UNIVERSIDADE FEDERAL DO ABC
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO DE ESTÁGIO SUPERVISIONADO I

GUILHERME ALVES AGOSTINELLI

Orientador: Prof. Dr. Francisco Isidro Massetto

Santo André – SP
2018

GUILHERME ALVES AGOSTINELLI

PRESTAÇÃO DE SERVIÇOS DE DESENVOLVIMENTO DE UM SISTEMA DE GESTÃO VETERINÁRIO

Relatório de estágio apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Federal do ABC como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Francisco Isidro Massetto

Santo André – SP
2018

DEDICATÓRIA

Dedico este trabalho aos meus familiares, professores e amigos que se mantiveram sempre ao meu lado e que me influenciaram a ser a pessoa, o aluno e o profissional que sou hoje.

AGRADECIMENTOS

Agradeço ao professor orientador Francisco Isidro Massetto por ter me aprimorado pessoal e profissionalmente e à empresa Felsan Soluções em TI por todo o suporte ao longo do desenvolvimento deste projeto, possibilitando que a base acadêmica oferecida durante o curso de Ciência da Computação pudesse ser aplicada em uma solução de mercado.

"Talvez não tenhamos conseguido fazer o melhor, mas lutamos para que o melhor fosse feito."

Martin Luther King

RESUMO

O presete relatório visa descrever as tarefas realizadas durante a prestação de serviços da empresa Agoyama Soluções em Tecnologia da Informação para a empresa Felsan Soluções em TI. As atividades foram desenvolvidas sob o escopo da disciplina de Estágio Supervisionado I do curso de Bacharelado em Ciência da Computação da Universidade Federal do ABC, totalizando 96 horas de trabalho. O trabalho consiste no desenvolvimento de partes de um sistema de gestão veterinário que pode ser utilizado nas rotinas de atendimento de clínicas, hospitais, consultórios, centros estéticos e *pet shops*. São mostradas as motivações para a realização do trabalho, a descrição das empresas envolvidas, além da apresentação dos resultados obtidos, da aprendizagem adquirida e dos conhecimentos acadêmicos necessários para o desenvolvimento das atividades.

Palavras-Chave: Desenvolvimento, Atividades, Sistema, Gestão, Veterinário.

ABSTRACT

This report aims to describe the tasks developed during the provision of services of the company Agoyama Soluções em Tecnologia da Informação to the company Felsan Soluções em TI. The activities were developed under the scope of the Estágio Supervisionado I course which is part of the Bachelor in Computer Science at Universidade Federal do ABC, totalling 96 hours of work. The work consists in developing parts of a veterinary management system that can be used by clinics, hospitals, aesthetic centers and pet shops. There is a demonstration of the motivations for the work, a description of the involved companies, and also the presentation of the results, of the acquired knowledge and of the academic concepts needed for the activities development.

Keywords: Development, Activities, System, Management, Veterinary.

Lista de Figuras

Lista de Tabelas

Sumário

1 INTRODUÇÃO

Este documento visa apresentar o trabalho desenvolvido na disciplina de Estágio Supervisionado I. Nesta seção, são apresentadas as inspirações do projeto, uma descrição da empresa tomadora de serviços, da empresa prestadora de serviços, e também uma visão geral das metas de negócio de cada empresa e dos objetivos com a realização deste trabalho.

1.1 DESCRIÇÃO DA EMPRESA CONTRATANTE

A Felsan Soluções em TI é uma empresa de Tecnologia da Informação atuante no mercado desde 2014 e é para ela que o projeto presente neste relatório está sendo desenvolvido. A empresa conta com um forte time de desenvolvimento e de vendas focados em construir e entregar soluções de qualidade para seus clientes.

As atividades desenvolvidas incluem análise e modelagem, construção de *Application Programming Interfaces* (APIs) e desenvolvimento de aplicações *web* e *mobile*. Como exemplo mais famoso, está o sistema de controle de ponto que pode ser utilizado via aplicativo ou via telefone, além de fornecer um painel *online* de gerenciamento de colaboradores para acompanhar horas trabalhadas, atrasos e horas extras.

1.2 DESCRIÇÃO DA EMPRESA CONTRATADA

A Agoyama Soluções em Tecnologia da Informação é uma *startup* de tecnologia fundada em 2018 que nasceu com o objetivo de dar vida às ideias, entregar produtos de qualidade e desenvolver soluções tecnológicas disruptivas e inovadoras.

Através da união de tecnologia, agilidade e boas práticas, a Agoyama presta serviços de consultoria e desenvolvimento de programas de computador *web* e *mobile* sob encomenda, além de possuir seus próprios projetos de tecnologia. O seu principal produto, que ainda está em fase de desenvolvimento, é a plataforma de agendamento de

serviços veterinários; por meio dela, os proprietários podem agendar e pagar serviços como consulta, banho e tosa diretamente por meio de uma aplicação *web*.

1.3 DESCRIÇÃO DAS MOTIVAÇÕES E DOS OBJETIVOS DO TRABALHO

Com o cenário positivo do mercado *pet*, um dos únicos que continuaram a apresentar crescimento mesmo em época de crise econômica [1], a Felsan enxergou a oportunidade de usar sua *expertise* de vendas para fornecer uma solução de gerenciamento veterinário para clínicas, consultórios, *pet shops*, centros estéticos e hospitais. Desse modo, resolvendo o problema de que muitos estabelecimentos ainda fazem o controle de clientes e prontuários em papel.

Devido à complexidade do projeto e para não sobrecarregar seu time de desenvolvimento, a Felsan fez uma parceria com a empresa Agoyama para que a mesma atuasse no desenvolvimento do sistema, enquanto a Felsan fica com a responsabilidade das atividades de *marketing* e de vendas do produto.

Concomitantemente, também enxergando o crescimento do mercado *pet*, a Agoyama decide por desenvolver seu principal produto, uma plataforma de agendamento de serviços veterinários. Com a consonância dos objetivos de negócio das duas empresas, a Agoyama entende a parceria como uma excelente oportunidade e decide por também desenvolver o projeto para a Felsan, já que futuramente poderá agregar a funcionalidade de gerenciamento no seu produto principal.

1.4 CARACTERÍSTICAS DA ÁREA DE REALIZAÇÃO DO TRABALHO

O trabalho é realizado de maneira flexível, acontecendo tanto na própria sede da Felsan na Av. Paulista, como na própria sede da Agoyama também na mesma região, ou de maneira remota. A jornada de trabalho é de segunda-feira à sexta-feira das 09:00 às 16:00 com 1 hora de almoço, totalizando 30 horas semanais.

A sede da Felsan se encontra no espaço de *coworking* da empresa Gowork, no qual possuem uma sala privativa onde trabalham em média 5 funcionários dos setores de Desenvolvimento, Suporte e Vendas. Os setores são responsáveis, respectivamente, pelo desenvolvimento de aplicações, pelo atendimento aos clientes (ajuda via e-mail ou telefone para a utilização dos sistemas e acompanhamento de problemas nas aplicações) e pelo planejamento de vendas do produto com base nas decisões de negócio.

A sede da Agoyama também se encontra em um espaço de *coworking*, porém da empresa Campus Inc. A Agoyama, por outro lado, não possui uma sala privativa e só utiliza o espaço de *coworking* e as salas de reunião.

Além disso, os espaços de *coworking* onde a Felsan e a Agoyama residem também podem ser utilizados para fomentar a geração de novos negócios e estimular o *networking* dentre as empresas que ali trabalham. Cada funcionário trabalha em seu próprio computador e a área física comporta suficientemente a quantidade total de pessoas.

São nesses dois lugares onde são realizadas as reuniões com o supervisor do estágio para que seja feito o acompanhamento do trabalho realizado e sejam trocadas experiências e boas práticas de atuação. O desenvolvimento é realizado majoritariamente no espaço do *coworking*, porém de tempos em tempos também é feito remotamente.

1.5 VISÃO GERAL DO TRABALHO

Nos três primeiros dias foram feitas reuniões a respeito dos objetivos do projeto para que o escopo do mesmo fosse pormenorizado, permitindo o entendimento dos dados a serem abordados e modelados. Dessa forma, foi possível realizar o alinhamento das atividades e dos métodos de trabalho.

A partir daí, deu-se início ao desenvolvimento do sistema de gestão, que precisou de uma série de conhecimentos abordados durante a graduação, dentre eles: meto-

dologias ágeis de desenvolvimento de *software*, banco de dados, programação orientada à objetos, programação para *web*, responsividade e servidores remotos. Durante a elaboração do relatório, o projeto segue em fase de desenvolvimento e a completude do sistema possui como prazo final, conforme estipulado em contrato de prestação de serviços, a data de 31 de maio de 2019.

2 ATIVIDADES DESENVOLVIDAS

Nesta seção, são apresentadas as atividades realizadas durante o período de estágio supervisionado, abrangendo as etapas desde a formulação até os resultados obtidos. Começa-se falando sobre o planejamento e escolha das metodologias, das tecnologias utilizadas e também a respeito da modelagem de dados. Em seguida, é detalhado o desenvolvimento das partes do sistema.

2.1 PLANEJAMENTO E ESCOLHA DAS METODOLOGIAS

O planejamento é uma tarefa essencial e uma das etapas mais importantes no desenvolvimento de um novo projeto. Sem ele, não se tem uma visão clara do produto e o fluxo de trabalho se torna caótico e confuso, podendo gerar desperdício de recursos no desenvolvimento de funcionalidades mal interpretadas e atrasos na entrega do projeto.

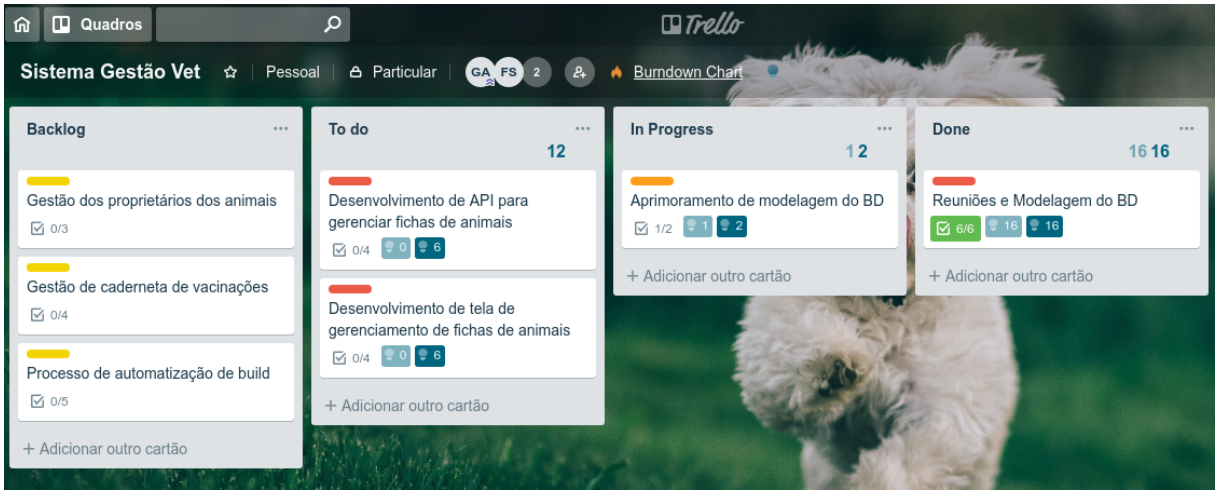
Foi pensando nisso que o foco durante as primeiras reuniões entre as empresas foi trocar o conhecimento de mercado que cada um tinha, mostrar as informações coletadas durante reuniões com clientes de estabelecimentos veterinários e compartilhar boas e más práticas de desenvolvimento e gestão.

Com isso, optou-se por utilizar a metodologia ágil Scrum e combiná-la com o sistema visual de gerenciamento Kanban, ambas explicadas mais à frente na seção de Fundamentação do relatório. A escolha foi feita pois já existia uma experiência prévia positiva dos envolvidos com as ferramentas citadas, ficando decidido que o supervisor

iria assumir o papel de *Scrum Master* e que seriam feitas reuniões diárias e semanais para que o acompanhamento do projeto fosse feito e também para que a metodologia de desenvolvimento pudesse ser discutida e aprimorada para se adaptar melhor às necessidades da equipe.

Tudo foi gerenciado por meio do sistema *online* Trello[1], onde são criados espécies de "cartões" virtuais onde são descritas cada uma das *features* e nos quais podem ser adicionados *checklists* e datas de entrega. Também por meio da ferramenta é possível arrastar facilmente uma *feature* de posição e adicionar etiquetas para categorizar melhor cada tarefa e sua prioridade.

Figura 2.1: Etapas do Kanban implementadas na ferramenta *online* Trello.



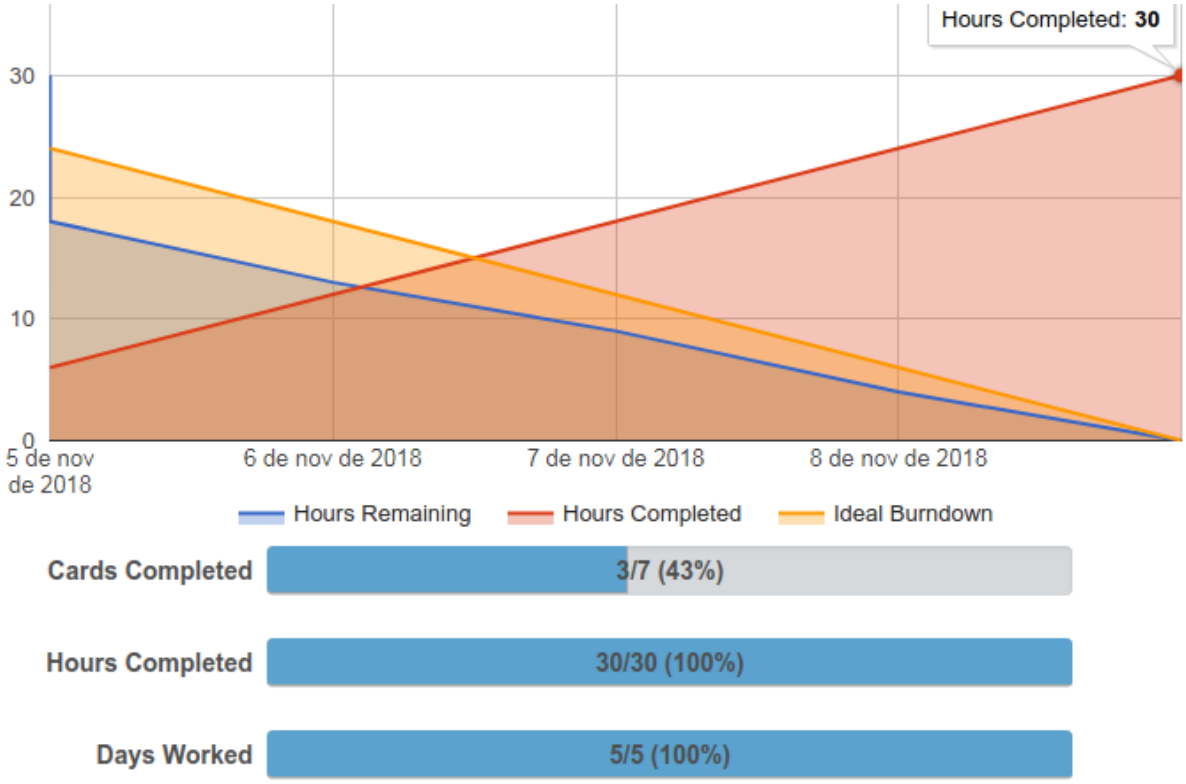
Fonte: elaborado pelo autor.

Também por meio do Trello foi possível adicionar um plugin[2] para o Scrum que permite a adição do tempo estimado e efetivamente gasto para cada tarefa, gerando um gráfico de *burndown* que mostra como as horas restantes se comportam em relação às horas totais de trabalho. A princípio, foi definido que cada *sprint* (iteração de desenvolvimento) duraria uma semana.

[1] Trello. Disponível em: <https://trello.com>

[2] Burndown for Trello. Disponível em: <https://trello.com/power-ups/569f8a70a115d18c5ea9af05/burndown-for-trello>

Figura 2.2: Gráfico de *burndown* do Scrum mostrando a realização de uma *sprint* de 30 horas.



Fonte: elaborado pelo autor.

2.2 ESCOLHA DAS TECNOLOGIAS

Com as reuniões e discussões sobre o projeto, ficou decidido que o sistema de gerenciamento para estabelecimentos veterinários seria desenvolvido como um sistema *web*, sendo utilizado Angular[3] no *front-end*, Node.js[4] no *back-end* e MySQL[5] como banco de dados relacional, todas essas tecnologias sendo melhor apresentadas na seção de Fundamentação.

A decisão de ser um sistema *web* foi feita para que futuramente o sistema pudesse ser integrado à solução de agendamento da Agoyama e também pensando que o dono do estabelecimento veterinário, ou outro funcionário encarregado, pudesse acessar o sistema de qualquer lugar com internet como o *desktop*, tablet ou celular. Como o Angular já tinha sido utilizado em projetos anteriores da Felsan, a tecnologia foi escolhida pois isso facilitaria o acompanhamento e manutenção futura do sistema.

[3]Angular. Disponível em: <https://angular.io/>
[4]Node.js. Disponível em: <https://nodejs.org/en/>
[5]MySQL. Disponível em: <https://www.mysql.com/>

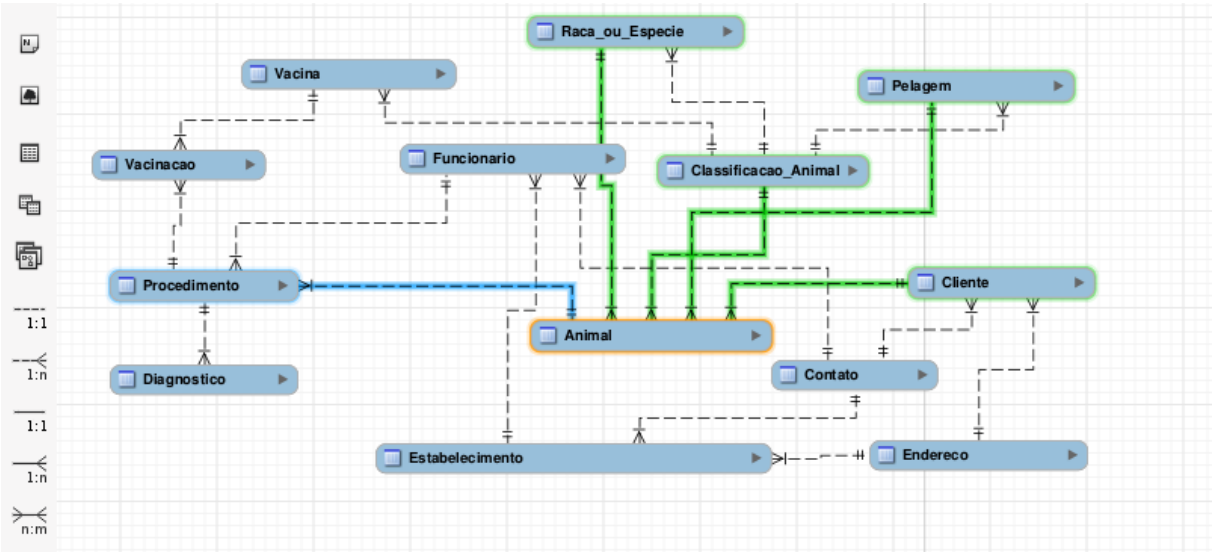
A escolha do Node.js foi em partes baseada na linguagem de desenvolvimento JavaScript, que é a mesma do *front-end* e portanto promoveria maior facilidade de entendimento. Outro fator levado em consideração foi o fato do conceito de arquitetura do Node.js poder ser facilmente estendido para que a API possa ser consumida em um futuro aplicativo *mobile*.

Para o armazenamento e manipulação de dados, o MySQL foi escolhido por se apresentar como um dos mais populares sistemas de gerenciamento de banco de dados gratuitos, com um grande potencial e performance. Além disso, a prévia exposição a sua sintaxe e funcionamento nas disciplinas de graduação conferiram maior domínio e facilidade para o desenvolvimento.

2.3 MODELAGEM DE DADOS

De modo a realizar a modelagem dos dados, foi utilizada a linguagem *Structured Query Language* (SQL) para criar o banco de dados relacional. O modelo relacional normalizado que mapeia as entidades e os relacionamentos de acordo com as cardinalidades mais adequadas para cada caso pode ser observado no Diagrama Entidade-Relacionamento (DER) da Figura 2.3.

Figura 2.3: Diagrama Entidade Relacionamento do projeto.



Fonte: elaborado pelo autor.

É possível observar que a principal tabela é a tabela Animal, já que é a entidade que possui a maior quantidade de relacionamentos - o que era esperado, dado que o animal é o objeto central do sistema.

Por meio dessa modelagem, é possível reutilizar informações como raça, espécie e pelagem para diferentes animais. Além disso, permite que cada cliente (proprietário do animal) possua mais de um *pet* cadastrado no sistema. Conforme estipulado em entrevistas com proprietários de clínicas e *pet shops*, cada cliente geralmente possui um cadastro de informações de contato e endereço.

Através da tabela Estabelecimento, cada estabelecimento que adquirir o sistema é cadastrado e são armazenadas suas informações básicas de contato e endereço, além do registro de seus funcionários. A princípio, o cadastro de funcionários é básico somente com o nome, porém no decorrer do período de prestação de serviços serão criadas seções no sistema para fazer a administração dos funcionários, sua área de atuação e seus privilégios de acesso.

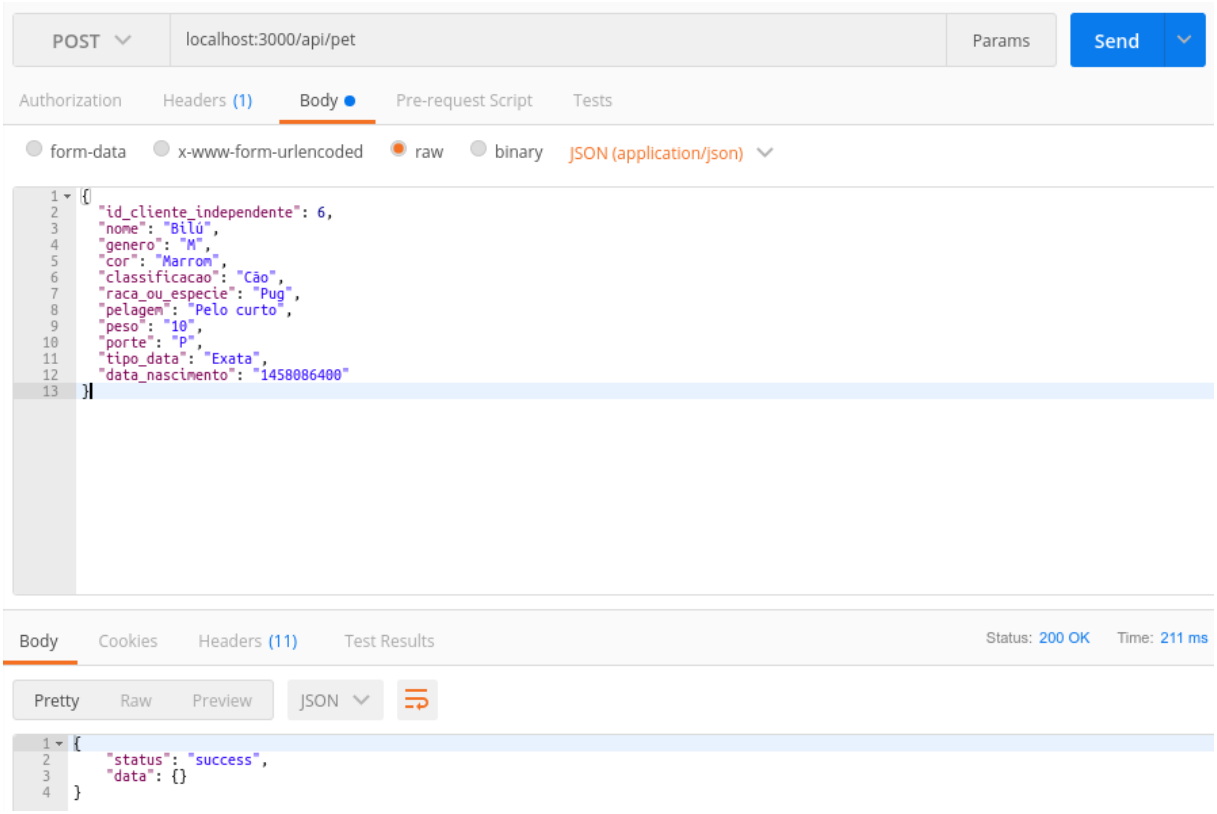
Por fim, essa modelagem permite o armazenamento de procedimentos como banhos, tosas, consultas, vacinações etc, servindo como fonte para a gestão das fichas dos animais como modo de substituir os prontuários em papel. O ponto central de cada atendimento é a tabela Procedimento, que pode possuir um diagnóstico associado e portanto servir como base de *tracking* de todas as informações clínicas do animal como anamnese e prescrições de remédios.

2.4 CRIAÇÃO DA API

Para que a aplicação *web* do *front-end* pudesse persistir o resultado das interações dos usuários com os formulários e botões, foi preciso primeiramente criar uma aplicação para rodar no servidor e fornecer uma interface de comunicação entre o *front-end* e o banco de dados. Para isso, foi criada uma aplicação utilizando Node.js para a qual é possível solicitar, gravar e editar recursos por meio de solicitações HTTP às rotas predefinidas. O formato padrão adotado foi o *JavaScript Object Notation* (JSON)

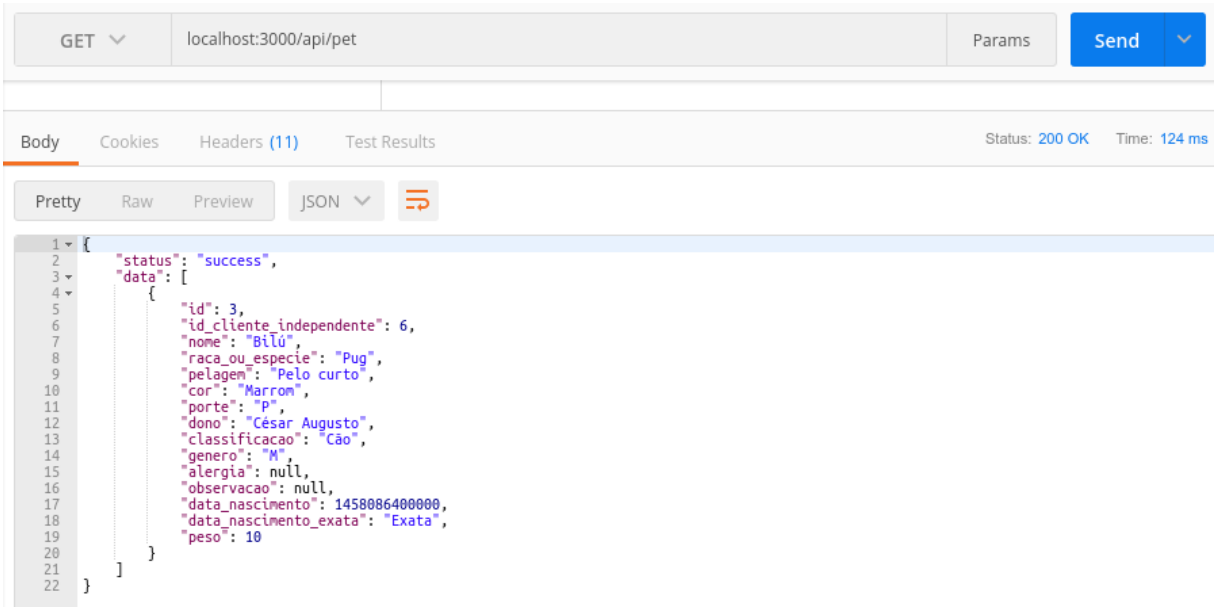
devido à compatibilidade intrínseca de objetos JavaScript com JSONs e também pelo fato do mesmo ser mais eficiente do que outros formatos como o famoso *EXtensible Markup Language* (XML) cuja sintaxe mais longa acaba por desperdiçar recursos de banda para ser trafegado.

Figura 2.4: Requisição POST para a rota /api/pet para cadastrar um novo animal no banco.



Fonte: elaborado pelo autor.

Figura 2.5: Requisição GET para a rota /api/pet que retorna o animal recém-cadastrado no banco.



Fonte: elaborado pelo autor.

O *software* Postman[6] foi utilizado para fazer o teste visual de cada uma das rotas e

[6]Postman. Disponível em: <https://www.getpostman.com/>

verificar que elas retornam as informações desejadas. Com ele é possível visualizar os cabeçalhos HTTP enviados em cada requisição, ver o *status* de resposta do servidor e "embelezar" o JSON retornado como resposta. Por ser muito legível, a notação JSON permite que o programador rapidamente consiga identificar a informação desejada no momento dos testes visuais.

2.5 GESTÃO DE PROPRIETÁRIOS E SEUS ANIMAIS

O primeiro passo do sistema foi construir a tela de cadastro de clientes (proprietários dos animais). Por meio do Angular e do Bootstrap[7], foi possível codificar uma interface responsiva e um formulário completo com máscaras e validações.

Figura 2.6: Formulário de cadastro de clientes.

Fonte: elaborado pelo autor.

Acessando a tela de consulta, é possível verificar e pesquisar rapidamente as informações desejadas dos clientes, otimizando um processo muito comum nas clínicas que é a busca de prontuários de papel em enormes ficheiros. O processo de alteração também é alcançável através desta tela.

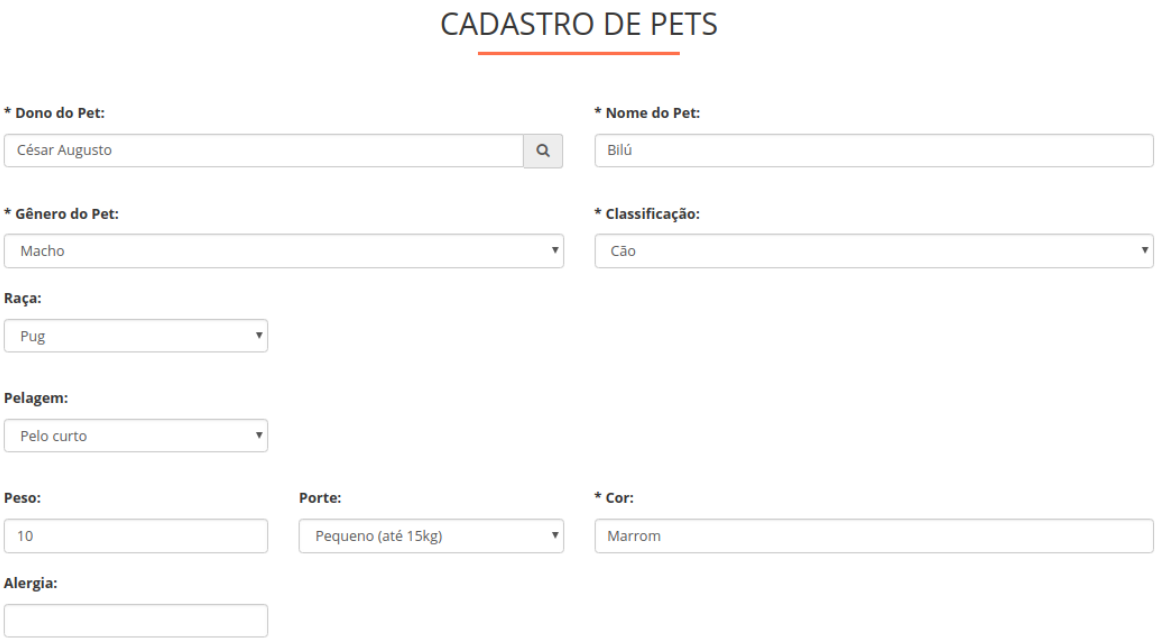
[7]Bootstrap. Disponível em: <https://getbootstrap.com/>

Figura 2.7: Tela de consulta de clientes.

Fonte: elaborado pelo autor.

De modo análogo, foram desenvolvidas telas do sistema para cadastro, consulta e alteração de animais.

Figura 2.8: Formulário de cadastro de animais.



Fonte: elaborado pelo autor.

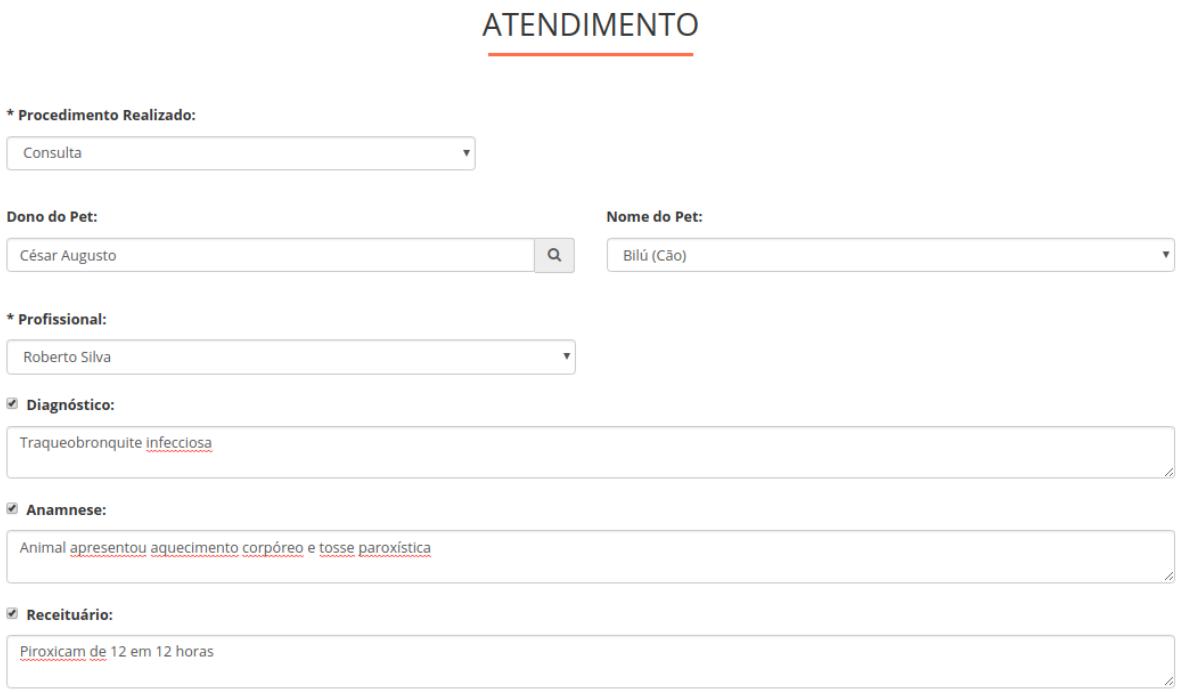
Através do formulário de cadastro, o usuário consegue rapidamente associar um dono ao *pet* sendo inserido. Já a tela de consulta é uma central rápida para fazer o acompanhamento do animal como consultar seu peso atual e verificar se possui alguma alergia antes de realizar algum procedimento. Com as novas interfaces de cadastro, consulta e alteração, estima-se que o tempo médio de espera de um cliente em um estabelecimento veterinário diminui cerca de 5 minutos quando comparado a um estabelecimento que ainda realiza controle em papel.

2.6 GESTÃO DE ATENDIMENTO E FICHA DOS ANIMAIS

Um dos pontos cruciais do sistema é o formulário de atendimento. Por meio dele, o usuário pode rapidamente cadastrar um procedimento clínico ou estético, informar o profissional responsável pelo procedimento e opcionalmente indicar o diagnóstico, os sintomas apresentados e o receituário.

Com isso, as informações ficaram centralizadas em um só lugar, fáceis de serem encontradas e permitindo uma melhor legibilidade tanto por parte dos proprietários dos animais ao lerem as instruções de medicação impressas, como também por parte dos veterinários ao lerem diagnósticos anteriores digitalmente; com isso, o tempo gasto para entender a escrita de outros profissionais é eliminado e o veterinário otimiza seu tempo.

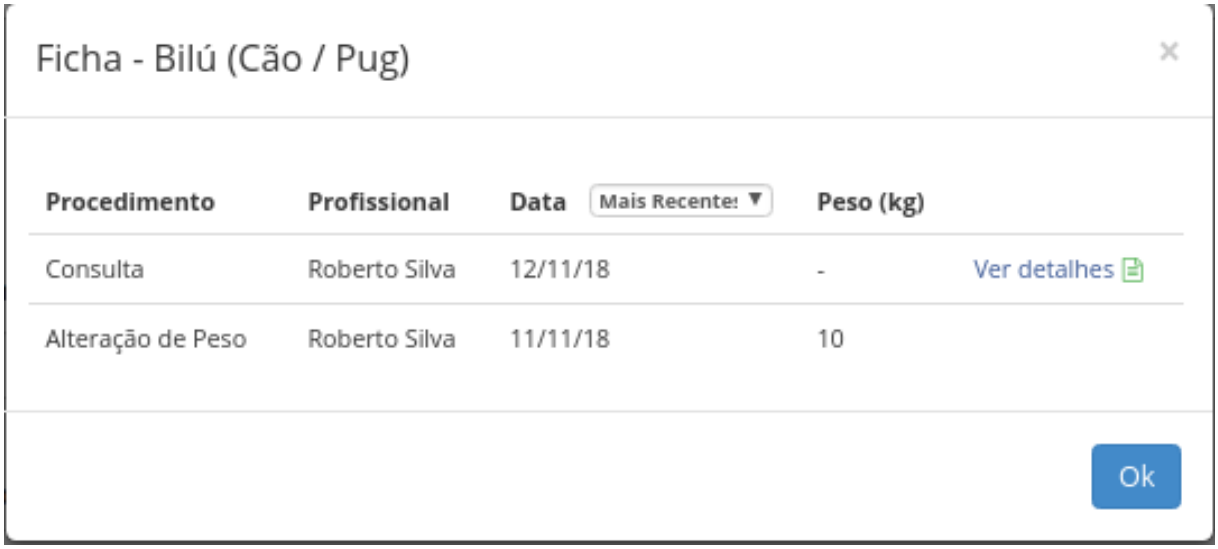
Figura 2.9: Tela de realização de atendimento.



Fonte: elaborado pelo autor.

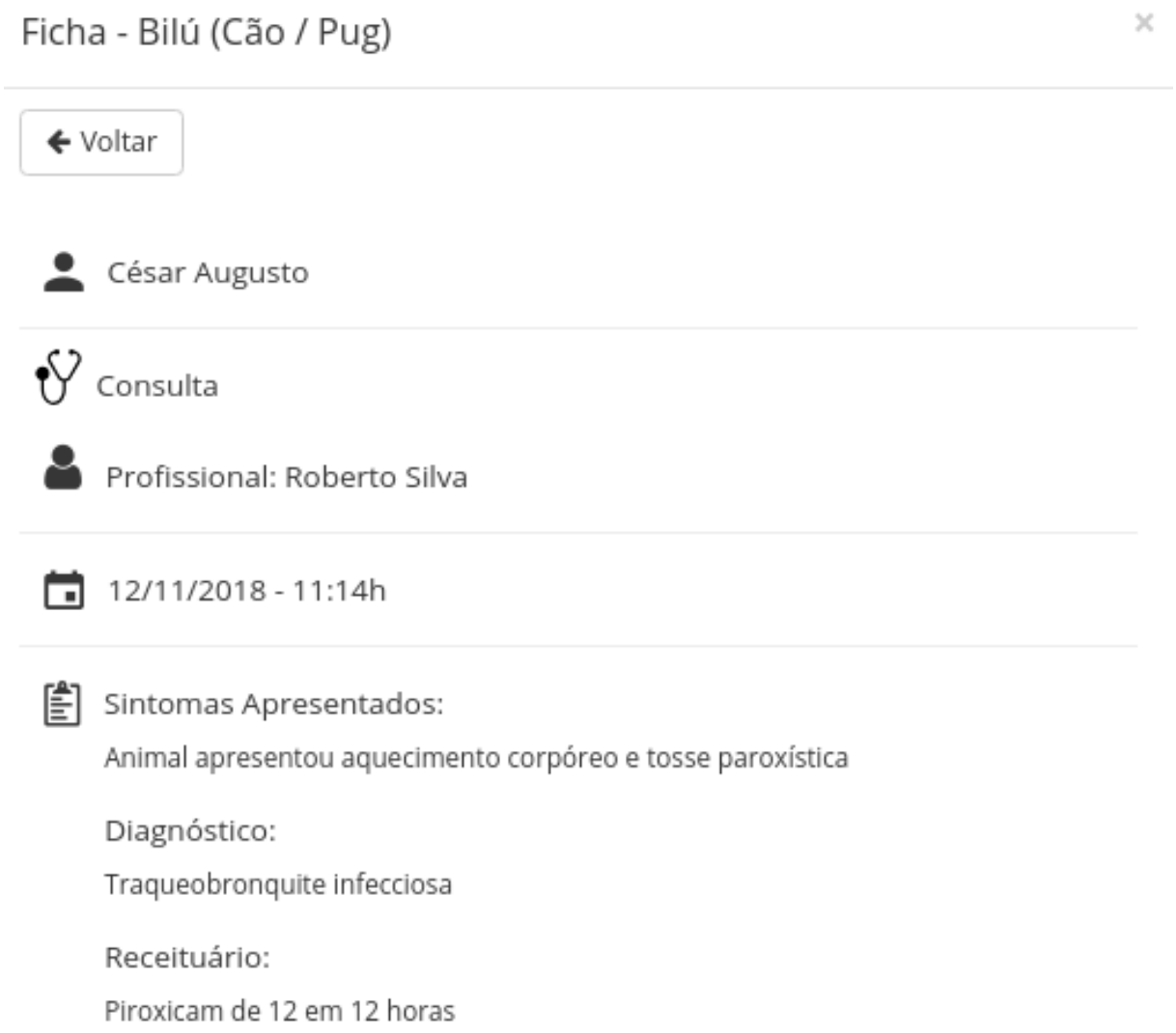
Após o cadastro do atendimento, todas as informações clínicas de cada animal ficam centralizadas em sua ficha eletrônica de maneira cronológica de modo que, com um simples clique, o veterinário consegue acessar os detalhes de cada procedimento.

Figura 2.10: Caixa de diálogo onde reside a ficha do animal.



Fonte: elaborado pelo autor.

Figura 2.11: Detalhes mostrados após o clique no link *Ver Detalhes* do procedimento desejado na ficha do animal.



Fonte: elaborado pelo autor.

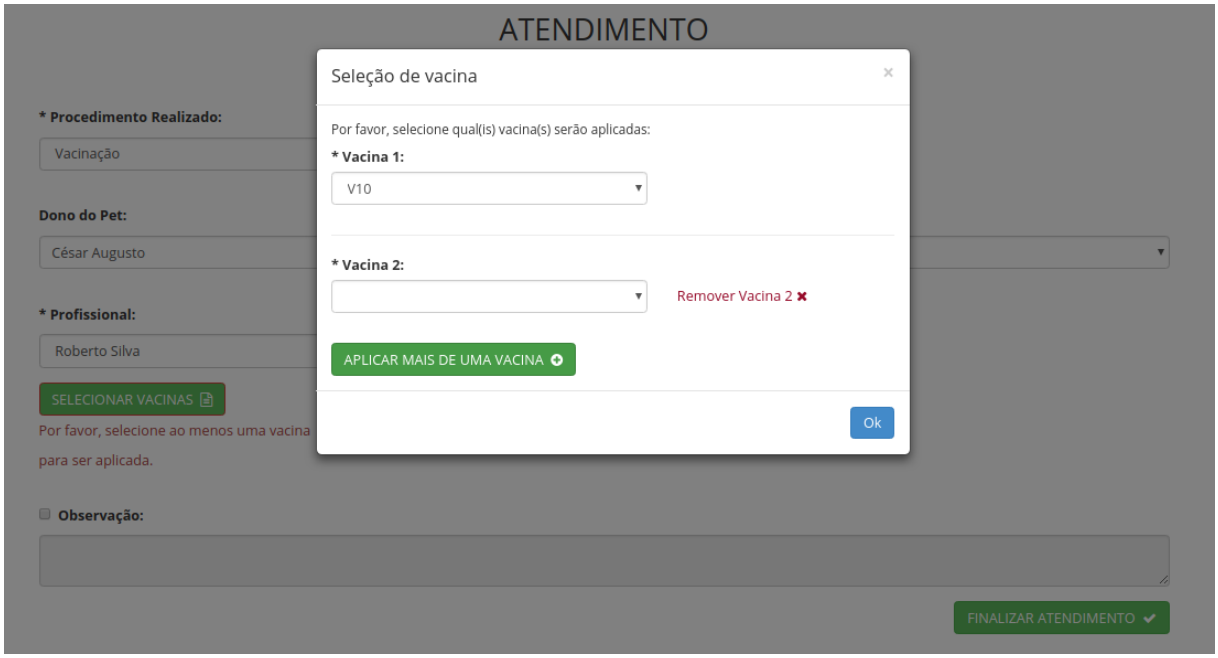
2.7 GESTÃO DE CADERNETA DE VACINAÇÕES

Um dos procedimentos mais importantes para o bem-estar dos animais é a aplicação de vacinas na periodicidade correta. Tendo isso em mente, foi crucial desenvolver um setor da aplicação para o cadastro de vacinações e para a consulta da caderneta de vacinações de cada animal cadastrado.

O primeiro passo é cadastrar um procedimento de vacinação e selecionar as vacinas que serão aplicadas; em seguida, já é possível ir para a tela de consulta de *pets*

e visualizar a caderneta desejada.

Figura 2.12: Tela de realização de atendimento sendo utilizada para cadastrar uma vacinação.



Fonte: elaborado pelo autor.

Figura 2.13: Caixa de diálogo referente à caderneta de vacinações do animal.

Fonte: elaborado pelo autor.

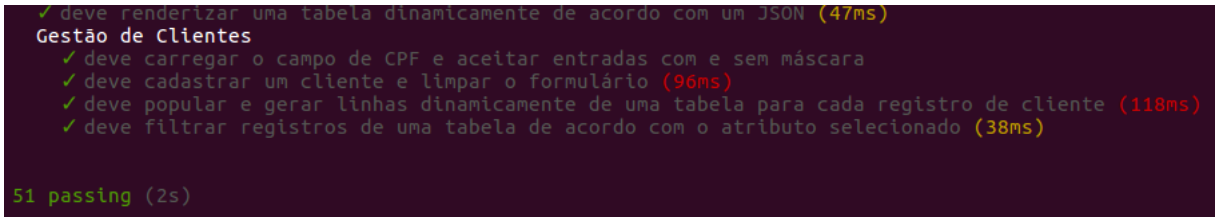
2.8 IMPLEMENTAÇÃO DE TESTES UNITÁRIOS

A importância dos testes unitários é imprescindível para a construção de *software* com qualidade e para que alterações no código fonte que gerem erros sejam facil-

mente identificadas. [2]

Tendo isso em mente, foram sendo construídos testes unitários em paralelo com o desenvolvimento de cada *feature*, abrangendo desde testes de interface até testes comportamentais para comprovar que os componentes seguiam o que foi definido nas regras de negócio.

Figura 2.14: Resultado da execução de 51 testes unitários bem-sucedidos.



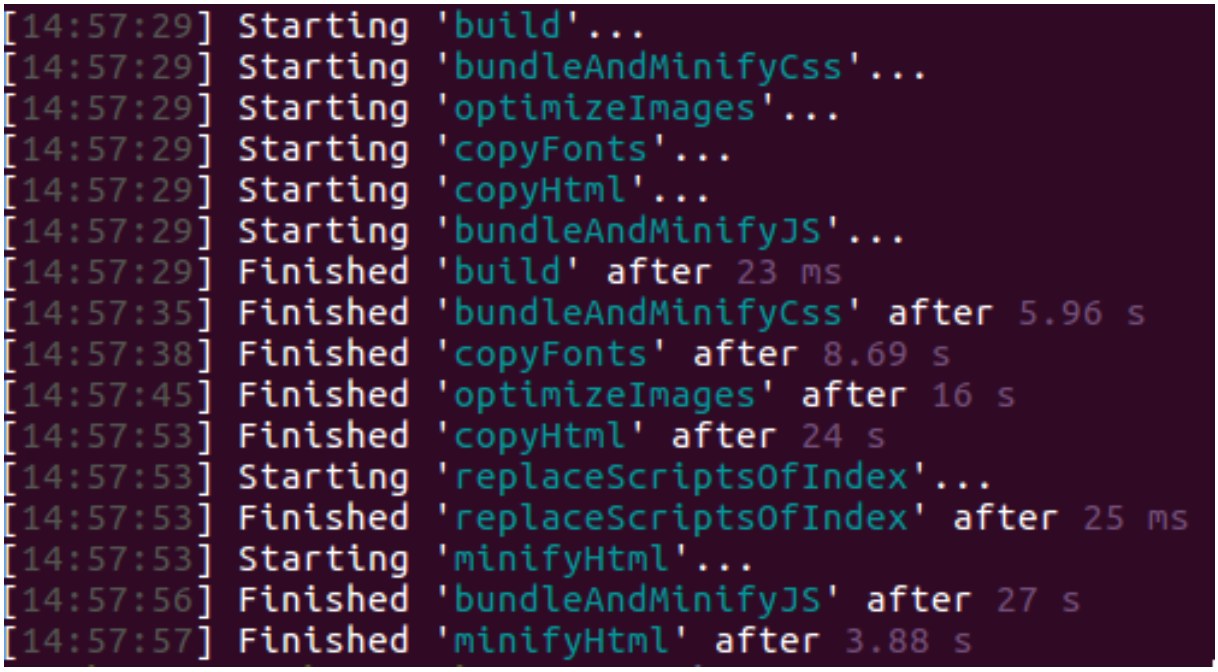
Fonte: elaborado pelo autor.

2.9 AUTOMATIZAÇÃO DE BUILD

Para que o código desenvolvido pudesse passar do modo de desenvolvimento para o modo de produção no *front-end*, foi necessário criar um processo de *build* definido como um conjunto de pequenas tarefas que são melhor explicadas na seção Fundamentação. O *software* que ajudou neste processo foi o *task runner* Gulp[8], através do qual foram criados mini processos sequenciais para realizar o *bundling*, otimização de imagens e minificação de arquivos .css, .html e .js.

[8]Gulp. Disponível em: <https://gulpjs.com/>

Figura 2.15: Etapas da execução do processo de *build*.



Fonte: elaborado pelo autor.

3 FUNDAMENTAÇÃO

Nesta seção, é explanada a fundamentação teórica e os conceitos acadêmicos necessários para a realização do projeto. Adicionalmente, são mostradas as correlações entre as atividades desenvolvidas e os conceitos abordados ao longo das disciplinas da graduação.

3.1 DISCIPLINAS CORRELACIONADAS

As disciplinas cursadas durante a graduação tiveram grande contribuição nas atividades realizadas. A Tabela 3.1 apresenta as disciplinas cursadas pelo aluno e que foram necessárias para o desenvolvimento das atividades.

Processamento da Informação	• Linguagens de programação • Modularização de códigos • Lógica de programação
Lógica Básica	• Operadores lógicos
Programação Orientada a Objetos	• Paradigma de programação orientada a objetos • Padrões de projeto de *software* e suas aplicações
Algoritmos e Estrutura de Dados	• Aplicações de estruturas de dados • Busca de elementos • Ordenação de elementos
Análise de Algoritmos	• Medidas de complexidade • Complexidade de tempo
Engenharia de Software	• Planejamento • Modelagem e especificação de requisitos • Gerência de projeto • Modelo Entidade-Relacionamento • Modelos de processos de desenvolvimento
Paradigmas de Programação	• Padrão declarativo • Programação funcional
Banco de Dados	• Banco de dados relacional • Linguagem SQL • Diagrama Entidade-Relacionamento • Normalização
Sistemas Distribuídos	• Servidores remotos • Modelos arquiteturais • REST API
Interface Humano-Máquina	• Heurísticas de Nielsen • Responsividade • Acessibilidade

Tabela 3.1: Disciplinas que contribuíram para a realização do projeto.

3.2 PLANEJAMENTO

Desenvolver um planejamento de entrega uma vez que a descrição formal do problema já foi descoberta, é algo complexo cognitivamente. É necessário um entendimento claro dos objetivos do planejamento, das restrições e também das preferências e da priorização de *features* por parte dos clientes. Apesar dessas informações serem incertas e em constante mudança, isso não significa que o planejamento deva ser feito com base na intuição. [3]

Tendo isso e os conceitos abordados na disciplina de Engenharia de Software como *guidelines*, foi decidido pela utilização de um modelo de desenvolvimento incremental para que partes do sistema fossem entregues antecipadamente. Com isso, foi possível receber um *feedback* rápido, atender primeiro as demandas prioritárias, além

de cada entrega consertar defeitos descobertos em versões entregues anteriormente.

3.3 METODOLOGIAS DE GERENCIAMENTO

Dado o curto espaço de tempo para entregar uma solução completa, seria mais do que necessário utilizar algum tipo de metodologia ágil. De acordo com um dos Princípios do Manifesto Ágil [4], a prioridade máxima deve ser satisfazer o cliente através de uma entrega contínua e antecipada de *software* de qualidade. Outro princípio é o de que, em intervalos regulares, deve-se refletir em como tornar o processo mais efetivo, ajustando o fluxo de trabalho conforme necessário.

Deste modo, foi escolhida a metodologia ágil Scrum, uma das mais populares e amplamente difundida em comunidades de tecnologia, além de ter sido explanada em detalhes na disciplina de Engenharia de Software. Com ela, o primeiro princípio citado acima é satisfeito já que pequenas *features* são agrupadas em *sprints* (iterações do processo de modo que, ao cabo de cada uma, seja entregue um pedaço funcional do *software*). Atendendo ao outro princípio citado, uma vez por semana é feita uma reunião para discutir sobre como o fluxo de trabalho poderia ser melhorado, além de também ser analisado o gráfico de *burndown* que mensura o desempenho do profissional de acordo com o número de horas resolvidas e o número de horas restantes para que a *sprint* seja concluída.

Foi em uma das reuniões que surgiu a ideia de utilizar o Kanban, que é baseado na ideia de que a quantidade de *Work In Progress* (WIP) deve ser limitada e portanto algo novo só deve ser começado depois que uma certa quantidade de trabalho que estava em progresso já tiver sido concluída. Dessa forma, o "Kanban é uma abordagem para trazer mudança para o ciclo de desenvolvimento do *software* e para a metodologia de gerenciamento do projeto". [5]

É válido ressaltar que o *Scrum Master* - uma espécie de gerente de projetos, mas com responsabilidade pautada pelo escopo do Scrum - foi o supervisor de estágio, responsável por definir no início da semana as atividades a serem desempenhadas e

a medida de esforço necessária para cada uma.

Deste modo, o Scrum e o Kanban foram adotados em conjunto e implantados por meio da ferramenta *online* Trello (adicionalmente com um plugin para Scrum) conforme a Figura 2.1, onde o quadro representa uma *sprint* e onde cada lista foi criada como um estágio do Kanban: “To do”, “In Progress” e “Done”. O limite de WIP adotado foi de 12 horas, inviabilizando a transferência de cartões para o *status* “In Progress” no fluxo de trabalho caso a soma de horas totais ultrapassasse o limite acordado. Adicionalmente, também foi criada uma lista intitulada “Backlog”, onde são colocadas as tarefas que ainda não foram discutidas e mensuradas, porém sobre as quais já se têm conhecimento.

3.4 ARMAZENAMENTO DE DADOS

Para efetuar o armazenamento dos dados, optou-se por utilizar um banco de dados relacional, utilizando para isso o MySQL, em partes devido a sua popularidade dentre os sistemas de gerenciamento de banco de dados gratuitos. Através dos conceitos vistos na disciplina de Banco de Dados, logo que as regras de negócio foram compreendidas já foi possível criar o banco, as tabelas e as chaves para representar a cardinalidade mais adequada utilizando a linguagem SQL.

Os motivos da escolha de um modelo relacional ao invés de um banco de dados NoSQL foram que, primeiramente, os dados se encaixavam muito bem com a modelagem relacional. Um ponto que poderia ser questionado é quanto à escalabilidade, já que Mohamed, Altrafi e Ismail [6] afirmam que bancos relacionais não escalam tão bem quanto bancos NoSQL; contudo, como a aplicação estaria alocada em um único servidor e a estimativa inicial do volume de dados e da base de usuários não era muito alta, o MySQL se apresentou como uma tecnologia suficiente para o momento.

Por já terem sido realizadas pesquisas com veterinários e profissionais que trabalham em estabelecimentos veterinários, a modelagem aconteceu de forma tranquila e não houveram alterações significativas em sua estrutura ao longo do processo.

3.5 DESENVOLVIMENTO WEB

Nesta subseção são apresentadas todas as tecnologias e os questionamentos que circundaram o desenvolvimento *web*, desde o *front-end* que é o ponto de partida das interações do usuário até o *back-end* que atua no servidor de forma transparente para o usuário e é responsável por manipular diretamente os dados armazenados.

3.5.1 HTML

O *HyperText Markup Language* (HTML) é a linguagem de marcação padrão para carregar o conteúdo (como textos, imagens, etc) de páginas *Web*. Ao contrário das linguagens de programação que são compiladas, o HTML é interpretado pelo navegador. Além disso, funciona como um sistema de *tags* para definir cabeçalhos, parágrafos, *hyperlinks*, entre outras funcionalidades.

Todas as páginas *web* foram escritas utilizando HTML, mais especificamente HTML5, que é a especificação mais nova e que possui recursos como armazenamento local, *autofocus* e dezenas de novos tipos de input. [7]

3.5.2 CSS, RESPONSIVIDADE e BOOTSTRAP

Para realizar a estilização e definir como os elementos HTML apareceriam na tela tanto do ponto de vista do *design* como do *layout*, foram utilizadas as *Cascading Style Sheets* (CSS).

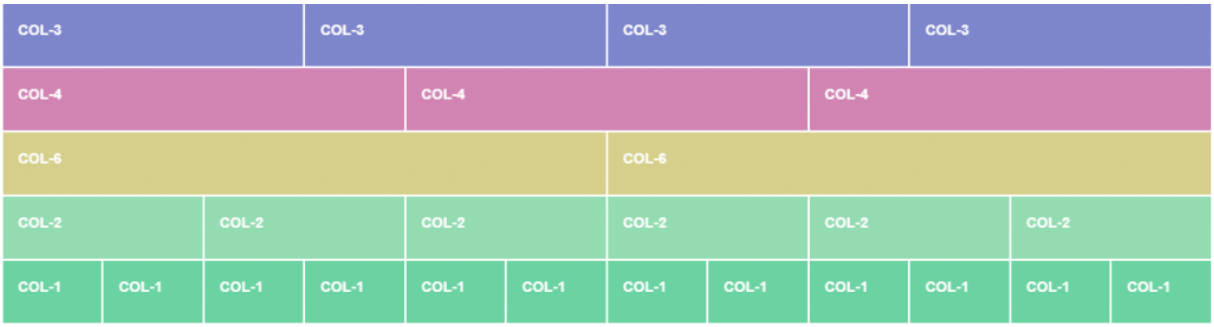
Nesse contexto, o CSS é muito útil pois remove do HTML a responsabilidade de definir como os elementos devem ser formatados, transferindo uma mistura confusa de *tags* HTML (como por exemplo a *<b>*) e atributos (como por exemplo o *color*) para arquivos CSS organizados que descrevem separadamente como cada pedaço da página deve ser estilizado e posicionado.

De acordo com estatísticas coletadas pela empresa StatCounter no período de outubro de 2018 [8], a navegação em dispositivos *mobile* é feita em 48.2% das ve-

zes, enquanto que a navegação por meio de *desktops* está logo abaixo com 47.78%. Com base nessas informações, na atualidade é imprescindível que o desenvolvimento de qualquer sistema *web* de qualidade seja feito pensando em como o *layout* irá se adaptar em diferentes tamanhos de tela.

Para tratar a questão da responsividade foi utilizado o *framework* Bootstrap, que possui uma série de classes facilitadoras e um sistema de *grid* responsiva análogo a um conjunto de linhas e colunas, ajudando em várias tarefas recorrentes do desenvolvimento, além de possuir uma estilização predefinida de elementos muito agradável.

Figura 3.1: Sistema de *Grid* do Bootstrap.



Fonte: Gridbox Blog [9]

Para especificar o tamanho da coluna nas resoluções *Extra Small* (celulares de até 767px), *Small* (tablets a partir de 768px), *Medium* (*desktops* a partir de 992px) e *Large* (*desktops* a partir de 1200px) basta utilizar, respectivamente, as classes CSS *col-xs-x*, *col-sm-x*, *col-md-x* e *col-lg-x*, onde *x* representa o tamanho da coluna (de 1 a 12).

É válido ressaltar que o desenvolvimento não só foi pensado na questão da responsividade, como também na usabilidade e na acessibilidade, aspectos respectivamente muito importantes por si só; além disso, esses aspectos ajudam no mantenimento do usuário nas páginas e também para ranquear bem em mecanismos de pesquisa, conforme estudado na disciplina de Interface Humano-Máquina.

3.5.3 JAVASCRIPT

O JavaScript é a linguagem padrão da *Web* e é responsável pela interatividade de uma página HTML. É nele que os conceitos de programação vistos nas disciplinas

Processamento da Informação, Lógica Básica e Algoritmos e Estruturas de Dados foram utilizados, abrangendo desde conhecimentos básicos como operadores lógicos e lógica de programação, até conceitos mais aprofundados como uso de funções de busca, ordenação e o entendimento das estruturas de dados presentes como a própria pilha de execução das funções JavaScript e a fila que permite o enfileiramento de respostas dos códigos assíncronos.

Cada trecho de código foi feito também sempre levando em conta a complexidade de tempo, aspecto importante visto na disciplina Análise de Algoritmos e que, se não feito de modo responsável, pode aumentar muito o tempo de carregamento das páginas e trazer resultados ruins como pouca retenção dos usuários e punições no ranqueamento em mecanismos de pesquisa.

O JavaScript é uma linguagem orientada a objetos por natureza, conforme visto na disciplina de Programação Orientada a Objetos. Contudo, possui diversas nuances de programação funcional, conceito apresentado na disciplina de Paradigmas de Programação e que não só agiliza o processo de desenvolvimento, como também parece deixar mais claro o entendimento de um dado trecho de código. Por ser uma linguagem de grande potencial e amplamente versátil, o JavaScript foi utilizado tanto no *front-end* como também no *back-end*.

3.5.4 ANGULAR

Segundo Gamma et al. [10], ter um código flexível e reutilizável é algo essencial e imprescindível para garantir a qualidade do *software*. Foi pensando nisso que o Angular foi escolhido como *framework* para facilitar tarefas do *front-end*, pois permite que trechos de código que aparecem em muitos lugares possam ser escritos em componentes enxutos e reutilizáveis. Outras opções de *frameworks* e bibliotecas também se encaixavam nos requisitos, porém o Angular foi escolhido devido à empresa contratante já ter uma experiência prévia com o mesmo, o que facilitou a supervisão.

Além disso, funcionalidades que não manipulam diretamente o *Document Object*

Model (DOM) podem ser escritas em *services* que são injetados como dependências de quaisquer componentes que precisem utilizá-lo, tornando o código modular e reutilizável. Tal reutilização de código é ainda mais útil no *front-end* pois menos código implica menos bytes baixados pelos usuários, o que torna o carregamento das páginas mais rápido e traz consigo menor *bounce rate* e melhor ranqueamento em mecanismos de pesquisa.

O Angular traz uma série de benefícios em relação ao uso de JavaScript puro, estando entre os pontos de destaque as expressões Angular (que transferem para o HTML o valor das variáveis definidas no JavaScript), *pipes* (que transformam uma dada entrada em uma *string* formatada como saída) e o suporte de roteamento para criar uma *Single Page Application* (SPA), fazendo com que o conteúdo da página que o usuário está vendo, juntamente à URL, mudem sem que ele sinta o efeito de um recarregamento total da página.

Como exemplo, na Figura 3.2 está apresentado o código HTML com expressões Angular, necessárias para efetuar o *binding* JS/HTML das propriedades *nome* e *cor* de um objeto fictício que representa um *pet*. Também é utilizado o *pipe uppercase* nativo do Angular para transformar o nome do animal em uma *string* com todas as letras maiúsculas.

Figura 3.2: Exemplo do código HTML de uma aplicação Angular que utiliza expressões e *pipe*.

Fonte: elaborado pelo autor.

Deste modo, na Figura 3.3 está apresentado o código JavaScript que define o componente relacionado ao *.html* mostrado anteriormente, evidenciando o potencial do Angular de modularização ao possibilitar a definição de componentes enxutos que seguem o Princípio de Responsabilidade Única, onde não existe mais de um motivo de mudança que possa levar o pedaço de código em questão a ser alterado. [11]

Figura 3.3: Exemplo do código de um componente de uma aplicação Angular.

```
1   import { Component, OnInit } from '@angular/core';
2   import { Pet } from '../pet';
3
4   @Component({
5     selector: 'app-pets',
6     templateUrl: './pets.component.html',
7     styleUrls: ['./pets.component.css']
8   })
9
10  export class PetComponent implements OnInit {
11    meuPet: Pet = {};
12
13    constructor() { }
14
15    ngOnInit() { }
16  }
```

Fonte: elaborado pelo autor.

3.5.5 NODE.JS

Uma aplicação *back-end* é responsável por lidar diretamente com o armazenamento, recuperação e manipulação dos dados, além de lidar diretamente com as regras de negócio e com o cálculo dos resultados a serem enviados para o *front-end*.

De modo a desenvolver as APIs alocadas no servidor foi escolhido o Node.js, um dos interpretadores de códigos JavaScript mais famosos e que vêm ganhando muita popularidade entre as escolhas de tecnologia para o desenvolvimento de aplicações

back-end. O interessante do Node.js em relação ao JavaScript utilizado no *front-end*, é que o Node.js vem com um conjunto de facilitadores que permitem acesso ao sistema de arquivos, requisições http, streams etc.

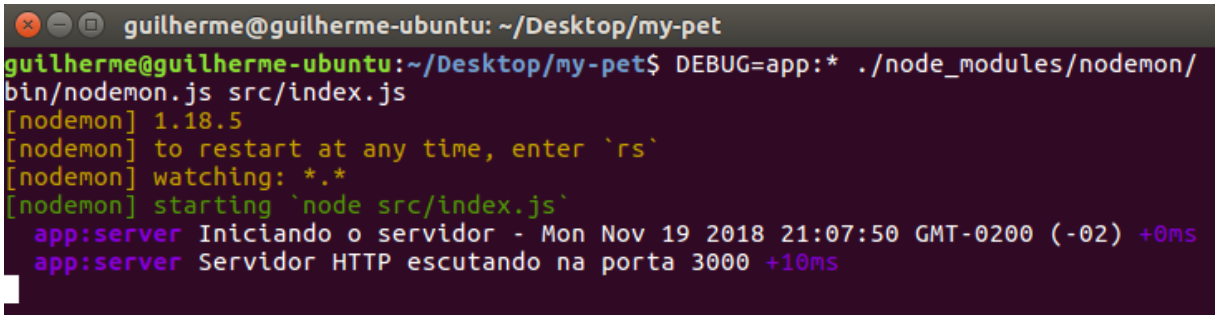
Nas Figuras 3.4, 3.5 e 3.6 é mostrado um exemplo de criação de um servidor Node.js que escuta por requisições no caminho raíz local e devolve como resposta um texto de apresentação *"Exemplo de Aplicação NodeJS!"*. Como pode ser observado, o Node permite que o código necessário para o *setup* inicial de uma aplicação *back-end* seja escrito em poucas linhas.

Figura 3.4: Exemplo do código JavaScript de uma aplicação Node.js.

```
//server.js
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Exemplo de Aplicação NodeJS!');
});

module.exports = app;

// ========================================================

//index.js
const http = require('http');
const server = require('./modules/server');
const debug = require('debug')('app:server');

const httpPort = process.env.PORT || 3000;
const httpServer = http.createServer(server);

debug('Iniciando o servidor - ' + new Date());
httpServer.listen(httpPort, '127.0.0.1', () => {
  debug("Servidor HTTP escutando na porta %d", httpPort);
});
```

Fonte: elaborado pelo autor.

Figura 3.5: Iniciando uma aplicação Node.js pelo terminal.



Fonte: elaborado pelo autor.

Figura 3.6: Resultado retornado pela aplicação Node.js ao receber uma requisição GET no caminho raíz *localhost:3000/*.

Fonte: elaborado pelo autor.

Para a construção da aplicação, foi usado o estilo de arquitetura *REpresentational State Transfer* (REST) visto na disciplina de Sistemas Distribuídos, na qual o servidor não grava o estado em que o cliente está, permitindo que o código do *back-end* seja desenvolvido de maneira completamente separada do *front-end*, onde cada requisição ao servidor é tratada de maneira separada e sem precisar ter conhecimento de quais foram as últimas interações do cliente solicitante.

A escolha do Node.js foi pautada em vários fatores, porém se destacaram os seguintes fatos:

- O JavaScript é uma linguagem simples, versátil e eficiente;
- O JavaScript é uma das linguagens mais populares e, portanto, caso outra pessoa precise dar manutenção no código futuramente, a chance é maior do indivíduo conhecer a linguagem pois ela costuma ser o ponto de entrada de um desenvolvedor *web*;
- Devido à linguagem ser a mesma entre *front-end* e *back-end*, é possível reutilizar código entre as partes;

- A construção de um sistema RESTful permite que os dados sejam consumidos de maneira simples por um futuro aplicativo *mobile* sem que sejam necessárias alterações no código.

3.6 TESTES UNITÁRIOS

Caso o *software* não passe por uma etapa de testes antes de sair do ambiente de desenvolvimento para o ambiente de produção, podem ocorrer problemas que comprometam gravemente seu uso, fazendo com que a empresa que o monetiza acabe por ter prejuízos e em alguns casos até ter sua reputação afetada negativamente.

De modo a garantir a estabilidade e confiabilidade do *software*, é imprescindível que o mesmo seja submetido a uma série de testes que visam avaliar se trechos de códigos se comportam bem e produzem o resultado esperado para diferentes entradas. Contudo, é preciso que a escrita dos testes sigam uma metodologia eficiente para evitar problemas comuns como: testes sendo escritos após a codificação completa da *feature* que ele irá testar, o que é propício ao esquecimento de certos detalhes por parte do programador; testes escritos por programadores diferentes daqueles que escreveram o código, que por não entenderem o código profundamente tendem a esquecer testes importantes; se os desenvolvedores dos testes se basearem na documentação e não no código em si, caso ocorra uma simples falha na atualização da documentação o teste pode ficar defasado; entre outros. [2]

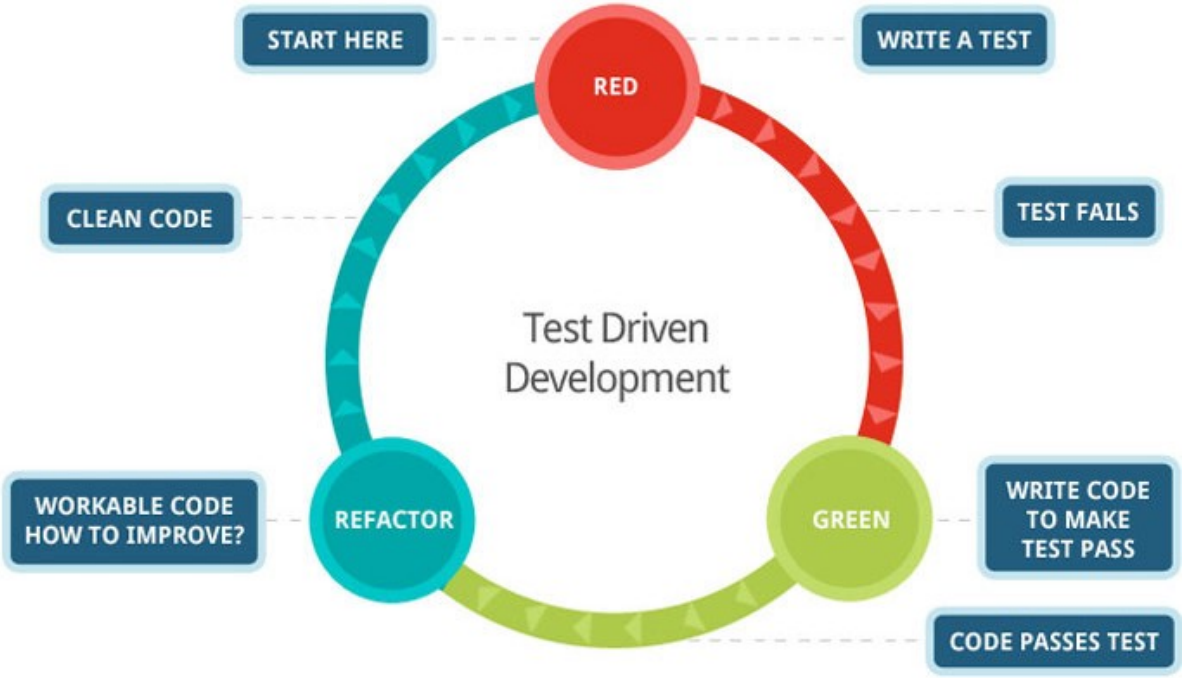
Para evitar esses problemas foi utilizada a metodologia *Test-driven Development*, a qual consiste em um desenvolvimento guiado a testes que segue uma sequência de três passos essenciais feitos pelo mesmo programador que desenvolve a funcionalidade, contornando todos os problemas citados anteriormente. Os três passos são:

1. Escrever um teste com a especificação de algo que o código deve fazer e vê-lo falhar;

2. Fazer a codificação necessária para o teste passar;

3. Com o código funcionando, refatorá-lo e garantir que o teste continua passando.

Figura 3.7: Etapas do Test-driven Development.



Fonte: Noteworthy - The Journal Blog [12]

3.7 AUTOMATIZAÇÃO DE BUILD

Uma das etapas mais cruciais do processo de desenvolvimento é o processo de *build*, que consiste na conversão do código em desenvolvimento onde até então ele só é acessível ao desenvolvedor, para o modo de produção onde ele vai pode ser usufruído por todos os usuários.

Em desenvolvimento *web*, uma série de medidas precisam ser tomadas de modo a garantir que o código esteja otimizado e pronto para entrar em produção, abrangendo tarefas de minificação de arquivos, *bundling*, modificações no código e otimização de imagens. Desse modo, foi utilizado o *task runner* Gulp para garantir que todas essas tarefas seriam sempre executadas e realizadas na ordem correta no momento de transformar o código para o modo de produção, automatizando o processo de *build*.

3.7.1 MINIFICAÇÃO

Nessa etapa, arquivos .css, .html e .js tiveram seus comentários removidos, os nomes das variáveis foram encurtados e todo tipo de redundância foi retirada; ao cabo do processo, os arquivos tiveram seu tamanho reduzido significativamente, diminuindo o número de *bytes* que os usuários têm que baixar ao acessar a página e melhorando a experiência de navegação.

3.7.2 BUNDLING E MODIFICAÇÕES NO CÓDIGO

Um dos principais problemas do protocolo HTTP/1.1 é que os navegadores só podem abrir um número limitado de conexões em paralelo ao mesmo domínio, portanto requisições adicionais acabam tendo que ir e voltar do servidor novamente (processo conhecido como *Round-trip Time*), ocasionando lentidão no carregamento da página.

Para contornar o problema, foi realizado o *bundling* dos arquivos, que consiste basicamente em concatenar todo o código do mesmo tipo em arquivos únicos. Em seguida, o código do arquivo *index.html* (ponto de entrada de toda página *web*) foi modificado de modo a substituir a inclusão dos *scripts* e css antigos pelo caminho dos novos arquivos gerados após a etapa de *bundling*.

3.7.3 OTIMIZAÇÃO DE IMAGENS

As imagens são um dos recursos mais “pesados“ no momento em que os usuários efetuam o carregamento das páginas. De modo a reduzir os impactos do seu carregamento, foram executados algoritmos de compressão para reduzir o tamanho dos arquivos de imagem sem que a perda de qualidade fosse significativamente perceptível.

4 CONSIDERAÇÕES FINAIS

A experiência de desenvolver um projeto relacionado com o curso foi muito positiva, ficando clara a grande importância dos conhecimentos acadêmicos abordados ao longo do curso de Ciência da Computação, ajudando na realização de uma série de tarefas que, por já terem sido abordadas nas disciplinas de graduação, tornaram-se muito menos difíceis de serem executadas.

A realização do presente projeto nos moldes da disciplina de Estágio Supervisionado foi algo muito benéfico, pois a presença de um supervisor da área de Computação fez com que problemas pelos quais o mesmo já tinha passado não acontecessem novamente no decorrer das atividades. Sendo assim, as experiências compartilhadas foram de grande utilidade e, por já possuir domínio sobre o assunto, o supervisor conseguia identificar o que poderia ser melhorado e dar sugestões no andamento das tarefas.

Um dos principais pontos de satisfação ao executar desde a modelagem dos dados até a construção de APIs e interfaces do sistema de gestão veterinário, foi poder ter colocado em prática e aplicado a base acadêmica adquirida nos anos de universidade em uma solução de mercado, gerando a boa sensação de produzir algo que será usufruído por outras pessoas e que as ajudará significativamente no dia a dia. Por meio das atividades realizadas, foi dado um grande passo na informatização de estabelecimentos veterinários que até então fazem o controle de clientes e dos prontuários de seus animais em papel ou em sistemas ruins, contribuindo ainda mais para o crescimento do mercado *pet* na medida em que os sistemas se modernizam e o atendimento é otimizado.

A experiência com o fluxo de trabalho e com as metodologias de gerenciamento de projetos foi muito boa pois o Kanban em conjunto com o Scrum harmonizou a organização das atividades com o acompanhamento do desempenho. Ademais, embora na primeira semana ainda houvesse uma certa dificuldade por não se ter tanta fluência com todas as tecnologias, pode-se dizer que a escolha das mesmas foi assertiva, seu aprendizado foi rápido, os resultados obtidos foram satisfatórios e portanto não foi

necessária a mudança de nenhuma delas.

A execução deste primeiro conjunto de tarefas trouxe uma maior maturidade e confiança para atuar no mercado de trabalho, contribuindo para a formação do aluno na medida em que ajudou a fixar os conceitos acadêmicos vistos ao longo do curso, e também colaborando para seu desenvolvimento como um profissional de qualidade e como um cientista da computação completo.

O fato das dificuldades terem sido rapidamente superadas se deveram em grande parte ao treinamento autodidata estimulado pela universidade desde o princípio da formação acadêmica, uma característica pessoal e profissional essencial e que deve ser mantida no desenvolvimento de novos trabalhos para a Felsan. Entre os trabalhos futuros para completar o sistema de gestão, encontram-se o desenvolvimento de novos setores do sistema para efetuar o gerenciamento de internações, o controle de hospedagens, além de melhorias como configurar os testes unitários para executarem em múltiplos navegadores e otimizações no carregamento das páginas.

5 REFERÊNCIAS

[1] CANAL DO EMPRESÁRIO INFOMONEY. Sem crise: mercado de pets no Brasil é o terceiro do mundo em faturamento. Disponível em: <https://www.infomoney.com.br/negocios/canal-do-empresario/noticia/7375940/sem-crise-mercado-pets-brasil-terceiro-mundo-faturamento>. Acesso em: 12 nov. 2018, 23:05.

[2] ASTELS, D. ***Test driven development****: A practical guide*. Prentice Hall Professional Technical Reference, 2003.

[3] RUHE, G.; SALIU, M. O. **The art and science of software release planning**. *IEEE software*, v. 22, n. 6, p. 47–53, 2005.

[4] MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT. Principles Behind The Agile Manifesto. Disponível em: <http://agilemanifesto.org/principles.html>. Acesso em: 16 nov. 2018, 23:15.

[5] KNIBERG, H.; SKARIN, M. ***Kanban and Scrum****: making the most of both*. Lulu.com, 2010.

[6] MOHAMED, M. A.; ALTRAFI, O. G.; ISMAIL, M. O. **Relational vs. nosql databases**: A survey. *International Journal of Computer and Information Technology*, v. 3, n. 03, p. 598–601, 2014.

[7] PILGRIM, M. ***HTML5****: up and running: dive into the future of web development*. O'Reilly Media, 2010.

[8] ESTATÍSTICAS DA STATCOUNTER. Desktop vs mobile vs tablet market share worldwide. Disponível em: <http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet>. Acesso em: 18 nov. 2018, 23:30.

[9] ANAND, R. Bootstrap 4 vs Foundation 6 Grid System. Disponível em: <http://blog.gridbox.io/2018/01/08/bootstrap-4-vs-foundation-6-grid-system/>. Acesso em: 17 nov. 2018, 23:55.

[10] VLISSIDES, J.; HELM, R.; JOHNSON, R.; GAMMA, E. **Design patterns**: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, v. 49, n. 120, p. 11, 1995.

[11] MARTIN, R. C. ***Agile software development****: principles, patterns, and practices*. Prentice Hall, 2002.

[12] DUBEY, V. Test Driven Development — Understanding the business better. Disponível em: <https://blog.usejournal.com/test-driven-development-understanding-the-business-better-9c4cae4cb990>. Acesso em: 17 nov. 2018, 23:30.