

Universidade do Minho
Mestrado em Engenharia Informática

Arquiteturas Aplicacionais e Sistemas Interativos Confiáveis

Trabalho prático

17 de junho de 2022



Carolina Oliveira
(pg47101)

Carolina Santejo
(pg47102)

Manuel Moreira
(PG47439)

Raquel Costa
(pg47600)

Sara Dias
(pg47667)

Conteúdo

1	Introdução	7
1.1	Motivação	7
1.2	Contextualização	7
1.3	Caracterização dos utilizadores	8
1.3.1	Cliente	8
1.3.2	<i>Petsitter</i>	8
1.4	Fluxo geral da plataforma	9
1.4.1	<i>Petsitter</i>	9
1.4.2	Cliente	9
2	Modelação	10
2.1	Modelo de domínio	10
2.2	Requisitos	11
2.2.1	Requisitos funcionais	11
2.2.2	Requisitos não funcionais	12
2.3	Diagrama de <i>Use Cases</i>	13
2.4	Especificação de <i>Use Cases</i>	14
2.4.1	Realizar um pedido	14
2.4.2	Configurar anúncio	15
2.4.3	Criar perfil de animal	16
2.5	Modelos de tarefas	17
2.5.1	Realizar um pedido	17
2.5.2	Configurar um anúncio	18
2.5.3	Criar perfil para o animal	19
2.6	Protótipos da Interface	20
2.6.1	Registar cliente	20
2.6.2	Registar <i>petsitter</i>	21
2.6.3	<i>Login</i>	21
2.6.4	Perfil de cliente	22
2.6.5	Criar perfil de animal	22
2.6.6	Página inicial do cliente	23
2.6.7	Consultar anúncio do <i>petsitter</i>	23
2.6.8	Realizar pedido	24
2.6.9	Perfil do <i>petsitter</i>	25
2.6.10	Editar anúncio	25
2.7	Máquina de estados	26
2.8	Diagrama de Componentes	27
2.9	Diagrama de Classes	28
2.10	Modelo lógico da base de dados	30
3	Arquitetura da Aplicação	32
4	Implementação	33

4.1	<i>Frontend</i>	33
4.1.1	<i>Componentes</i>	33
4.1.2	<i>Router</i>	34
4.1.3	<i>Store</i>	34
4.1.4	Pedidos à API	34
4.2	<i>Backend</i>	34
4.2.1	<i>Controllers</i>	34
4.2.2	<i>Beans</i>	34
4.2.3	Acesso à API externa	35
4.2.4	Base de dados	36
5	<i>Deployment</i>	37
5.1	Considerações iniciais	37
5.2	<i>Frontend</i>	37
5.3	<i>Backend</i>	38
6	Aplicação Desenvolvida	39
6.1	Login	39
6.2	Login erro	39
6.3	Registo de cliente	40
6.4	Registo de petsitter	41
6.5	Registo - Escolha do país	42
6.6	Registo - Escolha da cidade após escolhido o país	42
6.7	Registo - Campo vazio	43
6.8	Registo - menor de idade	43
6.9	Página inicial	44
6.10	Anúncio	44
6.11	Realizar um pedido	45
6.12	Perfil do cliente	45
6.13	Criar um perfil do animal	46
6.14	Criar um perfil do animal - campo vazio	46
6.15	Perfil do petsitter	47
6.16	Editar anúncio	47
6.17	Pedidos de um Petsitter	48
6.18	Preços estabelecidos pela plataforma	48
7	Análise Crítica	49
7.1	Considerações gerais	49
7.2	Testes de carga	49
7.2.1	Fazer Pedido	49
7.2.2	Interação 1 – Cliente	50
7.2.3	Interação 2 – Petsitter	51
8	Análise de usabilidade	54
8.1	Princípios de usabilidade	54

8.1.1	<i>Learnability</i>	54
8.1.2	<i>Flexibility</i>	54
8.1.3	<i>Robustness</i>	55
8.2	Heurísticas de Nielsen	56
8.2.1	<i>Visibility of system status</i>	56
8.2.2	<i>Match between system and the real world</i>	56
8.2.3	User control and freedom	57
8.2.4	Consistency and standards	57
8.2.5	Error prevention	58
8.2.6	Recognition rather than recall	58
8.2.7	Flexibility and efficiency of use	58
8.2.8	Aesthetic and minimalist design	59
8.2.9	Help users recognize and recover from errors	59
8.2.10	Help and documentation	59
8.3	Questionário PSSUQ	60
9	Conclusões	62
10	Trabalhos Futuros	63
10.1	Funcionalidades	63
10.2	<i>Deployment</i>	63

Lista de Figuras

2	Modelo de domínio	10
3	Diagrama de <i>Use Cases</i>	13
4	Especificação do use case - realizar pedido	14
5	Especificação do use case - configurar anúncio	15
6	Especificação do use case - criar perfil de animal	16
7	Modelo de tarefas - realizar um pedido	17
8	Modelo de tarefas - configurar um anúncio	18
9	Modelo de tarefas - criar perfil para o animal	19
10	<i>Mockup</i> - Registar cliente	20
11	<i>Mockup</i> - Registar <i>petsitter</i>	21
12	<i>Mockup</i> - <i>login</i>	21
13	<i>Mockup</i> - perfil de cliente	22
14	<i>Mockup</i> - criar perfil de animal	22
15	<i>Mockup</i> - página inicial do cliente	23
16	<i>Mockup</i> - perfil de <i>petsitter</i> visto por clientes	23
17	<i>Mockup</i> - realizar pedido	24
18	<i>Mockup</i> - perfil de <i>petsitter</i> visto por si próprio	25
19	<i>Mockup</i> - editar anúncio	25
20	Máquina de Estados	26
21	Diagrama de componentes	27
22	Diagrama de classes	28
23	Modelo lógico	30
24	Diagrama da Arquitetura	32
25	Diagrama de Deployment	37
26	Deployment do <i>Frontend</i>	38
27	Login	39
28	Login erro	39
29	Registo de cliente	40
30	Registo de <i>petsitter</i>	41
31	Registo - escolha do país	42
32	Registo - escolha da cidade	42
33	Registo - campo vazio	43
34	Registo - menor de idade	43
35	Página inicial	44
36	Anúncio	44
37	Realizar um pedido	45
38	Perfil do cliente	45
39	Criar um perfil do animal	46
40	Criar um perfil do animal - campo vazio	46
41	Perfil do Petsitter	47
42	Editar anúncio	47
43	Pedidos de um Petsitter	48
44	Preços estabelecidos pela plataforma	48

45	Funcionalidade ”Make Request”	50
46	Interação 1 – Cliente	51
47	Interação 2 – <i>Petsitter</i>	52
48	Response Time Graph	53
49	Filtros	55
50	Filtros	55
51	Processamento	56
52	57
53	Botão de cancelar operação	57
54	Utilizador menos de idade	58
55	Barra de navegação	59
56	Erro de login	59

1 Introdução

1.1 Motivação

Atualmente, tem-se notado um aumento no número de lares que têm animais de estimação. No entanto, as pessoas têm cada vez mais um estilo de vida acelerado, onde o equilíbrio entre a vida profissional e pessoal nem sempre é respeitado. Consequentemente, as pessoas encarregues por animais de estimação têm cada vez mais dificuldade em lhes proporcionar uma boa qualidade de vida. Para além disso, muitas vezes as pessoas acabam por não adotar, uma vez que este mesmo estilo de vida não o permite.

É neste contexto que a nossa aplicação surge, de forma a auxiliar essas pessoas através de serviços dedicados ao cuidado e atenção dos animais. O objetivo é criar uma plataforma onde as pessoas possam disponibilizar-se para cuidar de animais de estimação. Deste modo, os vários *petsitters* poderão fornecer serviços de higiene, treinar ou apenas tomar conta dos animais por um determinado período de tempo. Por vezes, os hotéis e creches de animais podem não ser viáveis em termos económicos, então a aplicação propõe-se a responder a esse problema.

1.2 Contextualização

A **PetIt** é uma plataforma de procura e contratação de serviços de *petsitting* com todas as funcionalidades essenciais para que o processo de procura e contratação de *petsitters* seja feito sempre de uma maneira simples, fácil e rápida.

Os principais intervenientes são o **Petsitter** e o **Cliente**. Passamos então à descrição das principais características da aplicação:

1. Permitir a *petsitters* e clientes registarem-se e iniciarem sessão;
2. Permitir ao cliente ter acesso aos anúncios dos vários *petsitters*, com informação sobre os serviços que prestam. O cliente deve ter a opção de poder filtrar os resultados dependendo do que procura.
3. Permitir ao cliente aceder às suas marcações futuras e à sua lista de animais. O cliente poderá adicionar novos animais à lista.
4. Permitir ao cliente adicionar um novo animal ao seu perfil, indicando o nome, espécie, raça, sexo, peso e uma fotografia.
5. Permitir ao cliente escolher um *petsitter* com um serviço que lhe agrade e fazer um pedido, selecionando a data, o serviço e o animal.
6. Permitir ao *petsitter* alterar o seu anúncio, podendo alterar a sua disponibilidade, a sua taxa, os serviços que presta e as espécies de animais sobre as quais presta os serviços.
7. Permitir ao *petsitter* aceder às suas marcações futuras, podendo filtrar por serviço, data e podendo pesquisar por clientes.

1.3 Caracterização dos utilizadores

De modo a concluir a fundamentação da fase inicial do desenvolvimento do projeto, tratamos de fazer um pequeno estudo para tentar perceber a possível abrangência da aplicação.

Por um lado, temos os *Petsitters*, e por outro os Clientes, que constituem a comunidade em geral, com aproximadamente entre os 18 e os 60 anos. Recorremos assim à caracterização de duas *personas* que passamos a apresentar de seguida:

1.3.1 Cliente

Diogo Pereira, 28 anos, Lisboa

Diogo é *district manager* de um conjunto de lojas do grupo *Inditex*, em Lisboa, e que acabou de se mudar da sua terra natal, Braga, para a cidade onde se encontram as suas lojas.

Ultimamente tem tido pouco tempo para passear com o seu cão, Loki, mas ainda não conhece ninguém em Lisboa a quem possa confiar o seu animal. Apesar de recuar que um desconhecido cuide do seu melhor amigo, não tem outra escolha e não quer que o seu animal passe a semana fechado no apartamento.

Para além disso, sabe que os preços na aplicação **Petit** são muito mais económicos que os preços de algumas empresas especialistas da área, e recebeu bom *feedback* de amigos próximos que já precisaram de utilizar a aplicação.

1.3.2 *Petsitter*

Joana Freitas, 18 anos, Lisboa

Joana é uma estudante do ensino secundário, e desde sempre teve uma afeição grande por todo o tipo de animais. A sua família até costumava ficar chateada quando passavam na zona dos répteis no jardim zoológico, que merecia uma atenção especial por parte de Joana, que tentava sempre pegar nas cobras.

É uma rapariga empenhada e determinada, sempre com solução para conseguir o que quer. Desde pequena que gosta de amealhar o seu próprio dinheiro para se sentir independente. Também foi desde aí que aliou ao seu gosto por animais o ganho de dinheiro. Aos 10 anos, começou por brincar com os cães dos vizinhos quando ia ao parque. Aos 13 anos, eles já lhe pediam para ir dar comida aos animais, se se ausentasse para férias. Desde aí que se disponibilizou para auxiliar os donos dos animais com mais tarefas (como levar à tosquia), conforme os vizinhos iam disseminando os seus serviços, e, claro, por um preço.

Desde que soube da existência da aplicação *Petit*, que uma amiga virtual sua usava na Bélgica, decidiu tornar o seu *hobby* em algo mais sério, que lhe traria gosto e também o dinheiro que desejava. Com um horário escolar não muito preenchido, consegue ajudar os clientes até ao fim de semana, e definir as suas próprias taxas com o valor que considera merecido. Sendo uma pessoa bastante organizada, valoriza o facto de poder ver o horário de cada marcação no seu perfil na aplicação.

1.4 Fluxo geral da plataforma

O fluxo geral da plataforma deve ser simples e minimalista. A plataforma deverá possuir como funcionalidades base o **registo**, tanto de clientes como de *petsitters*, a **realização** de pedidos por parte dos clientes e a **configuração** de anúncios por parte dos *petsitters*.

1.4.1 Petsitter

Qualquer pessoa pode **registar-se** na aplicação como *Petsitter*. Para tal é necessário fornecer as suas informações pessoais, como nome, *password*, data de nascimento, localização, número de telemóvel, entre outros.

Um *petsitter* que esteja autenticado pode, a qualquer momento, **consultar** o seu histórico de pedidos por realizar, accedendo para isso ao seu perfil, onde encontrará uma listagem dos pedidos que clientes registaram.

O *petsitter* pode **configurar** o seu anúncio, onde pode optar por alterar, remover ou adicionar serviços que preste, espécies com que trabalhe, a taxa que cobra, e a sua disponibilidade para cada dia da semana.

Numa versão futura da aplicação, o *petsitter* também será capaz de **visualizar** o perfil de um animal que esteja presente num dos seus pedidos, assim como **avaliar** um cliente com que tenha trabalhado.

1.4.2 Cliente

Para usufruir de um serviço da aplicação, um cliente deverá **registar-se** na aplicação como *Client*. Para tal, é necessário fornecer o seu *email*, nome, *password*, número de telemóvel e localização.

Um cliente que esteja autenticado pode **criar** um perfil para o seu animal de estimação, indicando o seu nome, espécie, raça, e opcionalmente doenças e uma breve descrição.

Na sua página inicial, os clientes poderão **consultar** anúncios de *petsitters*. Para **realizar um pedido**, o cliente deverá selecionar o anúncio que pretende e em seguida, escolher uma data e hora para o pedido, o serviço que pretende e um dos seus animais registados. Adicionalmente, o cliente pode **consultar** o seu histórico de pedidos agendados.

No futuro, o cliente também será capaz de **avaliar** um *petsitter* a quem tenha requisitado um serviço.

2 Modelação

2.1 Modelo de domínio

Em primeiro lugar, de modo a conseguirmos apresentar o funcionamento básico da nossa aplicação, decidimos criar um modelo de domínio que a representasse. Deste modo, através de um *brainstorming* de ideias realizado entre os elementos do grupo foi possível levantar as principais entidades do sistema em estudo. Resultando, assim, num conhecimento mais aprofundado do contexto do problema entre todas as partes envolventes.

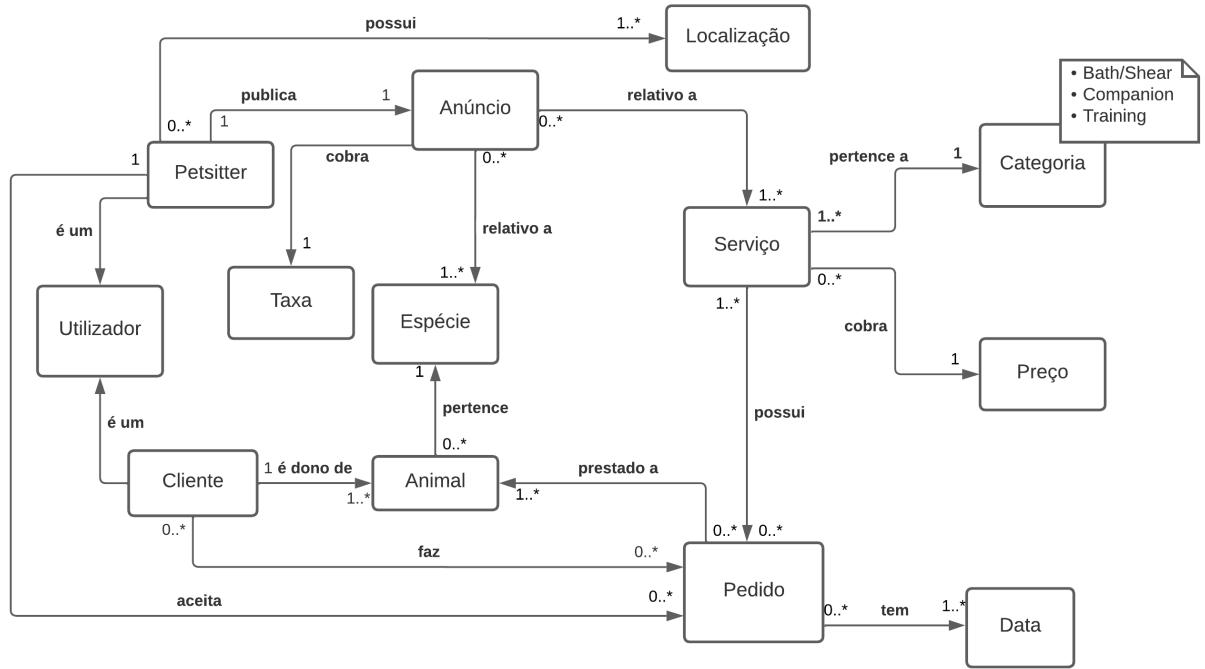


Figura 2: Modelo de domínio

Um **Utilizador** da aplicação pode ser um **cliente** ou um **petsitter**. Cada *petsitter* tem de indicar a sua **localização** na altura do registo e possui um único **anúncio**, onde deverá indicar quais as **espécies** de animais (cão, gato, etc.) com que trabalha, qual a **taxa** que cobra pelos serviços e os **serviços** de que dispõe. Cada serviço é identificado pela sua **categoria**, podendo esta ser *bath_shear*, *companion* e *training*, assim como um **preço** que é pré-definido pela plataforma. Este difere consoante a sua categoria e outros aspetos, como o peso do animal ou a duração do serviço.

Um cliente que esteja autenticado no sistema poderá registar **animais** e realizar **pedidos**, numa certa **data**, relativos a serviços prestados por *petsitters* registados.

2.2 Requisitos

Para realizar o levantamento de requisitos, foi crucial realizar uma investigação do domínio da aplicação, tal como visto anteriormente. Para tal, procurou-se saber o que é que os potenciais utilizadores da aplicação, tanto clientes como *petsitters*, pretendiam conseguir fazer nesta. Assim sendo, o grupo fez entrevistas a diversas pessoas que possuem animais de estimação, ou que gostariam de ganhar algum dinheiro fazendo algo que gostam. Recorreu-se então, tanto a familiares como amigos, para a obtenção de respostas e de uma exploração mais aprofundada do contexto do problema. Por este meio foi possível perceber quais as funcionalidades principais e mais procuradas que devem estar incluídas no sistema e qual a sua ordem de prioridade.

Em seguida, serão apresentados os requisitos funcionais e não funcionais que foram levantados.

2.2.1 Requisitos funcionais

Foi feita uma divisão dos requisitos funcionais por tipo de utilizador – **clientes** ou **petsitters** –, e quanto à importância – principais e secundários.

Cliente

Principais:

1. Permitir **registro** na aplicação, tendo de indicar *email*, nome, *password*, número de telemóvel, cidade, podendo fazer o registo dos seus animais de estimação.
2. Permitir o **login** na aplicação, indicando o *email* e *password*.
3. Permitir **criar perfil** para os seus animais, indicando o seu nome, espécie, peso, doenças e descrição, podendo ainda adicionar uma foto do seu animal.
4. **Consultar** anúncios.
5. **Consultar** perfil de um animal.
6. **Realizar um pedido** indicando o serviço, os animais, data.
7. **Consultar** histórico de pedidos agendados.

Secundários:

1. **Cancelar o pedido** de serviço realizado.
2. **Avaliar** o serviço prestado pelo *petsitter*.
3. **Permitir** eliminar o perfil dos seus animais.
4. **Editar** o seu perfil.

Petsitter

Principais:

1. Permitir **registro** na aplicação, tendo de indicar *email*, nome, *password*, data de nascimento, número de telemóvel e cidade.
2. Permitir o **login** na aplicação, indicando o *email* e *password*.
3. Permitir **publicar anúncio**, indicando os dias, horas, localizações, serviços que presta e as categorias, animais e preços desses serviços.
4. **Configurar** anúncio.
5. **Consultar** histórico de pedidos agendados.
6. **Editar** perfil.

Secundários:

1. **Aceitar** um pedido do cliente.
2. **Recusar** um pedido do cliente.
3. **Consultar** perfil de um animal.
4. **Avaliar** um cliente.

2.2.2 Requisitos não funcionais

1. O sistema deverá ser responsivo, adaptando a sua apresentação dependendo do dispositivo utilizado.
2. A interface do utilizador não deve conter linguagem que possa ser considerada discriminatória e ofensiva.
3. Na primeira utilização da aplicação, sem ajuda, 90% dos utilizadores deverá conseguir efetuar um pedido em menos de 7 minutos.
4. O sistema deve ser capaz de suportar vários pedidos em simultâneo.
5. O sistema deve validar os dados inseridos pelos utilizadores para prevenir erros.
6. O sistema deve responder às ações do utilizador em menos de 2 segundos, desprezando eventuais atrasos da rede.
7. A linguagem utilizada não deve recorrer aos termos técnicos relacionados com a implementação da plataforma.
8. O código fonte da aplicação deve conter comentários de modo a facilitar a sua manutenção.
9. Novas funcionalidades deverão ser adicionadas durante um horário em que se verifique um menor número de utilizadores ativos.

2.3 Diagrama de *Use Cases*

Identificados os requisitos e as principais entidades do sistema, foi altura de identificar e em seguida desenvolver o diagrama dos *use cases*. A equipa optou por apresentar apenas os casos de uso dos requisitos principais. Pintados a azul, estão identificados os casos de uso que o grupo considerou como sendo os mais importantes.

Como já se tinha referido, existem 2 tipos de utilizadores do sistema – clientes e *petsitters*. Ambos têm a possibilidade de realizar o **registro**, **login** e caso se encontre autenticado, consultar o seu histórico de pedidos.

Os *petsitters* têm como funcionalidades **configurar** o seu anúncio e **validar** pedidos.

Já os clientes poderão **consultar** anúncios de *petsitters*, **fazer um pedido** e **criar perfís** para os seus animais.

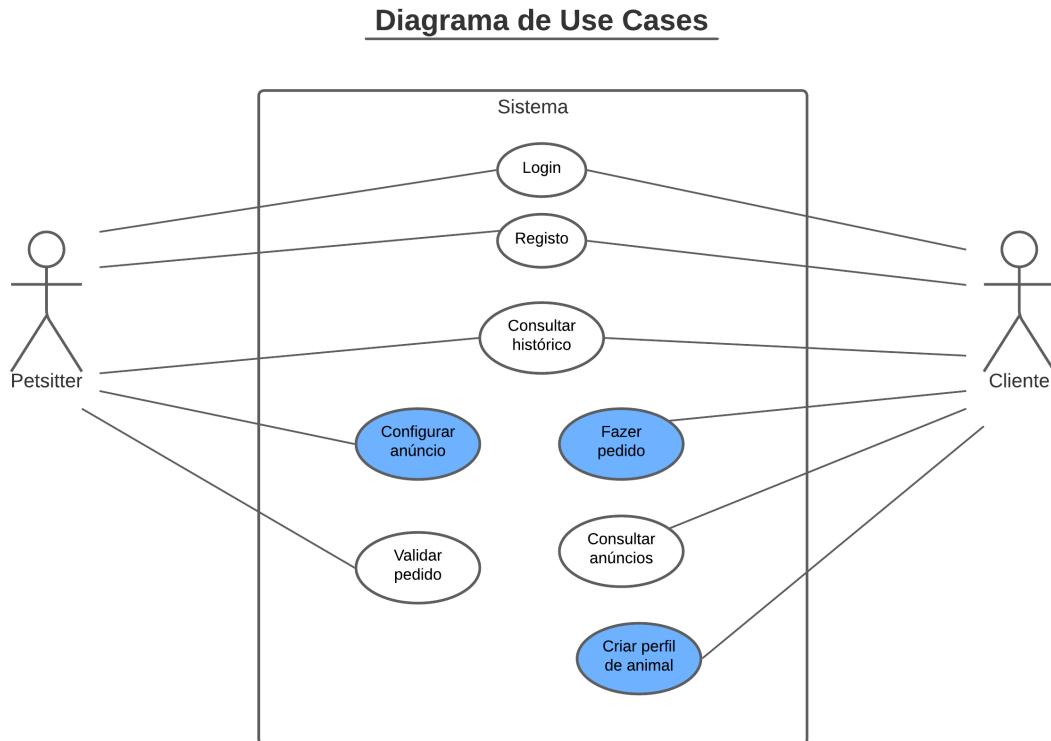


Figura 3: Diagrama de *Use Cases*

2.4 Especificação de *Use Cases*

Em seguida, vamos apresentar a especificação dos *use cases* assinalados a azul no diagrama.

2.4.1 Realizar um pedido

Uma das principais funcionalidades da aplicação é um cliente ter possibilidade de realizar um pedido. Para isto, terá de informar o sistema de qual o horário, o serviço e o animal que deseja para aquele pedido. O sistema passa então a ser responsável por verificar a disponibilidade do *petsitter* no horário que o cliente pretende. Se o *petsitter* estiver disponível, o pedido é registado no sistema. Caso contrário, o cliente é avisado que aquele horário não está disponível e se ainda tiver interesse em realizar o pedido, poderá escolher outro horário.

Use case		
Use Case:	Realizar um pedido	
Autor:	Cliente	
Pré-condição:	O cliente encontra-se autenticado	
Pós-condição:	O pedido foi registado no sistema	
	Autor	Sistema
Fluxo normal	1. Indica que quer realizar um pedido	
		2. Apresenta a página de registrar um novo pedido
	3. Seleciona o horário, animal e serviço que quer comprar	
		4. Verifica a disponibilidade do petsitter
		5. Regista o pedido na base de dados
Fluxo Exceção 1: [cliente não quer realizar um pedido](Passo 3)	3.1. A operação de configuração é cancelada	
Fluxo alternativo 1: [Petsitter não se encontra disponível no horário selecionado pelo cliente] (Passo 4)		4.1. Informa o cliente que o horário escolhido não está disponível
	4.2. Seleciona um novo horário	
		4.3 Volta a 4

Figura 4: Especificação do use case - realizar pedido

2.4.2 Configurar anúncio

Um *petsitter* a qualquer momento pode configurar o seu anúncio. Para isso, dependendo do que quer adicionar, remover ou alterar, deverá inserir a nova disponibilidade, serviço, espécies e/ou taxa que pretende. O sistema é responsável por depois verificar esses valores e alterar o estado do anúncio na base de dados.

Use case		
Use Case:	Configurar um anúncio	
Ator:	Petsitter	
Pré-condição:	O Petsitter encontra-se autenticado	
Pós-condição:	Configuração do anúncio foi bem sucedida	
	Ator	Sistema
Fluxo normal	1. Indica que quer configurar o seu anúncio	
		2. Apresenta o anúncio do petsitter
	3. Insere/remove serviços, espécies e/ou altera disponibilidade, taxa	
	4. seleciona guardar alterações realizadas ao anúncio	
		5. Regista as alterações ao anúncio
Fluxo exceção 1: [Petsitter quer cancelar a configuração](Passo 3 ou 4)	3/4.1. A operação de configuração é cancelada	

Figura 5: Especificação do use case - configurar anúncio

2.4.3 Criar perfil de animal

Um cliente que esteja autenticado pode adicionar um animal à plataforma, para que no futuro possa fazer pedidos para esse animal. Para criar um perfil para o seu animal de estimativa, o cliente deve inserir várias informações relevantes sobre o seu animal como nome, espécie, peso e doenças que este possa ter. Quando o cliente quiser gravar o perfil do animal, deverá selecionar a opção "Save", e a responsabilidade de registar o animal passa a ser do sistema.

Use case		
Use Case:	Criar perfil de animal	
Ator:	cliente	
Pré-condição:	O cliente encontra-se autenticado	
Pós-condição:	O novo animal foi registado	
	Autor	Sistema
Fluxo normal	1. Indica que adicionar um animal	
		2. Apresenta página de adicionar um novo animal
	3. Insere nome, espécie, raça, peso, doenças e descrição do animal	
	4. seleciona adicionar novo animal	
		5. Regista o novo animal no sistema
Fluxo exceção 1: [Cliente quer cancelar a operação](Passo 3)	3.1. A operação de adicionar novo animal é cancelada	

Figura 6: Especificação do use case - criar perfil de animal

2.5 Modelos de tarefas

Em seguida, o grupo decidiu desenvolver os modelos de tarefas para os principais casos de uso identificados na fase anterior, com o objetivo de descrever as ações dos utilizadores e estruturá-las numa hierarquia de tarefas.

2.5.1 Realizar um pedido

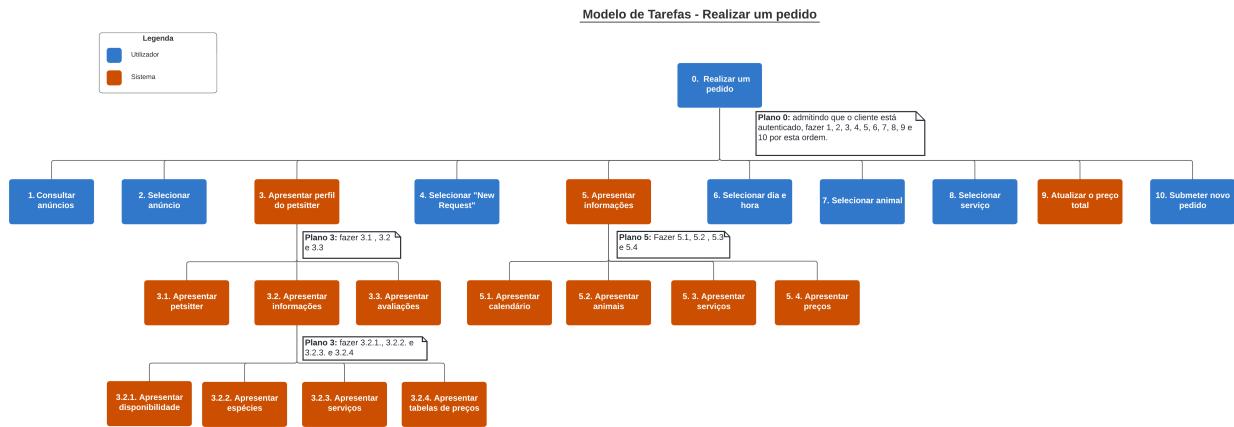


Figura 7: Modelo de tarefas - realizar um pedido

Para realizar um pedido, o cliente começa por consultar os anúncios disponíveis na página principal e selecionar aquele ao qual pretende registar um pedido. Em seguida, o sistema deverá apresentar as informações relativas ao anúncio selecionado. Se o cliente quiser prosseguir com a realização de um pedido àquele *petsitter*, deverá clicar no botão "*New request*", ao qual o sistema vai reagir apresentando as informações necessárias. O cliente deverá então preencher as informações pedidas e por fim submeter o novo pedido.

2.5.2 Configurar um anúncio

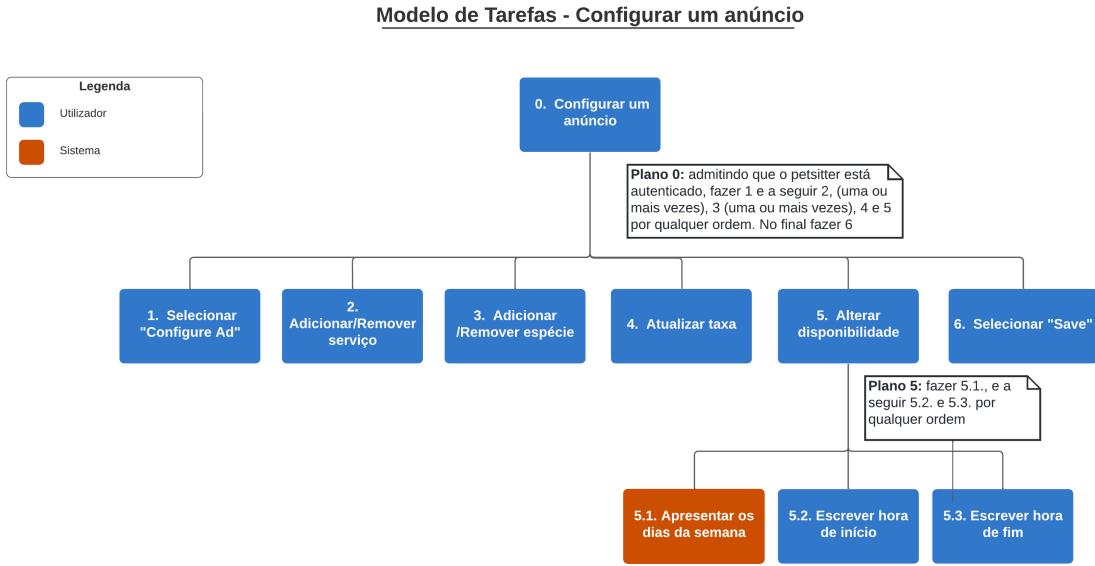


Figura 8: Modelo de tarefas - configurar um anúncio

Um *petsitter* que pretenda configurar o seu anúncio deverá selecionar a opção *"Configure Ad"* que se encontra no seu perfil, e alterar, remover ou adicionar as informações que quiser ao seu anúncio. Para finalizar, quando tiver realizado todas as alterações, o *petsitter* deverá clicar em *"Save"* para que o anúncio seja então gravado.

2.5.3 Criar perfil para o animal

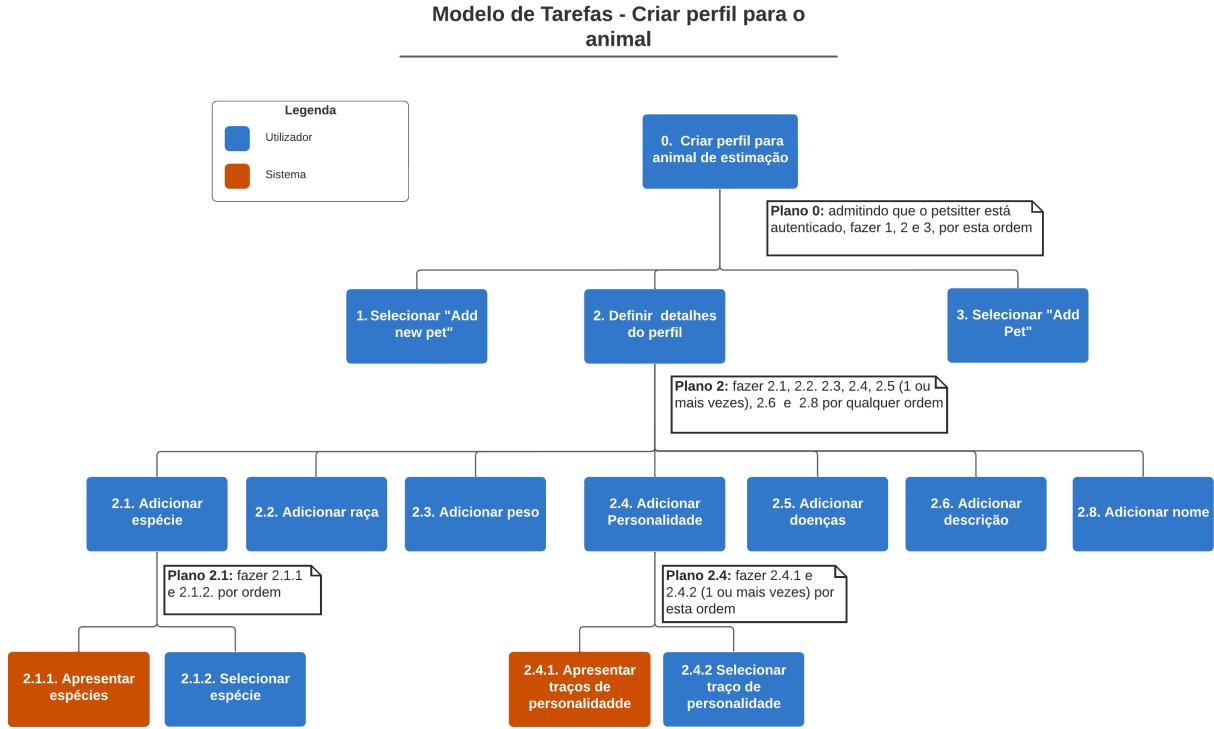


Figura 9: Modelo de tarefas - criar perfil para o animal

Para criar um perfil para um animal, o cliente tem de selecionar "*Add new pet*", que se encontra no seu perfil. Em seguida, deverá definir todos os detalhes relativos ao seu animal, como espécie, peso, etc, não sendo necessário seguir uma ordem específica. No final, para gravar o perfil do seu animal, deverá selecionar "*Add pet*".

2.6 Protótipos da Interface

Ainda nesta fase, foram elaborados protótipos de baixa fidelidade da *interface* gráfica da aplicação, recorrendo à ferramenta *Pencil*.

Ao longo do desenvolvimento destes protótipos, tivemos atenção à usabilidade da *interface*. Em seguida, encontram-se apresentados os protótipos finais.

2.6.1 Registar cliente

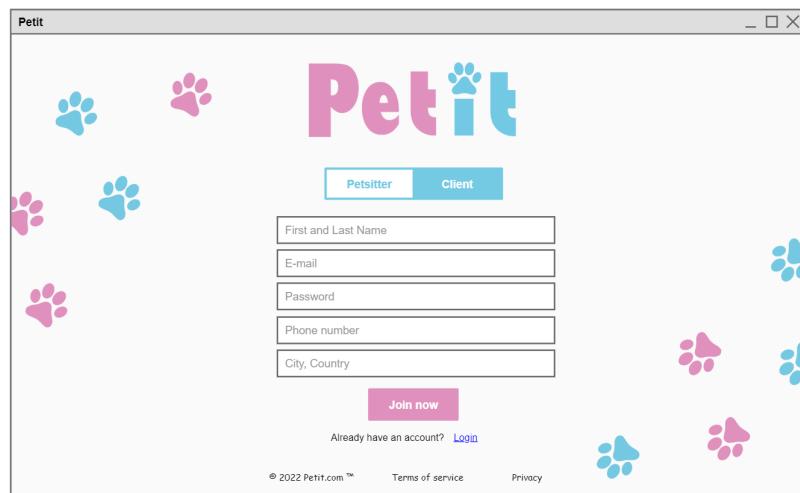


Figura 10: *Mockup* - Registar cliente

2.6.2 Registar *petsitter*

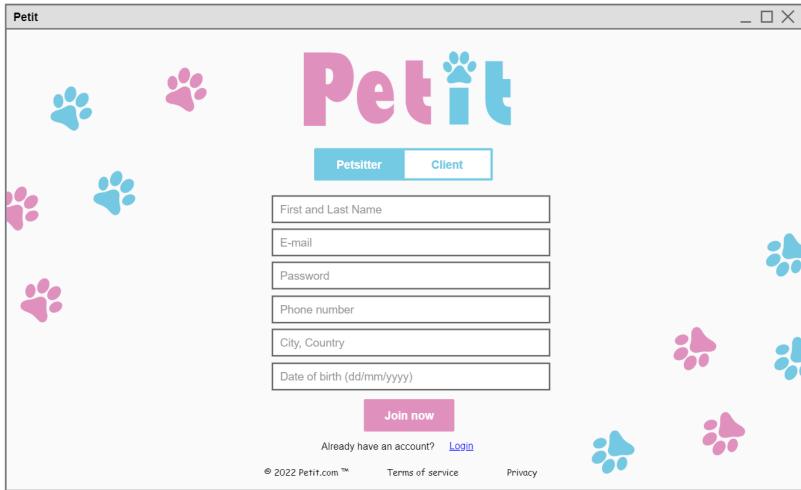


Figura 11: *Mockup - Registar petsitter*

2.6.3 *Login*

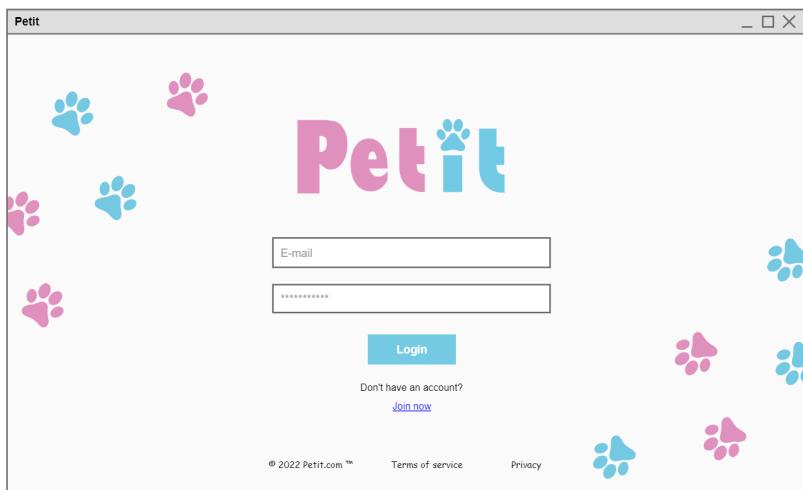


Figura 12: *Mockup - login*

Cliente

2.6.4 Perfil de cliente

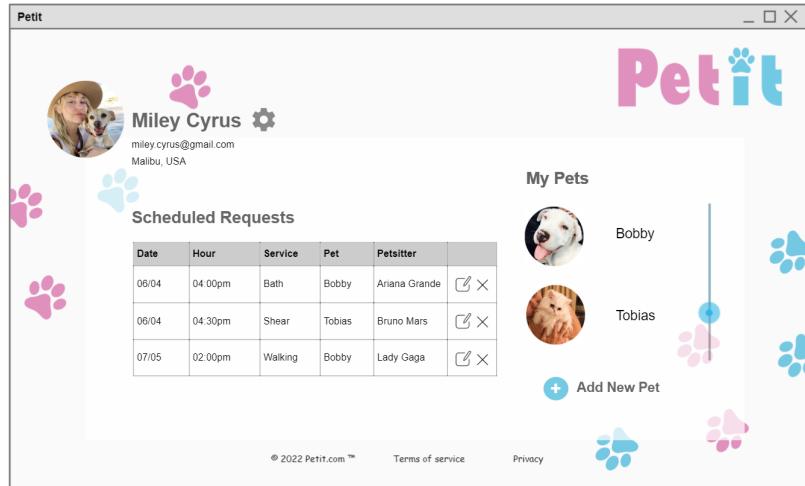


Figura 13: *Mockup - perfil de cliente*

2.6.5 Criar perfil de animal

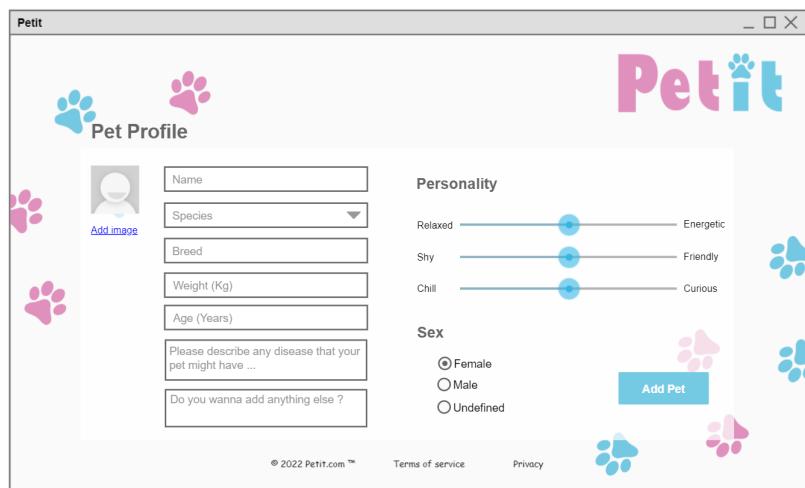


Figura 14: *Mockup - criar perfil de animal*

2.6.6 Página inicial do cliente

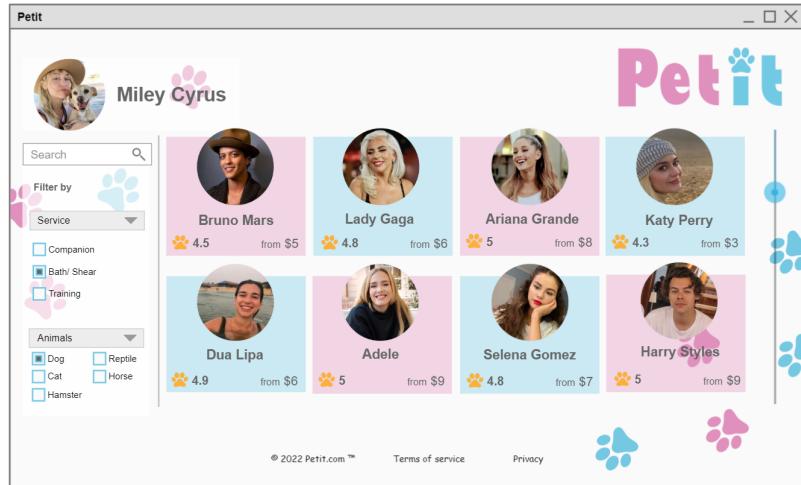


Figura 15: Mockup - página inicial do cliente

2.6.7 Consultar anúncio do *petsitter*

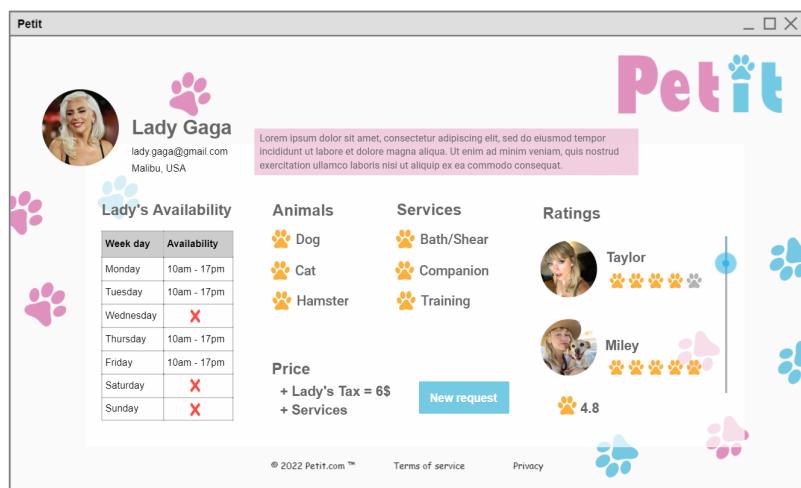


Figura 16: Mockup - perfil de *petsitter* visto por clientes

2.6.8 Realizar pedido

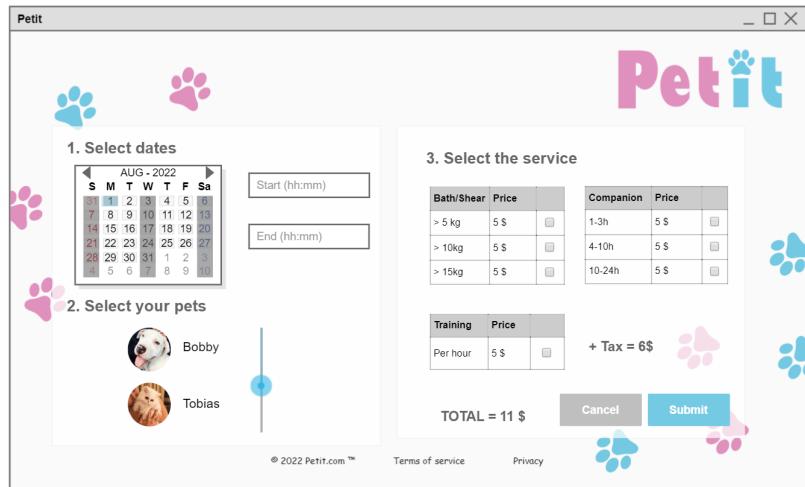


Figura 17: *Mockup* - realizar pedido

Petsitter

2.6.9 Perfil do *petsitter*

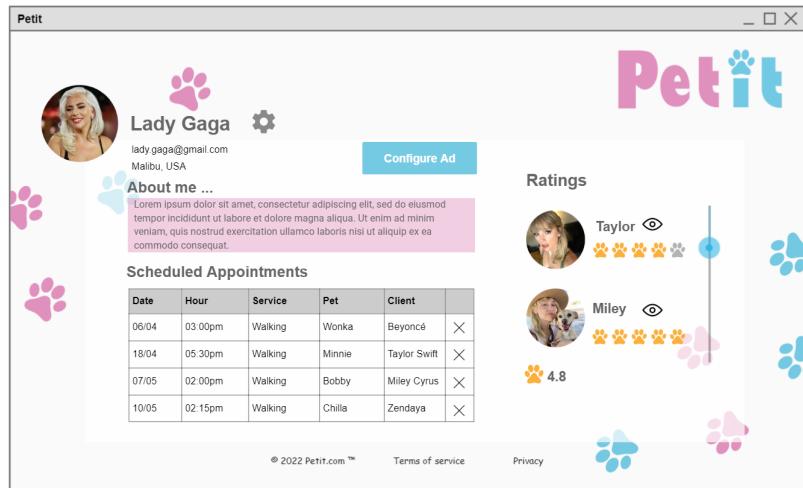


Figura 18: *Mockup - perfil de petsitter visto por si próprio*

2.6.10 Editar anúncio

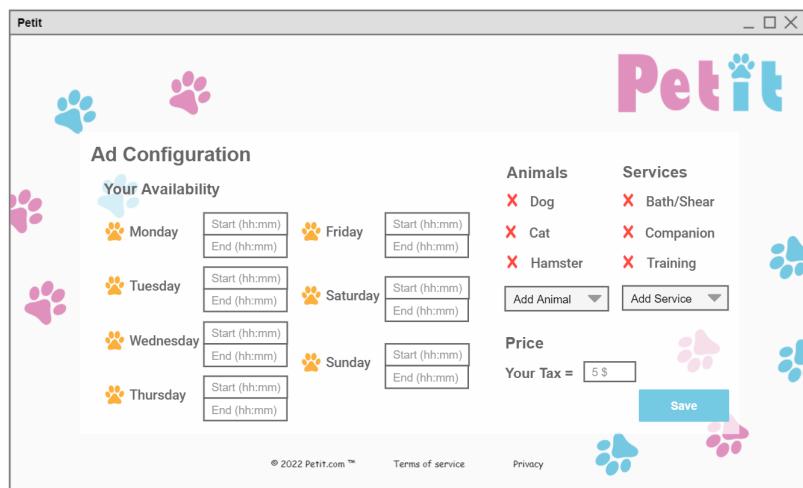


Figura 19: *Mockup - editar anúncio*

2.7 Máquina de estados

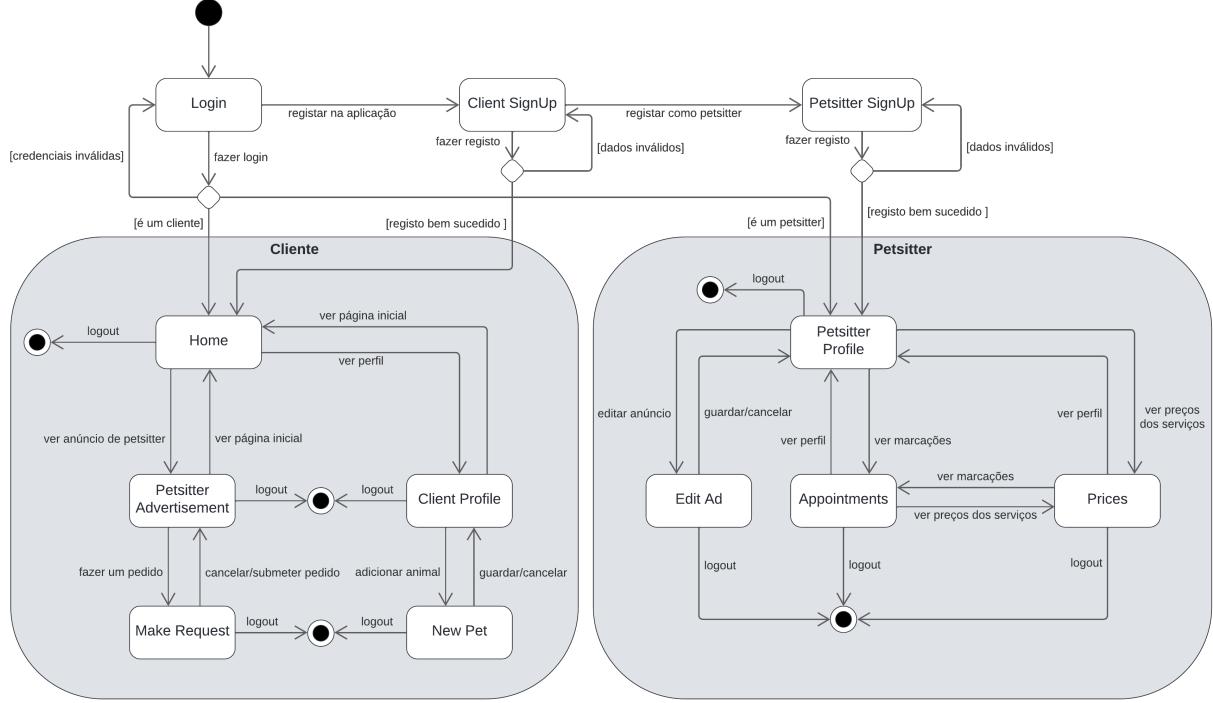


Figura 20: Máquina de Estados

Para definir o fluxo entre as várias *interfaces* da aplicação foi desenvolvido um diagrama máquina de estados.

Analizando a imagem apresentada de seguida, verifica-se que inicialmente é sempre apresentada a página de *login*. Caso o utilizador já esteja registrado na aplicação, pode fazer *login*, caso contrário deve efetuar o registo. Para isso, deve selecionar a opção de registo e escolher o seu tipo de utilizador (cliente ou *petsitter*). Tendo feito *login* ou registo, o utilizador é assim redirecionado para a sua página inicial, onde encontra diversas funcionalidades e um botão para fazer *logout* da aplicação. Em todos os momentos de interação do utilizador com o sistema, é possível que ele faça *logout* quando assim o pretender.

Caso o utilizador inicie sessão, com sucesso, como Cliente, vai encontrar na sua página inicial vários anúncios de vários *petsitters*. O cliente pode, assim, consultar um anúncio, e caso pretenda os serviços oferecidos por esse *petsitter*, pode fazer um pedido. Cancelando ou submetendo um pedido feito, volta à página onde estava – anúncio do *petsitter* selecionado. Voltando à página inicial, pode ainda consultar o seu próprio perfil, onde consegue adicionar um animal de estimação. Adicionando o animal ou cancelando essa ação, volta à página do seu perfil, onde poderá voltar à página inicial.

Por outro lado, caso o utilizador inicie sessão, com sucesso, como *Petsitter*, é redirecionado para o seu próprio perfil de prestador de serviços. Aqui tem várias opções para realizar

tarefas. Pode consultar as marcações que tem agendadas, e daí consultar os preços fixos estipulados para cada serviço que presta. Esta opção é também possível de ser selecionada a partir do perfil. Voltando à página inicial (perfil), através da consulta das marcações ou dos preços, tem ainda acesso à opção de editar o seu anúncio que é mostrado aos clientes. Guardando as alterações feitas ou cancelando essa ação, volta à página do seu perfil.

2.8 Diagrama de Componentes

Após o levantamento das responsabilidades do sistema, tanto através da exploração do domínio como também através da elaboração de *mockups* e modelos de tarefas que, por sua vez, permitiram uma melhor compreensão dos requisitos principais que o sistema deverá cumprir, foi necessária a determinação dos subsistemas que seriam usados no trabalho. Isto foi necessário pois a criação de subsistemas permite uma maior organização, uma vez que haverá o agrupamento dos métodos pelos subsistemas onde melhor se adequam. Além disto, esta prática é fundamental na questão do encapsulamento uma vez que cada subsistema implementa uma interface com os métodos que lhe foram atribuídos, impedindo isto o acesso ao seu "core" (o subsistema poderá conter métodos que não estão na interface e que não convém que "entidades" exteriores tenham acesso).

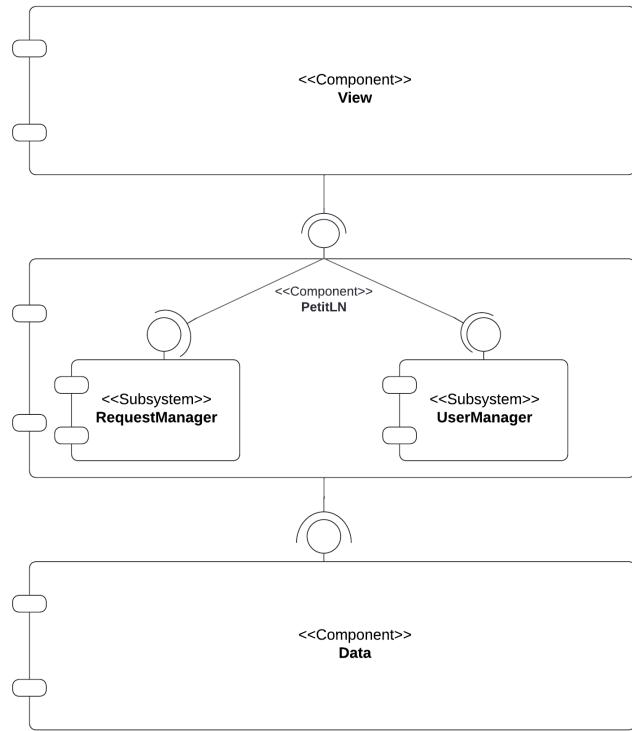


Figura 21: Diagrama de componentes

Componentes	Descrição
View	Componente responsável pela interface gráfica do utilizador.
PetitLN	Componente com a lógica de negócios da aplicação.
Data	Componente que representa a base de dados.

Subsistemas	Descrição
RequestManager	Subsistema da lógica de negócios encarregue de todas as funcionalidades que envolvem os utilizadores da aplicação.
UserManager	Subsistema da lógica de negócios encarregue de todas as funcionalidades relacionadas com anúncios e pedidos da aplicação.

2.9 Diagrama de Classes

Em seguida será apresentado o diagrama de classes PSM do sistema a ser desenvolvido. Com ele pretende-se destacar como os diferentes componentes do sistema se vão relacionar entre si, bem como o tratamento de cada entidade abordada.

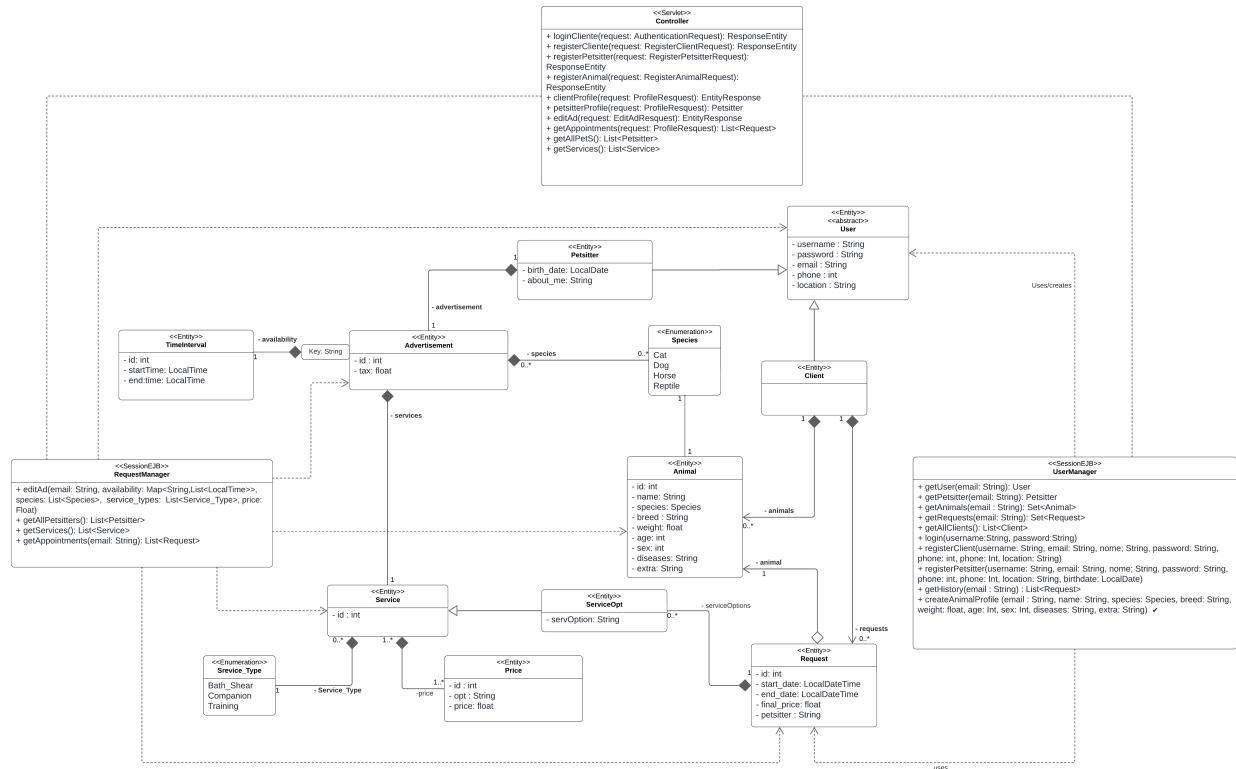


Figura 22: Diagrama de classes

Um dos fatores mais importantes do sistema é o utilizador que vai fazer uso da aplicação. Como já foi mencionado anteriormente, temos o *Client* e o *Petsitter*, duas subclasses da classe mãe *User*. O *User* guarda todas as informações comuns aos dois tipos de utilizador, e cada um deles especifica depois os seus atributos que não estão em comum.

Um *Client* pode ter mais do que um Animal, que pertence a uma espécie (*Species*) suportada pela aplicação. O *Client* pode também realizar os pedidos que quiser, sendo que cada *Request* é feito apenas para um dos animais do utilizador.

Um *Request* pode ser realizado para mais do que um serviço desde que ao mesmo animal. É escolhida a opção de serviço desejada (*ServiceOpt*, consoante peso do animal, por exemplo) e associada ao serviço correspondente. Cada *Service* apresenta o seu *Service_Type* e o *Price*, fixo e determinado pela equipa desenvolvedora da plataforma.

Para além do cliente que pode realizar pedidos, temos ainda o *Petsitter*, que atende esses pedidos. O *Petsitter* publica o seu *Advertisement*, especificando as espécies que aceita tratar e os serviços que pretende oferecer, bem como a sua disponibilidade semanal, com os *TimeInterval* em que está disponível.

De forma a podermos modelar a interação de todas as entidades e relações, e incluir a gestão do sistema de base de dados, apresentamos no diagrama o *package Beans*, onde estão as classes que fazem as operações de negócio na camada de dados da aplicação.

2.10 Modelo lógico da base de dados

A nossa base de dados foi gerada através de anotações do **Hibernate**, recorrendo ao **Microsoft SQL Server** e localizada no **Microsoft Azure**, e pode ser ilustrada através do seu modelo lógico, que se encontra abaixo:

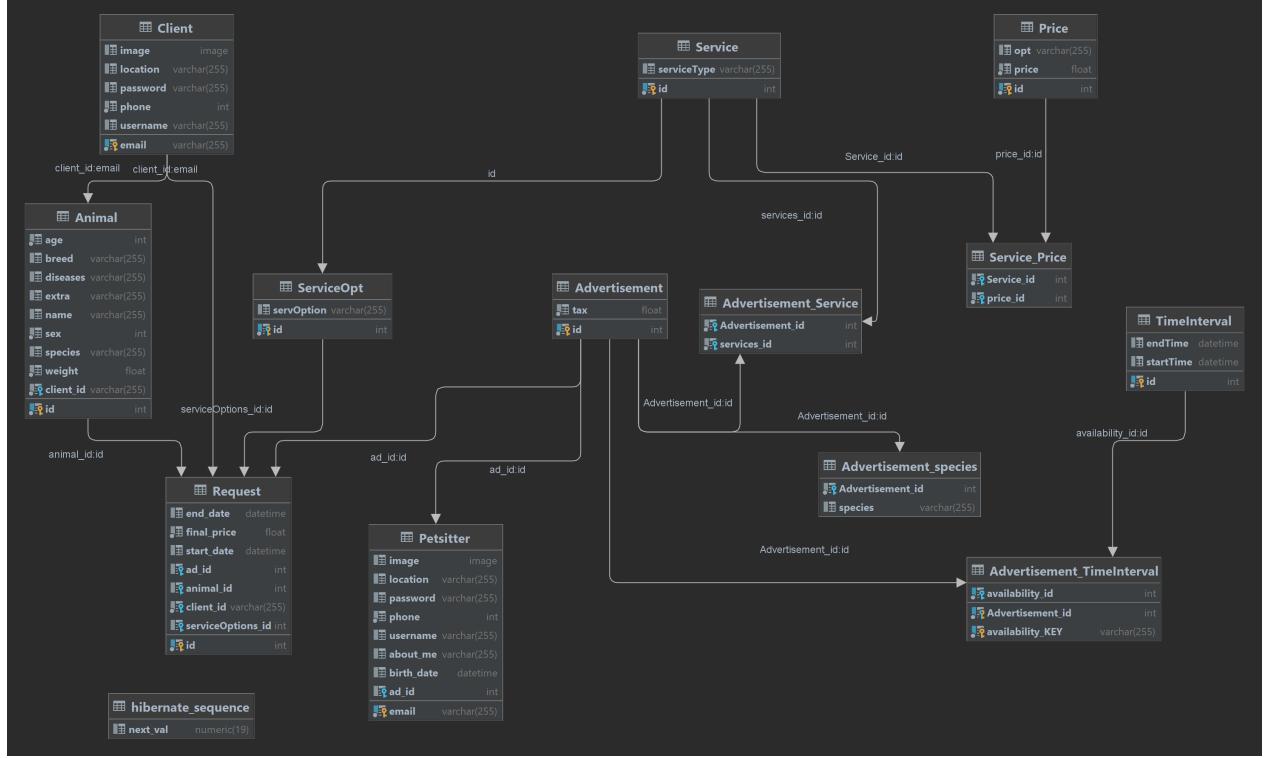


Figura 23: Modelo lógico

Analisando o modelo lógico, podemos verificar que existem 14 tabelas que passamos agora a resumir. Começando pela tabela do **Cliente**, podemos ver que ele contém atributos referentes ao seu nome de utilizador, número de telemóvel, *password*, localização e foto de perfil. Cada cliente é identificado pelo seu *email*, sendo esta a sua chave primária.

Passando pela tabela do **Animal**, podemos ver que tem os seguintes atributos: nome, idade, espécie, raça, sexo, peso, doenças e alguma informação extra. Para além disso, a sua chave primária é um id, e contém uma chave estrangeira com o id do cliente dono do animal.

Olhando agora para a tabela dos pedidos (**Request**), podemos ver que cada pedido é identificado por um id e contém os atributos referentes ao preço final, a data de início e fim, e ainda chaves estrangeiras com os ids do cliente que faz o pedido, do seu animal, do anúncio do *petsitter* e do serviço escolhido.

Em relação ao **Service**, esta tabela é identificada por um id e contém um só atributo: o tipo de serviço a escolher. Para além disso, temos a tabela **ServiceOpt**, que se refere à escolha do serviço por parte do cliente, contendo a opção como atributo e sendo identificada

por um id. Ainda em relação aos serviços, temos a tabela **Service_Price** que apenas contém as chaves estrangeiras para as tabelas do *Service* e do *Price*, tratando assim de relacionar as duas tabelas. A tabela **Price**, identificada por um id, contém as várias opções de horário e o respetivo preço.

Para além disso, temos a tabela que representa um anúncio (**Advertisement**), que têm um id como chave primária, e contém o atributo referente à taxa do *petsitter*. Para além desta, temos a tabela **Advertisement_Service** que relaciona as tabelas do anúncio e do serviço, contendo apenas os ids destas como chaves estrangeiras. Temos também a tabela **Advertisement_species**, que contém como atributo as espécies para qual o anúncio está disponível, e como chave estrangeira o id da tabela do anúncio respetivo.

Podemos encontrar ainda a tabela referente ao **Petsitter**, que, tal como o cliente, é identificado pelo seu *email*, e contém como atributos o seu nome de utilizador, número de telemóvel, *password*, localização, data de nascimento, foto de perfil e o texto incluído na secção "*About me*". A tabela do *petsitter* contém ainda o id da tabela do seu anúncio.

Para terminar, temos a tabela **Advertisement_TimeInterval** que relaciona as tabelas do anúncio e do intervalo de tempo, contendo os ids destas como chaves estrangeiras, e ainda uma *key* como chave primária. Nesta tabela, o id da tabela do anúncio serve também como chave primária. A tabela do intervalo de tempo (**TimeInterval**), que ainda não foi mencionada, é identificada por um id e contém a data de início e fim do intervalo de tempo.

Existe ainda a tabela **hibernate_sequence**, que é gerada automaticamente pelo *Hibernate*.

3 Arquitetura da Aplicação

Terminada a fase de modelação, passamos ao desenvolvimento da aplicação. Como primeiro passo, começamos por escolher quais as tecnologias e *frameworks* que vamos utilizar para o desenvolvimento da aplicação.

Assim sendo, o grupo começou por traçar a arquitetura da aplicação. Posto do princípio que iríamos implementar uma arquitetura MVC, seriam precisas à partida 3 tecnologia: uma direcionada ao *frontend*, uma para a lógica de negócio e uma terceira para a camada de dados.

Começando pela camada de dados, a escolha do **Hibernate** foi direta, dada a sua grande utilização no mercado e visto que temos vindo a trabalhar com a mesma na UC de Arquiteturas Aplicacionais.

Já para a lógica de negócio, depois de uma pesquisa sobre as várias *frameworks* que poderíamos utilizar no projeto, optamos por utilizar **Spring**, devido a fatores como a sua importância no mercado, à sua curva de aprendizagem e a existirem elementos da equipa que já tinham experiência com a mesma.

Já para o *frontend*, recorremos à utilização do **Vue.js**, devido à sua baixa curva de aprendizagem e a termos tido uma breve introdução à mesma na UC de Sistemas Interativos Confiáveis.

Assim, para sumarizar, a arquitetura que nos propomos implementar é a seguinte:

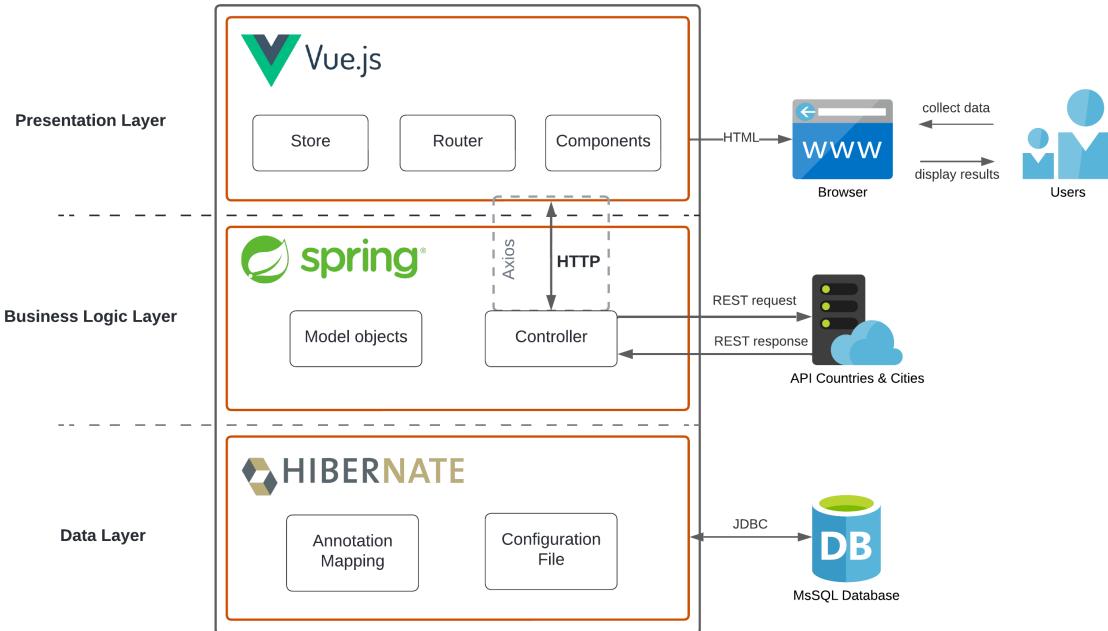


Figura 24: Diagrama da Arquitetura

4 Implementação

Antes de mais, foi necessário escolher a linguagem para programar na camada da lógica de negócio, para tal o grupo escolheu o **Java**, por se tratar de uma linguagem estável, robusta e bastante utilizada na indústria e em aplicações *web*. É de notar que o grupo, dividiu o sistema em duas aplicações, uma destinada ao **backend** e outra ao **frontend**, cada uma a correr na sua porta destinada. Por este meio, foi possível que o grupo trabalhasse assincronamente em cada uma das partes do trabalho, uma vez que numa fase inicial ambas estavam independentes uma da outra. Além disso, permite que futuramente, caso o grupo ache necessário, seja possível mudar a tecnologia de qualquer uma das camadas sem prejudicar as restantes. É também de realçar que este tipo de arquitetura implementada permite aumentar a capacidade de **escalabilidade** da aplicação fazendo réplicas apenas de partes da aplicação, tal como por exemplo, do *backend*, neste modo, é evitada a replicação total da aplicação.

Nas secções seguintes será abordado em maior detalhe o processo de desenvolvimento.

4.1 *Frontend*

Para o desenvolvimento da camada de apresentação optamos a *framework* **Vue.js**. Houve a preocupação de manter esta camada completamente independente das camadas inferiores, pelo que só realiza comunicação com estas através de uma API criada no *backend* para esse efeito.

4.1.1 *Componentes*

Para cada página e conteúdo repetido em várias páginas, como a barra de navegação, foram criadas componentes independentes que podem por isso ser reutilizadas. Em seguida, serão listadas as componentes desenvolvidas.

- *AppPet* - página de criar o perfil de um animal
- *Advertisement* - página de consulta de um anúncio
- *Appointments* - página em que o *petsitter* pode consultar os seus pedidos
- *ClientProfile* - página do perfil de cliente
- *EditAd* - página onde se edita um anúncio
- *Home* - página inicial
- *Login* - página do *login*
- *MakeRequest* - página onde o cliente faz um pedido
- *Nav* - barra de navegação
- *PetsitterProfile* - página do perfil do cliente

- *Prices* - página onde o *petsitter* pode consultar os preços estabelecidos pela plataforma para os diferentes serviços
- *SignupClient* - página de registo do cliente
- *SignupPetsitter* - página de registo do *Petsitter*

4.1.2 Router

Uma das funcionalidades que o *Vue.js* fornece é a utilização de *router*, que permite navegar entre páginas de forma fácil, associando a cada componente um *path*. Para além disto, o *router* possibilita que seja criado um histórico de páginas pelas quais o utilizador navegou, assim como proibir certos utilizadores de aceder a determinadas páginas, como por exemplo, utilizadores não autenticados acederem à página inicial da aplicação.

4.1.3 Store

Esta componente do *Vue* pode ser vista como um *container* que guarda o estado da aplicação, como, por exemplo, informações relativas ao utilizador atualmente autenticado. Adicionalmente, também é na *stores* que se realiza a maioria das comunicações com a API do *backend*.

4.1.4 Pedidos à API

Para efetuar pedidos à API do *backend*, utilizamos o *package axios*, que permite definir o *endpoint*, definir o tipo de método (*post*, *get*, ...), enviar o pedido e receber a resposta, se for caso disso.

4.2 Backend

Para conectar o *frontend* com o *backend*, implementamos uma **Rest API**, de forma a responder a qualquer pedido que seja feito pelo cliente no *frontend*. Desta forma, estes componentes são independentes e apenas utilizam uma API conhecida para comunicar.

4.2.1 Controllers

Como já foi referido, a *framework* utilizada para a implementação da lógica de negócios foi a *Spring*, na qual os *HTTP Request* são tratados por **Rest Controllers**. A equipa procedeu então à implementação de um *controller* que recebe todos os pedidos do *frontend*, faz o seu mapeamento no *backend* (*session beans*), de forma a obter a resposta adequada, e reencaminha-a para o *frontend*.

4.2.2 Beans

Entity beans

Os *entity beans* representam entidades guardadas em suporte persistente, armazenadas na base de dados. Os *entity beans* presentes na aplicação são:

- *Advertisement*: Classe que representa a entidade de anúncios de um *petsitter*.
- *Animal*: Classe que representa a entidade animal.
- *Client*: Classe que representa a entidade cliente da aplicação.
- *Petsitter*: Classe que representa a entidade *petsitter* da aplicação.
- *Request*: Classe que representa a entidade de um pedido que um cliente pode registar na aplicação.
- *Service*: Classe que representa a entidade serviço da aplicação.
- *ServiceOpt*: Subclasse do *Service*, utilizada para registar o serviço de um pedido.
- *TimeInterval*: Classe que representa um intervalo de tempo.
- *User*: Superclasse que representa um utilizador da aplicação, podendo este ser cliente ou *petsitter*.

Session beans

O outro tipo de EBJ implementado foi **Session beans**. Estes gerem processos ou tarefas, implementam a lógica de negócio, estabelecendo os relacionamentos entre as várias entidades do sistema e fazendo pedidos aos DAOs que comunicam com a base de dados.

Depois de deliberar sobre quantos *session beans* deveriam ser utilizados e quais as responsabilidades de cada um deles, a equipa optou por implementar apenas dois, que serão apresentados a seguir:

- *UserManager*: Classe responsável por todas as operações relacionadas com utilizadores do sistema, sejam estes clientes ou *petsitters*. Algumas das operações implementadas neste *bean* são: o registo e *login* de um utilizador, listagem de utilizadores e o registo de um animal.
- *RequestManager*: Classe responsável por todas as operações relativas aos anúncios e pedidos da aplicação. Algumas das operações que implementam são a configuração de um anúncio e a consulta do histórico de pedidos de um utilizador.

4.2.3 Acesso à API externa

No nosso trabalho, implementamos ainda uma funcionalidade que depende de uma API *REST* externa. Sendo que a API encontrada se trata de uma base de dados de todos os países do mundo e das suas cidades, foi fácil decidir onde a iríamos utilizar. Para tal, implementamos no registo, de um cliente ou *petsitter*, dois campos referentes ao país e à cidade, que fazem pedidos à API externa. Estas informações serão depois concatenadas numa só *string* que representa a localização do cliente ou *petsitter*.

4.2.4 Base de dados

Relativamente à camada de dados, como já foi dito anteriormente, optamos por utilizar o **Hibernate**, uma ferramenta ORM *open source*. Esta ferramenta é a mais reconhecida no mercado, dado o seu alto desempenho e confiabilidade. A configuração do *Hibernate* pode ser feita utilizando ficheiros de configuração ou anotações. A equipa decidiu utilizar anotações devido a considerar que é a abordagem mais intuitiva.

A base de dados do projeto recorre ao *Microsoft SQL Server* e encontra-se localizada na plataforma ***Microsoft Azure***.

Na lógica de negócio, todos os acessos aos dados localizados na base de dados são feitos através de **DAOs** geridos pelo *Hibernate*.

5 Deployment

Depois da implementar do sistema, iniciou-se a fase de deploy da aplicação para disponibilizar a mesma para o uso dos utilizadores. Neste tópico serão abordadas as decisões tomadas relativamente à instalação, tanto do *Frontend*, como do *Backend*. Consequentemente, será também explicado o modo como se fez a conexão entre os diferentes componentes da aplicação.

5.1 Considerações iniciais

Como se pode verificar no diagrama de deployment, a instalação da aplicação foi feita em dois dispositivos distintos.

Em primeiro lugar, a base de dados foi instalada nos servidores da Microsoft Azure, sendo que desta forma ficará sempre disponível a qualquer momento a partir de qualquer dispositivo.

Por outro lado, tanto o *frontend* como o servidor aplicacional foram instalados no dispositivo do utilizador. É de notar que a conexão entre o *frontend* e o *backend* é feita via pedidos HTTP. Para além disso, a acesso à base de dados realizou-se através do *driver JDBC* do *SQL Server*.

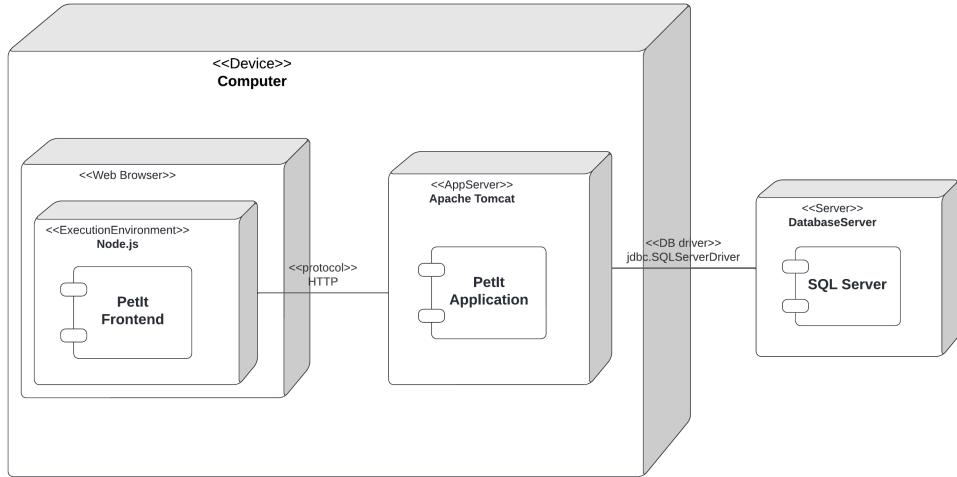


Figura 25: Diagrama de Deployment

5.2 Frontend

Relativamente ao deployment do *Frontend*, recorreu-se à utilização do runtime Node.js. Desta forma, foi assim possível instalar a aplicação do Vue.js num servidor que fica à escuta de pedidos do browser na porta 8081. Do mesmo modo, para conseguir enviar informações para o *backend* utilizou-se a *Axios*.

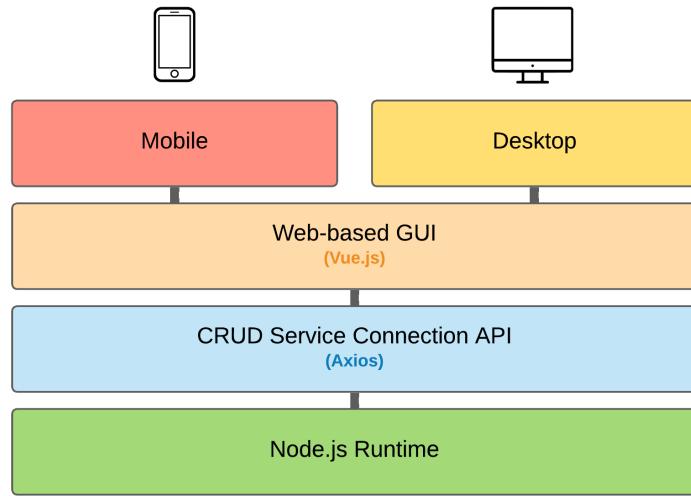


Figura 26: Deployment do *Frontend*

5.3 *Backend*

Relativamente ao Backend, este foi instalado em cima de um sevidor web java. O servidor escolhido foi o Apache Tomcat, sendo que este é constituído por um container de servlets. Desta forma, o backend fica à escuta de pedidos do *Frontend* na porta 8080.

6 Aplicação Desenvolvida

6.1 Login

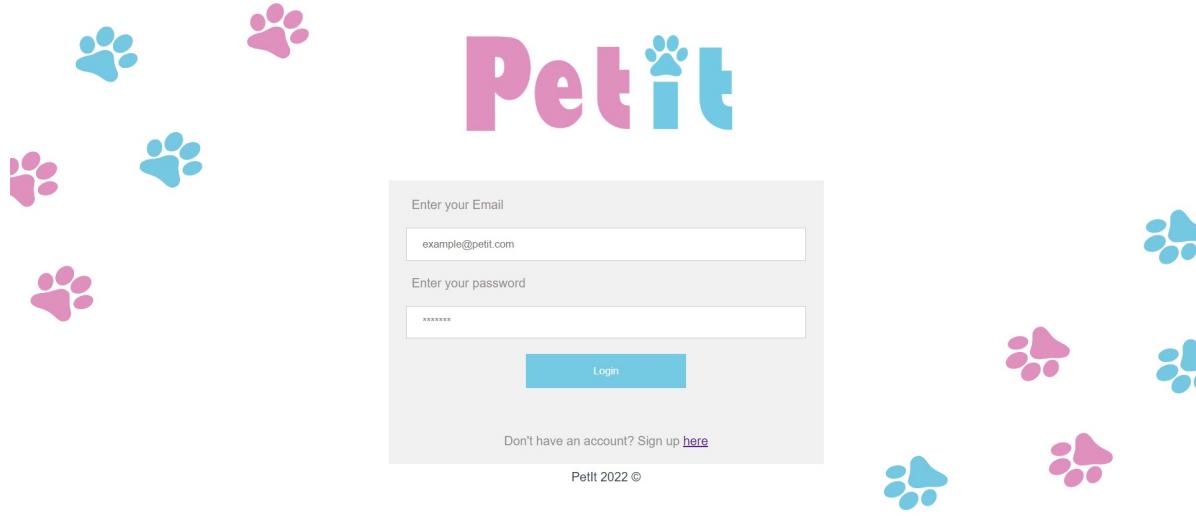


Figura 27: Login

6.2 Login erro

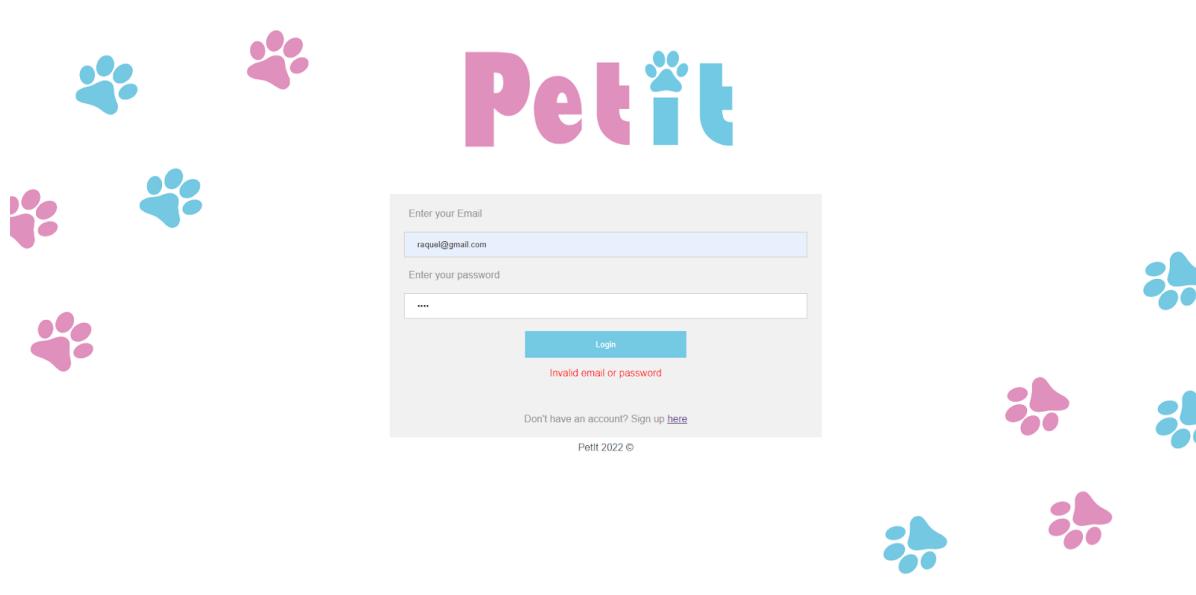


Figura 28: Login erro

6.3 Registo de cliente

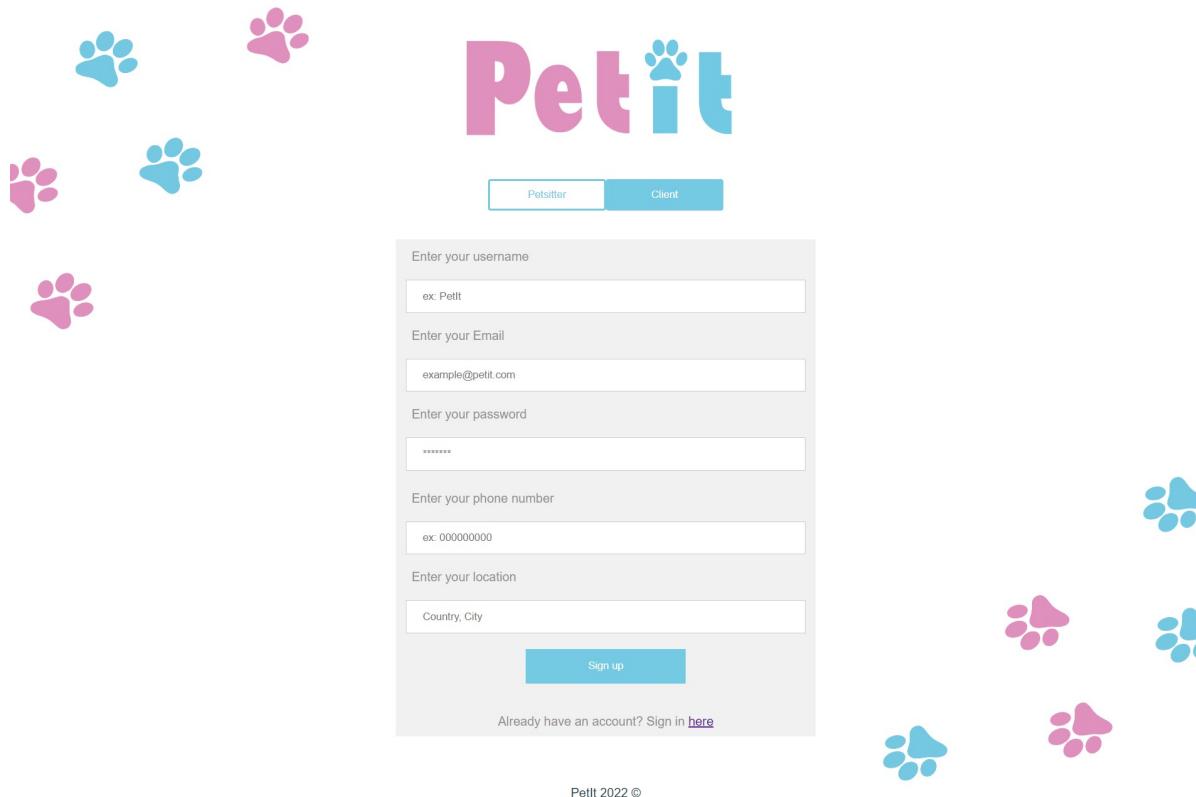


Figura 29: Registo de cliente

6.4 Registo de petsitter

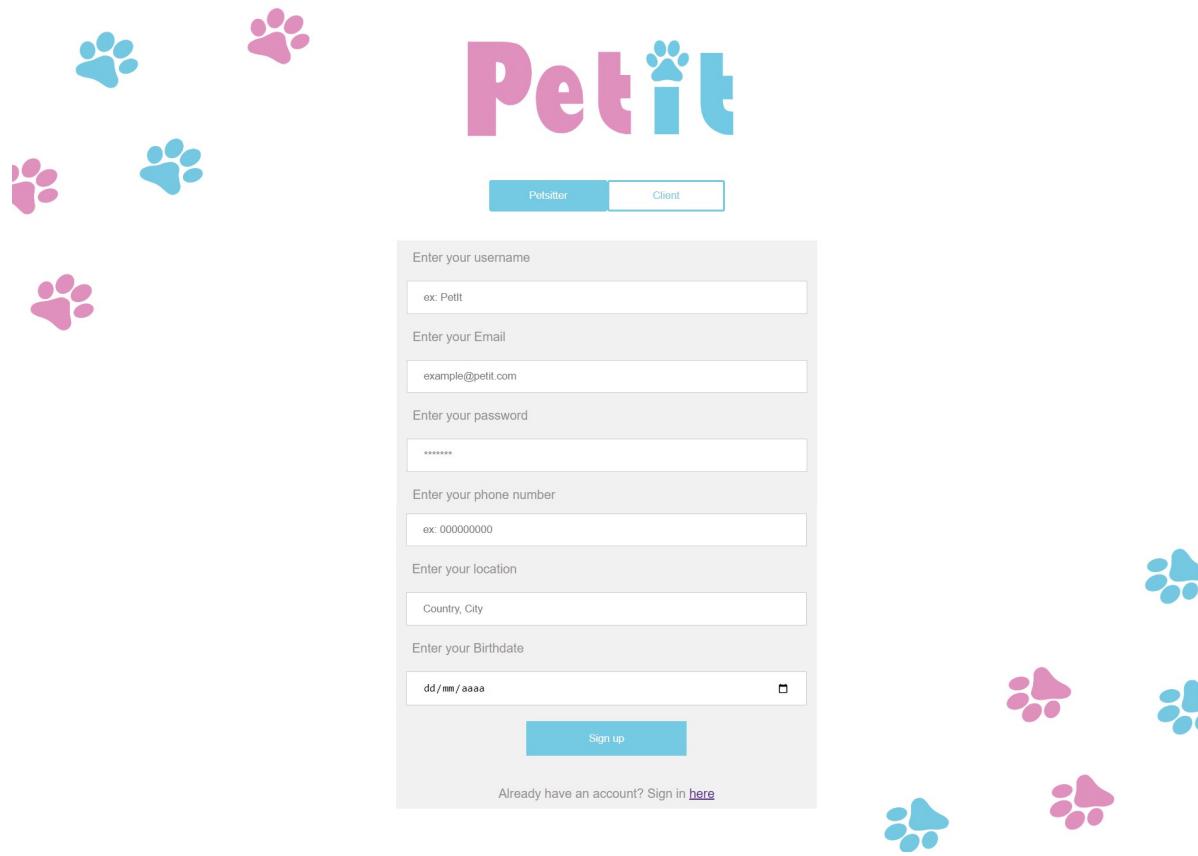


Figura 30: Registo de petsitter

6.5 Registo - Escolha do país

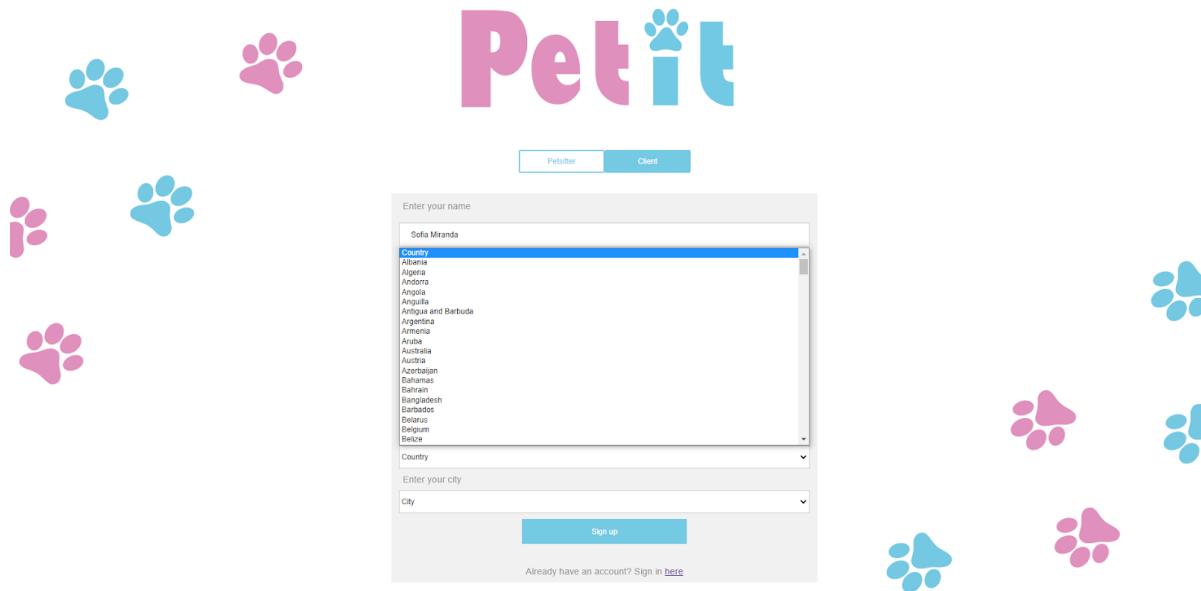


Figura 31: Registo - escolha do país

6.6 Registo - Escolha da cidade após escolhido o país

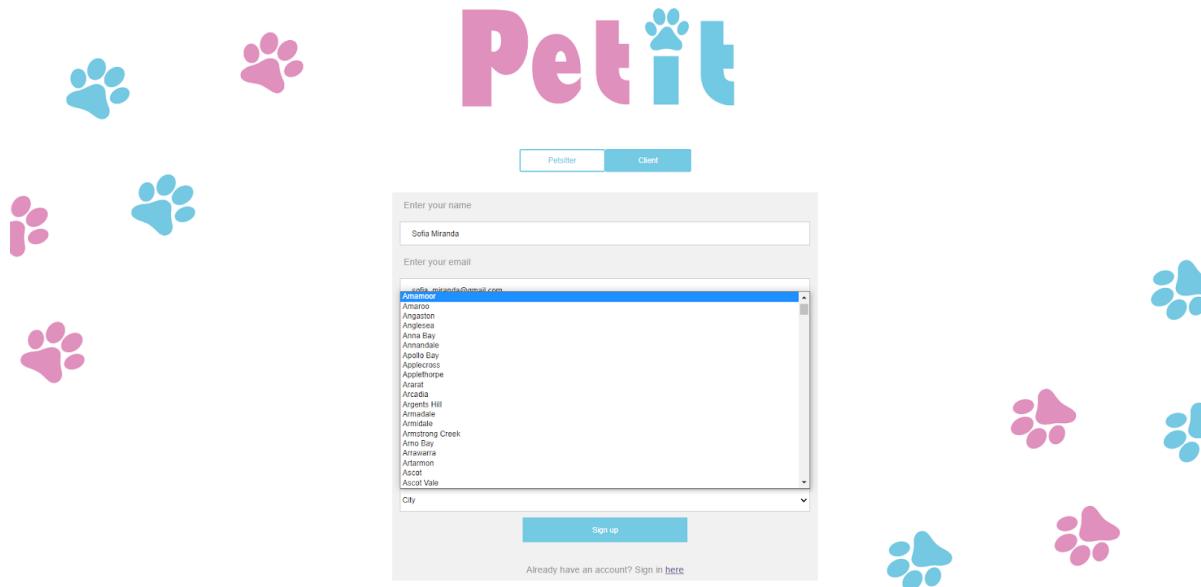


Figura 32: Registo - escolha da cidade

6.7 Registo - Campo vazio

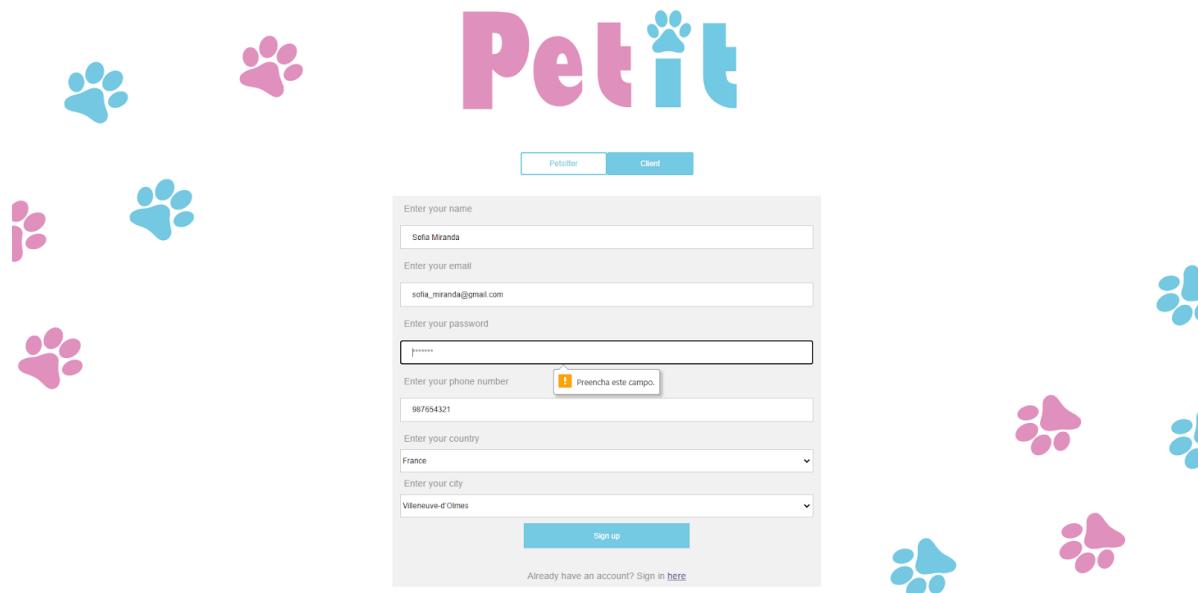


Figura 33: Registo - campo vazio

6.8 Registo - menor de idade

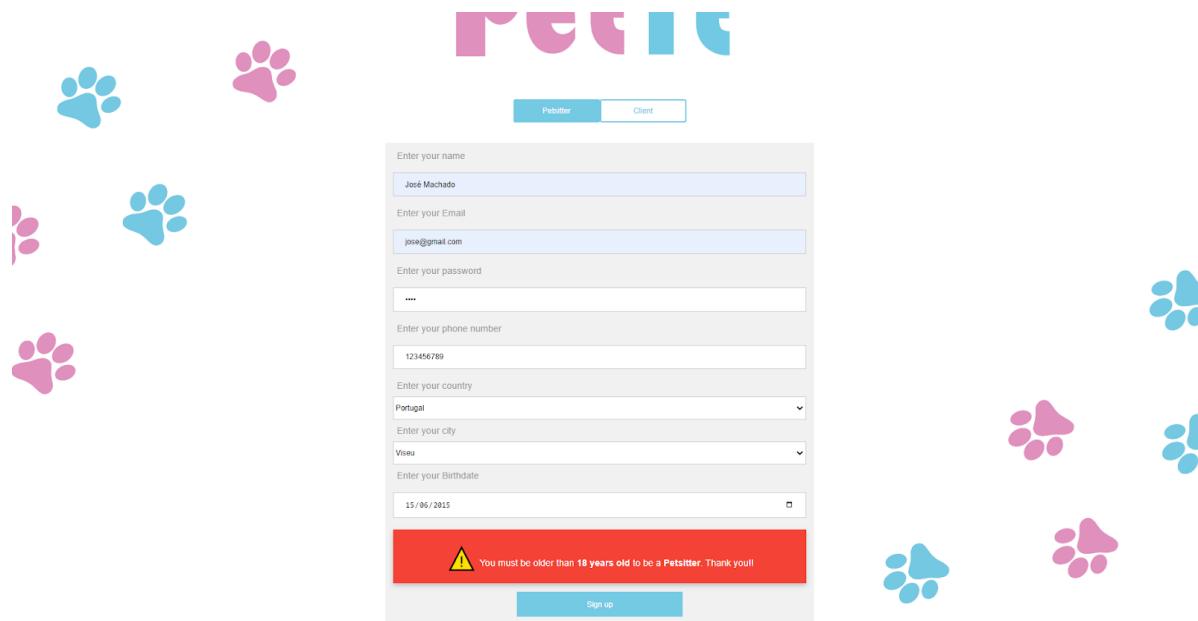


Figura 34: Registo - menor de idade

6.9 Página inicial

The screenshot shows the homepage of the Petit platform. At the top right is the logo "Petit". Below it is a navigation bar with "Home" and "Profile" buttons, and a "Logout" link. On the left, there is a sidebar titled "Filter by" with sections for "Services" (Companion, Bath/Shear, Training) and "Animals" (Dog, Cat, Reptile, Horse). Below this is a "Sort by" section with dropdown menus for "Name" and "Price". To the right, there is a grid of service provider profiles:

- Beatriz da Rosa**: Tax: 10.9€, Services: Companion, Training. Buttons: "Select", "About me" (with a photo).
- Baltasar Torquato**: Tax: 5.91€, Services: Training, Bath and Shear, Companion. Buttons: "Select", "About me" (with a photo).
- Gustavo da Costa**: Tax: 9.49€, Services: Companion. Buttons: "Select", "About me" (with a photo).
- Daniela Dinis**: Tax: 13.7€, Services: Companion. Buttons: "Select", "About me" (with a photo).
- Aman Muniz**: Tax: 14.19€, Services: Bath and Shear, Training. Buttons: "Select", "About me" (with a photo).
- Cláudia Lameira**: Tax: 8.96€, Services: Companion. Buttons: "Select", "About me" (with a photo).

Figura 35: Página inicial

6.10 Anúncio

The screenshot shows a detailed profile page for "Beatriz da Rosa". At the top right is the "Petit" logo. Below it is a navigation bar with "Home" and "Profile" buttons, and a "Logout" link. The profile picture of Beatriz da Rosa is shown, along with her name and contact information: Portugal, Évora, phone number 922939728, and email beatrizdarosa@gmail.com. To the right is a "About me" section with a message: "Hi, My name is Beatriz da Rosa and I love animals!". Below this are four sections: "Week day" (a table showing availability from Monday to Sunday), "Animals" (listing Dog, Cat, Horse), "Services" (listing Companion, Training), and "Prices" (+ My tax: 10.9 €, + Service price). At the bottom right is a "Make request" button.

Week day	Availability
Monday	09.00 - 20.00
Tuesday	09.00 - 20.00
Wednesday	09.00 - 20.00
Thursday	09.00 - 20.00
Friday	09.00 - 20.00
Saturday	Closed
Sunday	Closed

Figura 36: Anúncio

6.11 Realizar um pedido

The screenshot shows the 'Petit' app's user interface for creating a service request. At the top, there's a profile icon for 'Helena Cardoso'. Below it, a navigation bar with 'Home' and 'Profile' buttons, and a 'Logout' link on the right.

Make your request:

- Select the date**

Date	Time
Start: 30/06/2022	09:50
End: 30/06/2022	13:00
- Select your pet(s)**
 - Roberto (Dog icon)
 - Tobias (Cat icon)
 - Teresa (Horse icon)
- Select your service**

Companion Training

No. of Hours	Price
1-3	3 €
4-12	5 €
12-24	8 €

+ tax = 10.9 €

Total = 13.9 €

Buttons:

Figura 37: Realizar um pedido

6.12 Perfil do cliente

The screenshot shows the 'Petit' app's user interface for the client profile. At the top, there's a profile icon for 'Helena Cardoso'. Below it, a navigation bar with 'Home' and 'Profile' buttons, and a 'Logout' link on the right.

Scheduled Requests

Date	Start time	End time	Service	Pet	Petsitter
2022-06-28	09:00:00	12:00:00	Training	Tobias	Aman Muniz
2022-06-21	09:00:00	12:00:00	Companion	Roberto	Beatriz da Rosa
2022-06-17	10:00:00	13:00:00	Companion	Roberto	Beatriz da Rosa
2022-06-20	09:30:00	11:00:00	Companion	Roberto	Baltasar Torquato
2022-06-22	15:00:00	18:00:00	Companion	Roberto	Daniela Dinis
2022-06-20	09:00:00	12:00:00	Companion	Roberto	Daniela Dinis

My Pets

Pet
Teresa
Roberto
Tobias

Figura 38: Perfil do cliente

6.13 Criar um perfil do animal

The screenshot shows the 'Petit' app's profile creation screen. At the top, there's a user icon and the name 'Helena Cardoso'. Below that is a navigation bar with 'Home' (selected), 'Profile', and 'Logout'. On the left, there's a placeholder for a 'Pet photo' with a dog silhouette icon. The main form fields include:

- Name ***: ex. Bobby (input field)
- Age ***: Years: [] Months: [] (input fields)
- Species ***: [dropdown menu] (Species dropdown)
- Diseases**: Please describe any disease that your pet might have (text area)
- Breed**: ex.: Husky (input field)
- Note**: Do you want to add anything else? (text area)
- Sex ***: [dropdown menu] (Sex dropdown)
- Weight ***: ex: 5.0 (input field)

At the bottom right are two buttons: 'X Cancel' and '+ Add Pet'.

Figura 39: Criar um perfil do animal

6.14 Criar um perfil do animal - campo vazio

This screenshot shows the same profile creation screen as Figure 39, but with different input values and validation messages:

- Name ***: Sammy (input field)
- Age ***: 3 [] 3 [] (input fields)
- Species ***: Cat (dropdown value)
- Diseases**: none (text area)
- Breed**: siames (input field)
- Note**: Do you want to add anything else? (text area)
- Sex ***: Male (dropdown value)
- Weight ***: An input field containing 'ex: 5.0' has a yellow warning message box over it that says 'Preencha este campo.' (Fill this field.)

At the bottom right are 'X Cancel' and '+ Add Pet' buttons.

Figura 40: Criar um perfil do animal - campo vazio

6.15 Perfil do petsitter

The screenshot shows a user profile for Daniela Dinis. At the top, there is a placeholder for a profile picture, followed by the name "Daniela Dinis" and a paw print icon. Below the name are three navigation tabs: "Profile" (highlighted in blue), "Appointments" (grey), and "Prices" (grey). On the right, there is a "Logout" link and an "Edit Ad" button.

Contacts

- Portugal, Vila Real
- 986898845
- Email: danieladinis@gmail.com

About (Edit)

Hi,
My name is Daniela Dinis and I love animals!
Welcome to my advertisement...

Your Availability

Week day	Availability
Monday	09:00 - 20:00
Tuesday	09:00 - 20:00
Wednesday	12:00 - 18:00
Thursday	16:00 - 23:59
Friday	00:00 - 07:00
Saturday	Closed
Sunday	Closed

Your Advertisement

Animals	Services	Total Price
Dog Cat Reptile	Companion Bath_Shear	+ My tax: 13.7 € + Service price

PetIt 2022 ©

Figura 41: Perfil do Petsitter

6.16 Editar anúncio

The screenshot shows the "Ad Configuration" section for Letícia Azevedo. At the top, there is a placeholder for a profile picture, followed by the name "Letícia Azevedo" and a paw print icon. Below the name are three navigation tabs: "Profile" (grey), "Appointments" (highlighted in blue), and "Prices" (grey). On the right, there is a "Logout" link and a "Save" button.

Ad Configuration

Your availability:

Day	Start Time	End Time
Monday	09:00	20:00
Tuesday	09:00	20:00
Wednesday	12:00	18:00
Thursday	16:00	23:59
Friday	00:00	07:00
Saturday	-	-
Sunday	-	-

Animals

- Dog
- Cat
- Reptile
- Horse

Services

- Companion
- Bath/Shear
- Training

Price

Your tax:

PetIt 2022 ©

Figura 42: Editar anúncio

6.17 Pedidos de um Petsitter

The screenshot shows the PetIt platform interface. At the top, there's a user profile for 'Daniela Dinis' with a black silhouette icon. Below the profile are navigation tabs: 'Profile' (disabled), 'Appointments' (selected, highlighted in teal), and '\$ Prices'. On the far right is a 'Logout' button. The main content area is titled 'Scheduled Appointments' and displays a table of upcoming services. The table has columns for Date, Start time, End Time, Service, Pet, and Client. The data shows multiple entries for June 2022, mostly for 'Companion' services involving pets named Teresa, Roberto, Tobias, and Benjamin, with clients Helena Cardoso and Rosa Maria. To the right of the table is a 'Filter by' sidebar with fields for 'Search for clients...', 'Service' (set to 'Companion'), and a date range selector ('dd/mm/aaaa'). The background features decorative paw prints in pink and blue.

Date	Start time	End Time	Service	Pet	Client
2022-06-16	16:45:00	18:00:00	Bath_Shear	Teresa	Helena Cardoso
2022-06-20	09:00:00	12:00:00	Companion	Roberto	Helena Cardoso
2022-06-20	12:00:00	16:00:00	Bath_Shear	Tobias	Helena Cardoso
2022-06-20	16:48:00	18:48:00	Companion	Teresa	Helena Cardoso
2022-06-20	19:00:00	20:00:00	Bath_Shear	Roberto	Helena Cardoso
2022-06-21	14:00:00	17:00:00	Companion	Roberto	Helena Cardoso
2022-06-22	15:00:00	18:00:00	Companion	Roberto	Helena Cardoso
2022-06-23	22:00:00	07:00:00	Companion	Roberto	Helena Cardoso
2022-06-28	12:00:00	15:00:00	Companion	Benjamin	Rosa Maria
2022-06-27	12:00:00	13:00:00	Companion	Roberto	Helena Cardoso

PetIt 2022 ©

Figura 43: Pedidos de um Petsitter

6.18 Preços estabelecidos pela plataforma

The screenshot shows the PetIt platform interface. At the top, there's a user profile for 'Daniela Dinis' with a black silhouette icon. Below the profile are navigation tabs: 'Profile' (disabled), 'Appointments' (disabled), and '\$ Prices' (selected, highlighted in teal). On the far right is a 'Logout' button. The main content area is titled 'Services' and displays three tables: 'Bath/Shear', 'Companion', and 'Training'. The 'Bath/Shear' table shows price tiers based on weight: 0-7 kg at 10 €, 7-15 kg at 15 €, 15-25 kg at 20 €, 25-35 kg at 25 €, and >35 kg at 50 €. The 'Companion' table shows price tiers based on hours: 1-3 h at 3 €, 4-12 h at 5 €, and 12-24 h at 8 €. The 'Training' table shows a single entry for 'Per hour' at 6 €. The background features decorative paw prints in pink and blue.

Weight	Price
0-7 kg	10 €
7-15 kg	15 €
15-25 kg	20 €
25-35 kg	25 €
>35 kg	50 €

No. of Hours	Price
1-3 h	3 €
4-12 h	5 €
12-24 h	8 €

Hours	Price
Per hour	6 €

PetIt 2022 ©

Figura 44: Preços estabelecidos pela plataforma

7 Análise Crítica

7.1 Considerações gerais

Uma das fases mais importantes do desenvolvimento de software é analisar a sua capacidade. Desta forma, é possível antecipar futuros problemas, perceber a real capacidade do sistema e corrigir eventuais problemas que poderão surgir antes mesmo de disponibilizar a aplicação. Para além disso, a realização de testes permitirá detetar bugs e identificar *bottlenecks*, etc..

Assim sendo, foram realizados testes de carga com o objetivo de quantificar e avaliar algumas métricas relevantes tais como a capacidade de resistir a falhas, o cumprimento das funcionalidades pretendidas e a capacidade de resposta quando submetido a grandes quantidades de pedidos.

7.2 Testes de carga

A principal ferramenta que o grupo decidiu utilizar para realizar a análise de carga e concorrência da aplicação foi o **JMeter**. Esta, por sua vez, permite simular a utilização simultânea da aplicação por um dado número de utilizadores, ou seja, é possível definir qual o número de threads que pretendemos para testar a aplicação desenvolvida. Além disso, esta ferramenta possibilita o desenho de gráficos e sumários de acordo com vários parâmetros, tais como, o **throughput**, **average response time**, e **abort rate**. Assim sendo, o JMeter possibilita a execução sucessiva de vários pedidos HTTP.

Por este meio, após uma breve pesquisa, o grupo decidiu usar a extensão **BlazeMeter**, de forma a possibilitar a gravação de sessões (e os pedidos HTTP feitos durante as mesmas) que simulem a interação de um utilizador com o sistema.

Posto isto, o grupo decidiu focar-se em 3 diferentes interações: Uma dedicada ao controlo de concorrência que deverá ser feito quando dois clientes tentam fazer um mesmo pedido simultaneamente ao mesmo *petsitter*. De seguida as outras duas interações consistem, em duas sessões (uma para cada tipo de utilizador, ou seja uma para o *Petsitter* e outra para o cliente) que o grupo julga que serão as mais recorrentes quando a aplicação for lançada ao público.

Note-se que o grupo utilizou uma connection pool size de 150, uma vez que o valor *default* usado pelo Hibernate é bastante inferior a este.

7.2.1 Fazer Pedido

Tal como descrito anteriormente, é necessário ter uma especial atenção a esta funcionalidade, uma vez que o código correspondente trata-se de uma secção crítica, onde se lida com dados mutáveis. Assim sendo, o grupo decidiu usar a biblioteca **ReentrantLock** de forma a conseguir lidar com este problema.

De forma a testar se a aplicação realmente estava a conseguir controlar esta concorrência foi monitorizada a seguinte sessão:

- POST http://localhost:8080/api/getMakeRequest
- POST http://localhost:8080/api/makeRequest
- POST http://localhost:8080/api/petsitterProfileReq

Foram obtidos os seguintes resultados:

A. Fazer pedido: *controlo de concorrência*

<i>Nº Threads</i>	5	10	50	125	250
<i>Tempo médio de resposta (s)</i>	10.4	12.7	38.6	89.9	179.6
<i>Throughput (requests/min)</i>	46.5	66.0	84.0	90.0	96.0
<i>Abort Rate (%)</i>	80.0%	90.0%	98.0%	99.2%	99.6%

Figura 45: Funcionalidade ”Make Request”

Tal como é possível observar na figura, vemos que em cada coluna apenas uma thread é que corre sem erros, logo essa thread representa o cliente que é aceite pelo sistema para fazer o pedido ao petsitter. Todas as outras threads dão uma exceção. Comprovamos assim que a nossa aplicação é capaz de controlar a concorrência referente à funcionalidade principal da aplicação.

7.2.2 Interação 1 – Cliente

Esta interação consistiu nos seguintes pedidos HTTP:

- POST http://localhost:8080/api/login
- GET http://localhost:8080/api/allPetsitters
- POST http://localhost:8080/api/petsitterProfileReq
- GET http://localhost:8081/clientProfile
- POST http://localhost:8080/api/clientProfile

B. Interação Cliente

Nº Threads	50	100	150	200	250
Tempo médio de resposta (s)	17.4	23.2	33.5	34.6	41.9
Throughput (requests/sec)	5.1	7.5	7.4	9.1	9.4
Abort Rate (%)	0.0%	0.0%	0.0%	12.2%	27.4%

Figura 46: Interação 1 – Cliente

Como se pode verificar na tabela até às 150 threads verifica-se um aumento do tempo de resposta. Relativamente aos valores do throughput, contrariamente ao que era esperado, também se verificou um aumento, o que pode ser consequência do cálculo da média dos testes não ser o mais adequado. A partir de 200 threads já é visível que o sistema está perante um ponto de rotura, uma vez que já se verifica incapacidade para responder a todos os pedidos, como se pode verificar pela percentagem do *Abort rate*.

7.2.3 Interação 2 – *Petsitter*

Esta interação é caracterizada por:

- Login
- Petsitter Profile
- Edit Ad
- Petsitter's Appointments

Isto é, o petsitter faz login na aplicação, acede ao seu próprio perfil, e de seguida edita a sua Ad, com as alterações que pretende fazer. No final consulta os seus appointments, isto é os pedidos dos clientes feitos a si.

C. Interação *Petsitter*

<i>Nº Threads</i>	50	100	150	200	250
<i>Tempo médio de resposta (s)</i>	12	16.6	14.0
<i>Throughput (requests/sec)</i>	2.7	3.4	6.7
<i>Abort Rate (%)</i>	0.0%	0.0%	14.1%

Figura 47: Interação 2 – *Petsitter*

Assim sendo, a interação em relação a pedidos HTTP consistiu em:

- POST http://localhost:8080/api/login
- POST http://localhost:8080/api/petSitterProfile
- GET http://localhost:8081/editAd
- POST http://localhost:8080/api/getAd
- POST http://localhost:8080/api/petSitterProfile
- GET http://localhost:8081/petSitterProfile/appointments
- POST http://localhost:8080/api/appointments

Tal como é possível observar pela figura, é atingido um ponto de rutura entre as 100 e 150 threads, ou seja o sistema atingiu a sua capacidade máxima para responder a pedidos simultaneamente.

No seguinte gráfico encontra-se representado o tempo de resposta de cada pedido HTTP ao longo de tempo, onde é possível observar que aquele que tem um maior tempo é o de editar Ad.

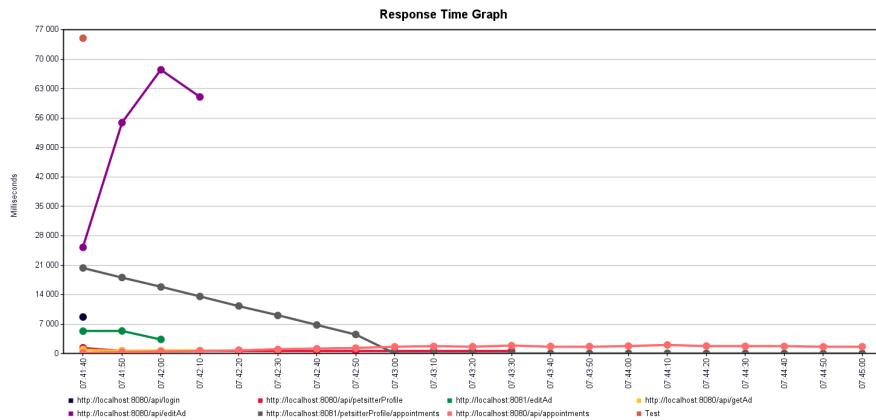


Figura 48: Response Time Graph

8 Análise de usabilidade

Segundo a norma ISO/IEC 25010:2011, para um produto ter qualidade tem que cumprir vários dimensões, incluindo a **usabilidade**.

8.1 Princípios de usabilidade

Ao desenvolvermos um *software* devemos seguir uma lista de princípios, intitulados **Princípios da usabilidade**, que funcionam como boas práticas no *design* de *interfaces* com o utilizador. Estes princípios dividem-se em três grandes grupos, que mencionamos abaixo.

8.1.1 *Learnability*

Em primeiro lugar temos os princípios de **Aprendizagem**, também como conhecidos por **Learnability**, que representam a facilidade com que novos utilizadores podem iniciar uma interação eficaz e alcançar máxima *performance*. Este princípio está dividido em cinco outros princípios: **Previsibilidade**, **Síntese**, **Familiaridade**, **Generalizabilidade** e **Consistência**.

No que diz respeito à **Previsibilidade**, temos, por exemplo, o botão ”Add pet” no perfil do cliente, que lhe dá a opção de adicionar um novo animal de estimação. Ao ver a sua lista de animais de estimação, o cliente está perante o princípio **Síntese**. Um exemplo de **Familiaridade** acontece quando o cliente está a marcar um pedido e escolhe o dia, através da utilização de um calendário, o que é familiar para qualquer utilizador.

8.1.2 *Flexibility*

Temos ainda os princípios de **Flexibilidade**, também conhecidos por **Flexibility**, que dizem respeito à multiplicidade de formas pelas quais o utilizador e o sistema trocam informações. Este princípio contém cinco outros princípios: **Iniciativa de diálogo**, **Multithreading**, **Migração das tarefas**, **Substituição** e **Personalização**.

Um exemplo de **Iniciativa de diálogo**, é a existência de filtros que aumenta no utilizador a sensação que controla a interação. Para além disso, o facto da nossa aplicação ser responsiva, faz parte do princípio da **Personalização**.

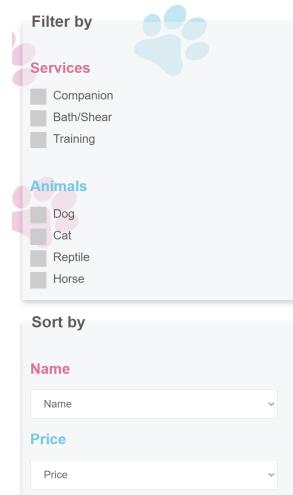


Figura 49: Filtros

8.1.3 *Robustness*

Para terminar, temos os princípios de **Robustez**, também conhecidos como **Robustness**, que dizem respeito ao nível de suporte fornecido ao utilizador na determinação do êxito e na avaliação do comportamento direcionado a objetivos. Este encontra-se dividido por quatro outros princípios: **Observabilidade**, **Recuperabilidade**, **Capacidade de resposta** e **Conformidade de tarefas**.

Podemos comprovar este princípio, por exemplo a **Observabilidade**, na página de *login*, pois quando este falha, é mostrada uma mensagem de erro. Para além disso, existem vários botões de "Back" que são um exemplo de **Recuperabilidade**, sendo que os utilizadores poderão sempre voltar atrás.



Figura 50: Filtros

8.2 Heurísticas de Nielsen

Para além dos vários princípios de usabilidade, temos também o conjunto das **Heurísticas de Nielsen**, que são o conjunto de *guidelines* de usabilidade mais conhecido. Estas heurísticas ajudam a perceber se atingimos o nosso objetivo de construir uma *interface* gráfica com o máximo grau de usabilidade possível.

Vale relembrar que, tanto as heurísticas como os princípios de usabilidade referidos anteriormente, foram revisitados várias vezes durante a implementação da aplicação, para nos certificarmos que construímos a melhor *interface* possível.

8.2.1 *Visibility of system status*

A primeira heurística diz respeito à **visibilidade do estado do sistema** e certifica-se que o programa mantém os utilizadores informados sobre o que se passa, através de *feedback* apropriado.

No caso da nossa aplicação, quando o *login* está a ser efetuado podemos ver, visualmente, que o sistema está a tratar do início de sessão devido a um ícone de *loading*.



Figura 51: Processamento

Para além disso, a *interface* tem o cuidado de informar o utilizador sobre o sucesso de todas as suas operações, nomeadamente quando é inserido um novo animal, submetido um *Request* ou guardadas as informações de um anúncio.

8.2.2 *Match between system and the real world*

A segunda heurística diz respeito à **correspondência entre o sistema e o mundo real** e pretende garantir que é utilizada uma linguagem familiar e que a informação é apresentada numa ordem natural e lógica.

No que diz respeito, por exemplo, à inserção de datas ou de horários, a nossa aplicação cumpre este requisito, sendo que a interface gráfica é bastante intuitiva e vai de encontro ao que o utilizador já está habituado no mundo real.

1. Select the date

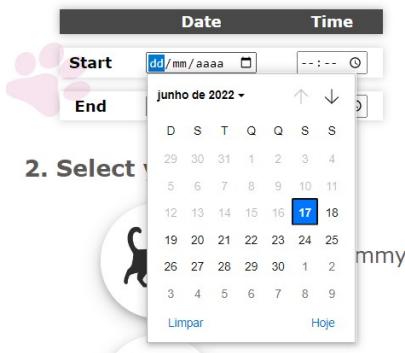


Figura 52

8.2.3 User control and freedom

A terceira heurística diz respeito ao **controlo e liberdade do utilizador** e certifica-se que são fornecidas "saídas de emergência" claramente marcadas aos utilizadores que executam ações por engano.

No caso da nossa aplicação, esta heurística é garantida, por exemplo, quando um utilizador está a visualizar o seu perfil e entra na página de inserir um novo animal. Nesta nova página é sempre possível encontrar um botão 'Cancel', onde o utilizador pode clicar para voltar atrás, caso não queira adicionar um novo animal. Este mesmo botão estará presente em interfaces de outras funcionalidades tais como editar um anúncio ou submeter um *Request*. Do mesmo modo, na página do anúncio de um *petsitter*, é também apresentado um botão "Back" para voltar à página anterior caso o utilizador assim o deseje.



Figura 53: Botão de cancelar operação

8.2.4 Consistency and standards

A quarta heurística diz respeito à **consistência e normas** que garante que a aplicação seguirá as convenções e normas da indústria e dela mesma.

Na nossa aplicação, tentamos seguir esta heurística durante todo o processo de desenvolvimento e para todas as *interfaces*. Isto pode ser comprovado com os botões de navegação que se encontram sempre do lado direito.

8.2.5 Error prevention

A quinta heurística diz respeito à **prevenção de erros** e, tal como o nome indica, tenta prevenir que os utilizadores cometam erros.

Em relação a este aspeto, temos como exemplo o registo de um novo *petsitter* em que se tentarmos inserir letras em vez de números no número de telefone, aparecerá uma mensagem que impedirá de prosseguir com o registo até ficar corrigido. Do mesmo modo, caso o utilizador indique uma data de nascimento cuja idade seja inferior a 18 anos o registo é impedido. Para além disso, é também impedida a submissão de campos vazios que sejam obrigatórios. Relativamente à página de submeter um '*Request*', é apresentada um mensagem de erro caso o utilizador selecione datas ou horas em que o *petsitter* esteja indisponível, sendo que só é possível fazer a submissão caso as mesmas sejam válidas.

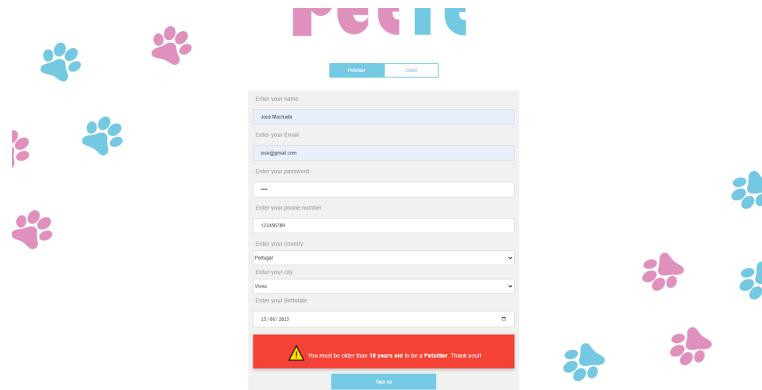


Figura 54: Utilizador menos de idade

8.2.6 Recognition rather than recall

A sexta heurística, **reconhecer em vez de recordar**, tenta minimizar a carga de memória do utilizador, fazendo que a informação necessária seja visível ou facilmente recuperável.

Como exemplo desta heurística, temos quando o cliente decide escolher visitar o anúncio de um *petsitter* depois de filtrar a sua *homepage*. No anúncio do *petsitter* irão ser mostradas novamente todas as informações pertinentes para poder fazer um pedido.

8.2.7 Flexibility and efficiency of use

A sétima heurística, **flexibilidade e eficiência de utilização**, afirma que a utilização de atalhos (ocultos dos utilizadores principiantes) podem acelerar a interação para o utilizador experiente.

A nossa **Navbar** inclui vários atalhos que o cliente ou o *petsitter* podem utilizar para facilitar a sua experiência na aplicação.

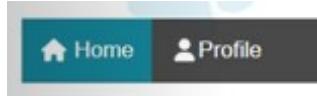


Figura 55: Barra de navegação

8.2.8 Aesthetic and minimalist design

A oitava heurística, **desenho estético e minimalista**, defende que as *interfaces* não devem conter informação que seja irrelevante ou raramente necessária.

Todas as *interfaces* da nossa aplicação foram criadas com esta heurística em mente, e pensamos que toda a nossa aplicação é exemplo de desenho estético e minimalista.

8.2.9 Help users recognize and recover from errors

A nona heurística defende a utilização de mensagens de erro em linguagem simples (sem códigos de erro), indicando o problema e uma possível solução, de modo a **ajudar os utilizadores a reconhecer, diagnosticar e recuperar de erros**.

Um exemplo desta heurística pode ser visto na página de *login*, pois quando este falha, é mostrada uma mensagem de erro. Outro exemplo desta heurística acontece quando o cliente tenta fazer um pedido para um dia ou hora que o *petsitter* não esteja disponível, é devolvida uma mensagem de erro que indica que o *petsitter* não se encontra disponível.

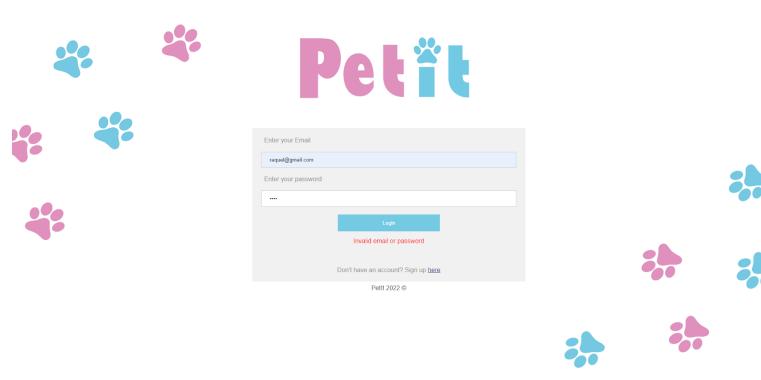


Figura 56: Erro de login

8.2.10 Help and documentation

A décima, e última, heurística, **Ajuda e documentação**, afirma que é melhor quando um sistema não precisa de qualquer explicação adicional, mas pode ser necessário fornecer informação.

Como acreditamos que a nossa aplicação é bastante simples e com um número limitado de funcionalidades, não sentimos necessidade de fornecer uma página de ajuda para esclarecer os utilizadores.

8.3 Questionário PSSUQ

Os questionários PSSUQ (Post-Study Usability Questionnaire) são utilizados para medir o grau de satisfação de um utilizador quanto a um sistema. Posto isto, a equipa decidiu desenvolver um questionário PSSUQ que foi enviado aos utilizadores alvo da aplicação, com o intuito de avaliar a interface gráfica do mesmo.

Este questionário consiste em 16 perguntas e pode ser consultado [aqui](#).

Depois de recolhidas as respostas do questionários, obtemos os seguintes resultados:

Pergunta	Pontuação
1 - Em geral, estou satisfeito com a facilidade de utilização do sistema.	1.41
2 - Foi fácil utilizar o sistema.	1.41
3 - Fui capaz de completar as tarefas rapidamente.	1.47
4 - Sinto-me confortável em utilizar este sistema.	1.41
5 - Foi fácil de aprender a utilizar o sistema	1.47
6 - Acredito que me poderia tornar rapidamente produtivo utilizando este sistema.	1.47
7 - O sistema deu-me mensagens de erro que me disseram como resolver o problema.	2.65
8 - Sempre que cometí um erro utilizando o sistema, eu consegui recuperar rapidamente e facilmente.	2.35
9 - As informações (como ajuda online, mensagens na tela e outra documentação) fornecida com este sistema era clara.	1.88
10 - Foi fácil encontrar as informações que eu precisava.	1.47
11 - As informações foram eficazes para me ajudar a completar as tarefas.	1.76
12 - A organização das informações nas páginas foi clara.	1.58
13 - A interface deste sistema é agradável.	1.18
14 - Gostei de utilizar a interface deste sistema.	1.59
15 - Este sistema tem todas as funcionalidades e capacidades que eu esperava.	1.64
16 - Em geral, estou satisfeito/a com o sistema.	1.35

As 16 perguntas podem ser organizadas em 4 subconjuntos diferentes para avaliar diferentes aspectos da interface, a satisfação geral do sistema (médias das 16 perguntas), utilidade (média das perguntas 1 a 6), qualidade de informação (médias das perguntas 7 a 12) e qua-

lidade da interface (média das perguntas 13 a 15).

Analisando as pontuações obtidas acima, obtivemos, para cada um dos 4 subconjuntos, as seguintes avaliações:

- Satisfação geral do sistema - 1.63
- Utilidade - 1.44
- Qualidade de informação - 1.95
- Qualidade da interface - 1.44

Observando os resultados obtidos para cada uma das avaliações (quanto menor o valor, melhor é), conseguimos perceber que o aspeto em que a aplicação teve pior avaliação foi na qualidade de informação, o que significa que como trabalho futuro a equipa iria focar neste aspeco da interface.

9 Conclusões

O primeiro desafio com que nos deparamos durante a realização do nosso projeto foi a conceção da ideia, pois desejávamos que a aplicação que desenvolvêssemos fosse útil e pudesse ser realmente implementada e utilizada no mundo real. Daí termos escolhido um serviço de *petsitting*, que acreditamos ser uma aplicação que poderá ser muito útil no dia-a-dia de todo o centenas ou milhares de pessoas, e é uma ideia que ainda não foi popularizada.

A primeira etapa de desenvolvimento, trata-se da etapa de modelação, fundamentação e planeamento, permitiu otimizar a organização do trabalho em equipa e garantir que a implementação da aplicação corre como planeado, para evitar ao máximo problemas relacionados com falta de tempo ou desorganização. Nesta etapa, começamos por desenvolver um modelo de domínio, que apresenta o funcionamento básico da nossa aplicação e identifica as principais entidades. De seguida, definimos os requisitos funcionais e não funcionais, de forma a percebermos as funcionalidades necessárias e que devem estar incluídas no sistema. Partindo destes requisitos, elaboramos o Diagrama de *Use Cases* e especificamos mais profundamente os mais relevantes, desenvolvendo depois os seus modelos de tarefas. Desenvolvemos ainda um Diagrama de Classes, percebendo mais concretamente como é que os dados estariam organizados de forma a solucionar todos os casos de uso de forma simples, prática e inteligente. Para além disso, desenvolvemos um Diagrama de Componentes, onde planeamos os vários componentes e subsistemas que vão fazer parte da nossa aplicação.

Desenvolvemos ainda várias *mockups* de possíveis *interfaces* gráficas da nossa aplicação, tentando perceber como poderíamos obter um resultado que fosse apelativo e ao mesmo tempo bastante usável. Esta foi uma fase muito importante no desenvolvimento da aplicação, pois a implementação baseia-se em tentar recriar o que aqui foi mostrado, respeitando sempre os princípios de usabilidade aprendidos. Para acompanhar as *mockups*, desenvolvemos um Diagrama de Máquina de Estados que define o fluxo das várias *interfaces*. Por fim, desenvolvemos um modelo lógico para a base de dados.

Na fase da implementação, fizemos uma análise exaustiva das melhores linguagens e tecnologias a utilizar e acabamos por escolher o *Vue.js* para a camada de apresentação, *Spring* para a lógica de negócio e *Hibernate* para a camada de dados. O *frontend* e o *backend* são módulos totalmente independentes, onde o *frontend* recorre a uma API criada no *backend* para o contactar.

A implementação do *frontend* tira partido da *framework* do *Vue*, nomeadamente da utilização de componentes pré-criados, de *routing* e de *storing*. No que toca ao *backend*, implementamos uma *Rest API*, de forma a poder comunicar com o *frontend*, criando um *controller* que faz o mapeamento dos pedidos do *frontend* no *backend* (através de *session beans*), e devolve a resposta ao *frontend*. Para além disso, utilizamos as anotações do *Hibernate* para a criação da base de dados.

Com a aplicação desenvolvida, apresentamos todas as *views* criadas, explicando o fluxo de utilização da plataforma. Por fim, fizemos uma análise de usabilidade, demonstrando exemplos do nosso cumprimento dos princípios de usabilidade e das heurísticas de Nielsen; e mostramos ainda informações sobre o questionário que realizamos.

10 Trabalhos Futuros

Nesta secção vão ser mencionados quais os aspetos que iriam ser tratados numa próxima fase do projeto.

10.1 Funcionalidades

Numa segunda fase do trabalho, o grupo iria focar-se nas funcionalidades que não teve tempo de implementar, funcionalidades secundárias, como a possibilidade do *petsitter* recusar um pedido, criar perfis para os clientes e ser possível avaliar *petsitters* e clientes.

Outro aspetto que ficou em falta realizar nesta fase do projeto, foram as fotos de perfil dos utilizadores e dos seus animais. Apesar do *frontend* da aplicação estar pronto para esta funcionalidade, ainda não existe conexão com o *backend* por isso quando um utilizador faz *upload* de uma foto, essa informação não está a ser passada ao *backend* e por isso não é armazenada.

10.2 Deployment

A nível do *deployment*, faríamos a replicação dos servidores de cada camada, base de dados, *backend* e *frontend*. Para isto, seriam adicionados também servidores *proxy* para realizar o balanceamento de carga entre as várias réplicas. Esta adição, permitiria responder a mais pedidos e de forma mais rápida, dado que para cada uma das camadas deixava de haver um único gargalo de *performance* e a carga passaria a estar distribuída.