UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

FACULDADES LUIZ MENEGHEL

ESTELA REDONDO

DESENVOLVIMENTO DE UM PROTÓTIPO DE WEBSITE FILANTRÓPICO UTILIZANDO PADRÕES DE DESENVOLVIMENTO NA FERRAMENTA CAKEPHP.

Bandeirantes

2009

ESTELA REDONDO

DESENVOLVIMENTO DE UM PROTÓTIPO DE WEBSITE FILANTRÓPICO UTILIZANDO PADRÕES DE DESENVOLVIMENTO NA FERRAMENTA CAKEPHP.

Trabalho de Conclusão de Curso submetido à Faculdade Luiz Meneghel da Universidade Estadual do Norte do Paraná, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Ms. Ricardo Gonçalves Coelho

Bandeirantes

2009

ESTELA REDONDO

DESENVOLVIMENTO DE UM PROTÓTIPO DE WEBSITE FILANTRÓPICO UTILIZANDO PADRÕES DE DESENVOLVIMENTO NA FERRAMENTA CAKEPHP.

Trabalho de Conclusão de Curso submetido à Faculdade Luiz Meneghel da Universidade Estadual do Norte do Paraná, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____

Prof. Ms. Ricardo Gonçalves Coelho
Orientador

_____

Prof. Ms. André Luís Andrade Menolli
Membro da banca examinadora

_____

Prof. Ms. Glauco Carlos Silva
Membro da banca examinadora

Bandeirantes, __ de_____ de 2009

AGRADECIMENTOS

Agradeço primeiramente a Deus, por sempre estar comigo tanto nos momentos bons quanto nos ruins e me ajudar a concluir mais uma etapa dos meus objetivos.

A meu orientador Prof. Msc. Ricardo, por todo o apoio e paciência em lidar com meus ataques estéricos (“pitis”), pelas conversas e acima de tudo pela amizade construída durante todos esses anos. Aos membros de minha banca Prof. Msc. André Luís Andrade Menolli e Prof. Msc. Glauco Carlos Silva por sempre estarem dispostos a me ajudar compreender qualquer tipo de dúvida; e a todos os demais professores que participaram significativamente da minha formação.

A meus pais, Sérgio e Telma, por todo incentivo dado até hoje, toda renuncia que fizeram pelo meu bem. Nem todas as palavras conseguiriam expressar o quanto sou grata por tudo o que fizeram e fazem por mim, me orgulho muito de tê-los como pais. Amo vocês demais!

A meu irmão Guilherme, por todo carinho e ternura demonstrado até hoje.

A todos os meus amigos e colegas que estiveram presentes em minha vida, me proporcionando momentos inesquecíveis. As minhas amigas de república (Iassudara, Priscila, Daniele, Betânia e Miguel). A Paula e claro, a Yeda que conviveu comigo durante três anos e compartilhou das minhas alegrias e tristezas. Em especial a minha irmã adotiva Renata, que sempre esteve presente em todos os momentos de minha vida desde que nos conhecemos. São muitas as pessoas, cada uma sabe da importância que têm em minha vida, embora seus nomes não sejam citados aqui, mas cada um tem um lugar especial reservado.

Obrigada a cada um de vocês que fizeram e fazem parte da minha vida, certamente jamais serão esquecidos.

RESUMO

Com o crescente avanço tecnológico ocorrido até então, a utilização da Internet se torna indispensável e com isso, ambientes capazes de vincular o relacionamento entre pessoas e animais vêm sendo criados, porém alguns ainda apresentam deficiências que influenciam em sua aceitação. Dessa forma, neste trabalho foi proposto o desenvolvimento de um protótipo de um *website* filantrópico que unifique o relacionamento entre pessoas e animais abandonados, destinado a atender os requisitos básicos que estes animais necessitam para sobreviver e possibilitar aos colaboradores uma resposta satisfatória à ajuda oferecida. Assim foi selecionado para o desenvolvimento da aplicação o *framework CakePhp* ,que emprega o padrão de arquitetura conhecido como MVC(Model View Controller) e o padrão de projeto *ActiveRecord*, permitindo ao desenvolvedor construir aplicações de maneira ágil e organizada. Com a utilização do *framework* foi constatado que o mesmo possui fácil configuração, disponibiliza um gerador de código para a interação com o BD(Banco de Dados) e faz a interação com as associações existentes entre as tabelas, o que possibilitou uma redução no tempo de desenvolvimento do protótipo. Além disso, permitiu a criação de visões de maneira muito mais fácil e com menos linhas de código, deixando evidente que sua finalidade é simplificar o processo de desenvolvimento através de recursos disponibilizados pelo mesmo.

Palavras-chave: Website filantrópico, Framework *Cakephp*, Benefícios.

ABSTRACT

With the increase of the technological advances occurring so far, the Internet has become indispensable and with this the environments capable of binding the relationship between people and animals are being created, but some of them still have some deficiencies that influence on its acceptance. Thus, this work was proposed to develop a prototype of a philanthropic website that unifies the relationship between people and abandoned animals, to meet the basic requirements that the animals need to survive and enable collaborators to a satisfactory response to the aid offered. It was selected for the development of the framework the CakePhp application, which uses patterns of projects known as MVC and ActiveRecord, allowing the developer to build applications flexibly and organized. This framework was chosen because in addition to having easy setup, it yet provides a generator of code for interaction with DB and makes the interaction with the associations among the tables, which allowed a reduction in development time of the prototype. It also allowed the creation of visions with more facility and with fewer lines of code, making clear that its purpose is to simplify the development process through the resources provided by the framework.

Key-words: Philanthropic website, CakePhp Framework, benefits

LISTA DE FIGURAS

LISTA DE SIGLAS

MVC – *Model View Controller*

BD – Banco de Dados

CRUD – *Create Retrieve Update and Delete*

HTML – *HyperText Markup Language*

XML – *Extensible Markup Language*

PDF – *Portable Document Format*

LDAP – *Lightweight Directory Acess Protocol*

Feeds RSS – *Really Simple Syndication*

App – Application

UML – *Unified Model Language*

HTTP – *HyperText Transfer Protocol*

PHP – *HyperText Preprocessor*

SQL – *Structured Query Language*

MER – Modelo Entidade Relacionamento

CSV – *Comma Separted Value*

ONG – Organização Não Governamental

SUMÁRIO

1 INTRODUÇÃO

Com a evolução tecnológica crescendo gradativamente, a busca por serviços dispostos na Internet se torna cada dia mais indispensável na vida das pessoas, tanto pela comodidade que essa tecnologia traz, quanto pela falta de tempo que as pessoas possuem, muitas vezes causadas por essa mesma evolução, uma vez que essa exige a superação das próprias expectativas no mercado de trabalho por parte dos indivíduos ligados a ela.

Essas modificações causadas por essa constante evolução da tecnologia nos obrigam a fazer alterações importantes em nossas vidas para que possamos acompanhar o desenvolvimento e as modificações trazidas por tal evolução, principalmente no que diz respeito ao mercado de trabalho.

Isso se reflete também à vida de cada individuo, pois estes acabam optando por preservarem maneiras estratégicas de garantirem a praticidade até em suas casas. Com isso a infra-estrutura em seus lares se torna adaptativa a pouca exigência de atenção e de tempo redobrado.

Com base nesse conceito, a cada dia que passa é mais comum nos depararmos além de outras coisas, com animais abandonados nas ruas sem nenhuma condição saudável de vida. Tal fato se reflete a população que se vê obrigada a conviver com tal atrocidade sem poder fazer muito para ajudar esses animais.

Pensando nesse princípio, a tecnologia vem empregando a idéia de criar ambientes que se assemelhem à vida real, e possibilitem a esses animais uma melhor condição de vida, fazendo com que estes tenham algum tipo de vínculo com pessoas. Alguns sites vêm sendo construídos para atender essa necessidade, porém, muitos deles possuem algumas deficiências, como por exemplo, a falta de respostas às ajudas que os cooperadores dão a estes animais; o que dificulta para estas pessoas saberem se realmente a ajuda que prestam estão ou não sendo de grande valia e, isso muitas vezes podem desacelerar o processo de aceitação por parte dos colaboradores em relação a este tipo de metodologia empregada.

Surge então a necessidade de se desenvolver novas aplicações confiáveis, que inibam as deficiências apresentadas por essas aplicações antigas,

sendo então possível o uso do *framework CakePHP;* O *CakePhp* foi escolhido porque permite uma instalação simples utilizando Servidor Web e Banco de Dados.

Segundo Golding (2007), o *CakePhp* é eficaz na utilização do padrão MVC (*Model, View, Controller*) e ocupa menos espaço no servidor, permite também o desenvolvimento de aplicações de forma ágil, os desenvolvedores escrevem poucas linhas de código, além de ser um dos mais completos *frameworks* existentes.

1.1 Objetivos

O objetivo geral deste projeto é o desenvolvimento de um protótipo de um *website* que unifique um ambiente de interação entre pessoas e animais, enfatizando a importância da colaboração de cada pessoa em beneficio da melhor qualidade de vida destes animais, bem como a confiança que a instituição irá passar para futuros colaboradores.

Dentre os objetivos específicos do projeto de pesquisa podem-se citar:

1. A oportunidade de propor soluções viáveis para o problema encontrado nas entidades analisadas que não possuem a metodologia de assegurar a veracidade dos seus propósitos.
2. Disponibilizar conteúdos inerentes aos animais presentes na instituição de acordo com os interesses e necessidades sugeridos pelos padrinhos.
3. Fazer com que esse *website* seja um canal de comunicação viável e eficaz para a relação entre animais e pessoas.
4. E por último contribuir com a sociedade de desenvolvedores, expondo as dificuldades, limitações e benefícios apresentados pelo *framework CakePhp* no decorrer da implementação do protótipo.

1.2 Justificativas

A motivação para o desenvolvimento deste trabalho se deu pela crescente procura por ferramentas que possibilitem a facilidade de compreensão e manutenção das aplicações criadas. Sendo esse trabalho desenvolvido seguindo o padrão de arquitetura MVC, traz também a possibilidade de futuramente ser de grande valia a quem se dispuser a fazer melhorias ou alterações no mesmo.

A escolha do *framework CakePhp* ocorreu pelo mesmo ser um *framework open source,* e se tornar referência na questão do desenvolvimento, pois além de possuir facilidade no aprendizado e proporcionar o desenvolvimento das aplicações de maneira rápida e eficaz (Manual do CakePhp, 2008), o *CakePhp* mantém a vantagem de não possuir complicações referentes ao servidor, já que roda em qualquer servidor que possua suporte ao PHP, tem suporte aos principais BD e possui praticamente os mesmo recursos que os outros frameworks oferecem.

O projeto desenvolvido busca contribuir com a sociedade tecnológica sobre resultados concretos em relação aos benefícios ocasionados pela utilização de tal *framework*.

Esse trabalho traz importância de cunho social, pois trata de um assunto que contribui na ajuda a animais abandonados, que muitas vezes se encontram em estados deploráveis por descuido da população a terem condições mínimas de bem estar.

Estudos feitos com base na análise de requisitos dos sites atuais constataram que estes não passam confiabilidade às pessoas que desejam contribuir com a instituição. Dessa forma, nesse trabalho pudemos aperfeiçoar as falhas apresentadas por estes outros sites e proporcionar uma maior credibilidade nesse novo exemplar.

1.3 Organização do trabalho

A estrutura do trabalho se apresenta distribuída da seguinte maneira: no capítulo 2 são apresentados os conceitos relacionados à arquitetura MVC e sobre o *CakePhp*. No capítulo 3 está a descrição da metodologia adotada no presente projeto. Já no capítulo 4 são apresentadas os diagramas de UML responsáveis pelo melhor entendimento da estrutura e funcionamento do protótipo.

E por fim no capítulo 5 são apresentados as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos relacionados à área de pesquisa do trabalho: conceitos da arquitetura MVC e do *framework CAKEPHP* .

2.1 MVC

MVC de programação é a aplicação de três camadas, também chamada de tríade, em que os objetos de diferentes classes retomam as operações relacionadas com o domínio da aplicação (modelo), a exibição do pedido do estado (visão), e a interação do usuário com o *model* e a *view* (controlador).

Para poder usar o paradigma MVC, sobretudo é necessário compreender a divisão do trabalho dentro da tríade, além de compreender também como as três partes dessa tríade comunicam-se uma com as outras (Burbeck**,**1992).

Segundo (Krasner, Pope, 1998), o objetivo da divisão de uma tríade implica em separar a parte que representa o modelo subjacente do domínio da aplicação (Model), da forma como o *model* é apresentado para o usuário (Controller), e da forma como o usuário interage com ela (View). Isolando-se essas unidades funcionais, torna-se mais fácil para o *designer* de aplicação compreender e modificar cada unidade especial sem ter que saber tudo sobre as outras unidades.

Na Figura 1 é ilustrada uma mesma aplicação sendo acessada e visualizada por várias interfaces, ou seja, a interface de um celular, a de um *palmtop*, ou de qualquer outro objeto podem ter acesso à mesma aplicação, sem que esta saiba quantas ou quais interfaces estão acessando-a.

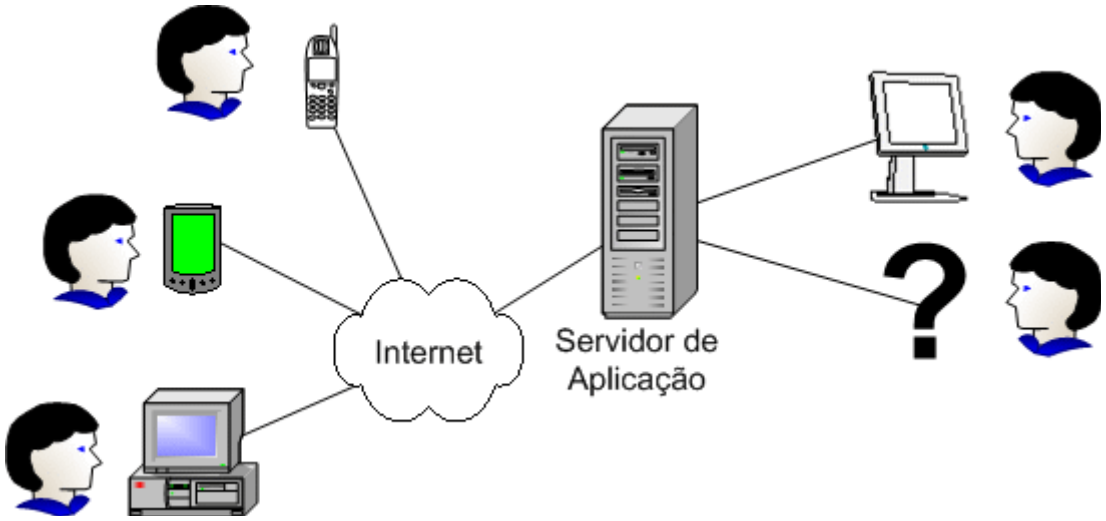

Figura 1: Aplicação sendo acessada por várias interfaces – Fonte: Almeida (2003).

2.1.1 Características

Segundo (Pereira; Zannata, 2004) as principais características do modelo MVC são:

1. Um dos primeiros padrões reconhecidos.
2. Foi utilizado inicialmente em Smalltalk-80.
3. Separação entre dados, apresentação e controlador, que gerencia as relações entre o modelo e a apresentação.
4. Separação entre a lógica e a apresentação.
5. Possui a vantagem de reutilização de código.
6. As responsabilidades são bem definidas.
7. Redução do esforço na camada de apresentação.
8. Padrão de projeto (projeto de objetos com responsabilidades bem distribuídas) que relaciona a interface do usuário e seus dados.

2.1.2 Camadas no MVC

1. Model

Para (Macoratti,2004) a camada de lógica de aplicação é o coração da aplicação, pois é ele que representa os dados e a regras de negócio que administram o acesso e alterações dos dados. O *model* permite ao *controller* a capacidade de acessar as diversas funcionalidades da aplicação encapsuladas pelo próprio *model* e mantém o estado pertinente do negócio, respondendo a pedidos de informação ou instruções sobre seu estado.

O *model* não possui conhecimento especifico sobre seus *controllers* e *views*, de modo que o próprio sistema é quem mantém a conexão entre esses *controllers* e *views* e quem gerencia a apresentação do seu estado pela *view*.

É essa camada responsável por tudo o que a aplicação irá fazer:

- Modela os dados e o comportamento por trás do processo de negócios;
- É encarregada apenas do armazenamento, manipulação e geração de dados;
- É um encapsulamento de dados e de comportamentos que independem da camada de apresentação.

2. View

Trata-se da interface da aplicação, através do qual o usuário interage com o Sistema. É também quem acessa os dados do modelo através de um *controller* e define como esses dados serão apresentados, ou seja, ela é quem é encarregada pelo mapeamento gráfico a um dispositivo. Não se preocupa com a forma onde a informação foi obtida e nem como, apenas exibe a informação.

Esta camada é responsável por:

- Incluir o elemento de exibição no cliente: HTML, XML, PDF, etc.
- É a camada de interface com o usuário.
- É usada para receber a entrada de dados e apresentar o resultado.

3. Controller

Segundo (Marston,2007) *controller* é onde são processadas todas as requisições feitas através da interface, ele é o meio pelo qual o usuário interage

com a aplicação, pois é quem interpreta o mouse e teclado do usuário e instrui o *model* a responder a tais ações relacionadas ao que foi inserido; Geralmente existe um *controller* para cada conjunto de funcionalidades relacionadas;

Para reforçar (Macorrati,2004) salienta que é o *controller* quem decide o fluxo da apresentação, servindo como uma camada intermediária entre a camada de apresentação e a lógica. Sendo assim, o *controller* verifica a ação do usuário e o resultado do processamento do *model* e seleciona uma visualização como parte da resposta à solicitação do usuário.

Esta camada é responsável por:

- Mapear e controlar as ações entre o *model* e a *view*;

2.1.3 Vantagens

De acordo com (Pereira; Zannata, 2004) as principais vantagens da utilização da arquitetura MVC é que:

1. Possui grande facilidade em manter, testar e atualizar sistemas múltiplos, já que administra múltiplos visualizadores utilizando um mesmo modelo**.**
2. É simples incluir novos clientes sem haver necessidade de modificar a aplicação inteira, apenas incluindo visualizadores e controles referentes a ele;
3. Por possuir modelo, visualizador e controle independentes, é possível o desenvolvimento em paralelo para estes.
4. Torna a aplicação escalável, ou seja, múltiplas visões para o mesmo modelo.
5. Permite integração de equipes e/ou divisão de tarefas.

O grande destaque da arquitetura MVC é a possibilidade de existir independência entre as camadas, o que facilita e muito a vida dos desenvolvedores de aplicações como mencionado até então, porém, ainda segundo (Pereira; Zannata, 2004), é importante lembrar que não é aconselhável à utilização para

pequenas aplicações, pois este tipo de modelo exige do pessoal relacionado ao desenvolvimento uma quantidade maior de tempo para poder analisar e modelar os sistemas, sem contar que é imprescindível que estes sejam especializados.

Como as vantagens se sobressaem em relação às restrições encontradas, muitas aplicações estão sendo desenvolvidas com *frameworks* que fazem uso desse modelo, tais como: Apache Struts, JSF (Java Server Faces), Catalyst, Django, XPT *framework*, entre vários outros incluindo o *CakePHP* que foi escolhido para ser utilizado nesse trabalho.

2.2 *CakePhp*

O *framework CakePHP* utiliza alguns tipos de padrões de desenvolvimento, dentre eles os conhecidos: *ActiveRecord* que é encarregado de fazer o mapeamento de ORM (objeto relacional), uma vez que a maioria dos BD (Bancos de Dados) é relacional e os *softwares* são orientados a objetos, então se faz necessário esse mapeamento para que os dois paradigmas sejam compatíveis entre si, em resumo, transforma a tabela relacional em classes de objetos e manipula os dados/objetos, como mencionado por Maseto (2006) *apud* Pulido (2006);e o MVC, como já mencionado anteriormente, que disponibiliza uma fácil manutenção da aplicação.

A construção é feita de forma independente, o que a torna também mais leve. Este padrão divide a aplicação em pacotes modulares que facilitam e agilizam o desenvolvimento; novas funcionalidades podem ser embutidas rapidamente, criando novas características as já existentes e permitindo aos desenvolvedores trabalharem simultaneamente na construção de novos protótipos e, de alterarem partes destes sem a interferência em outras partes.

Apesar de muitos *frameworks* fazerem uso do padrão MVC percebe-se que *CakePHP* tem mais eficiência na utilização desse padrão e ocupa menos espaço em seu servidor do que em relação, por exemplo o *Symfony* ou o Zend (*Golding*, 2007).

A Figura 2 exemplifica um modelo simples de requisição MVC em *CakePHP*. Vale lembrar que basicamente todas as aplicações serão baseadas nesse exemplo.

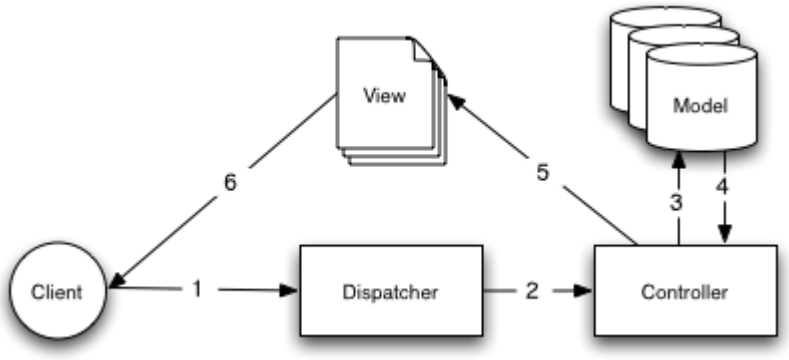

Figura 2: Requisição básica em MVC - Fonte: Manual do CakePhp(2008).

Com o propósito de ilustrar o processo do MVC digamos que um usuário qualquer, apenas clicou em um link de sua aplicação que gera a ação comprar.

1. O usuário clica no link apontando para a página que é responsável por àquela ação e seu *browser* faz uma requisição ao site;

2. O *dispatcher* (expedidor) verifica qual a URL requisitada e então encaminha ao *controller* correto;

3. O *controller* efetua então a lógica específica da aplicação. Por exemplo, verifica se o usuário está ou não logado;

4. O *controller* usa os *models* para poder acessar os dados da sua aplicação. Na maioria das vezes os *models* representam as tabelas existentes no banco de dados, e podem, por exemplo, no caso de um sistema de compras, trazer ao usuário as últimas compras do banco; mas podem também representar registros LDAP, *feeds* de RSS ou até mesmo arquivos do sistema.

5. Após ter executado a “mágica” sobre os dados, o *controller* é encarregado de repassar para a *view*. A *view* faz com que os dados sejam representados ao usuário da melhor forma possível. Normalmente no *CakePHP* essas *views* vêm no formato HTML, mas se for necessário podem ser facilmente exibidas em PDF, documento XML, ou outro formato qualquer.

6. E por último a *view* retorna o conteúdo ao *browser* do usuário (conteúdo esse proveniente dos dados emitidos pelo *controller* para a construção da página e conseqüentemente exibição para o cliente) (Manual do *CakePHP*,2008).

2.2.1 Vantagens

Em relação à estrutura do *CakePHP*, o (Manual do CakePHP,2008) ainda descreve que além das classes já conhecidas, MVC, ainda possui classes e objetos adicionais que acrescentam características intrínsecas a esse *framework*. Os *Components*, *Behaviors* e *Helpers*, por exemplo, são classes que proporcionam extensibilidade e reuso para adicionar funcionalidades à base MVC de maneira rápida.

A classe *Component* (Componentes) é a extensão do *Controller* e, que também auxilia na lógica deste. Se há a existência de uma lógica e o desejo de compartilhá-la entre os *controllers* (ou aplicações) esta classe é então utilizada. Com essa classe, ao invés de se escrever um método em cada controlador que utilize essa mesma lógica, empacota-se essa funcionalidade e compartilha seu método entre os controles através da criação de um componente.

O c*ontroller* também possui *callbacks*, que são utilizados caso haja a necessidade de se inserir uma lógica entre operações do núcleo do *CakePHP*.

Nele está incluso:

- beforeFilter (): esse comando é executado antes que o *controller* execute qualquer uma de suas ações, é de grande valia quando se deseja conferir permissões ou sessões ativas.
- beforeRender ():esse *callback* é pouco utilizado; é executado antes da *view* ser renderizada, porém depois da lógica do *controller*,
- afterFilter (): executado tanto depois que o *controller* tiver executado todas suas lógicas quanto a *view* sua renderização. Não há diferença entre afterRender () e afterFilter (), somente se houver no método do *controller* que faça a chamada manualmente para render () e exista alguma lógica depois dessa chamada.

A classe *Helper* (Ajudante) é a extensão da *View* e auxilia na lógica de visão. Assim como o *Component*, o *Helper* permite que a apresentação lógica seja acessada e compartilhada entre as visões existentes. O *CakePHP*

disponibiliza o reuso de código na visão (*view*) com a utilização de layout e *elements* (elementos), uma vez que a maioria das aplicações tem partes de códigos utilizados diversas vezes nas visões.

O *Behaviors* também adicionam funcionalidades, porém entre os modelos, dos quais são extensões. Aqui se tem a vantagem de poder especificar comportamentos de determinadas ações, e, por exemplo, ganhar funcionalidades para remover, alterar e adicionar dados.

Os modelos têm ainda como auxilio para a manipulação de dados os chamados *DataSource*, que são abstrações que permitem aos modelos manipularem diferentes tipos de dados. De fato no *CakePHP* a maior fonte de dados numa aplicação é via BD, entretanto,existe a possibilidade de escrever DataSources adicionais que possibilitam ao modelo representar um *feed* RSS(*Really Simple Syndication*)[1], arquivos CSV(*Comma Separted Value*)[2], entidades LDAP(*Lightweigth Directory Access Protocol*)[3],etc., além de permitir a associação de registros de diferentes fontes.

Aqui também se encontra:

- beforeFind (*mixed* $*queryData*): Como o próprio nome sugere é utilizado antes de qualquer operação de busca. É usada a variável $queryData que leva consigo informações sobre a busca atual, como: condições (*conditions*), campos (*fields*), etc.
- afterFind(*array* $*results*): Utilizado para fazer alterações nos resultados de uma busca realizada ou adotar decisões baseada nos resultados. É no parâmetro $*results* que serão retornadas informações da busca.
- beforeValidate (): Esse *callback* é usado quando se deseja alterar os dados antes de serem validados. Caso haja necessidade de se adicionar mais alguma coisa ou para validações mais complexas

---

[1] *Feed* RSS são listas de atualizações de conteúdo de um determinado site, escritos com especificações baseadas em XML.
[2] CSV é o formato de arquivo usado para troca de dados entre aplicações diferentes.
[3] LDAP é um protocolo de acesso à banco de dados especializados chamados de diretórios, usados na Internet e baseados no modelo cliente-servidor.

esta função também pode ser usada.

- beforeSave (): Caso seja necessário o manuseio de dados antes destes serem armazenados, como por exemplo, alguma formatação especial que o BD precise e que seja importante antes do armazenamento, usa-se essa função.

2.2.1.2 Diretório Application (App)

A pasta App é onde se encontra armazenada a aplicação desenvolvida pelo programador, separadas por subpastas:

- Config: Onde ficam armazenados todo tipo de arquivos de configuração, detalhes de conexão com o BD *e bootstrapping*.
- Controller: Onde estão armazenados os *controllers* de sua aplicação e seus componentes.
- Locale: Strings para internacionalização são guardadas em arquivos existentes nessa pasta.
- Model: É onde são armazenados os Modelos, *Datasources* e *Behaviors* da aplicação.
- Plugins: Armazena os diversos plugins que possam vir a existir, estes plugins são pacotes de *model, view* e *controller* que desempenham objetivos específicos que pode envolver diversos aplicativos.
- TMP: Onde serão armazenados arquivos temporários. Usada para armazenar certos tipos de informações, como por exemplo, descrição dos modelos.
- Vendors: Armazena bibliotecas externas, usadas tanto para *controllers* ou *views.*
- Views: Onde são guardados os arquivos de apresentação: *elements*, páginas de erros, *helpers* que têm as funções para formatar e apresentar os dados de forma útil, layout e arquivos de

visão.

- Webroot: Aqui são guardados arquivos públicos, como por exemplo, imagens e arquivos de *JavaScript*.

Estas subpastas que dividem a aplicação geram uma melhor disposição da mesma, possibilitando que tarefas, tais como manutenção, ocorra de maneira organizada.

2.2.2 Padrões de Desenvolvimento

CakePhp faz uso de alguns padrões de desenvolvimento em sua estrutura, e, dentre eles está os conhecidos MVC (que por já termos comentado anteriormente não iremos reescrever aqui), o *ActiveRecord*, o FrontController e o *Access Data Mapping.*

O *Active Record* conecta os objetos da regra de negócio e as tabelas do BD para criar a persistência dos dados no domínio do modelo, onde a lógica e os dados são apresentados de maneira encapsulada. De maneira mais simples, como aponta (Fowler, 2004), um objeto que envolve uma fila em uma tabela do BD ou view encapapsula o acesso a base de dados e acrescenta lógica de domínio naqueles dados, então o *Active Record* coloca a lógica de acesso a dados no domínio do objeto (que se encontra no *Model*), dessa forma todas as pessoas saberão ler e escrever seus dados (de/e) para o BD.

O *Active Record* geralmente tem métodos que fazem:

- Construção de uma instância do *Active Record* para setar uma fila de resultados SQL.
- Construção de uma nova instância para inserções posteriores na tabela.
- Métodos comumente de encapsulamento usados em *queries* SQL e retorno de objetos *Active Records*.
- Atualizam o BD e inserem os dados dentro do *Active Record*.

- Obtém e definem os campos.
- Implementam alguns pedaços de lógica de negócio.

Para (Fowler, 2004), *Active Record* é uma boa escolha para o domínio da lógica que não é tão complexa, tais como as funções estabelecidas pelo CRUD.

O *Front Controller* para (Lopes, 2003) é responsável por gerenciar o tratamento de uma solicitação (vinda de uma visão). Dessa forma ele permite a centralização de requisições e cria uma interface genérica para o processamento de comandos (solicitações). No *Cakephp* o *Front Controller* é implementado com o auxílio do *despatcher*, onde o mesmo analisa a requisição feita pelo usuário e encaminha ao *controller* correto, ele serve como gerenciador para a escolha da próxima visualização ao usuário e para mover o *controller* ao reuso. Sendo assim, o *Front Controller* é usado para definir uma solicitação vinda da *Web*; com isso diminui os códigos existentes entre as camadas de visão e controle; definindo controles de decisão, reduzindo a lógica de negócio e assim permitindo a reutilização de código. Em resumo, o *Front Controller* recebe todas as requisições vindas da *view*, distribui e analisa as solicitações e é responsável pelo tratamento de erros.

E por fim o *Access Data Mapping* que é responsável por fornecer diretrizes passo a passo que ajudam a criar associações entre as tabelas do BD, mapear as dimensões e membros, e executar a associação, uma vez, que se faz necessário o tratamento de persistência dos dados provindos de tabelas associativas (grava uma associação com uma tabela; com chaves estrangeiras para as tabelas que estejam vinculadas pela associação). Assim, ao se usar uma associação, pode-se definir quais os dados de origem que serão movidos, como os dados serão agregados e para que destino movê-los, como relatado por (Fowler, 2004).

3 METODOLOGIAS

O projeto desenvolvido foi realizado utilizando o método iterativo e incremental, sendo assim a cada fase do desenvolvimento foi examinado e havendo a necessidade de modificações ou adições de novas funcionalidades, estas eram então incrementadas, dessa forma, o processo evoluiu em versões, através da construção incremental e iterativa de novas funcionalidades até que o sistema completo estivesse construído, como mencionado por (Luiz, 2004).

O trabalho foi realizado baseando-se em pesquisas bibliográficas relacionadas à utilização da ferramenta em questão bem como o padrão que esta contempla em sua estrutura.

As consultas foram realizadas através de sites, por esta ferramenta ser muito nova e possuir muito pouco material. Também foi feito um estudo a respeito de sites existentes com a mesma finalidade e, que serviram como base para o desenvolvimento, possibilitando o levantamento dos requisitos existentes e quais eram necessários acrescentar.

Foi desenvolvido os diagramas UML (*Unified Model Language*) para que estes representassem e definissem os módulos relacionados à aplicação, suas funcionalidades e como estes deveriam interagir.

Para tal desenvolvimento foi utilizada o *Jude* (http://jude.change-vision.com/jude-web/index.html), uma ferramenta de modelagem para UML que é livre e apresenta de forma clara os dados para o usuário, além de possuir layout intuitivo.

Para a modelagem dos dados ou mais especificamente dos diagramas MER (Modelo Entidade Relacionamento) do BD foi usado o DBdesigner (http://fabforce.net/dbdesigner4/), uma ferramenta que permite a visualização das tabelas e um melhor entendimento do funcionamento do BD. Já para a implementação do banco foi utilizado o MySql(http://www.mysql.com/) com o auxílio da ferramenta *PhpMyAdmin* (http://www.phpmyadmin.net/home_page/index.php) que proporciona uma interface gráfica que torna possível a criação, remoção e alteração de tabelas, inserção, alteração e remoção de dados de maneira rápida e fácil, bem como a execução de comandos SQL(*Structured Query Language*) e

manipulação de campos chaves .

A linguagem utilizada para a implementação é mesma que aborda o *framework* escolhido, o PHP; utiliza também o servidor *web* livre denominado de Apache (http://www.apache.org/), responsável por aceitar requisições HTTP(Hypertext Transfer Protocol) de cliente e servi-los também com respostas HTTP, geralmente pelos navegadores.

O PHP, MySQL e o Apache foram utilizados com o auxílio do Vertrigo (http://vertrigo.sourceforge.net/), este programa os instala, configura e gerencia de forma fácil e prática. Todas as ferramentas utilizadas no desenvolvimento do projeto foram escolhidas por serem de uso livre e gratuito e por apresentarem alto grau de satisfação no desenvolvimento de funcionalidades aos quais são especificados.

4 PROTÓTIPO DO WEBSITE

O trabalho desenvolvido consiste em um protótipo de um *Website* que ajuda uma ONG (Organização Não Governamental) que presta serviços de assistencialismo a animais carentes e de rua, dessa forma, foram detectados alguns requisitos funcionais do sistema.

O sistema deve permitir a qualquer pessoa interessada conhecer mais sobre como funciona a instituição, as pessoas que já fazem parte, os idealizadores do projeto e os animais para a adoção.

Deve ainda permitir a pessoa se registrar como padrinho (colaborador) para ajudar a manter a instituição. Nesse caso, é fornecida uma lista com os animais que estão na fila de adoção e (se estes possuir) mostrar os padrinhos ligados a cada animal.

O sistema também deve fornecer as seguintes consultas:

- Uma lista com os principais parceiros da instituição, contendo nome, endereço, telefone, endereço eletrônico, etc., para possíveis identificações.

- Uma lista com declarações de cada pessoa que contribui com a instituição.

No caso do padrinho, oferecer consultas sobre histórico de seus afilhados, produtos e serviços adicionais oferecidos aos animais.

4.1 Casos de uso

- **Casos de uso do usuário**

1. Cadastrar no sistema

Este caso de uso se inicia quando o usuário decide se cadastrar no sistema.

1- O usuário solicita a página de cadastro.

2- O sistema retorna página solicitada.

3- O usuário digita os dados.

4- O sistema grava os dados do usuário.

Fluxo alternativo:

3.1 Se usuário já estiver cadastrado, sistema emite mensagem.

4.1- O sistema emite mensagem de confirmação.

4.2- Caso ocorra algum erro, o sistema emite mensagem e pergunta se deseja se cadastrar novamente.

Include: Verificar cadastro.

Esse caso de uso é executado no momento em que o usuário insere seus dados para cadastro.

1- O sistema verifica se o nome do usuário já existe.

2.1- Se existir, ele retorna mensagem para página de cadastro.

2. Listar animais:

Este caso de uso irá ser realizado quando o usuário tiver interesse de conhecer os animais que estão cadastrados no sistema.

1-O usuário faz requisição de lista dos animais presentes na instituição.

2-O sistema busca no banco a relação de animais existentes.

3-O sistema retorna a página com todos os animais encontrados.

- **Casos de uso do padrinho**

1. Adotar Animal

Este caso de uso se inicia quando o usuário já cadastrado deseja adotar um animal.

1-O padrinho faz requisição de adoção.

2-O sistema retorna a página para adoção.

3-O padrinho insere os dados necessários para a adoção.

4-O sistema registra os dados da adoção.

5- O sistema emite mensagem de confirmação.

2. Verificar Histórico do Animal:

Este caso de uso se inicia quando o padrinho deseja verificar como está o histórico de seu afilhado na instituição.

1- O padrinho solicita histórico do afilhado.

2- O sistema solicita qual o afilhado.

3- O padrinho seleciona o afilhado.

4- O sistema retorna histórico do animal em questão.

3. Comprar Produtos/Serviços:

Este caso de uso é executado quando o usuário deseja comprar algum produto ou serviço para um animal.

1- O padrinho solicita página de compras.

2- O sistema retorna página solicitada.

3- O usuário seleciona o produto/serviço desejado.

4- O sistema solicita dados do animal para quem foi feita a compra e do padrinho.

5- O sistema registra pedido.

Fluxo alternativo:

3.1- Se o produto/serviço em questão não estiver disponível o sistema emite mensagem e pergunta se deseja sair ou comprar outro produto/serviço.

4. Deixar depoimento:

Este caso de uso irá ser inicializado quando o padrinho já cadastrado quiser deixar seu depoimento.

1-O padrinho faz requisição da página de depoimento.

2-O sistema devolve página solicitada.

3-O padrinho insere seu depoimento.

4-O sistema grava a informação.

- **Casos de uso do administrador**

1. Gerenciar Animal:

Caso de uso por onde o administrador poderá administrar o cadastro dos animais.

1- O administrador escolhe uma opção (incluir, alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Alterar.

Variante 3: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro.

1.1.1- O administrador insere os dados do animal.

1.1.2- O sistema confirma cadastramento.

Alterar:

1.1- O administrador seleciona o código do animal a alterar.

1.1.1- O sistema exibe a tela de cadastramento com os dados do animal.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona o animal a excluir.

1.1.1- O administrador confirma exclusão.

2. Gerenciar Apadrinhamento:

Caso de uso por onde o administrador poderá administrar o cadastro dos padrinhos.

1- O administrador escolhe uma opção (incluir, alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Alterar.

Variante 3: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro.

1.1.1- O administrador insere os dados do padrinho.

1.1.2- O sistema confirma cadastramento.

Alterar:

1.1- O administrador seleciona o código do padrinho a alterar.

1.1.1- O sistema exibe a tela de cadastramento com os dados do padrinho.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona o padrinho a excluir.

1.1.1- O administrador confirma exclusão.

3. Gerenciar Produto/Serviço:

Caso de uso por onde o administrador poderá administrar o cadastro dos produtos e serviços oferecidos.

1- O administrador escolhe uma opção(incluir, alterar ou excluir).

2- O administrador seleciona opção produto/serviço.

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Alterar.

Variante 3: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro.

1.1.1- O administrador insere os dados da opção (produto/serviço).

1.1.2- O sistema confirma cadastramento.

Alterar:

1.1- O administrador seleciona o código do produto/serviço a alterar.

1.1.1- O sistema exibe a tela de cadastramento com os dados.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador informa o código do produto/serviço a excluir.

1.1.1- O administrador confirma exclusão.

4. Gerenciar Vídeos:

Caso de uso por onde o administrador poderá administrar os vídeos dos animais.

1- O administrador escolhe uma opção (incluir ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro do vídeo.

1.1.1- O administrador insere os vídeos desejados referente ao animal.

1.1.2- O sistema confirma cadastramento.

Excluir:

1.1- O administrador seleciona o código do vídeo a excluir.

1.1.1- O administrador confirma exclusão.

5. Gerenciar Fotos do Animal:

Caso de uso por onde o administrador poderá administrar as fotos

dos animais.

1- O administrador escolhe uma opção (incluir ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro das fotos.

1.1.1- O administrador insere a foto desejada referente ao animal.

1.1.2- O sistema confirma cadastramento.

Excluir:

1.1- O administrador seleciona o código da foto a excluir.

1.1.1- O administrador confirma exclusão.

6. Gerenciar Relatos:

Caso de uso por onde o administrador poderá administrar relatos referentes aos animais.

1- O administrador escolhe uma opção (incluir, alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Alterar.

Variante 3: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro do relato.

1.1.1- O administrador insere dados do relato desejado referente ao animal.

1.1.2- O sistema confirma cadastramento.

Alterar:

1.1- O administrador seleciona o código do relato a alterar.

1.1.1- O sistema exibe a tela de cadastramento do relato desejado.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona o relato a excluir.

1.1.1- O administrador confirma exclusão.

7. Gerenciar Adoção:

Caso de uso por onde o administrador poderá administrar as adoções realizadas na instituição.

1- O administrador escolhe uma opção (alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Alterar.

Variante 2: Excluir.

Alterar:

1.1- O administrador seleciona o código da adoção a alterar.

1.1.1- O sistema exibe a tela de cadastramento da adoção desejada.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona a adoção a excluir.

1.1.1- O administrador confirma exclusão.

8. Gerenciar Donativos:

Caso de uso por onde o administrador poderá administrar os donativos referentes aos animais.

1- O administrador escolhe uma opção (alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Alterar.

Variante 2: Excluir.

Alterar:

1.1- O administrador seleciona o código do donativo a alterar.

1.1.1- O sistema exibe a tela de cadastramento do donativo desejado.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona o donativo a excluir.

1.1.1- O administrador confirma exclusão.

9. Gerenciar Relatos:

Caso de uso por onde o administrador poderá administrar os depoimentos deixados pelos padrinhos.

1- O administrador escolhe uma opção (alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Alterar.

Variante 2: Excluir.

1.1- O administrador seleciona o código do depoimento a alterar.

1.1.1- O sistema exibe a tela de cadastramento do depoimento desejado.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona os depoimentos a excluir.

1.1.1- O administrador confirma exclusão.

10. Gerenciar Pedidos:

Caso de uso por onde o administrador poderá administrar os pedidos referentes aos animais.

1- O administrador escolhe uma opção (incluir, alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Alterar.

Variante 3: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro do pedido.

1.1.1- O administrador insere dados do pedido desejado referente ao animal.

1.1.2- O sistema confirma cadastramento.

Alterar:

1.1- O administrador seleciona o código do pedido a alterar.

1.1.1- O sistema exibe a tela de cadastramento do pedido desejado.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona o pedido a excluir.

1.1.1- O administrador confirma exclusão.

11. Gerenciar Item_pedido:

Caso de uso por onde o administrador poderá administrar relatos referentes aos animais.

1- O administrador escolhe uma opção (incluir, alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Alterar.

Variante 3: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro do item_pedido.

1.1.1- O administrador insere dados do item_pedido desejado referente ao animal.

1.1.2- O sistema confirma cadastramento.

Alterar:

1.1- O administrador seleciona o código do item_pedido a alterar.

1.1.1- O sistema exibe a tela de cadastramento do item_pedido desejado.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona o item_pedido a excluir.

1.1.1- O administrador confirma exclusão.

12. Gerenciar prestadoras:

Caso de uso por onde o administrador poderá administrar relatos referentes aos animais.

1- O administrador escolhe uma opção (incluir, alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Alterar.

Variante 3: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro da prestadora.

1.1.1- O administrador insere dados da prestadora .

1.1.2- O sistema confirma cadastramento.

Alterar:

1.1- O administrador seleciona o código da prestadora a alterar.

1.1.1- O sistema exibe a tela de cadastramento da prestadora desejada.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona prestadora a excluir.

1.1.1- O administrador confirma exclusão.

13. Gerenciar usuários administradores:

Caso de uso por onde o administrador poderá administrar os usuários administradores da instituição.

1- O administrador escolhe uma opção (incluir, alterar ou excluir).

2- O sistema confirma operação.

Variante 1: Incluir.

Variante 2: Alterar.

Variante 3: Excluir.

Incluir:

1.1- O sistema exibe tela para cadastro do administrador.

1.1.1- O administrador insere dados do administrador.

1.1.2- O sistema confirma cadastramento.

Alterar:

1.1- O administrador seleciona o código do administrador a alterar.

1.1.1- O sistema exibe a tela de cadastramento do administrador desejado.

1.1.2- O administrador faz alterações necessárias.

1.1.3- O sistema confirma alterações.

Excluir:

1.1- O administrador seleciona o administrador a excluir.

1.1.1- O administrador confirma exclusão.

14. Gerar Relatório:

Este caso de uso se inicia quando o administrador solicitar algum tipo de relatório.

1- O administrador solicita um relatório.

2- O administrador seleciona qual a preferência do relatório em questão.

3- O sistema retorna relatório com os dados filtrados.

Na figura 3 segue a modelagem do diagrama dos casos de uso descritos acima.

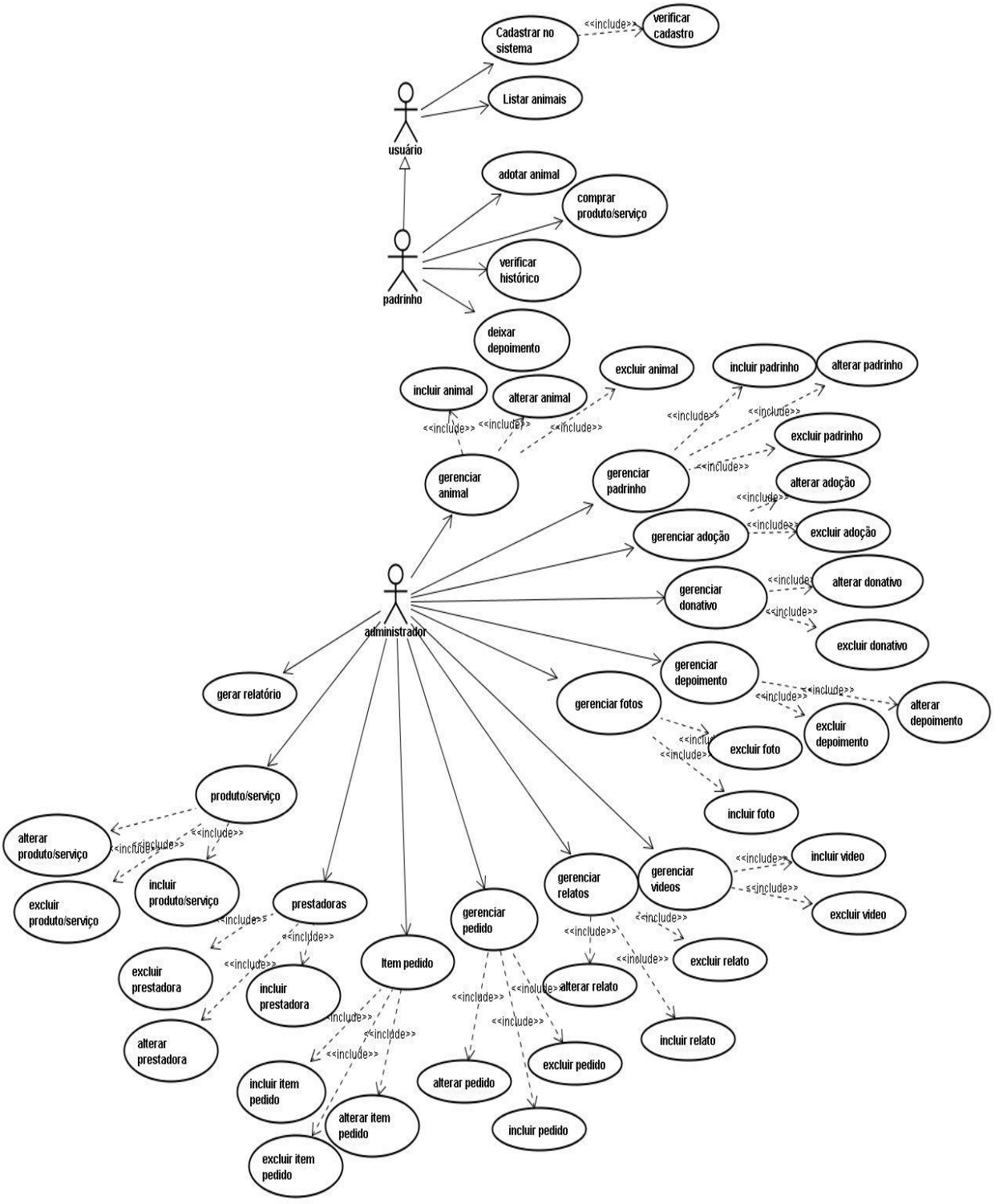

Figura 3: Casos de uso.

4.2 Diagrama de classe

Na figura 4 segue o diagrama de classe do protótipo em questão. É importante descrever aqui que existe a dependência no relacionamento das classes. Nota-se que a classe da lógica de negócio (*Model*) não faz chamada à classe *Controller*, da mesma forma que a *Controller* não faz chamadas a *View*. Além disso, a classe *View* não faz chamada diretamente a alguma função do *Model*, isso é feito através do *Controller*.

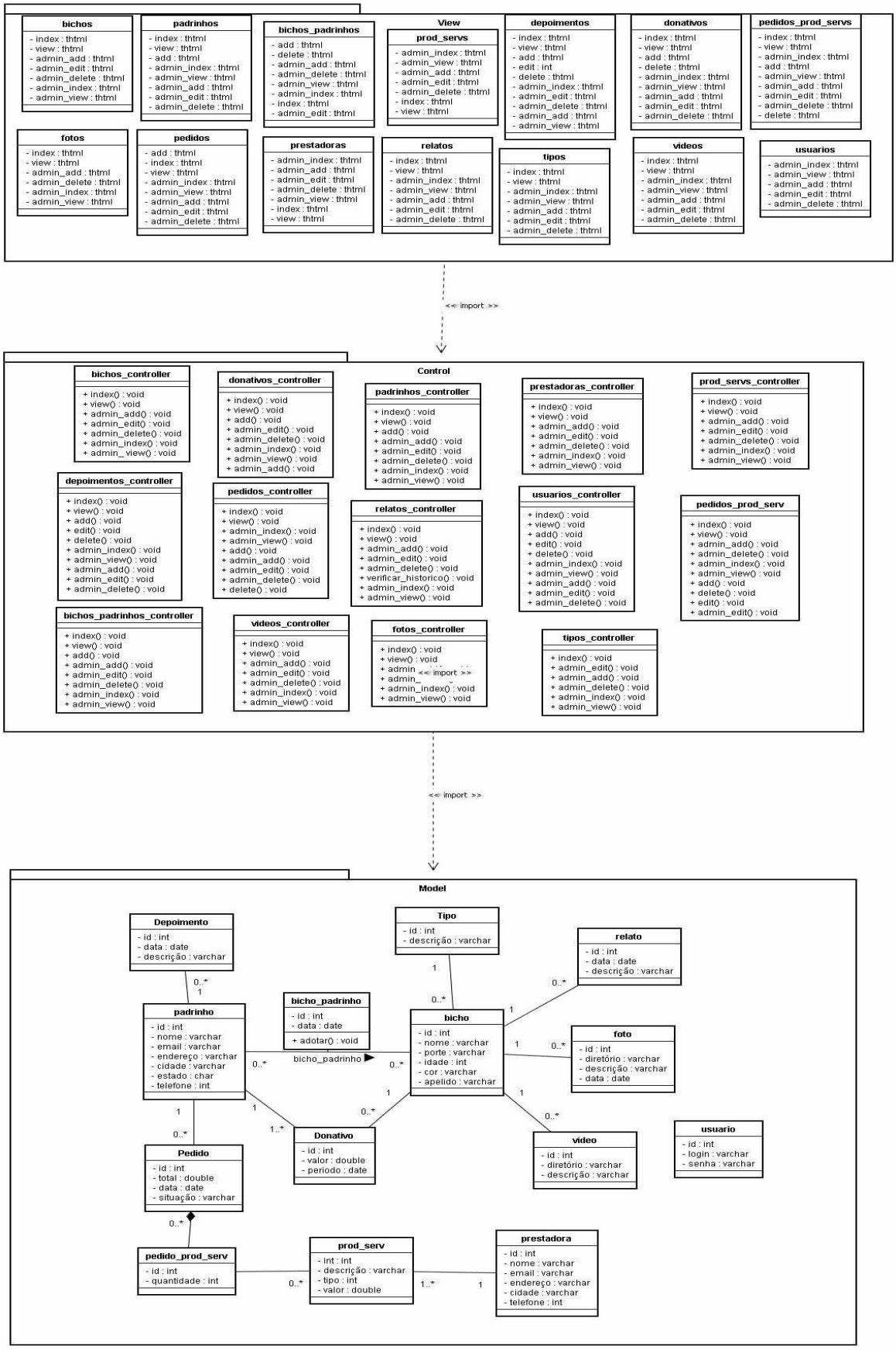

Figura 4: Diagrama de classe.

4.3 Diagrama de seqüência

Abaixo seguem os diagramas de seqüência correspondente ao protótipo em questão, começando com os diagramas relacionados aos usuários administradores que serão responsáveis pelo gerenciamento dos dados, seguindo pelos dos usuários que desejam conhecer mais sobre a instituição e por último os diagramas dos padrinhos. É importante lembrar que como o CakePhp faz uso do padrão *ActiveRecord,* a correspondência entre as classes e objetos com as tabelas do BD são feitas de forma automática, ou seja, a persistência do BD no CakePhp é feita implicitamente através de *Model* (acesso ao BD é feita através do *Model*),limitando o desenvolvedor a utilizar métodos auxiliares próprios do *Model,* também chamados de CRUD *(create Retrieve Update and Delete),* para que esse *Model* através do *ActiveRecord* possa então originar essa persistência, como descreve (Gabaldi, 2009).

1. Diagrama responsável por representar o login do usuário administrador no sistema.

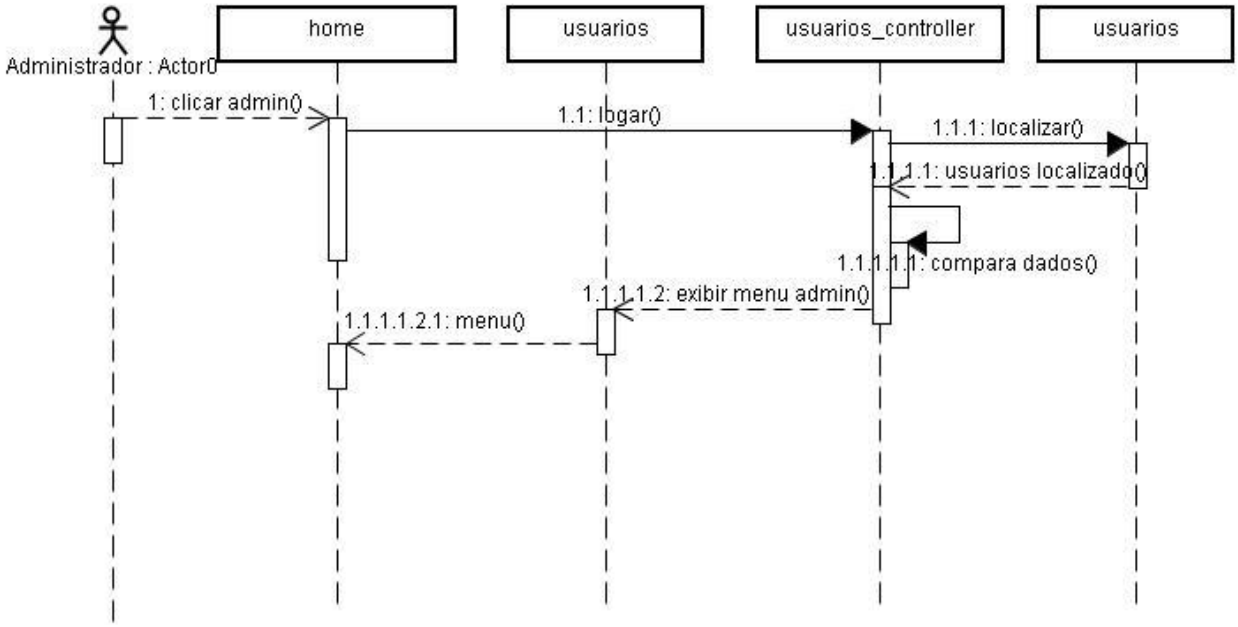

Figura 5: Logar administrador.

2. Diagrama responsável por representar o gerenciamento dos animais da instituição.

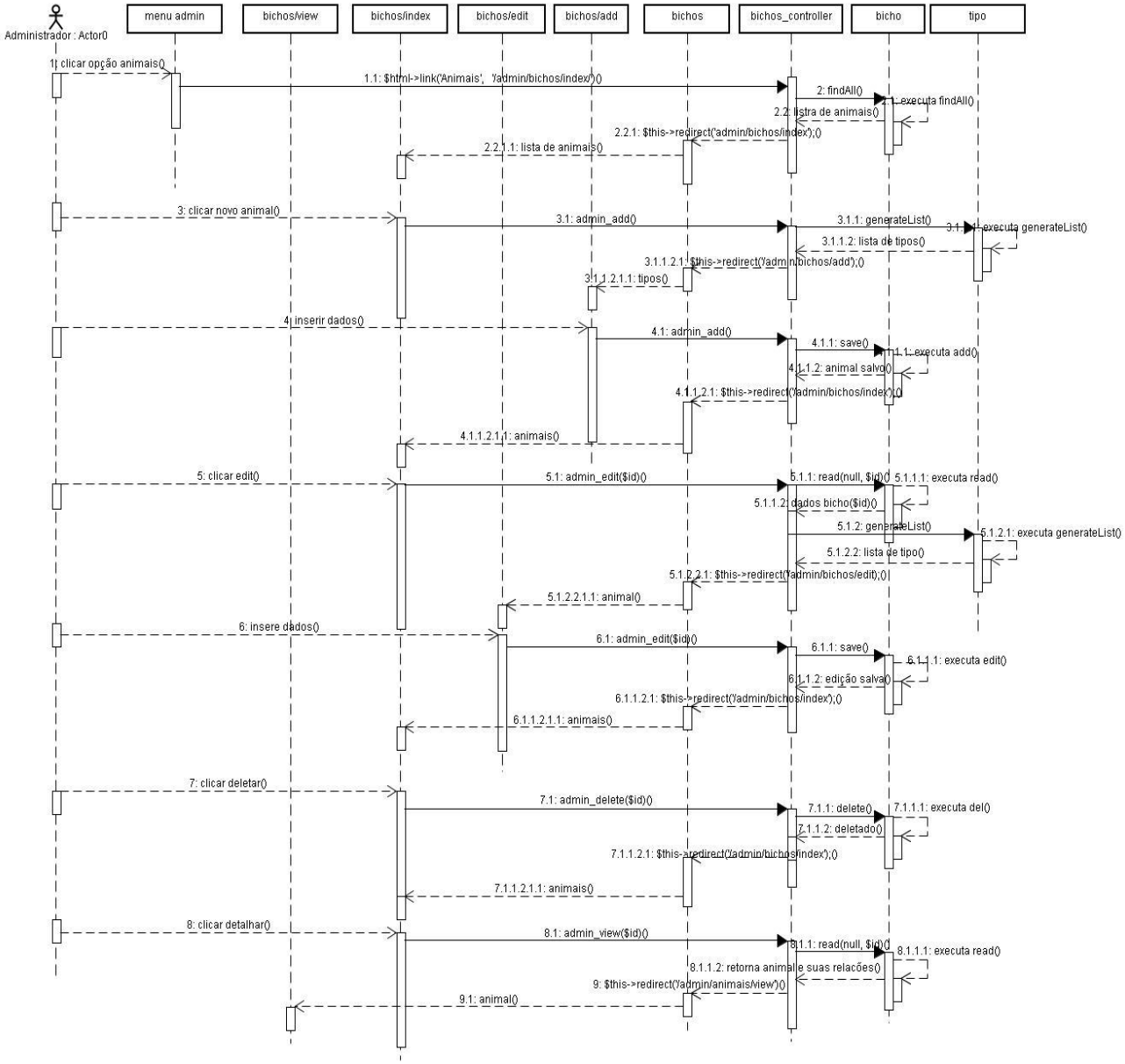

Figura 6: Gerenciar animais.

3. Diagrama responsável por representar o gerenciamento dos padrinhos cadastrados.

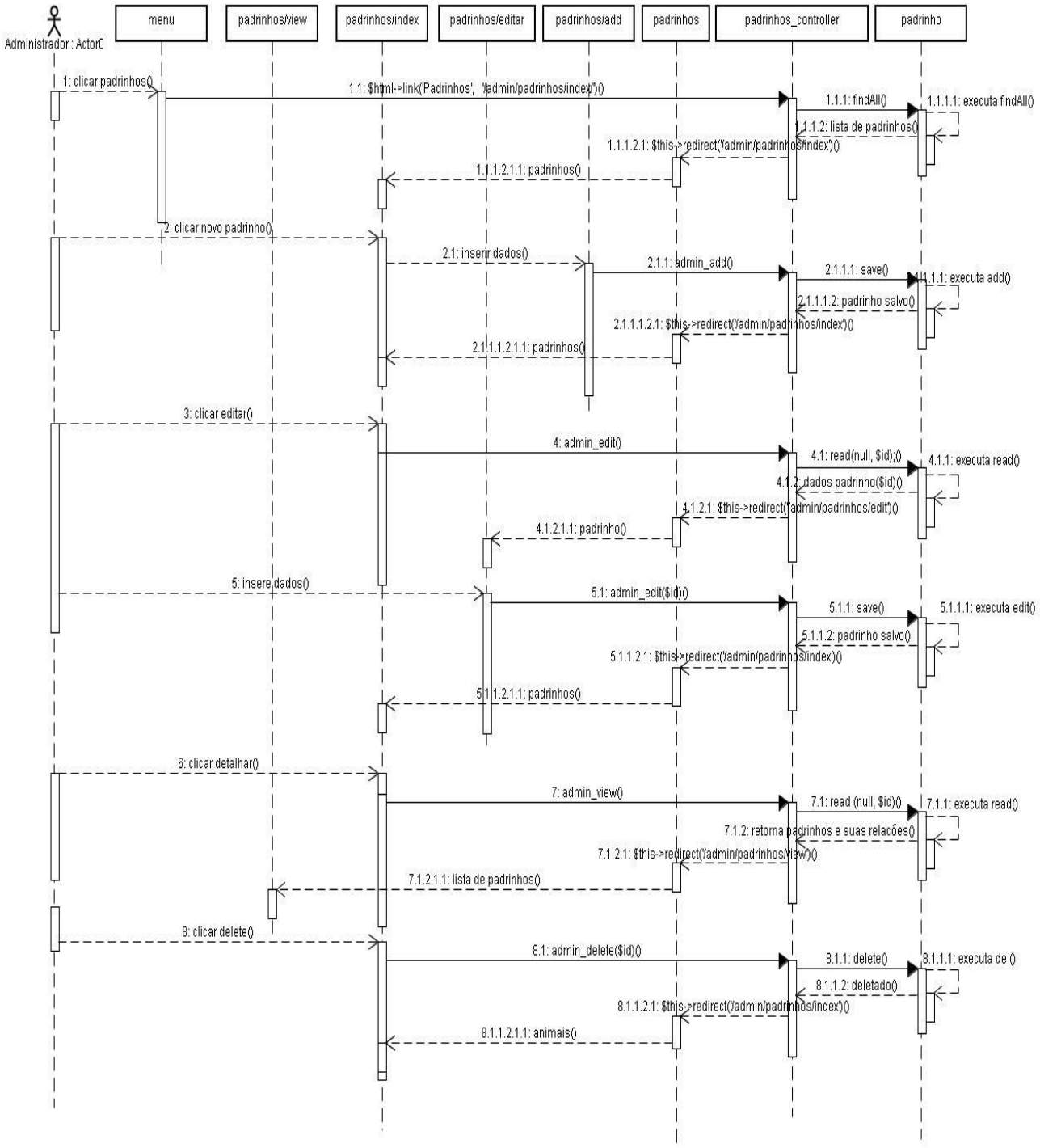

Figura 7: Gerenciar padrinhos.

4. Diagrama responsável por representar o gerenciamento das adoções realizadas, representada por convenção pela tabela bichos_padrinhos.

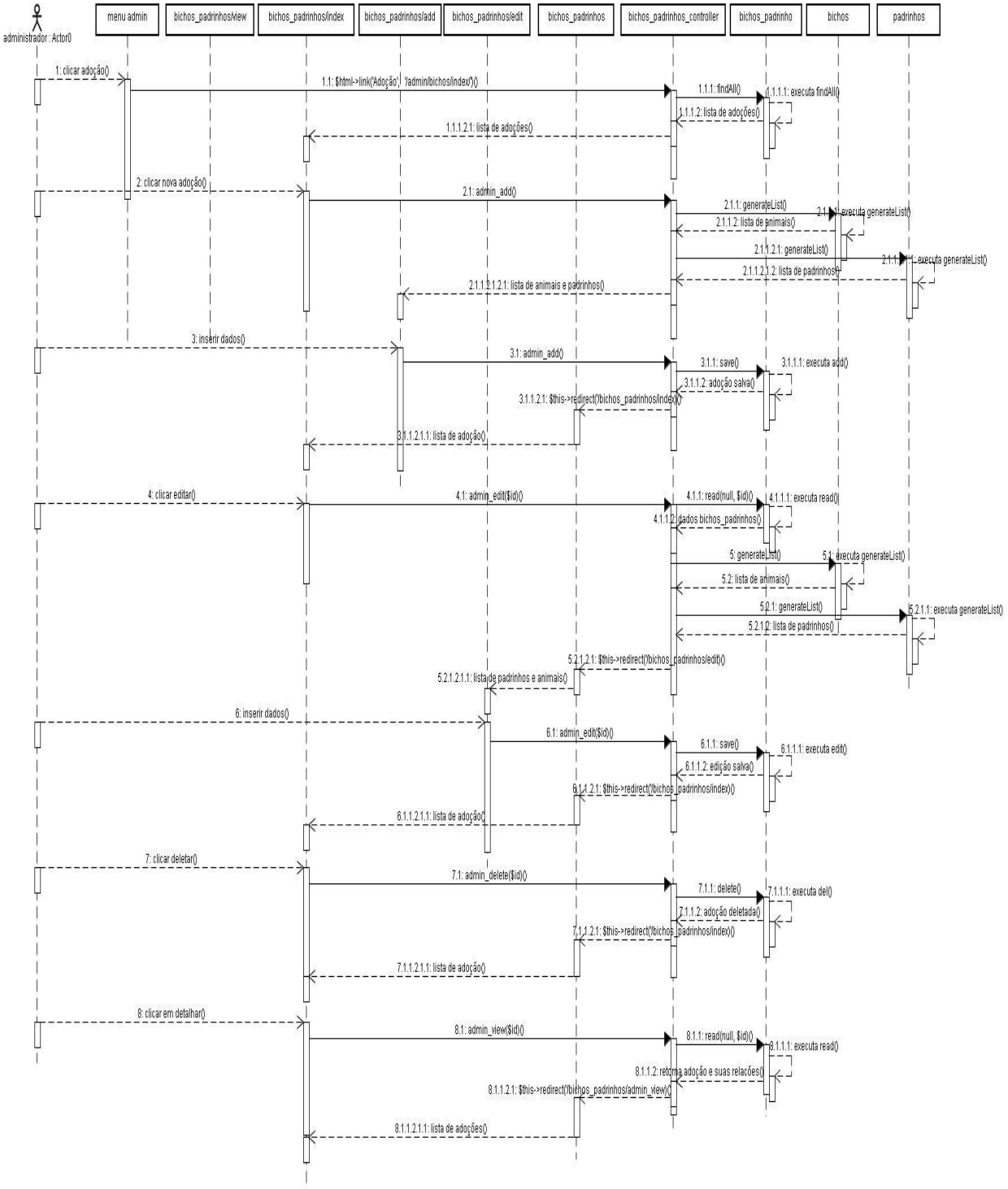

Figura 8: Gerenciar adoções.

5. Diagrama responsável por representar o gerenciamento dos donativos.

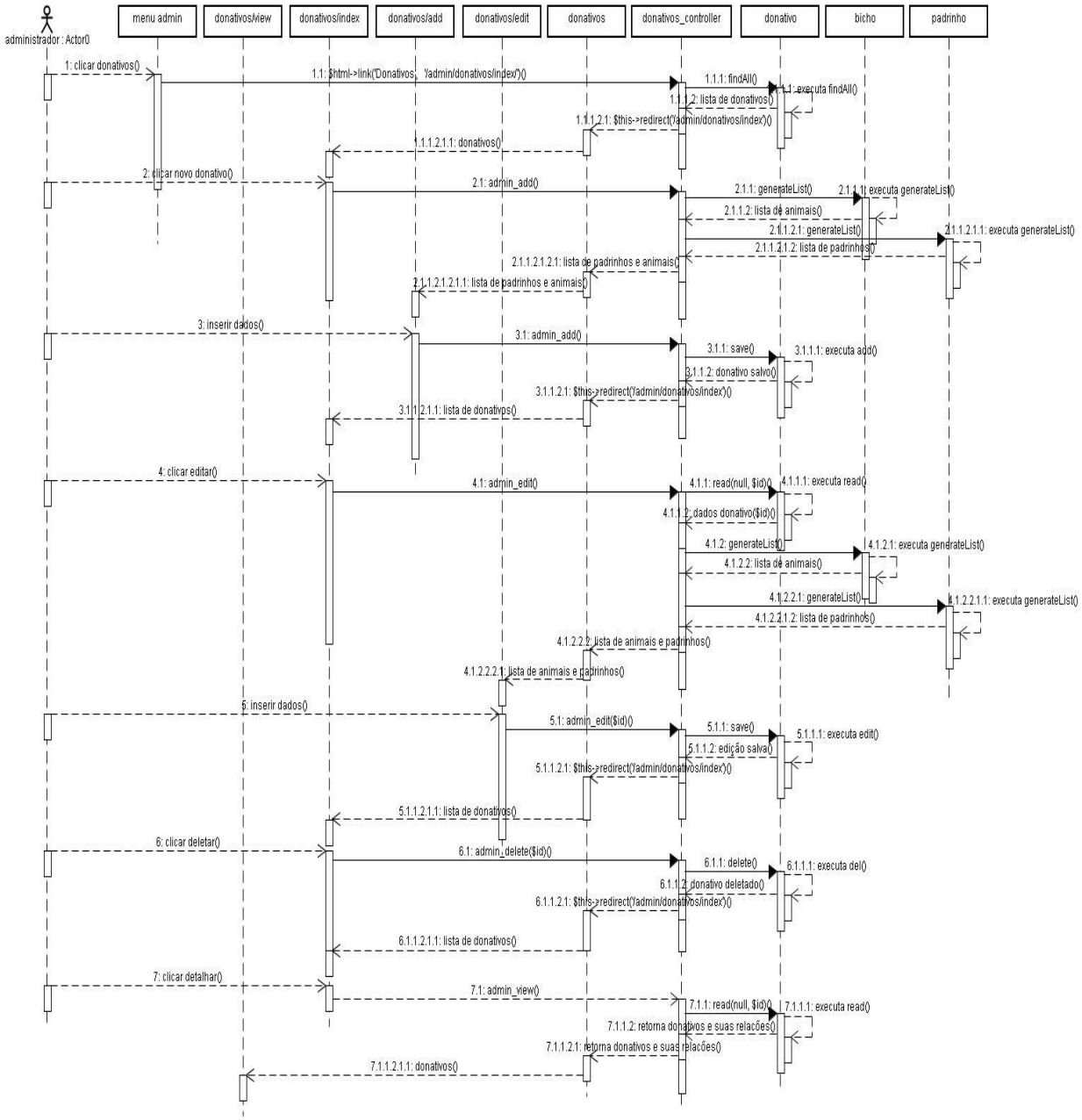

Figura 9: Gerenciar donativos.

6. Diagrama responsável por representar o gerenciamento dos depoimentos.

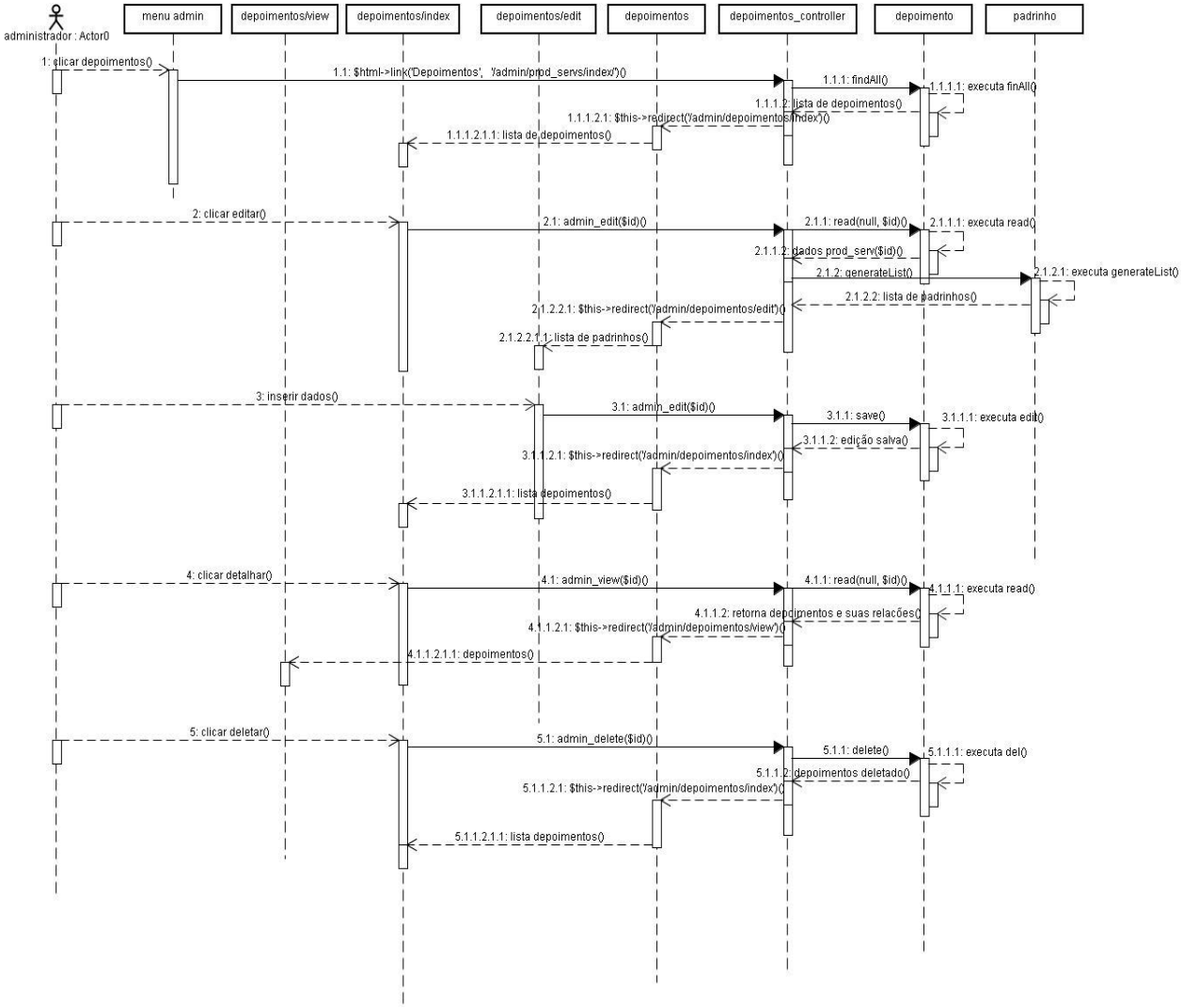

Figura 10: Gerenciar depoimentos.

7. Diagrama responsável por representar o gerenciamento das fotos.

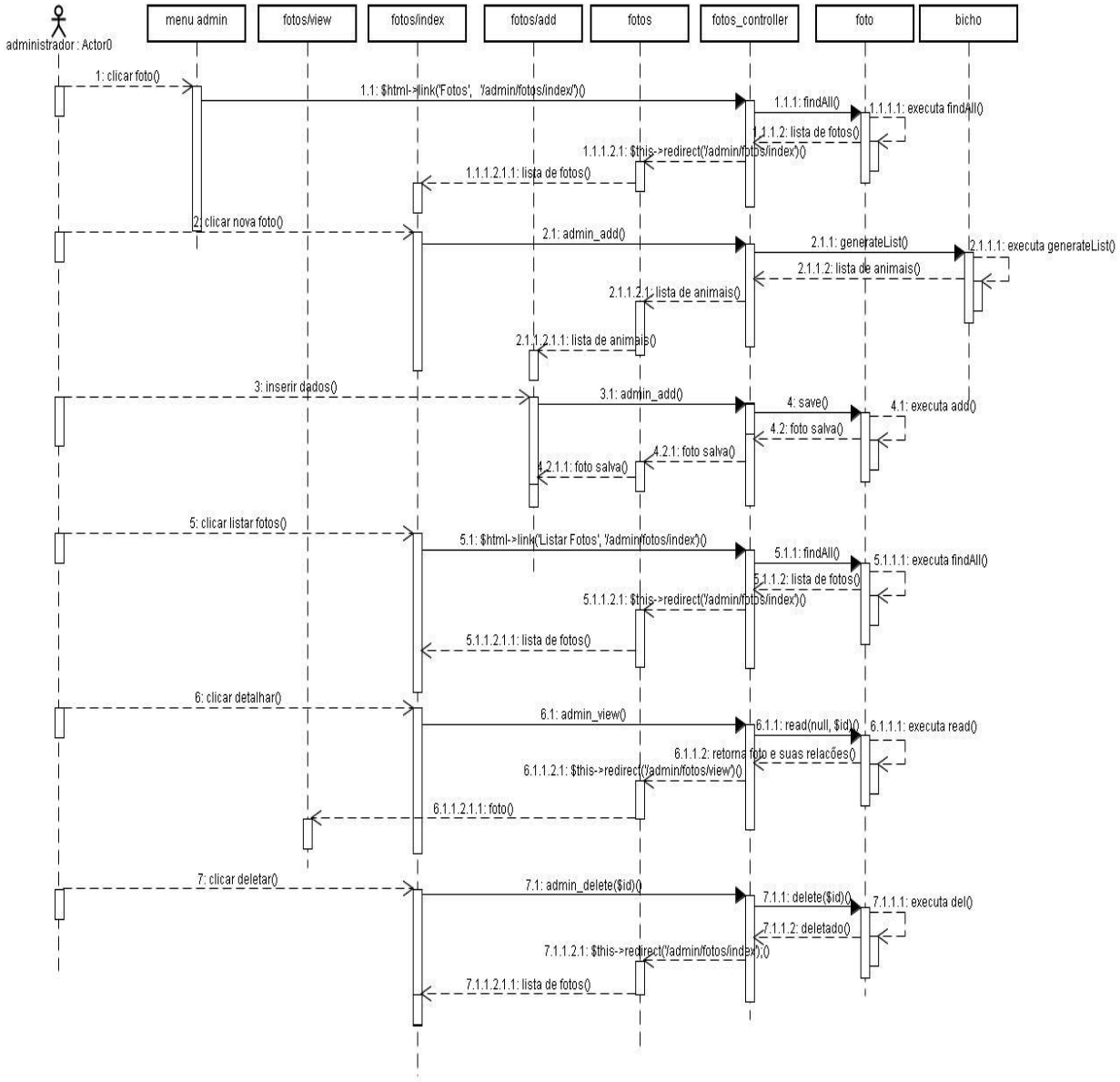

Figura 11: Gerenciar fotos.

8. Diagrama responsável por representar o gerenciamento dos pedidos.

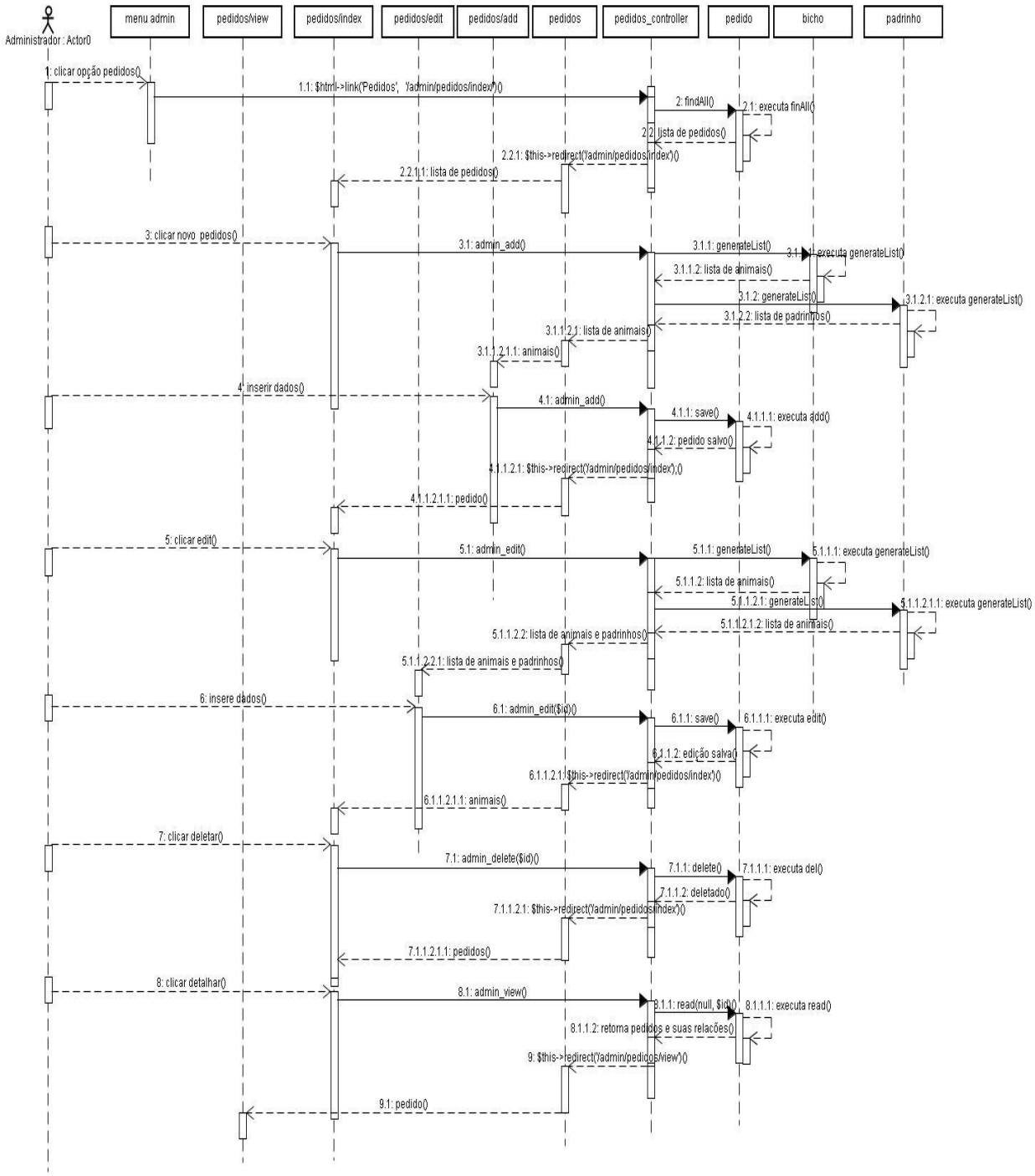

Figura 12: Gerenciar pedidos.

9. Diagrama responsável por representar o gerenciamento dos itens do pedido, que corresponde por convenção a tabela pedidos_prod_servs.

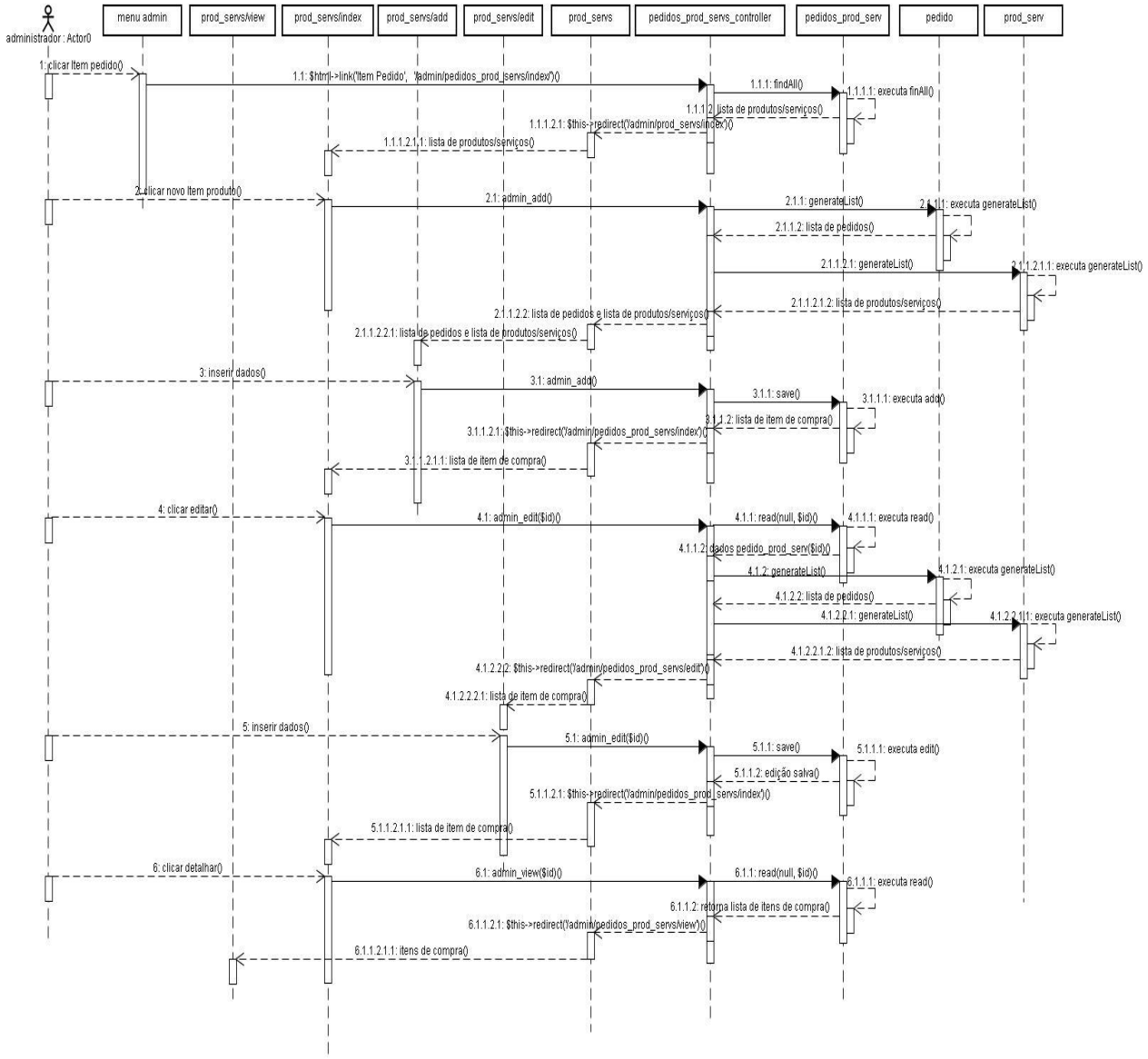

Figura 13: Gerenciar itens do pedido.

10. Diagrama responsável por representar o gerenciamento das prestadoras.

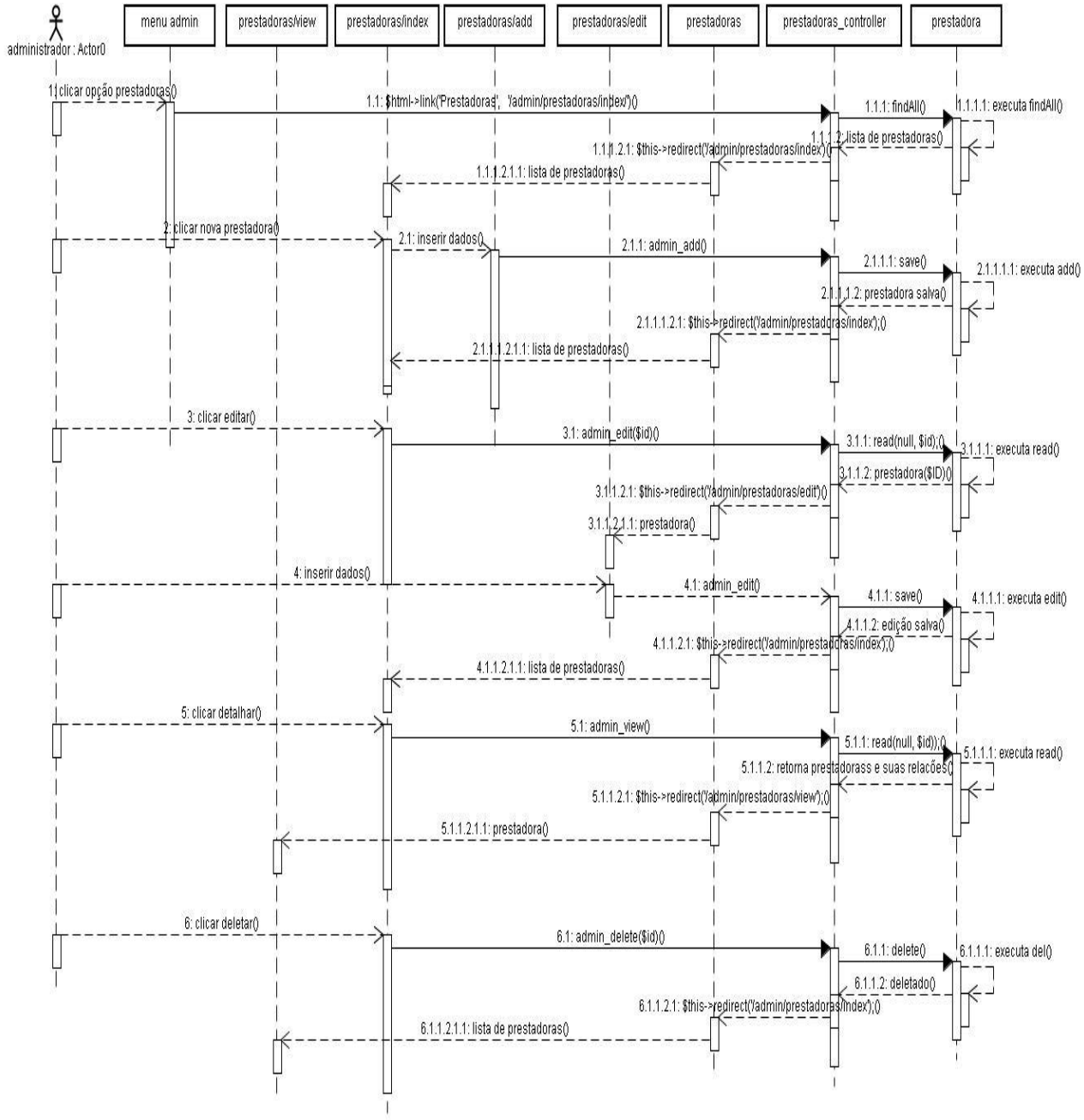

Figura 14: Gerenciar prestadoras.

11. Diagrama responsável por representar o gerenciamento dos produtos e serviços oferecidos.

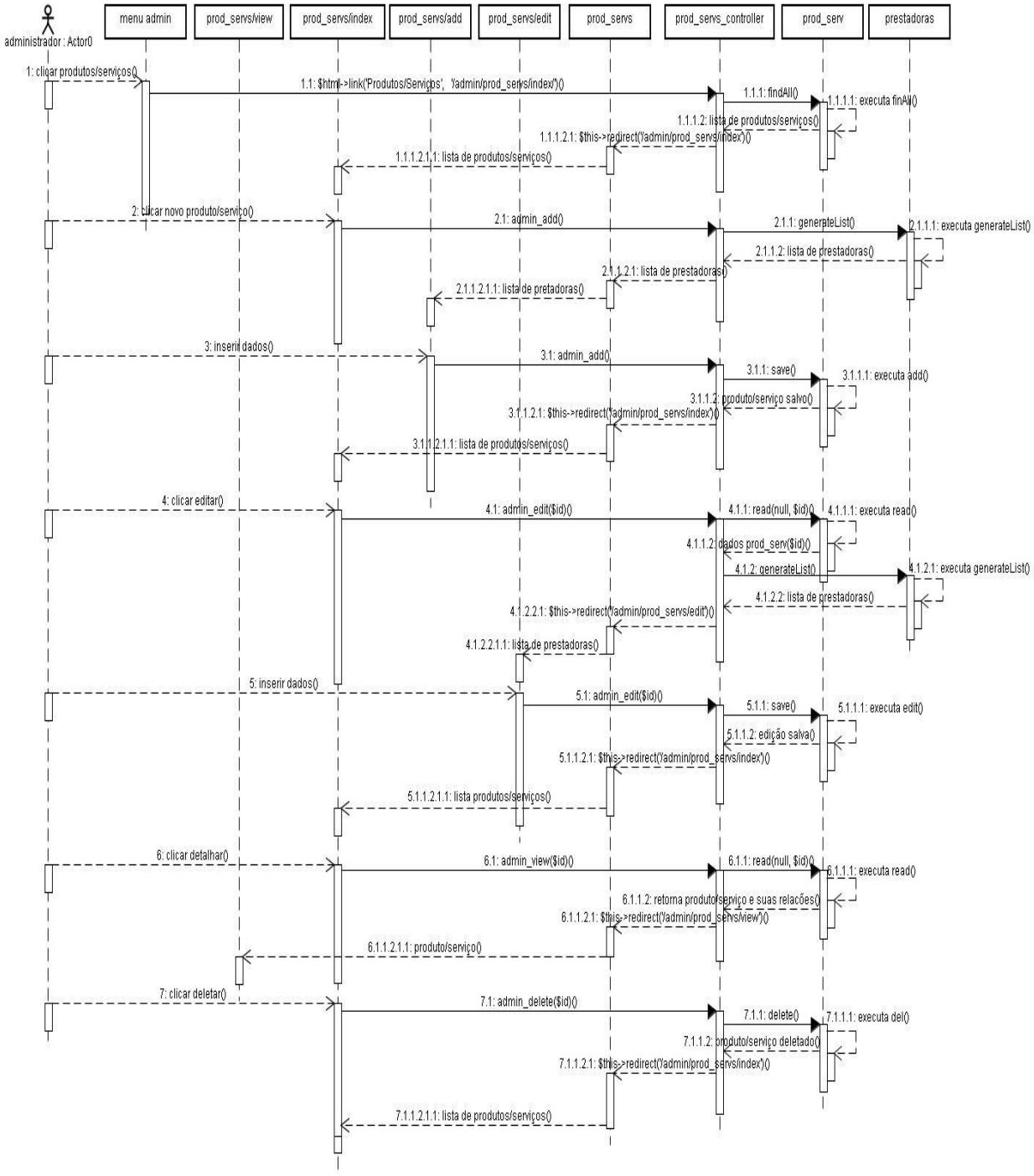

Figura 15: Gerenciar produtos/serviços.

12. Diagrama responsável por representar o gerenciamento dos relatos.

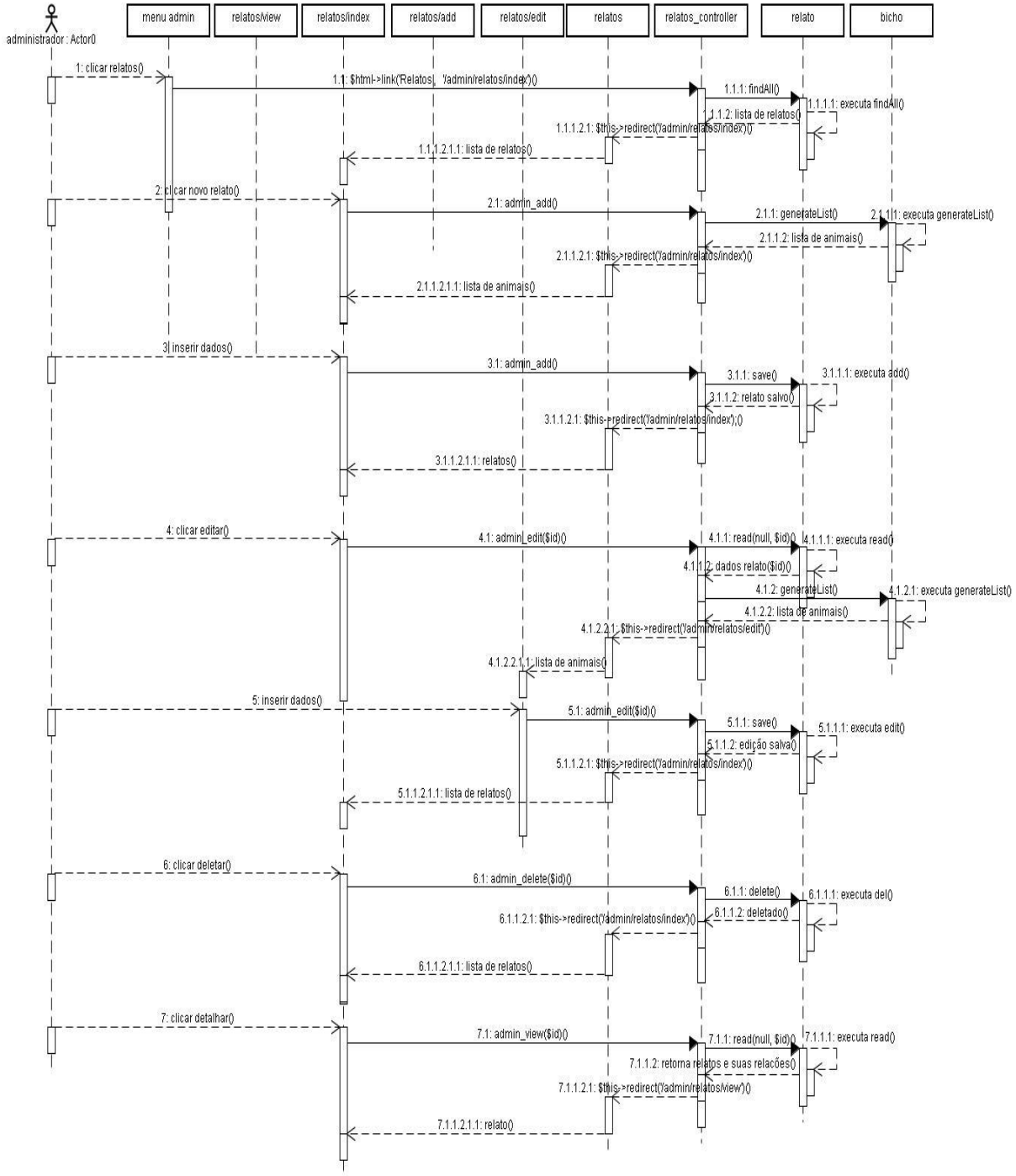

Figura 16: Gerenciar relatos.

13. Diagrama responsável por representar o gerenciamento dos tipos de animais que se encontram na instituição.

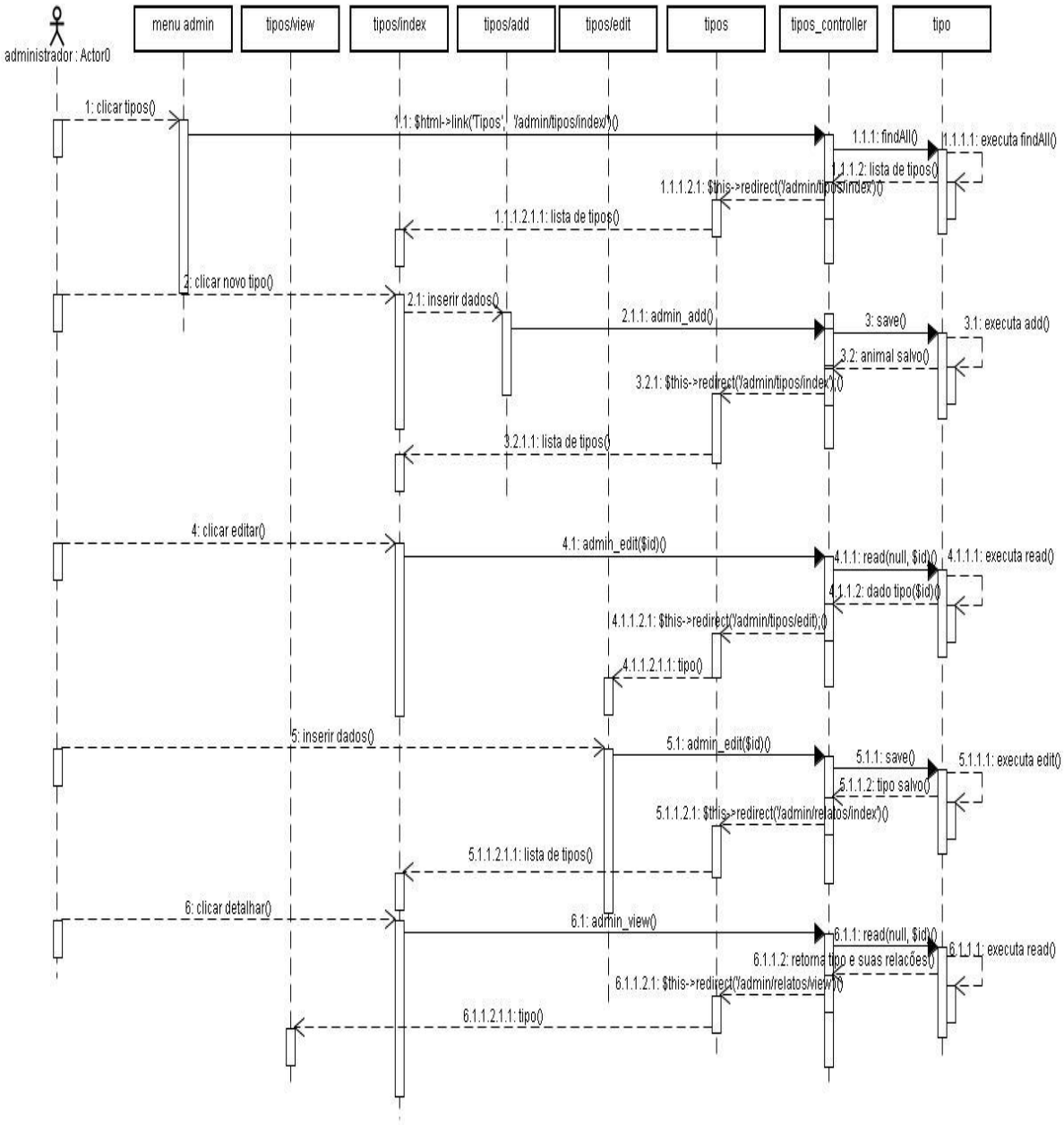

Figura 17: Gerenciar tipos.

14. Diagrama responsável por representar o gerenciamento dos usuários administradores.

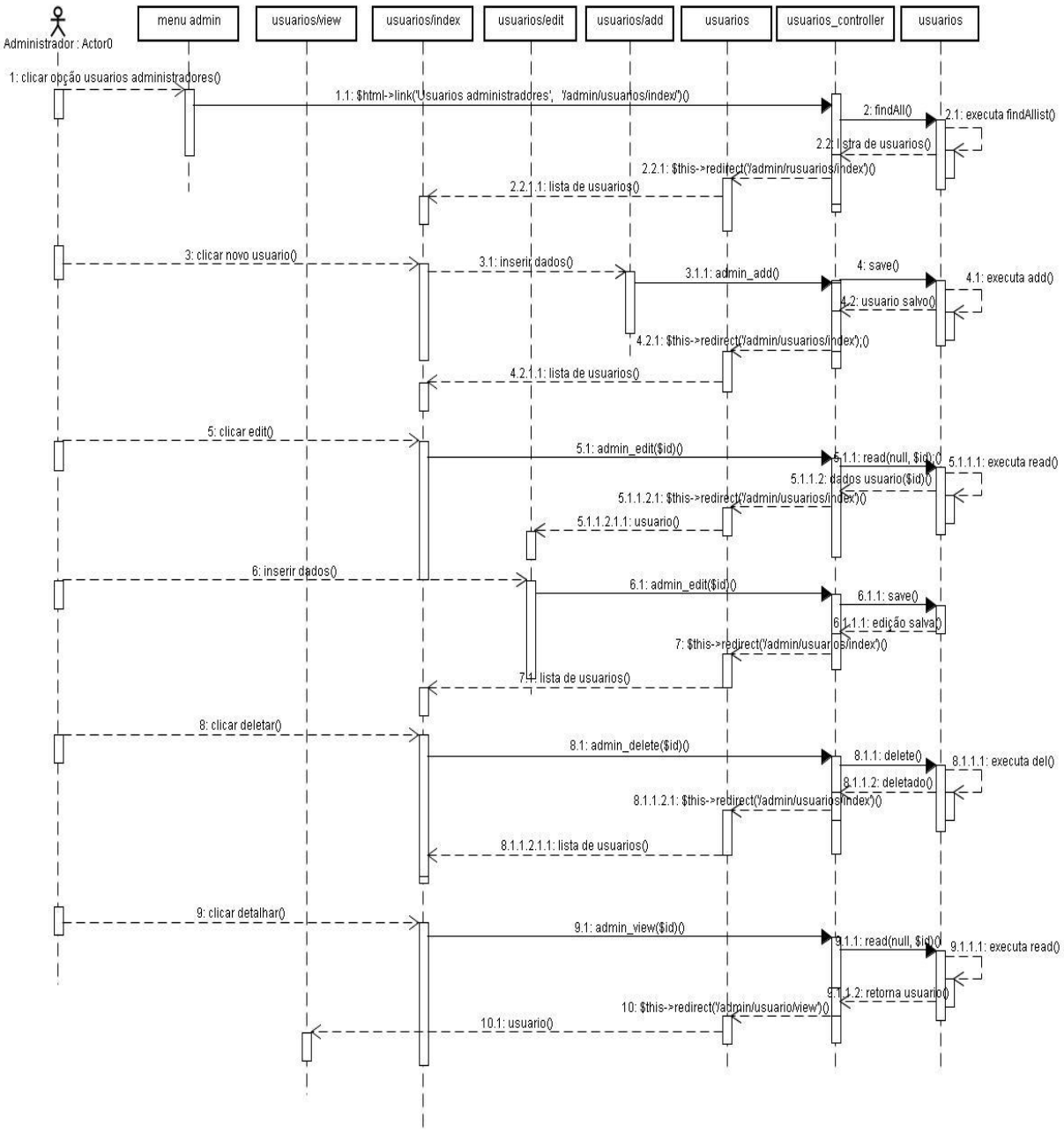

Figura 18: Gerenciar usuários administradores.

15. Diagrama responsável por representar o gerenciamento dos vídeos.

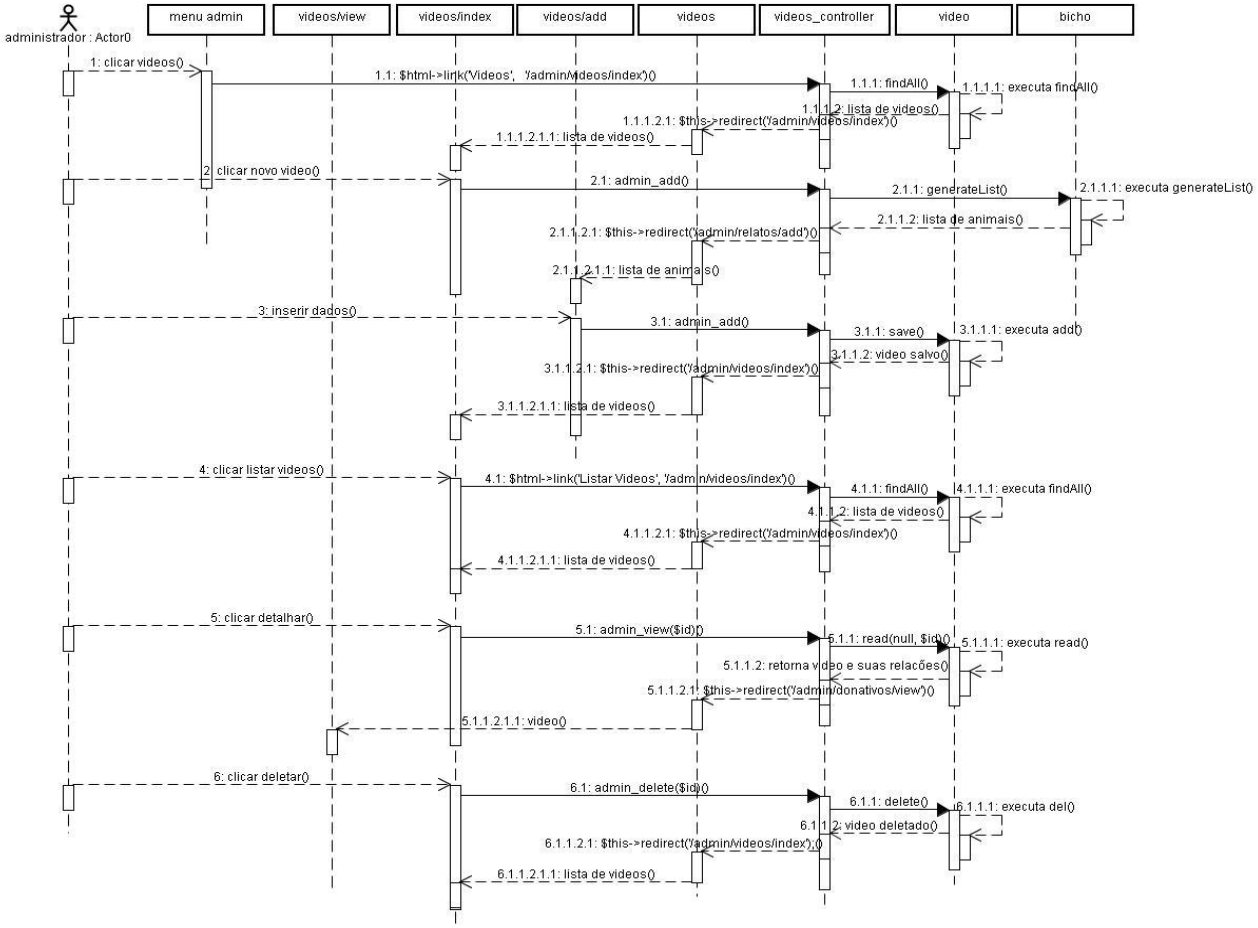

Figura 19: Gerenciar vídeos.

16. Diagrama responsável por representar o cadastro do usuário no sistema, para assim se tornar padrinho.

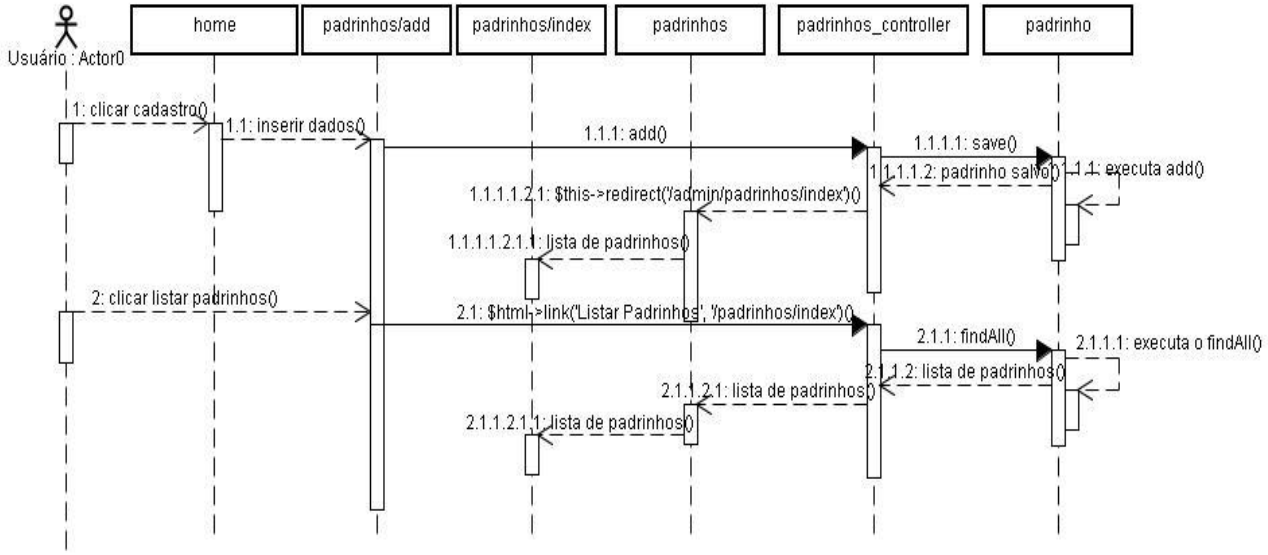

Figura 20: Cadastrar usuário no sistema.

17. Diagrama responsável por representar a visualização dos animais pelo usuário.

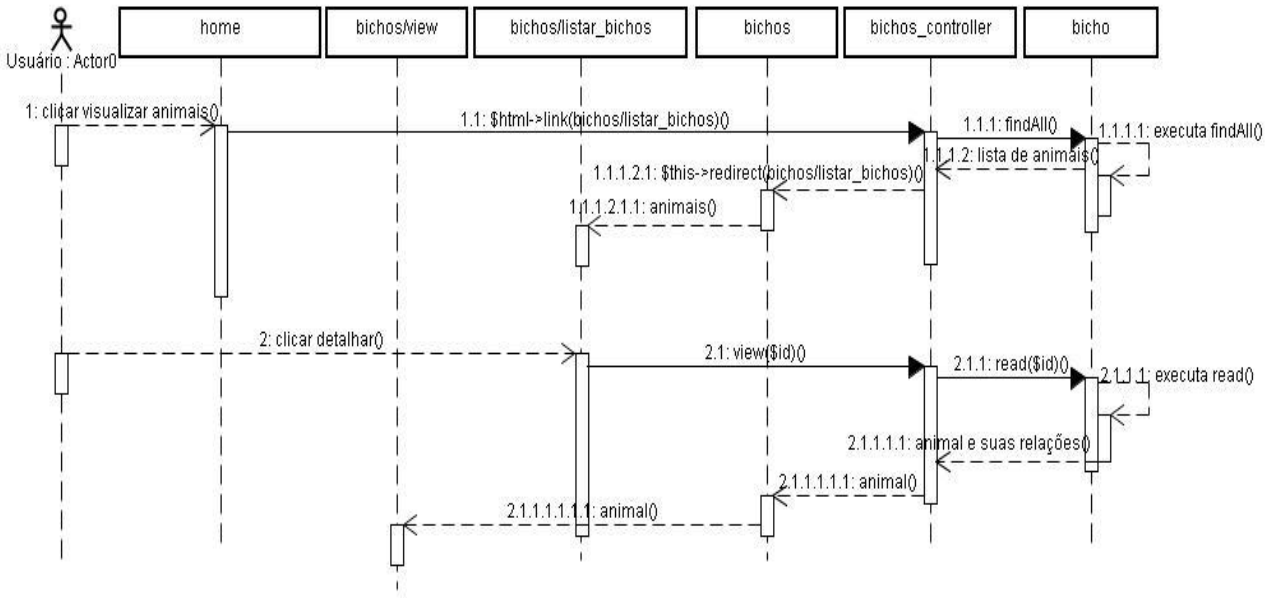

Figura 21: Listar animais.

18. Diagrama responsável por representar a adoção feita pelo padrinho.

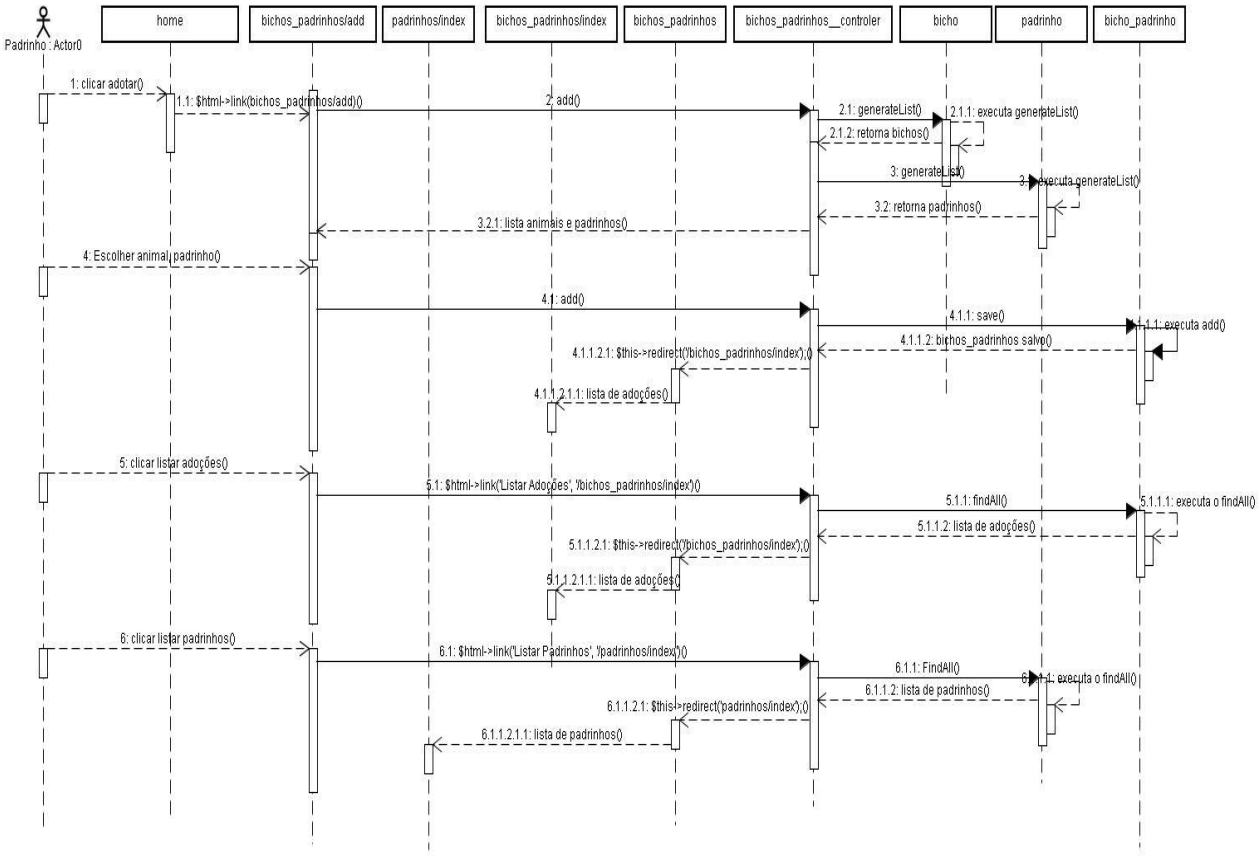

Figura 22: Adotar.

19. Diagrama responsável por representar a visualização dos relatos do animal pelo padrinho.

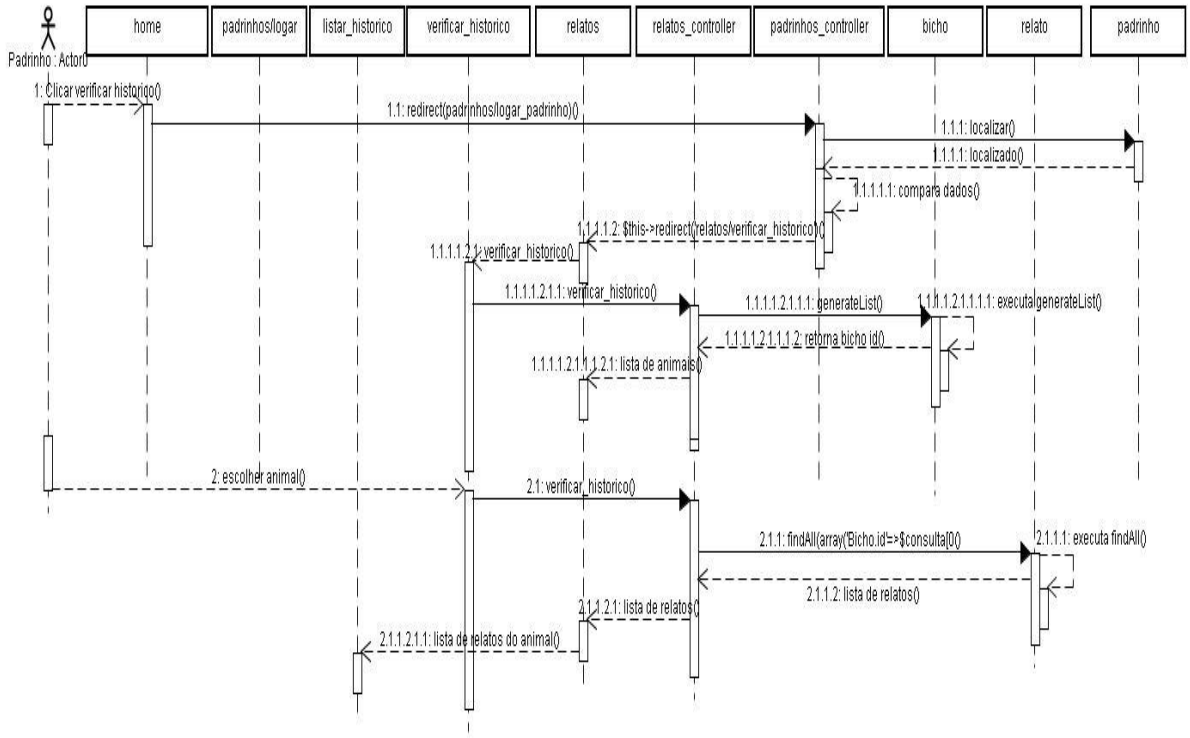

Figura 23: Verificar histórico.

20. Diagrama responsável por representar a adição de depoimentos pelos padrinhos.

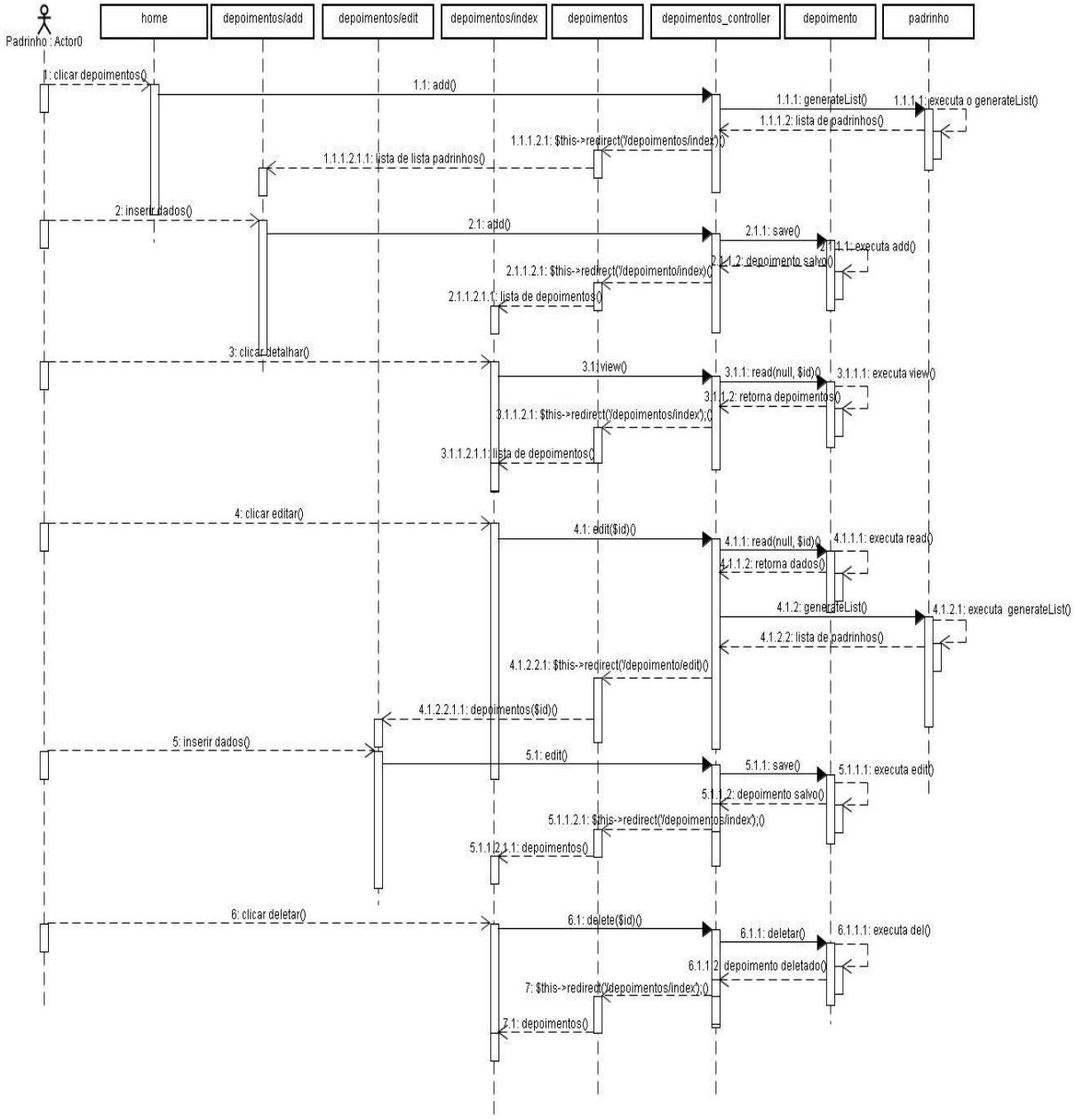

Figura 24: Deixar depoimento.

21. Diagrama responsável por representar a compra de produtos para os animais.

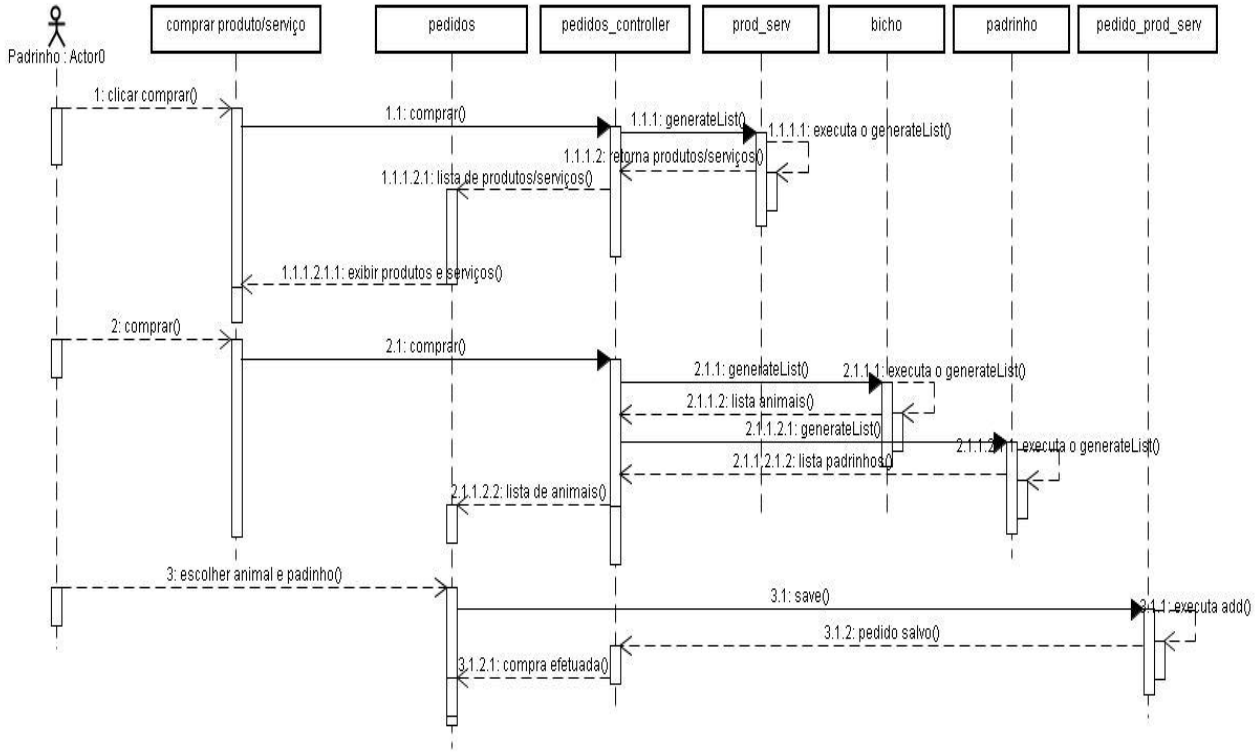

Figura 25: Comprar produto.

4.4 Modelagem do BD

A figura 26 mostra o modelo físico do BD utilizado no desenvolvimento do protótipo, com seus relacionamentos.

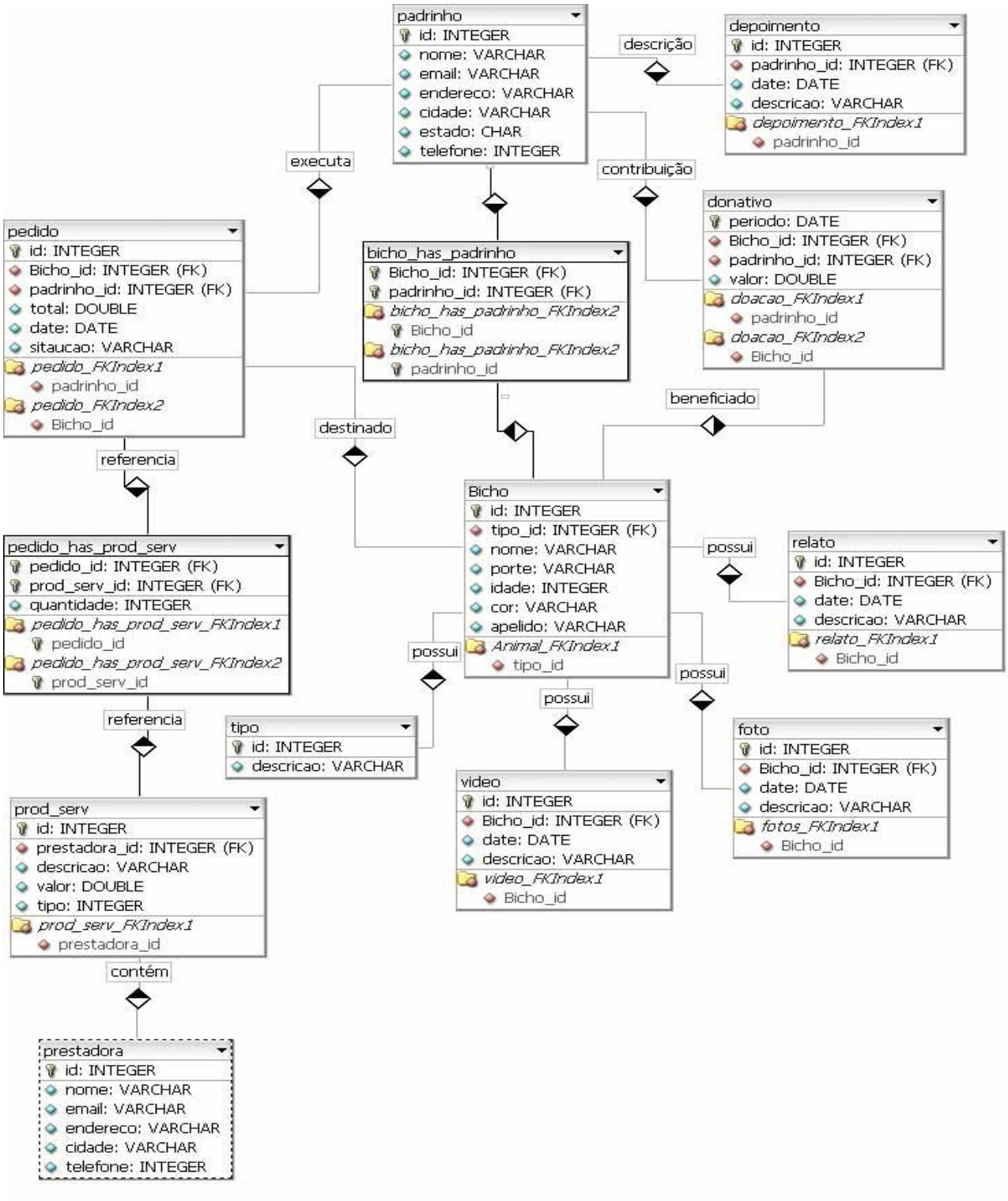

Figura 26: Modelagem do BD.

4.5 Telas do protótipo

Abaixo seguem três exemplares de telas do protótipo desenvolvido, sendo que a primeira representa a tela inicial, a segunda um exemplo do caso de uso inserir depoimento e a última o resultado apresentado pelo caso de uso citado no segundo exemplo.

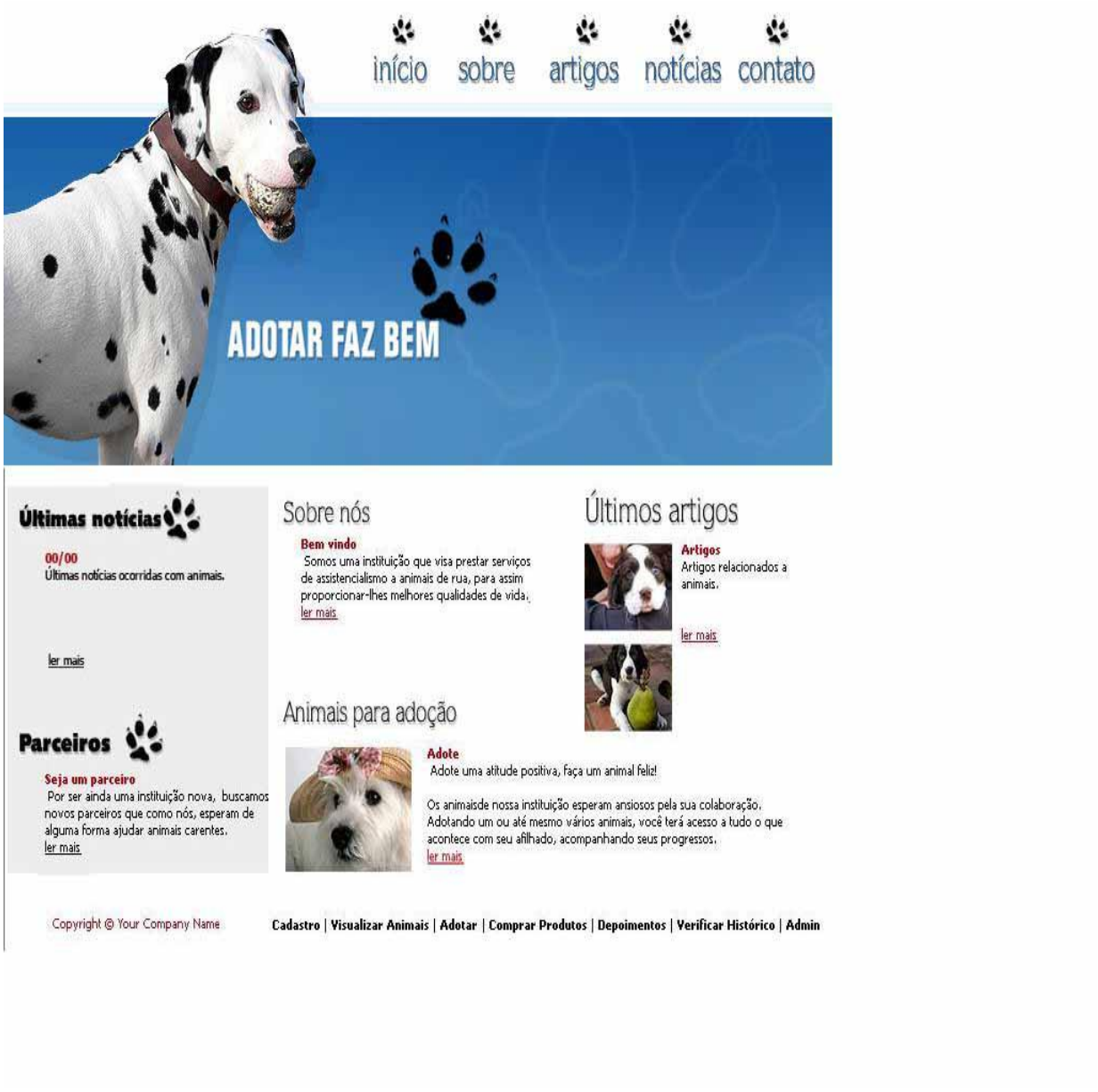

Figura 27: Tela inicial.

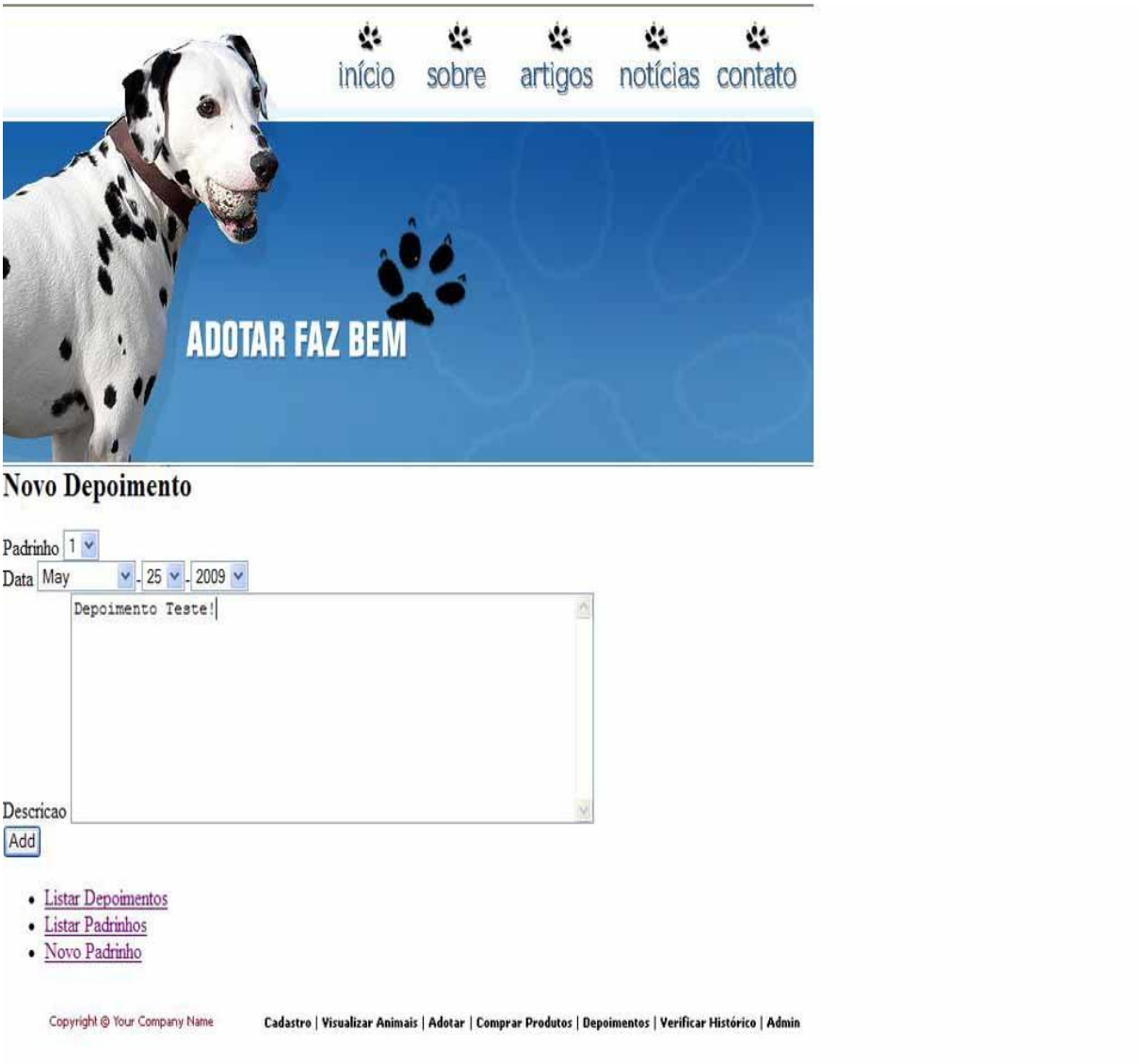

Figura 28: Tela deixar depoimento.

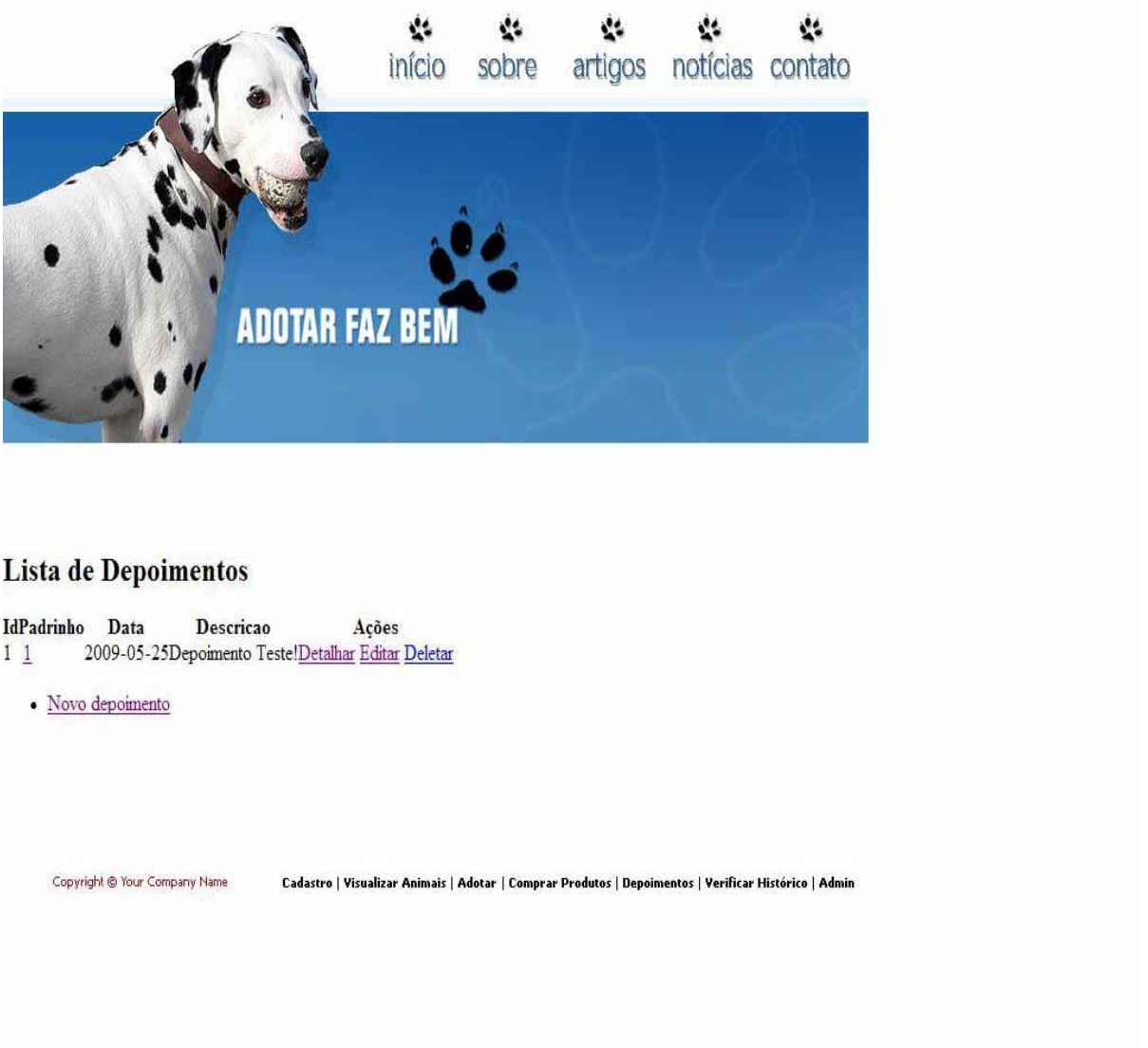

Figura 29: Tela depoimentos/index.

5 CONCLUSÕES E TRABALHOS FUTUROS

Em presença da necessidade de se conceder a animais abandonados a oportunidade de adquirir condições mínimas de bem estar, foi desenvolvido um protótipo de um *Website* filantrópico para possibilitar a interação entre as pessoas e esses animais, e assim facilitar, às pessoas que desejarem, a oportunidade de colaborar sem que precisem passar por adaptações em suas rotinas, mas fornecendo-lhes respostas às contribuições feitas, mantendo assim a veracidade dos propósitos da aplicação.

Esse protótipo consiste em uma aplicação feita em *CakePhp* e baseada no padrão de projeto MVC, que auxilia o desenvolvimento da aplicação de forma organizada, já que impõe a necessidade de manter a aplicação disposta nos devidos diretórios. Quanto ao *Model,* este representa os dados da aplicação e as regras de negócio que gerencia o acesso e modificações dos dados, ele encapsula funções, dentre elas a de apresentação e de armazenamento para o CRUD, funcionalidades estas ganhas com poucas linhas de código e nenhuma de SQL, o *controller* fica responsável por processar todas as requisições feitas pela interface e acessar o *model* para obter determinadas informações, de maneira sucinta, ele chama métodos sobre o *model* que altere o estado desse *model*; a *View* é capaz de obter dados do *model* e apresentá-los da melhor maneira para o usuário, os arquivos das *views* são escritos em php, que contém toda apresentação necessária para transformar os dados provindos do *controller* em um formato comum ao usuário, a *view* possui *helpers* que facilitam o desenvolvimento de formulários , criação de *links* e botões, muito mais simples e com menos código do que o próprio HTMl, estes *helpers* já vem como padrão no *CakePhp* e tornam as páginas mais leves. Vale lembrar que a configuração do BD é simples e não necessita ser especificada em cada arquivo da aplicação, uma vez que se é utilizado o nome da tabela do BD pela tríade, sendo que o *model* utiliza-o no singular, o *controller* no plural acrescido do underline e a *view* no plural.

Ainda em relação ao *CakePhp*, este possui também a vantagem de gerar código automático através do *bake* para a interação com o BD, também chamado de CRUD, e que é facilmente adaptado as necessidades do protótipo. O *bake* é uma excelente ferramenta que cria todas as funções CRUD relacionando as

tabelas associativas, acelerando assim, o processo de desenvolvimento, já que cuida dessas funções que são repetitivas.

Esse trabalho sintetiza a contribuição do desenvolvimento de aplicações baseadas no MVC, apesar de haver um maior número de arquivos dispostos na aplicação é visivelmente notório que a manutenção na aplicação é muito mais fácil, uma vez que se for necessário, a mudança será feita somente no que for preciso, de forma independente; assim como se for necessária a modificação do banco de dados, não haverá a necessidade de modificar a aplicação em si.

No decorrer do desenvolvimento do trabalho acorreram situações adversas na fase de implementação, no que diz respeito a compreensão da utilização do layout pelo *CakePhp*, pois o mesmo utiliza um layout padrão com *helpers* padrões que se encontra dentro do núcleo do *framework* e, outro problema que trouxe certas dificuldades foi de como apresentar dados provindo de um *model* para uma visão com nome diferente da ação que este estava executando, ambos tomando proporções maiores de tempo do que o esperado.

Quanto a intenção de propor soluções viáveis em relação aos problemas encontrados nas antigas instituições analisadas, podemos considerar que o protótipo alcançou esse objetivo de forma eficaz, uma vez que sua estrutura disponibiliza os conteúdos intrínsecos e relevantes dos animais que por ventura estejam cadastrados na instituição, garantindo assim a fidedignidade das intenções expostas nesse trabalho.

Podemos também atestar que esse protótipo de *website* é um canal de comunicação eficaz no que diz respeito a relação entre pessoas e animais, pois assegura que as pessoas que são colaboradoras tenham acesso a qualquer momento às ocorrências ocasionadas aos seus afilhados, em um breve espaço de tempo. Quanto a viabilidade do *Website*, este também se torna viável a medida que é possível realizar a implementação do mesmo, sem perdas peculiares de suas funcionalidades e sem trazer maiores dificuldade na sua implantação, bem como a isenção de custos, por ser totalmente construído com ferramentas livres.

Dentre as sugestões de trabalhos futuros, pode-se colocar a implantação do protótipo de *Website* filantrópico, a fim de concretizar os propósitos

do presente trabalho e implementar novas funcionalidades ao mesmo, para assim melhorá-lo consideravelmente.

REFERÊNCIAS

ALMEIDA, Rodrigo Rebouças de. **Model view controller: MVC**. Campina Grande, 2003.Disponível em: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm. Acesso em 19 de outubro de 2008.

Burbeck, Steve. **Applications Programming in Smalltalk-80(TM):How to use Model-View-Controller (MVC)**. 1992. Disponível em: http: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html. Acesso em 14 de outubro de 2008.

DBdesigner (Visual Database Design System), 2009. Disponível em: http://fabforce.net/dbdesigner4/. Acesso em 8 de outubro de 2008.

ESCOVEDO, Tatiana. **IssueNet: Um Framework para Avaliação Colaborativa de Tarefas**. Rio de Janeiro, 2007. 120f. Dissertação (Mestrado em Informática). Departamento de Informática, Pontifícia Universidade Católica. Disponível em: http://www.maxwell.lambda.ele.puc-rio.br/cgi-bin/PRG_0599.EXE/11336_1.PDF?NrOcoSis=36567&CdLinPrg=pt. Acesso em 14 de outubro de 2008.

Fowler, Martin. **Patterns of Enterprise Application Architecture**. 2004. Disponível em: http://books.google.com.br/books?id=FyWZt5DdvFkC&pg=PT187&lpg=PT187&dq=Active+Record+Martin+Fowler&source=bl&ots=eEAu_CWt4w&sig=5aIAU5GrjssSi4lUCskRA9Ofqyg&hl=pt-BR&ei=WKs5SoWCG5mMtgfJluThDA&sa=X&oi=book_result&ct=result&resnum=5#PPA1,M1. Acesso em: 16 de junho de 2009.

Gabaldi, Edgar. **Um Overview sobre o CakePhp**, 2009. Disponível em: http://www.projetofedora.org/aggregator. Acesso em: 16 de junho de 2009.

Golding, David. **THE NEWBIE'S INTRODUCTION TO CAKEPHP**. 2007.

Disponível: http://eleuln.googlepages.com/newbie-cakephp.pdf. Acesso em 20 de outubro de 2008.

Jude (Design & Modeling), 2009. Disponível em: http://jude.change-vision.com/jude-web/index.html. Acesso em 18 de maio de 2009.

Krasner, Glenn.; Pope, Stephen. **A cookbook for using the model view controller user interface paradigm in smalltalk-80**. In Journal of Object-Orientated Programming, volume 1(3). (1998).
Disponível em :
http://www.ics.uci.edu/~redmiles/ics227-SQ04/papers/KrasnerPope88.pdf.
Acesso em 15 de outubro de 2008.

Krasner, Glenn.; Pope, Stephen. **A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System.** 1998.
Disponível em: http://www.create.ucsb.edu/~stp/PostScript/mvc.pdf. Acesso em 15 de outubro de 2008.

Lopes, Salmito Yrleyjânder, **Padrões para J2EE**. 2003. Acesso em 16 de junho de 2009.

Luiz, Ronaldo Rezende Vilela. 2004. **Obtendo Qualidade de Software com o RUP**. Disponível em: http://www.javafree.org/artigo/871455/Obtendo-Qualidade-de-Software-com-o-RUP.html. Acesso em 16 de junho de 2009.

Macorrati, José Carlos. **Padrões de Projeto: O modelo MVC - Model View Controller**. 2004.
Disponível em: http://www.macoratti.net/vbn_mvc.htm. Acesso em 16 de outubro de 2008.

Manual do CakePHP. 2008.
Disponível em: http://www.manual.cakephp.com.br/doku.php. Acesso em 8 de outubro de 2008.

Marston, Tony. **The Model-View-Controller (MVC) Design Pattern for PHP**. 2004. Disponível em : http://www.tonymarston.net/php-mysql/model-view-controller.html. Acesso em 22 de outubro de 2008.

Maseto, Jonhy Maiki. Análise Avaliativa entre Frameworks Php, Unochapecó. 2006. Disponpível em: http://www2.unochapeco.edu.br/~jhony/tcc_final.pdf. Acesso em 20 de maio de 2009.

MySql, Data Base, 2008. Disponível em: http://www.mysql.com/. Acesso em 8 de outubro de 2008.

Pereira, Nice Brito Machado; Zanatta, Tuani. **Artigo sobre MVC (Model View Controller)**. 2004.
Disponível em: http://anacarol.blog.br/aulas/artigos_uteis/modelo_visualizacao_controle.pdf. Acesso em 14 de outubro de 2008.

PhpMyAdmin, 2008.
Disponível em: (http://www.phpmyadmin.net/home_page/index.php). Acesso em 8 de outubro de 2008.

APÊNDICE

APÊNDICE A - DOCUMENTAÇÃO DO PROJETO DE IMPLEMENTAÇÃO DO BD.

CakePHP segue algumas convenções em relação ao BD e nelas se enquadram usar todas as tabelas no plural; as tabelas geradas de relacionamentos n:n devem estar no plural também e em ordem alfabética, toda chave primária é do tipo id e chaves estrangeiras recebem o nome da tabela de onde é originada no singular, seguido do underline e id. Abaixo se encontra o BD projetado para o trabalho em questão e logo abaixo da estrutura das tabelas estão as restrições quanto às chaves estrangeiras de cada tabela.

Banco de Dados: `projeto`:

```
CREATE DATABASE `projeto` DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;

USE `projeto`;

Estrutura da tabela `bichos`.

CREATE TABLE IF NOT EXISTS `bichos` (
  `id` int(11) NOT NULL auto_increment,
  `tipo_id` int(11) NOT NULL,
  `nome` varchar(100) default NULL,
 `porte` varchar(20) default NULL,
  `idade` int(11) default NULL,
  `cor` varchar(20) default NULL,
`apelido` varchar(50) default NULL,
 PRIMARY KEY  (`id`),
 KEY `tipo_id` (`tipo_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;

Estrutura da tabela `bichos_padrinhos`.
```

```
CREATE TABLE IF NOT EXISTS `bichos_padrinhos` (
  `id` int(11) NOT NULL auto_increment,
  `bicho_id` int(11) NOT NULL,
  `padrinho_id` int(11) NOT NULL,
  PRIMARY KEY  (`id`),
  KEY `bicho_id` (`bicho_id`),
  KEY `padrinho_id` (`padrinho_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Estrutura da tabela `depoimentos`.

```
CREATE TABLE IF NOT EXISTS `depoimentos` (
  `id` int(11) NOT NULL auto_increment,
  `padrinho_id` int(11) NOT NULL,
  `data` date default NULL,
  `descricao` text,
  PRIMARY KEY  (`id`),
  KEY `padrinho_id` (`padrinho_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Estrutura da tabela `donativos`.

```
CREATE TABLE IF NOT EXISTS `donativos` (
  `id` int(11) NOT NULL auto_increment,
  `bicho_id` int(11) NOT NULL,
  `padrinho_id` int(11) NOT NULL,
  `valor` double NOT NULL,
  `periodo` date NOT NULL,
  PRIMARY KEY  (`id`),
```

```
  KEY `bicho_id` (`bicho_id`),
  KEY `padrinho_id` (`padrinho_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

Estrutura da tabela `fotos`.

CREATE TABLE IF NOT EXISTS `fotos` (
  `id` int(11) NOT NULL auto_increment,
  `bicho_id` int(11) NOT NULL,
  `data` date default NULL,
  `descricao` varchar(150) default NULL,
  PRIMARY KEY  (`id`),
  KEY `bicho_id` (`bicho_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

Estrutura da tabela `padrinhos`.

CREATE TABLE IF NOT EXISTS `padrinhos` (
  `id` int(11) NOT NULL auto_increment,
  `nome` varchar(200) default NULL,
  `email` varchar(150) default NULL,
  `endereco` varchar(200) default NULL,
  `cidade` varchar(100) default NULL,
  `estado` char(2) default NULL,
  `telefone` varchar(12) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

Estrutura da tabela `pedidos`.
```

```
CREATE TABLE IF NOT EXISTS `pedidos` (
  `id` int(11) NOT NULL auto_increment,
  `bicho_id` int(11) NOT NULL,
  `padrinho_id` int(11) NOT NULL,
  `total` double default NULL,
  `data` date default NULL,
  `situacao` varchar(100) default NULL,
  PRIMARY KEY  (`id`),
  KEY `bicho_id` (`bicho_id`),
  KEY `padrinho_id` (`padrinho_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

Estrutura da tabela `pedidos_prod_servs`.

CREATE TABLE IF NOT EXISTS `pedidos_prod_servs` (
  `id` int(11) NOT NULL auto_increment,
  `pedido_id` int(11) NOT NULL,
  `prod_serv_id` int(11) NOT NULL,
  `quantidade` int(11) NOT NULL,
  PRIMARY KEY  (`id`),
  KEY `pedido_id` (`pedido_id`),
  KEY `prod_serv_id` (`prod_serv_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

Estrutura da tabela `prestadoras`.

CREATE TABLE IF NOT EXISTS `prestadoras` (
  `id` int(11) NOT NULL auto_increment,
  `nome` varchar(150) default NULL,
```

```
  `email` varchar(100) default NULL,
  `endereco` varchar(150) default NULL,
  `cidade` varchar(100) default NULL,
  `telefone` int(11) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

Estrutura da tabela `prod_servs`.

CREATE TABLE IF NOT EXISTS `prod_servs` (
  `id` int(11) NOT NULL auto_increment,
  `prestadora_id` int(11) NOT NULL,
  `descricao` varchar(150) default NULL,
  `valor` double default NULL,
  `tipo` int(11) default NULL,
  PRIMARY KEY  (`id`),
  KEY `prestadora_id` (`prestadora_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

Estrutura da tabela `relatos`.

CREATE TABLE IF NOT EXISTS `relatos` (
  `id` int(11) NOT NULL auto_increment,
  `bicho_id` int(11) NOT NULL,
  `data` date default NULL,
  `descricao` text,
  PRIMARY KEY  (`id`),
  KEY `bicho_id` (`bicho_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

```
AUTO_INCREMENT=1 ;

Estrutura da tabela `tipos`.

CREATE TABLE IF NOT EXISTS `tipos` (
  `id` int(11) NOT NULL auto_increment,
  `descricao` varchar(150) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
AUTO_INCREMENT=1 ;

Estrutura da tabela `videos`.

CREATE TABLE IF NOT EXISTS `videos` (
  `id` int(11) NOT NULL auto_increment,
  `bicho_id` int(11) NOT NULL,
  `data` date default NULL,
  `descricao` varchar(200) default NULL,
  PRIMARY KEY  (`id`),
  KEY `bicho_id` (`bicho_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
AUTO_INCREMENT=1 ;

Restrições para as tabelas que possuem chaves estrangeiras:

Restrições para a tabela `bichos`.

ALTER TABLE `bichos`
  ADD CONSTRAINT `bichos_ibfk_1` FOREIGN KEY (`tipo_id`)
REFERENCES `tipos` (`id`);

    1. Restrições para a tabela `bichos_padrinhos`.

ALTER TABLE `bichos_padrinhos`
 ADD CONSTRAINT `bichos_padrinhos_ibfk_1` FOREIGN KEY (`bicho_id`)
REFERENCES `bichos` (`id`),
 ADD CONSTRAINT `bichos_padrinhos_ibfk_2` FOREIGN KEY (`padrinho_id`)
```

```
REFERENCES `padrinhos` (`id`);
```

Restrições para a tabela `depoimentos`.

```
ALTER TABLE `depoimentos`
  ADD CONSTRAINT `depoimentos_ibfk_1` FOREIGN KEY (`padrinho_id`) REFERENCES `padrinhos` (`id`);
```

Restrições para a tabela `donativos`.

```
ALTER TABLE `donativos`
  ADD CONSTRAINT `donativos_ibfk_1` FOREIGN KEY (`bicho_id`) REFERENCES `bichos` (`id`),
  ADD CONSTRAINT `donativos_ibfk_2` FOREIGN KEY (`padrinho_id`) REFERENCES `padrinhos` (`id`);
```

Restrições para a tabela `fotos`.

```
ALTER TABLE `fotos`
  ADD CONSTRAINT `fotos_ibfk_1` FOREIGN KEY (`bicho_id`) REFERENCES `bichos` (`id`);
```

Restrições para a tabela `pedidos`.

```
ALTER TABLE `pedidos`
  ADD CONSTRAINT `pedidos_ibfk_1` FOREIGN KEY (`bicho_id`) REFERENCES `bichos` (`id`),
  ADD CONSTRAINT `pedidos_ibfk_2` FOREIGN KEY (`padrinho_id`) REFERENCES `padrinhos` (`id`);
```

Restrições para a tabela `pedidos_prod_servs`.

```
ALTER TABLE `pedidos_prod_servs`
  ADD CONSTRAINT `pedidos_prod_servs_ibfk_1` FOREIGN KEY (`pedido_id`) REFERENCES `pedidos` (`id`),
  ADD CONSTRAINT `pedidos_prod_servs_ibfk_2` FOREIGN KEY (`prod_serv_id`) REFERENCES `prod_servs` (`id`);
```

Restrições para a tabela `prod_servs`.

```
ALTER TABLE `prod_servs`
  ADD CONSTRAINT `prod_servs_ibfk_1` FOREIGN KEY (`prestadora_id`) REFERENCES `prestadoras` (`id`);

Restrições para a tabela `relatos`.

ALTER TABLE `relatos`
  ADD CONSTRAINT `relatos_ibfk_1` FOREIGN KEY (`bicho_id`) REFERENCES `bichos` (`id`);

Restrições para a tabela `videos`.

ALTER TABLE `videos`
  ADD CONSTRAINT `videos_ibfk_1` FOREIGN KEY (`bicho_id`) REFERENCES `bichos` (`id`);
```