

**UNIVERSIDADE DO MINHO**

MESTRADO EM ENGENHARIA INFORMÁTICA

**ARQUITETURAS APLICACIONAIS  
SISTEMAS INTERATIVOS**



Bernardo MOTA	A77607
Cláudia CORREIA	A77431
Joana PEREIRA	A78275

1 de Julho de 2019

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Contextualização</b>	<b>2</b>
<b>3</b>	<b>Recolha de Informação</b>	<b>3</b>
<b>4</b>	<b>Definição do Produto</b>	<b>4</b>
<b>5</b>	<b>Objetivos</b>	<b>5</b>
<b>6</b>	<b>Utilizadores</b>	<b>6</b>
<b>7</b>	<b>Levantamento de Requisitos</b>	<b>8</b>
7.1	Requisitos Funcionais . . . . .	8
7.1.1	Registo . . . . .	8
7.1.2	Autenticar . . . . .	8
7.1.3	Logout . . . . .	9
7.1.4	Alterar dados dos animais . . . . .	9
7.1.5	Alterar serviços fornecidos . . . . .	9
7.1.6	Alterar tipos de animais . . . . .	9
7.1.7	Alterar horário . . . . .	9
7.1.8	Consultar Petsitters . . . . .	10
7.1.9	Consultar Dono . . . . .	10
7.1.10	Efetuar pedido . . . . .	10
7.1.11	Avaliar Petsitter . . . . .	11
7.1.12	Avaliar Dono . . . . .	11
7.1.13	Cancelar pedido . . . . .	11
7.1.14	Consultar pedidos pendentes . . . . .	11
7.1.15	Alterar dados pessoais . . . . .	12
7.1.16	Consultar perfil . . . . .	12
7.1.17	Chat . . . . .	12
7.1.18	Notificações . . . . .	12
7.1.19	Inativar utilizadores . . . . .	12
7.2	Requisitos Não Funcionais . . . . .	13
<b>8</b>	<b>Domínio do Problema</b>	<b>14</b>
8.1	Modelo de Domínio . . . . .	14
8.2	Definição das Entidades . . . . .	14
<b>9</b>	<b>Análise de funcionalidades</b>	<b>16</b>
9.1	Diagrama de Use Cases . . . . .	16
9.2	Atores . . . . .	16
9.3	Especificação dos Use Cases . . . . .	17

<b>10 Modelação de Tarefas</b>	<b>19</b>
10.1 Efetuar Pedido . . . . .	19
<b>11 Artefactos MDA</b>	<b>20</b>
11.1 PIM . . . . .	20
11.2 PSM Hibernate . . . . .	21
11.3 PSM Java EE . . . . .	23
<b>12 Prototipagem</b>	<b>23</b>
12.1 Página Inicial . . . . .	24
12.2 Opões de registo . . . . .	25
12.3 Registo de um dono e seus animais . . . . .	25
12.4 Login . . . . .	29
12.5 Página inicial de um dono autenticado . . . . .	30
12.6 Efetuar pedido . . . . .	31
<b>13 Implementação</b>	<b>32</b>
13.1 Arquitetura . . . . .	32
13.2 Tecnologias . . . . .	33
13.2.1 HTML+CSS . . . . .	33
13.2.2 Bootstrap . . . . .	33
13.2.3 JS . . . . .	33
13.2.4 Vue.js . . . . .	33
13.2.5 Apache . . . . .	33
13.2.6 JBoss . . . . .	33
13.2.7 Hibernate . . . . .	34
13.2.8 Java EE . . . . .	34
13.2.9 PostgreSQL . . . . .	34
13.3 Front-End . . . . .	34
13.3.1 Design Patterns . . . . .	34
13.3.2 Avaliação Heurística . . . . .	36
13.4 Back-End . . . . .	39
13.4.1 Servlets . . . . .	40
13.4.2 Enterprise Java Beans . . . . .	40
13.5 Acesso a Dados . . . . .	41
13.6 Web Services . . . . .	42
<b>14 Deployment</b>	<b>42</b>
14.1 Arquitetura . . . . .	42
14.2 Automatização do Deployment . . . . .	43
<b>15 Benchmarking</b>	<b>44</b>
<b>16 Conclusões e Trabalho Futuro</b>	<b>47</b>

## **Resumo**

Este documento diz respeito ao projeto proposto na especialização de Engenharia de Aplicações da Universidade do Minho, que consiste no desenvolvimento de uma aplicação que visa promover e intermediar serviços de *petsitting*.

Ao longo das secções seguintes será contextualizado o tema e apresentado todo o processo de desenvolvimento do produto, desde a fase de especificação, às fases de modelação e implementação. Por fim serão também abordados os tópicos de *deployment* e *benchmarking* da aplicação.

## 1 Introdução

O aumento do abandono de animais de estimação em Portugal é um tema que merece especial atenção. Em agosto de 2018 a Ordem dos Veterinários lançou um alerta para o aumento de animais abandonados, sendo que foi referida a subida de 22% no número de animais abandonados face ao ano anterior.

Os Canis e as Associações de animais desempenham um papel crucial no auxílio a animais abandonados, nomeadamente na captura, recolha, e restituição de animais para adoção. É exatamente por este motivo que estas instituições necessitam de todo o apoio que conseguirem da população.

Como este é um problema atual que requer a maior visibilidade possível, qualquer atitude de sensibilização no que toca ao abandono e bem-estar dos animais é importante na ajuda à attenuação do abandono dos mesmos.

Comovidos com a amplitude da situação, decidimos reunir com alguns dos responsáveis do Canil Municipal de Braga, com o objetivo de averiguar qual seria o melhor tema para a aplicação a desenvolver na especialização de Engenharia de Aplicações, de forma a contribuir positivamente para a erradicação do problema. Durante a discussão foi dado ênfase ao facto de, no momento de adoção, a maioria dos adotantes questionarem a existência de hotéis para animais na zona de Braga.

Uma das maiores causas de abandono é a falta de tempo por parte dos donos para cuidarem dos animais, ou até mesmo conviverem com eles de forma a tornar melhor a qualidade de vida tanto das pessoas como dos próprios animais. Outro dos casos que leva ao abandono é quando os donos vão viajar, ou estão ausentes durante grandes períodos de tempo, e não têm conhecimento da existência de estabelecimentos próprios ou de pessoas qualificadas que podem ficar encarregues de tratar dos animais.

Dado isto, optou-se por criar uma plataforma na qual os donos podem requisitar serviços especializados de tratamento/acolhimento temporário para os seus animais de estimação.

## 2 Contextualização

Na maioria dos casos, os animais necessitam de atenção por parte dos donos, de forma a manterem ambos uma relação saudável e uma boa qualidade de vida. No entanto, existem pessoas que adotam/compram animais de estimação sem pensarem nas consequências ou no tempo que necessitarão de despesar para cuidar dos mesmos. Quer por desinteresse ou cansaço, quer por trabalho em excesso, muitos donos abandonam os animais de estimação porque consideram que não têm disponibilidade suficiente para cuidarem dos animais. Isto acontece porque muita gente julga não ter alternativas para estes casos.

No que toca a férias, viagens de trabalho, ou até mesmo mudança de residência, existe quem pense que nestas situações a melhor (ou até mesmo a única) opção é abandonar os animais. Posto isto, torna-se fundamental alertar a sociedade para a existência de alternativas bastante viáveis, que até são mais usuais do que o que se possa imaginar.

É neste contexto que determinadas pessoas, usualmente designadas por *petsitters*, se disponibilizam a cuidar dos animais na casa dos donos, ou até mesmo na própria residência. Assim, os *petsitters* podem tanto cuidar dos animais durante longos períodos de tempo, como realizar serviços pontuais (passear um cão, alimentar um peixe, etc), de forma a poderem prestar auxílio em situações nas quais não existe possibilidade por parte do dono de cuidar dos seus animais.

Foi exatamente com o intuito de simplificar o processo de tratamento/acolhimento temporário de animais, que surgiu a aplicação TrustPet. Através desta aplicação, os donos poderão descrever os seus animais de estimação, e agendar serviços fornecidos por *petsitters*. Para além disso, de forma a que os donos possam escolher com pormenor qual a melhor opção para os seus animais, é-lhes também apresentada uma descrição detalhada dos *petsitters* e dos respetivos serviços que estes oferecem. De forma a ser possível opinar acerca tanto dos donos como dos próprios *petsitters*, os utilizadores podem efetuar *reviews* das experiências que tiveram.

### 3 Recolha de Informação

De forma a obter um conhecimento mais profundo na área em que se enquadra o projeto, antes de selecionar definitivamente o tema, decidimos reunir com os responsáveis do Canil de Braga numa conversa formal, de modo a poder fazer um primeiro levantamento de requisitos.

Após explicarmos o âmbito em que está envolvido o projeto, os responsáveis começaram por explicar o modo geral de funcionamento do Canil, e também das Associações de Animais com quem trabalham.

Salientaram que o maior problema não está na adoção dos animais, mas sim na falta de conhecimento que os donos dos animais têm acerca de assuntos relativos ao tratamento e cuidado temporário dos mesmos. Esta falta de conhecimento leva ao abandono dos animais, que são encaminhados para o canil.

Assim sendo, a conclusão retirada deste encontro, foi que é necessário facilitar a divulgação e utilização destas alternativas, que passam quer por hotéis para animais, quer por atividades de *petsitting*, por forma a diminuir o abandono de animais. Com base nisto, o projeto a desenvolver consiste numa aplicação que permite que um dono de um ou mais animais possa requisitar serviços de *petsitting*.

## 4 Definição do Produto

A aplicação em desenvolvimento tem como objetivo facilitar o processo de tratamento/acolhimento temporário de animais.

Tanto os donos como os *petsitters* deverão poder registar-se na aplicação, sendo que para isso cada um deverá especificar um conjunto de dados. No caso do dono, este deverá inserir os seus dados pessoais, os animais que possui, e as características de cada um dos animais. O *petsitter* deverá inserir também os seus dados pessoais, descrever quais os serviços que disponibiliza (alimentação, passeio, entretenimento, entre outros), indicar o preço de cada serviço, o tipo de animais que cuida e o horário em que se encontra disponível para efetuar pedidos.

O dono poderá solicitar serviços prestados pelos *petsitters*, especificando a data ou período de tempo que pretende usufruir desses serviços. Para efetuar uma escolha mais minuciosa, o dono poderá também consultar as características dos *petsitters*, nomeadamente os seus dados pessoais, quais os serviços que presta, o tipo de animais que cuida e as suas avaliações. Ao pesquisar por *petsitters*, o dono poderá procurar por nome ou endereço de e-mail, ou filtrar a informação de acordo com a localização, avaliação ou preço.

De forma a poderem descrever a experiência que tiveram, tanto os *petsitters* como os donos podem realizar *reviews*. Cada *review* diz respeito à atribuição de uma pontuação e, caso pretendam ser mais específicos, à realização de um comentário.

Uma vez que pode ocorrer algum tipo de imprevisto, ou que pode até existir um engano ao efetuar um pedido, tanto o cliente como o *petsitter* deverão conseguir cancelar os pedidos. Para saberem exatamente com o que contar, ambos os utilizadores poderão também consultar quais os pedidos que possuem pendentes.

Tal como seria de esperar, todos os dados inseridos aquando o registo, devem poder ser alterados, mais concretamente os dados pessoais, os animais de estimação no caso dos donos e os serviços e respetivos preços, horário de trabalho e tipos de animais que cuida no caso dos *petsitters*.

Por último, de maneira a que os pormenores fiquem o mais bem definidos possível, o *petsitter* e o dono devem poder comunicar através de mensagens privadas.

## 5 Objetivos

Com a utilização da aplicação TrustPet são vários os objetivos que se pretendem atingir, tanto em contexto social como para a própria aplicação em si.

Um dos principais objetivos é a divulgação do conceito de *petsitting*. Embora seja do conhecimento geral a existência de hotéis para animais, a maioria das pessoas não sabem que a prestação pontual de serviços é uma opção. Para além disto, *petsitting* é muitas vezes mais seguro e benéfico para os próprios animais, visto que não têm necessariamente de ser postos em contacto com outros animais, prevenindo assim situações desagradáveis de ansiedade e nervosismo, ou até mesmo o contágio de doenças indesejáveis. Outra das metas que se pretende alcançar é melhorar o bem-estar e qualidade de vida tanto dos donos como dos próprios animais, por exemplo entretendo os animais nos longos períodos de tempo em que estes estão sozinhos, ou permitindo ao dono ir de férias descansado enquanto os seus animais estão ao cuidado de um *petsitter*.

No que toca à utilização da aplicação em si, os objetivos são os seguintes:

- facilitar ao dono a consulta de *petsitters* na sua área;
- permitir ao *petsitter* divulgar os seus serviços de uma forma mais profissional;
- simplificar a requisição e o agendamento de serviços;
- prevenir que ocorram mal entendidos ao agendar um serviço, como usualmente pode acontecer quando os serviços são combinados informalmente entre um *petsitter* e um dono, evitando assim situações desagradáveis;
- permitir consultar de forma simples e rápida os pedidos que possuem pendentes, permitindo uma gestão mais eficiente do tempo;
- permitir ao *petsitter* conhecer antecipadamente alguns pormenores importantes acerca dos animais que irá cuidar, podendo precaver-se com o que for necessário;
- possibilitar que tanto o *petsitter* como o dono possam expressar as suas opiniões acerca das experiências que tiveram e consigam também saber a opinião de outras pessoas, para poderem tomar uma decisão mais ponderada.

## 6 Utilizadores

Os utilizadores da aplicação TrustPet podem ter o papel de donos de animais que solicitam serviços a um *petsitter*, de *petsitters* que fornecem serviços de cuidado a animais, ou de administrador do sistema.

**Nome** Dono

**Grupo etário** +18 anos

**Papel** Requisita os serviços de *petsitters* para os seus animais.

**Formação Académica** Ensino Básico

**Experiência tecnológica** Básica

**Outras características** Pode possuir vários animais

**Classe de utilizador** Direta

**Nome** *Petsitter*

**Grupo etário** +18 anos

**Papel** Fornece serviços de cuidado a animais.

**Formação Académica** Ensino Básico

**Experiência tecnológica** Básica

**Classe de utilizador** Direta

**Nome** *Administrador*

**Grupo etário** +18 anos

**Papel** Inativar utilizadores

**Formação Académica** Ensino Superior

**Experiência tecnológica** Avançada

**Classe de utilizador** Suporte

**Responsabilidade** Inativar os utilizadores que representam um risco para os restantes utilizadores (possuem baixa avaliação e comentários negativos)

## 7 Levantamento de Requisitos

Depois de analisar devidamente o domínio do problema, foram definidos os requisitos funcionais e não funcionais apresentados de seguida.

### 7.1 Requisitos Funcionais

#### 7.1.1 Registo

##### Requisito 1: Registo do utilizador

Um novo utilizador da aplicação pode registar-se como dono de animais ou *petsitter*.

##### Requisito 2: Registo do dono

No registo de um novo dono este deve introduzir os seus dados pessoais, obrigatoriamente, o nome, endereço de e-mail, data de nascimento, contacto telefónico, se tem jardim, morada, distrito, concelho e palavra-passe, e, opcionalmente, uma fotografia, e fazer, ou não, o registo dos seus animais.

##### Requisito 3: Registo do animal

No registo de um animal, o dono deve introduzir a informação do mesmo, obrigatoriamente, o nome, idade e espécie, e, opcionalmente, o porte, raça, sexo, fotografia, alergias, doenças, comportamentos estranhos, se tem as vacinas em dia, está desparasitado e se é esterilizado.

##### Requisito 4: Registo do *petsitter*

No registo de um novo *petsitter* este deve introduzir os seus dados pessoais, obrigatoriamente, o nome, endereço de e-mail, data de nascimento, contacto telefónico, se tem jardim, morada, distrito, concelho e palavra-passe, e, opcionalmente, uma fotografia. Tem também de registrar o tipo de animais que cuida, os serviços disponibilizados e o preço associado a cada um, assim como, o horário em que está disponível para realizar pedidos.

#### 7.1.2 Autenticar

##### Requisito 5: Autenticar-se na aplicação

Um utilizador previamente registrado, deve poder autenticar-se na aplicação utilizando as suas credenciais (e-mail e palavra-passe).

### 7.1.3 Logout

---

**Requisito 6: Logout da aplicação**

---

Um utilizador autenticado deve poder fazer *logout*.

### 7.1.4 Alterar dados dos animais

---

**Requisito 7: Adicionar animal**

---

Um dono deve poder registar um novo animal.

---

**Requisito 8: Remover animal**

---

Um dono deve poder remover um animal dos seus animais previamente registados.

---

**Requisito 9: Editar dados do animal**

---

Um dono deve poder editar os dados, exceto a espécie e o sexo, dos seus animais previamente registados.

### 7.1.5 Alterar serviços fornecidos

---

**Requisito 10: Alterar lista de serviços**

---

Um *petsitter* deve poder alterar os serviços de cuidado a animais que fornece, e os respetivos preços.

### 7.1.6 Alterar tipos de animais

---

**Requisito 11: Alterar tipos de animais**

---

Um *petsitter* deve poder alterar o tipo de animais que cuida.

### 7.1.7 Alterar horário

---

**Requisito 12: Alterar tipos de animais**

---

Um *petsitter* deve poder alterar o horário em que está disponível para efetuar pedidos.

### 7.1.8 Consultar Petsitters

#### **Requisito 13: Consulta geral de *petsitters***

Um dono deve ser capaz de consultar uma lista dos *petsitters* registados, com informação mais geral sobre estes, tais como o nome, a fotografia, a idade, a avaliação média, o distrito, o conselho e o número de *reviews* feitas ao mesmo.

#### **Requisito 14: Filtragem de resultados na pesquisa**

Um dono deve ser capaz de filtrar os *petsitters* através do distrito e do concelho, ordenar por avaliação média (ascendente ou descendente) e pesquisar por nome ou endereço de e-mail.

#### **Requisito 15: Consulta do *petsitter***

Um dono deve ser capaz de consultar a informação de um *petsitter* do qual tenha um pedido pendente (dados pessoais, tipos de animais que cuida e serviços fornecidos e respetivos preços).

### 7.1.9 Consultar Dono

#### **Requisito 16: Consulta do dono**

Um *petsitter* deve ser capaz de consultar a informação de um dono (dados pessoais e animais) que lhe tenha requisitado um pedido.

### 7.1.10 Efetuar pedido

#### **Requisito 17: Solicitação de serviços**

Um dono deve poder fazer um pedido a um *petsitter*, especificando quais do(s) seu(s) animal(ais) vão ser cuidados, o tipo de serviços prestados e a data de realização.

#### **Requisito 18: Consulta de *petsitters* para um pedido**

Um dono deve poder consultar a informação sobre os *petsitters* que correspondem ao pedido solicitado, como o nome, a fotografia, a idade, a avaliação média, o distrito, o conselho e o número de *reviews* feitas ao mesmo.

**Requisito 19: Filtragem de *petsitters* para um pedido**

Um dono deve poder filtrar os *petsitters* disponíveis para o seu pedido através do distrito e do concelho, ordenar por avaliação média ou por preço (ascendente ou descendente), e pesquisar por nome ou endereço de e-mail.

**Requisito 20: Conclusão de uma solicitação**

Um pedido deve ser registado após o dono escolher o *petsitter* que quer que realize o mesmo.

**7.1.11 Avaliar Petsitter****Requisito 21: Avaliação de um *petsitter***

Um dono deve poder avaliar um *petsitter*, ao qual requisitou um pedido, atribuindo uma classificação entre 0 e 5 e, opcionalmente, fazendo um comentário.

**7.1.12 Avaliar Dono****Requisito 22: Avaliação de um dono**

Um *petsitter* deve poder fazer uma avaliação a um dono que lhe tenha solicitado um pedido, atribuindo uma classificação entre 0 e 5 e, opcionalmente, fazendo um comentário.

**7.1.13 Cancelar pedido****Requisito 23: Cancelamento de pedidos**

Um utilizador deve poder cancelar um pedido pendente ao qual está associado.

**7.1.14 Consultar pedidos pendentes****Requisito 24: Consulta de pedidos pendentes**

Um utilizador deve ser capaz de consultar todos os pedidos ao qual está associado, que ainda não foram realizados.

### 7.1.15 Alterar dados pessoais

---

#### Requisito 25: Alteração de dados pessoais

---

Um utilizador, dono ou *petsitter*, deve poder alterar os seus dados pessoais, exceto o endereço de e-mail.

### 7.1.16 Consultar perfil

---

#### Requisito 26: Consulta de perfil

---

Um utilizador deve ser capaz de consultar toda a sua informação registada, incluindo as suas *reviews*.

### 7.1.17 Chat

---

#### Requisito 27: Visualizar mensagens pendentes

---

Um utilizador deve ser capaz de visualizar as mensagens de *chat* que possui pendentes.

---

#### Requisito 28: Envio de mensagens

---

Um utilizador deve ser capaz de enviar mensagens dentro de uma conversação que já tenha sido iniciada.

---

#### Requisito 29: Início de conversação por parte do dono

---

Um dono deve ser capaz de iniciar uma conversação com um *petsitter*.

---

#### Requisito 30: Início de conversação por parte do *petsitter*

---

Um *petsitter* deve ser capaz de iniciar uma conversação com um dono.

### 7.1.18 Notificações

---

#### Requisito 31: Visualizar notificações pendentes

---

Um utilizador deve ser capaz de visualizar as notificações que possui pendentes.

### 7.1.19 Inativar utilizadores

---

#### Requisito 32: Inativar utilizadores

---

O administrador deve ser capaz de inativar um dono ou um *petsitter*.

## 7.2 Requisitos Não Funcionais

### Requisito 33: Idade mínima de um utilizador

Um utilizador deve ter, pelo menos, 18 anos de idade.

### Requisito 34: Notificação do *petsitter* devido a uma solicitação de serviços

Quando um dono termina a solicitação de um pedido, o sistema deve notificar o *petsitter* correspondente.

### Requisito 35: Notificação do *petsitter* devido a um cancelamento de pedido

Quando um dono cancela um pedido, o sistema deve notificar o *petsitter* correspondente.

### Requisito 36: Notificação do dono devido a um cancelamento de pedido

Quando um *petsitter* cancela um pedido, o sistema deve notificar o dono correspondente.

### Requisito 37: Remoção de pedidos cancelados

Quando um utilizador cancela um pedido, o sistema deve eliminar esse pedido da base de dados, desaparecendo dos pedidos pendentes dos utilizadores envolvidos.

### Requisito 38: Remoção de pedidos cancelados

Quando um utilizador cancela um pedido, o sistema deve tornar esse pedido inativo, desaparecendo dos pedidos pendentes dos utilizadores envolvidos.

### Requisito 39: Remoção de um animal

Quando um dono remove um animal, o sistema deve tornar esse animal inativo, não constando no seu perfil e impossibilitando de requisitar serviços com esse animal.

### Requisito 40: Único administrador registado

A aplicação possui apenas um administrador que se encontra previamente registado na aplicação.

## 8 Domínio do Problema

De forma a simplificar a implementação do projeto optou-se por adotar uma abordagem seguida pela MDA (*Model Driven Architecture*), separando assim a funcionalidade do sistema das questões relativas ao desenvolvimento e às tecnologias utilizadas.

### 8.1 Modelo de Domínio

Tendo por base o conceito de *petsitting* foram definidas as diferentes entidades do sistema e a relação entre elas, concretizando assim um modelo independente da camada computacional CIM (*Computation Independent Model*) correspondente ao modelo de domínio. Esta etapa representa a primeira fase do ciclo de vida do processo MDA.

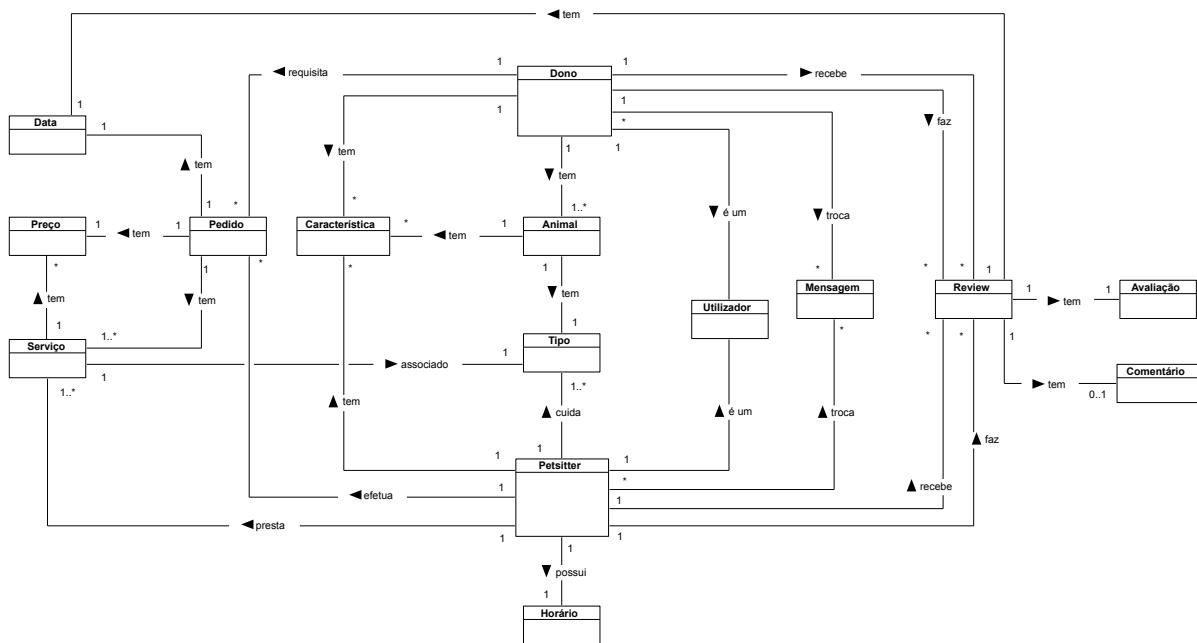


Figura 1: Modelo de Domínio.

### 8.2 Definição das Entidades

De seguida serão definidas cada uma das entidades presentes no modelo de domínio (figura 1).

**Utilizador** Dono ou *petsitter* que se registam e utilizam a aplicação.

**Dono** Pessoas que solicitam serviços a *petsitters*.

**Petsitter** Pessoas que prestam serviços de cuidado de animais.

**Animal** Animais de estimação pertencentes a um dono.

**Tipo** Tipo/espécie dos animais de estimação.

**Preço** Preço de um serviço ou preço total de um pedido.

**Serviço** Serviços fornecidos pelos *petsitters*.

**Pedido** Conjunto de serviços requisitados a um *petsitter*.

**Data** Data de agendamento de um pedido.

**Característica** Informação pessoal relativa a *petsitters*, donos ou animais.

**Review** Crítica que pode ser efetuada a um dono ou a um *petsitter*.

**Avaliação** Pontuação entre 0 e 5 atribuída numa crítica.

**Comentário** Comentário associado a uma crítica.

**Mensagem** Mensagens de texto trocadas entre donos e *petsitters*.

**Horário** Horas, relativas aos dias da semana, em que um *petsitter* está disponível para efectuar serviços.

## 9 Análise de funcionalidades

Com base nos requisitos especificados na secção 7, foram determinadas as principais funcionalidades do sistema, tendo em conta os diferentes tipos de utilizadores.

### 9.1 Diagrama de Use Cases

O diagrama de *use cases* desenvolvido para retratar as funcionalidades da aplicação foi o seguinte:

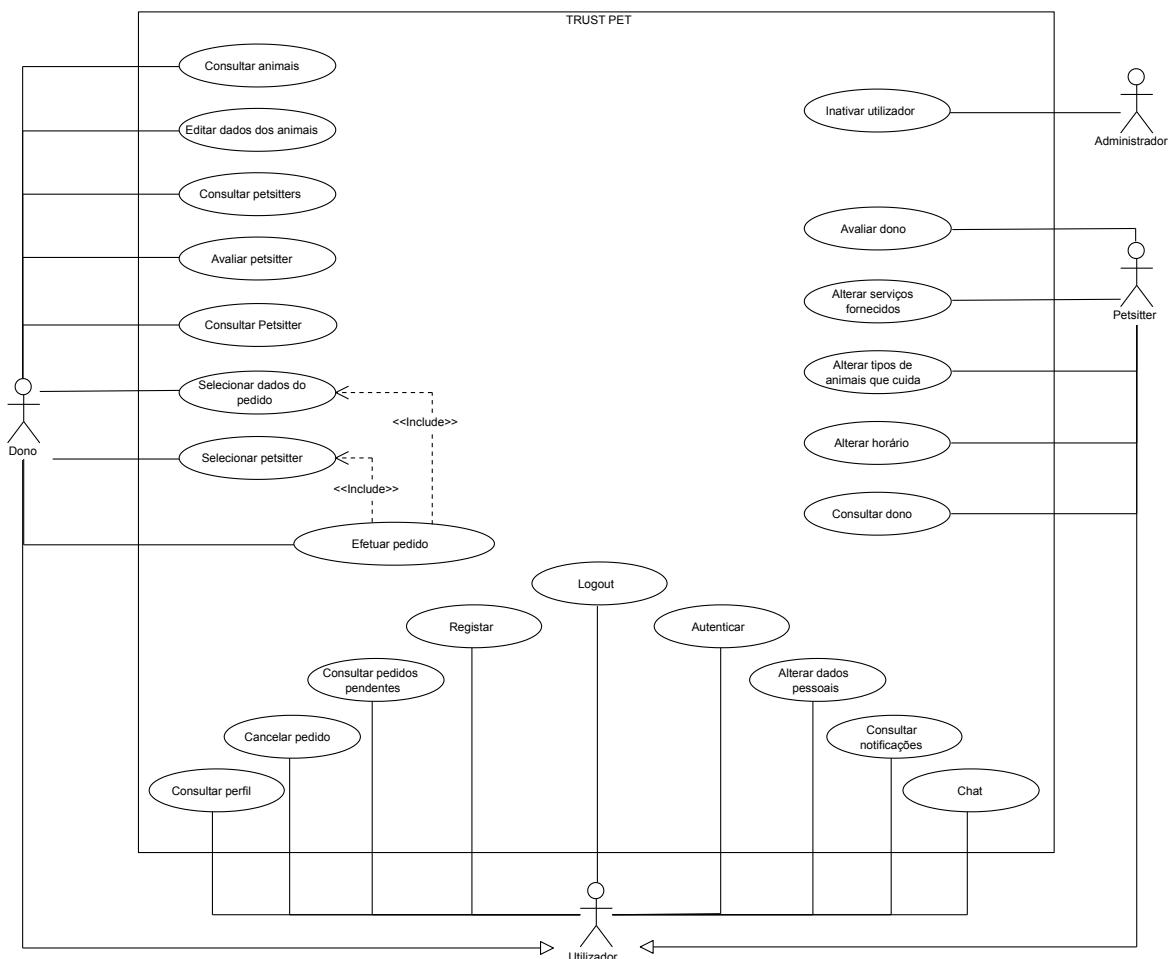


Figura 2: Diagrama de Use Cases.

### 9.2 Atores

Existem quatro tipos de atores no sistema:

**Utilizador** indivíduos que utilizam a aplicação para requisitar/publicitar serviços de *petsitting*

**Dono** indivíduo que requisita serviços de *petsitting*

**Petsitter** indivíduo que publicita serviços de *petsitting*

**Administrador** indivíduo que inativa utilizadores que representem um risco para a aplicação ou para os utilizadores da mesma

### 9.3 Especificação dos Use Cases

As especificações dos *use cases* existentes no diagrama 2 encontram-se abaixo.

**Registrar** Registar-se na aplicação

**Autenticar** Autenticar-se na aplicação

**Logout** Sair da aplicação

**Consultar perfil** Consultar os seus dados pessoais, as suas *reviews* e, caso se trate de um *petsitter*, os seus serviços, os seus tipos de animais e o seu horário

**Alterar dados pessoais** Editar os seus dados pessoais

**Consultar pedidos pendentes** Consultar detalhes relativos aos pedidos que possui pendentes

**Cancelar pedido** Cancelar algum pedido que possua pendente

**Consultar notificações** Consultar as suas notificações

**Chat** Iniciar uma conversa com algum *petsitter*

**Consultar Animais** Consultar os dados dos seus animais

**Editar dados dos animais** Editar os dados de algum animal

**Consultar petsitters** Consultar os *petsitters* registados na aplicação

**Consultar petsitter** Consultar o perfil de um *petsitter*

**Avaliar petsitter** Realizar uma *review* a um *petsitter*

**Efetuar pedido** Efetuar um pedido a um *petsitter*

**Selecionar dados do pedido** Selecionar a data, hora, e os serviços pretendidos para cada animal

**Selecionar petsitter** Selecionar o *petsitter* que irá efetuar o pedido

**Avaliar dono** Realizar uma *review* a um dono

**Alterar serviços fornecidos** Editar os serviços que fornece

**Alterar tipos de animais que cuida** Editar os tipos de animais que cuida

**Alterar horário** Alterar o horário a que está disponível para prestar serviços

**Consultar dono** Consultar o perfil de um dono

**Inativar utilizador** Tornar um utilizador inativo, para que não possa aceder mais à aplicação

## 10 Modelação de Tarefas

Uma vez feita uma análise focada no sistema e na forma como este é utilizado, decidiu-se também estudar o esforço do utilizador ao realizar determinadas tarefas.

### 10.1 Efetuar Pedido

A tarefa de efetuar um pedido, realizada pelo dono, é uma das principais e mais relevantes funcionalidades da aplicação. Desta forma, foi desenvolvido um diagrama HTA relativo a esta funcionalidade, com o objetivo de compreender melhor como o utilizador vai executar esta tarefa e como vai interagir com o sistema.

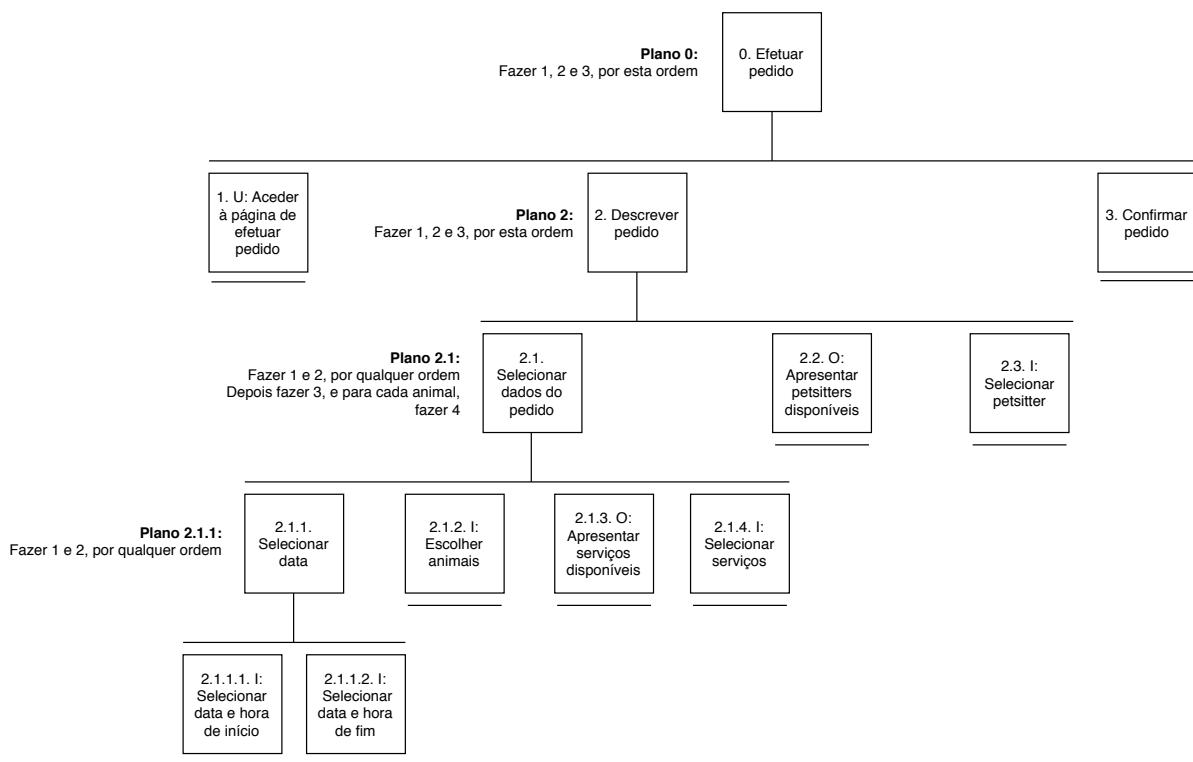


Figura 3: *Hierarchical Task Analysis* em diagrama da ação *efetuar pedido*.

Ao analisar o diagrama acima reparamos que, embora a tarefa em si seja algo relativamente complexo, a forma como o utilizador irá realizá-la é bastante simples visto que o sistema encarrega-se de apresentar a informação necessária, e o utilizador limita-se a selecionar os dados que pretende. Isto demonstra que, de facto, o esforço do utilizador é mínimo, indo de encontro ao que se pretende.

## 11 Artefactos MDA

Depois da concretização do CIM (figura 1), as fases seguintes do processo MDA são a criação de um PIM (*Platform Independent Model*), sendo este um modelo que descreve o núcleo dos componentes e serviços oferecidos pela lógica de negócio, e a criação dos PSMs (*Platform Specific Models*) relativos às tecnologias utilizadas.

### 11.1 PIM

Analizando os requisitos especificados na secção 7, obteve-se o PIM (*Platform Independent Model*) apresentado abaixo.

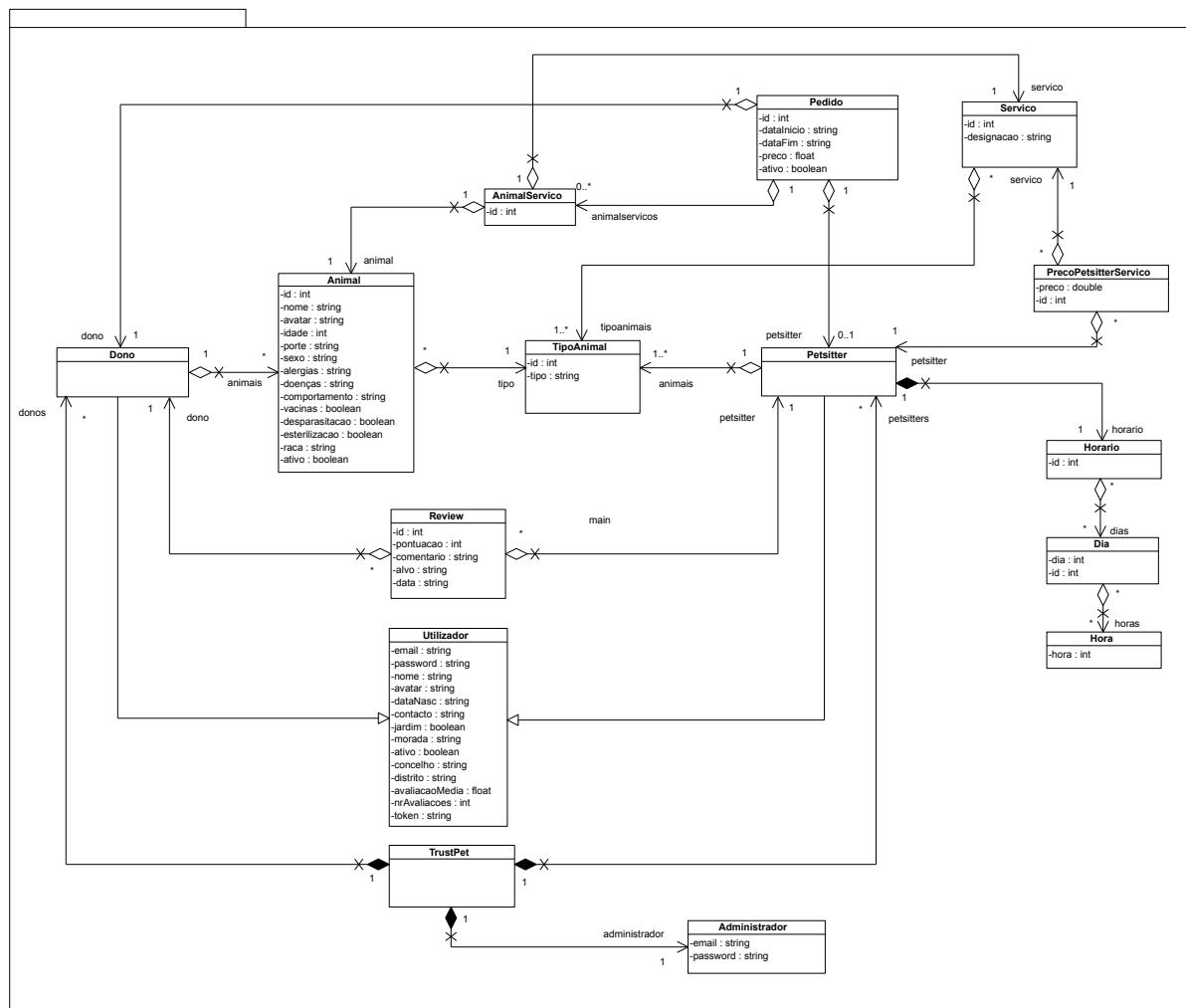


Figura 4: Diagrama de classes correspondente ao PIM.

## 11.2 PSM Hibernate

Tendo por base o modelo independente (figura 4), criou-se o PSM específico do Hibernate, especificando as entidades a persistir.

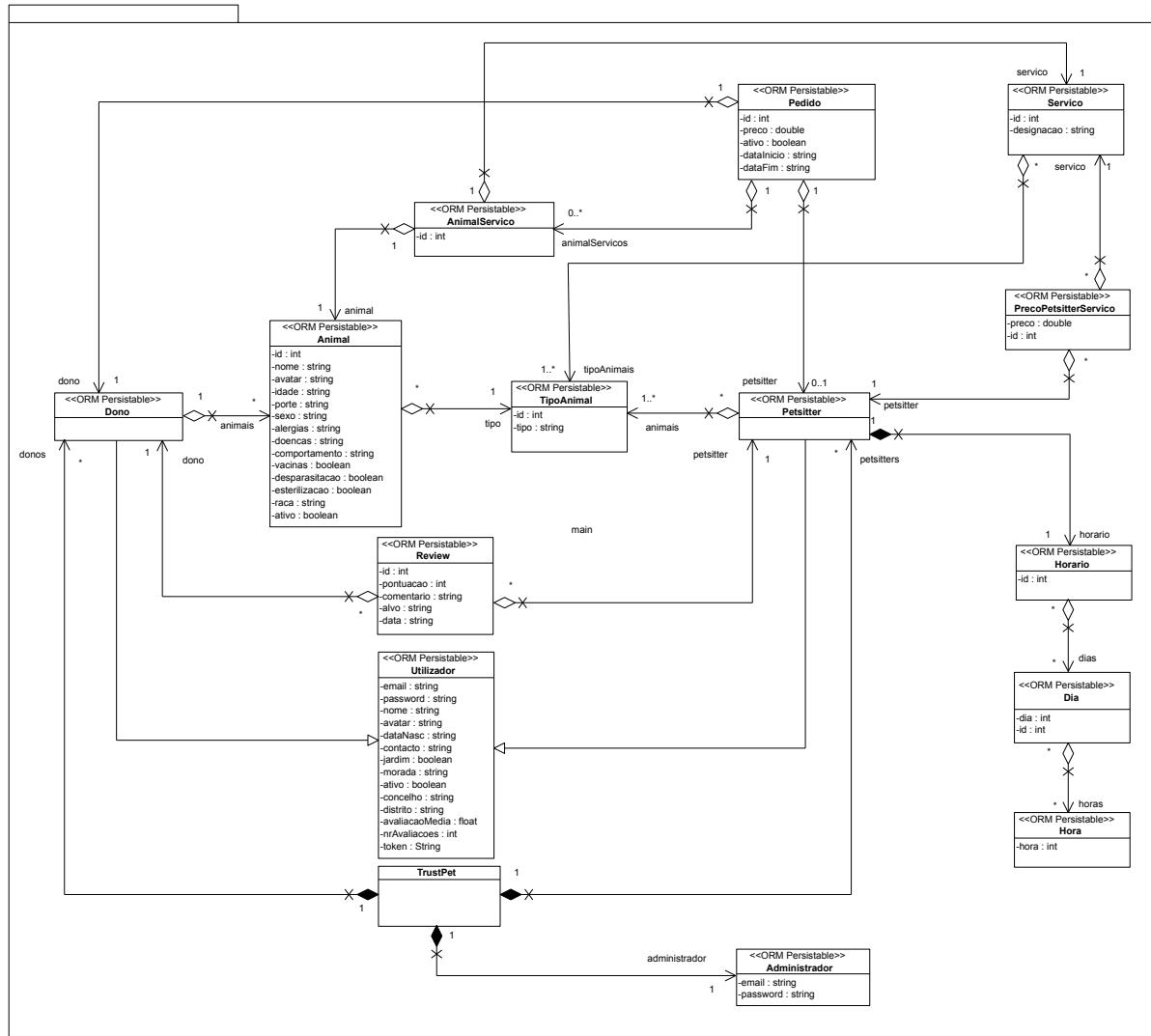


Figura 5: Diagrama de classes correspondente ao PSM específico do Hibernate.

Para gerar o código relativo à persistência foi efetuado um processo de *Model Driven Development Persistency*. Para este efeito utilizou-se a ferramenta Visual Paradigm, que gerou o modelo lógico da base de dados apresentado abaixo.

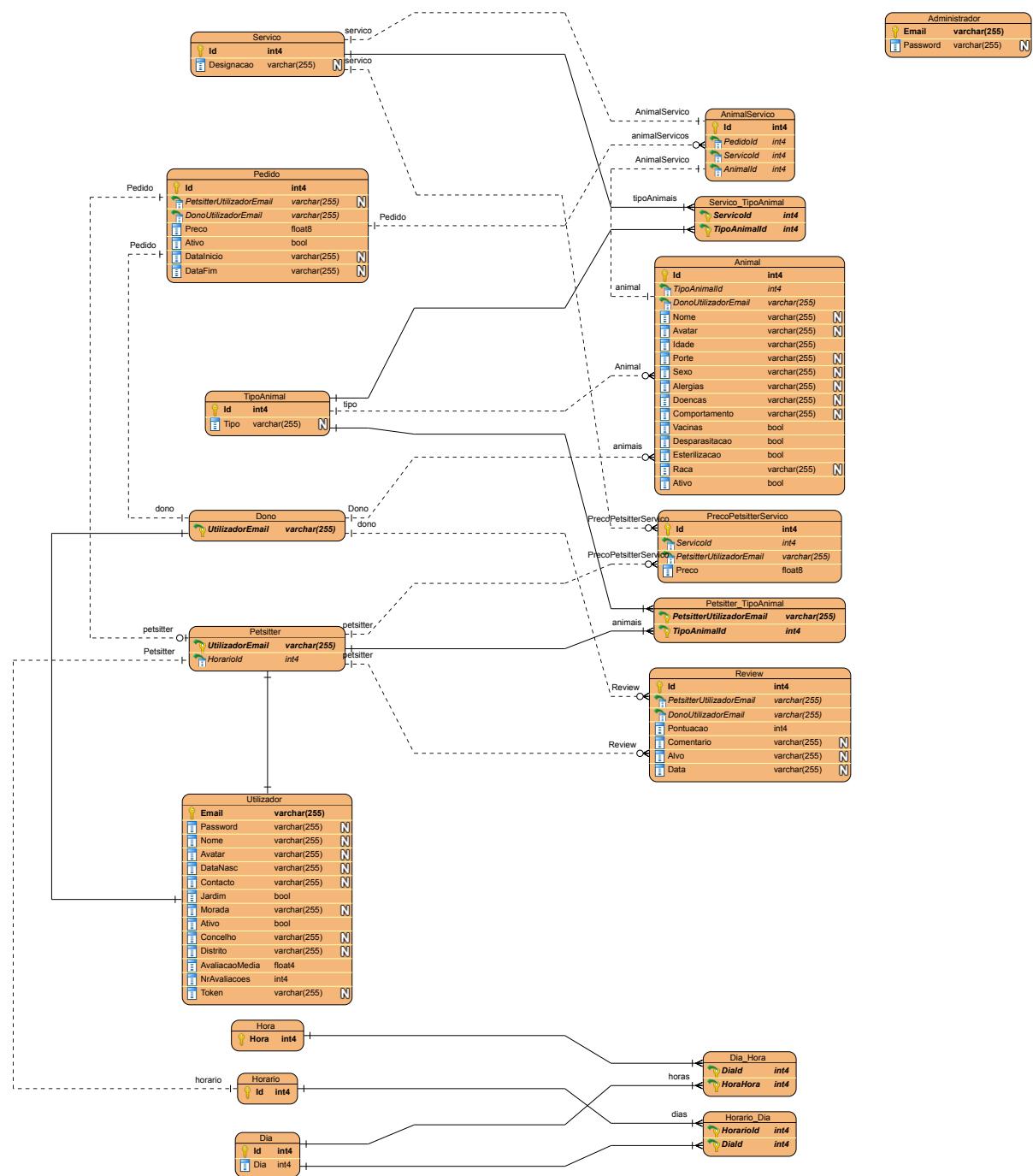


Figura 6: Modelo lógico gerado pelo Visual Paradigm.

É importante mencionar também que o mapeamento foi efetuado através de ficheiros XML, gerados também a partir do Visual Paradigm. Consideramos que esta seria uma melhor abordagem visto que mantém o código limpo e torna-o mais facilmente reutilizável.

### 11.3 PSM Java EE

Tendo por base o modelo independente (figura 4), criou-se o PSM específico do *Java Enterprise Edition* (Java EE), especificando os *Servlets*, os *Beans* e os *Facades* a implementar.

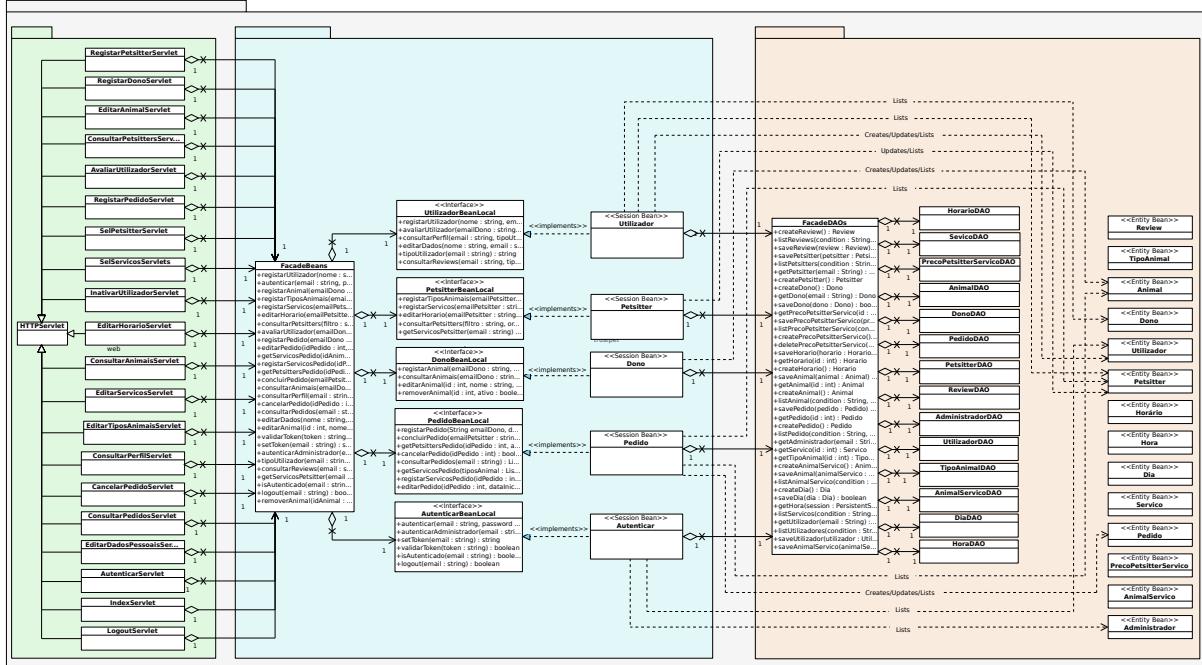


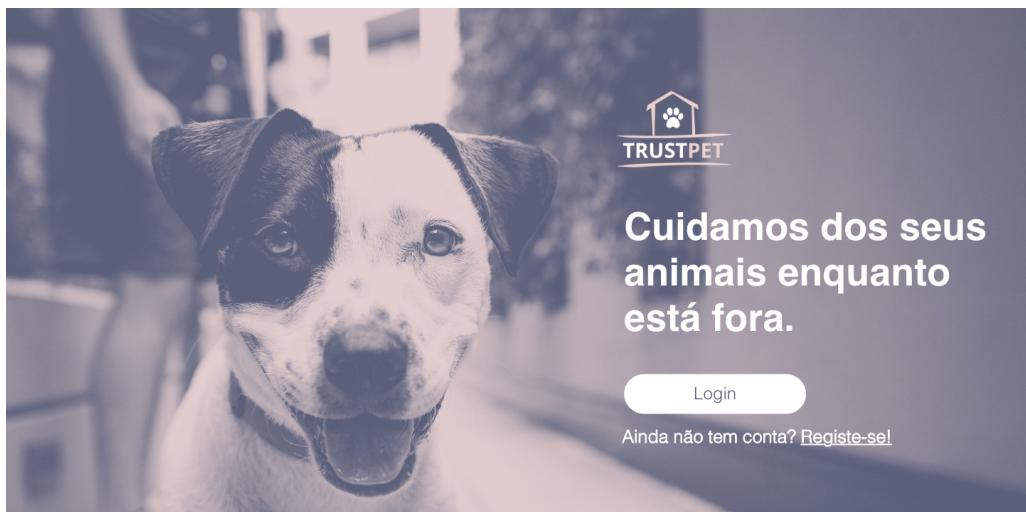
Figura 7: Diagrama de classes correspondente ao PSM específico do Java EE.

## 12 Prototipagem

A fase de prototipagem da UI da aplicação TrustPet foi inicializada através da criação de protótipos de baixa fidelidade em papel. Foi tomada esta iniciativa, porque, como se tratava de um primeiro esboço da interface, este serviria apenas de ponto de partida. Para além disso, este método não apresenta custos, e encoraja contribuições e o surgimento de novas ideias, permitindo avançar para outro tipo de protótipos com uma ideia da interface melhor definida.

De seguida, recorrendo à ferramenta de prototipagem Adobe XD, desenvolvemos os *mockups* de algumas interfaces e definimos o comportamento das mesmas. Esta ferramenta permitiu-nos criar as diferentes páginas *web* com interação por clique.

## 12.1 Página Inicial



**Opções destinadas às suas necessidades.**

<b>Passeio</b>  Passeamos o seu animal, limpamos dejetos e asseguramos a hidratação do mesmo.	<b>Petsitting em sua casa</b>  Tomamos conta do seu animal em sua casa, assegurando conforto, alimentação, hidratação, carinho e brincadeiras.	<b>Petsitting em casa do petsitter</b>  Tomamos conta do seu animal, assegurando conforto, alimentação, hidratação, carinho e brincadeiras.
---	--	---



Figura 8: Protótipo da página inicial.

## 12.2 Opções de registo

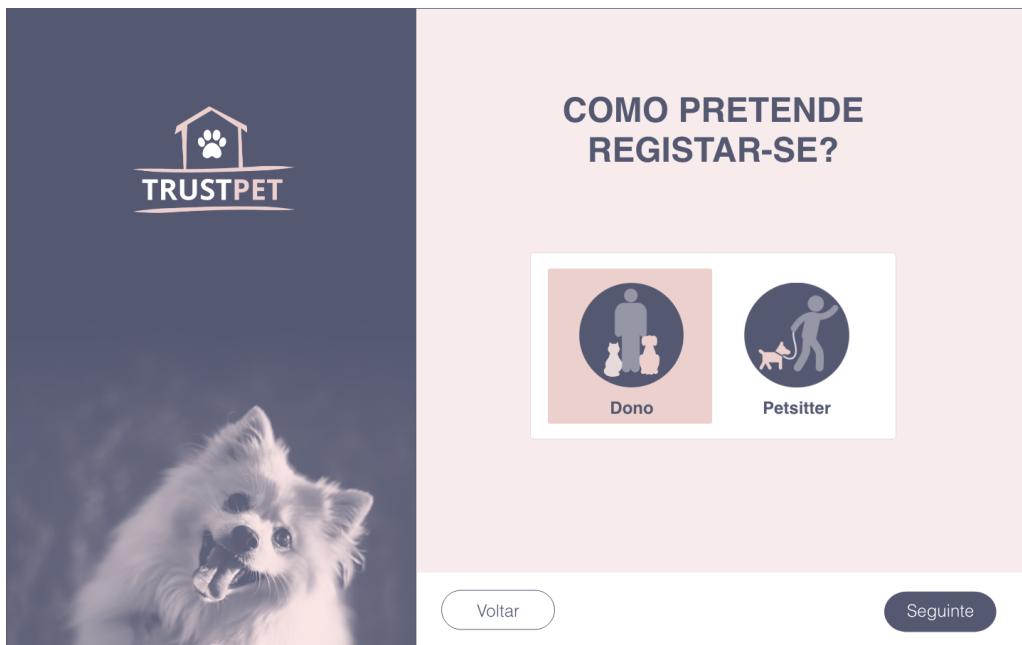


Figura 9: Protótipo da página de seleção do tipo de registo.

## 12.3 Registo de um dono e seus animais

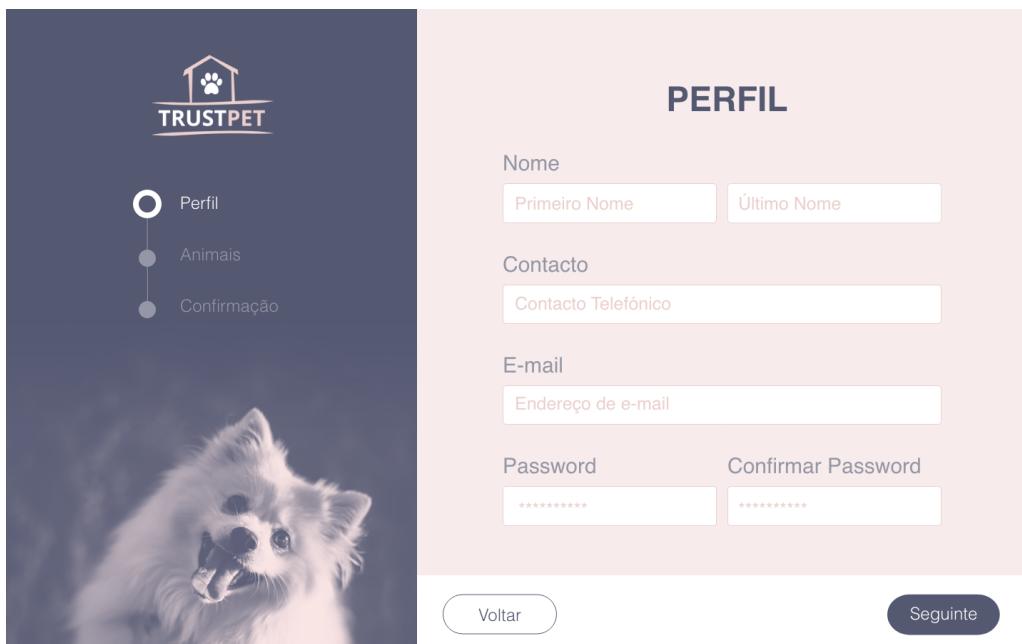


Figura 10: Protótipo da página de registo de dados pessoais de um dono.



Figura 11: Protótipo da página de registo de dados pessoais de um dono.



Figura 12: Protótipo da página de seleção do tipo de animal do dono.

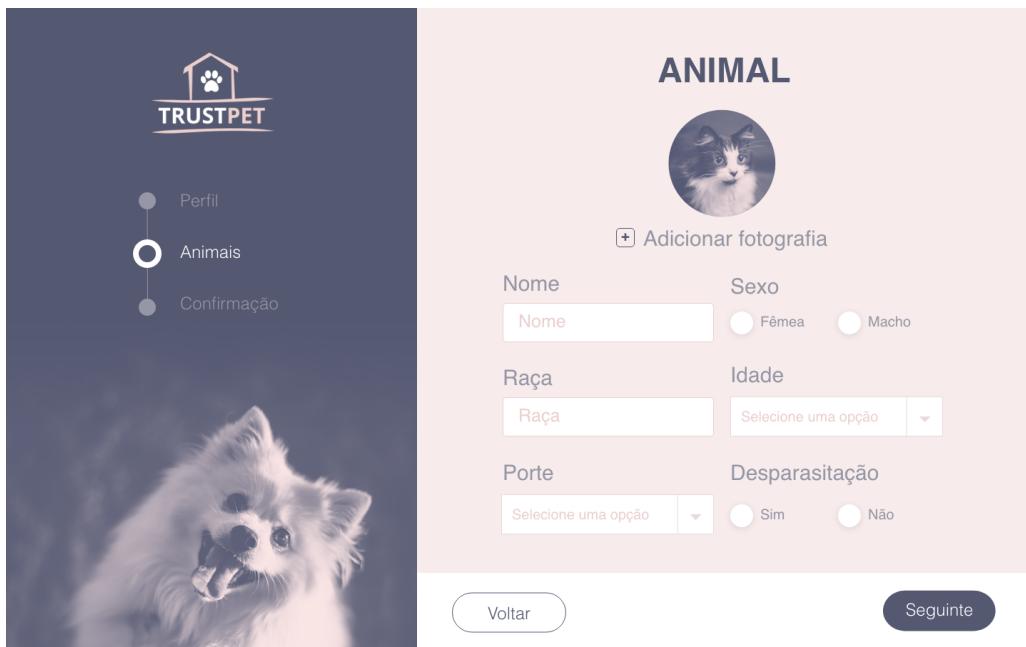


Figura 13: Protótipo da página de registo de um animal.



Figura 14: Protótipo da página de registo de um animal.

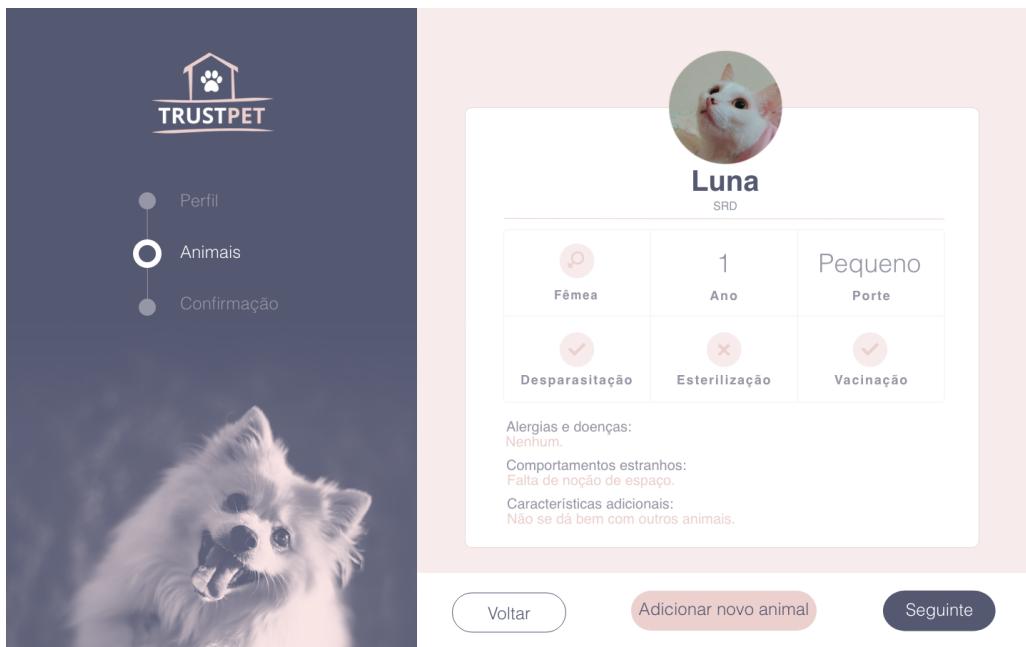


Figura 15: Protótipo da página de confirmação dos dados de um animal.

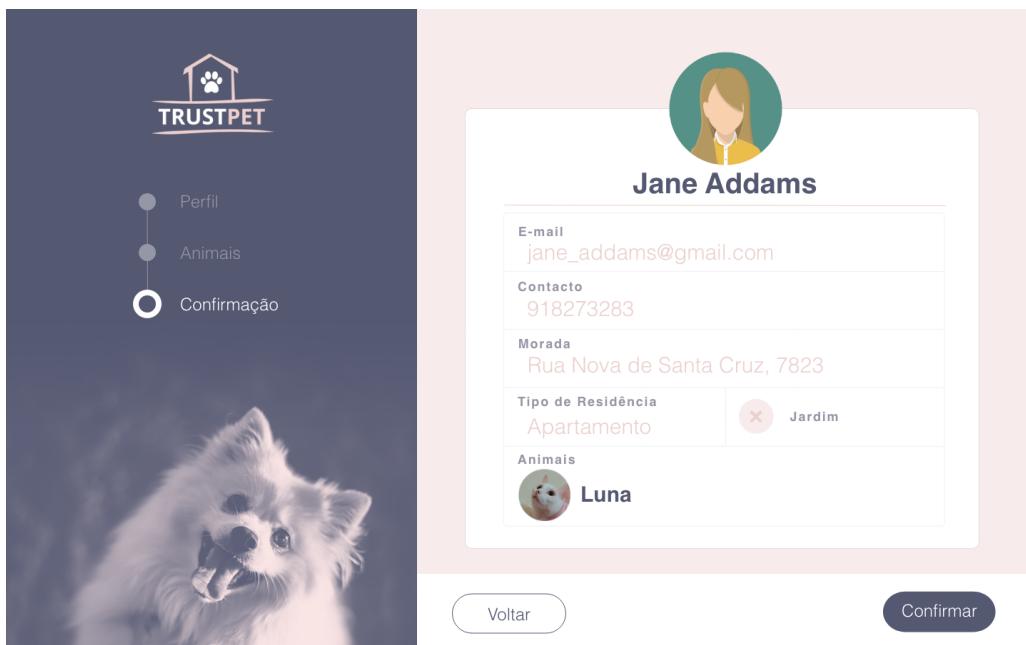


Figura 16: Protótipo da página de confirmação dos dados de um dono.

## 12.4 Login

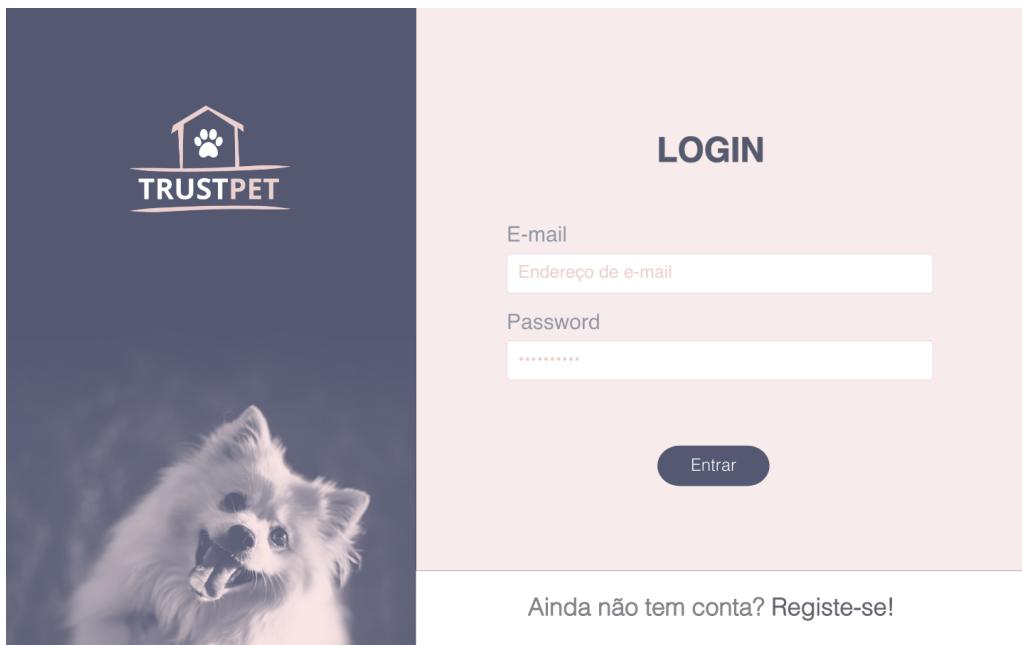


Figura 17: Protótipo da página de *login* na aplicação.

## 12.5 Página inicial de um dono autenticado



Figura 18: Protótipo da página inicial de um dono autenticado.

## 12.6 Efetuar pedido

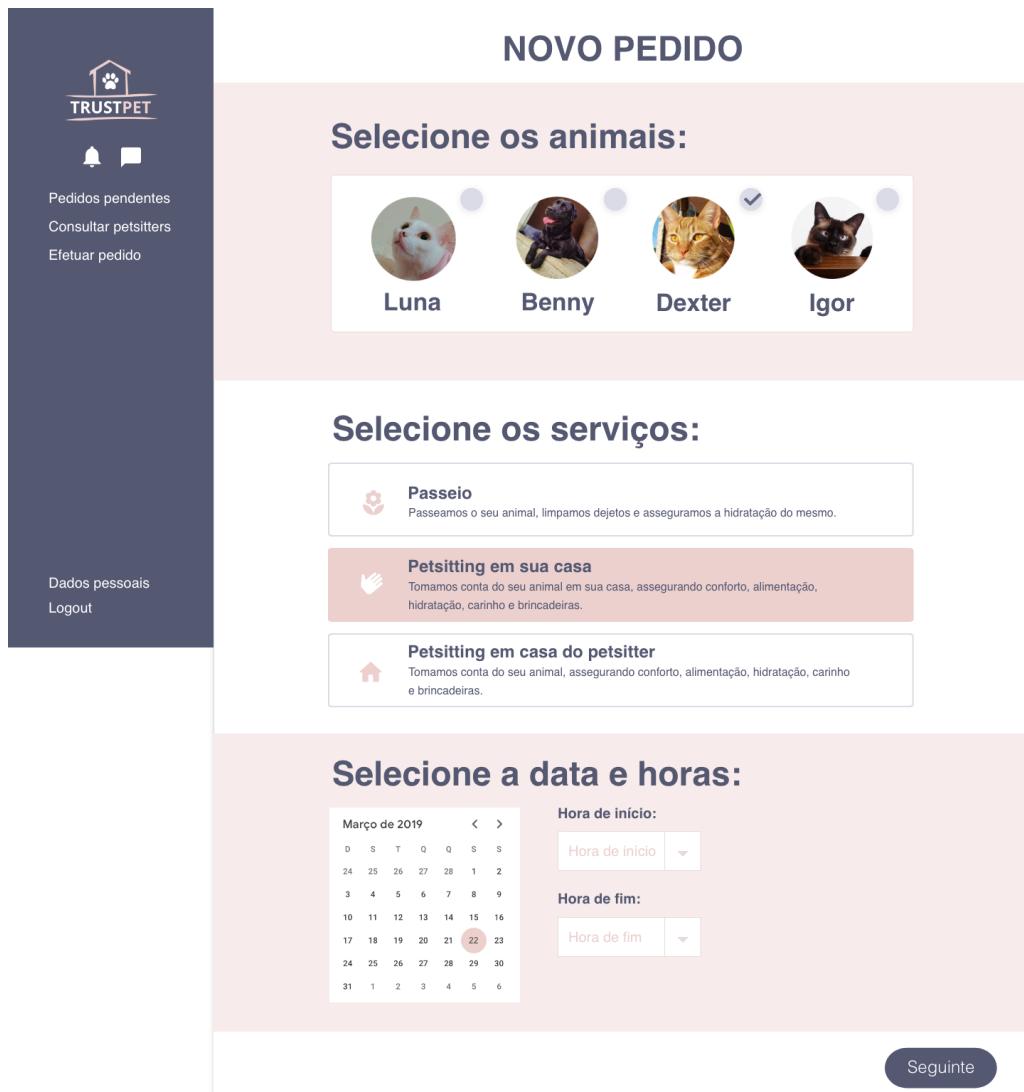


Figura 19: Protótipo da página de requisição de um pedido.

## 13 Implementação

Nas secções abaixo será descrito todo o processo de implementação do projeto, assim como as decisões tomadas ao longo do mesmo.

### 13.1 Arquitetura

No desenvolvimento da aplicação utilizou-se uma arquitetura multicamada, tal como a apresentada na figura abaixo.

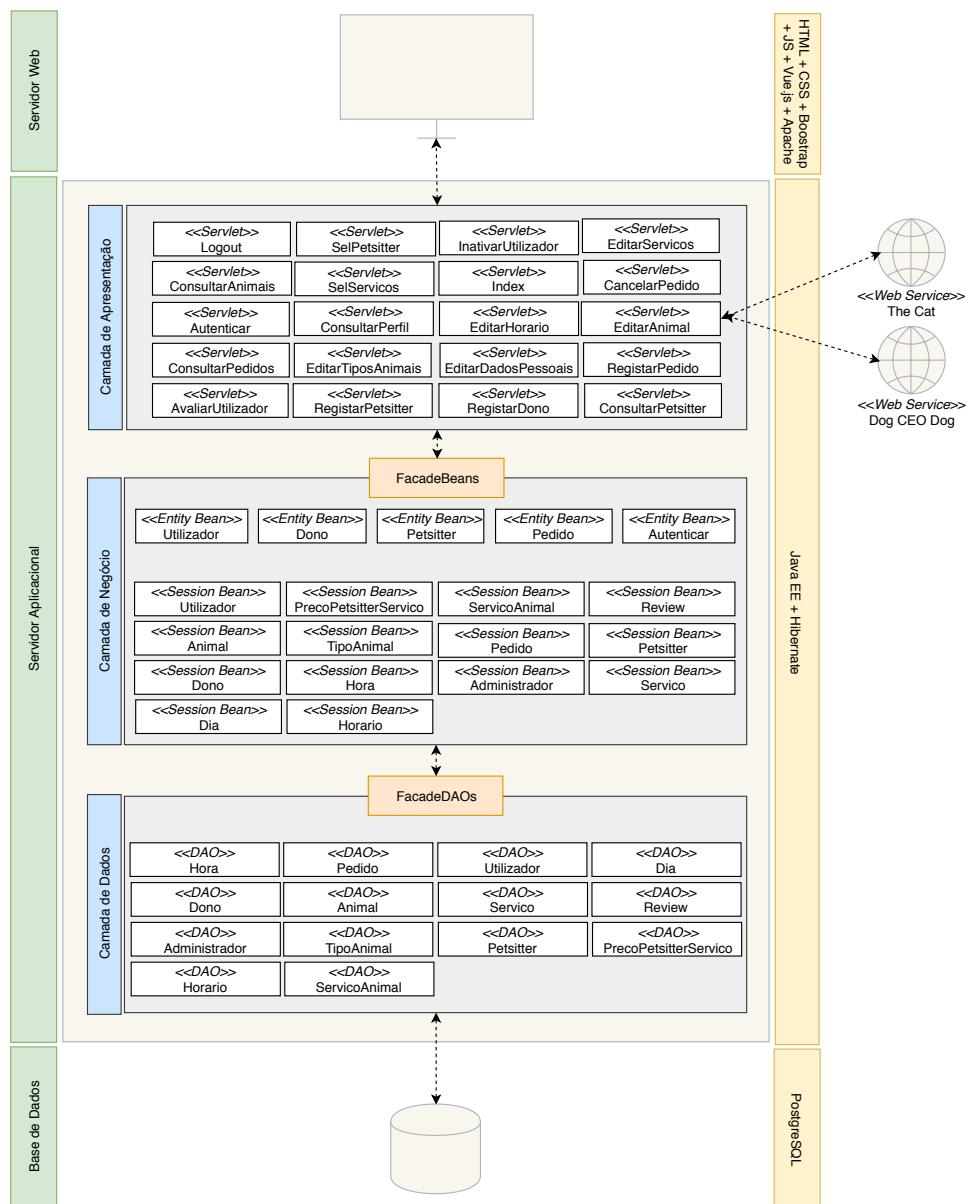


Figura 20: Arquitetura da aplicação TrustPet com especificação das tecnologias utilizadas.

## 13.2 Tecnologias

Tal como consta na figura 20, foram utilizadas diversas tecnologias em cada componente da aplicação. As razões que levaram à seleção de cada uma das tecnologias especificadas serão apresentadas de seguida.

### 13.2.1 HTML+CSS

Para a construção das páginas *web* utilizou-se a HTML e CSS. A escolha do HTML recaiu no facto de esta ser a linguagem *standard* para o desenvolvimento de páginas *web*. Já CSS foi utilizado para descrever o estilo das páginas HTML e a forma como os seus elementos são apresentados ao utilizador.

### 13.2.2 Bootstrap

De forma a conseguir desenvolver uma interface responsiva, recorreu-se a Bootstrap. Esta ferramenta simplifica bastante a utilização de determinados componentes, que caso contrário seriam mais complicados de implementar.

### 13.2.3 JS

Por forma a responder a eventos que ocorrem na página, recorreu-se a *scripts* em JavaScript, dado que esta é a principal tecnologia utilizada atualmente para este efeito. Foi também utilizado JQuery para verificar o *input*, por exemplo, na validação de formulários.

### 13.2.4 Vue.js

Para além das ferramentas já descritas, utilizou-se também o Vue.js para definir métodos, variáveis e componentes que são utilizados nas diferentes páginas *web*. O principal motivo desta escolha foi o facto de Vue.js possibilitar a criação de páginas interativas.

### 13.2.5 Apache

Como *web server* foi utilizado o Apache HTTP Server para servir conteúdo estático e para fazer o balanceamento de carga entre os servidores aplicacionais. Optamos por esta tecnologia visto ser bastante utilizada pela comunidade e ter muita documentação, o que facilitou a sua utilização.

### 13.2.6 JBoss

Como servidor aplicacional recorreu-se ao JBoss uma vez que fornece uma *stack* completa de Java Enterprise Edition (JEE), incluindo EJBs, e, para além disto contém um *servlet container*.

### 13.2.7 Hibernate

O Hibernate foi a *framework* eleita para fazer o mapeamento objeto-relacional. Para além de ser uma das *frameworks* mais utilizadas para Java, é suportada pelo Visual Paradigm, o que facilita bastante a sua utilização. Outra vantagem do Hibernate que foi tida em conta, foi o facto de este poder ser integrado com qualquer base de dados.

### 13.2.8 Java EE

Da plataforma JEE foram utilizados Enterprise Java Beans (EJB) e *Servlets*. Recorreu-se a EJB na camada aplicacional sobretudo devido à sua elevada portabilidade, uma vez que, tendo sido desenvolvidos podem ser implementados em qualquer servidor compatível com EJB. Posto isto, um grande benefício desta tecnologia é que ao poder aceder a *beans* remotamente, é possível escalar a aplicação de uma forma muito mais eficiente, colocando num servidor remoto os *beans* mais pesados.

### 13.2.9 PostgreSQL

O sistema de gestão de base de dados utilizado foi o PostgreSQL. Os principais motivos que levaram a esta decisão foram o facto de esta ser uma tecnologia gratuita e possuir uma comunidade ativa, existindo assim imenso apoio e documentação auxiliar.

## 13.3 Front-End

Para o desenvolvimento do *front-end* foram utilizadas várias tecnologias, nomeadamente, HTML, CSS, JS, Vue.js e Bootstrap. A utilização conjunta destas ferramentas permitiu a criação das interfaces responsivas e com o seu conteúdo renderizado do lado do cliente.

### 13.3.1 Design Patterns

Aquando o desenvolvimento da interface da aplicação, foram tidos em conta os diversos *design patterns* existentes, com o objetivo de reutilizar as boas práticas de *design* já definidas. Para além disso, deixa os utilizadores familiarizados com a aplicação, pois são elementos com que já se depararam em outras interfaces, facilitando a utilização da plataforma por parte dos mesmos.

Os *design patterns* que considerámos são os que se encontram em *Patterns in Interaction Design*<sup>1</sup>, e aqueles que considerámos mais relevantes de utilizar no contexto da aplicação são os apresentados de seguida.

**Registration** Considerando que os utilizadores necessitam de fornecer os seus dados para usufruir das funcionalidades da aplicação, é necessário que estes se registem de forma a armazenarem os seus dados, podendo ser usados sempre que necessário.

---

<sup>1</sup><http://www.welie.com/index.php>

**Form** Tendo em conta a grande quantidade de dados que os utilizadores necessitam de registar, foram utilizados formulários de forma a que estes os possam preencher e enviar para o servidor.

**Input Error Message** Quando o utilizador utiliza a aplicação, pode deparar-se com erros, quer seja de incorreções de *input* de dados, quer seja um erro que ocorra na aplicação. Desta forma, é necessário informar os utilizadores da ocorrência desses erros e de como os poderá corrigir.

**Login** O utilizador deve autenticar-se na aplicação para poder tirar partido das suas funcionalidades. Tendo em conta que estas necessitam dos dados do utilizador, é estritamente necessário o seu registo e *login*.

**Headerless Menu** Quando um utilizador tem acesso às suas funcionalidades na plataforma, deve ter disponível um local onde pode consultar e aceder a essas mesmas funcionalidades. Posto isto, decidimos implementar duas barras laterais, uma para os utilizadores do tipo dono e outra para os utilizadores do tipo *petsitter*, implementando o *pattern Headerless Menu*. Estas encontram-se divididas em secções, sendo que a primeira apresenta as funcionalidades mais relevantes na aplicação para esse utilizador. Como estas ainda são várias, foi necessário implementar este menu verticalmente.

**Home Link** Quando os utilizadores se autenticam na aplicação, são levados para a sua página principal. A partir daí podem aceder a outras páginas da aplicação. Como os utilizadores podem querer voltar a aceder à página principal em qualquer momento, é importante mantê-la disponível em todas as páginas da aplicação. Por esta razão, foi incluído o logótipo da aplicação nas barras laterais do utilizador, que possui uma ligação para a página principal.

**Retractable Menu** Tendo em conta a possibilidade de os utilizadores acederem à aplicação em diferentes ecrãs, é necessário considerar essas mudanças durante o processo de desenvolvimento das interfaces. Por essa razão, a barra lateral dos utilizadores pode ser ativada ou desativada, sendo possível utilizar mais espaço do ecrã para mostrar o conteúdo da página.

**Action Button** Considerando a importância de algumas funcionalidades da aplicação, recorremos também ao *pattern action button* para destacar essas ações. Por exemplo, na ação de um dono efetuar um pedido a um *petsitter* foi implementado um botão para chamar mais atenção para esta tarefa.

**Stepping** Existem determinadas funcionalidades na aplicação que necessitam de passar por diversos passos para serem executadas. É o caso do registo de um *petsitter* por exemplo, que tem de primeiro fazer o registo dos seus dados pessoais, depois indicar o tipo de animais que cuida, os serviços fornecidos e o horário. Tendo em conta o número de passos que devem ser seguidos, é importante mostrar esta informação ao utilizador.

**Search Box** Considerando que existem vários *petsitters* na aplicação, deve ser possível que os donos pequisem por estes. Desta forma, recorremos a este *pattern* para implementar esta funcionalidade.

**Rating** Tendo em conta o facto de a aplicação ter por base o fornecimento de serviços de *petsitting*, é necessário garantir que o cliente recebe o melhor serviço prestado possível, e que o prestador consiga também realizar o melhor serviço possível. Desta forma, foi também implementado um serviço de classificações que podem ser consultadas tanto pelos donos, como pelos *petsitters*.

**Comment Box** Para além da classificação, os utilizadores podem também deixar comentários sobre outros utilizadores aos quais requisitaram ou forneceram serviços, informando outros sobre aspetos que gostaram, ou não.

**Constraint Input** Em determinadas ações, é necessário que o *input* dado pelo utilizador siga um determinado formato. Foi então seguido este *pattern* para garantir que os dados inseridos estivessem corretos.

### 13.3.2 Avaliação Heurística

Por forma a avaliar as interfaces desenvolvidas, decidimos realizar uma avaliação heurística, seguindo as Heurísticas de Nielsen. Para cada uma destas, vamos apresentar um exemplo em que foi considerada.

**Visibilidade do estado do sistema** O exemplo considerado mostra o sistema a informar o utilizador que fez *login* na aplicação e que está, portanto, autenticado.



**Compatibilidade entre o sistema e o mundo real** A linguagem utilizada em toda a aplicação tenta seguir aquela que é utilizada pelos utilizadores.

**Controlo e liberdade para o utilizador** Quando o utilizador está a efetuar o registo dos seus dados pessoais, pode voltar para a página onde estava, ou continuar, ou seja, as saídas estão bem assinaladas.

A screenshot of a web-based registration form. The form is divided into two columns. The left column contains a field labeled "Distrito" with the placeholder "Introduza o seu distrito" and a checkbox labeled "Tem Jardim?". The right column contains a field labeled "Concelho" with the placeholder "Introduza o seu concelho". At the bottom of each column are two buttons: "Voltar" (Back) on the left and "Seguinte" (Next) on the right. The overall design is clean and modern, using a light gray color scheme.

**Consistência e Padronização** O esquema utilizado é o mesmo em todas as interfaces, e são usados os mesmos componentes para representar o mesmo tipo de conteúdos. Por exemplo, a informação do perfil de um dono é mostrada da mesma forma para um *petsitter*, ou seja, quando cada um acede ao perfil do outro já está familiarizado com o esquema.

**Prevenção de erros** Um utilizador está impedido de fazer o seu registo se introduzir dados incorretos ou não preencher todos os campos obrigatórios.

The screenshot shows a registration form with several fields and validation messages:

- Nome:** Placeholder "Introduza o seu nome". Error message: "Preencha este campo."
- E-mail:** Placeholder "Introduza o seu endereço de e-mail". Error message: "Introduza um e-mail válido."
- Password:** Placeholder "Introduza uma password". Error message: "Introduza uma password com no mínimo 5 caracteres."
- Data de nascimento:** Placeholder "dd/mm/aaaa". Error message: "Deve ter no mínimo 18 anos para se poder registar."
- Contacto:** Placeholder "Introduza o seu contacto telefónico". Error message: "Introduza um contacto válido."
- Fotografia:** Placeholder "Introduza o URL da fotografia". Error message: "Preencha este campo."
- Morada:** Placeholder "Introduza a sua morada". Error message: "Preencha este campo."
- Distrito:** Placeholder "Introduza o seu distrito". Error message: "Preencha este campo."
- Concelho:** Placeholder "Introduza o seu concelho". Error message: "Preencha o formulário corretamente."
- Tem Jardim?**: A checkbox field.

**Reconhecimento em vez de memorização** Um utilizador não necessita de se lembrar, por exemplo, do tipo de conta que escolheu anteriormente quando quer fazer o seu registo. É-lhe indicado que tipo de conta está a criar quando preenche o formulário.

A screenshot of a mobile application form for creating a new pet sitter account. The title "NOVA CONTA PETSITTER" is at the top. Below it is a navigation bar with tabs: "Dados Pessoais" (selected), "Tipos de Animais", "Serviços Fornecidos", and "Horário". The "Nome" (Name) field is labeled and contains the placeholder "Introduza o seu nome" (Please enter your name).

**Eficiência e flexibilidade de utilização** Quando um utilizador se encontra no formulário de *login*, pode utilizar a tecla *Enter* para acionar o botão *Entrar*.

A screenshot of a mobile application login form. It features two input fields: "E-mail" (Email) with the placeholder "Endereço de e-mail" (Email address) and "Password" (Password) with the placeholder "\*\*\*\*\*". Below the password field is a "Entrar" (Enter) button.

**Estética e design minimalista** Em geral, todas as interfaces desenvolvidas apresentam apenas o conteúdo necessário, tentando este ser sempre apresentado da forma mais simples possível. No exemplo seguinte, o *petsitter*, para editar o tipo de animais que cuida,

necessita apenas de clicar nas imagens, para as selecionar ou não, sendo que estas ficam mais transparentes caso não estejam selecionadas, ou mais opacas caso contrário. Para além disto, é apenas apresentada o nome e uma imagem de forma a simplificar o conteúdo.



**Ajuda no reconhecimento diagnóstico e recuperação de erros** Um dono só pode efetuar um pedido, caso tenha animais registados. Assim, se este não possuir nenhum animal e tentar efetuar um pedido, é-lhe apresentada uma mensagem de erro e apresentada uma solução para corrigir o problema, neste caso, o registo de um animal.



### 13.4 Back-End

De acordo com a arquitetura de uma aplicação multicamada Java EE, a camada de negócio é composta por três tipos de componentes: os *Servlets*, que processam os pedidos HTTP ao servidor aplicacional e retornam a resposta ao cliente, obtida através da invocação de EJBs, os *Enterprise Java Beans*, que encapsulam a lógica da aplicação e os *Data Access Objects*, que disponibilizam uma interface para ler e escrever dados na camada de dados.

### 13.4.1 Servlets

A estrutura geral dos *servlets* implementados consiste nos seguintes passos: *parse* da informação do pedido, validação do *token* de autenticação, invocação dos métodos do EJB através do *FacadeBeans* e envio da informação da resposta em formato JSON. Foi implementado um *servlet* para cada *use case*.

Para *servlets* que processam pedidos POST, o *parse* consiste na transformação do JSON no corpo do pedido para estruturas apropriadas. Para *servlets* que processam um pedido GET com parâmetros, o *parse* consiste na extração dos valores da *query string*.

Todos os *servlets* à exceção do de autenticação verificam se o *token* de autenticação se encontra no cabeçalho do pedido, e validam o mesmo para obter a identidade do utilizador e verificar se tem permissão para invocar a operação.

Após invocar os métodos dos *EJBs* correspondentes e de obter o resultado destes, é construída a resposta para retornar ao cliente, com o sucesso da operação e dados adicionais relativos à operação invocada. A resposta é dada em JSON, que é interpretado pelo *web server*, sendo que este funciona como o cliente dos *servlets*.

Para além disto, os *servlets* de autenticação e registo de utilizadores encriptam a *password* do utilizador por razões de segurança.

### 13.4.2 Enterprise Java Beans

De acordo com a modelação apresentada anteriormente, foram implementados cinco *EJBs stateless* diferentes. Todos os *beans* acedem à camada de dados através do *FacadeDAOs*, sendo este um *Facade* para os diversos DAOs do Hibernate.

O *AutenticarBean* encapsula a lógica de autenticação da aplicação, incluindo as seguintes funcionalidades:

- autenticação de todos os tipos de utilizadores através do *e-mail* ou do *token*;
- verificação se um utilizador está ligado;
- *logout*;
- atribuição do *token* de autenticação.

O *UtilizadorBean* trata das operações de:

- registo dos dois tipos de utilizadores;
- edição dos dados de utilizador;
- avaliação de utilizadores;
- consulta de avaliações;
- consulta de perfil;
- desativação de utilizadores;

- deteção do tipo de utilizador;

É importante realçar que a avaliação de um utilizador desencadeia o cálculo da sua avaliação média.

O *DonoBean* implementa a lógica relacionada com o registo, edição, consulta e remoção dos animais do Dono. A consulta de animais só retorna animais que estejam ativos.

O *PetsitterBean* encapsula a lógica de registo e edição de dados relacionados com o *petsitter* (horário, tipos de animais que trata e preço dos serviços) e da consulta destes mesmos dados. Visto que os dados do *petsitter* são obrigatórios para este poder usufruir da aplicação, a conta do *petsitter* só é ativada no fim do registo destes dados, ao contrário do dono, cuja conta pode ser criada sem animais registados.

Finalmente, o *PedidoBean* implementa a lógica do processo de fazer um pedido:

- registo dos dados selecionados pelo cliente, como os animais, as datas, os serviços, o *petsitter* e o preço;
- cálculo dos dados relevantes a mostrar ao cliente durante o processo de criação do pedido, como os serviços disponíveis para os animais selecionados, os *petsitters* disponíveis para satisfazer o pedido e os respetivos preços;
- cancelamento de pedidos;
- consulta de pedidos.

O cálculo dos *petsitters* disponíveis para fazer o pedido segue as seguintes regras:

- o *petsitter* deve ser capaz de realizar todos os serviços selecionados pelo cliente, assim como ser capaz de tratar dos tipos de animais do pedido
- o *petsitter* deve ter um horário compatível com o pedido
- o *petsitter* não deve ter outro pedido agendado para o mesmo período

### 13.5 Acesso a Dados

Para aceder à base de dados foram utilizados os DAOs e as classes de persistência gerados pelo Visual Paradigm a partir do PSM Hibernate apresentado na figura 5.

Foi implementado um *FacadeDAO* para simplificar a interação com a interface dos DAOs gerados, com as operações necessárias de leitura e escrita de objetos persistentes (*get*, *save*, *create* e *list*).

Visto que não foram utilizadas sessões, a gestão das sessões do Hibernate (*PersistentSession*) é efetuada pelo servidor aplicacional.

### 13.6 Web Services

No desenvolvimento da aplicação recorreu-se a dois serviços externos para fornecimento de imagens através da obtenção do seu *URL*. Os *Web Services* utilizados foram o *The Cat*<sup>2</sup> e o *Dog CEO Dog*<sup>3</sup>. O primeiro fornece imagens aleatórias de gatos e o segundo de cães. Para ambos os serviços, é carregado o conteúdo dos *links* e filtrada a resposta JSON obtida, recolhendo apenas o *URL* da imagem. Este conteúdo é utilizado como avatar de gatos e cães registados na aplicação sem fotografia.

## 14 Deployment

Relativamente ao *deployment* da aplicação, foi utilizada uma arquitetura específica, que será descrita na próxima secção. Para além disso fez-se balanceamento de carga para melhorar a performance da aplicação e foi utilizada uma ferramenta de automatização para simplificar o *deployment*.

### 14.1 Arquitetura

De forma a não existirem pontos únicos de falha que possam comprometer o serviço, decidiu-se utilizar dois servidores aplicacionais. Com o objetivo de melhorar a performance da aplicação utilizaram-se dois servidores *web* para servir conteúdo estático. Tendo mais do que um servidor *web*, conseguimos também tolerar faltas a este nível. No que toca ao armazenamento de dados, utilizou-se apenas uma base de dados PostgreSQL.

Para efetuar o balanceamento de carga entre os dois servidores *web* e os dois servidores aplicacionais utilizaram-se duas instâncias de HAProxy, que efetuam o balanceamento segundo um algoritmo *Round-robin*.

Tal como se pode observar na figura, embora o cliente aceda ao endereço do HAProxy que efetua o balanceamento entre os dois servidores *web*, ao serem renderizadas as páginas são feitos *fetches* aos servidores aplicacionais.

---

<sup>2</sup><https://www.programmableweb.com/api/cat>

<sup>3</sup><https://www.programmableweb.com/api/dog-ceo-dog>

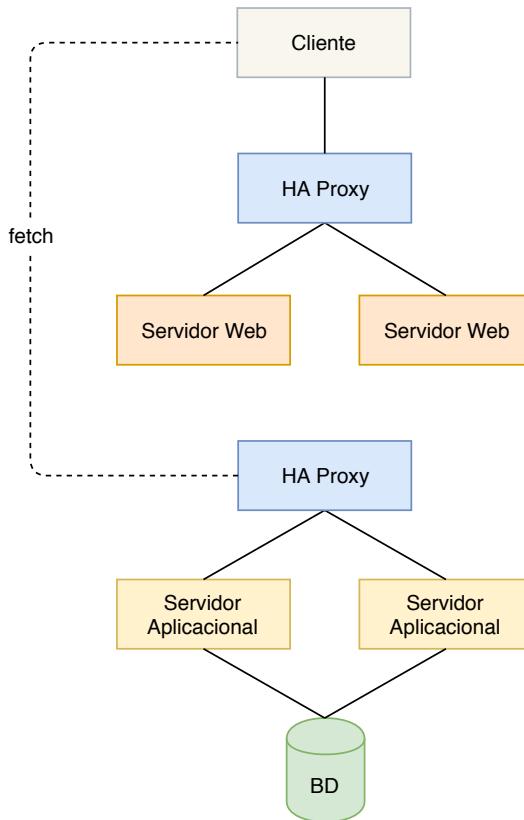


Figura 21: Arquitetura final da aplicação TrustPet.

## 14.2 Automatização do Deployment

Para automatizar e simplificar o processo do *deployment* dos vários componentes da aplicação, recorreu-se a Docker. A utilização desta ferramenta permite o *deployment* de imagens (ficheiros com a configuração de um componente) em *containers*, com benefícios para a portabilidade e performance da aplicação.

Para cada um dos componentes apresentados na figura 21 foram criadas imagens Docker e posteriormente foram lançadas *containers* a correr essas mesmas imagens.

## 15 Benchmarking

Para testar o desempenho da aplicação sobre carga, foram realizados testes de carga através do Locust, uma ferramenta de testes de carga baseada em eventos.

A ferramenta permite a definição de tarefas para cada cliente simulado, assim como a estipulação dos tempos de espera para cada cliente entre a execução de tarefas. Cada tarefa consiste num pedido ao servidor aplicacional e na análise da resposta para verificar se esta é correta. Os pedidos correspondem às diversas invocações que são feitas ao servidor aplicacional no decorrer da aplicação.

Foram definidas tarefas para todos as operações que o servidor aplicacional suporta, de maneira a aproximar o teste de carga à utilização real da aplicação.

Para além da definição de tarefas, o Locust permite atribuir um peso a cada tarefa, definindo a probabilidade de um dos clientes fazer um certo pedido. Esta funcionalidade foi utilizada para atribuir um peso maior a operações mais comuns, como um dono consultar os seus animais, e um peso menor a operações menos comuns, como um *petsitter* editar os serviços que disponibiliza.

Foi estabelecida uma hierarquia de tarefas de modo a simular o fluxo de lógica presente na aplicação e a dividir as tarefas de utilizador, dono e *petsitter*. Cada cliente é atribuído um email único ao ser criado, correspondendo a uma conta de dono ou *petsitter*, de acordo com as probabilidades definidas. Para garantir que o processo de fazer um pedido se mantinha consistente, foram utilizadas as capacidades de sequenciar tarefas do Locust.

Depois da definição do comportamento dos clientes, é possível correr os testes com um dado número de clientes. Foram corridos testes com diferentes números de clientes a usar a aplicação concorrentemente para testar a escalabilidade da aplicação.

O Locust permite analisar as estatísticas de cada teste, especificando para cada tipo de pedido:

- o número de pedidos feitos;
- o número de pedidos com sucesso e com falha;
- os tempos mínimos, máximos, médios e medianos dos pedidos;
- o tamanho médio da resposta;
- o número de pedidos por segundo.

A seguir são apresentados os diferentes testes corridos e os resultados obtidos. Foram medidos os *Requests Per Second* (RPS), o tempo médio e mediano para vários números de clientes.

Número de clientes	RPS	Tempo Médio (ms)	Tempo Mediano (ms)
50	12	10	8
100	23	10	8
200	49	13	8
500	104.5	373	28

Podemos verificar que o sistema escala bem até aos 500 clientes, aumentando os RPS e não sofrendo alterações muito significativas aos tempos de resposta médios e medianos. Ao analisar mais detalhadamente as estatísticas recolhidas, podemos verificar que as operações leves (consultas e edição de dados simples) não sofrem um aumento de tempo de resposta considerável, enquanto os tempos de resposta das operações mais pesadas (processo de fazer um pedido) aumentam consideravelmente.

Devido aos limites computacionais do hardware utilizado, o teste de 500 clientes não é fiável, visto que o gasto aos recursos provenientes de correr a aplicação mais os gastos dos testes afetam os resultados negativamente do teste.

Finalmente, é apresentada uma página com os dados relativos ao teste de 200 clientes, com dados mais detalhados. Aqui podemos comparar os tempos de resposta para os vários pedidos, chegando a conclusões sobre as operações mais pesadas para o servidor aplicacional.

Type	Name	# Requests	# Fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
POST	/trustpet_war_exploded.war/Autenticar	444	0	6	7	3.298044204711914	31.190156936645508	73	0.6
POST	/trustpet_war_exploded.war/CancelarPedido	5	0	24	24	12.00413703918457	33.899784088134766	16	0.2
GET	/trustpet_war_exploded.war/ConsultarAnimais	346	0	7	9	4.157066345214844	41.39610491262207	1585	4.7
GET	/trustpet_war_exploded.war/ConsultarPedidos	495	0	8	9	3.895988001098633	68.44687461853027	315	8.1
GET	/trustpet_war_exploded.war/ConsultarPerfil	782	0	8	10	3.86428833078125	77.55494117736816	844	12.3
POST	/trustpet_war_exploded.war/ConsultarPerfil	472	0	8	9	4.274129867553711	62.454938888549805	1156	7
GET	/trustpet_war_exploded.war/ConsultarPetitlers	169	0	13	26	8.212804794311523	675.3978729248047	109199	2.9
POST	/trustpet_war_exploded.war/EditarAnimal	203	0	14	16	4.781934280395508	46.34881019592285	14	2.5
POST	/trustpet_war_exploded.war/EditarDadosPessoais	232	0	12	14	3.954172134399414	155.5171012878418	16	2.6
POST	/trustpet_war_exploded.war/EditarHorario	67	0	70	63	4.144906997680664	255.4781436920166	16	0.8
POST	/trustpet_war_exploded.war/EditarServicos	84	0	41	39	4.148960113525391	128.96728915625	16	1.5
GET	/trustpet_war_exploded.war/Index	47	0	7	8	4.028081893920898	49.6251583093652	38	0.1
GET	/trustpet_war_exploded.war/Logout	109	0	12	13	4.527091979980469	38.61713409423828	16	1.7
POST	/trustpet_war_exploded.war/RegistrarPedido	180	0	16	19	10.161161422729492	57.74283409118652	1143	3.1
POST	/trustpet_war_exploded.war/SelPetitler	21	0	15	17	10.943174362182617	35.91203689575195	16	0.8
POST	/trustpet_war_exploded.war/SelServicos	24	0	56	65	37.4491247521973	123.08001518249512	12255	0.5
Total		3680	0	9	13	3.298044204711914	675.3978729248047	5682	49.4

Figura 22: Resultados obtidos na execução de um teste de 200 clientes no Locust

Podemos verificar que uma das operações do processo de fazer um pedido, correspon-

dente ao *servlet SelServicos*, apresenta tempos de resposta acima dos outros pedidos, devido à sobrecarga causada pelo processamento dos *petsitters* disponíveis, que foi identificada como a tarefa de lógica mais pesada.

## 16 Conclusões e Trabalho Futuro

Embora nem todos os requisitos impostos inicialmente tenham sido cumpridos, nomeadamente a integração de um *chat* para os donos terem a possibilidade de combinar pormenores com os *petsitters*, e a utilização de notificações, considera-se que, de uma forma geral, a realização do projeto foi bem sucedida.

Relativamente à lógica de negócio existe um fator crucial que poderia ser melhorado relativamente aos horários dos *petsitters*. Nesta fase consideramos que, nos casos em que são feitos pedidos de *petsitting* temporários, isto é, pedidos que se estendem por mais de um dia, os *petsitters* ficam com os seus horários sobrelotados. Estes cenários acabam por se afastar da realidade visto que o simples facto que um *petsitter* ter, por exemplo, de tomar conta de um cão durante  $x$  dias, faz com que não possa receber mais nenhum pedido de outros clientes. A forma como a compatibilidade dos horários é gerida quando é efetuado um pedido deveria ser drasticamente alterada, visto que a forma como foi implementada pode prejudicar o negócio dos *petsitters* que dependem da aplicação para prestar serviços.

Relativamente à arquitetura adotada, embora os servidores aplicacionais e os servidores *web* sejam toleradas falhas através da redundância, no caso da base de dados isto já não acontece. Sendo assim, futuramente deveria ser utilizada alguma tecnologia, como por exemplo, DRBDs, para implementar a replicação da base de dados, prevenindo que ocorram falhas irreversíveis.

Ainda no que toca à arquitetura, ao realizar o *benchmarking* apercebemo-nos também que toda a lógica relativa à realização de pedidos é bastante pesada e provoca um impacto significativo na performance do serviço. Desta forma, uma melhoria que poderia também ser efetuada era correr remotamente o EJB *PedidoBean*.

Relativamente ao *front-end*, apesar de termos implementado todas as interfaces relativas às funcionalidades previstas, estas poderiam ser aperfeiçoadas por forma a melhorar a experiência do utilizador, nomeadamente otimizar o tempo de renderização das páginas do lado do cliente. Para além disso, existem interfaces que poderiam ser melhoradas em termos de conteúdo e apresentação, por forma a facilitar a realização de algumas tarefas. É exemplo disso as páginas de registo e de edição do horário de um *petsitter*.