COLEIRA INTELIGENTE:
Desenvolvimento de um Protótipo para Monitoramento de Sinais Vitais de Cães e Gatos

Ana Laura Alvino de Souza
Graduanda em Engenharia de Software – Uni-FACEF
any.alvino@gmail.com

Saulo Sousa Andrade
Graduando em Engenharia de Software – Uni-FACEF
andradesaulo.developer@gmail.com

Geraldo Henrique Neto
Mestre em Ciências com Ênfase em Informática Médica - FMRP-USP
geraldohenrique@alumni.usp.br

Carlos Eduardo de França Roland
Mestre em Desenvolvimento Regional – Uni-FACEF
roland@facef.br

Resumo
O mercado *pet* está cada vez mais aquecido, porém, o setor de saúde animal não utiliza efetivamente tudo que a tecnologia da informação oferece, ficando para trás nos avanços tecnológicos comparados com o setor de saúde humana. Assim, a ideia deste trabalho é apresentar o desenvolvimento de um protótipo de sistema que monitora sinais vitais de cães e gatos, através de sensores que aferem frequência cardíaca e saturação de oxigênio, e que apresenta graficamente o *status* do *pet* por meio de um aplicativo, abordando desta forma o conteúdo programático disciplinar do curso de Engenharia de Software. Com o desenvolvimento deste protótipo foi possível conhecer a viabilidade de uma solução específica. Um sistema de monitoramento pode facilitar atividades de tutores e veterinários e prolongar a vida dos animais. Procurou-se fazer uma revisão teórica a fim de embasar o trabalho, além de realizar o processo da elicitação e documentação de requisitos para o desenvolvimento do protótipo. Estudou-se a saúde de *pets* através de pesquisa e entrevista com um veterinário. Na criação do aplicativo utilizou-se as tecnologias Dart, Flutter e SQLite, e no circuito elétrico foram utilizados sensores adequados e as ferramentas Fritzing para esboçar o circuito e Arduino IDE para codificá-lo. Ao final do projeto foi constatado que é possível de fato, implementar um sistema capaz de registrar os sinais vitais do *pet* e apresentá-los ao usuário. Os autores se preocuparam em criar um protótipo de qualidade que demonstrasse que este tipo de solução é interessante para beneficiar a saúde dos *pets* e simplificar o dia a dia de tutores e veterinários.

Palavras-chave: Mercado *pet*. Saúde animal. Protótipos.

Abstract
The Brazilian pet industry has been growing quite fast, however, the health sector hasn't been leveraging all the information technology can offer, staying behind in the technological advances compared to the human health sector. Thus, the idea of this

essay is to present the development of a system prototype which tracks vital signs of dogs and cats, through sensors that measure the heart rate and oxygen saturation; and graphically shows the pet status, through a mobile app; therefore, encompassing the Software Engineering course syllabus. With the development of this prototype it was possible to evaluate the viability of a specific solution. A monitoring system may facilitate activities of pet owners and veterinarians, and extend the animals' life. A literature review was made to support this essay, as well as gathering and documenting requirements for the prototype's development. The authors studied the health of pets by researching and making an interview with a veterinarian. Dart, Flutter and SQLite technologies were used in order to create the app. In addition to the need of using the appropriate sensors for the electric circuit, it was necessary as well to use the Fritzing tool to design the circuit, and the Arduino IDE tool to code it. At the end of the project, it was observed that it is possible indeed to implement a system capable of recording the vital signs of pets and showing them to the user. The authors focused on creating a quality prototype that could demonstrate that this type of solution is interesting to benefit the health of pets and to simplify the day-to-day of pet owners and veterinarians.

Keywords: *Pet industry. Animal health. Prototypes.*

1 Introdução

A relação do homem com outros animais existe há muito tempo. A relação dos homens com os cães, por exemplo, deixou de ser uma relação de concorrência por alimento há pelo menos 30 mil anos (CORREIO BRAZILIENSE, 2011). A domesticação dos cães nesse período trouxe benefícios, como alertas de perigos iminentes e proteção contra outros animais selvagens.

A diminuição do tamanho das famílias e o ambiente pós-moderno tiraram o animal do papel apenas de animal de estimação para o papel de membro da família. Muitos demonstram afeto pelo animal considerando-o até mesmo como um filho: o caráter afetivo das relações que eram totalmente preenchidas com filhos tem sido trespassado para cães e gatos (SOARES, 2009).

De acordo com o IBGE (2020), o Brasil possuía, em 2019, 39,4 milhões de domicílios com algum gato ou cachorro. O crescente aumento no setor voltado ao *pet* colocou o Brasil em segundo lugar de maior mercado de produtos *pets*, correspondendo a 6,4% de participação global (MARTINS, 2021), abrindo novos caminhos, negócios e produtos voltados a esse setor.

Em um levantamento feito pela SNA (Sociedade Nacional da Agricultura) em 2017, foram entrevistadas 1406 pessoas, que foram divididas em dois grupos: 796 delas para saber se possuíam animal de estimação e 610 para identificar suas características. Nos dados da pesquisa foi constatado que 33% dos entrevistados gostariam de adquirir um plano de saúde para seu *pet*, mas não o fazem por condições financeiras (SNA, 2017).

Roque Pellizzaro Junior, Presidente do SPC (Serviço de Proteção ao Crédito), em entrevista para a SNA em 2017, afirmou que "O tratamento humanizado dos *pets* abre inúmeras oportunidades de negócios e evidencia a força de um mercado bilionário que deve se diferenciar ainda mais nos próximos anos" (SNA, 2017).

Ao observar o grande amor e afeto dos brasileiros pelos seus *pets*, e não deixando de se atentar a este mercado extremamente aquecido, percebeu-se que

seria interessante a criação de uma solução focada na saúde dos animais para que estes vivessem mais.

Os autores tiveram a ideia de uma solução capaz de dar mais controle aos tutores e veterinários na saúde de seus animais, uma solução que monitorasse e apresentasse os sinais vitais, como frequência cardíaca e saturação de oxigênio dos *pets*. Porém, para a criação de uma solução de alto nível que seja segura, performática e que atenda as necessidades do mercado, são necessários recursos, como tempo, dinheiro e pessoas, que fogem do escopo de um trabalho acadêmico. Optou-se, portanto, pela criação de um protótipo para um sistema deste tipo.

Ao decorrer deste trabalho, será apresentado como foram utilizadas teorias e técnicas da engenharia de *software* aprendidas durante o curso para desenvolver um protótipo de monitoramento de animais de estimação. O protótipo analisará, através de sensores, os sinais vitais, e os salvará em um aplicativo no qual o usuário poderá acompanhar o *status* do *pet*.

Durante o projeto, foi necessário, mais especificamente, realizar o levantamento e análise de requisitos; a modelagem e o projeto arquitetural do sistema; a prototipação das telas; a modelagem e implementação do banco de dados; e o desenvolvimento do sistema em si.

Com a criação do protótipo foi possível constatar que uma solução desta natureza é viável de fato. Um sistema como esse pode facilitar a vida dos tutores e veterinários, e prolongar a vida dos animais. Um monitoramento diário pode contribuir para que tutores amparem e protejam seus *pets*, tomando medidas precoces antes que algum problema de saúde apareça repentinamente. Este projeto visa também contribuir para o progresso da tecnologia da informação no setor.

Para desenvolver o projeto foi realizado primeiramente uma revisão teórica acerca dos métodos e ferramentas da engenharia de software, utilizando livros, tais como *Engenharia de Software*, de Ian Sommerville, e *Engenharia de Software: Conceitos e Práticas*, de Raul Wazlawick.

O levantamento dos requisitos foi feito por meio de discussões entre os autores e uma entrevista com um profissional da medicina veterinária. Após o levantamento, os requisitos foram analisados e documentados.

Foram utilizados diagramas para a modelagem do sistema. Estes diagramas foram produzidos com o *software* Draw.io.

Uma vez que coleiras de animais de estimação são leves e pequenas, decidiu-se utilizar as menores versões possíveis dos componentes de *hardware* necessários para o circuito elétrico do protótipo.

Para facilitar a montagem do circuito elétrico, este foi desenhado utilizando o *software* Fritzing. A programação do circuito foi feita utilizando IDE (*Integrated Development Environment*) e linguagem de programação próprias de placas Arduino.

Foi realizada também a modelagem do banco de dados. O modelo conceitual utilizou o programa Draw.io. Já o modelo lógico utilizou a ferramenta DBDesigner. O banco de dados foi implementado com o SGBD (Sistema de Gerenciamento de Banco de Dados) SQLite.

Para o aplicativo *mobile*, foi feita a prototipação das telas por meio do *software* Figma. O aplicativo foi desenvolvido por meio da linguagem de programação Dart e do SDK (*Software Development Kit)* Flutter.

2 Revisão Teórica

2.1 Engenharia de Software

De acordo com Sommerville (2019), o mundo atual não consegue se manter sem a utilização de *softwares*. Pressman e Maxim (2020), também se valem deste pensamento ao dizer que o *software* é uma tecnologia indispensável para as áreas de negócios, ciência e engenharia. As pessoas apostam o emprego, o conforto, a segurança, o entretenimento, as decisões e as próprias vidas no *software* computacional.

Sommerville (2019, p. 3) afirma que “sistemas de *software* são abstratos e intangíveis. [...] Não há limites naturais para o potencial do *software*”. Segundo Pressman e Maxim (2020), *softwares* não se desgastam. Diferentemente do *hardware*, o *software* não é suscetível às condições físicas do ambiente. Porém, como bem diz Sommerville (2019), devido a esta falta de limites físicos, sistemas de *software* podem se tornar extremamente complexos, difíceis de entender e caros de se alterar em pouco tempo.

Percebendo a importância e complexidade do *software*, é possível chegar a uma grande verdade, atestada por Pressman e Maxim (2020): que o *software* em todas suas formas e em todos seus domínios de aplicação deve ser engenhado, ou seja, deve haver uma engenharia de *software*.

Antes de nos aprofundarmos no conceito de engenharia de *software*, é razoável definir o que é *software*. *Software*, segundo Pressman e Maxim (2020), poderia ser definido como instruções ou programas de computador que quando executados entregam funcionalidades, funções e performance desejadas. Contudo a definição de *software* consegue ser ainda mais abrangente. Sommerville (2019) define que no contexto da engenharia de *software*, o *software* é um programa computacional juntamente com toda sua documentação, bibliotecas, *sites* de suporte e configurações associadas necessárias para que ele seja útil.

Agora que o *software* já foi definido, resta saber o que seria mais especificamente a engenharia de *software*. De acordo com Sommerville (2019, p. 7), “a engenharia de *software* é uma disciplina relacionada a todos os aspectos da produção de *software*, desde os estágios iniciais da especificação, até a manutenção depois que o sistema passa a ser usado”.

Segundo Pressman e Maxim (2020), a engenharia de *software* é uma tecnologia feita de camadas. Estas camadas, ilustradas na Figura 1 abaixo, são as seguintes: foco na qualidade, processo, métodos e ferramentas.

Figura 1 - As camadas da engenharia de software

Fonte: PRESSMAN; MAXIN, 2020, p. 9

Começando pela camada inferior, Pressman e Maxim (2020) explicam que qualquer abordagem de engenharia, incluindo a engenharia de *software*, deve se assentar em um comprometimento organizacional para a qualidade. A qualidade seria, portanto, o terreno que dá suporte a toda engenharia de *software*.

De acordo com os mesmos autores, a camada de processo é a fundação da engenharia de *software*. Um processo é uma coleção de atividades, ações e tarefas que são realizadas para criar um produto. No contexto do *software*, segundo Sommerville (2019), processo é uma sequência de atividades que conduzem à produção de um produto de *software*.

Os métodos de engenharia de *software* fornecem o saber técnico para a construção do *software*. Eles se apoiam em um conjunto de princípios básicos que governam cada área da tecnologia e incluem atividades de modelagem e outras técnicas descritivas. Já as ferramentas dão suporte automatizado ou semi-automatizado para o processo e os métodos (PRESSMAN; MAXIM, 2020).

Neste artigo examinaremos a primeira camada da engenharia de *software*, a qualidade.

2.1.1 Qualidade de Software

Paladini (2012) diz que qualidade é uma palavra de domínio público empregada em todas as áreas. Todos a entendem de diferentes maneiras e uma vez que todos já a conhecem não é possível defini-la intuitivamente de forma precisa. Porém Garvin (1984) tenta condensar as várias definições de qualidade em cinco grandes abordagens:

- A abordagem *transcendental* argumenta que a qualidade é uma propriedade universal que não consegue ser definida de forma precisa; aprendemos a identificá-la somente pela experiência.
- A abordagem de *produto* define qualidade como uma variável precisa e mensurável relacionada aos atributos de um produto, por exemplo, um processador computacional com mais rapidez e durabilidade que outros no mercado terá possivelmente mais qualidade.
- A abordagem de *usuário* relaciona a qualidade com as necessidades e desejos dos indivíduos, sendo que os produtos que melhor satisfazem os objetivos dos usuários são aqueles com maior qualidade.
- A abordagem de *manufatura* define que qualidade é atender às especificações, isto é, se um produto estiver de acordo com as especificações que foram estabelecidas ao projetá-lo, logo este terá qualidade.
- Finalmente, a abordagem de *valor* define qualidade em termos de custos e preços; um produto de qualidade é aquele com preços aceitáveis para os diferentes consumidores.

Qualquer que seja a definição de qualidade adotada, ela poderá englobar uma ou mais dessas abordagens e outras mais. No âmbito do *software*, Pressman e Maxim (2020) definem qualidade como um processo de *software* efetivo aplicado de maneira a criar um produto útil que entrega valor mensurável para aqueles que o produzem e para aqueles que o utilizam.

De acordo com Sommerville (2019) a qualidade de *software* é subjetiva pois ela é amplamente baseada em requisitos não funcionais, ou seja, atributos de qualidade subjetivos tais como usabilidade, confiabilidade e eficiência.

Visto a importância da qualidade, cabe gerenciá-la. Sommerville (2019) diz que a gestão da qualidade garante que os *softwares* desenvolvidos atendam aos objetivos pretendidos. São partes da gestão o controle e a garantia da qualidade.

Garantia de qualidade é a definição de processos e padrões que conduzem a um produto de alta qualidade e a introdução de processos de qualidade no processo de manufatura, ou no caso dos *softwares*, no processo de desenvolvimento. Já o controle de qualidade seria a aplicação destes processos e padrões (SOMMERVILLE, 2019). Além disso, como falam Pressman e Maxim (2020), a garantia de qualidade também consiste de uma série de funções de auditoria e relatórios que avaliam a efetividade e completude das ações de controle de qualidade.

Softwares geram valor quando são de qualidade. Mas para *softwares* serem de qualidade existem custos. Pressman e Maxim (2020) explicam que o custo da qualidade inclui todos os custos envolvidos na busca pela qualidade e os custos derivados da falta dela. O custo da qualidade pode ser dividido em custos associados à prevenção, avaliação e falhas.

Custos de prevenção incluem o custo de atividades de gestão necessárias para planejar e coordenar todas as atividades de controle e garantia da qualidade, o custo de atividades técnicas para desenvolver modelos e requisitos completos, custos de planejamento de testes e o custo de todo treinamento associado a essas atividades.

Custos de avaliação incluem o custo de conduzir revisões técnicas, o custo de coletar dados e mensurá-los, e o custo de testagem e depuração.

Custos relacionados a falhas são aqueles que não existiriam se nenhum erro fosse encontrado antes da entrega do *software* aos consumidores. Custos relacionados a falhas podem ser divididos em custos de falhas internas e de falhas externas. Custos de falhas internas estão envolvidos na detecção de erros antes da entrega do *software*. Custos de falhas externas se relacionam com defeitos encontrados depois que o software já foi entregue.

Para finalizar, Pressman e Maxim (2020) resumem que a qualidade de *software* é alcançada através de métodos de engenharia de *software*, práticas de gerenciamento sólidas, e um controle de qualidade compreensível - tudo sustentado por uma infraestrutura de garantia da qualidade.

2.2 Sistemas Embarcados

Em nosso cotidiano diversos objetos das mais diversas áreas contêm sistemas embarcados embutidos em seus interiores. Micro-ondas, máquinas de lavar, aparelhos de ar-condicionado, câmeras, *smartbands*, termômetros digitais, semáforos, todos estes são exemplos de objetos com sistemas embarcados dentro de si.

Segundo White (2011), um sistema embarcado é um sistema computadorizado construído exclusivamente para o objetivo de sua aplicação. Os objetivos de sistemas embarcados são mais limitados do que os de um computador de uso geral, uma vez que o *software* e *hardware* destes sistemas costumam apresentar restrições.

As restrições dependem do sistema embarcado, por exemplo, em alguns sistemas os *softwares* devem ser determinísticos ou de tempo real, em outros o *software* deve ser tolerante a falhas, e em outros ele necessita interromper suas operações ao primeiro sinal de falha.

White (2011) ainda diz que sistemas embarcados são projetados para tarefas específicas e na maioria das vezes necessitam otimizar seus recursos. Os recursos que podem ser otimizados incluem memória RAM *(Random-access memory)*, memória ROM *(Read-only memory)*, velocidade do processador, duração da bateria e número de periféricos. Estes recursos são intercambiáveis, por exemplo, se for necessário mais velocidade talvez seja possível diminuir a duração da bateria e vice-versa.

2.3 Banco de Dados

Banco de dados é uma coleção organizada de informações ou dados estruturados, normalmente armazenados eletronicamente em um sistema de computador (ORACLE, 2021a).

De acordo com Elmasri e Navathe (2011), o papel desempenhado pelo banco de dados em quase todas as áreas em que os computadores são usados é crítico. Os dados de um banco são de diferentes tipos, como imagens, fotos e números, por exemplo.

Elmasri e Navathe (2011) expõem algumas propriedades que são implícitas no uso do termo banco de dados, entre elas:

- Existe relações do banco de dados com o mundo real, mini-mundo, no qual mudanças nesse mini-mundo são refletidas no banco de dados;
- Banco de dados é uma coleção logicamente coerente de dados com algum significado inerente, uma variedade aleatória de dados não pode ser chamada de banco de dados;
- A projeção, construção e população do banco de dados com dados é feita para uma finalidade específica. Existem aplicações as quais são concebidas a grupos interessados.

Em suma, qualquer mudança no mundo real que tenha relação a algum banco de dados, deve ter sua atualização precisa, para não ocasionar dados irregulares e errôneos.

A manutenção de um banco de dados é feita por um sistema gerenciador de banco de dados (SGBD - *Database Management System*). Elmasri e Navathe (2011) trazem como característica de SGBD, ser um sistema de *software* de uso geral que permite aos usuários criar e manter um banco de dados, ou seja, sua definição, construção, manipulação e compartilhamento.

A linguagem padrão de SGBDs é o SQL(*Structured Query Language*), que aborda instruções de definição, consulta e atualização de dados. Essa linguagem é responsável por permitir a manutenção em um banco de dados (ELMASRI E NAVATHE, 2011).

2.3.1 Visualização de Dados

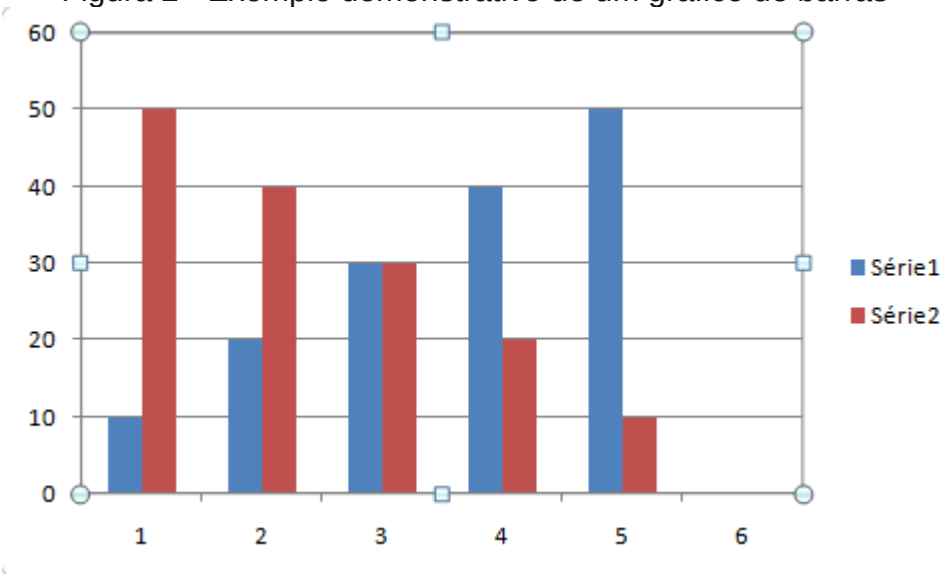
A visualização dos dados é a representação e apresentação de dados que exploram a habilidade de percepção visual com a finalidade de aumentar a compreensão (KIRK, 2012).

Em bancos de dados é utilizada a visualização de dados. De acordo com a IBM (2021), existem dois tipos de visualizações: as visualizações dinâmicas, que são atualizadas automaticamente quando um ou mais objetos são criados ou alterados e as visualizações estáticas, que necessitam de serem atualizadas

manualmente. Ambas as formas trazem informações dos dados para o usuário de forma gráfica (ORACLE, 2021b).

Abaixo seguem dois exemplos (Figura 2 e Figura 3) de tipos de visualização de dados em forma de gráfico.

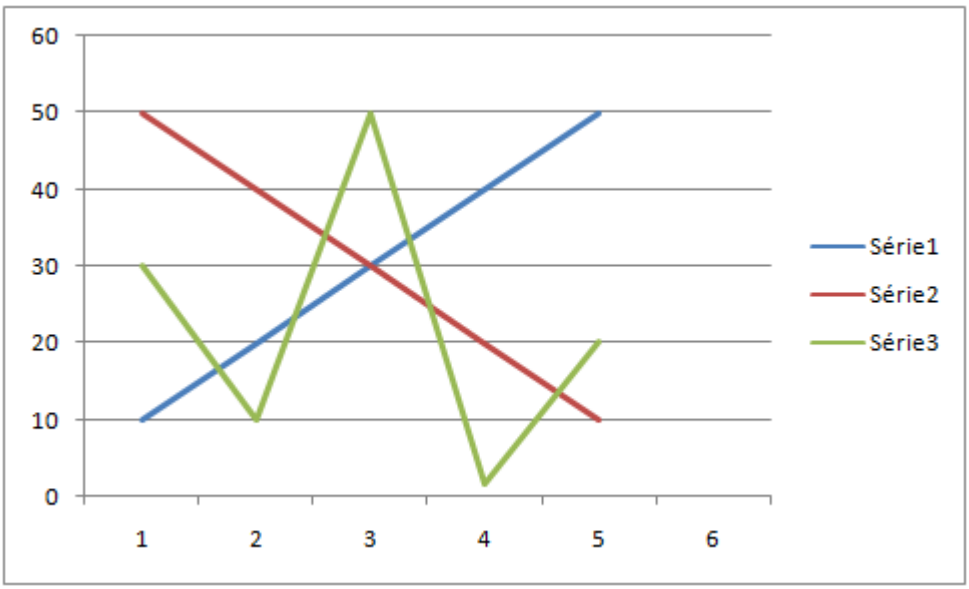
Figura 2 - Exemplo demonstrativo de um gráfico de barras



Fonte: Os autores.

Neste gráfico foram simuladas as colunas vermelhas e azuis com os mesmos valores de 10 a 50, porém em posições opostas.

Figura 3 - Exemplo demonstrativo de um gráfico de linhas



Fonte: Os autores.

Neste gráfico transformou-se a Figura 2 em um gráfico de linhas, acrescentando a ela uma terceira linha com elementos aleatórios fora do padrão seguido.

3 Planejamento e Escopo do Projeto

Antes de dar início ao desenvolvimento de qualquer projeto, como este, por exemplo, é interessante fazer o planejamento e definição do escopo. Para planejar e definir um escopo, é necessário elicitar, analisar e documentar os requisitos do projeto. Seguindo estas etapas obtém-se um melhor entendimento do que deve ser feito para desenvolver o projeto, e das características específicas a serem seguidas, não ocorrendo assim qualquer desvio dos objetivos principais da solução.

3.1 Engenharia de Requisitos

Sommerville e Sawyer (1997) definem que os requisitos são definidos durante os estágios iniciais do desenvolvimento de um sistema, como uma especificação do que pode ser implementado. Eles são descrições de como o sistema deve se comportar ou de uma propriedade ou atributo do sistema. Eles também podem restringir o processo de desenvolvimento do sistema.

A engenharia de requisitos consiste no processo de elicitar, analisar e documentar requisitos. Sendo assim, algumas técnicas de tal processo foram utilizadas durante o ciclo de vida de desenvolvimento deste protótipo.

Uma das técnicas utilizadas foi a de *brainstorm*, ou seja, levantar e explorar ideias das mais variadas possíveis. Em um primeiro momento, os autores a utilizaram para definir qual seria o tema deste projeto, levantando a ideia de um dispositivo capaz de aferir os sinais vitais de *pets*, especificamente cães e gatos. E em um segundo momento, para elicitar as funcionalidades e a arquitetura que poderiam constituir o sistema.

Depois de elicitados os requisitos, foi feita a análise deles, levando em consideração a viabilidade das ideias levantadas. Nesta análise também utilizou-se da ajuda de professores experientes na área de desenvolvimento de *software*, via reuniões *online* recorrentes.

Também foi feito um acompanhamento com um veterinário, por meio da técnica de entrevista, para esclarecimento de dúvidas referentes à saúde animal e obtenção de informações essenciais para a construção do dispositivo, a definição da arquitetura e de casos de uso do sistema. A transcrição das entrevistas pode ser encontrada no GitHub (ANDRADE; SOUZA, 2021).

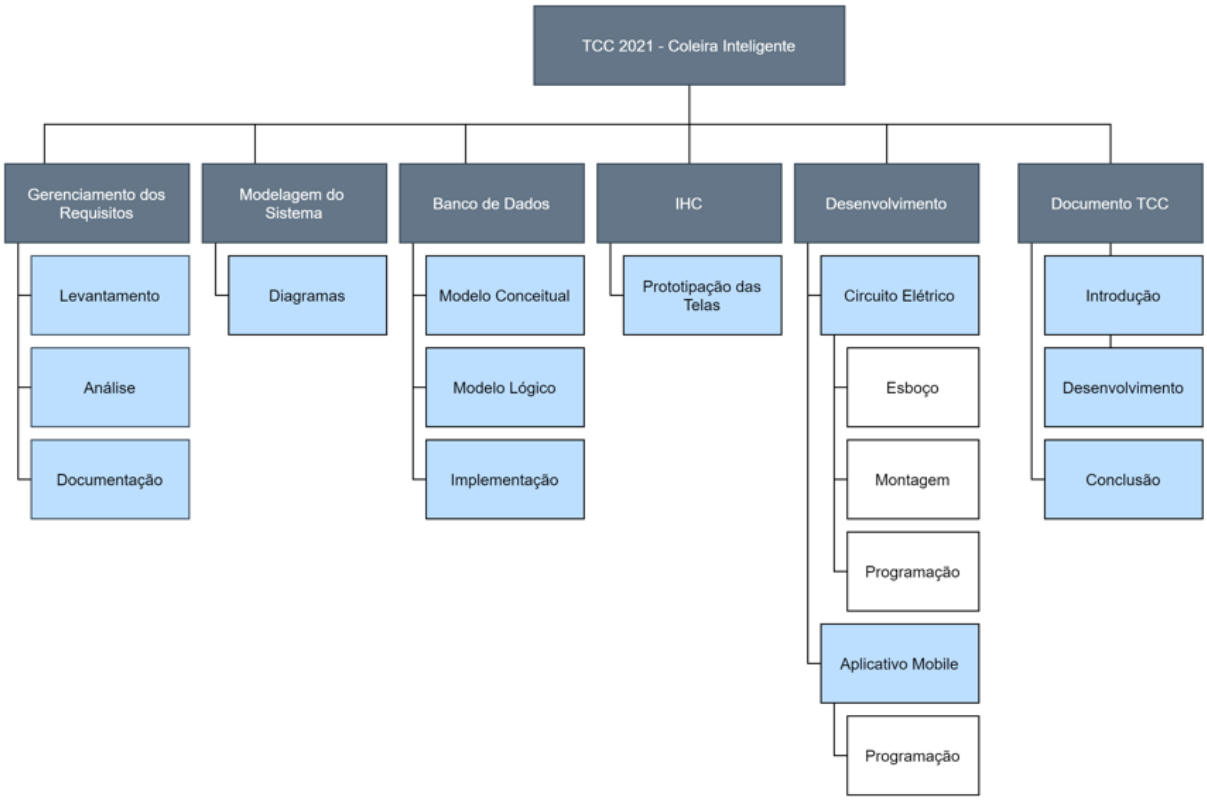
A etapa de documentação dos requisitos foi feita incrementalmente conforme foram feitas as etapas de elicitação e análise.

3.2 EAP (Estrutura Analítica do Projeto)

Uma vez definido o escopo foi possível fazer a Estrutura Analítica do Projeto (EAP), a qual consiste em pegar todo o processo do projeto e dividi-lo em etapas, especificando-as com mais detalhes em subetapas, para ser possível a divisão de responsabilidades entre os autores. De acordo com o PMI (2013), o EAP é a decomposição hierárquica do escopo total do trabalho a ser executado pela equipe do projeto a fim de alcançar os objetivos e criar as entregas exigidas.

O EAP da Coleira Inteligente foi decomposto nas etapas apresentadas na Figura 4.

Figura 4 - EAP do sistema



Fonte: Os autores.

3.3 Cronograma e Atribuição das Tarefas

"O cronograma do projeto é uma saída de um modelo de cronograma que apresenta a conexão de atividades com datas, durações, marcos e recursos planejados. O cronograma do projeto inclui pelo menos uma data de início e de término planejadas para cada atividade" (PMI, 2013). Utilizar um cronograma auxilia no acompanhamento da evolução do projeto e dos prazos de entregas.

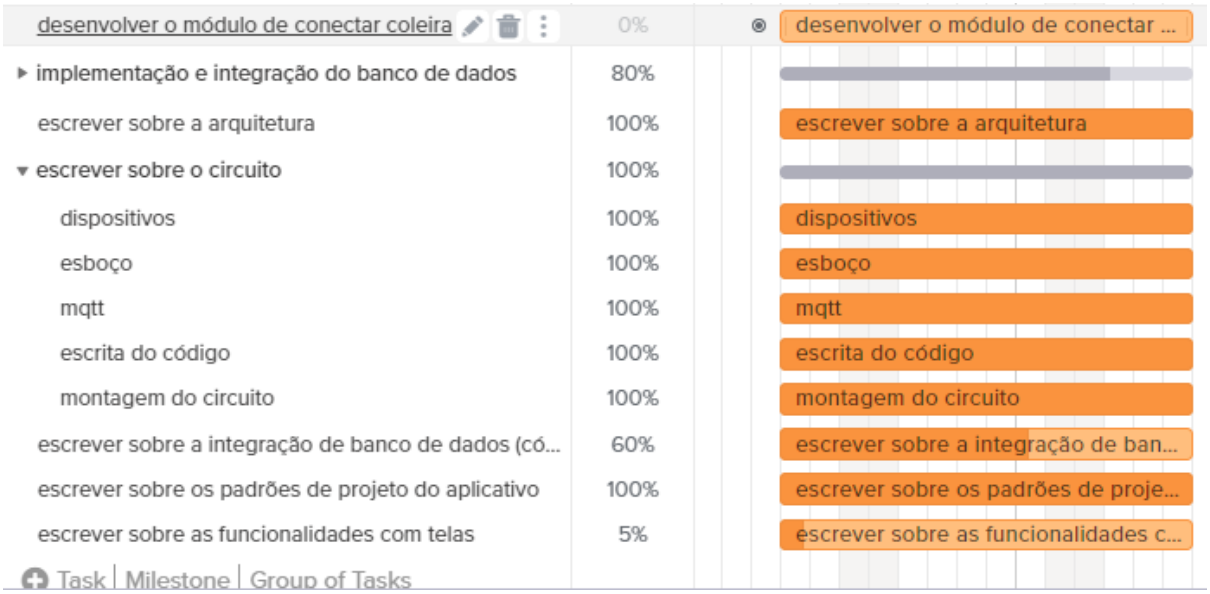
Com base no EAP e na data de entrega deste artigo, foi criado o cronograma. Neste cronograma foram inseridas as tarefas a serem realizadas, e posteriormente atribuídas a cada um dos autores. A fim de facilitar a visualização das tarefas que foram distribuídas, optou-se por utilizar o gráfico de barras para a exibição adequada de todas as informações contidas no cronograma.

Os gráficos de barras, também chamados de diagrama Gantt,

> "[...] frequentemente são usados em apresentações gerenciais. Para controle e comunicação gerencial, a atividade de resumo mais ampla e mais abrangente [...] sendo mostrada em relatórios de gráficos de barras. " (PMI, 2013)

Dentre as ferramentas de cronograma do tipo gráfico de barras existentes no mercado atual, os autores optaram pela plataforma gratuita TeamGantt. Esta ferramenta implementa todas as características de um gráfico de barras, torna o processo de criar e dividir tarefas bastante prático, e notifica sobre a evolução das tarefas por email. Na Figura 5, é apresentado o cronograma do projeto utilizando esta ferramenta.

Figura 5 - Parte do cronograma das tarefas



Fonte: Autores

4 Modelagem do Sistema

Após a realização do processo de elicitação e análise dos requisitos, foi possível realizar o processo de modelagem do sistema, gerando assim uma imagem mais nítida de como o sistema será estruturado e desenvolvido. Todos os diagramas e documentações se encontram no GitHub (ANDRADE; SOUZA, 2021).

4.1 Arquitetura

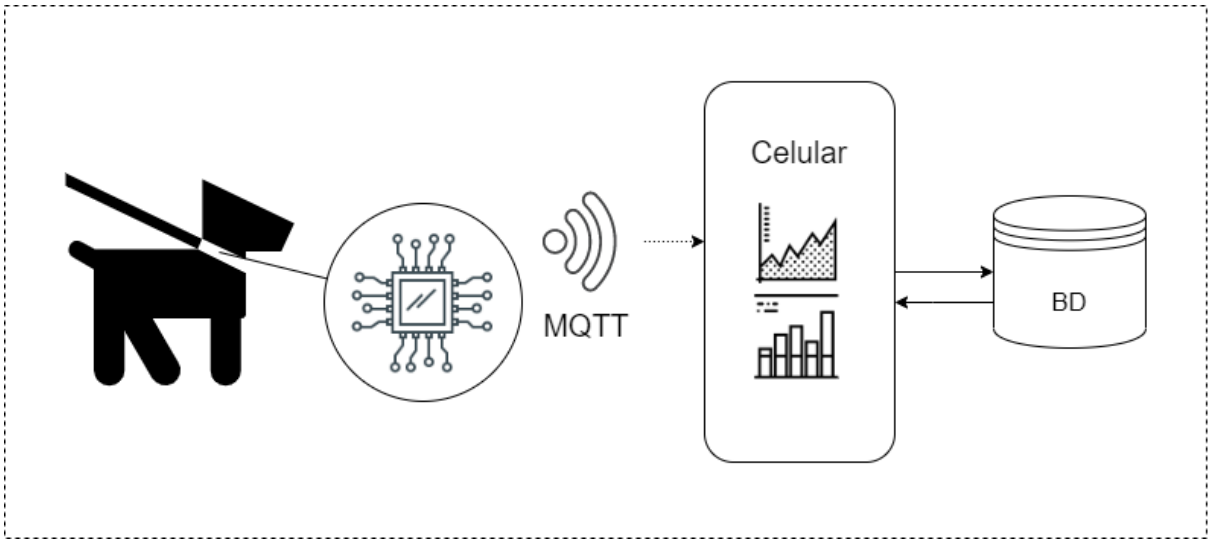
Sommerville e Sawyer (1997), explicam que o modelo de arquitetura de um sistema ajuda o engenheiro de requisitos a identificar requisitos que envolvam mais de um subsistema, e essa não identificação pode causar problemas caso ocorram mudanças no sistema a nível global.

Para a modelagem arquitetural, os autores utilizaram como base os requisitos já elicitados, identificando todo o processo de funcionamento do sistema, desde a aferição do sinal vital do *pet* até a visualização gráfica dos dados no dispositivo móvel, e depois realizando uma representação gráfica para melhor compreensão da arquitetura planejada.

De forma sucinta o funcionamento do sistema se daria da seguinte forma: o usuário acessaria o aplicativo do dispositivo móvel, cadastraria o seu animal e conectaria a coleira ao aplicativo através de uma rede sem fio e, conforme as aferições dos sinais vitais fossem feitas, o usuário conseguiria visualizar graficamente o *status* do *pet* praticamente em tempo real. No caso deste protótipo a coleira seria apenas um circuito elétrico capaz de simular uma aferição feita em um animal.

Detalhes de cada componente da arquitetura estão presentes na Seção 5 deste artigo. Dentre os componentes a serem detalhados estará o MQTT (*MQ Telemetry Transport*), o protocolo responsável pela comunicação entre circuito elétrico e dispositivo móvel. A arquitetura está representada graficamente na Figura 6.

Figura 6 - Representação gráfica da arquitetura



Fonte: Os autores.

4.2 Documento de Requisitos

O documento de requisitos deste protótipo foi elaborado pelos autores a partir dos requisitos encontrados. Neste documento foram colocadas por exemplo as classes de usuário e as características do protótipo referentes ao tipo de ambiente operacional. Também foram listadas as tecnologias e as linguagens de programação necessárias.

Para as funcionalidades do sistema, foram documentados os requisitos funcionais. É apresentado na Tabela 1 um exemplo de um requisito funcional utilizado no sistema.

Tabela 1 - Exemplo de requisito funcional do sistema

ID e nome:	**RF-2 - Parâmetros dos sinais vitais**
Descrição:	**Parâmetros ideais de frequência cardíaca de pets adultos:** Cães: 80 - 120 BPM Gatos: 110 - 130 BPM O sistema emitirá alertas quando a frequência for diferente destes intervalos. **Parâmetros ideais de saturação de oxigênio de pets adultos:** Cães e gatos: 95% - 100% Abaixo de 95% a saturação é preocupante e abaixo de 90% é crítica. O sistema emitirá alertas diferentes nas duas situações. Os gráficos de visualização devem ter indicações visuais

	quando algum sinal vital estiver fora do ideal. O sistema deve checar se os sinais vitais estão no intervalo ideal para emitir os alertas e realizar as indicações visuais. Este requisito está relacionado aos **UC-2** e **UC-3**.

Fonte: Os autores.

Também foram documentados os casos de uso considerados principais. Wiegers e Beatty (2013) explicam que um caso de uso descreve a sequência de interações de um ator externo em um sistema, para obter algum valor. A elaboração dos casos de uso foi feita de acordo com as funcionalidades e requisitos encontrados do sistema. Na Tabela 2 a seguir, é apresentado um exemplo de uma documentação de um caso de uso.

Tabela 2 - Exemplo de caso de uso do sistema

ID e nome:	**UC-3 Visualizar dados do animal**
Ator primário:	Usuário
Criado por:	Saulo e Ana
Pré-condições:	PRE-1. Que exista algum animal já cadastrado. PRE-2. Que tenha registrado ao menos 1 aferição dos sinais vitais do animal.
Fluxo normal:	**3.0 Visualizar os dados** 1 Escolher animal 2 Visualizar a última aferição em tempo real ou a aferição escolhida (consultar 3.1) 3 Visualizar o gráfico padrão ou um gráfico personalizado (consultar 3.2)
Fluxo alternativo:	**3.1 Visualizar aferição escolhida** 1 Escolher a aferição desejada no gráfico 2 Visualizar a aferição escolhida **3.2 Visualizar gráfico personalizado** 1 Alterar as opções do gráfico padrão. 2 Visualizar novo gráfico.
Pós-condições:	Nenhuma
Exceções:	Nenhuma

Fonte: Os autores.

Após a criação dos requisitos e os casos de uso, foi criada a matriz de rastreabilidade entre eles. Os autores também realizaram a criação de um dicionário de dados para a modelagem do banco de dados e sobretudo para que os dados se mantivessem consistentes durante todo o processo de desenvolvimento. O documento de requisitos completo se encontra disponível no GitHub (ANDRADE; SOUZA, 2021).

4.3 Diagrama de Casos de Uso

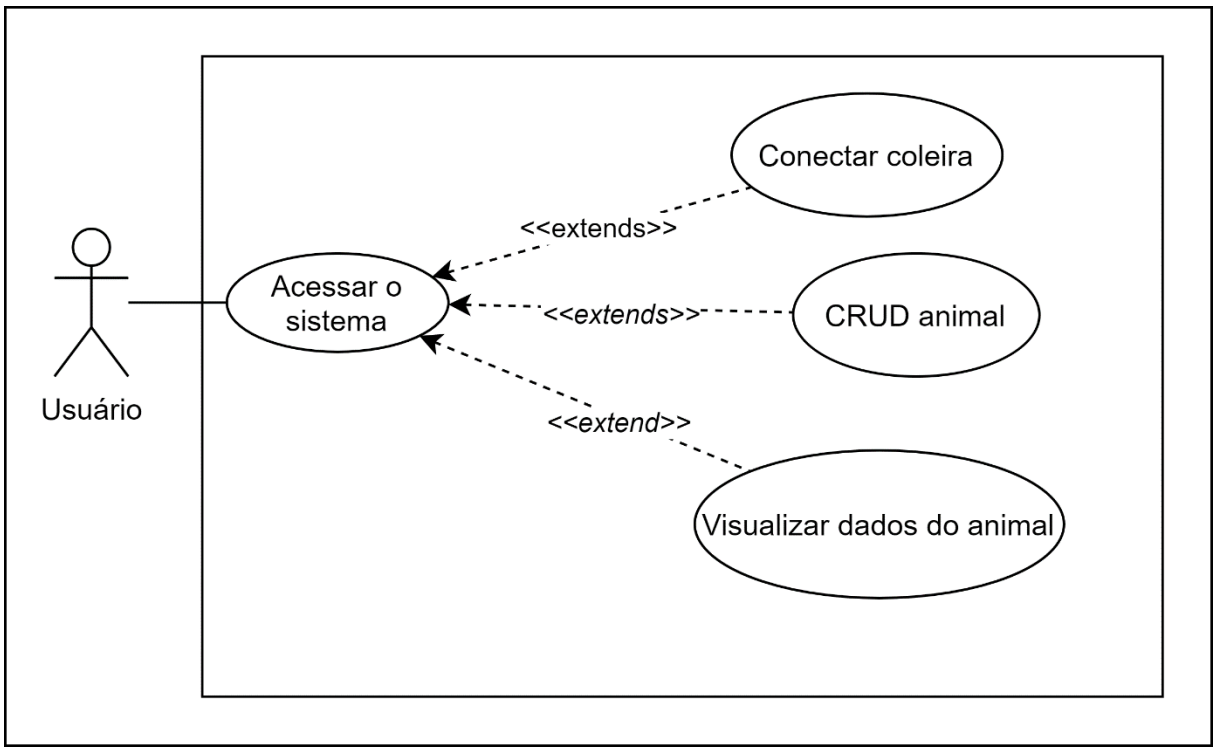
De acordo com Wiegers e Beatty (2013) os diagramas de caso de uso fornecem uma representação de alto nível dos requisitos de usuário.

O diagrama de casos de uso do projeto foi baseado nos casos de uso já documentados. Ele foi elaborado pelos autores seguindo os padrões UML (*Unified Modeling Language*), com o propósito de apresentar as principais ações que seriam feitas através do dispositivo móvel, e identificar os papéis do usuário, do sistema e do banco de dados.

Ações como conexão da coleira, operações CRUD (*Create, Read, Update, Delete*) nos dados do animal e a visualização desses dados são relacionadas diretamente ao sistema, podendo ocorrer de forma simultânea ou não, enquanto o usuário realiza o seu acesso na aplicação.

O diagrama de casos de uso é apresentado na Figura 7 abaixo.

Figura 7 - Diagrama de casos de uso



Fonte: Os autores.

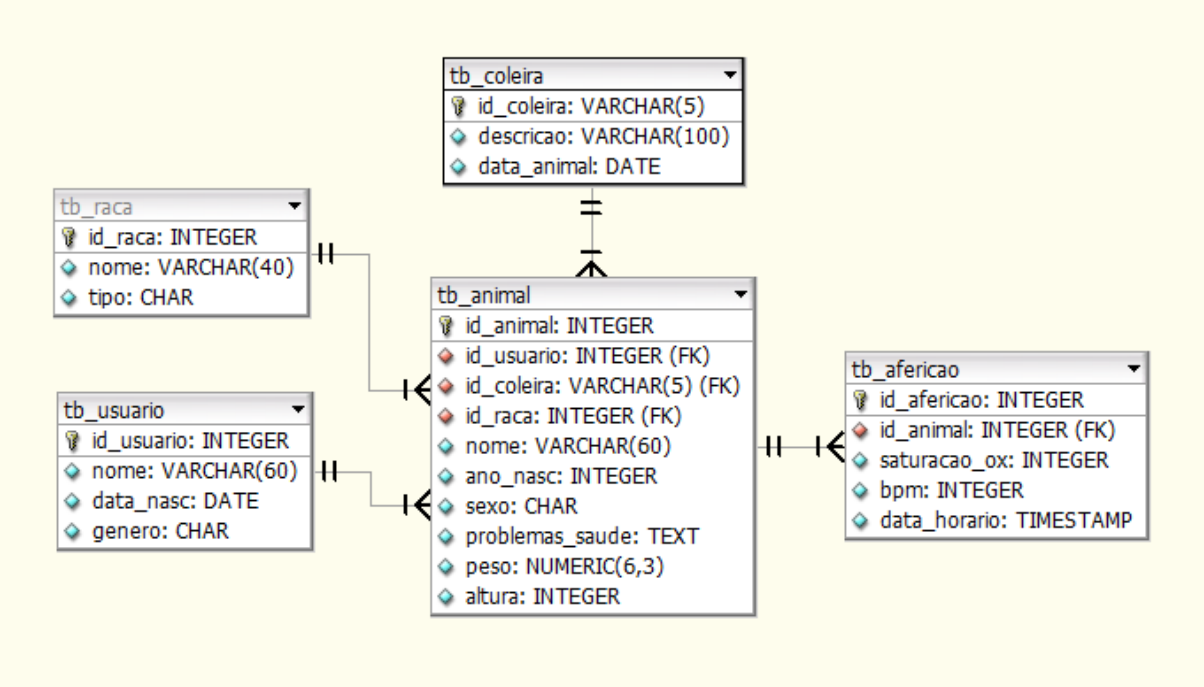
4.4 Modelagem de Dados

O processo de modelagem de dados, realizado a partir do documento de requisitos, permitiu que as entidades do sistema, tais como usuários e animais, fossem modeladas da melhor maneira possível.

O primeiro modelo criado, o modelo conceitual, teve a sua versão preliminar baseada nas entidades, cada uma com seus atributos, como nome do animal ou a métrica do sinal vital, por exemplo, além do relacionamento entre elas.

O segundo modelo, o lógico, possui mais detalhes em relação ao primeiro modelo criado, contendo o tipo de dado de cada atributo, relacionamentos, garantindo a integridade relacional do sistema. A Figura 8 abaixo apresenta a versão desse modelo.

Figura 8 - Modelo lógico



Fonte: Os autores.

5 Desenvolvimento

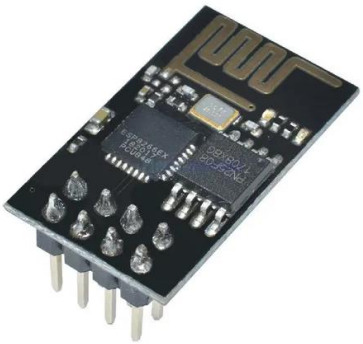
5.1 Desenvolvimento do Circuito Elétrico

Os autores estudaram as funcionalidades de vários componentes elétricos, analisando quais seriam os mais adequados no circuito do protótipo. O primeiro componente escolhido foi o MAX30100 (Figura 9)**,** sensor fundamental na aferição dos sinais vitais. Idealmente, a comunicação do circuito com a internet deveria ser simples, por isso escolheu-se o componente ESP-01 (Figura 10), um módulo *wi-fi* que foi usado posteriormente para a implementação do protocolo MQTT. Também foi utilizado um adaptador USB (*Universal Serial Bus*) (Figura 11) para alimentação do circuito e transferência de códigos fonte para o ESP-01.

Figura 9 - MAX30100

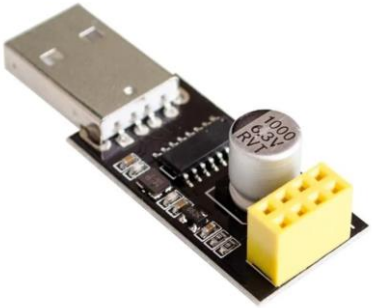
Fonte: MERCADO LIVRE, [S.D]a

Figura 10 - ESP-01



Fonte: MERCADO LIVRE, [S. D]b

Figura 11 - Adaptador USB

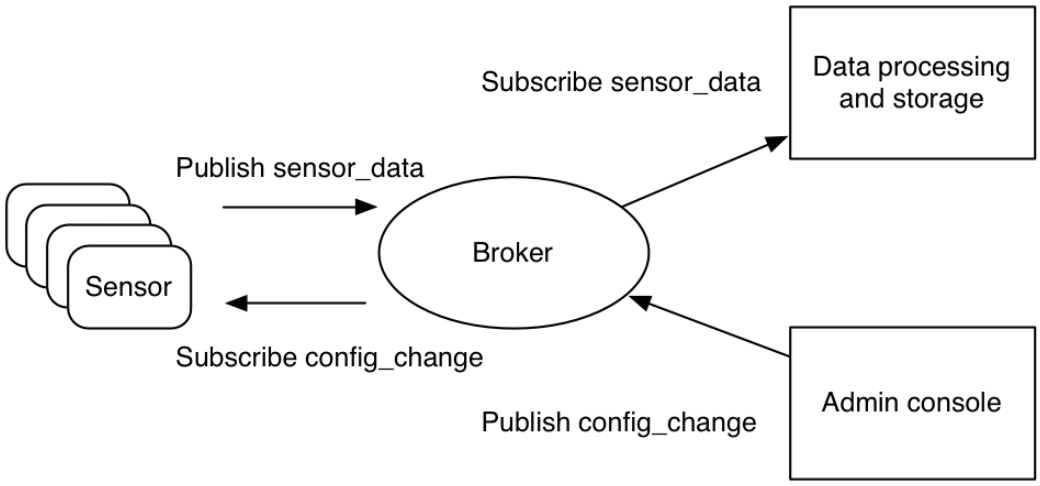


Fonte: MERCADO LIVRE, [S.D]c

MQTT é um protocolo de mensagens de publicação/assinatura para a comunicação de dispositivos remotos utilizando pouco código e uma rede mínima (MQTT, 2020). Ele também minimiza o risco da perda de dados em situações como falta de rede de *internet* no momento de envio. Os autores optaram por utilizar este protocolo com base nos requisitos, e em estudos sobre seu modo de operação.

Nesse protocolo, existem alguns termos relevantes como *broker*, cliente, publicação e assinatura, e tópico. O funcionamento do MQTT, ilustrado na Figura 12, se dá da seguinte maneira: sensores publicam mensagens em tópicos, que seriam como *tags* que os clientes assinam. As publicações são feitas em um *broker*, um servidor seguro de armazenamento temporário. A partir deste *broker*, as mensagens são enviadas para os clientes. Somente os clientes assinantes do tópico das mensagens enviadas pelos sensores poderão visualizá-las.

Figura 12 - O modelo de publicação e assinatura do MQTT para sensores de IoT



Fonte: YUAN, 2017

De acordo com a arquitetura deste protótipo, exemplificada na Figura 6, para que o dispositivo móvel receba os dados do circuito elétrico, ele deve realizar uma conexão com o *broker*. O recebimento dos dados pelo dispositivo é feito por meio do processo de publicação e assinatura, resumido nos parágrafos anteriores. Após essas informações chegarem ao dispositivo móvel, elas são armazenadas no banco de dados local.

O circuito elétrico deste protótipo seria como uma coleira. Em uma solução específica para o mercado, haveriam várias dessas coleiras. O tópico de assinatura das coleiras foi definido em um formato que garantisse a identificação específica de cada uma delas. O formato consiste da abreviação da palavra “aferição”, seguida por cinco caracteres alfanuméricos, gerados aleatoriamente. Um tópico que segue estas especificações é o **“**AFER5Y3C2**”**, utilizado neste projeto.

Após a aquisição dos componentes, os autores trabalharam na montagem do circuito eletrônico, utilizando a ferramenta Fritzing para o esboço (Figura 14) e a ferramenta Arduino IDE para sua codificação (Figura 13). O esboço e o código na íntegra se encontra disponível para consulta no GitHub (ANDRADE; SOUZA, 2021).

Arduino IDE é um ambiente de desenvolvimento *open-source* (ARDUINO, 2018) que foi utilizado pelos autores para a programação do código do circuito e a testagem de seu funcionamento.

Fritzing é uma ferramenta/biblioteca *open-source* para o desenho e montagem digital do circuito e dos seus componentes eletrônicos (FRITZING, 2021).

Figura 13 - Trecho do código do circuito

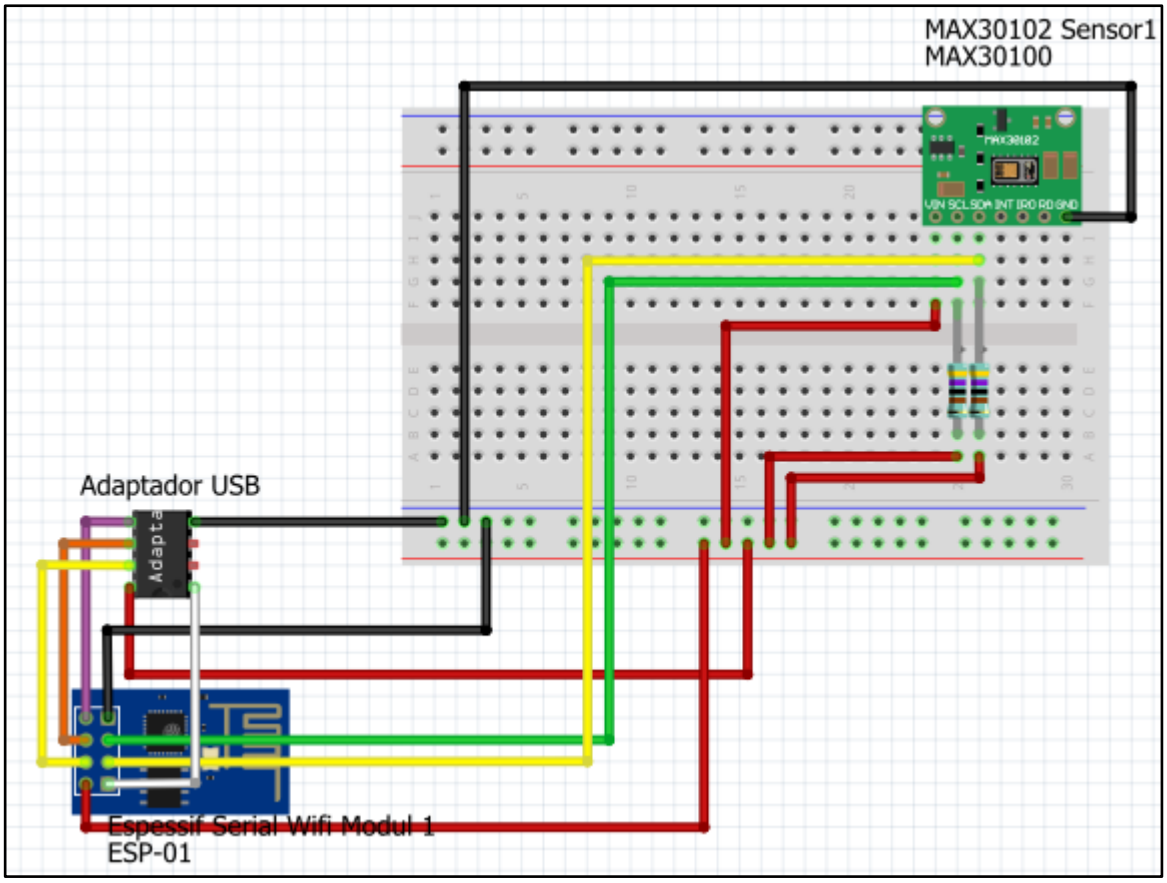
```
// Para ambos, se o valor for 0 significa que o sinal vital é inválido.
if (millis() - tempoUltimaAfericao > INTERVALO_AFERICAO_MS) {
    Serial.println("Enviando BPM e OX");
    payload = String(sensor.getHeartRate()) + "-" + String(sensor.getSpO2());
    Serial.print(sensor.getHeartRate());
    Serial.print("-");
    Serial.print(sensor.getSpO2());
    MQTT.publish(TOPICO_PUBLISH, payload.c_str());

    tempoUltimaAfericao = millis();
}

MQTT.loop();
```

Fonte: Os autores.

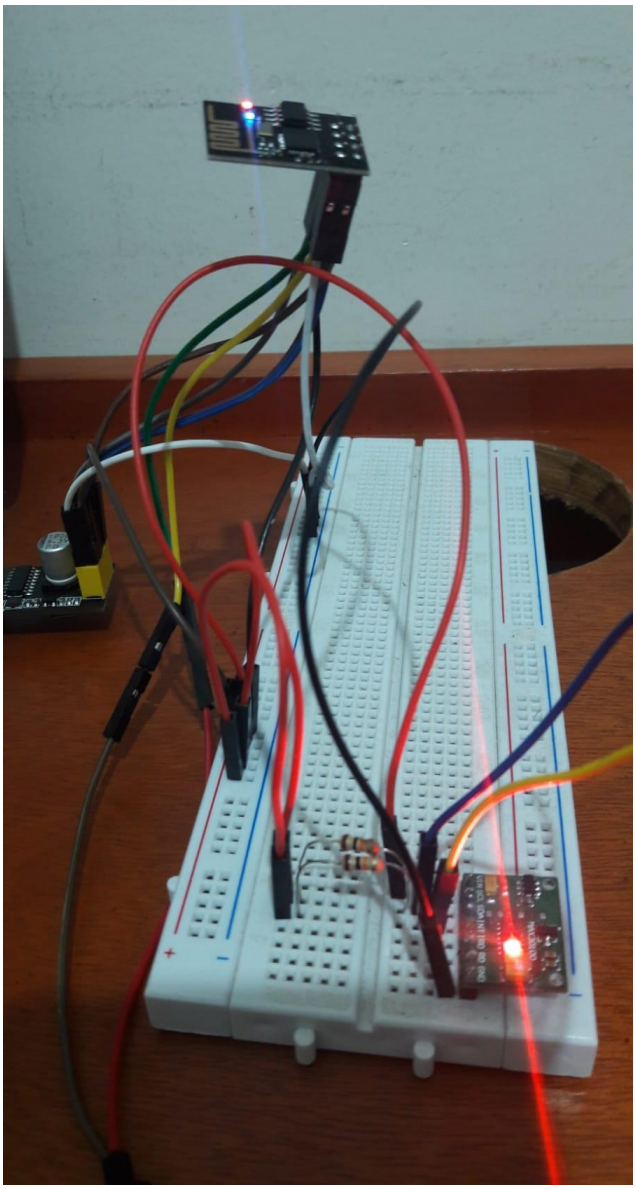
Figura 14 - Esboço do circuito elétrico



Fonte: Os autores.

Abaixo é apresentado uma foto do circuito montado funcionando (Figura 15).

Figura 15 - Circuito elétrico montado



Fonte: Os autores.

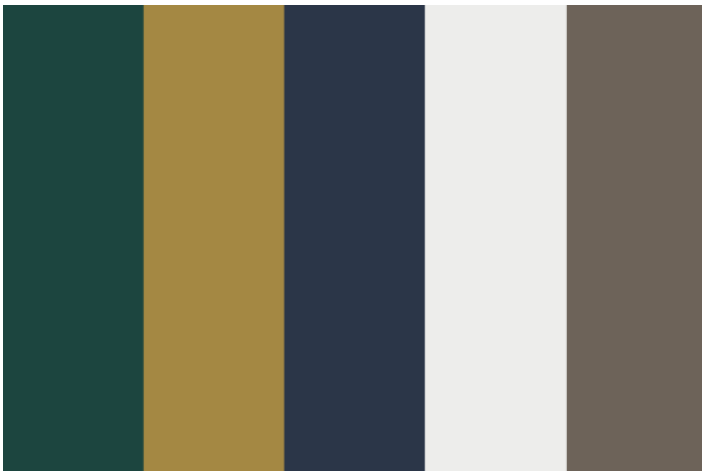
Após a implementação do circuito elétrico, os autores trabalharam no desenvolvimento do aplicativo. Detalhes do desenvolvimento serão apresentados a seguir.

5.2 Desenvolvimento do Aplicativo

Antes de partir para o desenvolvimento, foi realizada a prototipação das telas, levando em consideração um *design* gráfico que facilitasse a interação do usuário.

Foi escolhida uma paleta de cores que fosse moderna mas que inspirasse levemente as ideias de natureza e animais. Essa paleta é apresentada na Figura 16 abaixo.

Figura 16 - Paleta de cores do aplicativo

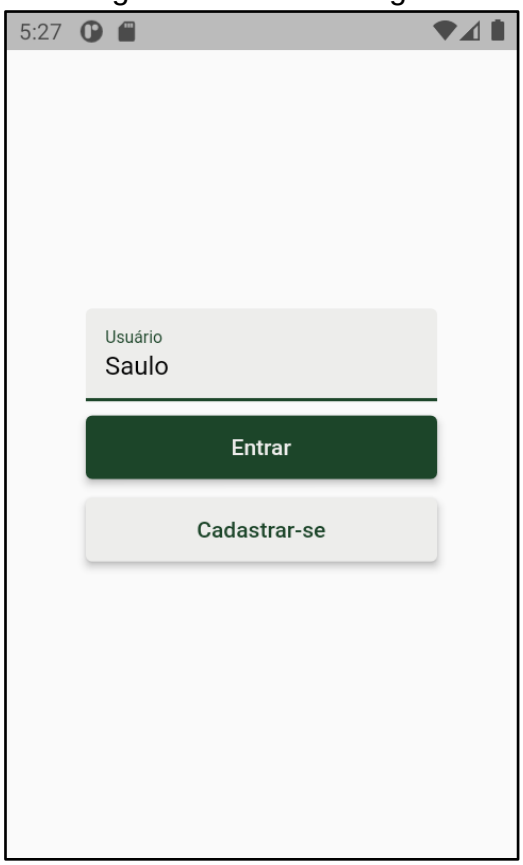


Fonte: Os autores.

5.2.1 Telas e Funcionalidades

A tela inicial do *app* será a tela de *login* (Figura 17). Como este sistema é um protótipo, pensou-se em um *login* simplificado com apenas o campo de nome de usuário. Porém, antes que o usuário possa se autenticar, é necessário que ele realize seu cadastro. A tela de cadastro (Figura 18) possui os campos de nome, gênero e data de nascimento.

Figura 17 - Tela de login



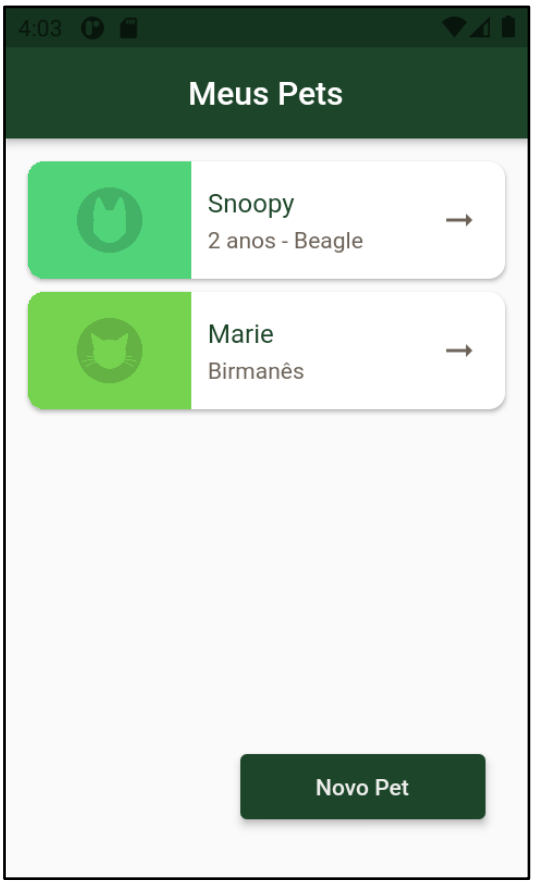
Fonte: Os autores.

Figura 18 - Tela de cadastro de usuário

Fonte: Os autores.

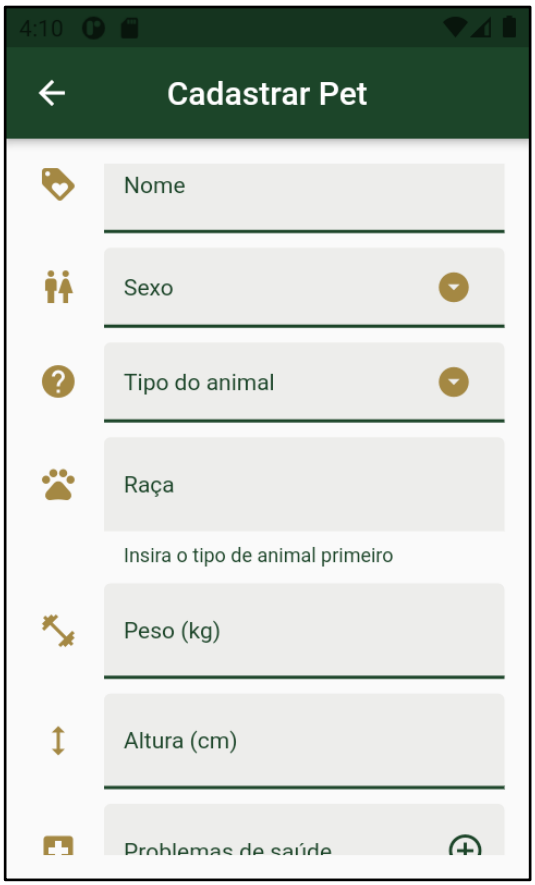
Após o *login*, o usuário acessará a tela “Meus Pets” (Figura 19) na qual ele poderá cadastrar novos *pets* ou acessar seus animais já cadastrados. Na tela de cadastro de *pets* (Figura 20), o usuário pode inserir informações como nome, idade e problemas de saúde do animal.

Figura 19 - Tela “Meus *Pets*”



Fonte: Os autores.

Figura 20 - Tela de cadastro de *pet*

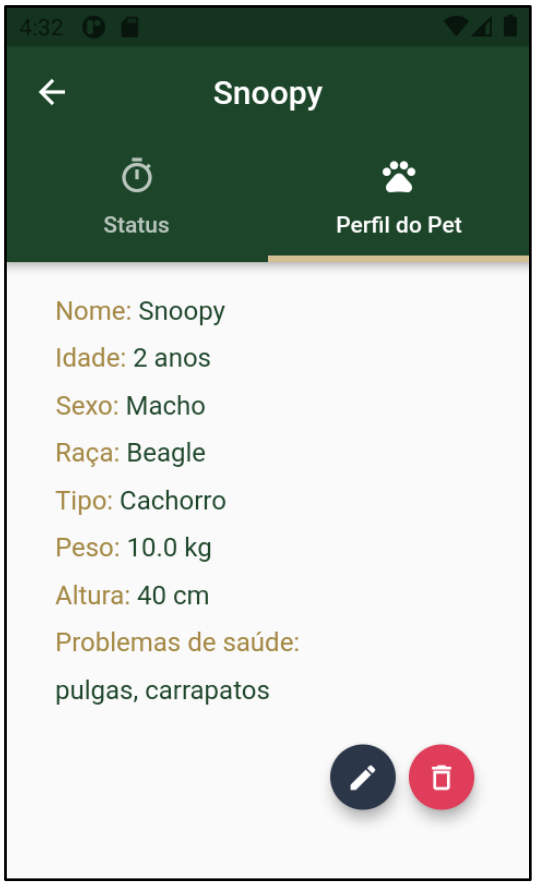


Fonte: Os autores.

Ao consultar um animal, o usuário tem acesso à tela de “Perfil” (Figura 21), na qual ele pode encontrar todas as informações do *pet*, além de poder editar as informações ou mesmo excluir o *pet* cadastrado.

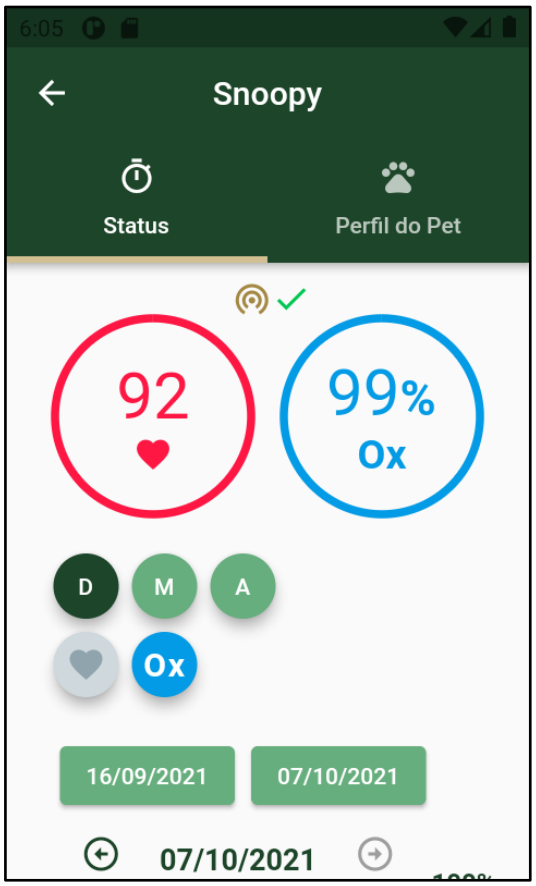
O usuário também tem acesso à tela de “*Status*” (Figura 22 e Figura 23), na qual é possível visualizar os dados da última aferição realizada além dos gráficos de todas as aferições já feitas, podendo escolher o período dessas aferições. Quando um animal é cadastrado, o usuário deverá se conectar à coleira primeiro.

Figura 21 - Tela de “Perfil”



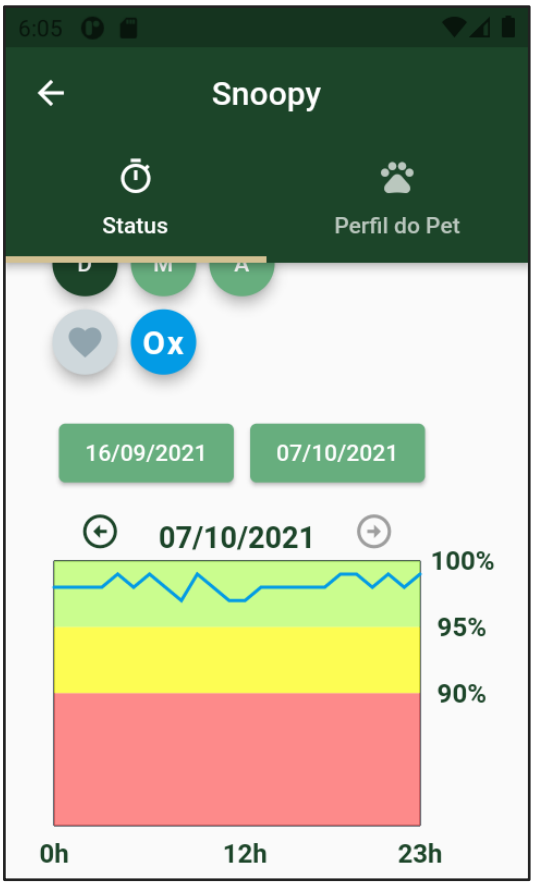
Fonte: Os autores.

Figura 22 - Tela de “*Status*” - Sinais Vitais



Fonte: Os autores.

Figura 23 - Tela de “*Status*” - Gráficos



Fonte: Os autores.

5.2.2 Implementação

Na criação do aplicativo, foi escolhido o SDK (*Software Development Kit*) Flutter, pela facilidade de manutenção, por ser de código aberto e pelo domínio desta tecnologia pelos autores. Flutter é o *kit* de ferramentas de UI (*User Interface*) portátil do Google para a criação de aplicativos multiplataforma compilados de forma nativa a partir de uma única base de código (FLUTTER, 2021).

Os autores implementaram as telas e funcionalidades, utilizando Flutter e a linguagem de programação Dart. No código da aplicação programou-se a conexão com o circuito elétrico, e a comunicação do protocolo MQTT para obter os dados aferidos do sensor. A Figura 24 apresenta um trecho do código de comunicação entre a aplicação e o circuito elétrico. A íntegra do código fonte se encontra no GitHub (ANDRADE; SOUZA, 2021).

Figura 24 - Trecho do código para comunicação

```
void onConnected(String idColeira) {
  print('MQTT conectado');
  client!.subscribe("AFER$idColeira", MqttQos.atLeastOnce);
  client!.updates!.listen((List<MqttReceivedMessage<MqttMessage?>>? c) {
    final MqttPublishMessage recMess = c![0].payload as MqttPublishMessage;
    final List<String> msg =
        MqttPublishPayload.bytesToStringAsString(recMess.payload.message).
        split("-");
    repository.setAfericao(idColeira: msg[0], bpm: int.parse(msg[1]), ox: int.
    parse(msg[2]), data: DateTime.parse(msg[3]));
  });
}
```

Fonte: Os autores.

5.2.2.1 Padrões de Projeto

O código do aplicativo foi desenvolvido utilizando dois modelos de padrões de projeto: State e MVC (*Model-View-Controller*).

Gamma et al. (1995), conhecidos como GoF (*Gang of Four*), trazem o significado de padrões de projeto como estruturas de soluções frequentemente usadas para problemas corriqueiros no desenvolvimento de *softwares*.

O padrão State permite que um objeto altere sua ação quando o seu estado interno muda (GAMMA et al., 1995).

Na Figura 25 do projeto, utilizando o padrão State, é apresentada a classe *PetAfericaoState*, ou seja, a classe do estado de aferição do *pet*. Cada classe filha representa diferentes estados operacionais. A classe *PetPage*, por exemplo, muda o seu estado para *PetAfericaoStateSucess* quando ela consegue consultar as aferições de um *pet*.

Figura 25 - Exemplo de implementação do padrão *"State"*

```
3   abstract class PetAfericaoState {}
4
5   class PetAfericaoStateEmpty extends PetAfericaoState {}
6
7   class PetAfericaoStateLoading extends PetAfericaoState {}
8
9   class PetAfericaoStateSuccess extends PetAfericaoState {
10    final List<AfericaoModel> afericoes;
11    PetAfericaoStateSuccess({
12      required this.afericoes,
13    });
14  }
15
16  class PetAfericaoStateFailure extends PetAfericaoState {
17    final String message;
```

Fonte: Os autores.

O padrão MVC desacopla objetos para aumentar a flexibilidade e a reutilização das UI (*User Interface*). *Models* são objetos que representam entidades, *Views* são as interfaces gráficas, e os *Controllers* definem a forma como as interfaces reagem à entrada dos usuários (GAMMA et al. 1995). Por exemplo, para uma mesma tela de aplicativo, existem modelos que variam suas operações conforme a interação do usuário.

É apresentado na Figura 26 um exemplo de um *Controller* da tela do cadastro e edição de *pets*, que intermedia uma requisição, fazendo uma operação no banco de dados de acordo com o cenário de criação ou edição de um *pet.*

Figura 26 - Exemplo de um *Controller* no padrão “MVC”

```
7   class CadastroEdicaoPetController {
8     final AnimalRepository animalRepository = AnimalRepositorySqlite();
9     Function(CadastroEdicaoPetState state)? onListen;
10    CadastroEdicaoPetState state = CadastroEdicaoPetStateEmpty();
11
12    createAndGetAnimal({required int idUser, required int idRaca, required String
      nome, required String sexo, required int? anoNasc, required String?
      problemasSaude, required double? peso, required int? altura}) async {
13      update(CadastroEdicaoPetStateLoading());
14      try {
15        final AnimalModel animal = await animalRepository.createAndGetAnimal
          (idUser: idUser, idRaca: idRaca, nome: nome, sexo: sexo, anoNasc:
          anoNasc, problemasSaude: problemasSaude, peso: peso, altura: altura);
16        update(CadastroPetStateSuccess(animal: animal));
17      } on AnimalExistenteException catch (e) {
18        update(CadastroEdicaoPetStateFailure(message: e.toString()));
```

Fonte: Os autores

5.2.2.2 Implementação do Banco de Dados

A partir do modelo lógico criado pelos autores, foi utilizada a biblioteca de banco de dados SQLite, uma biblioteca totalmente gratuita e de fácil compreensão da linguagem SQL. Uma vez que está embutida em praticamente todos os dispositivos móveis, esta biblioteca é utilizada principalmente no desenvolvimento *mobile*, porém não se limitando apenas ao *mobile*.

O SQLite, se define como pequeno, rápido e de confiança, sendo um SGBD embutido, auto-suficiente, *serverless*, de configuração-zero e transacional (SQLite, 2021).

Foi utilizada a classe auxiliar *DatabaseHelper* para realizar a configuração e criação do banco de dados ao primeiro acesso do aplicativo. Foram implementadas todas as operações SQL necessárias. Na Figura 27 abaixo, por exemplo, é apresentada a operação de consulta das aferições. O código completo se encontra no GitHub (ANDRADE; SOUZA, 2021).

Figura 27 - Trecho do código de operação de consulta das aferições

```
if (dataInicio == null || dataFim == null) {
  maps = await db.query(
    'tb_afericao',
    where: 'id_animal = ?',
    whereArgs: [idAnimal]
  );
  // print(maps.toString());
} else {
  maps = await db.query(
    'tb_afericao',
    where: 'id_animal = ? AND data_horario >= ? AND data_horario <= ?',
    whereArgs: [idAnimal, dataInicio, dataFim]
  );
  // print(maps.toString());
}
```

Fonte: Os autores.

6 Discussões e Resultados

A implementação do circuito, sua montagem e codificação foram concluídas com êxito. Contudo, devido a um problema elétrico sem uma causa conhecida, os sinais vitais não puderam ser capturados durante o funcionamento do sensor. Por uma casualidade, descobriu-se que o sensor passava a captar os sinais vitais quando a ponta de um resistor avulso entrava em contato com os pinos SCL e SDA do MAX30100 na placa que eram realizadas as conexões. Os autores fizeram um estudo referente ao mau funcionamento do circuito. Pensou-se na possibilidade do problema estar relacionado ao pino INT do MAX30100 em conexão com o ESP-01. A presença de poucas conexões no ESP-01 impossibilitou testes mais aprofundados com o pino INT no circuito criado.

7 Conclusão

No Brasil existem milhões de *pets*, muitas vezes considerados como membros da família. Observando este setor extremamente aquecido atualmente, os autores tiveram uma ideia de uma solução voltada à saúde animal e à comodidade dos tutores e veterinários. Os autores tiveram como objetivo principal analisar se uma solução específica seria viável, para tanto eles conseguiram criar um protótipo que simulasse o monitoramento dos sinais vitais dos *pets* e que apresentasse em um aplicativo o *status* do animal e as aferições realizadas. Com as técnicas de Engenharia de *Software* e o conhecimento aprendido no decorrer do curso de graduação o objetivo foi alcançado com êxito. Uma solução específica para o mercado *pet* é viável de ser desenvolvida de fato.

Neste projeto foi demandado um investimento particular dos autores para aquisição dos componentes eletrônicos e aprofundamento técnico de tecnologias de desenvolvimento *mobile.*

Um dos maiores desafios do projeto foi aprender e manusear circuitos elétricos e *hardware*. No sensor MAX30100 por exemplo, houve um problema elétrico que dificultou a efetiva captura dos sinais vitais dos *pets.*

Para projetos futuros que utilizem os mesmos componentes deste trabalho é importante se atentar ao problema elétrico discutido, buscando novos componentes ou novas formas de realizar a implementação, de forma a obter uma solução plausível para o problema.

Em uma solução de alto nível para o mercado seria interessante que os dados fossem armazenados em um serviço de nuvem para que o sistema esteja disponível para os usuários em qualquer dispositivo a qualquer momento. A nuvem armazenaria um grande volume de dados, e seria necessária uma estrutura de *big data* para o adequado funcionamento do sistema e também para um processo de *data science* a fim de obter *insights* valiosos para o mercado *pet* a partir dos dados agrupados.

Referências

ANDRADE, Saulo Sousa; SOUZA, Ana Laura Alvino de. **Repositório contendo documentação e código-fonte realizados para o projeto**. Disponível em: <https://github.com/AnaLauraA/coleira-inteligente>. Acesso em 16 out. 2021

ARDUINO. **Arduino:** About. 2018. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em 29 set. 2021.

CORREIO BRAZILIENSE. **Amizade entre cães e homens começou há 30 mil anos e influenciou a evolução.** 2011. Disponível em: <https://www.correiobraziliense.com.br/app/noticia/ciencia-e-saude/2011/10/26/interna_ciencia_saude,275638/amizade-entre-caes-e-homens-comecou-ha-30-mil-anos-e-influenciou-a-evolucao.shtml>. Acesso em 09 mar. 2021.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de banco de dados**. Tradução: Daniel Vieira. 6. ed. São Paulo: Pearson Addison Wesley, 2011. Título original: Fundamentals of database systems.

FLUTTER. **Flutter:** FAQ. [S.D]. Disponível em: <https://flutter.dev/docs/resources/faq>. Acesso em 19 set. 2021.

FRITZING. **Fritzing:** Home. [S.D]. Disponível em: <https://fritzing.org/>. Acesso em 29 set. 2021.

GAMMA, Erich et al. **Elements of reusable object-oriented software**. Reading, Massachusetts: Addison-Wesley, 1995.

GARVIN, David A. What Does “Product Quality” Really Mean?. **Sloan Management Review**, Cambridge, v. 26, n. 1, p. 25-45, Fall 1984.

IBGE. Presença de animais. *In:* IBGE. **Pesquisa nacional de saúde 2019:** informações sobre domicílios, acesso e utilização dos serviços de saúde. Rio de Janeiro: IBGE, 2020. p. 25-26.

IBM. **Visualizações de Banco de Dados**. 2021. Disponível em: <https://www.ibm.com/docs/pt-br/mam-saas/7.6.0.6?topic=structure-views>. Acesso em 15 out. 2021.

KIRK, Andy. **Data Visualization: a successful design process**. Packt publishing LTD, 2012.

MARTINS, Rachel. **Mesmo com a pandemia, mercado pet cresce no Brasil e no Espírito Santo.** A Gazeta, 2021. Disponível em: <https://www.agazeta.com.br/colunas/rachel-martins/mesmo-com-a-pandemia-mercado-pet-cresce-no-brasil-e-no-espirito-santo-0321>. Acesso em 10 mar. 2021.

MERCADO LIVRE. **Sensor Frequência Batimentos Cardíacos Max30100 Oxímetro.** [S.D]a. Disponível em: <https://produto.mercadolivre.com.br/MLB-

1285051389-sensor-frequncia-batimentos-cardiacos-max30100-oximetro-_JM>. Acesso em 14 out. 2021.

MERCADO LIVRE. **Adaptador Usb Serial Ttl Para Wifi Esp8266 Esp-01 Ch340g.** [S.D]b. Disponível em: <https://produto.mercadolivre.com.br/MLB-1112486006-modulo-wifi-esp-01-esp8266-porta-serial-arduino-_JM>. Acesso em 14 out. 2021

MERCADO LIVRE. **Adaptador Usb Serial Ttl Para Wifi Esp8266 Esp-01 Ch340g.** [S.D]c. Disponível em: <https://produto.mercadolivre.com.br/MLB-1772679631-adaptador-usb-serial-ttl-para-wifi-esp8266-esp-01-ch340g-_JM>. Acesso em 14 out. 2021.

MQTT. **MQTT:** The Standard for IoT Messaging. 2020. Disponível em: <https://mqtt.org>. Acesso em 19 set. 2021.

ORACLE. **Banco de dados:** O Que É um Banco de Dados?. 2021a. Disponível em: <https://www.oracle.com/br/database/what-is-database/>. Acesso em 13 out. 2021.

ORACLE. **Business Analytics:** O Que é Visualização de Dados?. 2021b. Disponível em: <https://www.oracle.com/br/business-analytics/what-is-data-visualization/>.Acesso em 15 out. 2021.

PALADINI, Edson Pacheco. **Gestão da Qualidade** - Teoria e Prática. 3. ed. São Paulo: Atlas, 2012.

PMI. **Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK®)**. 5. ed. Newtown Square: Project Management Institute, Inc, 2013.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Software Engineering**: a practitioner's approach. 9th ed. New York: McGraw-Hill Education, 2020.

SNA. **Pets são considerados membros da família por 61% dos donos, diz pesquisa.** 2017. Disponível em: <https://www.sna.agr.br/pets-sao-considerados-membros-da-familia-por-61-dos-donos-diz-pesquisa/>. Acesso em 10 mar. 2021.

SOARES, Dimitre Braga. **Animais de Estimação e Direito de Família.** IBDFAM, 2009. Disponível em: <https://ibdfam.org.br/artigos/531/Animais+de+Estimação+e+Direito+de+Família>. Acesso em 09 mar. 2021

SOMMERVILLE, Ian. **Engenharia de Software**. Tradução: Luiz Cláudio Queiroz. 10. ed. São Paulo: Pearson Education do Brasil, 2019. Título original: Software Engineering.

SOMMERVILLE, Ian; SAWYER, Pete. **Requirements Engineering:** A Good Practice Guide. Chichester: John Wiley & Sons Ltd, 1997.

SQLITE. **SQLite:** About. [S.D]. Disponível em: <https://www.sqlite.org/about.html>. Acesso em 19 set. 2021.

WHITE, Elicia. **Making Embedded Systems:** Design Patterns for Great Software. Sebastopol: O'Reilly Media, Inc, 2011.

WIEGERS, Karl; BEATTY, Joy. **Software Requirements.** Redmond: Microsoft Press, 2013.

YUAN, Michael. **Conhecendo o MQTT:** Por que o MQTT é um dos melhores protocolos de rede para a Internet das Coisas?. IBM, 2017. Disponível em: <https://developer.ibm.com/br/articles/iot-mqtt-why-good-for-iot/>. Acesso em 12 out. 2021.