Perquirere, 13 (1): 122-134, jul. 2016

http://perquirere.unipam.edu.br

Desenvolvimento de solução multiplataforma para o setor agropecuário

Development of multiplatform solution for the agricultural sector

Gabriel Petrovick Oliveira dos Santos
Bacharel em Sistemas de Informação (UNIPAM).
E-mail: petrovickg@hotmail.com

Eduardo Henrique Silva
Especialista em Engenharia de Software e docente do curso de Sistemas de Informação (UNIPAM).
E-mail: eduardohs@unipam.edu.br

Resumo: Este artigo apresenta o desenvolvimento de um sistema *web* multiplataforma para controlar compra, transferência e produção de ração, núcleo e ingrediente, gerenciamento de estoque e emissão de relatórios, propiciando melhor gestão das atividades desenvolvidas pelas empresas do ramo agropecuário e a visualização de informações gerenciais através de um aplicativo *mobile*. Na produção do *software*, foram utilizadas tecnologias com o objetivo de diminuir o acoplamento entre as camadas, a partir da criação de *web services*, da injeção de dependência e da utilização do padrão MVC (*Model, View e Controller*). Desse modo, foi construído um sistema para que funcione de forma rápida, de fácil manutenção e prático para utilização das empresas dessa área por permitir o controle *online*.
Palavras-chave: Computação Distribuída. Spring Framework. Multiplataforma.

Abstract: This article presents the development of a multiplatform web system for controlling the purchase, transferring, and the production of feed, core and ingredient, managing inventory and the emission of reports, providing management of the activities developed by agricultural companies and the visualization of management information through a mobile application. While creating the software, some technologies were used to decrease the dependencies on the software layers, such as, the creation of web services, dependency injection and MVC (Model, View, Controller) pattern. That way, it was built a system to work quickly, easy to maintain and practical for use of this area by companies to allow online control.
Keywords: Distributed computing. Spring Framework. Multiplatform.

---

1 INTRODUÇÃO

Com a crescente evolução da tecnologia, os usuários passaram a utilizar cada vez mais os sistemas de informação em rotinas do cotidiano e de trabalho, nas quais buscam comodidade, praticidade e maior rapidez nas atividades desenvolvidas, sejam elas de lazer ou de trabalho.

A procura e o uso de tecnologia nas atividades desempenhadas por usuários trazem como resultado um aumento no fluxo de dados dentro e fora das empresas, ou

seja, nas nuvens (*Cloud*). Assim, há a necessidade de se recorrer a alternativas viáveis, capazes de atender a essa crescente demanda.

A informatização das empresas do ramo agropecuário permite obter, dentre outras vantagens, uma melhor gestão das rotinas desempenhadas e ganho em produtividade, tanto das atividades realizadas quanto da mão de obra empregada. Isso contribui para uma melhor qualidade de serviços internos e, consequentemente, aos clientes dessas empresas, beneficiando, assim, todos os envolvidos.

Existem *softwares* no mercado que realizam atividades similares ao descrito neste artigo. Porém, alguns deles não fornecem informações em tempo hábil e mantêm os dados de forma descentralizada, gastando um tempo considerável para consolidação desses dados para a gerência.

Baseado nesse contexto, este artigo visa descrever a elaboração de um sistema voltado ao ramo agropecuário com acesso distribuído em multiplataforma, isto é, disponível tanto em computadores quanto em celulares. Nessa área de plataformas móveis, percebe-se uma busca cada vez maior pela informatização, sendo que as empresas buscam atingir seus objetivos imediatos por meio da utilização da tecnologia e da capacidade de expansão com o intuito de atender às futuras solicitações de forma contínua.

2 REFERENCIAL TEÓRICO

Nesta seção, são descritas as arquiteturas e metodologias utilizadas no desenvolvimento do Sistema Agropecuário. São elas: arquitetura MVC (*Model, View e Controller*), injeção de dependência, computação distribuída e REST (*Representational State Transfer*).

2.1 MVC

O padrão arquitetural MVC visa fragmentar a aplicação em camadas, a fim de facilitar a programação em equipe, de tornar o código limpo, além de permitir o reaproveitamento de código.

Segundo Gamma *et al.* (2000, p. 20), o MVC é composto de três objetos: o *Model* (Modelo), a *View* (Visão) e o *Controller* (Controlador). O Modelo é a representação em objeto da vida real, a Visão representa a interface do usuário e o Controlador tem por objetivo tratar as entradas dos usuários e fazer a intermediação entre o Modelo e a Visão, isto é, fazer a mediação da comunicação entre o que o usuário faz na Visão e como o Modelo responde. Anteriormente ao MVC, todos os objetos eram agrupados de forma a não separar as camadas. Com a utilização dessa arquitetura, tem-se como benefícios a organização, a flexibilidade e a reutilização.

2.2 INJEÇÃO DE DEPENDÊNCIA

Na injeção de dependência, os objetos têm suas dependências no ato da construção e permitem ao desenvolvedor desacoplar e focar no que é importante na aplicação (DEINUM *et al.*, 2012, p. 29, tradução nossa).

A ideia básica dessa técnica é alterar o modo como os objetos são criados. Em vez de usar a sintaxe "*new*" para criar os objetos, utilizam-se injetores com o objetivo de injetar uma implementação específica no objeto de destino (YENER; THEEDOM, 2015).

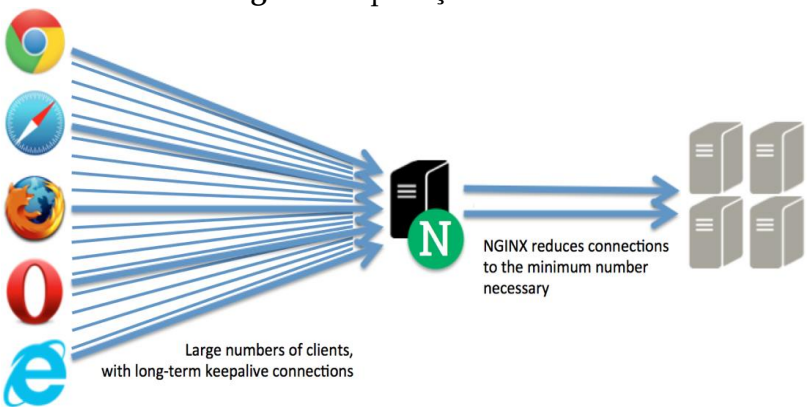
2.3 COMPUTAÇÃO DISTRIBUÍDA

De acordo com Colouris *et al.* (2013), na computação distribuída, os computadores são coordenados a partir da utilização de mensagens enviadas via rede, quando estão interligados.

Com essa tecnologia, é possível utilizar diversos computadores para uma atividade afim, ou seja, se há muitas requisições e um único servidor, torna-se lento ou incapaz de respondê-los, então a computação distribuída auxilia na divisão de requisições entre os servidores configurados, minimizando, assim, os riscos de *overload* nos servidores.

Uma das aplicações que funciona dessa forma é a aplicação NGINX, que é um servidor *web* rápido e leve que permite inúmeras configurações em prol de se obter melhor performance, permitindo o controle de balanceamento dos servidores, evitando a sobrecarga e minimizando uma possível falha. Como pode ser observado na Figura 1, ao utilizar o NGINX, um único servidor recebe as requisições de vários clientes e as mesmas são redirecionadas para diversos outros servidores, possibilitando acelerar a entrega do conteúdo.

Figura 1: Aplicação NGINX



Fonte: GARRETT, 2014.

Com a utilização da aplicação NGINX, é possível acelerar a entrega do conteúdo, aumentar a segurança e fornecer escalabilidade à aplicação.

2.4 REPRESENTATION STATE TRANSFER - REST

O estilo de arquitetura REST descreve como usuários devem projetar e desenvolver arquitetura para a *web* moderna. Ela tem ênfase em escalabilidade das interações de componentes, generalização das interfaces, implementação independente

de componentes e componentes intermediários para reduzir a latência das interações, reforçar a segurança e encapsular sistemas legados (FIELDING, 2000, tradução nossa).

Há uma tendência em usar interfaces simples e genéricas em sistemas distribuídos, e isso é exemplificado pela interface mínima oferecida pela *World Wide Web* e pela estratégia REST nos serviços *web*. Essa estratégia contribui para o baixo acoplamento por reduzir a dependência em relação a nomes de operação específicos (COLOURIS *et al.*, 2013).

Dessa forma, o REST surgiu para facilitar o desenvolvimento de serviços *web* usados para construir sistemas de forma distribuída.

3 METODOLOGIA

O *software* proposto ao ramo agropecuário foi desenvolvido baseado em conceitos de Engenharia de Software. Assim, esse processo de desenvolvimento foi divido nas seguintes fases: requisito, análise, desenvolvimento e transição, permitindo um melhor gerenciamento do que seria feito, conforme demonstrado no Quadro 1.

Quadro 1: Divisão das Fases

FASE	DESCRIÇÃO
Requisito	• Visitas técnicas com o intuito de conhecimento do ambiente da empresa; • Reuniões junto ao cliente com a finalidade de identificação da proposta do novo sistema.
Análise	• Organização dos requisitos levantados; • Divisão de tarefas; • Elaboração do diagrama de entidade de relacionamento – DER.
Desenvolvimento	• Codificação do sistema de acordo com o que foi levantado nas fases de requisito e análise; • Criação do Banco de Dados; • Reuniões com a intenção de definir prioridades para o desenvolvimento seguinte; • Testes de stress; • Entrega do software à empresa solicitante, de acordo com as prioridades.
Transição	• Treinamento aos usuários; • Evolução do sistema.

Fonte: Dados do trabalho

Além da divisão das fases, para melhor gerenciamento ao longo da elaboração do *software*, foram utilizadas as ferramentas listadas no Quadro 2, as quais tornaram possível a idealização do sistema e a conclusão de acordo com o esperado.

Quadro 2: Ferramentas

FERRAMENTA	DESCRIÇÃO
Eclipse	Ferramenta utilizada no desenvolvimento
Java	Linguagem de programação
JMeter	*Framework* usado nos testes de *stress* (Simulação de diversos usuários)
Tomcat	*Container web*
NGINX	Servidor *Proxy* utilizado para fazer *load balancing* (Acesso distribuído)
Xcode	Desenvolvimento de aplicativo para *iOS*
Android Studio	Desenvolvimento de aplicativo para A*ndroid*

Fonte: Dados do trabalho

Após o término de cada requisito, foi realizado um treinamento e disponibilizada a funcionalidade, com a finalidade de a empresa ter acesso e validar o requisito desenvolvido.

4 DESENVOLVIMENTO E RESULTADO

Neste tópico estão detalhadas as fases seguidas ao longo do desenvolvimento do *software*.

4.1 REQUISITO

Na fase inicial do desenvolvimento do *software*, o foco é entender o que o cliente deseja, como e para quando deveria ser feito. Desse modo, a fim de abstraírem-se os requisitos necessários para as fases seguintes, foram feitas visitas técnicas e entrevistas com os funcionários e com o proprietário da empresa, levantando-se registros com o intuito de realizar a estruturação do projeto a ser executado.

Ao término da etapa de levantamentos de requisitos, foi dado início à fase seguinte, ou seja, à análise.

4.2 ANÁLISE

Nesta fase, os requisitos levantados anteriormente foram analisados e organizados, de forma a contribuir para a organização e para o melhor gerenciamento do que seria desenvolvido nas fases seguintes.

De acordo com a análise feita, os requisitos foram divididos em tarefas para a aplicação *web*, conforme mostrado no Quadro 3.

Quadro 3: Divisão de Tarefas – WEB

TAREFAS				
Cadastro de Grupo	Cadastro de Categoria	Cadastro de Ingrediente	Cadastro de Núcleo	Cadastro de Ração
Cadastro de Compra de Ingrediente	Cadastro de Compra de Núcleo	Cadastro de Compra de Ração	Cadastro de Transferência de Ingrediente	Cadastro de Transferência de Núcleo
Cadastro de Transferência de Ração	Cadastro de Produção de Núcleo	Cadastro de Produção de Ração	Gráfico de Compra	Gráfico de Transferência
Gráfico de Produção	Relatório de Produto	Relatório de Fórmula	Relatório de Compra	Relatório de Transferência
Relatório de Produção	Relatório de Estoque	Relatório de Estoque de lançamento	Relatório de Custo de Produção	Relatório de Unidade / Produto
Relatório de Categoria / Produto	Cadastro de Unidade	Cadastro de Usuário	Cadastro de Fornecedor	

Fonte: Dados do trabalho

Além dos requisitos feitos para a aplicação *web*, foram levantadas também necessidades para aplicativos *mobile*, conforme mostra o Quadro 4.

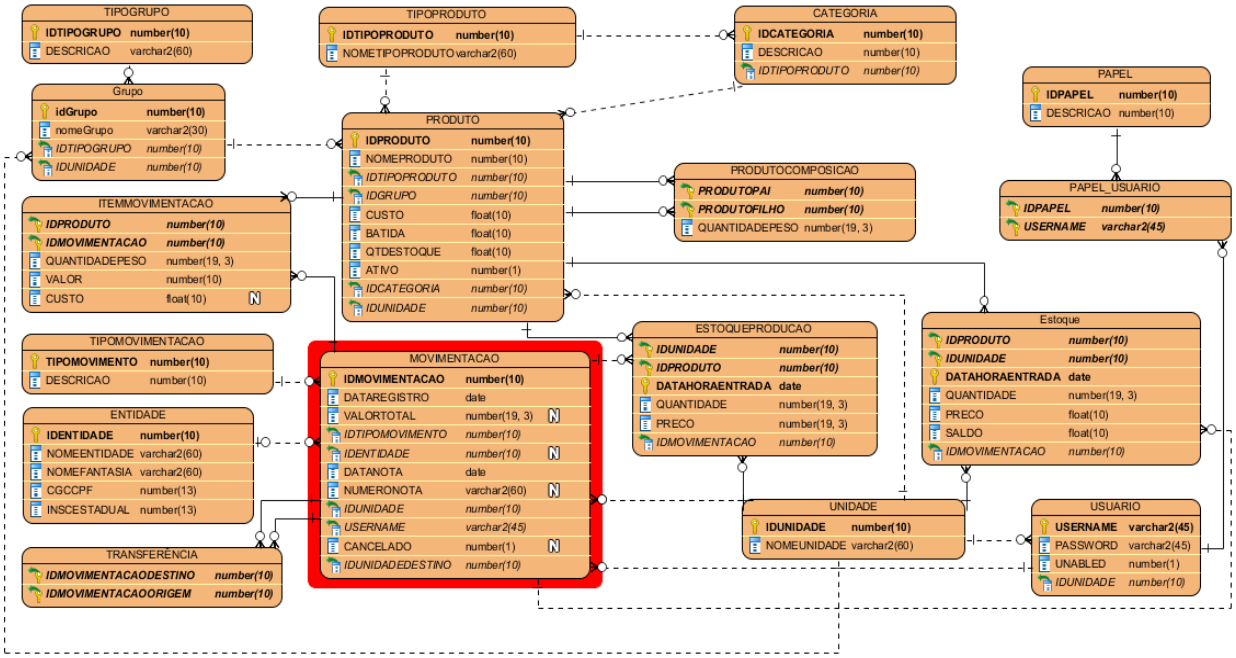
Quadro 4: Divisão de Tarefas – Mobile

TAREFAS				
Pesquisa de Movimentação	Pesquisa de Produto	Quantidade de Movimentação (Gráfico)	Quantidade de Quilos Movimentados Mensalmente	Número de Movimentação por Unidade (Gráfico)
Número de Movimentação por Unidade (Tabela)	Custo de Movimentação Mensal (Gráfico)			

Fonte: Dados do trabalho

Após ser feita a análise dos requisitos, a fim de se concluir essa etapa, foi feita a elaboração do diagrama de entidade e relacionamento (DER), conforme a Figura 2.

Figura 2: DER



Fonte: Dados do trabalho

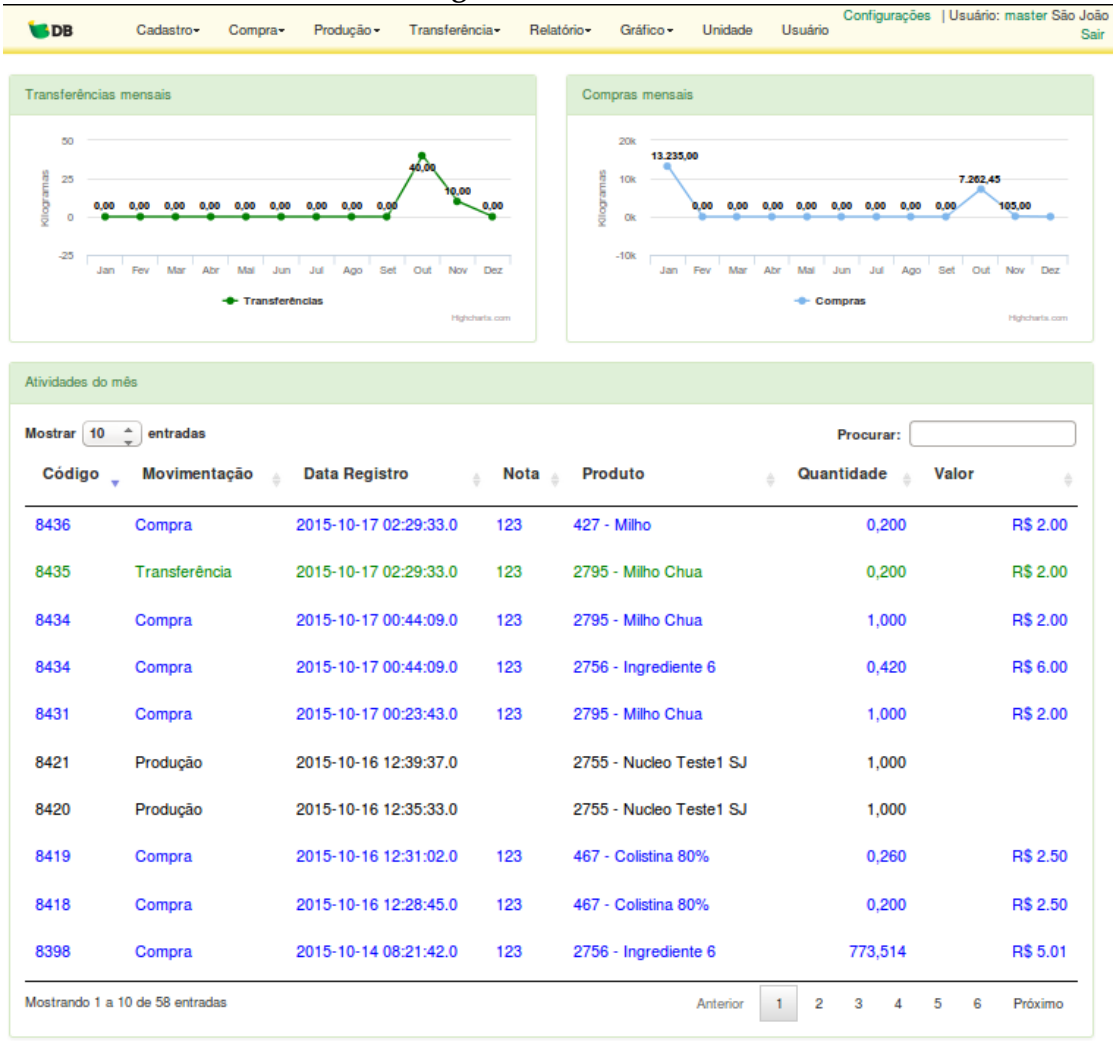
Como se pode observar no diagrama acima, a tabela *movimentação* armazena, em uma única tabela, a compra, a transferência e a produção de ingredientes, núcleos e rações, para facilitar o desenvolvimento desses requisitos. Com sua finalização, foi iniciada a fase seguinte: o desenvolvimento.

4.3 DESENVOLVIMENTO

A fase de desenvolvimento foi iniciada após o término das fases de requisito e de análise, que permitiram a compreensão da necessidade do novo sistema e a organização do que seria desenvolvido.

Na tela inicial do *software web*, além do resumo contendo o gráfico de transferências, compras mensais e movimentações do mês, conforme mostra a Figura 3, há também algo de muita importância para o negócio desse tipo de empresa, que são os *feedback*s que esse sistema traz logo de início do mesmo; podendo, assim, ter uma maior noção do que acontece dentro da empresa sem muitos *clicks* dentro da plataforma.

Figura 3: Tela Inicial



Fonte: Dados do trabalho

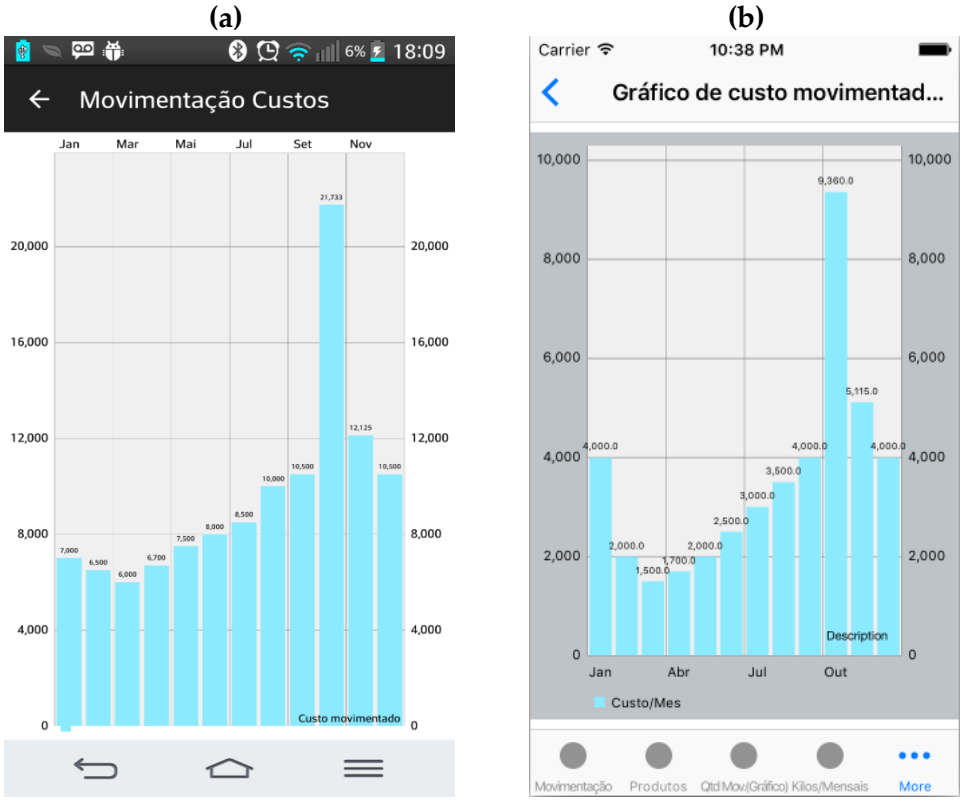
Na Figura 3, pode-se observar que a tela inicial apresenta as seguintes guias e funcionalidades:

- Cadastro: permite cadastrar fornecedor, grupo, categoria, ingrediente, núcleo e ração.
- Compra: registra as compras de ingrediente, núcleo e ração.
- Produção: registra a produção de núcleo e ração.
- Transferência: registra as transferências de ingrediente, de núcleo e de ração entre as unidades.
- Relatório: permite a emissão de relatórios de fórmula, de produto, de compras, de transferências, de produção, de estoque, de estoque lançamento, de custo de produção estoque, de unidade/produto e de categoria/produto.
- Gráfico: apresenta, em formato de gráfico, as transferências, as compras e as produções.
- Unidade: permite cadastrar e editar unidades.

- Usuário: permite cadastrar os usuários do sistema, bem como as permissões de usuário e unidade pertencente.
- Configuração: permite ao administrador escolher as unidades que poderão ter estoque negativo.

As aplicações A*ndroid* e *iOS* tiveram funções voltadas para a área gerencial, como a representação gráfica de movimentações mensais com os custos, conforme apresentado na Figura 4, sendo a Figura 4(a) o dispositivo *Android* e a Figura 4(b) o dispositivo *iOS.*

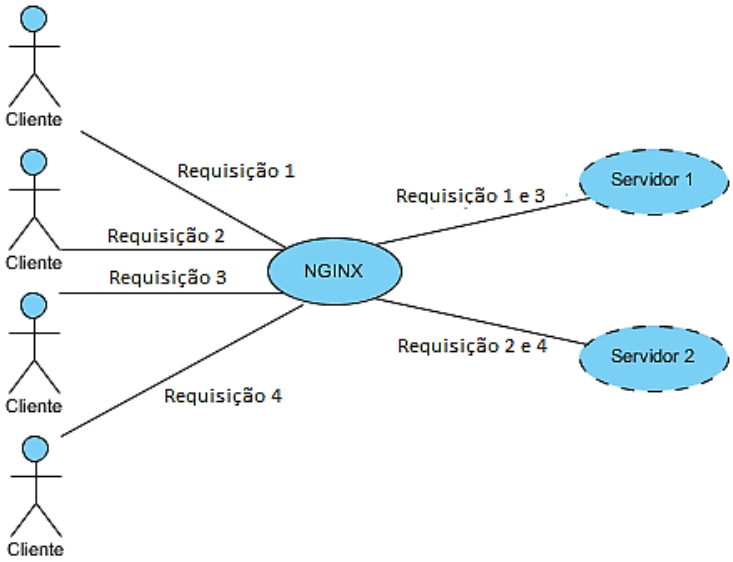
Figura 4: Android/iOS



Fonte: Dados do trabalho

Baseado na computação distribuída, o projeto contou com a utilização de uma arquitetura que permite a distribuição de carga dos servidores, podendo, os mesmos, dividir as tarefas sem sobrecarregar e afetar o desempenho do sistema. Assim, foi empregada a aplicação NGINX, possibilitando fazer um *load balancing*, dividindo as requisições entre diversos servidores. Por exemplo, num ambiente composto por três servidores, são feitas quatro requisições simultâneas, sendo que um dos servidores será utilizado para distribuir as requisições e os outros dois receberão duas requisições cada, consequentemente diminuindo a sobrecarga dos mesmos, conforme mostra a Figura 5.

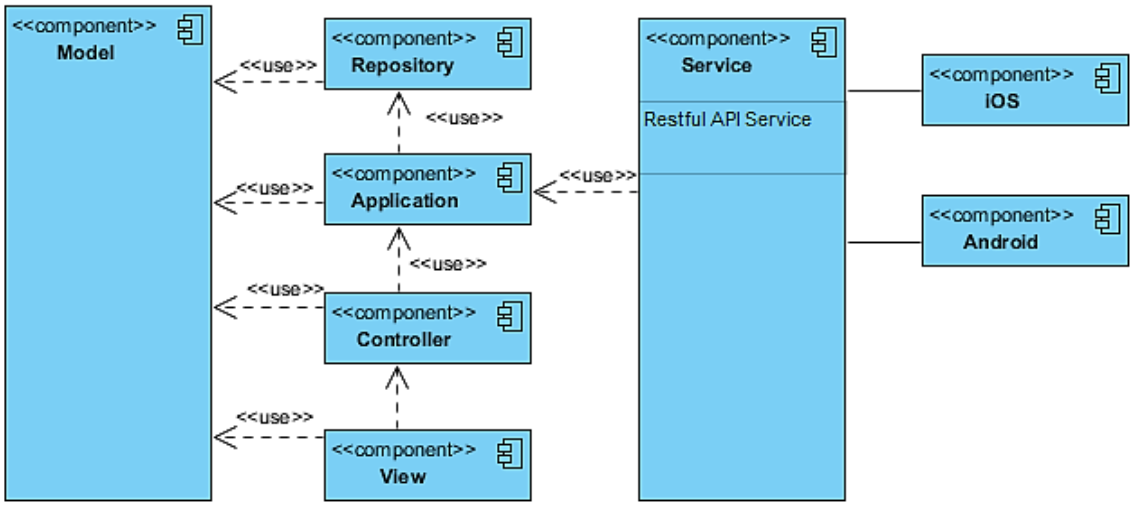
Figura 5: Load Balancing



Fonte: Elaboração própria, 2015

Na arquitetura interna do sistema, foi feita uma divisão visando diminuir o acoplamento e facilitar a manutenção, conforme mostra o diagrama de componentes da Figura 6. O componente V*iew* (*browser*) requisita o controlador que permite buscar, salvar, editar e deletar dados pelo componente A*pplication.* Essa camada contém as regras de negócio do sistema, que utiliza a camada *Repository* que, por sua vez, é responsável por fazer a transação com o banco de dados. Já o componente *Business* possui a estrutura do banco, ou seja, é uma cópia em forma de orientação a objetos do banco de dados. Por fim, o componente *Service* é responsável por responder às requisições feitas em REST para sistemas externos, como *Android* e iOS.

Figura 6: Diagrama de Componentes

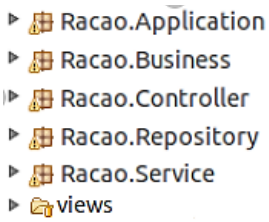


Fonte: Elaboração própria, 2015

Na codificação, o sistema foi dividido nas seguintes camadas: *Model*, *View*, *Controller*, *Application*, *Repository* e *Service*, baseando-se na arquitetura MVC, como

mostra a Figura 7, contribuindo para a organização do código e, consequentemente, na facilidade de manutenção.

Figura 7: Arquitetura do Sistema



Fonte: Dados do trabalho

No desenvolvimento da aplicação, foi usada a injeção de dependência para que os objetos tivessem suas dependências no ato da construção da aplicação, tornando as camadas do *software web* menos acopladas, conforme trecho do código da Figura 8.

Figura 8: Código com Injeção de Dependência

```
31    @Autowired
32    public IProdutoRepository produtoRepository;
33    @Autowired
34    public IItemMovimentacaoRepository itemMovimentacaoRepository;
35    @Autowired
36    public IMovimentacaoRepository movimentacaoRepository;
37    @Autowired
38    public IProdutoComposicaoRepository produtoComposicaoRepository;
39    @Autowired
40    public ITipoProdutoRepository tipoProdutoRepository;
41    @Autowired
42    public IGrupoRepository grupoRepository;
43    @Autowired
44    public IUnidadeRepository unidadeRepository;
45    @Autowired
46    public IEstoqueRepository estoqueRepository;
47    @Autowired
48    public IEntidadeRepository entidadeRepository;
```

Fonte: Dados do trabalho

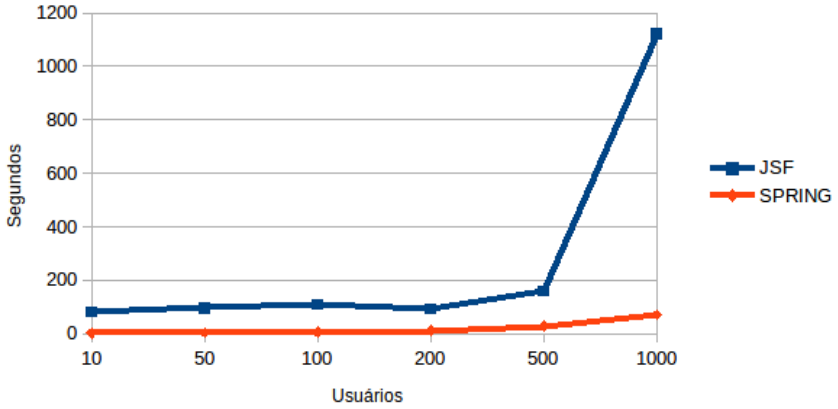
Em toda a estrutura das camadas, foi utilizado o *Spring*, um *framework open-source* mantido pela empresa *Pivotal Software Incorporation*. Atualmente, a Oracle disponibiliza para, desenvolvimento *web*, o *framework* JSF (*Java Server Faces*), baseado na arquitetura MVC. Devido a testes realizados em funções específicas, o *framework Spring* mostrou desempenho elevado em relação ao *framework* utilizado na linguagem oficial.

Para concluir essa etapa, foram feitos testes de *stress*. A Figura 9 mostra um gráfico com os resultados obtidos, permitindo comparar o *Spring Framework* (*Spring Data* e *Spring* MVC), usado para desenvolvimento do sistema, com o JSF/*Hibernate*. Nele estão descritos os tempos de resposta dos *frameworks* citados, possibilitando a comparação dos mesmos.

Como pode ser observado, o *Spring Framework* torna-se cada vez mais rápido à medida que mais usuários acessam a aplicação, em relação ao JSF/*Hibernate*. De acordo com os testes em que o acesso simultâneo *online* variou entre 10 e 1000 usuários, o

tempo de resposta para o *Spring Framework* foi de 0,12 a 2,44 segundos, enquanto que, para o JSF/*Hibernate*, variou de 2,40 a 7,50 segundos, comprovando, assim, a sua eficiência no sistema proposto.

Figura 9: Teste de *Stress*



Fonte: Dados do Trabalho

Com o teste realizado, foi possível comprovar que o *Spring Framework*, em relação ao JSF/*Hibernate*, foi o mais indicado para o *software* descrito, pois possibilita o acesso distribuído, a divisão em camadas e o melhor desempenho.

Após o término dessa etapa, foi realizada a última fase descrita no trabalho, que é a fase de transição.

4.4 TRANSIÇÃO

A última fase da elaboração do projeto, descrita como transição, é marcada pela entrega do sistema proposto, a fim de realizar os testes finais, bem como o treinamento inicial. Assim, os usuários poderão tirar as dúvidas remanescentes, conhecer e aprender a utilizar o novo *software*.

5 CONCLUSÃO

A elaboração do sistema agropecuário *multiplaforma* contou com as aplicações *Android* e *iOS*, que foram desenvolvidas para a parte gerencial, enquanto a aplicação *web* contém todos os requisitos propostos. As mesmas tiveram como objetivo melhorar o gerenciamento das atividades realizadas pelas empresas do ramo agropecuário, por meio de tecnologias reconhecidas no mercado, com capacidade de expansão, possibilitando a atualização de relatórios para a análise em tempo real, visto que alguns *softwares* disponíveis no mercado possuem falhas nas apresentações de resultados gerenciais.

Durante o desenvolvimento da aplicação *web*, foram utilizadas arquitetura MVC, que possibilitam a organização do código do projeto, injeção de dependência e computação distribuída, a fim de garantir a qualidade do sistema desenvolvido, em termos de manutenibilidade, desempenho e escalabilidade. Como forma de validação

quanto ao desempenho, foram feitos testes de *stress* de comparação entre *Spring MVC*/*Spring Data* com JSF/*Hibernate*. Há também, previstos na nova versão do *Java EE* 8, novos conceitos similares aos empregados no *Spring Framework*, enfatizando, assim, sua eficiência.

Já na construção das aplicações *mobile*, foram utilizadas ferramentas nativas, como *Android Studio* para *Android* e *Xcode* para *iOS*, que facilitam a agilidade de produção do sistema para ambas as plataformas.

Ao final, com a utilização do sistema proposto pela empresa do ramo agropecuário, espera-se que o mesmo contribua para uma melhor rotina de trabalho e para um melhor desempenho nas atividades desenvolvidas, trazendo comodidade e praticidade aos seus utilizadores, pela empresa, já que o desenvolvimento do produto baseou-se no cotidiano vivido pelos seus usuários e foram utilizadas as ferramentas de desenvolvimento adequadas a sistemas desse porte.

Como proposta de trabalho futuro sugere-se o desenvolvimento de um sistema de controle de animais, utilizando os mesmos princípios do projeto descrito neste artigo. Na proposta a ser realizada, o *software* irá permitir o controle da reprodução (amamentação, desmama e reprodutores), ganho de peso (engorda e alimentação) e sanidade (medicação), organizando, assim, os processos da rotina dessa área.

REFERÊNCIAS

COLOURIS, George *et al. Sistemas distribuídos, conceitos e projetos.* Porto Alegre: Bookman, 2013. 1032 p.

DEINUM, Marten *et al. Pro Spring MVC With Web Flow.* New York: Apress, 2012. 596 p.

FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures.* 2000. 180 f. Tese (Doutorado) - Curso de Philosophy, University Of California, Irvine, California, 2000. Disponível em: < http://migre.me/rtbe7 >. Acesso em: 05 set. 2015.

GAMMA, Erich *et al. Padrões de projeto*: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000. 364 p.

GARRETT, Owen. *HTTP Keepalive Connections and Web Performance*. 2014. Disponível em: <https://www.nginx.com/blog/http-keepalives-and-web-performance/#gs.GEtF_JE>. Acesso em: 22 out. 2015.

YENER, Murat; THEEDOM, Alex. *Java EE Design Patterns.* Indianapolis: Wrox, 2015. 264 p.