ANTONIO MENEGHETTI FACULDADE - AMF

CURSO DE BACHAREL EM SISTEMAS DE INFORMAÇÃO

FÁBIO IVAN BORCHARDT

MELHORIA DE SISTEMA LEGADO DO SETOR AGROPECUÁRIO UTILIZANDO API

RESTINGA SECA/RS

2016

FÁBIO IVAN BORCHARDT

MELHORIA DE SISTEMA LEGADO DO SETOR AGROPECUÁRIO UTILIZANDO API

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação como requisito parcial para obtenção do título de Bacharel.
Orientador(a): Profª. Me**. Fernando Luís Herrmann**

RESTINGA SECA/RS

2016

FÁBIO IVAN BORCHARDT

DJANGO REST FRAMEWORK
MELHORIA DE SISTEMA LEGADO

Trabalho de Conclusão apresentado ao curso de Sistemas de Informação como requisito parcial para obtenção do título de Bacharel.

Banca Examinadora:

Orientador (a):
Prof. Ms. Fernando Luís Herrmann
Faculdade Antonio Meneghetti

Membro:
Prof. Esp. José Luiz da Silva Rodrigues Filho
Faculdade Antonio Meneghetti

Membro:
Prof. Ms. Leonardo Guedes
Faculdade Antonio Meneghetti

Restinga Seca/RS
2016

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela vida a mim concebida, aos meus pais, que passaram por tudo e todos pra que eu tivesse uma vida boa e confortável, que nunca mediram esforços para que eu pudesse estudar e concluir esta graduação. A minha namorada Luciana Bulegon, que esteve ao meu lado em boa parte do tempo do curso, e que sempre me apoiou. Aos professores, que me ensinaram muito mais que a matéria obrigatória, me ensinaram a ser um profissional. E por fim, aos amigos que por muitas vezes me convidaram para alguma festa, mas não podia ir, pois precisava fazer algum trabalho ou estudar para alguma prova.

RESUMO

BORCHARDT, Fábio Ivan. **Melhoria de Sistema Legado do setor Agropecuário utilizando API.** 2016. 31 páginas. Trabalho de conclusão do Curso de Sistemas de Informação como requisito parcial para a obtenção do grau Bacharel. Faculdade Antônio Meneghetti. Curso de Sistemas de Informação, Recanto Maestro – Restinga Seca/RS, 2016.

A cada instante nos deparamos com novas tecnologias nos mais diversos setores do mercado. Isso vem se intensificando também no setor agropecuário, mais especificamente o setor da pecuária leiteira, que há pouco tempo não era considerado um meio propenso a consumir certos tipos de tecnologias. Desta forma, o presente trabalho abordará esta questão e desenvolverá uma API sobre um software legado, bem como, apresentar a linguagem Python e o uso do framework Django, que pouco a pouco vem crescendo em meio a tantos outros frameworks com o mesmo nível de agilidade e robustez.

Palavras-chave: Gestão do rebanho; Sistema Web, Django, Python, Pecuária de Leite;

ABSTRACT

BORCHARDT, Fábio Ivan. **Melhoria de Sistema Legado do setor Agropecuário utilizando API.** 2016. 31 páginas. Trabalho de conclusão do Curso de Sistemas de Informação como requisito parcial para a obtenção do grau Bacharel. Faculdade Antônio Meneghetti. Curso de Sistemas de Informação, Recanto Maestro – Restinga Seca/RS, 2016.

Every moment we come across new technologies in the most diverse sectors of the market. This has also intensified in the agricultural sector, more specifically the dairy sector, which was not considered a means of consuming certain types of technology. In this way, the present work will focus on this issue and develop a API on legacy software, as well as presenting the Python language and the use of the Django framework, which has gradually been growing in the midst of many other frameworks with the same level of agility And robustness.

Keywords: Herd management; Web System, Django, Python, Dairy Cattle;

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A tecnologia hoje é um fator decisivo para a manutenção de um sistema de produção, ou seja, o uso, dependendo do meio, determinará se esta produção será eficiente a ponto de se manter a atividade, tanto familiar ou empresarial.

Para assegurar este argumento, empresas investem milhões em tecnologia voltada ao agricultor, e uma parte desta chega ao produtor leiteiro que pouco a pouco vem se fazer presente no dia a dia da fazenda.

A internet é uma grande responsável por este avanço da tecnologia, pois traz informação e comodidade no gerenciamento da propriedade que consequentemente acabam aumentando a produção. Para acompanhar esse avanço é imprescindível que se tenha acesso à informações relevantes a qualquer momento e local, para que se possa tomar alguma medida no exato momento em que algum fato ocorra.

Neste sentido, as API's – Application Programming Interface – surgiram para que se possa acessar dados de forma unificada para que outras empresas possam utilizar suas informações sem se preocupar como que o processo é realizado. Assim como as API's, o MVC – Model-View-Controller – mostra-se uma ótima alternativa dentro dos padrões existentes no mercado, pois separa toda a regra de negócio da parte visual, tornando mais legível o código e facilitando a manutenção.

O framework Django reúne essas características que são extremamente decisivas na nova realidade de desenvolvimento web. Sua contribuição recai sobre o foco nas equipes, que estão cada vez mais desacopladas.

Com base nisso, questiona-se: como desenvolver uma aplicação que implemente esta tecnologia e traga benefícios tanto para o desenvolvedor quanto para o produtor?

Para atender este problema, tem-se os seguintes objetivos.

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

Apresentar uma API que, disponibilize informações importantes sobre o rebanho, que possam contribuir na tomada de decisão em sua fazenda.

1.1.2 OBJETIVO ESPECÍFICO

1). Estudar o contexto do produtor rural e a gestão de um rebanho de gado;

2). Identificar o comportamento de maior potencial do rebanho;
3). Estudar o framework Django;
4). Propor uma API que disponibilize as informações através do framework Django.

1.2 JUSTIFICATIVA

Com o notório avanço de tecnologias que aumentam a produção, tanto agrícola quanto bovina, vários cenários estão mudando, quem não entra na onda da modernização, acaba ficando fora do mercado. Por este motivo, criar uma API que disponibilize os dados de forma simples e prática, para que outros serviços possam utilizar essas informações, seja através de alguma aplicação web ou por um aplicativo de celular, trazendo comodidade, segurança e rapidez na hora de gerenciar seu rebanho. Desta forma, é um grande avanço para aqueles que por vários motivos, não conseguem consumir as tecnologias de ponta que o mercado oferece, para que sua propriedade possa prosperar.

Como o desenvolvimento web tem ganho destaque e tornando-se cada vez mais presente na rotina dos programadores, escolheu-se utilizar o framework Django por ser de fácil interpretação e de rápido desenvolvimento. Além disso, verificou-se que ele é muito seguro e eficaz, trazendo comodidade e satisfação, além de agilidade.

2 REFERENCIAL TEÓRICO

Neste item serão apresentados os conceitos necessários para o entendimento do projeto do sistema e como ele será construído. A linguagem de programação Python será apresentada, assim como o framework Django, que conta com melhorias para a linguagem.

2.1 CONTEXTO DO PRODUTOR RURAL

> Grandes transformações têm marcado a produção de leite brasileira nos últimos anos. Tais mudanças estão associadas, principalmente, aos impactos advindos da estabilização monetária, da desregulamentação do mercado (fim do controle estatal sobre os preços), da abertura econômica e da mudança nos padrões de consumo da população, que exigem dos produtores recorrentes adaptações no sentido de se modernizarem, buscando adequar-se à nova conjuntura e melhorar a competitividade. A despeito da queda nos preços ao produtor, a taxa média de crescimento da produção, na década dos 90, foi de 4% a.a., cifra alcançada, sobretudo, graças às mudanças tecnológicas que proporcionaram uma redução nos custos de produção. (GOMES, 2000)

Segundo Gomes, um dos maiores estados produtor de leite é Minas Gerais, que corresponde a 30% da produção nacional (IBGE, 2000), enquanto o Rio Grande do Sul produz cerca de 10% da produção brasileira.

Pode notar-se que de alguns anos para cá se produz mais com menos, ou seja, se uma determinada raça produzia 30 kg de leite por dia, hoje, geneticamente modificada, produz cerca de 15 a 20% mais, devido a cruza com outras raças que se adaptam mais ao ambiente em questão.

Todo este avanço se deve a uma boa gestão do rebanho, tanto com uso de softwares quanto ao uso de anotações, que de alguma forma auxiliam o produtor na hora de fazer algum procedimento ou simplesmente alterar a dieta do animal.

2.2 PYTHON

> Python é uma linguagem de programação de propósito geral, frequentemente aplicada em funções de script. Ela é comumente definida como uma linguagem de script orientada a objetos – uma definição que combina suporte para POO com orientação global voltada para funções de script. Na verdade, as pessoas frequentemente usam o termo “script”, em vez de “programa”, para descrever um arquivo de código Python. (LUTZ e ASCHER, 2007)

Python é uma linguagem de alto nível, ou seja, todo o código pode ser lido por nós com facilidade se distanciando da linguagem de máquina encontrada em todo o computador, que representa as sequências de instruções. Também é orientada a objetos, onde os objetos possuem dados próprios e são definidos para estruturar o programa.

"Python utiliza tipagem dinâmica, o que significa que o tipo de uma variável é inferido pelo interpretador em tempo de execução (isto é conhecido como Duck Typing)" (BORGES, 2010), ou seja, quando declarada as variáveis, não recebem um tipo específico, pois não são declaradas especificamente, sendo assim, os tipos variam de acordo com a implementação do sistema, porém sempre assumem um único tipo por vez.

De acordo com o site Computerwold[1], o nome Python foi originado do grupo humorístico *Monty Python*, criador do programa *Monty Python's Flying Circus*. A linguagem Python e seu interpretador estão disponíveis para diversas plataformas, como Linux, MacOS X e Windows. De acordo com o site da *Python Software Foundation*[2], Python foi lançada em 1991 por Guido van Rossum e é gerenciada pela *Python Software Foundation*, uma organização sem fins lucrativos.

O modelo de desenvolvimento da linguagem atualmente é comunitário, tendo como base a linguagem C e como prática de implementação é em Cpython, tornando o reaproveitamento da base de código em C/C++ existente em um ambiente, pois os dois sistemas continuarão a existir em C, porém podendo-se fazer uso deles em Python.

Python busca priorizar um código legível ao invés do desempenho. O esforço do programador tem maior enfoque que o esforço computacional, buscando reduzir o primeiro, mesmo que seja necessário aumentar o segundo. Ela é capaz de combinações de sintaxe com módulos e frameworks desenvolvidos por terceiros ou com os recursos de sua biblioteca padrão.

Um dos pontos fortes da linguagem é a qualidade de software. Foi desenvolvida para ter um visual mais agradável e ser de leitura fácil pois em vez de utilizar delimitadores, como chaves, usa espaços em branco e endentação, que é obrigatório, para separar os blocos de código, um novo bloco é identificado pelo aumento da endentação conforme podemos visualizar na Figura 1.

[1]The A-Z of Programming Languages: Python – fonte retirada do site http://www.computerworld.com.au/article/255835/az_programming_languages_python/?fp=4194304&fpid=1&pf=1

[2]Python Software Foundation – fonte retirada do site http://www.python.org/psf/

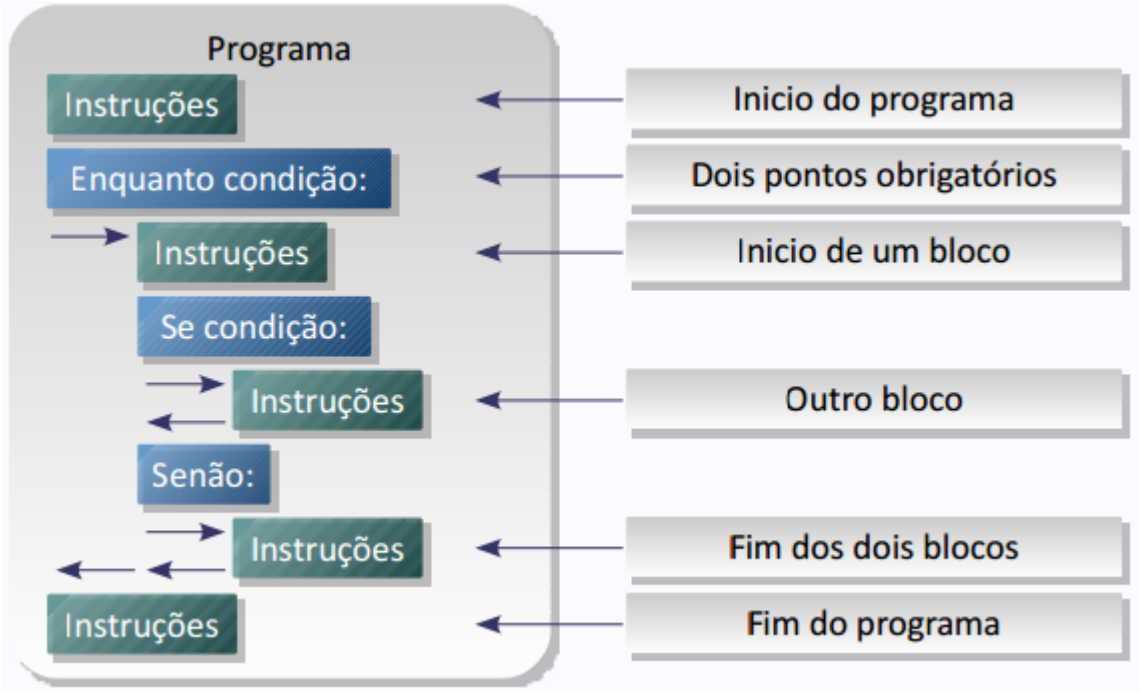

Figura 1: Exemplo da estrutura de código Python

São muito frequentes os erros durante a codificação, e ocorrem muito mais quando usado editores de texto comuns, que por sinal torna a programação mais lenta, pois não há auxilio algum sobre a linguagem, o que não ocorre quando usado uma IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado). A IDE "são pacotes de software que integram várias ferramentas de desenvolvimento em um ambiente consistente, com o objetivo de aumentar a produtividade do desenvolvedor" (BORGES,2010). Acompanhado de uma extensão PEP8 (*Python Style Guide Checker*), varre todas as linhas a procura de alguma infração ou código desnecessário.

Na maioria das versões do Linux e também no Mac OS, o interpretador Python já vem pré-instalado, no caso do Windows é necessário fazer o download pelo site oficial que atualmente se encontra na versão 3.5.1, com diversas funcionalidades muito diferentes da versão 2.7.5 que é a mais utilizada pelos desenvolvedores.

2.3 DJANGO

"O Django é um framework voltado para o desenvolvimento web, escrito em python, utilizando o padrão de projetos MVC" (FONSECA e BRAGA, 2009)

Django é um framework Python escrito com o intuito de não repetir códigos ao desenvolver um sistema e permitir o desenvolvimento rápido de projetos, devido ao fato de

algumas aplicações já terem sido previamente utilizadas. De acordo com o site oficial do Django[3], o nome Django foi inspirado no músico de jazz Django Reinhardt e foi criado no Lawrence Journal-World, para gerenciar o site jornalístico na cidade de Lawrence, no Kansas.

Este framework possui um padrão de três camadas: Models, Templates e Views. Os Models, são classes, modelos dos objetos que posteriormente serão guardados no banco de dados. A camada Templates, reúne as páginas HTML do projeto, contendo toda a parte visual da aplicação, e por último a View, onde as classes serão instanciadas, e criado toda a lógica que se faz necessário para dar funcionalidades as páginas que montarão as URL's, assim cada método será desenvolvido especificamente para cada página.

As principais características do Django são:

1. Mapeamento Objeto-Relacional (ORM): significa que toda a interface de comunicação com a tabela do banco de dados se dará através da classe, onde a modelagem será determinada pelos atributos, tornando banco abstrato ao programador, se só irá se preocupar com a lógica do sistema.
2. Interface Administrativa: o Django possui uma interface administrativa padrão que já vem embutido em sua instalação, que é montada com os objetos relacionados nas classes. Muito fácil para criar a interface, mas há dificuldades na hora de modificá-la pelo motivo de seus elementos serem padrões.
3. Formulários: além da interface administrativa, formulários podem também ser gerados diretamente dos objetos do banco de dados, incluindo validações básicas de cada campo.
4. URL's Elegantes: total liberdade na criação das URL's, pois o programador pode cria-las como ficar melhor, usando expressões regulares para inserir números, letras e até outras informações.
5. Internacionalização: basta apenas configurar o idioma da aplicação, para que ela passa a operar sobre este idioma, tornando muito mais fácil a troca.

2.4 MODEL

Classe que mapeia as tabelas do banco de dados e que contém campos e comportamentos essenciais aos dados.

Este arquivo será indispensável no sistema, pois ele contém toda a relação do banco de dados com o sistema, e também serve de base para a criação de formulários, pois todo o tratamento de campos se dá nos models.

[3]Django - https://www.djangoproject.com/ Acessado em: 20 de outubro de 2016

2.5 VIEW

As *views* contém toda a lógica de desenvolvimento do projeto, cada view contém métodos com o objetivo de realizar um comportamento para uma determinada URL.

Nesta parte do sistema é que tudo acontece, aqui são tratadas todas as regras de negócio, todos dos dados.

2.6 URL

Django usa expressões regulares configuradas no módulo urls.py para analisar as URL's das requisições e invocar a view apropriada para cada padrão de URL (RAMALHO, 2010).

Um arquivo de URL's bem formatado e limpo, garante uma boa legibilidade do código e um bom entendimento do mapa do sistema, que usando expressões regulares[4] conseguem deixar a URL com um aspecto bem mais amigável.

Portanto, o caminho das URL's contém letras e caracteres especiais, contendo parâmetros diferenciados ao longo do cominho que, podem ser opcionais ou não. O final de uma ULR é marcado pelos sinais ^ e $, respectivamente.

2.7 API

De acordo com o site Canaltech[5], API é um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na web. A sigla API refere-se ao termo em inglês "Application Programming Interface" que significa "Interface de Programação de Aplicativos".

Este recurso é criado quando uma empresa pretende que outros desenvolvedores desenvolvam produtos utilizando seus serviços. Há várias empresas que disponibilizam seus códigos e instruções para serem utilizados em outras áreas da maneira mais conveniente possível. Um exemplo básico são os sistemas de pagamento online, onde por meio de seu código original, muitos outros sites e aplicações utilizam seus dados adaptando da melhor forma a fim de utilizar o serviço.

Desta forma, há várias coisas que as APIs pode fazer, como:

[4] Expressões regulares são uma composição de símbolos, caracteres com funções especiais, que, agrupados entre si e com caracteres literais, formam uma sequência, uma expressão

[5] O que é API? Fonte retirada do site https://canaltech.com.br/o-que-e/software/o-que-e-api/ acessado em: 26 de outubro de 2016

- APIs entregam funcionalidades a websites;
- APIs estão por trás das aplicações/softwares online (SaaS);
- APIs suportam as aplicações móveis (Apps Mobile);
- APIs conectam objetos físicos, através dos conceitos extremamente interessantes de Internet das Coisas;
- Etc.

Conforme o site ProgrammableWeb, há cerca de 15 mil APIs públicas hoje em funcionamento, as quais não são consideradas as API restritas a usuários específicos e não abertas a toda internet.

2.8 DJANGO REST FRAMEWORK

Considerado uma extensão para o framework Django, traz uma série de funcionalidades prontas que facilitarão a implementação de APIs e a manipulação de arquivos no formato JSON. Do inglês (JavaScript Object Notation), ou seja, Notação de objetos JavaScript, é uma formatação de troca de dados de fácil leitura e interpretação, sua estrutura dá-se como uma coleção de pares nome/valor, tornando comum a todas as linguagens.

Uma de suas vantagens é a sua autenticação, utilizando o padrão OAuth 2.0, que permite que a aplicação não manipule diretamente nomes de usuário e senhas.

3 METODOLOGIA

3.1 ABORDAGEM DE PESQUISA

Este trabalho será do tipo de pesquisa qualitativa, pois segundo Godoy[6], este tipo de pesquisa se dá de modo que o pesquisador vai a campo, ou seja, vai em busca do fenômeno onde ele ocorre, considerando todos os pontos que se façam relevantes.
Será abordado o estudo de caso que, segundo Godoy, é uma análise profunda do objeto em estudo, neste caso um software legado[7], analisando quanto às melhorias a serem implementadas e suas novas funcionalidades.

3.2 CONTEXTO DA PESQUISA

A pesquisa se dará na empresa Chip Inside, proprietária do sistema a ser analisado, tendo como ferramenta de desenvolvimento o editor de texto Atom, disponibilizado pelo site <github.com>. Esse editor de texto é específico para programação, utiliza alguns plug-ins (extensões que modificam a caracterialidade do sistema, aumentando suas funções e implementações) para agilizar o desenvolvimento do software.

3.3 SISTEMA LEGADO

As empresas têm investido cada vez mais em tecnologia no planejamento, execução e avaliação de seus processos, esperando que obtenham um bom retorno prolongado de seus investimentos. Desta forma, tentam também prolongar a vida útil de seus sistemas, de forma que possam usufruir o máximo possível de tempo. Mas nem sempre a tecnologia que foi empregada no software consegue atender as demandas e necessidades tanto da empresa quanto dos clientes, sendo necessário assim, a substituição por um sistema mais moderno.

Com base no texto anterior, será abordado uma análise do software CManager, da empresa Chip Inside Tecnologia, sistema para monitoramento do rebanho utilizando dados da C-Tech, dispositivo que capta sinais vitais do animal como atividade e ruminação, e transforma em dados que se possa medir e fazer um parecer sobre o animal diagnosticando, por exemplo, se o animal está com alguma doença ou se encontra cio.

[6]GODOY, Arilda Schmidt, Pesquisa Qualitativa Tipos Fundamentais. RAE - Revista de Administração de Empresas, São Paulo, v. 35, n. 3, p. 20-29, 1995
[7]Programa de computador que já está em uso

O sistema é todo desenvolvido utilizando a linguagem Object Pascal da Borland, conhecida como Delphi em sua versão 7, uma poderosa linguagem orientada a objeto mas que possui ainda uma parte da antiga linguagem estruturada. Muito didática e de fácil aprendizado, utilizada ainda para implementar pequenos sistemas que não precisa de muitas regras de negócio. Apesar de ainda ser utilizada, muitos componentes não são mais atualizados nessa versão, tornando mais árduo o trabalho de melhoria do software. Como pode ser visto na Figura Figura 2, o sistema possui um design mais antigo, bordas quadriculadas e pouca customização.

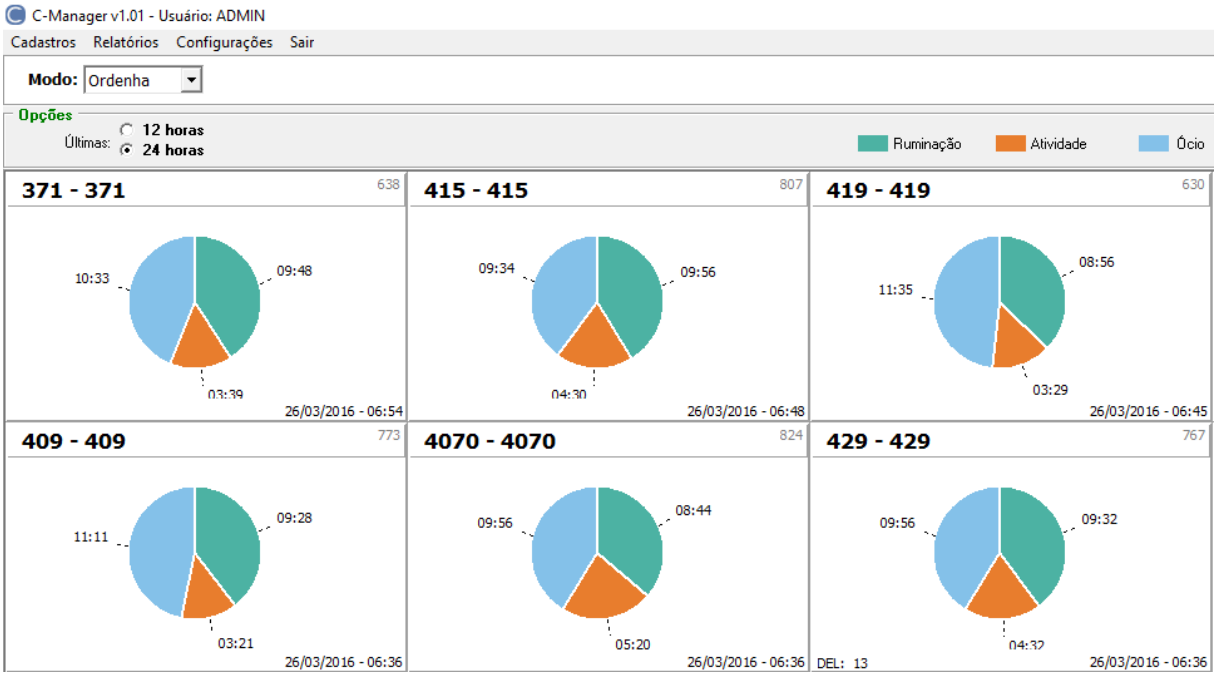

Figura 2: Design Sistema C-Manager

Além disto, as regras de negócio estão implementadas no mesmo local onde são tratados os eventos da interface gráfica, tornando a manutenção do código lenta e complicada. Pois, ao alterar qualquer instrução em um método, é necessário saber onde aquele método é utilizado para que as devidas alterações possam também ser feitas nestes lugares. Esse tipo de procedimento é complexo e suscetível a erros.

```
procedure TfrmVacaDetalhada.wwDBGrid1RowChanged(Sender: TObject);
  if sqlColeirasBRINCO.AsString <> '' then
  begin
    sqlAnimal.SQL.Text := 'select animal.nome, animal.data_ultimo_parto, '+
                          'animal.data_ultima_insem,'+
                          ' animal.data_nascimento'+
                          ' from animal'+
                          ' where animal.brinco = ' +
                          sqlColeirasBRINCO.AsString + '';

    sqlAnimal.Open;

    lbNomeVaca.Caption    := 'Vaca '+ sqlAnimalNOME.asString +'';
    if sqlAnimalDATA_ULTIMO_PARTO.AsDateTime > 0 then
    begin
      lbldel.Caption         := IntToStr(CalculaDEL(
                                sqlAnimalDATA_ULTIMO_PARTO.AsDateTime));
      lblultimoparto.Caption := sqlAnimalDATA_ULTIMO_PARTO.AsString;
    end
    else
    begin
      lbldel.Caption         := '';
      lblultimoparto.Caption := '';
    end;
    lblultimainse.Caption    := sqlAnimalDATA_ULTIMA_INSEM.AsString;
    lblnascimento.Caption    := sqlAnimalDATA_NASCIMENTO.AsString;
  end
  else
  begin
    lbldel.Caption         := '';
    lblultimoparto.Caption := '';
    lblultimainse.Caption  := '';
    lblnascimento.Caption  := '';
  end;
end;
```

Figura 3: Exemplo de código em ObjectPascal

Conforme apresentado na Figura Figura 3, pode-se notar que o trecho de código busca os dados do banco, seta os valores para seus respectivos lugares e ainda faz o tratamento desses dados na tela. Este tipo de programação não é indicada por padrões de projeto, pois dificulta muito o entendimento e também para realizar testes automatizados.

Com base nas premissas apresentadas, e para que seja retirado todo o proveito que o dispositivo pode oferecer, nada melhor que evoluir para outra linguagem de programação. Uma linguagem que pode tornar o software muito mais abrangente, sendo possível acessar os dados através de dispositivos mobile, utilizando um aplicativo que se comunique com a *API*, e exponha esses dados em forma de gráficos ou afins.

3.4 ESTRUTURA DO PROJETO

Neste item será abordado a estrutura utilizada, a simplificação do projeto e a forma de consumir ou acessar a API por meio de *web requests*.

Utilizando a linguagem de programação Django, acompanhado do pacote Rest, que adiciona novas funcionalidades pra o framework, como facilitar a implementação de APIs, tornei o software muito mais abrangente.

3.4.1 ESTRUTURA

Existem várias boas práticas de desenvolvimento e uma delas é dividir partes do código em diretórios, que na estrutura do framework são chamadas de *apps*, que tem a função de manter organizado as diversas áreas que um sistema pode ter.

Na Figura Figura 4, pode-se ver a linha de comando que cria este diretório:

```
~/Documentos/Projetos/api$ django-admin startapp core
```

Figura 4: Criação de Apps

Todos os *apps* criados pelo código apresentado na Figura Figura 4 terão os mesmos arquivos gerados. No entanto, cada conjunto de arquivos terá o seu *app* correspondente, mas com o desenvolvimento diferenciado de acordo com as necessidades do sistema.

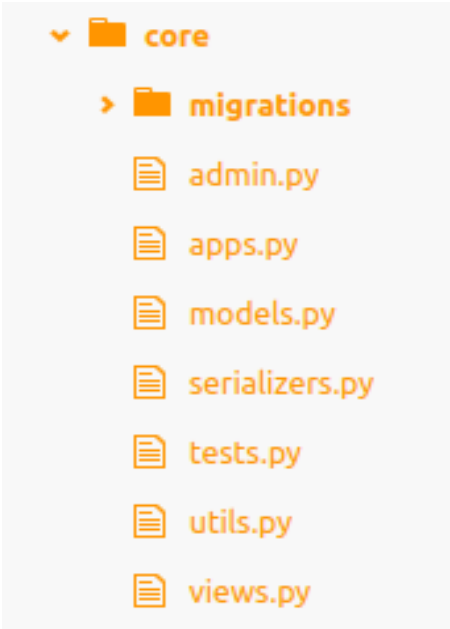

Figura 5: Estrutura do app core

A Figura Figura 6 apresenta a estrutura geral da API, incluindo o *app* core e os arquivos padrão do próprio Django.

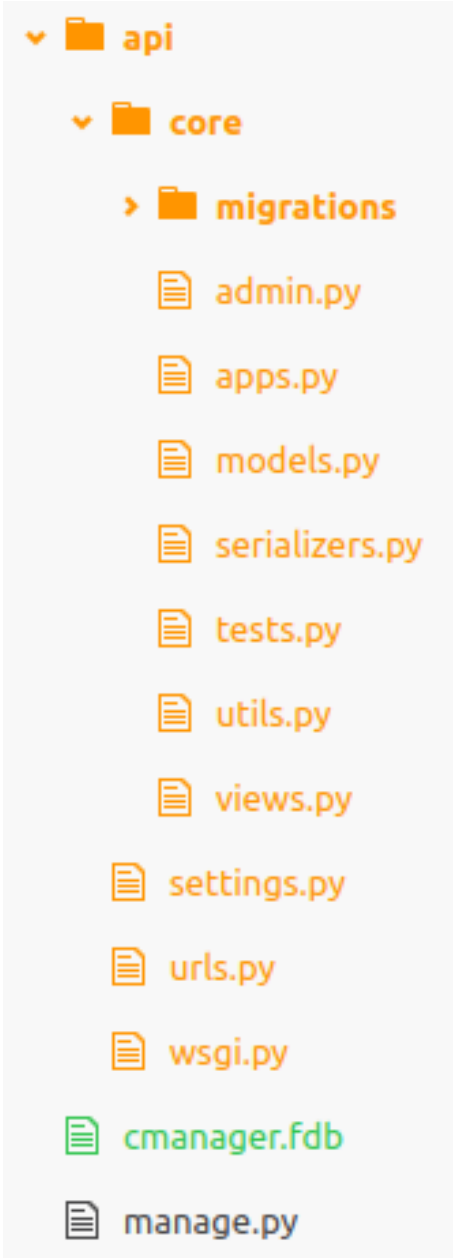

Figura 6: Estrutura geral da API

Dentro da pasta api, pode-se ver os arquivos *settings.py* e *urls.py.* O arquivo *settings.py* é responsável por armazenar todas as configurações para que a aplicação possa funcionar de forma correta. Dentro do arquivo há uma sessão[8] chamada *DATABASES,* que contém toda a configuração referente ao banco de dados, como usuário, senha, IP do servidor,

[8]Bloco de instruções ou código

porta http e o nome do banco de dados bem como o driver que fará a conexão da aplicação ao banco de dados como pode ser visto na Figura Figura 7.

```
DATABASES = {
    'default': {
    'ENGINE': 'firebird',
    'NAME': '/home/fabio/Documentos/Projetos/api/cmanager.fdb',
    'USER': 'SYSDBA',
    'PASSWORD': 'masterkey',
    'HOST': '127.0.0.1',
    'PORT': '3050',
    }
}
```

Figura 7: Exemplo de configuração do Banco de dados

Ao realizar o comando da Figura Figura 8, o sistema busca as informações da sessão DATABASES para poder inspecionar o banco de dados e gerar os modelos no arquivo *models,py* do *app* core.

```
~/Documentos/Projetos/api$ python manage.py inspectdb > models.py
```

Figura 8: Exemplo de criação de modelos via banco de dados

Esta é uma das vantagens do Django, poder criar os modelos através do banco de dados, tornando muito mais ágil o desenvolvimento.

Os modelos são a representação fiel do banco de dados, cada classe representa uma tabela do banco, cada linha da classe um campo, contendo todos seus atributos, conforme pode ser visto na Figura Figura 9.

```python
class Animal(models.Model):
    codigo = models.IntegerField(primary_key=True)
    nome = models.CharField(max_length=60)
    brinco = models.CharField(unique=True, max_length=20)
    registro = models.CharField(max_length=40, blank=True, null=True)
    data_nascimento = models.DateField(blank=True, null=True)
    data_cadastro = models.DateField(blank=True, null=True)
    cod_raca = models.ForeignKey('Raca', models.DO_NOTHING,
                                 db_column='cod_raca', blank=True, null=True)
    cod_comp_raca = models.ForeignKey('CompRaca', models.DO_NOTHING,
                                      db_column='cod_comp_raca', blank=True, null=True)
    cod_categoria = models.ForeignKey('Categoria', models.DO_NOTHING,
                                      db_column='cod_categoria')
    inativo = models.CharField(max_length=1, blank=True, null=True)
    observacao = models.CharField(max_length=1000, blank=True, null=True)
    cod_pai = models.ForeignKey('self', models.DO_NOTHING,
                                db_column='cod_pai', blank=True, null=True)
    cod_mae = models.ForeignKey('self', models.DO_NOTHING,
                                db_column='cod_mae', blank=True, null=True)
    cod_rebanho = models.ForeignKey('Rebanho', models.DO_NOTHING,
                                    db_column='cod_rebanho', blank=True, null=True)
    data_ultimo_parto = models.DateField(blank=True, null=True)
    data_ultima_insem = models.DateField(blank=True, null=True)
    vaca_prenha = models.CharField(max_length=1, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'animal'
```

Figura 9: Modelo de classe da tabela do banco de dados

Conforme pode ser visto, cada campo possui suas especificações, que são relevantes, como: o tipo de dados, tamanho máximo de caracteres, se pode ser nulo e para as chaves estrangeiras, a qual tabela o campo representa. Deste jeito, cada vez que necessitamos alterar algo no banco de dados, não é necessário fazê-lo diretamente, basta alterar no modelo e sincronizar os dados através das *migrations*, que não arquivos que contém todas as alterações que o banco sofreu ao longo do desenvolvimento.

Para gerar este arquivo das *migrations,* é necessário que o banco já esteja criado e funcionando perfeitamente. Após executa-se o comando da Figura Figura 10, no terminal e automaticamente o arquivo é gerado, contendo em seu nome, a data em que foi criado, para facilitar a procura, caso se faça necessário.

```
~/Documentos/Projetos/api$ python manage.py makemigrations
```

Figura 10: Comando de criação da migration

Este tipo de procedimento possibilita a fácil manutenção do sistema, pois todas as alterações que se façam necessário no banco de dados, estarão todas reunidas dentro de uma mesma pasta, bastando apenas executar o comando acima, que o próprio Django alterará a base de acordo com a ordem cronológica dos arquivos.

Outro arquivo que é de fundamental importância é o *urls.py*, responsável por criar as chamadas para as views, que são responsáveis pela lógica de desenvolvimento do projeto.

Neste arquivo, como pode ser visto na Figura Figura 11, temos no início a importação dos arquivos e bibliotecas necessárias, logo após, uma expressão regular que formará a *url* completa para cada ação. Note que, cada linha contida dentro da coleção *urlpatterns*, é uma chamada para uma ação ou para um outro arquivo de *urls* de outro *app*.

```
from django.conf.urls import url
from django.contrib import admin
from rest_framework.authtoken import views

import api.core.views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^api-token-auth/', views.obtain_auth_token),
]
```

Figura 11: Arquivo urls.py

Conforme dito no parágrafo anterior, as *views*, que são, responsáveis por buscar todos os dados do banco e organizar, de forma que, não há necessidade de fazer qualquer tratamento na hora de exibir os dados na tela.

Cada método dentro do arquivo é uma ação, que será referenciada no arquivo de *urls*, deixando assim, todo o desenvolvimento desacoplado, Veja um exemplo na Figura Figura 12.

```
@api_view(['GET'])
def list_backup(request, pk):
    #    Lista todos os backups conforme a chave
    try:
        backup = Backup.objects.get(fazenda_id=pk)
    except Backup.DoesNotExist:
        return Response(status=404)

    serializer = BackupsSerializer(backup)
    return JSONResponse(serializer.data)
```

Figura 12: Exemplo de um método da view

Este método possui a função de buscar todas as informações referentes ao backup do banco de dados, conforme o identificador da fazenda, após os dados são enviados para uma outra classe que serializará esses dados automaticamente, ou seja, formatará esses dados conforme o modelo do banco de dados, retornando em formato JSON, conforme o exemplo na Figura Figura 13.

```
class BackupsSerializer(serializers.ModelSerializer):

    class Meta:
        model = Backup
        fields = ('nome', 'data_criacao', 'tamanho')
```

Figura 13: Código da Classe BackupsSerializer

A classe da Figura Figura 13 tem somente a função de definir de qual modelo e de que campos será feito a serialização. Desta forma se torna muito mais ágil o processo, dando menor chance ao erro, já que a classe base que faz este processo já vem junto no pacote do Django Rest Framework.

4 RESULTADOS

Após elencar todas as especificações e demonstrar como a estrutura do projeto foi construída, é hora de apresentar os resultados.

4.1 REQUISIÇÕES

Toda API funciona por meio de requisições http que através da utilização dos verbos GET, POST, PUSH e DELETE, manipulam e retornam dados de acordo com cada implementação.

Tudo se inicia com a autenticação, uma das melhores vantagens de se utilizar o Django Rest Framework, pois já vem implementada bastando apenas indicar nas configurações o tipo de autenticação. Neste trabalho foi utilizado o tipo *token*, que consiste em enviar o usuário e senha obtendo como resposta uma sequência de 40 caracteres que sempre permanece a mesma para o usuário.

```
http POST 127.0.0.1:8000/api-token-auth/ username='teste'
password='teste123'
```

Figura 14: Exemplo de requisição de token

O resultado desta requisição é um JSON contendo algumas informações do servidor e o token, conforme a Figura Figura 15.

```
HTTP/1.0 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Date: Sun, 13 Nov 2016 19:11:51 GMT
Server: WSGIServer/0.2 CPython/3.5.1+
X-Frame-Options: SAMEORIGIN

{
    "token": "86d726f72813c06e8a6179503ae2b58e0602afcf"
}
```

Figura 15: Resultado da requisição de login

Após obter o *token* é necessário enviá-lo a cada requisição que se faça, conforme a Figura Figura 16.

```
http localhost:8000/criar_fazenda/
'Authorization: Token 0a38239ff14eb005f8ecfa2e9b0ee10f7ca5992b'
nome=Teste3 proprietario=ivan
```

Figura 16: Requisição para cadastro de fazenda

Desta forma, traz muito mais segurança e confiabilidade, pois ficar transitando dados confidenciais a todo momento nem sempre é válido, uma vez que alguém de má-fé pode interceptar essas informações e utilizá-las para invadir o servidor e roubar os dados.

O resultado da requisição da Figura Figura 16 resulta em um JSON contendo os mesmos dados enviados, com cabeçalhos da requisição, informações do servidor e o *status 201* de objeto criado, enfatizando que a requisição teve sucesso. Maiores detalhes podem ser observados na Figura Figura 17.

```
HTTP/1.0 201 Created
Allow: OPTIONS, POST
Content-Type: application/json
Date: Sun, 13 Nov 2016 21:09:23 GMT
Server: WSGIServer/0.2 CPython/3.5.1+
Vary: Accept
X-Frame-Options: SAMEORIGIN

{
    "nome": "AMF",
    "proprietario": "Fábio Ivan Borchardt"
}
```

Figura 17: Resultado da requisição

Os dados que são enviados são os campos elencados na classe *Serializer*. Desta forma, pode-se customizar da melhor maneira para que não transitem dados desnecessários evitando assim lentidão na comunicação. Esse é o padrão de retorno do *framework*, que poderá sofrer alterações caso não se encaixe ao projeto que ser quer desenvolver.

5 CONCLUSÃO

Uma API traz várias vantagens para qualquer organização, como por exemplo, conectividade com outros sistemas e integração de dados de outras aplicações. Com isso, os sistemas trazem dados mais precisos através da troca de informações entre eles.

Outro aspecto que se pode citar é o fato de levar uma tecnologia nova para o campo. Praticamente todos os grandes avanços nesta área são voltados ao maquinário agrícola e poucos direcionados para a gestão da fazenda e ao setor agropecuário, uma das grandes motivações de realizar este trabalho.

5.1 CONTRIBUIÇÕES

O caso aqui estudado trouxe novas possibilidades, como por exemplo, criar um aplicativo de celular para que o produtor possa ter acesso a alguns dados que são importantes para o gerenciamento do rebanho.

No quesito aprendizagem e conhecimento a linguagem de programação Python já é utilizada há muitos anos, mas somente em 2005 que o framework Django foi associado à linguagem, deixando-a apta para desenvolvimento web.

Além de possuir grande aceitação no mercado de trabalho por causa da agilidade na implementação de diversos tipos de sistemas, o *framework* Django também pode trazer vários benefícios ao ambiente acadêmico, principalmente pela facilidade no desenvolvimento, pela fácil depuração do código e manipulação de elementos do sistema.

O *framework* Django cresce cada vez mais e é atualizado frequentemente, trazendo mais elementos e facilitando o desenvolvimento. Além disso, diversas extensões são criadas por desenvolvedores, a fim de facilitar algumas rotinas tornando a reutilização de código mais habitual. Algumas delas foram utilizadas ao longo do desenvolvimento deste projeto.

5.2 LIMITAÇÕES DO ESTUDO

Como o projeto foi desenvolvido sob uma linguagem que não é apresentada ao curso de Sistemas de Informação, a maior dificuldade foi aprender a linguagem, entender como funciona uma API, e de que forma o Django trabalha com os dados.

Este trabalho é um protótipo e existem muitas coisas a serem melhoradas, como por exemplo, o desempenho. No entanto, requerem um estudo mais profundo sobre o framework, visando identificar as melhorias que se façam necessárias.

REFERÊNCIAS

GOMES, S. T. **Economia da Produção do Leite**. Belo Horizonte: Itambé, 2000.

GODOY, Arilda Schmidt **Pesquisa Qualitativa:** Tipos Fundamentais. RAE, São Paulo, v. 35, n. 3, p. 20-29, 1995.

FONSECA, V. P. da; BRAGA, F. M. **Django,** Desenvolvimento Ágil para a Web. Tocantins, 2009.

LUTZ, Mark; ASCHER, David. **Aprendendo Python**. Ed. Bookman Companhia, 2007.

BORGES, Luiz E. **Python para Desenvolvedores**. Ed. Novatec, 2014.

RAMALHO, Luciano. **Django 101 Release 1.** 2010.

PELOI, Ricardo. **O que são APIs.** Disponível em: http://sensedia.com/blog/apis/o-que-sao-apis-parte-1-introducao/. Acessado em: 01/10/2016.