

# POLI TÉCNICO GUARDA

**Escola Superior de Tecnologia e Gestão**

---

**ADOTA.ME**

---

PROJETO EM CONTEXTO DE ESTÁGIO  
NA PROJECTBOX  
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA  
INFORMÁTICA

**Gonçalo Silva**  
**Novembro / 2022**



# **Escola Superior de Tecnologia e Gestão**

---

## **ADOTA.ME**

---

PROJETO EM CONTEXTO DE ESTÁGIO  
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA  
INFORMÁTICA

Professor(a) Orientador(a): Maria Clara Santos Pinto Silveira

**Gonçalo Silva**  
**Novembro / 2022**



## Agradecimentos

Quero em primeiro lugar endereçar um forte agradecimento a todos os professores que tive durante a frequência do curso em Engenharia informática. Pois sem ajuda deles não teria crescido pessoalmente durante os passados três anos da licenciatura. Um obrigado em especial à professora Maria Clara Silveira por todo o seu apoio, dedicação e disponibilidade que me facultou e por me orientar sempre no sentido de melhorar o meu trabalho.

Também quero agradecer à empresa ProjectBox por me possibilitarem a realização do projeto em contexto de estágio. Obrigado pela oportunidade. Sem ela teria dúvidas acerca dos meus conhecimentos a nível de preparação para o mercado de trabalho.

Por fim quero agradecer aos meus pais, por todo o apoio nesta fase mais difícil e onde houve mais momentos de pressão e ansiedade. Sempre mostraram presença para me ajudar e para contornar os momentos mais difíceis.



## Ficha de Identificação

### **Aluno**

**Nome:** Gonçalo Miguel Reis e Silva

**Número:** 1703826

**Licenciatura:** Engenharia Informática

### **Estabelecimento de Ensino**

Instituto Politécnico da Guarda (IPG)

Escola Superior de Tecnologia e Gestão (ESTG)

### **Entidade Acolhedora do Estágio**

**Nome:** ProjectBox, Lda.

**Morada:** Rua das Camélias, 60 Viseu

**Contacto Telefónico:** 232 281 357

**Duração do Estágio:** 20/06/2022 - 19/08/2022

### **Supervisor de Estágio**

**Nome:** Alex Silva

**Função:** Gestor do Projeto

## **Docente Orientador de Estágio**

**Nome:** Maria Clara Santos Pinto Silveira

**Grau Académico:** Doutoramento em Engenharia Eletrotécnica e de Computadores



## Resumo

O presente relatório descreve o projeto realizado ao longo do estágio, no âmbito da unidade curricular Projeto de Informática, integrada na Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

O projeto desenvolvido consiste num sistema *web*, cujo principal objetivo é adotar, apadrinhar e fazer pedidos sobre: animais domésticos que tenham desaparecido ou que sejam encontrados e ainda, adicionar animais ao catálogo do tipo canídeos e felinos. O sistema ainda fornece uma estrutura integrada onde o utilizador se regista, inicia sessão e realiza pedidos no sistema. Assim é possível monitorizar os diversos pedidos e feitos por que utilizadores.

Para o desenvolvimento do projeto foi utilizada a metodologia de desenvolvimento ágil, *Kanban*. A parte *backend* do sistema foi desenvolvida em Node.js com recurso à base de dados PostgreSQL. Já no desenvolvimento *frontend* foi utilizada a *framework* Bootstrap.

Neste documento, são descritos ainda, aplicações existentes selecionadas, casos de uso do sistema, processos de implementação referentes às tecnologias utilizadas, interfaces do utilizador e testes ao *software* produzido.

**Palavras-Chave:** *Web*, Node.js, PostgreSQL, Bootstrap, *software*;



## Abstract

This report describes the project carried out during the internship, within the scope of final project of the Computer Engineering degree at the Higher School of Technology and Management, Polytechnic of Guarda.

The project developed consists of web system whose main objective is to adopt, sponsor and make requests about: domestic animals that have disappeared or are found and add animals to the catalog of the type canids and felines. The system also provides an integrated structure where the user registers, logs on, and makes requests on the system. Thus, it is possible to monitor the various requests and made by which users.

For the development of the project, the agile development methodology, Kanban, was used. The backend part of the system was developed in Node.js using the PostgreSQL database. In frontend development, the Bootstrap framework was used.

In this document, there are also selected existing applications system use cases, implementation processes related to the technologies used, user interfaces and tests of the software produced.

**Keywords:** *Web*, Node.js, PostgreSQL, Bootstrap, *software*;



# Índice

Agradecimentos .....	i
Ficha de Identificação .....	i
Resumo .....	iii
Abstract.....	v
Índice de Figuras .....	ix
Lista de Siglas e Acrónimos .....	11
1. Introdução .....	12
1.1 Enquadramento e motivação .....	12
1.2 Caracterização sumária da instituição de acolhimento .....	13
1.3 Descrição do problema .....	13
1.4 Objetivos.....	14
1.5 Metodologia de desenvolvimento: Kanban.....	15
1.6 Estrutura do relatório.....	16
2 Estado da Arte.....	17
2.1 Canil-Gatil Intermunicipal de Tomar .....	18
2.2 Canil Intermunicipal das Terras de Santa Maria.....	19
2.3 Associação MIDAS .....	20
2.4 Análise crítica ao estado da arte .....	21
3 Análise de requisitos .....	23
3.1 Diagrama de contexto .....	25
3.2 User stories .....	26
3.3 Diagrama de casos de uso .....	30
3.4 Diagramas de atividades e de estados .....	31
3.5 Diagrama de classes .....	34
4 Tecnologias e ferramentas .....	39
4.1 Visual Studio Code.....	39
4.2 JavaScript e NodeJS.....	41
4.3 ExpressJS.....	43
4.4 PostgresSQL.....	43
4.5 Dbeaver .....	44
4.6 SQL Shell .....	45
4.7 HTML5 .....	46
4.8 CSS3 .....	47

4.9	Bootstrap.....	48
4.10	Git/GitHub.....	49
4.11	Gitignore.....	50
4.12	Figma.....	50
4.13	Draw.io.....	51
4.14	jQuery.....	52
5	Implementação.....	53
5.1	Instalações das bibliotecas e dependências.....	53
5.2	Implementação do servidor local.....	57
5.3	Rotas, verbos HTTP e JavaScript embebido.....	58
5.4	Constituição das vistas.....	62
5.5	Base de dados.....	66
5.5.1	Criação da base de dados e utilização do método UUID .....	67
5.5.2	Utilização da API pg-promise e da biblioteca Bcrypt.....	69
5.6	Interfaces da aplicação web.....	72
5.6.1	Interface Login/Registo.....	72
5.6.2	Interface para ver características dos animais .....	73
5.6.3	Interface para adotar animal .....	74
5.6.4	Interface consultar e validar pedidos.....	76
6	Testes.....	77
7	Conclusão.....	82
	Bibliografia.....	84
	Anexos.....	87
	A 1. Criação das tabelas do ficheiro createTables.sql.....	87
	A 2. Modal Login partilhada pelo Figma.....	89
	A 3. Interface características do animal partilhada pelo Figma .....	90
	A 4. Interface para adotar animal partilhada pelo Figma.....	90
	A 5. Interface consultar e validar pedidos partilhada pelo Figma .....	92

## Índice de Figuras

Figura 1 - Quadro Kanban do projeto Adota.me [3].....	16
Figura 2 - Catálogo de animais para adoção [4].....	19
Figura 3 - Layout e animais para adoção [5].....	20
Figura 4 - Layouts Associação MIDAS [7].....	21
Figura 5 - Hierarquia de Requisitos [8] .....	24
Figura 6 - Diagrama de Contexto .....	25
Figura 7 - Diagrama de casos de uso.....	30
Figura 8 - Diagrama de atividades para pedidos do tipo Adotar/Apadrinhar/Adicionar animal/Reportar animal desaparecido/Avistar animal desaparecido .....	32
Figura 9 - Diagrama de estados quando é criado um pedido.....	33
Figura 10 - Diagrama de Classes .....	36
Figura 11 - Ambiente de desenvolvimento do VSCode.....	40
Figura 12 - Exemplo NodeJS .....	42
Figura 13 - ExpressJS.....	43
Figura 14 - PostgresSql.....	44
Figura 15 - Ambiente visual e de desenvolvimento Dbeaver.....	45
Figura 16 - SQL Shell.....	46
Figura 17 - HTML5.....	47
Figura 18 - CSS3 .....	48
Figura 19 - Bootstrap .....	49
Figura 20 - Git e GitHub .....	49
Figura 21 - Gitignore .....	50
Figura 22 - Layouts do projeto no Figma.....	51
Figura 23 - jQuery.....	52
Figura 24 - Arquivos do projeto .....	54
Figura 25 - Ficheiro package.json .....	55
Figura 26 - Implementação do servidor.....	58
Figura 27 - Criação dos caminhos para as Rotas no app.js.....	59
Figura 28 - Rota donate.js .....	60
Figura 29 - Ilustração entre a rota donate.js e a sua respetiva vista.....	61
Figura 30 - Código do conteúdo header.....	62
Figura 31 - Ilustração do header para a rota donate.js .....	62
Figura 32 - Layout catálogo e página principal.....	63
Figura 33 - Pasta layouts e partials.....	64
Figura 34 - Vista default.ejs.....	65
Figura 35 - Vista noaside.ejs .....	65
Figura 36 - Definição da biblioteca express-ejs-layouts no ficheiro app.js .....	66
Figura 37 - Criação do schema, tabela user e login.....	68
Figura 38 - Aplicação da função gen_random_uuid() .....	69
Figura 39 - Conexão à base de dados .....	70
Figura 40 - Excerto do ficheiro registry.js .....	70
Figura 41 - Modal registo .....	71
Figura 42 - Interface de Login.....	73
Figura 43 - Interface características do animal.....	74
Figura 44 - Interface para adotar animal.....	75
Figura 45 - Interface consultar e validar pedidos .....	76
Figura 46 - Caso de teste para criar utilizador.....	77
Figura 47 - Validação dos campos para pedido "Adicionar animal" .....	78

Figura 48 - Insert para o pedido apadrinhar animal.....	79
Figura 49 - Estado dos pedidos.....	80
Figura 50 - Query que filtra os pedidos pelo status .....	80
Figura 51 - Status dos pedidos .....	81
Figura 52 - Query para pedido aprovado .....	81
Figura 53 - Query para pedido reprovado.....	81

## Índice de Tabelas

Tabela 1 - Comparação entre aplicações diferentes.....	22
--	----



## Lista de Siglas e Acrónimos

API	Application Programming Interface
CSS	Cascading Style Sheet
HTML	HyperText Markup Language
IA	Inteligência artificial
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MIDAS	Movimento Internacional para a Defesa dos Animais
NPM	Node Package Manager
SQL	Structured Query Language
UC	Unidade curricular
UML	Unified Modeling Language
URL	Uniform Resource Locator

# 1. Introdução

O presente documento relata todo o trabalho desenvolvido no projeto Adota.me, realizado pelo aluno Gonalo Miguel Reis e Silva, no mbito da Unidade Curricular (UC) “Projeto de Informtica”. UC do terceiro ano de Engenharia Informtica para obteno deste curso da Escola Superior de Tecnologia e Gesto do Instituto Politcnico da Guarda.

O relatrio apresenta detalhadamente todos os objetivos da construo do sistema Adota.me, desde a recolha de dados, passando pelos processos de engenharia de software, at aos testes e publicao da aplicao. Foi desenvolvido no mbito, Projeto em contexto de estgio, sobre tutela da empresa ProjectBox.

## 1.1 Enquadramento e motivao

O convite destinado a abraar este projeto foi o facto de existirem diversos canis/gatis, ou s canis distribudos pelos diversos municpios de Portugal, e no haver uma maior promoo em relao aos habitantes das instalaes canil/gatil. Os municpios que tm instalaes de acolhimento para animais abandonados possuem diretamente nas suas pginas web um tipo de anncio simples e genrico referente  promoo de adoo de animais vtimas de abandono. No entanto, as publicaes feitas pelos municpios no seu meio de comunicao geral, no so suficientes para acelerar o processo global de adoo de um animal.

A proposta, face ao problema que aqui est em causa, como pretenso, ser alojar no site principal do municpio de So Pedro do Sul, um novo meio de comunicao capaz de mostrar aos utilizadores, o catlogo onde se encontram vrios animais disponveis para adoo. Isto faz com que os utilizadores interessados em adotar um animal no tenham de se deslocar a um canil/gatil mais prximo para ver os animais. Basta consultarem o catlogo disponvel online e realizarem pedidos de adoo pelo sistema *web*.

Esta ideia nasceu do interesse pessoal. A ideia foi apresentada  empresa ProjectBox para tentativa de ser desenvolvida em estgio, que acabou por ser aceite pela mesma entidade.

Por outro lado, foi realizada uma chamada telefónica ao município de São Pedro do Sul a explicar o propósito da aplicação. O resultado foi positivo por parte do município. Mostraram interesse e aceitação acerca de, mais tarde verem a aplicação *web*, tal como ler o presente documento.

## 1.2 Caraterização sumária da instituição de acolhimento

A empresa de acolhimento para a concretização deste projeto chama-se ProjectBox [1], sediada em Pascoal, na cidade de Viseu. É uma empresa inovadora na área das Tecnologias de Informação e de Comunicação, que se destaca no desenvolvimento de software.

O ambiente dentro da empresa é agradável, tal como o espaço de trabalho, é aberto, espaçoso, acolhedor, simples, e possui um ar recheado de características juvenis.

No dia-a-dia há convivência entre pessoas ligadas às áreas de informática e design. Durante as atividades laborais são feitas pausas para reuniões sobre projetos que estão em desenvolvimento dentro da empresa, pausas para descansar e para dialogar entre os colaboradores, o que contribui para o bem-estar da equipa.

Além disto, a ProjectBox é moderna, pró-ativa e multidisciplinar. Pois, adotam de tecnologias que estão no seu auge, por exemplo, o JavaScript, GitHub, etc. A empresa também usufrui das metodologias ágeis, agilidade e eficiência nas entregas e na execução do projeto como um todo, por exemplo, os quadros Kanban do software Jira e a metodologia ágil SCRUM. Todas estas tecnologias adotadas pela ProjectBox resumem-se na palavra organização. O que contribui para o máximo de performance e comunicação da equipa, bem como, para a redução de riscos. Assim o resultado do software produzido será de alta qualidade.

## 1.3 Descrição do problema

Sabe-se que existem milhares de animais abandonados, doentes ou mal tratados, e ninhadas sem abrigo que acabam por serem abatidas, por causa de: outros animais,

atrocidades humanas ou falta de alimento. A este problema acresce também o facto de a maioria dos animais do tipo canídeos e felinos não se encontrarem esterilizados ou mesmo vacinados, tornando difícil o controlo de pragas e colocando em risco a saúde pública.

Para tentar obter algum controlo sobre a reprodução de animais abandonados, seria necessário encontrá-los, resgatá-los e esterilizá-los. Como é impossível concretizar totalmente a ideia anterior, na melhor das hipóteses seria prestar atenção a qualquer tipo de população de animais abandonados. Assim, caso se avistasse alguma, o objetivo seria recolhê-la e encontrar alguém capaz de adotar definitivamente estes animais.

Deste modo, se diminuía casos entre: animais mortos por fome, vítimas de atropelamentos ou brigas com outros animais. Ter-se-ia mais controlo sobre animais abandonados, o que evitaria a reprodução de futuras gerações que seriam as próximas vítimas de abandono ou espécies selvagens.

## 1.4 Objetivos

O principal objetivo do projeto é implementar um sistema *web* que permite a adoção de animais do tipo canídeos e felinos residentes em canis e gatis.

A solução que se propõe neste projeto baseia-se no desenvolvimento de uma plataforma web, onde estarão disponíveis os perfis virtuais dos animais, permitindo a utilizadores externos demonstrarem interesse na adoção/apadrinhamento sobre os animais disponíveis no sistema. Será possível consultar dados bancários para doar dinheiro para suporte das despesas de um determinado animal. É possível reportar o desaparecimento de animais domésticos, tal como, fazer comunicados caso se avistem animais domésticos desaparecidos. Também é possível adicionar animais ao sistema através de pedidos.

O Gestor da plataforma, faz toda a gestão do catálogo dos animais e responderá aos pedidos dos utilizadores externos. Por exemplo, quando o utilizador adiciona um animal ao sistema, ou faz uma proposta de adoção/apadrinhamento, primeiramente estas adições são convertidas em pedidos e enviados para o Gestor para consulta e validação. O Gestor possui de uma interface em que consulta os pedidos propostos pelos utilizadores. Perante esta

interface, o Gestor poderá consultar os dados preenchidos nos formulários referentes aos pedidos propostos pelos utilizadores. Podendo assim ajudar na validação dos pedidos por parte do Gestor.

Os perfis dos animais, para além da fotografia, apresentarão, também, as suas características quanto à sua fisionomia e informações de saúde. Sendo possível ler sobre cada animal as seguintes características e condições: nome, tipo (cão ou gato), género, data de nascimento, tamanho, pelo, raça, cor, vacinas, ração, estado de saúde e cuidados redobrados a ter com o animal em específico.

## 1.5 Metodologia de desenvolvimento: Kanban

Um dos maiores desafios para quem vai começar a implementar algum software, é conseguir garantir que a execução do projeto seja bem feita e que a entrega final seja coerente com o que foi proposto inicialmente, na análise de requisitos. O que faz com que se precise de algum método que auxilie a focar nas tarefas que se têm que desempenhar ao longo do desenvolvimento do software.

“Para isso nasceram” as metodologias ágeis, capazes de trazerem mais flexibilidade para o processo de planeamento e desenvolvimento de software. Há diversas metodologias ágeis disponíveis a serem adotadas conforme as necessidades do projeto, alguns exemplos são: *SCRUM*, *XP*, *Kanban*, etc. Metodologias ágeis são métodos adotados em projetos que facilitam a resolução de problemas, assim como, ajudam na questão da estruturação e organização de qualquer projeto. [2]

A metodologia ágil adotada durante a realização do projeto Adota.me foi o modelo *Kanban* do *software* Jira. Esta metodologia consiste na criação das diversas tarefas impostas pelo projeto e vão sendo distribuídas conforme o estado da tarefa pelas seguintes colunas nomeadas de: “Não Iniciado”, “Em Progresso” e “Concluído”.

A Figura 1 é exemplo de algumas tarefas importantes e reais, para boa prática na organização do projeto.

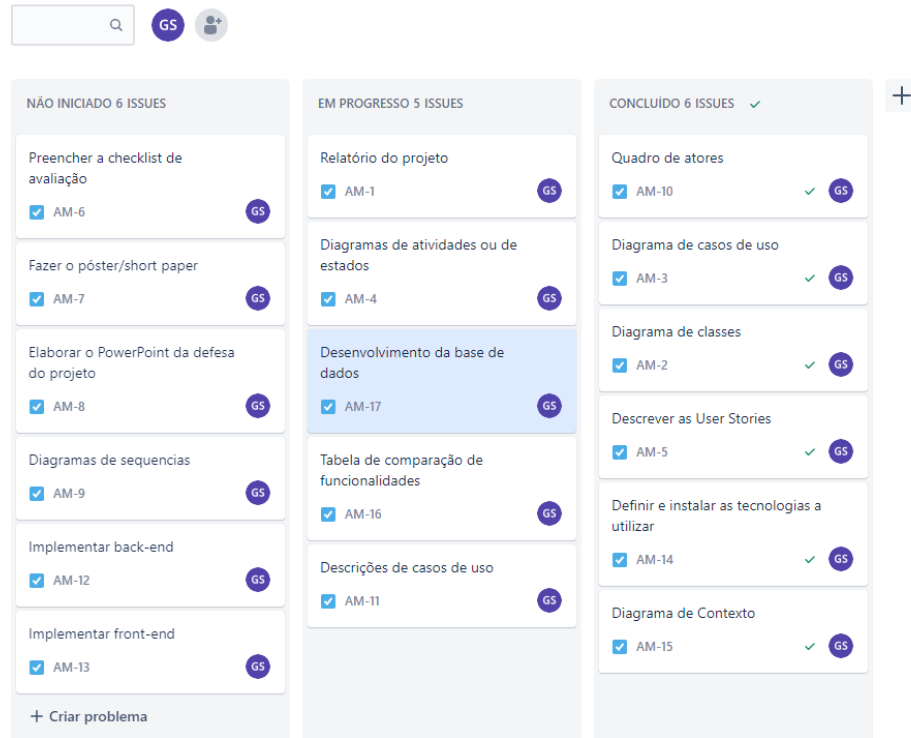


Figura 1 - Quadro Kanban do projeto Adota.me [3]

Todas as tarefas da Figura 1 serviram para partir o problema em dois, e a partir das metades, separar em pequenas tarefas que ao serem completadas resultam no produto final que será o sistema Adota.me.

Como o projeto Adota.me foi desenvolvido a solo, não houve a necessidade de escolher outra metodologia ágil, por exemplo a metodologia *SCRUM*. Na metodologia *SCRUM*, há a obediência de entregar o software produzido ao cliente ou ao responsável do projeto em diversos períodos até o projeto estar concluído. Dado este facto, foi decidido dividir o projeto pelas tarefas a realizar como mostra a Figura 1.

## 1.6 Estrutura do relatório

O presente relatório tem uma estrutura organizada por capítulos dos quais serão apresentados a seguir:

Capítulo 1 – Introdução: Este é o capítulo atual que pretende apresentar o projeto realizado e fornecer uma contextualização generalizada do problema proposto.

Capítulo 2 – Estado da Arte: Neste capítulo realiza-se um estudo e análise de outras plataformas no mercado com o objetivo de identificar lacunas e oportunidades de melhorias.

Capítulo 3 – Metodologia de desenvolvimento: Neste capítulo é descrita a metodologia de trabalho aplicada a este projeto, a metodologia de desenvolvimento Kanban.

Capítulo 4 – Análise de requisitos: Neste capítulo é apresentado o estudo sobre a identificação e documentação de requisitos.

Capítulo 5 – Tecnologias e Ferramentas: Neste capítulo são apresentadas as diversas tecnologias utilizadas durante o desenvolvimento do projeto Adota.me.

Capítulo 6 – Implementação: Neste capítulo é apresentada a descrição da implementação de maneira a fundamentar a lógica utilizada no decurso do desenvolvimento do projeto para corresponder aos requisitos do trabalho.

Capítulo 7 – Testes: Neste capítulo são realizadas experiências utilizando o sistema final de forma a validar os objetivos propostos.

Capítulo 8 – Conclusão: Neste capítulo final são apresentadas as conclusões do trabalho, analisando os objetivos concretizados e possível trabalho futuro.

## 2 Estado da Arte

Este projeto nasceu do interesse pessoal enquanto estava em período de aulas na licenciatura em Engenharia Informática. Tendo em conta que a ideia não seja muito popular, juntando o gosto do tema de desenvolvimento para a *web*, porque não concretizar o projeto? E de facto a ideia que surgiu no ambiente académico foi concretizada no final do curso, durante o projeto em Contexto de estágio.

Uma vez abordado o objetivo do que se pretendia fazer, o ponto de partida foi verificar se já existiam soluções para o mesmo problema. Após feita alguma pesquisa foram

encontradas poucas aplicações com o mesmo propósito. Das aplicações selecionadas, foi feita uma análise do que se poderia retirar e melhorar para o desenvolvimento da aplicação *web* para adoção e apadrinhamento de animais.

Perante as três aplicações selecionadas, foi feita uma análise das mesmas, apenas uma delas se encontra com os objetivos mais próximos que o projeto Adota.me tem em comum. E um dos maiores fatores que se pretende implementar é concretizar uma adoção ou apadrinhamento *online* de um animal que se encontre disponível no catálogo. E ainda poder realizar pedidos do tipo adicionar animal, reportar animal doméstico desaparecido ou encontrado. Por esta razão, das três aplicações selecionadas, apenas uma serve este propósito.

Dos três *sites* nacionais selecionados, será feita uma breve análise dos mesmos neste subtópico. Os *sites* analisados foram: Canil-Gatil Intermunicipal de Tomar, Canil Intermunicipal das terras de Santa Maria e Associação MIDAS (Movimento Internacional para a Defesa dos Animais).

## 2.1 Canil-Gatil Intermunicipal de Tomar

O *site* Canil-Gatil Intermunicipal de Tomar [4] desenvolvido pela NOESIS sob supervisão da Comunidade Intermunicipal do Médio Tejo possui um catálogo de animais para adoção, desaparecidos e encontrados no *site* Canil-Gatil de Tomar, este apenas dispõe de informações sobre os animais. Não é possível preencher um formulário sobre adotar determinado animal presente no catálogo *online*. Encontram-se também poucas informações sobre os diversos animais que se encontram disponíveis no catálogo para adoção. Não existe a funcionalidade para apadrinhar um animal. Logo, o *site* Canil-Gatil Intermunicipal de Tomar não vai ao encontro do que se espera na aplicação Adota.me



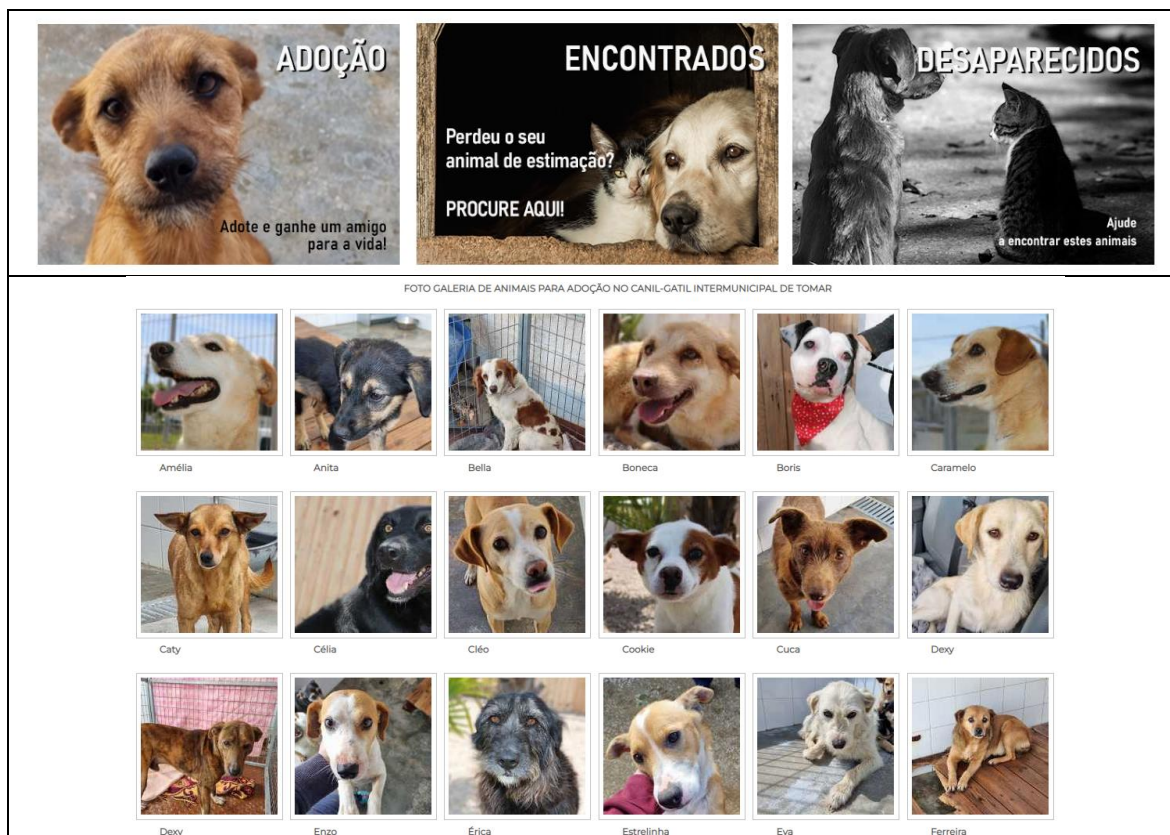


Figura 2 - Catálogo de animais para adoção [4]

A Figura 2, mostra os três tipos de catálogos que se encontram disponíveis no *site* Canil-Gatil intermunicipal de Tomar. Catálogos para adoção, desaparecidos e encontrados. Na parte mais abaixo da Figura 2 vê-se o catálogo com os diversos animais que se encontram no canil para adoção.

## 2.2 Canil Intermunicipal das Terras de Santa Maria

O *site* Canil Intermunicipal das Terras de Santa Maria [5] foi desenvolvido pela empresa *Livetech*. O *site* apresenta um vasto catálogo de animais para se adotar e também existe uma caracterização única para cada animal. Porém, a divulgação no *site* deste município trata-se apenas de animais canídeos. E para efetuar um pedido de adoção, não poderá ser feito *online*.

A página *web* também apresenta um catálogo de animais desaparecidos. Outra vantagem deste *site* é que dá para comunicar um animal desaparecido, preenchendo um formulário disponível no mesmo *site* com os dados do animal desaparecido e do dono.



Figura 3 - Layout e animais para adoção [5]

Na Figura 3 vê-se um menu sobre o que dá para navegar no *site* Canil Intermunicipal das Terras de Santa Maria. Assim como mostra o catálogo alusivo aos animais que estão disponíveis para adotar deste canil. Ao selecionar no menu lateral “Adoptados”, o *site* mostra o catálogo de animais que foram adotados. Poderá ser uma funcionalidade onde desperte mais interesse ao utilizador para adotar um canídeo. A seguir, ao selecionar “Desaparecidos”, o sistema mostra o catálogo de animais que se encontram desaparecidos. E por fim, ao selecionar “Reportar desaparecido”, o sistema mostra o formulário para preencher com dados sobre o desaparecimento de um animal doméstico.

## 2.3 Associação MIDAS

O *site* Associação MIDAS [6] foi desenvolvido pela empresa Netinbound. Na aplicação MIDAS é possível explorar um vasto catálogo de animais onde é possível adotar ou apadrinhar cães e gatos. Porém, não é possível adicionar animais como pedido do tipo adicionar animal, reportar animais desaparecidos e comunicar avistamento de animais perdidos. Entretanto na aplicação MIDAS dá para: ser sócio e contribuir para as despesas dos animais, agendar uma sessão de esterilização para algum animal, visualizar e comprar brinquedos/objetos alusivos a animais e à Associação MIDAS, e ainda é possível fazer voluntariado na associação.

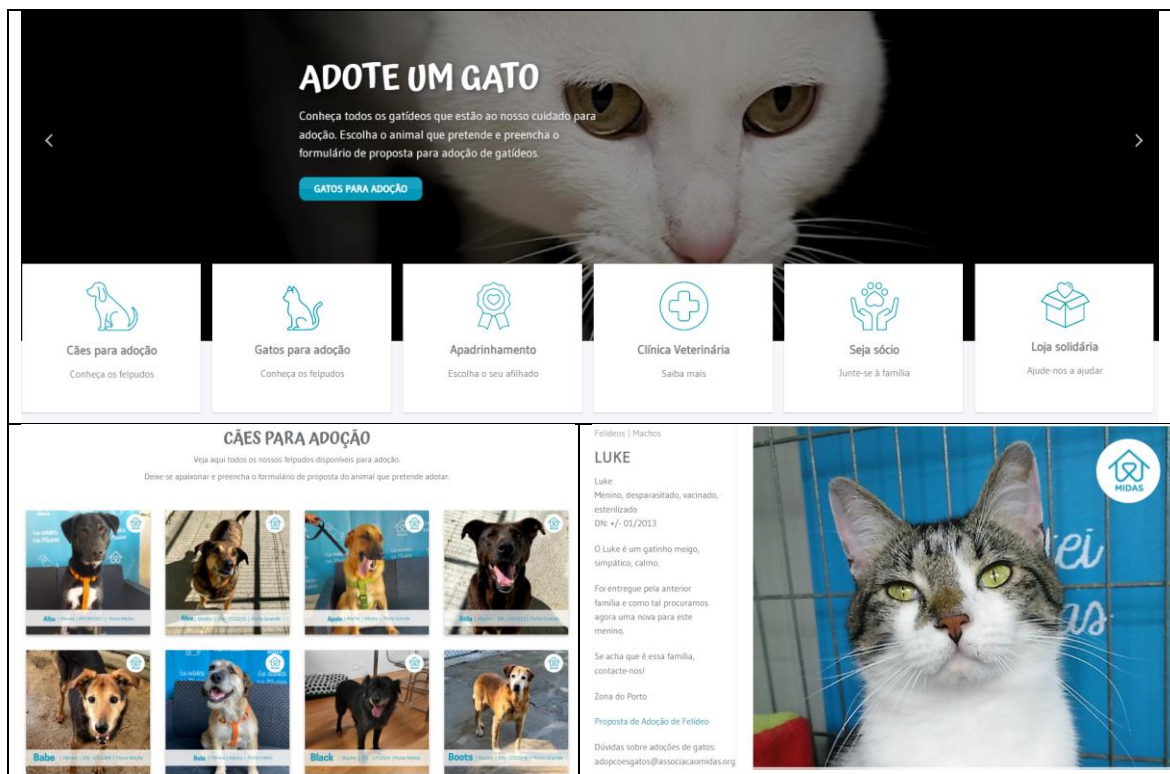


Figura 4 - Layouts Associação MIDAS [7]

A Figura 4 mostra três *layouts* diferentes do *site* Associação MIDAS. Na imagem do topo vêm-se algumas das funcionalidades do que possa fazer neste *site*, por exemplo: cães para adoção, gatos para adoção, apadrinhamento, entre outros. Na imagem do canto inferior esquerdo vê-se o catálogo de cães para adoção. E na imagem do canto inferior direito observa-se o perfil e as características sobre um animal que está disponível para adoção ou apadrinhamento. É o *site* que mais se aproxima com a aplicação Adota.me.

## 2.4 Análise crítica ao estado da arte

A Tabela 1 apresenta o que cada *site* contém em relação às funcionalidades analisadas de acordo com a pesquisa feita e explicitada no subtópico anterior, bem como as funcionalidades da aplicação Adota.me.

Tabela 1 - Comparação entre aplicações diferentes

Plataforma Funcionalidades	Canil/gatil intermunicipal de Tomar	Canil intermunicipal das terras de Santa Maria	Associação MIDAS, sediada em Matosinhos	Adota.me
Catálogo para adoção	•	•	•	•
Catálogo para apadrinhar			•	•
Catálogo de desaparecidos	•	•		•
Ver características dos animais	•	•	•	•
Reportar animal perdido		•		•
Comunicar caso aviste animal perdido	•	•		•
Visualizar dados bancários			•	•
Consultar notificações				•
Adicionar animais através de pedidos				•
Catálogo dos animais encontrados	•			
Catálogo dos animais adotados		•		

O que se pretende implementar no sistema Adota.me vai de encontro com as ideias aplicadas nos *sites* analisados como se verifica na Tabela 1. Os três *sites* analisados forneceram ótimas ideias e soluções para realizar os objetivos pretendidos para a criação da solução Adota.me.

O *site* canil-gatil Intermunicipal de Tomar tem como principal objetivo promover a adoção de um animal do tipo cão e gato. No entanto não possibilita ao utilizador fazer um pedido do tipo adotar ou apadrinhar um animal disponível via *online*. O que de facto perde para o sistema Adota.me. Pois neste, pretendem-se resolver processos de adoção e apadrinhamento.

O *site* canil Intermunicipal das Terras de Santa Maria contém uma das bases sólidas semelhante ao que se pretende implementar no projeto Adota.me. Essa base é o simples facto de ser disponibilizado no *site* diversas informações relevantes e alusivas ao perfil dos seus animais que mantêm no canil. E também o facto de ser possível preencher um formulário que diz respeito ao reporte de algum animal doméstico que se encontre desaparecido. Porém, como referido, trata-se apenas sobre adoção de animais canídeos, o que perde para o sistema Adota.me que tratará de adoções sobre cães e gatos.

A associação MIDAS é uma mais-valia e é o que mais se assemelha, em comparação com o sistema Adota.me porque é um *site* onde contém mais funcionalidades alusivas aos pedidos que se podem realizar sobre animais domésticos ou para adoção. É um *site* que para além de ser possível fazer pedidos de adoção e apadrinhamento, é possível fazer um pedido para esterilização de colónias de gatos. Além disto, para quem é sócio da Associação MIDAS, usufrui de serviços veterinários com mais vantagens. As únicas desvantagens encontradas no *site* da associação MIDAS em relação ao sistema Adota.me são: a impossibilidade de criar pedidos do tipo “adicionar animal ao catálogo”, reportar um animal desaparecido e comunicar um animal perdido caso seja visto.

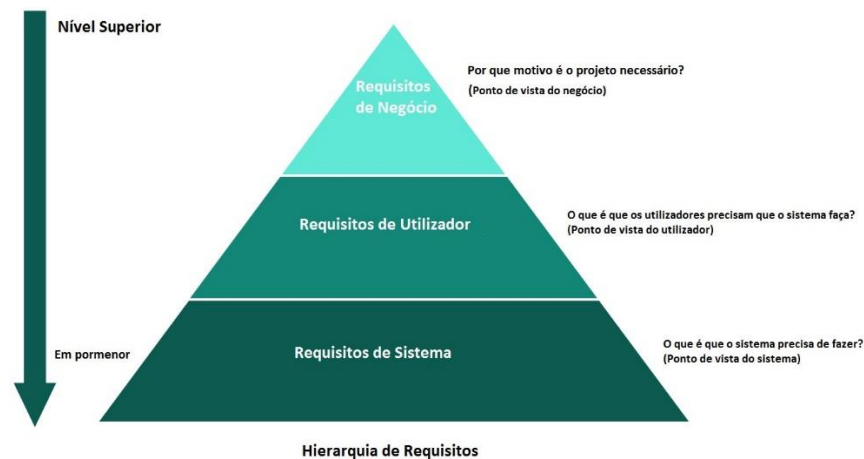
Tendo em conta as ideias analisadas perante os três sites, relativamente com os objetivos que se pretendem implementar na solução Adota.me, espera-se que a solução proposta seja no mínimo uma evolução do que são os *sites* canil-gatil Intermunicipal de Tomar e o canil Intermunicipal das Terras de Santa Maria.

### 3 Análise de requisitos

Os requisitos são resultados do estudo imposto a determinado software antes da sua implementação propriamente dita, e são características que o software final deverá cumprir.

Existem dois tipos de requisitos: os requisitos funcionais e os requisitos não funcionais. Os requisitos funcionais descrevem o comportamento exigido e as funções que o sistema deve desempenhar. Por exemplo, os casos de uso são requisitos funcionais. Os requisitos não funcionais são aqueles que estão relacionados com o uso da aplicação, por exemplo, em termos de desempenho, segurança e disponibilidade do sistema.

A Figura 5 representa a Hierarquia de Requisitos e revela-nos a importância dos mesmos em relação ao negócio, utilizador e sistema.



*Figura 5 - Hierarquia de Requisitos [8]*

Numa das primeiras reuniões com a empresa, para mostrar os requisitos de software recolhidos, os colaboradores da equipa ProjectBox, concordaram que o sistema deverá possibilitar:

- Consultar catálogo dos animais para adoção/apadrinhamento e animais desaparecidos
- Consultar as características dos animais disponíveis para adoção/apadrinhamento e dos animais desaparecidos
- Fazer pedidos de adoção/apadrinhamento
- Ver dados bancários para contribuir para as despesas dos animais
- Reportar algum animal doméstico caso tenha desaparecido
- Fazer comunicado caso se aviste algum animal doméstico desaparecido
- Fazer pedidos para adicionar animais
- Ver notificações sobre atividades no sistema, quer para o utilizador quer para o gestor
- Validar pedidos

Estes são os requisitos funcionais que o sistema deverá cumprir no ponto de vista do utilizador, aquilo que o sistema deve fornecer ao utilizador, como mostra a Figura 5. No ponto de vista do sistema, o presente documento descreve detalhadamente aquilo que o sistema deve fazer.



No ponto de vista do negócio, Figura 5, o projeto Adota.me será um concorrente à altura das aplicações existentes no mercado com o mesmo propósito. Pois o projeto em causa tem funcionalidades que outros projetos não exigiram. E também pelo facto que este projeto poderá despertar o interesse pelas pessoas em começar a ganhar hábitos de adotar um animal em vez de comprar animais.

### 3.1 Diagrama de contexto

O diagrama de contexto, em engenharia de software é o diagrama que define parte das funcionalidades do sistema, assim como os atores que interagem com o mesmo sistema. A Figura 6 mostra o diagrama de contexto apropriado à aplicação Adota.me.



Figura 6 - Diagrama de Contexto

Descrições dos atores e fluxos:

- **Adotante/Padrinho/Madrinha** – são utilizadores que podem: adotar ou apadrinhar animais; registarem-se no sistema; consultar catálogos; ver características dos animais; consultar notificações sobre as atividades que vão realizando no sistema. Exemplo: Quando fazem um pedido de adoção, se o pedido de adoção for aceite, estes utilizadores receberão uma notificação a informar que o pedido foi aceite.
- **Dador** – é o utilizador que pode: fazer pedidos de doação de animais para o sistema; registar-se no sistema;
- **Relator** – é o utilizador que pode: criar pedidos do tipo “reportar animal desaparecido” e “comunicar caso aviste animal desaparecido”; registar-se no sistema;

- Gestor – é o utilizador que pode: Consultar e validar pedidos; adicionar animais ao sistema sem ter de passar pelo método de validação.

## 3.2 User stories

*User stories* ou histórias do utilizador, são descrições concisas das necessidades do utilizador sobre o ponto de vista desse mesmo utilizador [9]. As *User stories* são sempre escritas de uma forma simples e objetiva. Têm sempre de especificar o ator, a ação e a funcionalidade desejada.

Segue-se na seguinte lista, as *User stories* e respetivos critérios de aceitação abordadas no projeto:

### User Story “Consultar catálogo”

- Como Adotante/Padrinho/Madrinha, pretendo consultar o catálogo dos animais disponíveis para adoção/apadrinhamento de modo que encontre um animal que se identifique comigo.

Critérios de aceitação:

- Aceder à página web
- Acionar “catálogo”

### User Story “Ver características”

- Como Adotante/Padrinho/Madrinha, pretendo consultar as características únicas de cada animal para que possa saber mais sobre o animal. Por exemplo, por cada animal gostaria de informar-me sobre as seguintes características: género, idade, porte, pelagem, raça, cor, boletim de vacinação, ração, estado do animal, cuidados redobrados a ter com o animal em específico.

Critérios de aceitação:

- Aceder à página web
- Acionar “Catálogo”
- Acionar “Características” sobre o animal selecionado



### User Story “Adotar”

- Como Adotante, pretendo adotar o animal que está no catálogo.

Critérios de aceitação:

- Aceder à página web
- Login válido como utilizador
- Acionar “Catálogo”
- Acionar “Características” sobre o animal à escolha
- Acionar “Adotar”
- Preencher formulário e submeter

### User Story “Apadrinhar”

- Como Padrinho/Madrinha, caso não tenha disponibilidade para adotar e possa dedicar de algum tempo que tenha disponível por semana a um animal que goste, pretendo apadrinhá-lo.

Critérios de aceitação:

- Aceder à página web
- Login válido como utilizador
- Acionar “Catálogo”
- Acionar “Características” sobre o animal à escolha
- Acionar “Apadrinhar”
- Preencher o formulário e submeter

### User Story “Visualizar dados bancários para donativo”

- Como Adotante/Padrinho/Madrinha, pretendo visualizar uma página no sistema que contenha dados bancários para contribuir para as despesas dos animais.

Critérios de aceitação:

- Aceder à página web
- Acionar “Suporte”

### User Story “Reportar desaparecido”

- Como Relator, caso algum animal doméstico que tenha e desapareça, pretendo comunicar à página web o acontecimento, adicionando os dados do meu animal.

Critérios de aceitação:

- Aceder à página web
- Login válido como utilizador
- Acionar “Pedidos”
- Acionar “Reportar desaparecido”
- Preencher formulário e submeter

### User Story “Comunicar caso aviste desaparecido”

- Como Relator, pretendo fazer um comunicado na página *web* caso aviste algum animal que se encontre desaparecido no catálogo “Desaparecidos” do sistema.

Critérios de aceitação:

- Aceder à página web
- Login válido como utilizador
- Acionar “Catálogo”
- Acionar “Desaparecidos”
- Acionar “Comunicar desaparecido” sobre o animal em questão
- Preencher o formulário e submeter

### User Story “Consultar desaparecidos”

- Como Adotante/Padrinho/Madrinha/Dador/Relator, pretendo visualizar o catálogo referente a animais que se encontrem desaparecidos. Poderá ser uma ajuda que facilita a busca por animais nesta situação.

Critérios de aceitação:

- Aceder à página web
- Acionar “Catálogo”
- Acionar “Desaparecidos”

### User Story “Ver notificações”

- Como Utilizador, pretendo consultar o histórico de notificações.  
Pretendo saber se os meus pedidos foram aceites ou não.

Critérios de aceitação:

- Aceder à página web
- Login válido como utilizador
- Acionar “Notificações”

### User Story “Consultar e validar pedidos”

- Como gestor, pretendo consultar os diversos pedidos colocados pelos utilizadores e tomar uma decisão acerca das propostas.

Critérios de aceitação:

- Aceder à página web
- Login válido como gestor
- Acionar “Pedidos”
- Selecionar opção entre “aceitar” ou “recusar” sobre o pedido em causa

### User Story “Adicionar animais”

- Como Gestor, pretendo adicionar animais ao catálogo de modo a promovê-los na adoção/apadrinhamento.

Critérios de aceitação:

- Aceder à página web
- Login válido como gestor
- Acionar “Pedidos”
- Acionar “Adicionar animal”
- Preencher formulário e submeter

### 3.3 Diagrama de casos de uso

Definidos os atores pertencentes ao sistema e os seus respetivos casos de uso, na Figura 7 segue-se, a ilustração do sistema num diagrama de casos de uso.

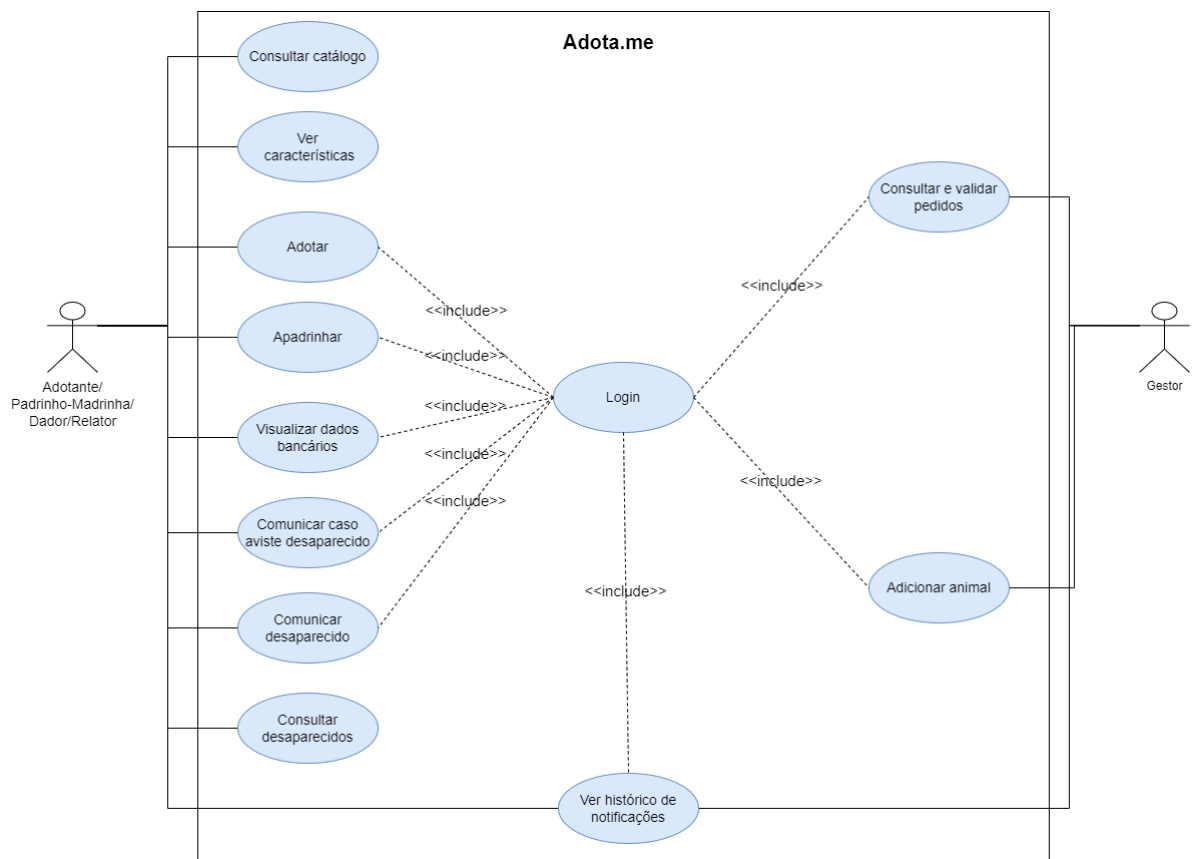


Figura 7 - Diagrama de casos de uso

Através da Figura 7, é possível ver os atores intervenientes no sistema Adota.me. Assim como é representado no diagrama, também se encontram todas as funcionalidades do sistema ligadas aos respetivos atores do sistema.

### 3.4 Diagramas de atividades e de estados

Para fácil leitura e compreensão de como são feitos os pedidos do tipo adotar, apadrinhar e adicionar animal, elaborou-se um diagrama de atividades, Figura 8, onde se encontra a informação real para o fluxo de trabalho, para estes três tipos de pedidos, entre utilizadores e o sistema.

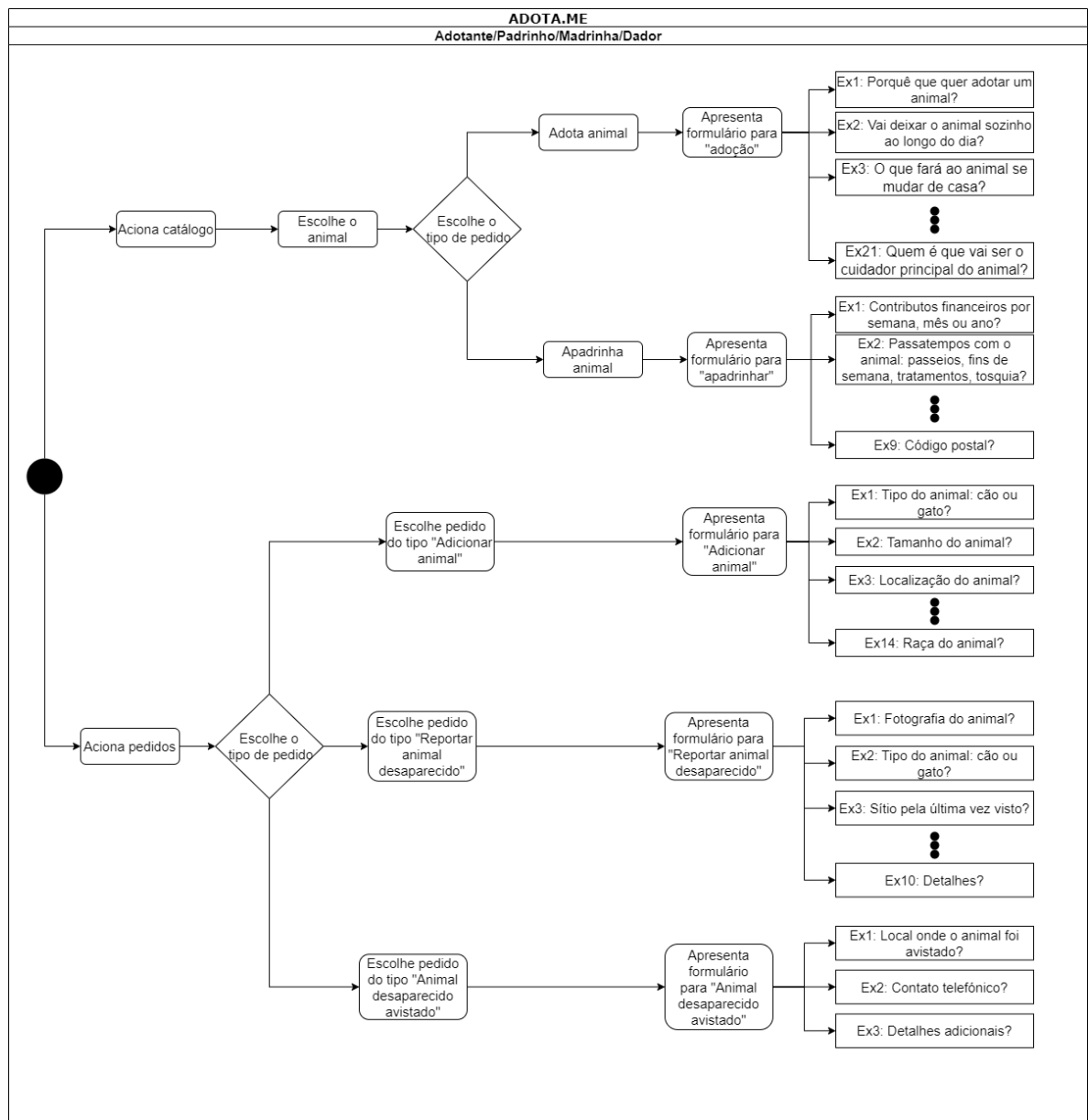


Figura 8 - Diagrama de atividades para pedidos do tipo Adotar/ Apadrinhar/ Adicionar animal/ Reportar animal desaparecido/ Avistar animal desaparecido

Analisando a Figura 8, o círculo escuro representa o início do diagrama onde se encontra um caminho bidirecional. Seguindo o caminho superior da figura, é mostrado o caminho que levará o utilizador a selecionar o catálogo, escolher o animal e por último decidir o tipo de pedido, entre adoção ou apadrinhamento para o animal escolhido. Mais à frente no diagrama de atividades, encontra-se uma amostra de perguntas reais e implementadas no sistema, que o utilizador terá de responder consoante o tipo de pedido selecionado. Para cada tipo de pedido existem diversas perguntas diferentes entre os

questionários. Por exemplo, para o pedido “adotar animal” existem 21 perguntas. Para o pedido “apadrinhar animal” existem 9 perguntas.

Seguindo o caminho inferior da Figura 8, são mostrados os passos que o utilizador terá de realizar para fazer pedidos do tipo “adicionar animal”, “reportar animal desaparecido” e “animal desaparecido avistado”. Por fim, também se encontra uma amostra de perguntas retiradas dos formulários implementados. Para o pedido “adicionar animal” existem 14 perguntas. Para o pedido “reportar animal desaparecido” existem 10 perguntas. E por fim, para o pedido “animal desaparecido avistado” existem 3 perguntas.

De forma a melhor se compreender o processo desde que os pedidos do tipo “adotar”, “apadrinhar”, “adicionar animal” e “reportar animal desaparecido” sejam criados até serem validados pelo gestor, encontra-se na Figura 9 o diagrama de estados detalhando o funcionamento face a esta situação.

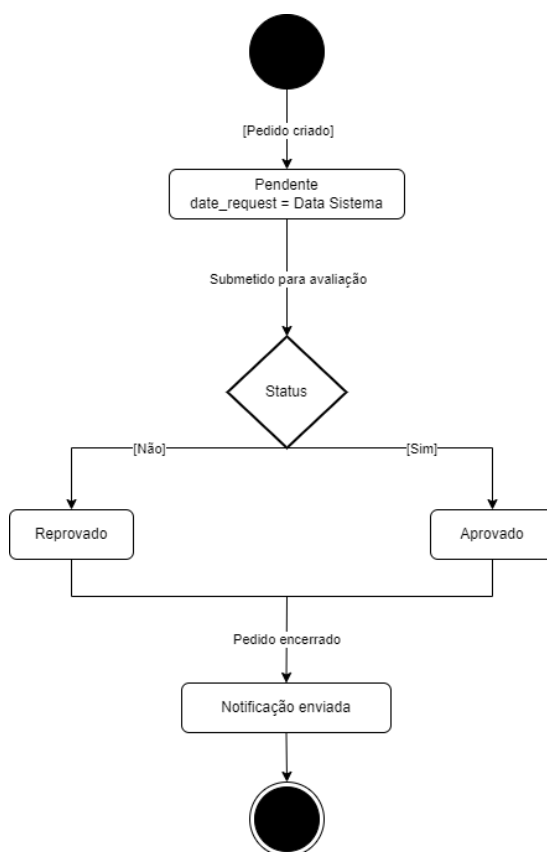


Figura 9 - Diagrama de estados quando é criado um pedido

Como se pode observar pela Figura 9, o círculo escuro com uma seta de transição representa o início do diagrama. Quando o pedido é criado, de imediato fica com o status ‘Pendente’. Além disso, quando o pedido é criado, a data do pedido fica registada com a data do sistema. E o pedido fica submetido para avaliação, ou seja, o pedido é enviado para o gestor, para que este possa analisar e validar o pedido.

Restam dois status para atribuir ao pedido. O status ‘Aprovado’ e ‘Reprovado’. Se o gestor aceitar o pedido, este fica com o status aprovado, o pedido dá-se como encerrado e de seguida o utilizador receberá a notificação sobre o seu pedido como foi aceite. Por outro lado, se o gestor reprovar o pedido, este fica com o status reprovado, o pedido dá-se como encerrado e de seguida o utilizador receberá a notificação sobre o seu pedido como foi negado.

O motivo pela qual o pedido é aceite, é devido ao facto de o gestor concordar com os dados submetidos num dos pedidos feitos pelo utilizador. O gestor viu coerência e credibilidade no pedido imposto pelo utilizador. Por outro lado, o motivo pela qual o pedido é negado pelo gestor, deve-se ao facto de, quando o gestor analisa o pedido feito pelo utilizador, vê que o utilizador não tem coerência ou condições suficientes no seu pedido para ser aprovado.

### 3.5 Diagrama de classes

Diagrama de classes é a representação da estrutura do sistema através de classes. É constituído por entidades, atributos e relacionamentos. Cada classe é única e possui um grupo de atributos e operações que descrevem a sua entidade.

Ao elaborar um sistema vão existir várias classes que podem estar ou não relacionadas com outras classes. Por norma existem os relacionamentos de: 1...1 e \*...\* que se estenderá para uma relação entre classes de 1...\*.

Uma boa prática que se deve ter durante a construção de uma classe é seguir um conjunto de regras que levará a classe estar “otimizada”. A esse conjunto de regras a seguir dá-se o nome de normalização.



Para a classe se encontrar no primeiro nível de normalização, ao qual se chama 1FN, a classe deve:

- ter uma chave primária; A chave primária é única e nunca pode ser nula.
- cada coluna é atômica;
- não há grupos repetidos de colunas.

Ao validar estas três regras podemos avançar para a 2FN que deve encontrar-se:

- na 1FN;
- cada coluna não chave depende da totalidade da chave primária.

Por último, a classe deve encontrar-se:

- na 2FN;
- cada coluna não chave só é dependente da chave primária.

A Figura 10 é a representação do diagrama de classes elaborado para a solução do sistema Adota.me. Durante a construção deste diagrama teve-se em conta o respeito por todas as regras acima descritas.

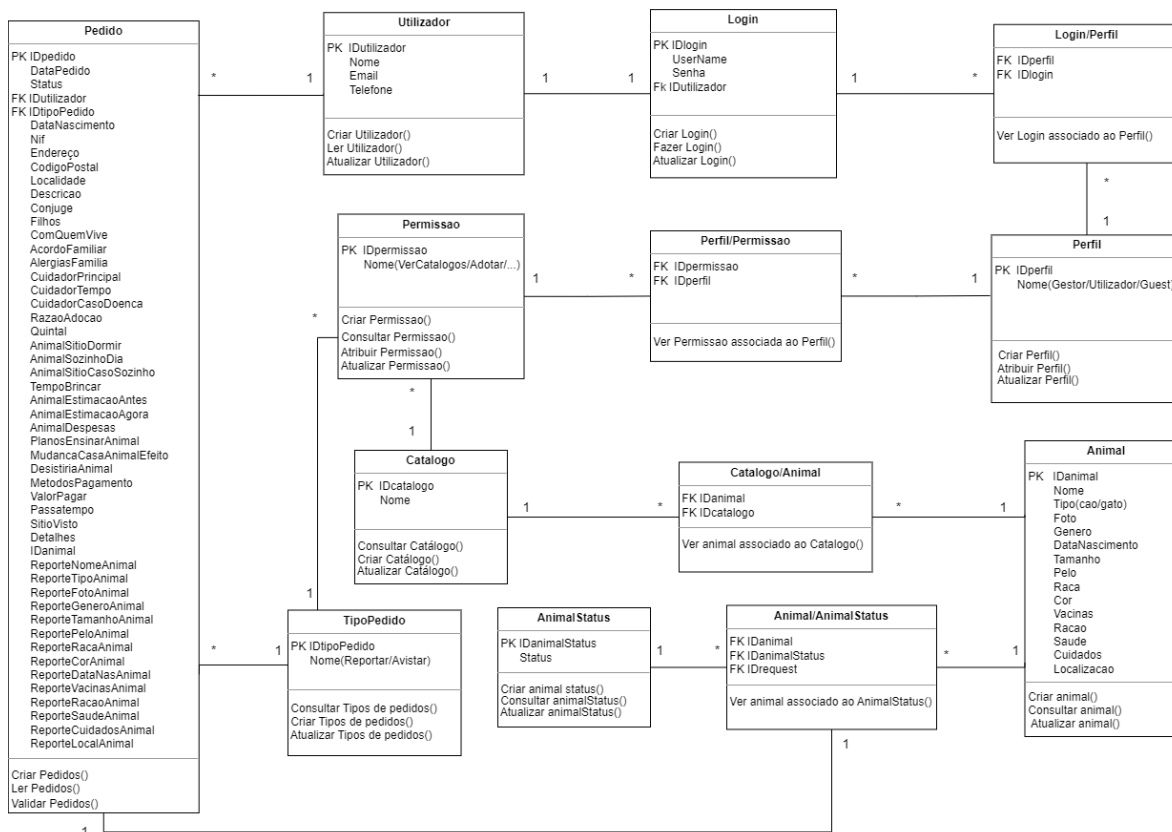


Figura 10 - Diagrama de Classes

A Figura 10 é a representação do diagrama de classes do projeto Adota.me. É uma ótima ajuda usar este diagrama pois mapeia de forma clara a estrutura do sistema referenciando todas as classes constituintes do projeto.

Durante a elaboração do diagrama de classes, pensou-se em implementar no sistema, um tipo de mecanismo onde se atribuíam privilégios do sistema a um determinado tipo de utilizador. Para concretizar esta implementação decidiu-se incluir no diagrama de classes da Figura 10, as classes “Perfil” e “Permissao”.

Privilégios do sistema são funcionalidades do sistema, que aqui se entendem por permissões. Por exemplo, a classe “Permissao”, contém um “IDpermissao” e um “nome”. Na qual o “nome” contém funcionalidades do sistema que são: ver catálogos, ver características dos animais, fazer pedidos do tipo adotar, apadrinhar, adicionar animal, reportar animal desaparecido e avistar animal desaparecido, ver dados bancários, ver notificações e validar pedidos.

Para que se possa atribuir privilégios aos utilizadores, é necessário que os utilizadores possuam pelo menos um tipo de perfil, da classe “Perfil”. A classe “Perfil” é constituída por um “IDperfil” e um “nome”, nome do perfil. Neste caso, o nome do perfil poderá ser do tipo: utilizador, gestor ou convidado.

Tendo privilégios do sistema associados aos perfis, pretendem-se implementar as seguintes condições:

- O perfil “convidado” tem os seguintes privilégios: ver catálogos, ver características dos animais e ver dados bancários.
- O perfil “utilizador” tem os seguintes privilégios: ver catálogos, ver características dos animais, fazer pedidos do tipo adotar, apadrinhar, adicionar animal, reportar animal desaparecido e avistar animal desaparecido, ver dados bancários e consultar notificações.
- O perfil “gestor” tem os seguintes privilégios: ver catálogos, ver características dos animais, fazer pedidos do tipo adotar, apadrinhar, adicionar animal, reportar animal desaparecido e avistar animal desaparecido, ver dados bancários, ver notificações e validar pedidos.

Desta maneira é possível construir um sistema fechado, organizado, amigo do consumidor e mais seguro. O motivo pela qual se teve esta decisão é o simples facto de que qualquer utilizador não registado no sistema, possua um perfil de convidado. Ou seja, para quem não queira registar-se no sistema, também não fica totalmente impedido de usufruir sobre certas funcionalidades do sistema. E é mais seguro porque não é vantajoso realizar um pedido de adoção, por exemplo, sem estar registado no sistema.



## 4 Tecnologias e ferramentas

No capítulo anterior foram reunidas as informações acerca dos requisitos funcionais do sistema. Foram identificados casos de uso e os atores do sistema, assim como foi elaborado um diagrama de classes capaz de guardar a informação útil para solucionar o problema.

Neste capítulo segue-se uma breve explicação sobre cada uma das tecnologias utilizadas e que mais se identificavam com desenvolvimento do projeto Adota.me.

### 4.1 Visual Studio Code

Visual Studio Code (VSCode) é um editor de código disponível para *Windows*, *macOS* e *Linux*. VSCode tem suporte integrado para *JavaScript*, *Node.js* e tem um ecossistema de extensões para outras línguas de programação, tais como *C++*, *C#*, *Java*, *Python*, *PHP*, etc. [10] Também prevalece de diversas extensões que facilitam o trabalho durante a fase de implementar código. Por exemplo, as extensões que auxiliaram durante a realização do projeto foram:

- Auto Rename Tag v0.1.10: Renomeia automaticamente *tags HTML/XML*
- Code Spell Checker v2.2.5: Ajuda a detetar erros ortográficos comuns;
- Color Highlight v2.5.0: Estiliza cores *CSS* encontradas nos *scripts*;
- EditorConfig for VS Code v0.16.4: Substitui as configurações de trabalho do utilizador pelas configurações encontradas nos arquivos;
- EJS language support v1.3.1: Destaca a sintaxe *EJS*, *Javascript* e *tags HTML*. Inclui autocompletar *Javascript*;
- ESLint v2.2.6: Ferramenta de análise de código estático para identificar padrões problemáticos encontrados no *JavaScript*;
- HTML CSS Support v1.13.0: *HTML* e complementos de atributos de classes;
- IntelliCode v1.2.22: Fornece assistência através de IA para desenvolvimento de funcionalidades escritas em *Python*, *TypeScript/JavaScript* e *Java*;
- IntelliCode API Usage Examples v0.1.2: Fornece diretamente exemplos de código;

- IntelliCode Completions v1.0.14: Prevê linhas inteiras de código com base no contexto atual da aplicação;
- JavaScript (ES6) code snippets v1.8.0: Contém excertos de código para *JavaScript* na sintaxe *ES6*;
- Path Intellisense v2.8.1: *Plugin* que completa automaticamente nomes de arquivos;
- Prettier – Code formatter v9.5.0: É um formatador de código opinativo;
- Todo Tree v0.0.215: Pesquisa pelas *tags* de comentários do tipo *TODO* no espaço de desenvolvimento;

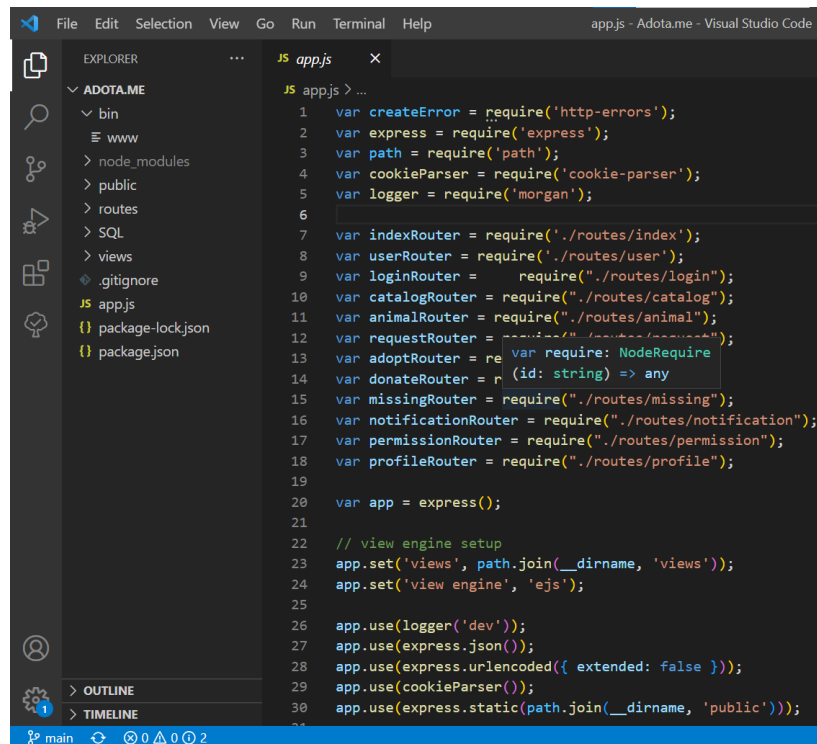


Figura 11 - Ambiente de desenvolvimento do VSCode

A Figura 11 mostra o ambiente de desenvolvimento do Visual Studio Code. Podemos ver as pastas, nas quais as principais são: rotas para mapear os caminhos da aplicação *web*, vistas que são as interfaces do utilizador, a pasta *SQL* que contém ficheiros relativos à construção da base de dados, e o ficheiro *app.js* onde se encontra implementado o servidor

do projeto. Podemos visualizar também os ficheiros secundários que seriam: *package.json* e *.gitignore*.

Todos estes ficheiros são elementos constituintes do projeto Adota.me. É neste ambiente constituído por pastas e ficheiros, onde se vai desenvolver todo o trabalho necessário até ter-se o *software* final.

## 4.2 JavaScript e NodeJS

JavaScript é uma linguagem de programação de alto nível. É uma linguagem caracterizada por ser orientada a objetos, imperativa, declarativa ou funcional. É tanto utilizada para o desenvolvimento *backend* como para *frontend*, dito isto por ser conhecida e talvez a mais utilizada num ambiente de produção *web*.

NodeJS é a Plataforma *open-source* que interpreta código JavaScript, sendo o seu principal propósito, construir aplicações *i/o (input/output)* nas quais serão *server-side* e *event-driven*.

Parte do *backend* implementado no projeto foi realizado em *NodeJS*. A Figura 12 mostra parte do *NodeJS* implementado no projeto Adota.me.



```
router.get("/:type", function (req, res, next) {
  var type = req.params.type;
  var catalog = req.query.catalog || "Adotar e Apadrinhar";

  db.any(
    `select a.*, AGE(now(), a.birth_date) as age from adotame.animal a
    inner join adotame.catalog_animal ca
    on a.id_animal = ca.id_animal
    inner join adotame.catalog c
    on c.id_catalog = ca.id_catalog
    where a.type ${type == "all" ? "<>" : "="} $1
    and c.name = $2
    and a.id_animal not in (
      select a.id_animal from adotame.animal a
      inner join adotame.catalog_animal ca
      on a.id_animal = ca.id_animal
      inner join adotame.catalog c
      on c.id_catalog = ca.id_catalog
      inner join adotame.animal_animal_status aas
      on a.id_animal = aas.id_animal
      inner join adotame.animal_status as2
      on as2.id_animal_status = aas.id_animal_status
      where a.type ${type == "all" ? "<>" : "="} $1
      and c.name = $2
      and as2.status = 'Adotar'
      or as2.status = 'Apadrinhar'
      or as2.status = 'Reportar desaparecido'
    )`,
    [type, catalog]
  )
  .then((rows) => {
    console.log(rows);
    res.render("catalog/index", {
      type: req.params.type,
      animals: rows,
      title: "Catálogo",
      paragraph: "Esperamos que encontre algum amiguinho novo!!",
    });
  })
  .catch((err) => {
    console.log(err);
    res.render("error", {
      error: err,
      message: "Not possible render this page",
    });
  });
});
```

Figura 12 - Exemplo NodeJS

Como se pode visualizar através da Figura 12, vêm-se variáveis declaradas, *queries* e *promises* respectivas à rota. Isto tudo escrito em JavaScript, mas executado pelo ambiente NodeJS, interpretador do *JavaScript*.

Uma *promise* em JavaScript representa um objeto onde é utilizado para desenvolver operações assíncronas. Estes objetos, terão sempre um dos seguintes estados: pendente, realizada ou rejeitada.



### 4.3 ExpressJS

ExpressJS é a uma *framework* baseada em NodeJS, para a qual serve para construir aplicações *web* onde se usam abordagens e princípios do NodeJs *event-driven*.

No projeto Adota.me foi usada a *framework* ExpressJS para implementar o servidor responsável por processar todos os pedidos realizados na plataforma *web*.

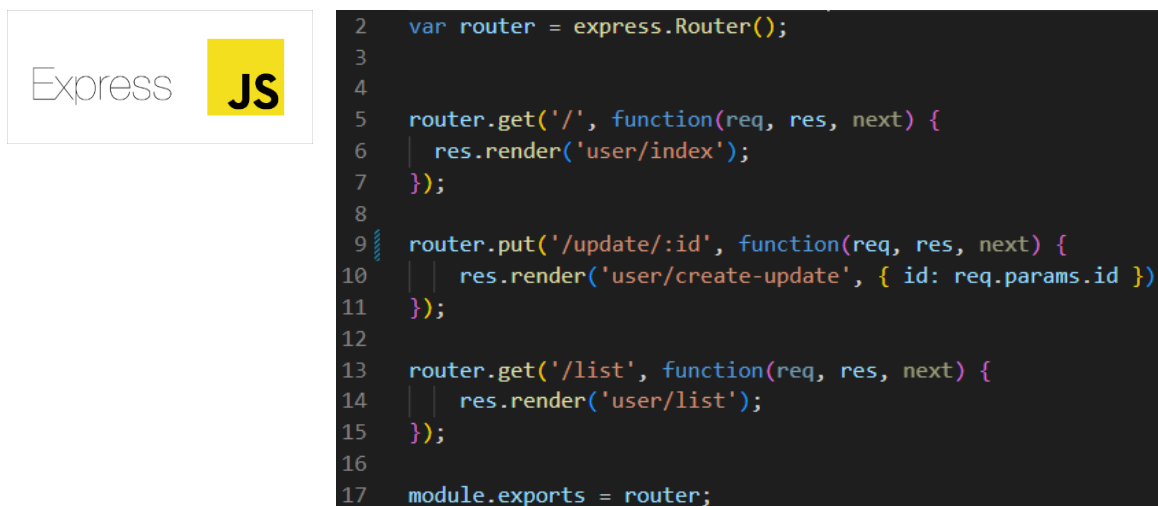


Figura 13 - ExpressJS

A Figura 13 é exemplo do que o sistema irá apresentar, em forma de página *web* consoante a rota inserida pelo utilizador. Através desta figura é possível saber o que se tem de escrever como endereço no *browser* e o que será *renderizado* consoante o endereço. Caso o endereço escrito no *browser* não se encontre implementado na rota, será visto no *browser* uma mensagem a dizer “Página não encontrada”.

### 4.4 PostgreSQL

PostgreSQL, mais conhecido pelo nome *Postgres*, é o *software open-source* orientado às bases de dados relacionais, com extensão da linguagem de programação *SQL*. [11]

É utilizado para armazenar dados primários ou para construir *Data Warehouse's* para aplicações *web*, móveis ou analíticas.

A Figura 14 mostra uma *query* real do projeto Adota.me escrita em *sql*.



```

27•select * from adotame.permission p
28 inner join adotame.profile_permission pp
29 on p.id_permission = pp.id_permission
30 inner join adotame.profile p2
31 on p2.id_profile = pp.id_profile
32 where p2.name = 'admin';

```

	id_permission	name	id_permission	id_profile	id_profile
1	09dc1448-6ed1-4325-9939-6a9eba9d3571	Gerir Sistema	09dc1448-6ed1-4325-9939-6a9eba9d3571	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
2	adb46aa1-9673-43c1-954f-1e2006fd5a86	Validar Pedidos	adb46aa1-9673-43c1-954f-1e2006fd5a86	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
3	045e5f7-ad39-4c6c-bbf6-bfb1d40224ae	Reportar Desaparecido	045e5f7-ad39-4c6c-bbf6-bfb1d40224ae	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
4	b61528cd-66fb-4441-bd2e-387901ab61b9	Comunicar Aviso Desaparecido	b61528cd-66fb-4441-bd2e-387901ab61b9	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
5	f5aa0738-916f-48c5-8860-f753dab79719	Adicionar Animal	f5aa0738-916f-48c5-8860-f753dab79719	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
6	54781702-b4cb-4dd5-aae2-b13215bc0f24	Ver Catalogos	54781702-b4cb-4dd5-aae2-b13215bc0f24	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
7	55e6963b-9273-475f-8710-c7358a4e3034	Ver Caracteristicas dos Animais	55e6963b-9273-475f-8710-c7358a4e3034	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
8	213a32ef-a71c-42b6-be59-f3629c73c753	Adotar	213a32ef-a71c-42b6-be59-f3629c73c753	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
9	e0a8975-34ac-4a30-ba22-111945d9b682	Apadrinhar	e0a8975-34ac-4a30-ba22-111945d9b682	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
10	eb04673-a093-4622-8717-124bab4e7ef5	Doativos	eb04673-a093-4622-8717-124bab4e7ef5	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b
11	68417d5b-abef-40ed-b679-f59326742c11	Ver Historico de notificacoes	68417d5b-abef-40ed-b679-f59326742c11	2dc18adb-3a0f-4702-8f3f-7b2def36355b	2dc18adb-3a0f-4702-8f3f-7b2def36355b

Figura 14 - PostgreSQL

A Figura 14, além de mostrar a construção de uma *query* construída em linguagem *Sql*, mostra também o resultado da *query*. Esta *query* trata de visualizar as permissões que o perfil chamado “*admin*” tem associadas.

Já a própria *query* implementada mostra uma operação do tipo *select* à tabela *permission* do *schema* ‘*adotame*’. Onde a partir da tabela *permission* realizam-se *inner joins* entre as tabelas *profile\_permission* e *profile*, também pertencentes ao *schema* ‘*adotame*’. A palavra-chave *inner join*, seleciona registos que possuem valores correspondentes entre as tabelas *permission*, *profile\_permission* e *profile*. Ao realizar *inner join* na *query* tem-se atenção em igualar as chaves estrangeiras às chaves primárias das respetivas tabelas para não causar erros, como se pode ver nas linhas 29 e 31 da *query* na Figura 14. Por fim, na linha 32 na representação da *query*, foi selecionado o atributo *name* da tabela *profile* onde se pretende que o valor do *name* seja ‘*admin*’. Nesta linha, ‘*p2*’ representa a tabela *profile*, ‘*p2*’ é o *alias* à tabela *profile*.

#### 4.5 Dbeaver

Dbeaver é o software de administração de bases de dados. Foi uma ferramenta útil no aspeto em que oferecia uma interface da base de dados criada para o projeto. Neste ambiente,

encontra-se o *schema* criado, assim como todas as tabelas *populadas*, *views*, *functions*, *sequences*, entre outros objetos pertencentes ao tema de desenvolvimento de bases de dados.

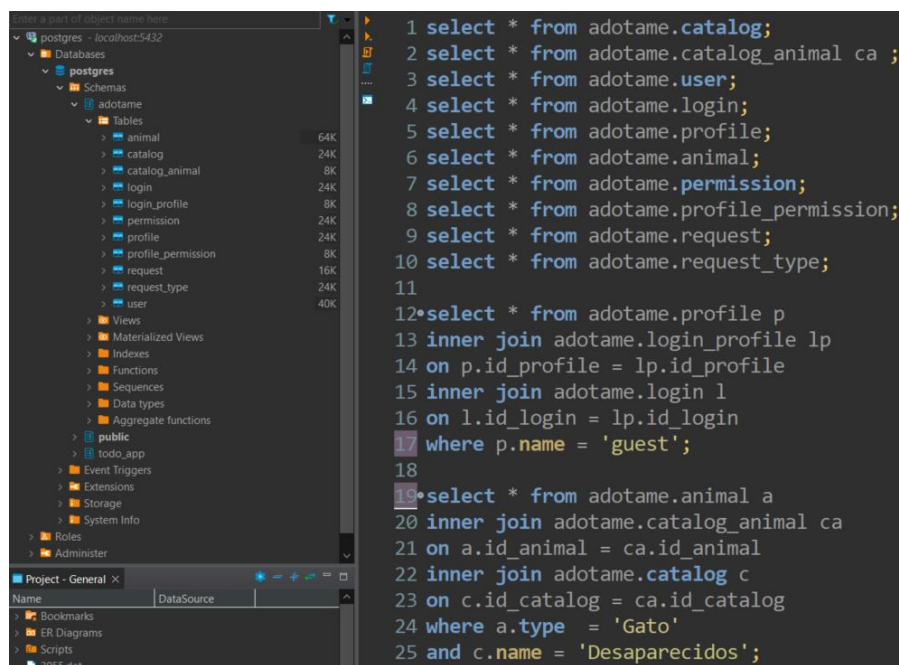


Figura 15 - Ambiente visual e de desenvolvimento Dbeaver

Como mostra a Figura 15, a ferramenta permite a construção de *queries*, tal como o manuseamento da própria base de dados construída. Todas as *queries* que se encontram nas rotas do projeto Adota.me, primeiramente foram desenvolvidas no ambiente de desenvolvimento *Dbeaver*. Este é ambiente de desenvolvimento mais apropriado para construir *queries* ou outro tipo de matéria relacionada com base de dados durante o estágio. Porque *DBeaver* é um *IDE* (*Integrated Development Environment*) relacionado às bases de dados. E para garantir que no desenvolvimento de qualquer *query* necessária para o projeto fosse bem construída, *DBeaver* é o *IDE* ideal para testar a *query* realizada.

## 4.6 SQL Shell

SQL Shell é o terminal onde permite executar comandos *SQL* com intuito de manusear diversas formas a base de dados do projeto. Esta ferramenta possibilitou: a criação do *schema*, a criação das tabelas e seus atributos, a inserção de registos nos campos das tabelas e a construção de *queries*.

```
SQL Shell (psql)
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
postgres=# \i C:/Users/ProjectBox/Documents/Adota.me/SQL/createTables.sql
psql:C:/Users/ProjectBox/Documents/Adota.me/SQL/createTables.sql:1: NOTICE:  schema "adotame" already exists
CREATE SCHEMA
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
postgres=# \i C:/Users/ProjectBox/Documents/Adota.me/SQL/seedData.sql
DO
postgres=# select * from adotame.user;
      id_user      | name | email      | phone
-----+-----+-----+-----
 3a40e8bc-3195-499a-9dfb-6ef2cd68af29 | admin | admin@gmail.com | 909897987
 baeaa802-e478-4c75-897b-869d3d65f19f | guest | guest@guest.com | 286594743
 538d4c33-8582-4f93-ac85-79b17fa5607e | user  | user@user.com   | 984759408
(3 rows)

postgres=# select * from adotame.login;
      id_login      | username | password |      id_user
-----+-----+-----+-----
 96bca662-385e-4e53-9376-5c5d756ff1e1 | admin   | admin123 | 3a40e8bc-3195-499a-9dfb-6ef2cd68af29
 04fc5199-6c53-4453-8275-fc86a73c7020 | guest   | guest123 | baeaa802-e478-4c75-897b-869d3d65f19f
 8fc21106-6f33-413d-b66c-c89c7db7da52 | user    | user123  | 538d4c33-8582-4f93-ac85-79b17fa5607e
(3 rows)

postgres=# select id_user from adotame.user where name = 'admin';
      id_user
-----
 3a40e8bc-3195-499a-9dfb-6ef2cd68af29
(1 row)
```

Figura 16 - *SQL Shell*

Como se vê na Figura 16, a ferramenta *SQL Shell* permite executar scripts em formato SQL, visualizar as tabelas com registos e construir queries. Foi útil trabalhar com esta ferramenta pois possibilitou o facto de aprender e utilizar comandos *Shell* na base de dados construída para o projeto.

## 4.7 HTML5

HTML5, acrónimo para *Hypertext Markup Language* (nível 5), é uma linguagem de marcação para implementar a estrutura de qualquer página *web*.

O HTML5 é a versão mais recente do tradicional *HTML*. Contudo, HTML5 traz novas funcionalidades e algumas delas são:

- Suporte para áudio, vídeo, e outros conteúdos multimédia
- O código escrito em *HTML5* suporta compatibilidades multiplataforma
- Novos eventos e tags



```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8  </head>
9  <body>
10
11 </body>
12 </html>

```

Figura 17 - HTML5

A Figura 17 mostra a estrutura de qualquer documento HTML. Todos os projetos *web* possuem este pilar representativo e marcado através de várias *tags* HTML.

## 4.8 CSS3

CSS3, acrónimo para *Cascading Style Sheets* (nível 3) é a linguagem de marcação que permite adicionar estilos e fazer animações aos elementos criados em *HTML*. A versão do CSS3 traz consigo suporte a *layouts* responsivos, novas transições e animações, sombras e melhores estilos.

No projeto Adota.me, as classes criadas em CSS encontram-se num ficheiro à parte, o que permite uma melhor organização em termos de ficheiros e pastas constituintes do projeto como um todo.



```
.fill {
  display: flex;
  justify-content: center;
  align-items: center;
  overflow: hidden;
}

.fill img {
  object-fit: cover;
  min-width: 100%;
  min-height: 100%;
}

.border-middle{
  border: 3px solid #CE8927;
  border-radius: 1rem;
  width: 20vw;
  margin: auto;
  min-width: 300px;
  box-shadow: 1px 1px 30px;
}
```

Figura 18 - CSS3

A Figura 18 mostra apenas três das muitas classes que existem no projeto Adota.me construídas em CSS. Uma classe em CSS, é um conjunto de propriedades que irão definir o estilo de um elemento HTML. Existem classes escritas em CSS que determinam certo comportamento no estilo da página *web*. A essas classes chamam-se de *pseudo-classes*.

## 4.9 Bootstrap

Bootstrap é uma *framework* para desenvolvimento *front-end* de páginas *web* e aplicações *web*, com o intuito de serem responsivas e a pensar no desenvolvimento *mobile-first*.

Na realização do projeto, o Bootstrap foi muito útil devido à utilidade das classes que a própria *framework* traz consigo implementadas. O que faz com que não se perda tempo na construção de novas classes em CSS puro.

A versão do Bootstrap utilizada no projeto Adota.me foi a versão 5.2 [12].



```
<div class="d-flex  
justify-content-center  
align-items-center">
```

Figura 19 – Bootstrap

A Figura 19 representa uma *div* com uma classe proveniente da *framework* Bootstrap. Podemos afirmar que esta *div* encontra-se alinhada ao centro, quer na horizontal quer na vertical. Porém para representar esta classe em CSS puro, teria de ser escrita com mais palavras, teríamos de dar um nome à classe, e ainda teríamos de chamar a nossa classe feita dentro da *div*. Para simplificar esse outro efeito, passam-se as propriedades desejadas diretamente dentro da *div* através do atributo *class*.

#### 4.10 Git/GitHub

GitHub é um sistema de gestão de versões grátis e *open-source* que usa o *Git* [13]. Uma das primeiras fases na construção do projeto, foi criar um repositório local que tem a pasta raiz que contém todas as pastas e ficheiros do projeto. Ao longo do desenvolvimento do código, cada funcionalidade bem-sucedida, ou parte dela, é merecedora de um *commit*. Um *commit* é uma mensagem curta que descreve resumidamente o que foi feito perante a funcionalidade que se está a implementar.

Quando se pretende expor o repositório remotamente no *software* *GitHub*, usa-se a funcionalidade *Git-push*. Ao fazer *push*, todas as pastas e ficheiros modificados, e *commits* locais, serão enviados para o repositório remoto que está no *GitHub*.



```
git add .  
git commit -m "Feita a validação do login de um utilizador registado no sistema"  
de um utilizador registado no sistema  
n(-)  
git push
```

Figura 20 - Git e GitHub

Durante a implementação do projeto, foi guardado todo o trabalho feito no repositório *online*, no seguinte link: <https://github.com/goncaloMRsilva/Adota.me>.

Na Figura 20 está representado em forma de comandos *git*, como se deve fazer um commit e de seguida enviar para o GitHub.

#### 4.11 Gitignore

Gitignore “comunica” ao *Git* quais são os ficheiros que devem ser ignorados quando é feito *commit* no repositório do projeto que se encontra no *GitHub*.

No *site Gitignore* foi criado o ficheiro “.gitignore” que filtra todo o lixo *Windows*, *macOS*, *node*, etc. Por consequente este ficheiro foi colocado para a pasta raiz do projeto para que não haja ficheiros desnecessários no repositório remoto do projeto.



Figura 21 - Gitignore

A Figura 21 mostra uma parte do *site* Gitignore, onde na barra de pesquisa inserem-se os nomes dos ficheiros que se pretendam ignorar.

#### 4.12 Figma

Figma é uma ferramenta para criação e partilha de *designs* sobre *layouts* de *websites*, aplicações, logotipos e muito mais [14].

Todos os *layouts* do sistema Adota.me, criados pelos *designers* Camila Paya e Carlos Daniel, encontram-se partilhados na plataforma Figma. Através do *software* Figma



consegue-se ter acesso a todos os *layouts* desenhados pelos *designers*, exemplos na Figura 22. Além de se poderem visualizar todas as páginas desenhadas para o sistema, é possível ver comprimentos e dimensões de elementos únicos pertencentes aos *layouts* desenhados. Por exemplo, um botão do sistema é um elemento único e que se encontra em vários *layouts* desenhados. Através do Figma é possível saber sobre este botão, o espaço real que terá de ocupar no ecrã. Pode-se saber também as cores do botão, o estilo e fonte das letras pertencentes ao botão, etc.

Sendo assim, o software Figma é uma ótima ajuda durante a implementação *frontend* para o projeto, pois informa o programador sobre características e propriedades dos diversos elementos que constituem os *layouts* desenhados.

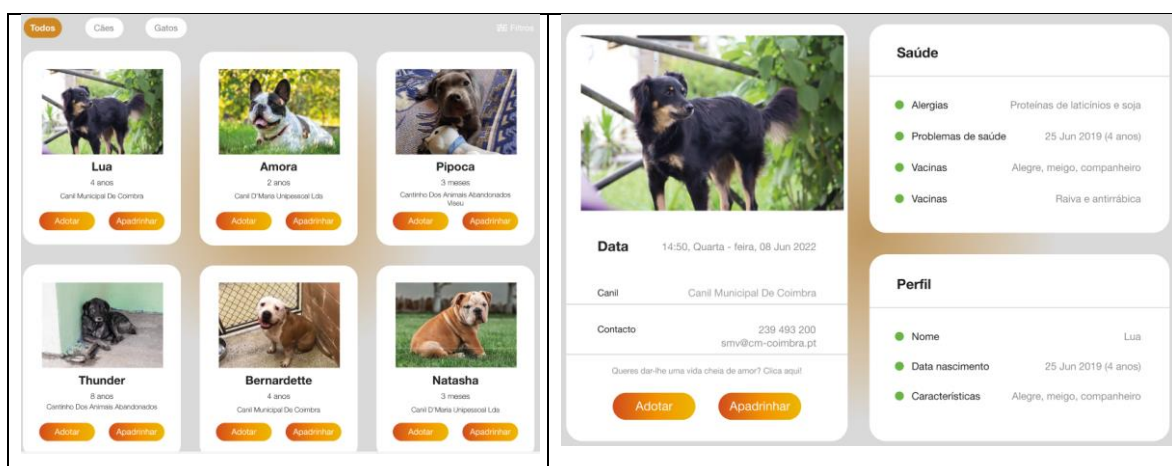


Figura 22 - Layouts do projeto no Figma

A Figura 22 mostra a página da plataforma Figma, onde se encontram os *layouts* do projeto Adota.me. A imagem à esquerda apresenta o catálogo dos animais. A imagem à direita representa o perfil apenas de um animal.

#### 4.13 Draw.io

O Draw.io é uma aplicação *online* gratuita, onde foram realizados todos os fluxogramas e diagramas, da linguagem *Unified Modeling Language* (UML).

UML é uma linguagem visual que serve para especificar, planejar, construir e modular documentos, diagramas e fluxogramas sobre projetos dentro do campo da engenharia de software.

No presente relatório, todos os diagramas realizados estão representados em notação UML. Por exemplo, ver Figura 10. Esta notação ajuda a simplificar funcionalidades complexas em casos mais acessíveis de visualizar, para isso, é necessário recorrer ao fluxograma ou diagrama mais adequado para resolver tal problema.

#### 4.14 jQuery

jQuery é uma biblioteca de funções escritas em JavaScript. Pensada para facilitar o trabalho dos programadores, principalmente no requisito de criar efeitos e animações nas páginas *web*. [15]



```
$("#registryModal").modal("hide");  
$("#modal-thankModal").modal("show");
```

Figura 23 – jQuery

A Figura 23 mostra um pedaço de código em sintaxe jQuery. A primeira linha de código é passado, como parâmetro, o *id* da modal registo, na qual o nome do *id* é “*registryModal*” e de seguida no método *modal* é passado como parâmetro a *string* *hide*. Em resumo, o que faz esta linha de código é esconder a modal com determinado *id*. A seguinte linha de código faz o mesmo que a linha de código anterior só que faz com a modal com o *id* “*modal-thankModal*” seja *visível*.

Uma modal nada mais é do que uma caixa de diálogo, uma janela/*popup*, que aparece sobre a página corrente.

## 5 Implementação

Como foi possível ver no capítulo anterior, foram caracterizadas todas as tecnologias que este projeto assim exigiu. Tecnologias escolhidas pelo supervisor da empresa, na qual são tecnologias atuais e que permitiram a construção do sistema Adota.me.

No presente capítulo, vão ser pormenorizadas todas as etapas desenvolvidas para uma eficiente solução do projeto Adota.me, que se realizou em contexto de estágio.

A primeira etapa do estágio, como já referida foi recolher funcionalidades e estruturar os diversos casos de uso. A segunda etapa do estágio foi debater e estudar sobre quais tecnologias serviriam melhor para a concretização do projeto. Nomeadamente, acerca das linguagens de programação que iriam ser usadas quer para a construção da base de dados, quer para a implementação do *backend*, quer para o desenvolvimento do *frontend*. Uma vez que, parte da implementação do *backend* foi realizada em Node.js e para o desenvolvimento *frontend*, recorreu-se à *framework* Bootstrap. Todos os layouts do projeto encontram-se no *software* Figma, e foi por meio deste *software* que toda a implementação *frontend* foi guiada.

### 5.1 Instalações das bibliotecas e dependências

Neste tópico serão apresentadas as dependências que o projeto necessita para funcionar corretamente. Dependências são bibliotecas externas que a aplicação vai utilizar, de modo a facilitar a implementação ao programador.

Na pasta do projeto, as dependências situam-se num ficheiro chamado *package.json*. Este ficheiro nada mais é do que um ficheiro no formato *JSON* (*Java Script Object Notation*), ou seja, um ficheiro de objetos onde se encontram as dependências do sistema Adota.me.

A Figura 24 mostra a estrutura dos arquivos do projeto Adota.me, realçando o arquivo *package.json*.

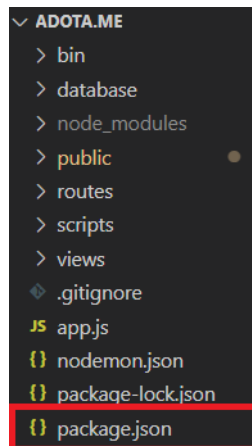


Figura 24 - Arquivos do projeto

Para a criação do ficheiro em questão, é necessário executar na consola do *VSCode*, o comando proveniente do *Node*, que é: “*npm init*” ou “*npm initializer*” [16]. Este comando ao ser executado mostra algumas questões de resposta opcional sobre: o nome que se quer atribuir ao *package* do projeto, uma descrição, um endereço de repositório *git*, o nome do autor, entre outras. Além disto, este comando ao ser executado, informa ao ambiente de desenvolvimento que vamos dar início ao projeto.

A Figura 25 mostra o conteúdo do ficheiro *package.json*. Este ficheiro mostra quais são as bibliotecas instaladas no projeto, tal como, o comando que é necessário executar para pôr o projeto a executar no *browser*.

```

1  {
2    "name": "adota.me",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www",
7      "start:dev": "SET DEBUG=adota.me:* && nodemon ./bin/www"
8    },
9    "dependencies": {
10     "bcrypt": "^5.0.1",
11     "connect-livereload": "^0.6.1",
12     "cookie-parser": "~1.4.4",
13     "debug": "~2.6.9",
14     "ejs": "^3.1.8",
15     "express": "~4.16.1",
16     "express-ejs-layouts": "^2.5.1",
17     "http-errors": "~1.6.3",
18     "livereload": "^0.9.3",
19     "morgan": "~1.9.1",
20     "passport": "^0.6.0",
21     "passport-local": "^1.0.0",
22     "pg-promise": "^10.11.1",
23     "uuid": "^8.3.2"
24   },
25   "devDependencies": {
26     "nodemon": "^2.0.19"
27   }
28 }

```

Figura 25 - Ficheiro *package.json*

Como se pode visualizar a partir da Figura 25, o ficheiro *package.json* contém quatro objetos identificáveis. O objeto principal tem o mesmo nome do ficheiro e contém: o nome do projeto, a versão e os objetos chamados de: *scripts*, *dependencies* e *devDependencies*.

Primeiramente vê-se o objeto chamado *devDependencies* que se encontra na Figura 25, iniciado na linha 25. O nome deste objeto traduzido significa “dependências de desenvolvimento”, o que quer dizer que esse objeto só será útil enquanto o projeto se encontra em processo de desenvolvimento.

Como se pode ver, o objeto *devDependencies*, contém um atributo chamado *nodemon* com valor da versão mais recente do *nodemon* [17]. *Nodemon* é uma ferramenta do *Node.js*, que ajuda na base de desenvolvimento da aplicação. A descrição para esta ferramenta deve-se ao facto de que é um compilador em tempo real. Ou seja, quando na pasta raiz do projeto houver ficheiros de desenvolvimento alterados, *nodemon* automaticamente reinicia a aplicação com as alterações feitas. Em caso de erros na aplicação, *nodemon* informa

instantaneamente. Nodemon é uma excelente ajuda neste requisito, pois assim não há a necessidade de reiniciar o servidor a cada passo de desenvolvimento que se faça.

Visto agora o objeto chamado *scripts* que se encontra na Figura 25, iniciado na linha 5. Este objeto contém dois atributos nomeados de *start* e *start:dev*. Estes dois atributos do objeto *scripts* são muito importantes porque são eles que fazem iniciar a aplicação no *browser*.

Mas porquê que existem dois atributos no mesmo objeto com o mesmo propósito, capazes de executar a aplicação? A razão para a qual se têm dois atributos capazes de correr a aplicação deve-se ao facto de que o atributo chamado *start*, ao ser executado através do comando “*npm start*”, além de iniciar a aplicação, irá iniciar a aplicação num modo de produção, como se fosse o produto final. Como se pode ver o valor do atributo *start* é a rota do ficheiro binário, o cérebro da aplicação.

Por outro lado, a razão pela qual se tem o atributo *start:dev*, é que ao ser executado através do comando “*npm run start:dev*”, a aplicação será executada num ambiente de desenvolvimento, inacabado. E é executada nestas condições para que caso exista algum erro de aplicação, o *nodemon* informará sobre esse erro e será corrigido no ambiente de desenvolvimento.

Por fim, vê-se o objeto chamado *dependencies*, situado na Figura 25, linha 9. Este objeto contém as bibliotecas mais relevantes durante a implementação do projeto, e são: *bcrypt*, *Ejs* (*Embedded JavaScript*), *express*, *express-ejs-layouts*, *http-errors*, *pg-promise* e *Universally Unique Identifiers (UUID)*.

Para efetuar a instalação de qualquer biblioteca aqui presente foi executado na consola do *VsCode* o comando: “*npm install <package\_name>*”. Para obter esta informação, foi consultada a documentação oficial do *npm* (*Node Package Manager*) em [16]. *Node Package Manager* é uma biblioteca com mais de 800 000 *packages*. Portanto *npm* é o comando que instala, atualiza ou desinstala *packages* ou bibliotecas de uma aplicação, pertencentes ao *Node.js*.

Cada uma das bibliotecas citadas anteriormente têm a sua função e foram desenvolvidas para resolver determinado problema. Nos seguintes tópicos vão ser descritas

todas as bibliotecas, assim como serão descritas com exemplos reais, frutos da implementação.

## 5.2 Implementação do servidor local

No tópico anterior foi explicado como é que se instalaram as bibliotecas necessárias para o projeto, onde foi dado o nome de dependências. Também foram evidenciadas diferenças entre a aplicação estar em modo desenvolvimento, ou em modo produção. Tal como o comportamento da aplicação perante algum destes dois ambientes. Caso não se dê nenhuma indicação correta entre o comando que executa a aplicação no modo de desenvolvimento ou no modo produção, o sistema não executa no *browser*.

Porém, para executar a aplicação na *web* é necessário um servidor, neste caso um servidor *web* local, responsável pelo processamento das páginas do *site*, requisitadas pelos clientes através de *browsers*. Com a ajuda das tecnologias *ExpressJs* e *NodeJs*, foi fácil e minimalista criar o servidor local, capaz de processar todos os pedidos requeridos pelos utilizadores.

Atendendo às capacidades da tecnologia *ExpressJs* [18], vê-se na Figura 26 um excerto de código para a criação do servidor. A criação do servidor situa-se entre dois locais diferentes. Um situa-se no ficheiro nomeado de *app.js* e o outro localizado na pasta *bin*, o ficheiro *www*. O motivo que se levou a tomar esta decisão, de criar o servidor em dois locais, foi o facto de que no ficheiro *app.js* seria o mais indicado apenas para definir as rotas do projeto.

Ficheiro app.js	Ficheiro bin/www
<pre> 2  const express = require("express"); 22 const app = express(); </pre>	<pre> 7  var app = require('../app'); 9  var http = require('http'); 14 var port = 3000; 16 app.set('port', port); 22 var server = http.createServer(app); 28 server.listen(port); </pre>

Figura 26 - Implementação do servidor

Analizando ao detalhe a Figura 26, a coluna que diz respeito ao ficheiro *app.js*, primeiramente, na linha 2, é declarada a constante nomeada *express* que requer ter instalado o módulo *express*. De seguida é criada a constante *app* onde é chamada a função *express()*.

No ficheiro *bin/www*, linha 7, da Figura 26, é criada a variável *app* que recebe o caminho do ficheiro *app.js*. De seguida é criado o servidor do tipo *http*, onde requer ter instalado o módulo *http*. É criada também uma variável para a porta na qual o servidor terá de se ligar, à porta 3000. Na linha 16, através da função *set()* [19], que recebe como parâmetros um nome e um valor, a variável *app* terá assinado o valor da porta ao nome *port*, como primeiro parâmetro.

Ainda no mesmo passo, na linha 22, a função *createServer()* [20], é a responsável por transformar o computador num servidor do tipo *http*. E é onde se passa como parâmetro a variável *app* que contém o valor da porta do servidor. Por último é informar o servidor, *server*, através da função *listen()* [21], que está a escutar a porta 3000.

### 5.3 Rotas, verbos HTTP e JavaScript embebido

Com o servidor implementado e a funcionar corretamente, é possível criar diversas rotas que são necessárias para mapear todas as páginas do *site* no projeto Adota.me.

Uma vez que ainda não foram mostradas as constantes onde se encontram armazenados os caminhos das rotas, a Figura 27 mostra a criação dos caminhos para as rotas e a sua exportação no ficheiro *app.js*.



```

8  const indexRouter = require("./routes/index")
9  const userRouter = require("./routes/user")
10 const loginRouter = require("./routes/login")
11 const catalogRouter = require("./routes/catalog")
12 const animalRouter = require("./routes/animal")
13 const requestRouter = require("./routes/request")
14 const adoptRouter = require("./routes/adopt")
15 const donateRouter = require("./routes/donate")
16 const missingRouter = require("./routes/missing")
17 const notificationRouter = require("./routes/notification")
18 const permissionRouter = require("./routes/permission")
19 const profileRouter = require("./routes/profile")
20 const registryRouter = require("./routes/registry")

56 app.use("/", indexRouter)
57 app.use("/user", userRouter)
58 app.use("/login", loginRouter)
59 app.use("/catalog", catalogRouter)
60 app.use("/animal", animalRouter)
61 app.use("/request", requestRouter)
62 app.use("/adopt", adoptRouter)
63 app.use("/donate", donateRouter)
64 app.use("/missing", missingRouter)
65 app.use("/notification", notificationRouter)
66 app.use("/permission", permissionRouter)
67 app.use("/profile", profileRouter)
68 app.use("/registry", registryRouter)

87 module.exports = app

```

Figura 27 - Criação dos caminhos para as Rotas no *app.js*

Com mais detalhe, a imagem mais à esquerda da Figura 27 mostra a criação dos caminhos das rotas. Onde para cada rota está associada uma constante que armazenará o seu caminho em particular. A seguir, a imagem mais à direita da Figura 27, a constante *app* (ver declaração na Figura 26) através da função *use()* [22], receberá como primeiro parâmetro um caminho. É o caminho para o qual a função de *middleware* [23] está a ser chamada. Como segundo parâmetro, recebe a constante onde está a ser armazenada o caminho da rota. Por fim, a imagem ao fundo da Figura 27 mostra a instrução que comunica ao *NodeJs* quais são os bits de código que queremos exportar daquele ficheiro em específico.

Com tudo isto em mente, tem-se as bases para manipular e especificar a ação que cada rota irá influenciar no comportamento da aplicação.

Ainda neste tópico, será explicado como se implementa uma rota, o que é um verbo *http* [24] e a sua importância para a rota. No fundo, será analisada ainda com mais detalhe a *framework ExpressJs* e a biblioteca *Ejs* [25].

Uma rota é a resposta da aplicação ao cliente para determinado e particular propósito, na qual poderá ser um endereço *URL*. Cada rota é acompanhada por um dos verbos *http*. Verbos *http* são métodos que dependentemente da rota, executam operações do tipo: criar, ler, atualizar e eliminar. A tradução para estes quatro verbos será respetivamente: *Post*, *Get*, *Put* e *Delete*. Esta é a forma correta para se utilizarem os verbos *http* consoante a rota.

Porém existem mais verbos *http*, apenas são citados neste documento os verbos *http* utilizados durante a implementação do projeto.

Sabendo o que são verbos *http* e rotas, a Figura 28 apresenta uma das rotas implementadas do projeto. A rota *donate.js*. Com este pequeno pedaço de código, abordam-se matérias relacionadas com rotas aplicando a *framework ExpressJs*, verbos *http* e uma introdução à sintaxe *Ejs*.

```
1 var express = require('express');
2 var router = express.Router();
3
4 router.get('/', function(req, res, next) {
5   res.render('donate/index', {title: 'Doações', paragraph: 'Faça uma doação para ajudar os animais'});
6 });
7
8 module.exports = router;
```

Figura 28 - Rota *donate.js*

Analizando o pedaço de código da Figura 28, na linha 2 é declarada a variável *router* que recebe a criação de um objeto do tipo *router*, que servirá para lidar com as solicitações do cliente. Logo de seguida na linha 4, à rota é chamado o verbo *http*, que por sua vez é o método *get*. Este método recebe dois parâmetros: um caminho e uma função *middleware*. O caminho “/” representa “/*donate*”, (ver linha 63 da imagem à direita da Figura 27). E funções *middleware*, são funções que têm acesso ao objeto requerido (*req*) e conforme o objeto requerido, a função dá resposta a esse pedido (*res*).

Em suma, quando o cliente solicita no endereço do browser, a direção “/*donate*”, a rota da Figura 28 está preparada para ler, *get*, o endereço que o cliente inseriu e de seguida, através do segundo parâmetro *res* da *function*, este irá responder através do método *render()* [26], e apresentará a vista com endereço *donate/index*. A Figura 29 é a representação da realidade do que foi descrito no parágrafo anterior.

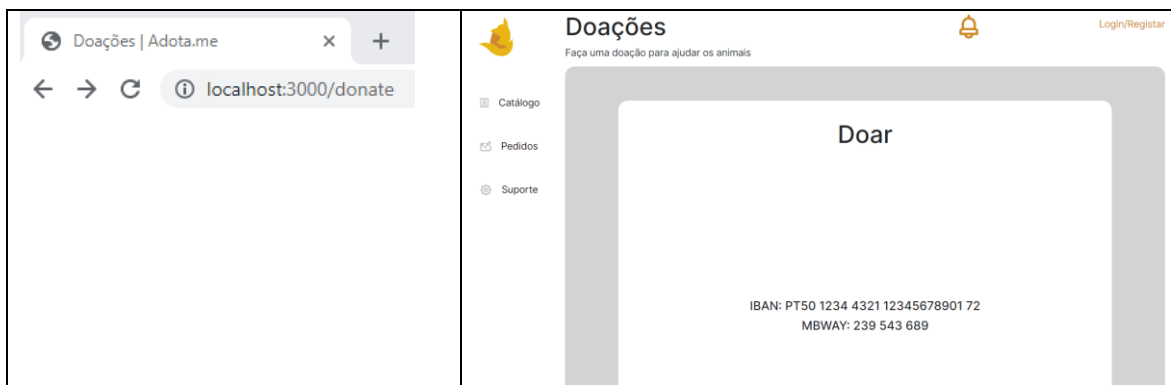


Figura 29 - Ilustração entre a rota *donate.js* e a sua respetiva vista

Como mostra a imagem à direita da Figura 29, é a página estática implementada com exemplo de dados bancários para fazer uma doação para uma instituição canil/gatil que tenha adotado o projeto Adota.me.

Ainda neste subtópico. Porquê que na linha 5 da Figura 28, no método *render()* é passado como segundo parâmetro um objeto com dois atributos, com valores do tipo *string*, nomeados de *title* e *paragraph*, respetivamente?

A resposta para esta pergunta, necessita-se da introdução ao conceito *Embedded JavaScript, Ejs*. *Ejs* é um dos *templates* mais populares que suporta a linguagem *JavaScript*. A resposta à questão escrita, é o simples facto de se poderem passar variáveis da secção das rotas para a secção das vistas. E é a vista *partials/header* que receberá os valores das variáveis *title* e *paragraph* através da sintaxe do *template Ejs*.

Uma breve nota: Nos *layouts* do projeto Adota.me, os conteúdos das suas páginas têm a estrutura de *header*, *aside* e *body*. Exceto a página principal, vista *home/index* que só tem conteúdo do tipo *header* e *body*. Portanto no desenvolvimento do projeto, dentro da pasta *views*, existe uma pasta chamada de *partials*, onde contém o desenvolvimento do conteúdo *header* e *aside*. O conteúdo *header* e *aside* é único e foi reaproveitado para todo o projeto consoante as páginas que assim o exigiam. Por outro lado, o conteúdo do tipo *body*, é desenvolvido em particular, dentro da respetiva vista.

Mais uma vez voltando à resposta da questão e para concluir, é a vista “*partials/header*” que recebe as variáveis passadas na rota *donate.js* e não a vista *donate/index*, pois nesta vista só é desenvolvido conteúdo do tipo “*body*”. Ou seja, na vista

*donate/index*, só foi programado o conteúdo que será “renderizado” dentro da *tag* “*body*”. Sendo assim não há a necessidade de criar um novo documento HTML só para a vista *donate/index*.

A Figura 30 mostra o código que produz o *header* do projeto Adota.me.

```
1 <header class="header">
2   <div class="d-flex">
3     <div class="d-flex justify-content-center login logo-width">
4       <a href="/"></a>
5     </div>
6     <div class="d-flex flex-column pt-5 text-width">
7       <h1><%= title %></h1>
8       <p><%= paragraph %></p>
9     </div>
10    <div class="d-flex justify-content-center bell-width pt-5">
11      <a href="/notification"></a>
12    </div>
13    <div class="d-flex justify-content-end text-login">
14      <a class="login a-prop cursor-style" data-bs-toggle="modal" data-bs-target="#loginModal">Login/Registrar</a>
15    </div>
16  </div>
17 </header>
```

Figura 30 - Código do conteúdo header

Concluindo, as linhas 7 e 8 da Figura 30 mostram respectivamente, uma *tag* `<h1>` que recebe o valor que vem da rota da variável *title* dentro da sintaxe fornecida pelo *Ejs* (`<%= %>`). E uma *tag* `<p>`, onde recebe o valor da variável *paragraph* proveniente da rota *donate.js*. Os valores recebidos pelas variáveis *title* e *paragraph*, serão as *strings* armazenadas na rota *donate.js*.

O resultado do código acima visto para a rota *donate.js* está ilustrado na Figura 31.



Figura 31 - Ilustração do header para a rota *donate.js*

Este é o exemplo real, implementado no projeto Adota.me, como mostra a Figura 31. E como se pode verificar, o valor atribuído às variáveis *title* e *paragraph* da Figura 28, linha 5, encontram-se realçadas na Figura 31.

## 5.4 Constituição das vistas

No tópico anterior foi analisada a situação como as rotas estão ligadas às vistas e como se passam informações das rotas para as vistas através da biblioteca instalada *Ejs*. Estudaram-se os verbos *http* e como estes têm importância nas rotas do projeto. Também ficou deixada

uma breve nota acerca de como está feita a estrutura do projeto Adota.me em relação às vistas.

No presente tópico, será mostrado com mais detalhe a estrutura das vistas e como elas estão organizadas no projeto consoante a apresentação das mesmas no *browser*. Será através deste assunto que se apresenta a biblioteca instalada no projeto, nomeada de *express-ejs-layouts* [27].

Como foi referido na nota escrita no tópico anterior, no projeto existem dois tipos diferentes de *layouts*. A Figura 32 é um exemplo real e apresenta a diferença entre os dois tipos diferentes de *layouts*.

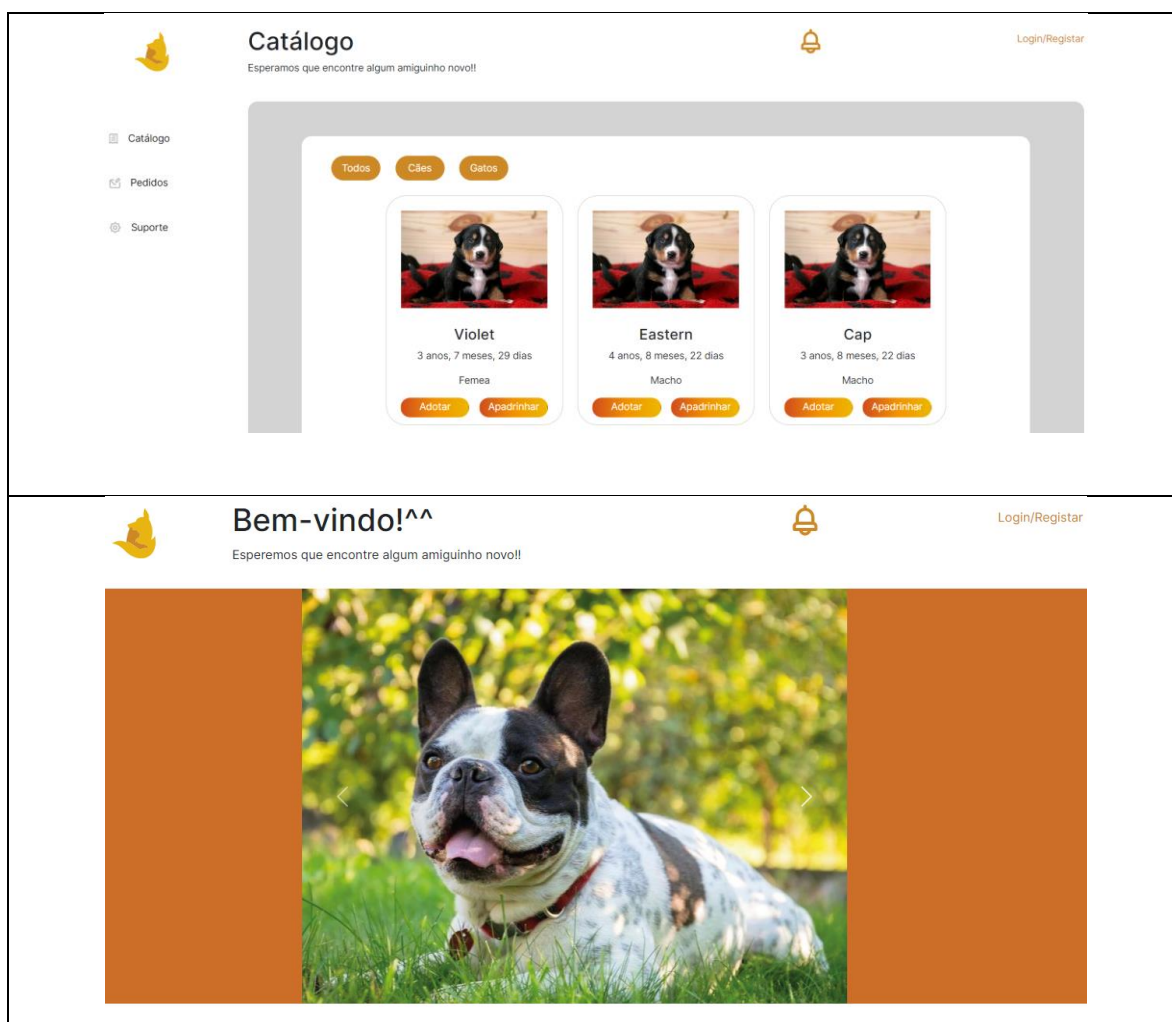


Figura 32 - Layout catálogo e página principal

A diferença entre as duas imagens da Figura 32, é que a imagem abaixo não tem conteúdo do tipo “*aside*”, ou seja, não contém os botões “Catálogo”, “Pedidos” e “Suporte” em relação à imagem de cima. Quer dizer que o projeto encontra-se preparado para receber *layouts* com conteúdo que preencha o cabeçalho, lateral e corpo da página ou somente cabeçalho e corpo da página.

É relevante salientar que as imagens presentes no catálogo do sistema, e como mostra a Figura 32, foram tiradas através das aplicações: *The cat API* [28] e *The dog API* [29]. De maneira a preencher o campo “fotografia” dos animais na base de dados, utilizaram-se estas duas *API*’s, que de facto preenchem o campo “fotografia” dos animais com imagens aleatórias de cães e gatos.

As pastas *layouts* e *partials* que se encontram na Figura 33, são pastas que no projeto estão dentro da pasta das vistas.



Figura 33 - Pasta *layouts* e *partials*

Na imagem à esquerda da Figura 33, o ficheiro *default.ejs* representa a estrutura de qualquer página web que disponha de conteúdo do tipo cabeçalho, lateral e corpo da página. Por outro lado, o ficheiro *noaside.ejs* apenas dispõe de conteúdo do tipo cabeçalho e corpo da página.

Ainda na Figura 33, na imagem mais à direita, encontram-se os ficheiros que dizem respeito ao conteúdo particular desenvolvido sobre o cabeçalho e conteúdo lateral, *header.ejs* e *side-bar.ejs*, respetivamente. O ficheiro *header.ejs* é onde está implementado o cabeçalho do sistema Adota.me. O exemplo implementado do *header.ejs* pode-se ver pela Figura 31. O ficheiro *side-bar.ejs* é onde está implementado o conteúdo lateral do sistema Adota.me. O exemplo implementado do *side-bar.ejs* são, simplesmente, os botões “Catálogo”, “Pedidos” e “Suporte”, onde se podem ver pela Figura 32 na imagem acima. O ficheiro *scripts.ejs* apenas contém *tags script* que fazem ligação com a *framework Bootstrap* [12].

Abrindo o ficheiro *default.ejs* presente na pasta *layouts*, existe a estrutura de um documento HTML, onde a Figura 34 apenas mostra o conteúdo dentro da *tag body* do documento HTML do ficheiro *default.ejs*.

```
23 <body>
24   <div class="d-flex flex-column">
25     <%- include('../partials/header.ejs') %>
26     <div class="d-flex flex-row">
27       <%- include('../partials/side-bar.ejs') %>
28       <div class="d-flex flex-column container-app rounded-4 h-100">
29         <div class="bg-white d-flex flex-grow-1 rounded-4 flex-column p-4">
30           <%- body %>
31         </div>
32       </div>
33     </div>
34   </div>
35   <%- include('../partials/scripts.ejs') %>
36   <%- include('../modals/login.ejs') %>
37   <%- include('../modals/registry.ejs') %>
38   <%- include('../modals/modal-thank.ejs') %>
39   <%- include('../modals/decision.ejs') %>
40 </body>
```

Figura 34 – Vista *default.ejs*

Na Figura 34 a realçado, encontra-se a explicação para o motivo na qual o ficheiro *layout.ejs* é responsável apenas por apresentar vistas com conteúdo do tipo cabeçalho, lateral e corpo. Porque nas linhas 25 e 27 através da biblioteca *express-ejs-layouts*, são incluídos através da função *include()* os caminhos dos ficheiros que se pretendem incluir no ficheiro *default.ejs* para que possam ser apresentados. Na linha 30 a realçado, será executado o corpo da vista em particular.

Por outro lado, abrindo o ficheiro *noaside.ejs* presente na Figura 35, é representado um novo documento *HTML*. Nesta figura apenas está a ser mostrado a *tag body* do documento *HTML*.

```
23 <body>
24   <div class="d-flex flex-column">
25     <%- include('../partials/header.ejs') %>
26     <%- body %>
27   </div>
28   <%- include('../partials/scripts.ejs') %>
29   <%- include('../modals/login.ejs') %>
30   <%- include('../modals/registry.ejs') %>
31   <%- include('../modals/modal-thank.ejs') %>
32   <%- include('../modals/decision.ejs') %>
33 </body>
```

Figura 35 - Vista *noaside.ejs*

O novo documento *HTML* tem a missão de apresentar páginas *web* apenas com o propósito de servir conteúdo do tipo cabeçalho e corpo da página. Para isso, a Figura 35 mostra o código que executa o que foi descrito.

A Figura 36 mostra no ficheiro *app.js*, a inicialização da biblioteca *express-ejs-layouts*, assim como a programação para os caminhos dos ficheiros que se encontram na pasta *layouts*.

```
6   const expresslayouts = require("express-ejs-layouts")

37  app.use(expresslayouts)
38  app.set("views", path.join(__dirname, "views"))
39  app.set("layout", "layouts/default")
40  app.set("layoutNoAside", "layouts/noaside")
```

Figura 36 - Definição da biblioteca *express-ejs-layouts* no ficheiro *app.js*

Com a explicação citada neste tópico, neste momento, é possível perceber-se a estrutura e o que está a ser executado por detrás das imagens da Figura 32.

## 5.5 Base de dados

Nos tópicos anteriores foi explicado através de ilustrações e texto, como o fluxo de dados é transmitido pelo projeto Adota.me, apenas com recurso a memória alocada. Memória do computador reservada para quando a aplicação é executada, necessita de ir ao encontro de variáveis declaradas para resolver determinada ação. Por exemplo, as rotas do projeto encontram-se em memória alocada, como mostra a Figura 27.

Neste tópico, é descrito como funciona a base de dados construída e apropriada para a solução deste projeto. Também é explicada a *API* utilizada no projeto, que serviu para estabelecer a ligação entre operações *backend* e a base de dados propriamente dita do projeto Adota.me. Será também abordado o assunto sobre a utilização do método *UUID* e da biblioteca *Bcrypt*.



### 5.5.1 Criação da base de dados e utilização do método UUID

Uma das primeiras etapas na implementação do projeto foi construir uma base de dados adequada à situação do problema, na qual o projeto Adota.me visa responder. Feito o estudo e a análise de um conjunto de classes, como mostra a Figura 10, que propõe suporte onde será possível armazenar todos os dados que serão utilizados consoante a utilização do sistema. Por exemplo, o projeto deve guardar as informações de um utilizador que se regista no sistema. A classe *User* da Figura 10 tem o propósito para guardar os dados de um determinado utilizador que se registou. Outro exemplo, e relacionado com o exemplo anterior, é o facto de que quando algum utilizador se regista no sistema, automaticamente estará a fornecer dados para fazer *login* no sistema. E como se pode ver através da Figura 10, a tabela *User* e *Login* têm um relacionamento de um para um. Ou seja, um *user* tem acesso a um *login*.

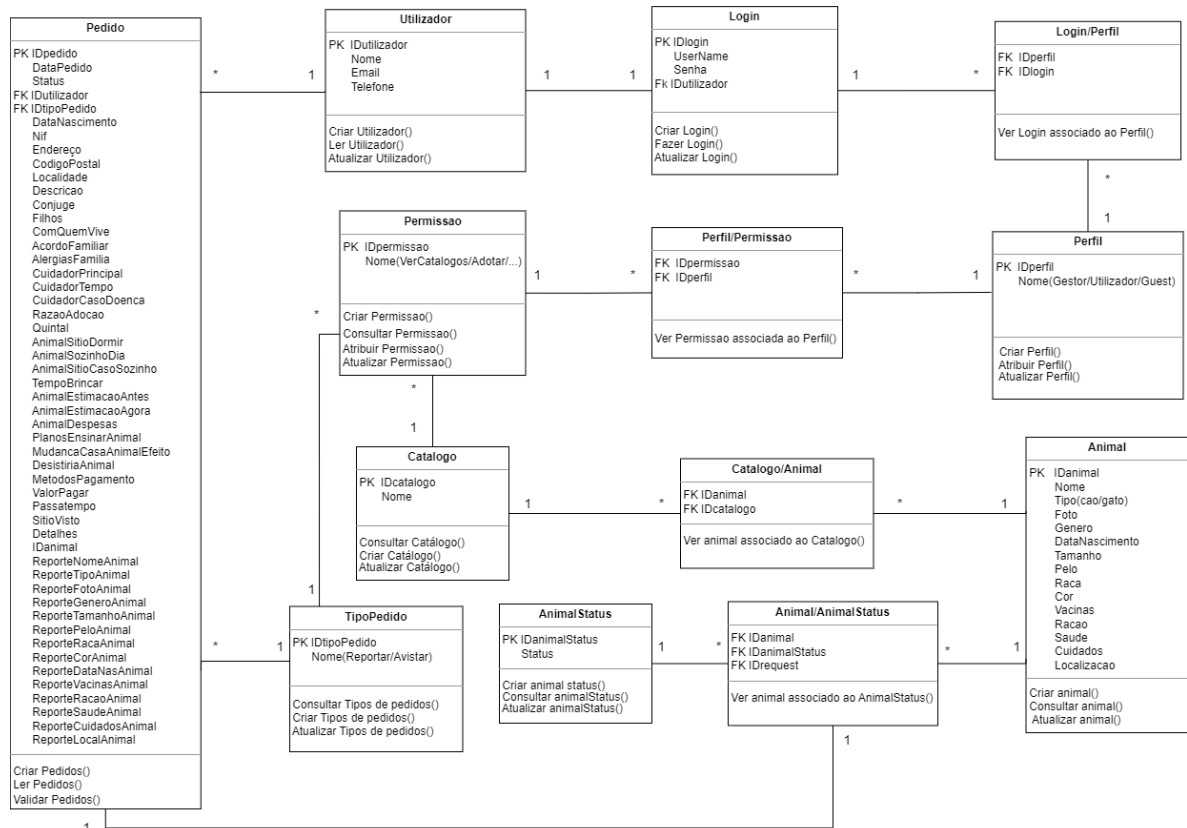


Figura 10 - Diagrama de Classes

Em termos práticos, a Figura 37 mostra um excerto do ficheiro *createTables.sql* que se encontra por completo nos Anexos A 1. Criação das tabelas do ficheiro *createTables.sql*. A

seguinte figura mostra a criação do *schema* chamado “adotame” e as criações das tabelas *user* e *login*.

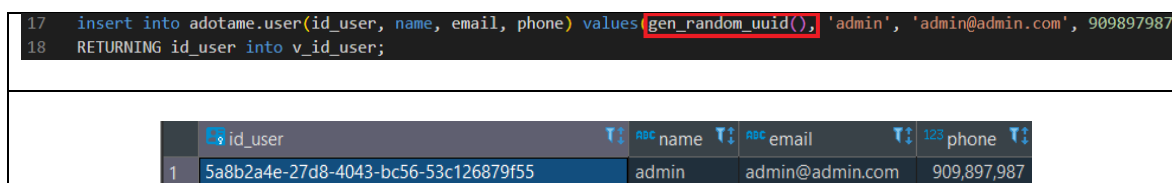
```
1  create schema if not exists adotame;
2
3  create table if not exists adotame.user(
4      id_user uuid primary key,
5      name varchar(250) not null,
6      email varchar(100) unique not null,
7      phone bigint null
8  );
9
10 create table if not exists adotame.login(
11     id_login uuid primary key,
12     username varchar(250) not null,
13     password varchar(100) not null,
14     id_user uuid references adotame.user (id_user)
15 );
```

Figura 37 - Criação do *schema*, tabela *user* e *login*

Como já referido, no parágrafo anterior, a tabela *user* e *login* possuem uma relação. Se existe alguma relação entre duas tabelas da base de dados, logo uma dessas duas tabelas terá de ter um atributo que faça a correspondência da ligação entre as duas tabelas. Esse atributo designa-se por chave estrangeira. No exemplo prático, como o *login* depende de um *user* criado, a tabela *login* vai ter como chave estrangeira, a chave primária da tabela *user*. Na linha 14 da Figura 37, é onde se encontra a linha de código que faz a ligação entre as duas tabelas.

Como mostra a Figura 37, por exemplo nas linhas 4, 11 e 14 da mesma figura, encontra-se um tipo de dados do tipo *UUID* [30]. Que tipo de dados é este? *UUID* é um tipo de identificador único. O que faz? Este identificador é gerado através da função *gen\_random\_uuid()* [31], função que foi utilizada no ficheiro *seedData.sql*. Serve para fins de gerar *id*'s aleatórios para popular as chaves primárias e estrangeiras das tabelas. Recorreu-se a este tipo de dados, *UUID*, porque é um tipo de dados mais seguro para proteger os *id*'s na base de dados. Em vez de se preencherem os *id*'s da base de dados com uma gama de valores em sequência, utilizou-se o *UUID* porque gera de imediato um *id* com uma sequência de 32 dígitos.

A Figura 38 mostra a aplicação da função `gen_random_uuid()` fornecida pelo *PostgreSQL* no ficheiro *seedData.sql*. De seguida, na imagem abaixo vê-se o resultado da mesma função.



```
17 insert into adotame.user(id_user, name, email, phone) values(gen_random_uuid(), 'admin', 'admin@admin.com', 909897987)
18 RETURNING id_user into v_id_user;
```

	id_user	name	email	phone
1	5a8b2a4e-27d8-4043-bc56-53c126879f55	admin	admin@admin.com	909,897,987

Figura 38 - Aplicação da função `gen_random_uuid()`

O resultado visto na coluna `id_user` da imagem do fundo da Figura 38, representa um identificador gerado do tipo *UUID*. É uma sequência de 32 dígitos em hexadecimal separados por 5 grupos.

### 5.5.2 Utilização da API *pg-promise* e da biblioteca *Bcrypt*

No subtópico anterior foi mostrado um exemplo como foram criadas duas das tabelas da base de dados do projeto, assim como foi mostrado como estas tabelas estão ligadas e como foram inseridos dados nas tabelas.

No presente subtópico será mostrado como foi feita a ligação entre o *schema* *adotame* e o *backend* do sistema. Também será demonstrado a aplicação da biblioteca *Bcrypt* [32] fornecida pelo *NodeJs*.

Primeiro de tudo, a *API pg-promise* [33] foi disponibilizada pelo *NodeJs*. Esta *API* foi utilizada para se ligar à base de dados implementada e a partir daí, usou-se de modo para executar *queries* e transações no *backend* da aplicação. Como já foi referido neste documento, *pg-promise* é uma dependência do projeto *Adota.me*, como foi referido no tópico Instalações das bibliotecas e dependências.

Para fazer a ligação entre a base de dados ao backend do projeto, foi criado um ficheiro onde se encontra a respetiva ligação, como mostra a Figura 39.

```

1  const pgp = require("pg-promise")();
2  const connectionString = "postgresql://postgres:*****@localhost:5432/postgres";
3
4  const db = pgp(connectionString);
5
6  module.exports = db;

```

Figura 39 - Conexão à base de dados

A Figura 39 mostra a ligação à *API pg-promise*, bem como os dados que constroem o caminho dessa ligação. Dados relativos ao nome da base de dados onde se encontra o *schema*, palavra-chave, servidor e porta.

Feita esta ligação, podem-se construir *queries* no backend da aplicação, a partir dos dados que se encontram inseridos nas tabelas gerados pelo ficheiro *seedData.sql*.

Um exemplo prático onde é ilustrada uma transação através da *API pg-promise* é a Figura 40. Uma transação em base de dados é o resultado lógico entre uma ou várias operações realizadas na própria base de dados. Através da Figura 40 também será explicada a biblioteca *Bcrypt* responsável para guardar as palavras-chaves na base de dados de modo encriptadas.

```

14  router.post("/", function (req, res, next) {
15    console.log("Register api ", req.body)
16    var frontName = req.body.name
17    var frontEmail = req.body.email
18    var frontEmail1 = req.body.email1
19    var frontPassword = req.body.password
20    var frontPassword1 = req.body.password1
21    var frontPhone = req.body.phone
22
23    if (frontEmail !== frontEmail1) {
24      res.status(400).json({ message: "E-mail don't match" })
25    } else if (frontPassword !== frontPassword1) {
26      res.status(400).json({ message: "Password don't match" })
27    } else {
28      const saltRounds = 10
29      bcrypt.hash(frontPassword, saltRounds, function (err, hash) {
30        if (err) {
31          res.status(500).json({ message: "failed encrypt pass" })
32        } else {
33          db.tx(async (t) => {
34            var user = await t.one(
35              `insert into adotame.user(id_user, name, email, phone) values($1, $2, $3, $4) returning id_user`,
36              [crypto.randomUUID(), frontName, frontEmail, frontPhone]
37            )
38
39            await t.none(
40              `insert into adotame.login(id_login, username, password, id_user) values($1, $2, $3, $4)`,
41              [crypto.randomUUID(), frontEmail, hash, user.id_user]
42            )
43          })

```

Figura 40 - Excerto do ficheiro *registry.js*

Na Figura 40 observa-se a rota *registry.js* cuja missão será validar e inserir dados acerca de um utilizador que se registe no sistema Adota.me. Entre as linhas 16 e 21 vê-se um conjunto de variáveis declaradas que recebem os dados provenientes do *frontend* quando o utilizador preenche o formulário registo.

O formulário “registro” implementado é apresentado em *modal* como mostra a Figura 41.

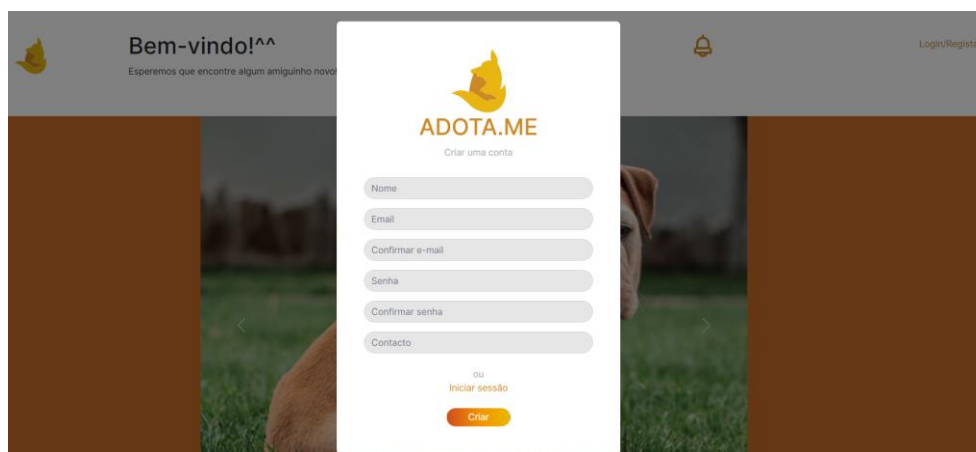
The image shows a registration modal for 'ADOTA.ME'. At the top, it says 'Bem-vindo!^^' and 'Esperemos que encontre algum amiguinho novo'. The modal has a white background with a yellow dog icon and the text 'ADOTA.ME' and 'Criar uma conta'. Below this are input fields for 'Nome', 'Email', 'Confirmar e-mail', 'Senha', 'Confirmar senha', and 'Contacto'. There is a link 'ou Iniciar sessão' and a yellow 'Criar' button at the bottom. The background of the modal is a blurred image of a dog's head.

Figura 41 - Modal registro

Como existem dois campos do tipo Email e Senha no formulário da *modal*, como mostra a Figura 41, há a necessidade de se validar os dois *inputs* dados pelo utilizador, quer nos campos do tipo email, quer nos campos do tipo senha. A esta validação deve-se dizer: o campo “Email” preenchido terá de ser igual ao campo “Confirmar e-mail”, e o campo “Senha” preenchido terá de ser igual ao campo “Confirmar senha”.

Para isso, as linhas 23 e 25 da Figura 40, têm o propósito de efetuar a validação dos campos Email e Senha. Caso o utilizador se engane no preenchimento destes campos, aparecerá uma mensagem apelativa ao respetivo erro que o utilizador cometeu. Por outro lado, se tudo bem, é recorrida à função *hash()* da biblioteca *Bcrypt* para encriptar a senha que o utilizador colocou no campo Senha do formulário. Esta implementação encontra-se na linha 29 da Figura 40.

De seguida são inseridos na base de dados, os dados que o utilizador colocou no formulário pertencente à *modal*. Em termos de implementação para inserir os dados do utilizador na base de dados, foi através de uma função assíncrona onde foi realizada uma transação com duas operações do tipo *insert*. Esta transação situa-se entre as linhas 33 e 43 da Figura 40.

Porquê a necessidade de efetuar uma transação com duas operações do tipo *insert* para os dados de um único formulário?

Como se pode ver na Figura 40, linhas 35 e 40, através da sintaxe *SQL* faz-se uma operação do tipo *insert* à tabela *user* e *login*, respetivamente. Como já foi citado no subtópico anterior, quando se cria um utilizador, automaticamente cria-se o seu *login* para aceder ao sistema.

## 5.6 Interfaces da aplicação web

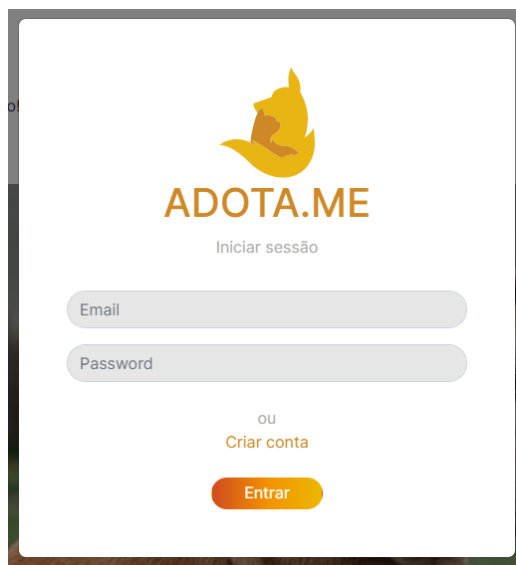
Antes de iniciar o desenvolvimento da aplicação *web*, foram realizadas reuniões com os designers da empresa, de modo a poderem analisar o sistema que se pretendia implementar, e para poderem começar a desenhar os diversos layouts da aplicação Adota.me. Como referido no subtópico 4.12, foi dito que, os *layouts* para aplicação Adota.me, desenhados pelos *designers* da empresa ProjectBox, foram partilhados pessoalmente através do *software* Figma.

As interfaces mostradas no presente subtópico, são imagens reais e capturadas resultantes da implementação do sistema Adota.me. As interfaces que foram partilhadas pessoalmente pelo Figma e que vão ser citadas neste subtópico encontram-se nos Anexos, entre as páginas 89 e 92.

Com o plano do projeto bem estruturado, design pronto, base de dados construída e *populada*, foi possível passar à próxima etapa do projeto: produzir as interfaces do sistema.

### 5.6.1 Interface Login/Registo

Todos os utilizadores não registados poderão visualizar o catálogo de animais disponíveis para adoção/apadrinhamento, porém não poderão adotar ou apadrinhar um animal se não estiverem registados no sistema. No entanto “devem dirigir-se” à interface de registo ilustrada na Figura 41.



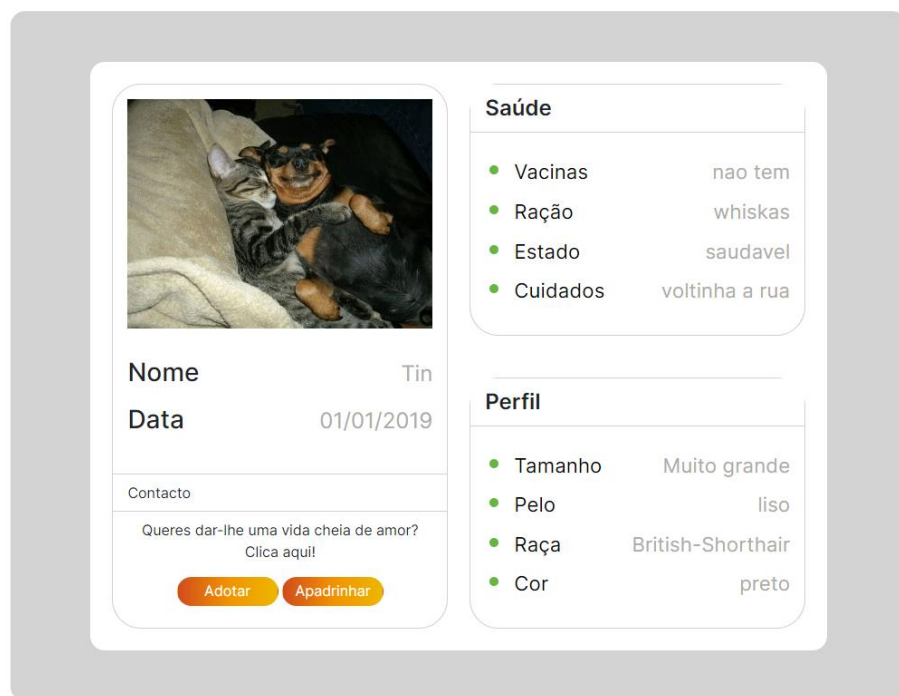
*Figura 42 - Interface de Login*

A Figura 42, depois do utilizador registar-se na aplicação, poderá fazer *login* no sistema. O registo e o *login* são realizados na mesma interface e para ambas é feita a validação de dados.

No anexo A 2. Modal Login partilhada pelo Figma, encontra-se a interface pretendia na implementação.

### 5.6.2 Interface para ver características dos animais

A interface para consultar as características dos animais tem como principal objetivo, dar a saber ao utilizador mais detalhes sobre o animal selecionado. Nesta interface é possível ler sobre os seguintes aspetos dos animais: nome, data de nascimento, vacinas, ração, estado de saúde, cuidados a ter com o animal, tamanho, pelo, raça e cor, tal como mostra a Figura 43.



*Figura 43 - Interface características do animal*

A Figura 43, é possível ser visualizada quando o utilizador se encontra no catálogo do sistema e clica na foto do animal no *card*. Este é o único caminho que o utilizador pode fazer para acessar a este conteúdo do sistema.

No anexo A 3. Interface características do animal partilhada pelo Figma, encontra-se a interface pretendia na implementação.

### 5.6.3 Interface para adotar animal

Sendo a adoção de um animal um dos pontos focais deste projeto, deu-se bastante atenção e ênfase para que o formulário do pedido do tipo adotar ficasse o menos extenso possível, mas que também cumprisse com as questões necessárias para que um gestor validasse pela positiva o pedido de adoção.

O utilizador da plataforma preenche os vários dados do formulário. Todos os dados do formulário são de preenchimento obrigatório pois pretende-se analisar o pedido com máximo detalhe, sendo possível avaliar o utilizador, se este tem condições favoráveis para adotar o animal escolhido.



Responde a este inquérito para avaliarmos se tem o que é preciso para adotar um novo companheiro.

Responde a este inquérito para avaliarmos se tem o que é preciso para adotar um novo companheiro.

☐ Sim ☐ Não

☐ Sim ☐ Não

☐ Sim ☐ Não

☐ Sim ☐ Não



Violet  
25/01/2019

☐ Sim ☐ Não

☐ Sim ☐ Não

☐ Sim ☐ Não

☐ Sim ☐ Não

☐ Sim ☐ Não

☐ Sim ☐ Não

☐ Sim ☐ Não

Enviar

75

Como mostra o formulário da Figura 44, nota-se que se pretende analisar o “quão forte será a amizade entre o adotante e o animal”. Ou seja, o papel do gestor será muito importante na validação do pedido. Porque a partir dos dados submetidos no formulário pelo utilizador, o gestor terá de realizar uma análise completa ao pedido colocado pelo utilizador, não podendo encontrar faltas de coerência na análise do formulário, ou notar que o utilizador não tem condições suficientes para que possa adotar o animal.

No anexoA 4. Interface para adotar animal partilhada pelo Figma, encontra-se a interface pretendia na implementação.

É de salientar que na realização deste projeto, as questões presentes no formulário da Figura 44 foram construídas com base nas questões para adoção do *site* MIDAS [7].

#### 5.6.4 Interface consultar e validar pedidos

Conforme os pedidos são feitos pelos utilizadores, serão de imediato enviados para a interface do gestor. Quem faz a validação dos pedidos é o gestor do *site*. Deste modo pensou-se numa interface onde o gestor poderia consultar os vários tipos de pedidos criados pelos utilizadores e consultar os dados dos formulários preenchidos também pelos utilizadores.

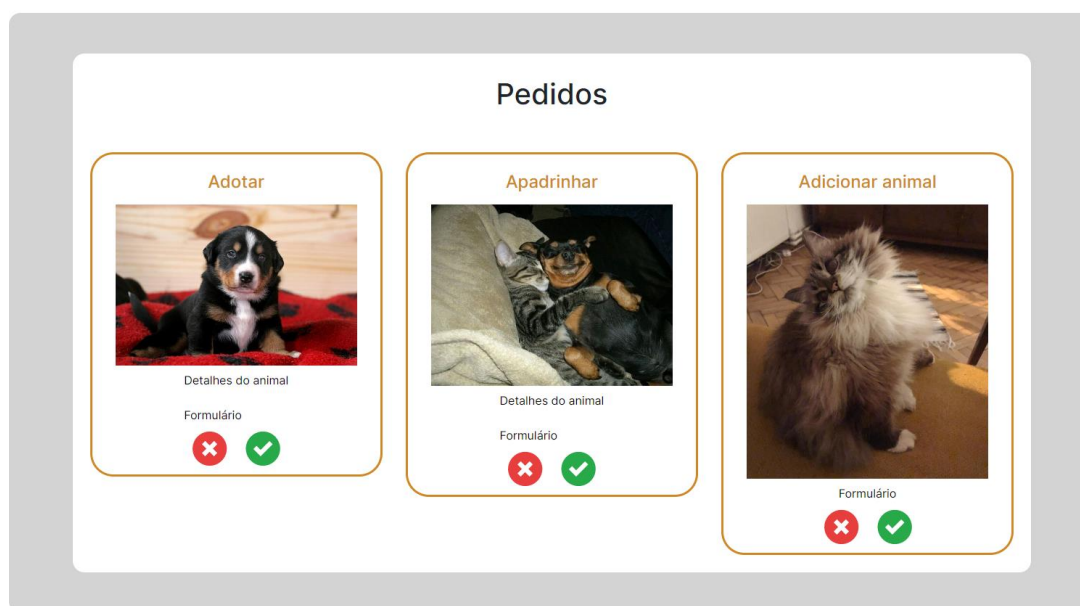


Figura 45 - Interface consultar e validar pedidos

Como mostra a Figura 45, esta é a interface implementada para validar pedidos. Como se pode ver, os pedidos que estão criados são do tipo: adotar, apadrinhar e adicionar animal.

No anexo A 5. Interface consultar e validar pedidos partilhada pelo Figma, encontra-se a interface pretendia na implementação.

## 6 Testes

Os testes são uma componente bastante importante em ter em prática, quer no desenvolvimento do projeto, quer no *software* final produzido. Uma vez que permitem verificar aspetos do sistema tais como funcionalidades e aspetos relacionados com a interação do sistema com o utilizador. Sendo assim, ao longo do desenvolvimento do sistema foram realizados vários testes para verificar se as funcionalidades e a interação com o utilizador não garantiam erros de desenvolvimento. E se se ocorresse algum erro, o passo de seguida a realizar seria resolver o erro em causa.

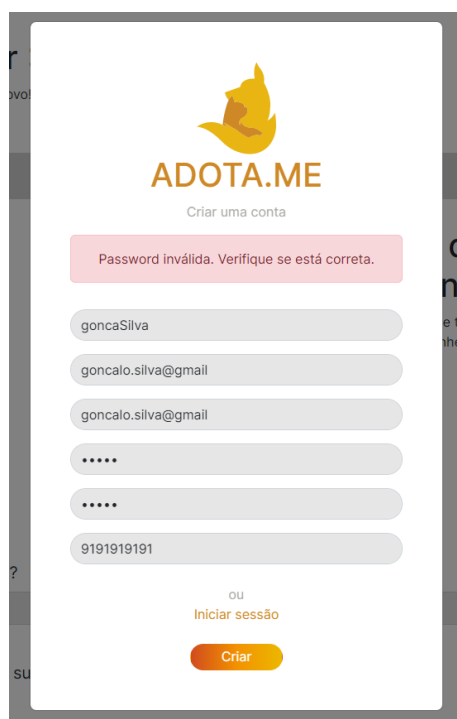
The image shows a mobile application interface for 'ADOTA.ME'. At the top, there is a logo of a yellow cat and the text 'ADOTA.ME'. Below the logo, it says 'Criar uma conta'. A pink error message box displays 'Password inválida. Verifique se está correta.'. There are several input fields: a name field with 'goncaSilva', an email field with 'goncalo.silva@gmail', another email field with 'goncalo.silva@gmail', two password fields (both masked with dots), and a phone number field with '9191919191'. At the bottom, there is a link 'ou Iniciar sessão' and a yellow 'Criar' button.

Figura 46 - Caso de teste para criar utilizador

Um caso para o teste de erros ou para testar a validação feita por parte do sistema, foi escolhida a interface onde se regista o utilizador. Como se pode ver através da Figura 46, foi

feito um teste onde se coloca propositalmente a *password* incorreta. E o resultado, é que o sistema informa o utilizador que se enganou a escrever a *password*. A informação sobre o erro do utilizador, encontra-se no *banner*, representado a vermelho como mostra a figura acima.

Outro caso de teste que se considerou relevante, é sobre a validação dos dados para o formulário do pedido “adicionar animal”. Nomeadamente para os campos do formulário: “nome do animal” e “data de nascimento do animal”. A Figura 47 mostra o excerto de código que realiza a validação citada anteriormente.

```
21 router.post("/create", function (req, res, next) {
22   var name = req.body.name;
23
24   var get_birth_date = new Date(req.body.birth_date);
25   var getSysDate = new Date();
26
27   if (name.length > 15 || name.length < 2) {
28     res.send("Nome para este animal é inválido!");
29   } else if (
30     get_birth_date.toLocaleString("pt-PT", {
31       year: "numeric",
32       month: "numeric",
33       day: "numeric",
34     }) >
35     getSysDate.toLocaleString("pt-PT", {
36       year: "numeric",
37       month: "numeric",
38       day: "numeric",
39     })
40   ) {
41     res.send("Data inserida não pode ser maior do que a data atual!");
```

Figura 47 - Validação dos campos para pedido "Adicionar animal"

Pela Figura 47, linha 21, vê-se implementada a rota para a criação do pedido “adicionar animal”. Como referido anteriormente, a missão é validar o campo “nome do animal” e “data de nascimento do animal”. Esta validação é feita perante a rota da Figura 47.

Primeiramente para validar o “nome do animal”, impôs-se que o “nome do animal” não pode ser constituído por mais de 15 letras e por menos de 2 letras. Caso o utilizador dê o nome ao animal perante esta constituição, o utilizador receberá um aviso a dizer “Nome para o animal é inválido!”. Em código, esta condição está representada nas linhas 27 e 28 da Figura 47.

A seguinte validação a ser feita, é validar a “data de nascimento do animal”. Ou seja, quando o utilizador insere a data de nascimento do animal, esta data não poderá ser superior à data do sistema. Para concretizar esta validação, primeiramente transforma-se num objeto

do tipo *Date*, a data de nascimento do animal inserida pelo utilizador, guardada na variável *get\_birth\_date*, como mostra a linha 24 da Figura 47. Na linha 25 da Figura 47, encontra-se a variável *getSysDate*, responsável por armazenar a data do sistema.

Com as variáveis necessárias para realizar a validação da “data de nascimento do animal”, entre as linhas 29 e 41 da Figura 47, verifica-se que, se a data de nascimento do animal for superior à data do sistema, o sistema informa o utilizador com a seguinte mensagem: “Data inserida não pode ser superior do que a data atual!”.

Outro caso de teste escolhido, é verificar que, quando é criado o pedido do tipo adotar, apadrinhar ou adicionar animal, feito pelo utilizador, o pedido será enviado para a interface onde o gestor consulta e valida pedidos. Escolhido, por exemplo, o pedido do tipo apadrinhar animal, primeiramente verificam-se se os dados preenchidos pelo utilizador no formulário do pedido apadrinhar animal, são inseridos na base de dados. Propriamente na tabela dos pedidos, como mostra a Figura 48.

```
119 db.tx(async (t) => {
120   var request = await t.one(
121     `insert into adotame.request(id_request, date_request, status, id_user, id_request_type, birth_date, nif,
122     address, postal_code, locality, phone, financial_payment_method, value_amount, hobby, id_animal)
123     values($1, now(), 'Pendente', '79734220-0e4c-4d00-ba84-d0c8c2f7f8d6', 'c1cdc8a4-5124-4398-9b96-1e5615a81de3',
124     $2, $3, $4, $5, $6, $7, $8, $9, $10, $11)
125     returning id_request`,
126     [
127       uuidv4(),
128       req.body.birth_date,
129       req.body.nif,
130       req.body.address,
131       req.body.postal_code,
132       req.body.locality,
133       req.body.phone,
134       req.body.financial_payment_method,
135       req.body.value_amount,
136       req.body.hobby,
137       req.body.animal_ID,
138     ]
139   );
```

Figura 48 - Insert para o pedido apadrinhar animal

Através da Figura 48, na linha 121, consta-se que de facto existe uma operação do tipo *insert* na tabela *request*, pedido. Por outro lado, deve-se verificar se na base de dados se o pedido ficou de facto registado. Ver Figura 49.

	id_request	date_request	status	id_user	id_request_type
1	fe1bd4e6-5f99-4e2d-83a5-40abc5ea000a	2022-11-12	Reprovado	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	080f4ac5-5e1c-4fba-8044-5a45331d7826
2	e47cb928-9e5f-497b-a6c7-d15bfb58707c	2022-11-12	Reprovado	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	c1cdc8a4-5124-4398-9b96-1e5615a81de3
3	12998f42-71d9-46e5-a97f-14f7ab4aaca5	2022-11-12	Reprovado	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	2f958f74-0e11-44f6-a343-05df75e5afd6
4	b9ffe6ce-c614-4ec9-9b35-000aec0a6792	2022-12-15	Pendente	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	c1cdc8a4-5124-4398-9b96-1e5615a81de3

Figura 49 - Estado dos pedidos

Verifica-se pela Figura 49 que o pedido criado do tipo apadrinhar animal encontra-se registrado na base de dados com o campo *status* pendente. Isto quer dizer que o pedido ainda não foi validado pelo gestor do sistema.

Para concluir o processo de teste em questão, tem-se de verificar que o pedido criado com o *status* pendente, aparecerá na interface do gestor. Para isso criou-se a *query* que filtra os pedidos através do campo *status*, como mostra a Figura 50.

```

22 router.get("/list", function (req, res, next) {
23   var status = req.query.status || "Pendente";
24   db.any(
25     `select rt.id_request_type, rt.request_name, r.report_animal_photo, a.photo, a.id_animal, r.id_request
26     from adotame.request r
27     inner join adotame.request_type rt
28     on r.id_request_type = rt.id_request_type
29     left join adotame.animal a
30     on r.id_animal = a.id_animal
31     where r.status = ${status},
32     [status]
33   )

```

Figura 50 - Query que filtra os pedidos pelo status

A interface do gestor onde consulta e valida os pedidos, é fruto gerado pelo código da Figura 50. A *query* proposta faz uma operação do tipo *select* a vários campos da tabela pedido, tipo de pedido e animal, onde o campo *status* da tabela pedido tem de ter o valor “Pendente”. Desta forma, o gestor só visualizará na sua interface pedidos que ainda não tenham sido analisados e validados.

O próximo caso de teste será verificar que, quando os pedidos sejam aceites ou recusados pelo gestor, os pedidos nestas condições fiquem invisíveis na mesma interface. Como foi dito no caso de teste anterior, apenas aparecerão pedidos com o *status* pendente na interface do gestor. Porém, quando o gestor aceita o pedido, o *status* do pedido na base de dados passará para o *status* “Aprovado”. E caso o pedido seja reprovado, o *status* na base de dados passará para “Reprovado”. Ou seja, os *status* dos pedidos variam entre os *status* “Pendente”, “Aprovado” e “Reprovado”, como mostra a Figura 51.

	id_request	date_request	status	id_user	id_request_type
1	fe1bd4e6-5f99-4e2d-83a5-40abc5ea000a	2022-11-12	Reprovado	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	080f4ac5-5e1c-4fba-8044-5a45331d7826
2	e47cb928-9e5f-497b-a6c7-d15bfb58707c	2022-11-12	Reprovado	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	c1cdc8a4-5124-4398-9b96-1e5615a81de3
3	12998f42-71d9-46e5-a97f-14f7ab4aaca5	2022-11-12	Reprovado	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	2f958f74-0e11-44f6-a343-05df75e5afd6
4	dd5b34f4-c142-4194-91b6-ebd5368ee032	2022-12-15	Pendente	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	080f4ac5-5e1c-4fba-8044-5a45331d7826
5	b9ffe6ce-c614-4ec9-9b35-000aec0a6792	2022-12-15	Aprovado	79734220-0e4c-4d00-ba84-d0c8c2f7f8d6	c1cdc8a4-5124-4398-9b96-1e5615a81de3

Figura 51 - Status dos pedidos

A Figura 51 mostra os possíveis status que os pedidos poderão ter consoante a decisão do gestor acerca da validação dos pedidos. A Figura 52, linha 81 mostra a *query* responsável que faz o *update* na base de dados onde torna o pedido com *status* “Aprovado” caso o gestor aprove os pedidos.

```

73  router.put("/accept/:idRequest", function (req, res) {
74      var id_req = req.params.idRequest;
75      var request_name = req.body.idRequestName;
76      var idAnimal = req.body.idAnimal;
77      console.log(request_name);
78      db.tx(async (t) => {
79          await t
80              .one(
81                  `UPDATE adotame.request
82                     SET status = 'Aprovado'
83                     where id_request = ${id_req}`,
84                  [id_req]
85              )

```

Figura 52 - Query para pedido aprovado

```

55  router.put("/reprove/:idRequest", function (req, res) {
56      var id_req = req.params.idRequest;
57      db.one(
58          `UPDATE adotame.request
59             SET status = 'Reprovado'
60             where id_request = ${id_req}`,
61          [id_req]
62      )

```

Figura 53 - Query para pedido reprovado

A Figura 53 mostra a *query* responsável por fazer o *update* na base de dados que torna o pedido com *status* “Reprovado”, caso o gestor reprove os pedidos.

## 7 Conclusão

Ao longo deste projeto foi desenvolvido um sistema *web* onde se pode adotar ou apadrinhar animais do tipo canídeos e felinos. A ideia do trabalho ainda atinge a necessidade de preocupação sobre animais domésticos que se possuam. Ou seja, a plataforma desenvolvida está preparada para receber pedidos sobre animais domésticos que desapareceram ou que sejam encontrados. No entanto, ainda é possível criar pedidos de “adição de animais” ao sistema, para tornar o catálogo de adoção/apadrinhamento mais abundante e apelativo.

Inicialmente foram traçados objetivos para a implementação do sistema *web* com a colaboração dos *designers* da empresa ProjectBox, e recorrendo ao estudo sobre o diagrama de casos de uso. Posteriormente foi realizado um estudo, onde foram selecionadas três aplicações que iam ao encontro dos objetivos da aplicação Adota.me. Das três aplicações selecionadas, apenas uma se enquadrava melhor em relação ao que se pretendia implementar na solução Adota.me.

De seguida, foi selecionada a metodologia ágil que melhor se enquadrava para o desenvolvimento do sistema. Escolheu-se a metodologia ágil Kanban, devido ao facto de possuir mais controlo sobre as atividades propostas para a elaboração do trabalho.

A próxima etapa, foi efetuada uma análise de requisitos onde foi possível estudar e recolher as funcionalidades que o sistema *web* deveria ter. Visto que, durante a análise de requisitos priorizaram-se os requisitos funcionais. Todos os requisitos funcionais que se pretendiam implementar no sistema, estão descritos na forma sobre histórias de utilizador. Como a aplicação não se encontra totalmente implementada, pensou-se em documentar todas as histórias de utilizador, para que mais tarde, quando se volte a desenvolver o sistema Adota.me, não haja dúvidas acerca do desenvolvimento sobre as histórias do utilizador.

Uma vez definidos os requisitos do sistema, começou-se a implementar a solução final. No decorrer da solução, foram surgindo algumas dúvidas e erros de implementação, onde através de várias tentativas pessoais para resolver os problemas, ou com a ajuda do supervisor, as dúvidas e problemas foram esclarecidos.



Ao longo do desenvolvimento da solução, cada passo programado era testado, ou seja, foram realizados vários testes de *software* ao longo da implementação. Testes variados sobre: construção das rotas e o que elas vão *renderizar*, *queries* provenientes da base de dados que mostram resultados no sistema, construção do *frontend* onde foi imposta a regra de desenvolvimento *mobile first*, validação dos dados inseridos pelo utilizador, e por fim, inserção correta dos dados inseridos pelo utilizador na base de dados.

Como trabalho futuro, prevê-se melhorias no sistema Adota.me. Por exemplo, a realização de uma interface mais dinâmica para o utilizador, onde este poderia consultar todo o seu histórico de atividades que foi realizando no sistema. Outro exemplo que não foi implementado e que se desejaria no sistema atual, seria fechar o sistema com os perfis/permisões. Também se deve corrigir o erro de *login*, acesso ao sistema. Ou seja, ao fim de registado o utilizador no sistema, quando o mesmo utilizador vai fazer login no sistema, o sistema não reconhece a *password* do utilizador. Talvez este erro seja proveniente da encriptação das *passwords* na base de dados. Além disso poderiam ser feitas algumas alterações nos catálogos do sistema *web*. Poder-se-iam implementar no topo dos catálogos, vários filtros de acordo com as características dos animais, de modo a oferecer aos utilizadores uma pesquisa mais acessível sobre os animais que procuram.

Por fim e concluindo este relatório deve-se dizer, que a realização deste projeto em estágio foi, sem dúvida uma experiência muito positiva e enriquecedora. Permitiu o desenvolvimento de competências e a aplicação dos conhecimentos adquiridos ao longo da Licenciatura em Engenharia Informática. Avaliando o resultado final, a satisfação pessoal é positiva com o trabalho desenvolvido, e com a certeza de que a participação neste projeto permitiu crescer pessoalmente a todos os níveis.

## Bibliografia

- [1] ProjectBox, “PROJECTBOX,” PROJECTBOX, [Online]. Available: <https://projectbox.pt/>. [Acedido em 26 novembro 2022].
- [2] N. Tameirão, “METODOLOGIAS ÁGEIS: O QUE SÃO E QUAIS OS PRINCIPAIS TIPOS,” sambatech, 20 08 2021. [Online]. Available: <https://sambatech.com/blog/insights/metodos-ageis/>. [Acedido em julho 2022].
- [3] Jira Software, “Jira Software - Adota.me,” Jira Software, 20 abril 2022. [Online]. Available: <https://ad0tame.atlassian.net/jira/software/projects/AM/boards/1>. [Acedido em 27 outubro 2022].
- [4] Câmara Municipal de Tomar, “Tomar Cidade Templária,” NOESIS, 2020. [Online]. Available: <http://www.cm-tomar.pt/index.php/pt/viver/canil/animais-encontrados>. [Acedido em 28 junho 2022].
- [5] Associação de Municípios de Terras de Santa Maria, “Canil intermunicipal associação de municípios das terras de Santa Maria,” Livetech, 2017. [Online]. Available: <https://amtsm.pt/pt/canil-intermunicipal/sobre-o-ciamtsm-1/>. [Acedido em 23 junho 2022].
- [6] Movimento Internacional para a Defesa dos Animais, “Associação MIDAS,” Netinbound, 2022. [Online]. Available: <https://www.associacaomidas.org/>. [Acedido em 28 junho 2022].
- [7] Movimento Internacional para a Defesa dos Animais, “PROCESSO DE ADOÇÃO – CANÍDEO,” 2022, [Online]. Available: <https://www.associacaomidas.org/proposta-de-adopcao-canideo/>. [Acedido em junho 2022].
- [8] Edward Duffus, “Programa Africano para a Melhoria Acelerada do Registo Civil e Estatísticas Vitais,” APAI-CRVS, [Online]. Available: <http://www.crvs-dgb.org/pt/activities/analise-e-design/8-definir-os-requisitos-do-sistema/#:~:text=Os%20requisitos%20do%20sistema%20s%C3%A3o,%E2%80%99CHierarquia%20dos%20Requisitos%E2%80%99D%20abaixo..> [Acedido em 24 junho 2022].
- [9] G. Primo, “USER STORIES – O QUE SÃO? COMO USAR?,” scrumhalf, [Online]. Available: <https://blog.myscrumhalf.com/user-stories-o-que-sao-como-usar/>. [Acedido em julho 2022].
- [10] Wikipedia, “Wikipedia - Visual Studio Code,” Wikipedia, 19 janeiro 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Visual\\_Studio\\_Code](https://en.wikipedia.org/wiki/Visual_Studio_Code). [Acedido em agosto 2022].
- [11] PostgreSQL, “PostgreSQL: What is PostgreSQL?,” PostgreSQL, [Online]. Available: <https://www.postgresql.org/about/>. [Acedido em agosto 2022].
- [12] Bootstrap, “Bootstrap,” Bootstrap, [Online]. Available: <https://getbootstrap.com/docs/5.2/getting-started/introduction/>. [Acedido em agosto 2022].
- [13] L. Torvalds, “git --everything-is-local,” Microsoft Corporation, 10 abril 2008. [Online]. Available: <https://git-scm.com/doc>. [Acedido em agosto 2022].
- [14] Figma, “Components, Styles, and documentation,” Figma, 27 setembro 2017. [Online]. Available: <https://www.figma.com/best-practices/guide-to-developer-handoff/components-styles-and-documentation/>. [Acedido em julho 2022].
- [15] Khan Academy, “Khan Academy - jQuery,” Khan Academy, [Online]. Available: <https://www.khanacademy.org/computing/computer-programming/html-js-jquery/jquery-intro/v/what-is-jquery>. [Acedido em 27 outubro 2022].
- [16] Npm, “NPM Docs,” GitHub, Microsoft Corporation, npm, 2009. [Online]. Available: <https://docs.npmjs.com/cli/v8/commands/npm-init>. [Acedido em agosto 2022].
- [17] NodeJs, “Npm Docs,” NodeJs, 2022. [Online]. Available: <https://www.npmjs.com/package/nodemon>. [Acedido em agosto 2022].

- [18] NodeJs, “ExpressJs,” NodeJs, 2022. [Online]. Available: <https://www.npmjs.com/package/express>. [Acedido em agosto 2022].
- [19] GeeksforGeeks, “Express.js | app.set() Function,” GeeksforGeeks, 10 outubro 2020. [Online]. Available: <https://www.geeksforgeeks.org/express-js-app-set-function/>. [Acedido em agosto 2022].
- [20] GeeksforGeeks, “Node.js http2.createServer() Method,” GeeksforGeeks, 3 novembro 2020. [Online]. Available: <https://www.geeksforgeeks.org/node-js-http2-createserver-method/?ref=gcse>. [Acedido em agosto 2022].
- [21] GeeksforGeeks, “Express.js | app.listen() Function,” GeeksforGeeks, 6 novembro 2021. [Online]. Available: <https://www.geeksforgeeks.org/express-js-app-listen-function/>. [Acedido em agosto 2022].
- [22] GeeksforGeeks, “Express.js | app.use() Function,” GeeksforGeeks, 18 junho 2020. [Online]. Available: <https://www.geeksforgeeks.org/express-js-app-use-function/?ref=gcse>. [Acedido em agosto 2022].
- [23] GeeksforGeeks, “Difference between Software and Middleware,” GeeksforGeeks, 9 dezembro 2020. [Online]. Available: <https://www.geeksforgeeks.org/difference-between-software-and-middleware/?ref=gcse>. [Acedido em agosto 2022].
- [24] Mozilla , “HTTP request methods,” mdn web docs, [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>. [Acedido em agosto 2022].
- [25] Npm, “Embedded JavaScript templates,” NodeJs, maio 2022. [Online]. Available: <https://www.npmjs.com/package/ejs>. [Acedido em agosto 2022].
- [26] GeeksforGeeks, “Express.js res.render() Function,” GeeksforGeeks, 19 agosto 2021. [Online]. Available: <https://www.geeksforgeeks.org/express-js-res-render-function/>. [Acedido em agosto 2022].
- [27] Npm, “express-ejs-layouts,” NodeJs, 2021. [Online]. Available: <https://www.npmjs.com/package/express-ejs-layouts>. [Acedido em agosto 2022].
- [28] A. Forshaw, “TheCatApi // Developer Experience,” Aden Forshaw, 2012. [Online]. Available: <https://docs.thecatapi.com/>. [Acedido em agosto 2022].
- [29] A. Forshaw, “The Dog API - Dogs as a Service,” Aden Forshaw, 2012. [Online]. Available: <https://docs.thedogapi.com/api-reference>. [Acedido em agosto 2022].
- [30] PostgreSQL, “UUID Type,” PostgreSQL, 8 setembro 2022. [Online]. Available: <https://www.postgresql.org/docs/current/datatype-uuid.html>. [Acedido em agosto 2022].
- [31] PostgreSQL, “UUID Functions,” PostgreSQL, 8 setembro 2022. [Online]. Available: <https://www.postgresql.org/docs/current/functions-uuid.html>. [Acedido em agosto 2022].
- [32] Npm, “bcrypt,” NodeJs, 2020. [Online]. Available: <https://www.npmjs.com/package/bcrypt>. [Acedido em agosto 2022].
- [33] V. Tomilov, “pg-promise,” Vitaly Tomilov, 2015. [Online]. Available: <https://github.com/vitaly-t/pg-promise#about>. [Acedido em agosto 2022].



## Anexos

### A 1. Criação das tabelas do ficheiro createTables.sql

```
create schema if not exists adotame;

create table if not exists adotame.user(
    id_user uuid primary key,
    name varchar(250) not null,
    email varchar(100) unique not null,
    phone bigint null
);

create table if not exists adotame.login(
    id_login uuid primary key,
    username varchar(250) not null,
    password varchar(100) not null,
    id_user uuid references adotame.user (id_user)
);

create table if not exists adotame.profile(
    id_profile uuid primary key,
    name varchar(250) not null
);

create table if not exists adotame.login_profile(
    id_login uuid references adotame.login (id_login),
    id_profile uuid references adotame.profile (id_profile)
);

create table if not exists adotame.permission(
    id_permission uuid primary key,
    name varchar(250) not null
);

create table if not exists adotame.profile_permission(
    id_permission uuid references adotame.permission (id_permission),
    id_profile uuid references adotame.profile (id_profile)
);

create table if not exists adotame.catalog(
    id_catalog uuid primary key,
    name varchar(250) not null
);

create table if not exists adotame.animal(
    id_animal uuid primary key,
    name varchar(100) not null,
    type varchar(100) check(type in('Cao', 'Gato', 'Todos')),
    photo varchar(500) not null,
    gender varchar(100) check(gender in('Macho', 'Femea')),
    birth_date date not null,
    size varchar(50) check(size in('Muito pequeno', 'Pequeno', 'Medio',
'Grande', 'Muito grande')),
```

```

    fur varchar(100) null,
    breed varchar(300) null,
    color varchar(250) null,
    vaccines varchar(500) null,
    portion varchar(300) null,
    health varchar(500) null,
    cares varchar(1000) null,
    place_belongs varchar(500) null
);

create table if not exists adotame.catalog_animal(
    id_catalog uuid references adotame.catalog (id_catalog),
    id_animal uuid references adotame.animal (id_animal)
);

create table if not exists adotame.request_type(
    id_request_type uuid primary key,
    request_name varchar(250) check(request_name in('Reportar
desaparecido', 'Avistar desaparecido', 'Adicionar animal', 'Adotar',
'Apadrinhar'))
);

create table if not exists adotame.request(
    id_request uuid primary key,
    date_request date not null,
    status varchar(100) check(status in('Aprovado', 'Reprovado',
'Pendente')),
    id_user uuid references adotame.user (id_user),
    id_request_type uuid references adotame.request_type
(id_request_type),
    birth_date date null,
    nif bigint null,
    address varchar(250) null,
    postal_code varchar(100) null,
    locality varchar(100) null,
    phone bigint null,
    description varchar(5000) null,
    married varchar(50) check(married in('Sim', 'Nao')) null,
    childs varchar(50) check(childs in('Sim', 'Nao')) null,
    live_with varchar(500) null,
    home_agreement varchar(50) check(home_agreement in('Sim', 'Nao'))
null,
    allergies_in_relatives varchar(50) check(allergies_in_relatives
in('Sim', 'Nao')) null,
    main_caregiver_name varchar(250) null,
    caregiver_long varchar(50) check(caregiver_long in('Sim', 'Nao'))
null,
    caregiver_illness_name varchar(250) null,
    why_adopt varchar(1000) null,
    yard varchar(50) check(yard in('Sim', 'Nao')) null,
    animal_sleep_place varchar(500) null,
    animal_loneless_daytime varchar(50) check(animal_loneless_daytime
in('Sim', 'Nao')) null,
    animal_alone_place varchar(500) null,
    playtime varchar(50) check(playtime in('Sim', 'Nao')) null,
    pet_before varchar(50) check(pet_before in('Sim', 'Nao')) null,
    pet_nowdays varchar(50) check(pet_nowdays in('Sim', 'Nao')) null,

```

```

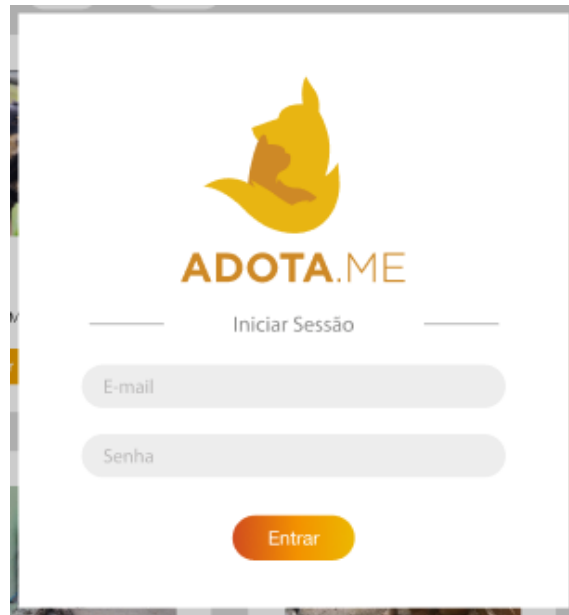
        animal_cares_expenses varchar(50) check(animal_cares_expenses
in('Sim', 'Nao')) null,
        teach_plans varchar(1000) null,
        moving_home_animal_effects varchar(1000) null,
        give_up_circumstances varchar(1000) null,
        financial_payment_method varchar(50) check(financial_payment_method
in('Semanal', 'Mensal', 'Anual')) null,
        value_amount int null,
        hobby varchar(1000) check(hobby in('Passeios', 'Fins de semana',
'Divulgacao do animal', 'Vacinacao', 'Tratamentos', 'Tosquia')) null,
        last_seen_place varchar(2000) null,
        seen_place varchar(5000) null,
        details varchar(5000) null,
        id_animal uuid null,
        report_animal_name varchar(500) null,
        report_animal_type varchar(100) check(report_animal_type in('Cao',
'Gato', 'Todos')) null,
        report_animal_photo varchar(500) null,
        report_animal_gender varchar(100) check(report_animal_gender
in('Macho', 'Femea')) null,
        report_animal_size varchar(50) check(report_animal_size in('Muito
pequeno', 'Pequeno', 'Medio', 'Grande', 'Muito grande')) null,
        report_animal_fur varchar(100) null,
        report_animal_breed varchar(100) null,
        report_animal_color varchar(250) null,
        report_animal_birth_date date null,
        report_animal_vaccines varchar(500) null,
        report_animal_portion varchar(500) null,
        report_animal_health varchar(500) null,
        report_animal_cares varchar(1000) null,
        report_animal_location varchar(500) null
);

create table if not exists adotame.animal_status(
    id_animal_status uuid primary key,
    status varchar(100) check(status in('Adotar', 'Apadrinhar', 'Reportar
desaparecido'))
);

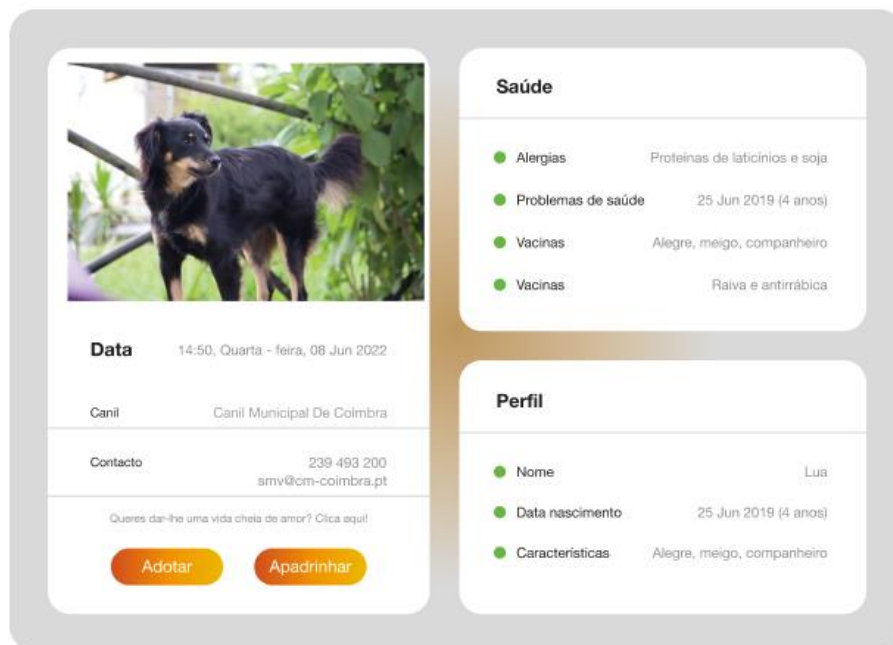
create table if not exists adotame.animal_animal_status(
    id_animal uuid not null references adotame.animal (id_animal),
    id_animal_status uuid not null references adotame.animal_status
(id_animal_status),
    id_request uuid not null references adotame.request (id_request),
    constraint pk_animal_status primary key (id_animal, id_animal_status,
id_request)
);

```

## A 2. Modal Login partilhada pelo Figma



A 3. Interface características do animal partilhada pelo Figma



A 4. Interface para adotar animal partilhada pelo Figma




### Tens a certeza que queres uma nova companhia?

Responde a este inquérito para avaliarmos se tens o que é preciso para adotar um novo companheiro.

1. És casado/a? (Caso seja, escreve o nome da pessoa no espaço em branco)

☐ Sim
☐ Não



**Lua**  
25 Jun 2019 (4 anos)

2. Tens filhos?

☐ Sim
☐ Não

3. Com quem vives?

4. Todas as pessoas em sua casa concordam com a adoção?

☐ Sim
☐ Não

5. Alguém em sua casa tem alergias ou asma?

☐ Sim
☐ Não

6. Quem é que vai ser o cuidador principal?

7. Está preparado/a para cuidar deste animal por 10-15 anos?

☐ Sim
☐ Não

8. Quem é que vai cuidar do animal se ficar doente ou incapaz de o fazer?

9. Porque é que quer adotar um animal?

10. Onde vives? (Morada completa)

11. Tens quintal?

☐ Sim
☐ Não

12. Onde é que o animal vai dormir?

13. Vai deixar o cão sozinho ao longo do dia?

☐ Sim
☐ Não

14. Onde é que o cão vai ser deixado sozinho?

15. Vai ter tempo para brincar com o seu cão?

☐ Sim
☐ Não

16. Já teve algum animal de estimação antes?

☐ Sim
☐ Não

17. Tem atualmente algum animal de estimação? (Se sim, quais)

☐ Sim
☐ Não

18. Tem possibilidades de pagar todas as despesas veterinárias?

☐ Sim
☐ Não

19. Como planeia ensinar o seu animal?

20. O que fará ao animal se mudar de casa?

21. Em que circunstâncias desistiria do animal?

22. O que fará com o animal se não conseguir ficar com ele?

☐ Autorizo a utilização dos dados inseridos, em conformidade com a Política de Privacidade

Enviar

## A 5. Interface consultar e validar pedidos partilhada pelo Figma

