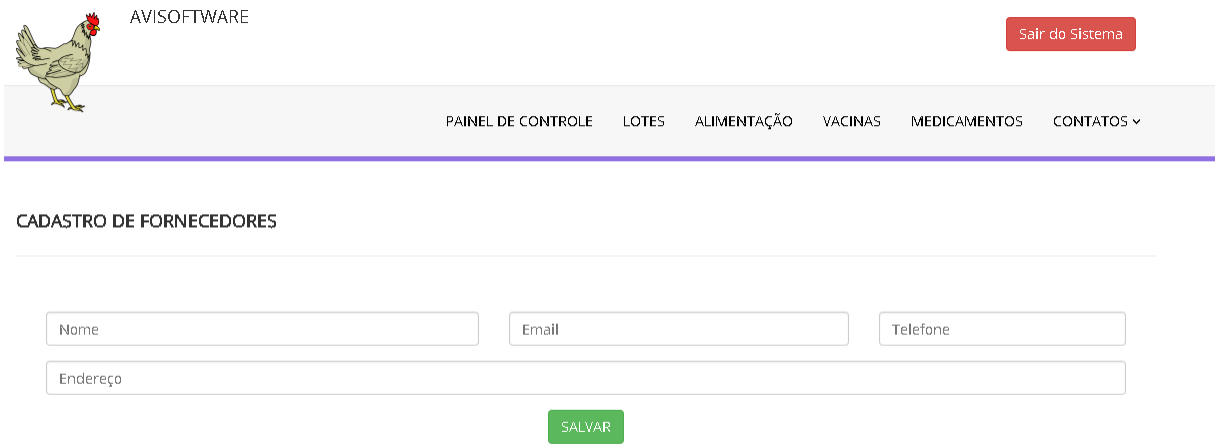


Fonte: Autor da Pesquisa

12. Protótipo – Listagem dos Fornecedores

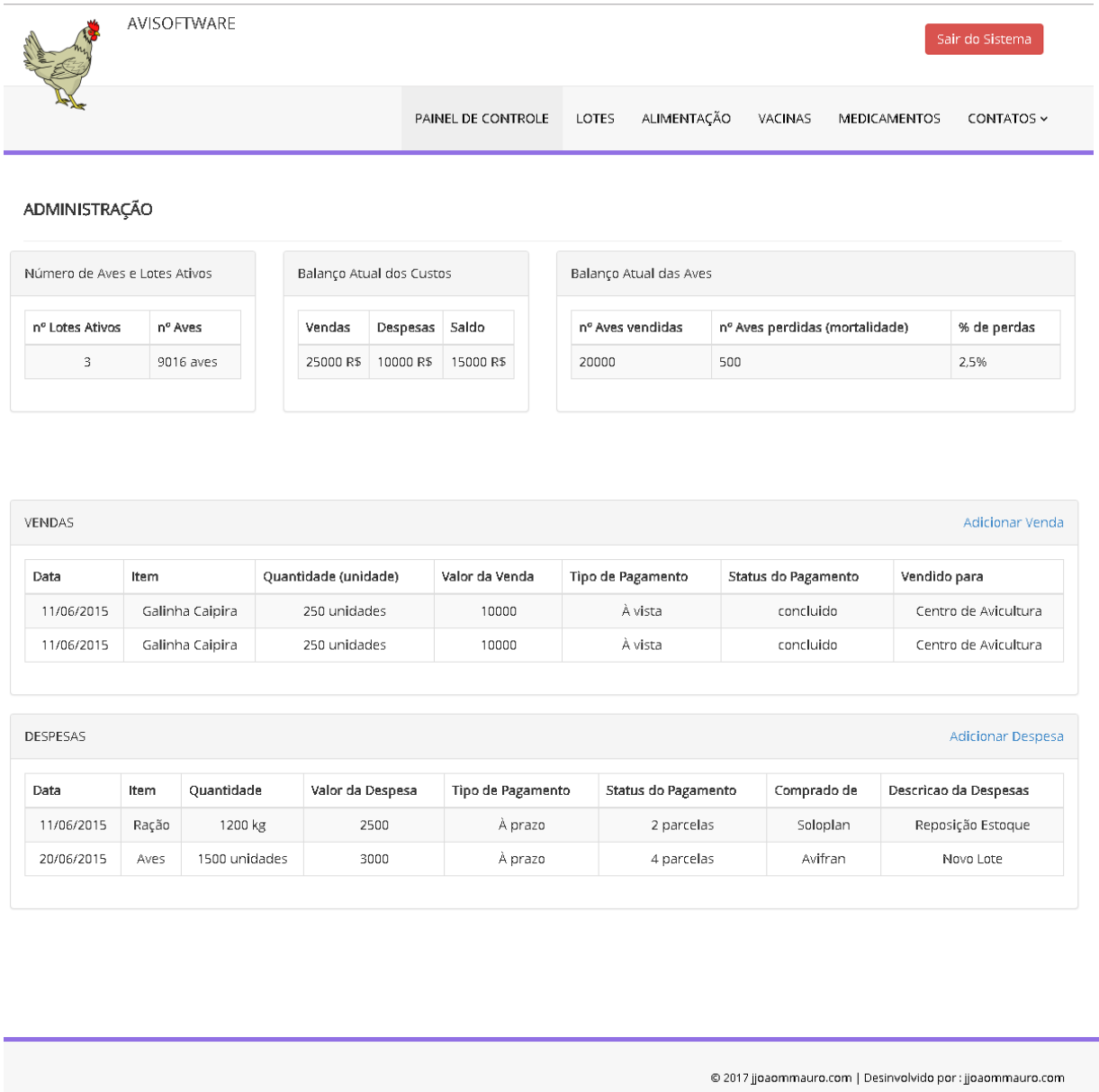
Fonte: Autor da Pesquisa

13. Protótipo - Cadastro dos Fornecedores



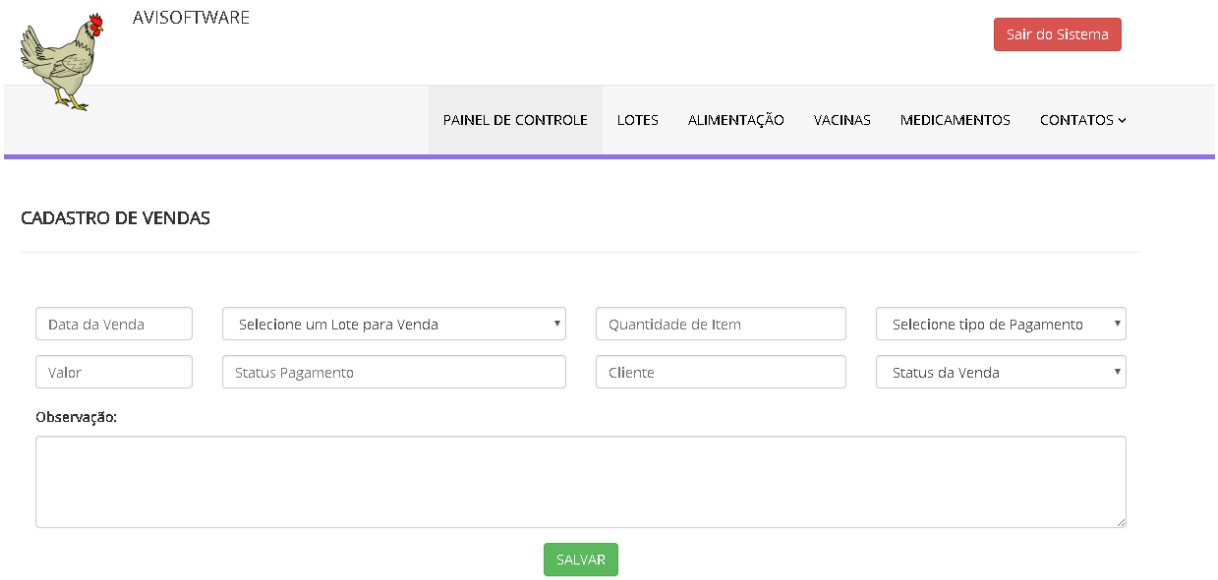
Fonte: Autor da Pesquisa

14. Protótipo – Tela Administrador



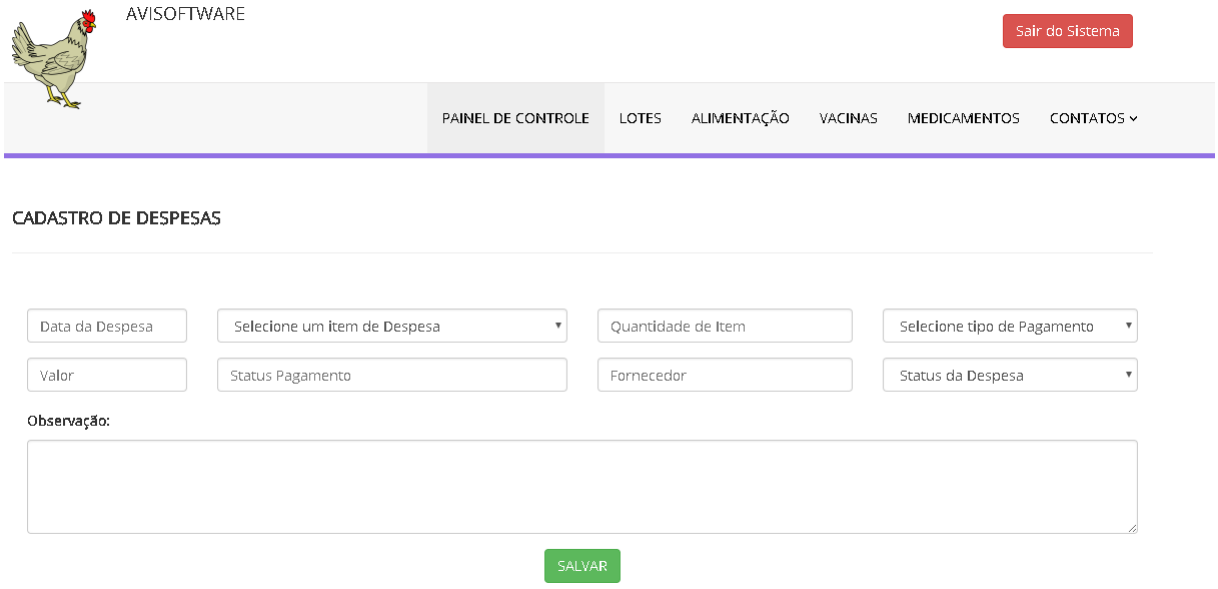
Fonte: Autor da Pesquisa

15. Protótipo – Tela Cadastro Venda



Fonte: Autor da Pesquisa

16. Protótipo – Tela Cadastro Despesa



Fonte: Autor da Pesquisa