Relatório Técnico - iOS Module

VetCare

Unidade Curricular:	Computação Móvel
Data:	Aveiro, 8 de Junho de 2018
Autores:	76538: Raquel Oliveira Ramos 72099: Ana Rita Antunes Santiago
Resumo:	O VetCare é uma aplicação móvel que permite uma melhor realização dos planos de tratamentos aos animais internados em clínicas veterinárias. Estes planos de tratamentos consistem essencialmente em lembretes de medicações a administrar e tratamentos a realizar apesar de a aplicação ter outras funcionalidades também importantes e úteis para todas as clínicas veterinárias.

Índice

1 Introdução

O VetCare, aplicação móvel desenvolvida em Swift 4.1 pelo grupo, vem de encontro à necessidade de auxiliar nos cuidados a animais internados.

Nos dias de hoje, o que se vê essencialmente em clínicas veterinárias passa por informação em papel ou dados presentes nos computadores em softwares desenvolvidos para estas clínicas. Mas e se fosse possível haver maneira de lembrar os médicos e enfermeiros veterinários de que têm realizar determinada tarefa ou administrar determinada medicação em determinado animal sem recorrer a *post-it*? E se fosse possível identificar animais rapidamente? E se fosse possível ainda obter toda a informação do animal através de um clique rápido sem ter procurar entre enormes quantidades de papelada ou sem ter que se deslocar para ir ao computador? É aqui que surge o VetCare.

O objetivo do VetCare é essencialmente agir como lembrete e fonte de informação dos tratamentos a serem realizados e da medicação que terá que ser administrada aos animais. A aplicação desenvolvida conta também com a implementação de outras funcionalidades bastantes úteis para os médicos e enfermeiros veterinários que irão ser referidas seguidamente.

A aplicação, com versão agora em iOS, segue os mesmos moldes, com alguns ajustes, que o módulo desenvolvido previamente para Android.

2 Conceito da Aplicação

A aplicação móvel desenvolvida encontra-se orientada para todos os trabalhadores de clínicas veterinárias sejam estes médicos ou enfermeiros veterinários.

O principal cenário de uso da aplicação será para relembrar um médico ou enfermeiro veterinário acerca de tratamentos a efetuar a determinado animal ou até mesmo que medicação administrar, quando e qual a dose da mesma. Após efetuar a respetiva tarefa, o funcionário que a efetuou, consegue dar essa mesma tarefa por concluída apenas com um toque.

Um outro cenário de uso bastante crucial e pedido como requisito da aplicação móvel antes do desenvolvimento da mesma, foi a rápida identificação do animal, seja este qual for e em qualquer momento ou lugar.

Para ajudar ao principal objetivo da aplicação, ou seja, para ajudar a tratar e nomeadamente monitorizar os animais internados, é ainda possível observar os biosinais de cada animal em forma de gráfico (mais especificamente, batimentos cardíacos e temperatura) ao longo de cada dia.

3 Design da Interface Visual

View Controllers

Enquanto que em Android, cada programador pode escolher entre o uso de *Activity* ou *Fragment*, em iOS tal não acontece sendo que cada interface de utilizador (UI) que o programador desenvolve denomina-se de *View Controller*. O *Storyboard* foi o mecanismo usado para o desenvolvimento das interfaces que compõem aplicação desenvolvida.

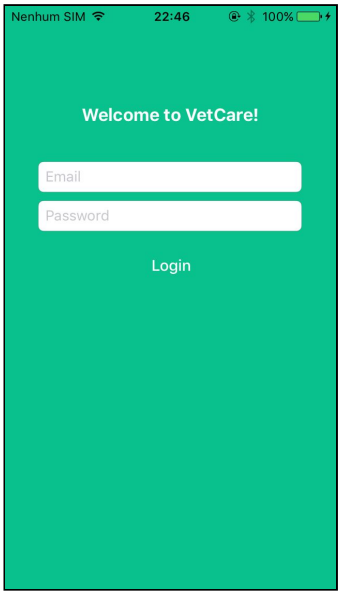

Fig. 1: Ecrã de Login

Fig. 2: Task List and Tab Bar

Layouts

Todas as UI foram construídas com o uso de *constraints* e recorrendo também ao *Auto Layout*. Ao recorrer ao *Auto Layout*, é possível que o mesmo design de uma UI para um dispositivo móvel da Apple (tal como iPhone ou iPad), seja adaptado automaticamente para os vários dispositivos adaptando-se sempre ao tamanho do ecrã do dispositivo. Desta maneira, e aplicando esta boa prática no desenvolvimento de aplicações para iOS, todas as interfaces se adaptam a qualquer dispositivo móvel.

Tab Bar

A *Tab Bar* é um elemento importante da UI principal pois permite navegar para qualquer um dos *View Controllers* onde estão implementadas as principais funcionalidades da aplicação. Como este componente é de extrema importância que seja intuitivo para o utilizador, a *Tab Bar* foi implementada com ícones e títulos intuitivos de modo ao utilizador conseguir facilmente navegar na aplicação sem necessitar de ajuda.

Fig. 3: Tab Bar

Cores

As cores usadas na aplicação tiveram em conta as típicas cores dos espaços de saúde bem como dos uniformes dos funcionários que trabalham nestes locais. As cores usadas para a aplicação em Android foram as mesmas cores usadas para a aplicação em iOS. Em Android é possível observar-se o tema, ou seja, as cores padrão escolhidas para a aplicação na seguinte na imagem. Em iOS, não existe um sítio específico onde as cores escolhidas se encontrem expostas, então para efeitos de consulta, encontra-se presente na imagem abaixo as mesmas cores (retirada do Android Studio).

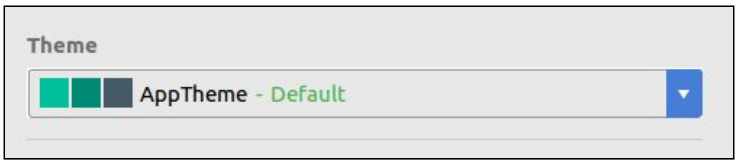

Fig. 4: Esquema de Cores

Ícones

Todos os ícones escolhidos para a UI têm como principal objetivo serem intuitivos para o utilizador ter uma boa experiência com a aplicação.

Os ícones e suas dimensões foram escolhidos de acordo com o local em que estes se encontram na UI: seja por exemplo na *Tab Bar* ou na *Navigation Bar*. Na Figura 5 é possível observar-se o ícone da aplicação que se enquadra no propósito da mesma. É possível ainda observar-se os ícones da *Tab Bar* na Figura 6.

Fig. 5: Ícone da Aplicação

Fig. 6: Ícones da Tab Bar

Botões

Ao contrário do que acontecia em Android, em iOS não é aconselhável o uso dos *Floating Action Buttons* logo para os botões foram usados botões simples seja em forma de ícone ou texto.

Fig. 7: Botão em forma de Texto

Fig. 8: Botões em forma de Ícone

4 Arquitetura da Solução

Diagrama

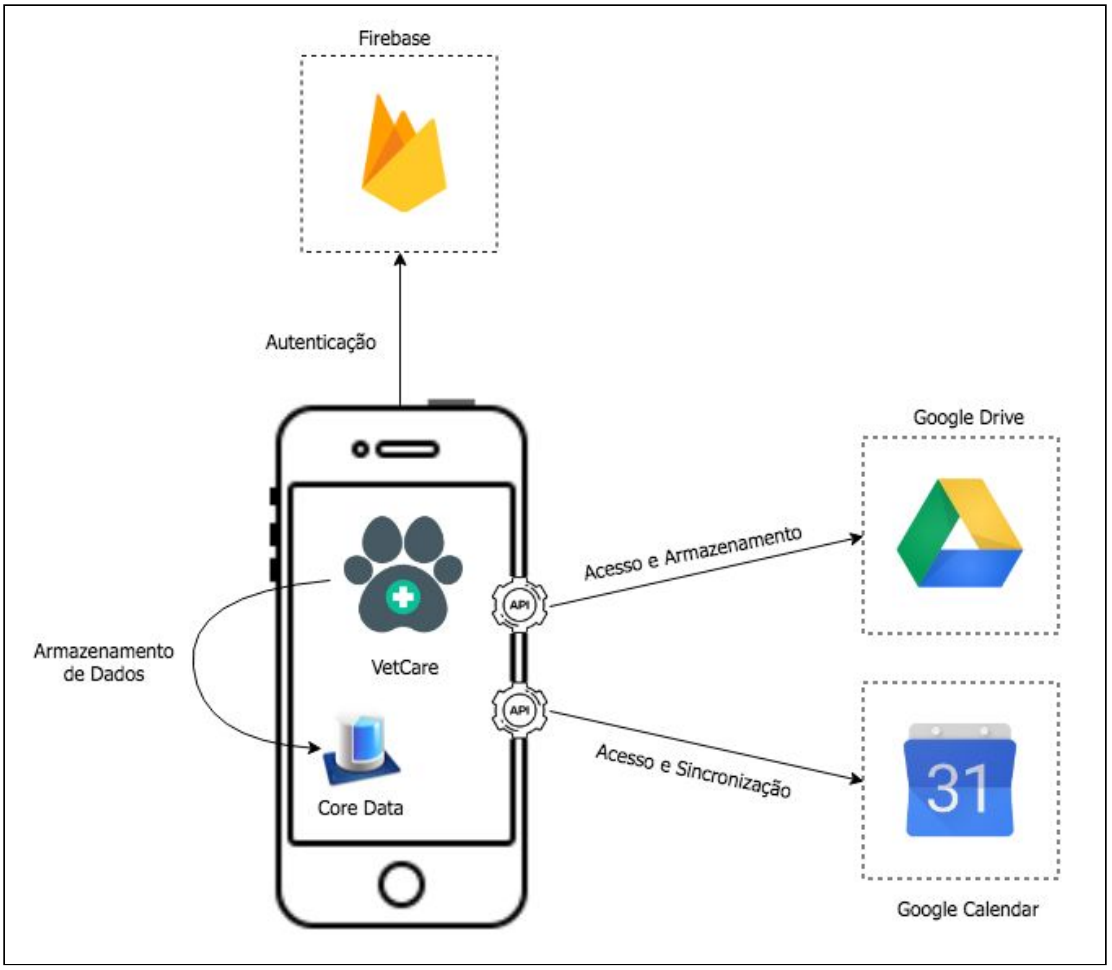

Fig. 9: Arquitetura

A imagem apresentada acima representa a arquitetura global do projeto desenvolvido pelo grupo. É possível observar-se a aplicação móvel VetCare no centro que usa para autenticação dos utilizadores a plataforma Firebase (parte superior do diagrama) onde estão definidos os utilizadores com acesso à aplicação. Em análise ao diagrama, pode-se ainda verificar que a aplicação móvel comunica com duas API: Google Drive API e Google Calendar API (lado direito do diagrama). Posteriormente irá ser explicado o porquê de se ter implementado o acesso a estas duas API. Como base de dados da aplicação, foi implementado o uso de *Core Data*.

Modelos de Dados

Na aplicação móvel desenvolvida, estão envolvidos 6 tipos de dados diferentes. Estes tipos de dados são:

- **Animal**: informações do animal quando este dá entrada na clínica veterinária como por exemplo o seu nome, uma imagem identificadora do mesmo e o peso quando deu entrada;

- **Owner**: informações acerca do dono nomeadamente número de telemóvel caso seja necessário contactar o mesmo;
- **Internment**: informações básicas do internamento, como por exemplo o motivo pelo qual foi à clínica e que médico veterinário viu o animal na sua entrada;
- **Procedure**: procedimentos realizados ao animal enquanto este se encontra internado;
- **Medicine**: medicamentos administrados ao animal enquanto este se encontra internado;
- **Historic**: procedimentos de rotina realizados em consultas prévias ao seu internamento na clínica;

Os tipos de dados e relações entre os mesmos encontram-se ainda representados na imagem seguinte:

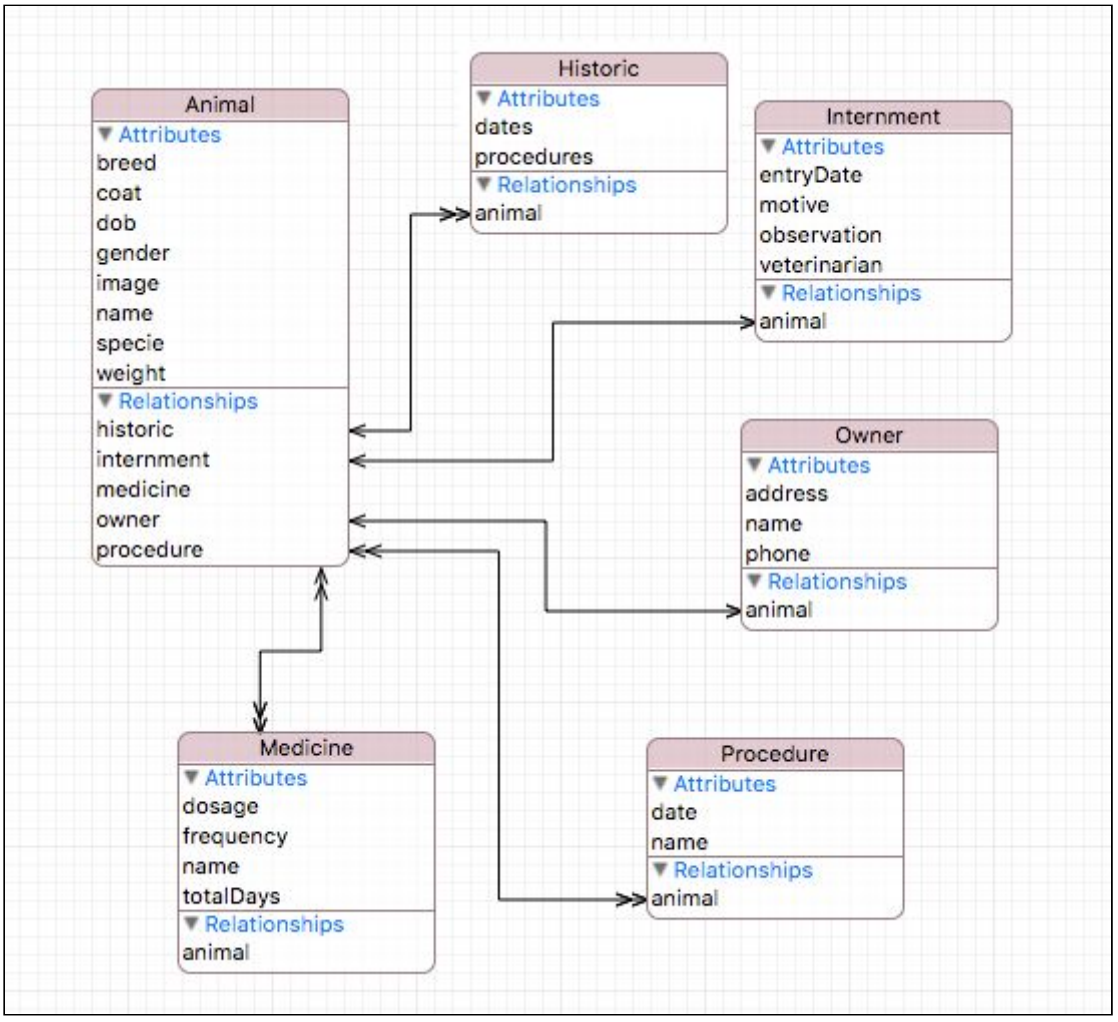

Fig. 10: Base de Dados

Back-end

A parte de autenticação foi realizada através da plataforma Firebase. Consideramos que o uso desta plataforma é de extrema importância para a nossa aplicação pois permite a autenticação de várias clínicas veterinárias.

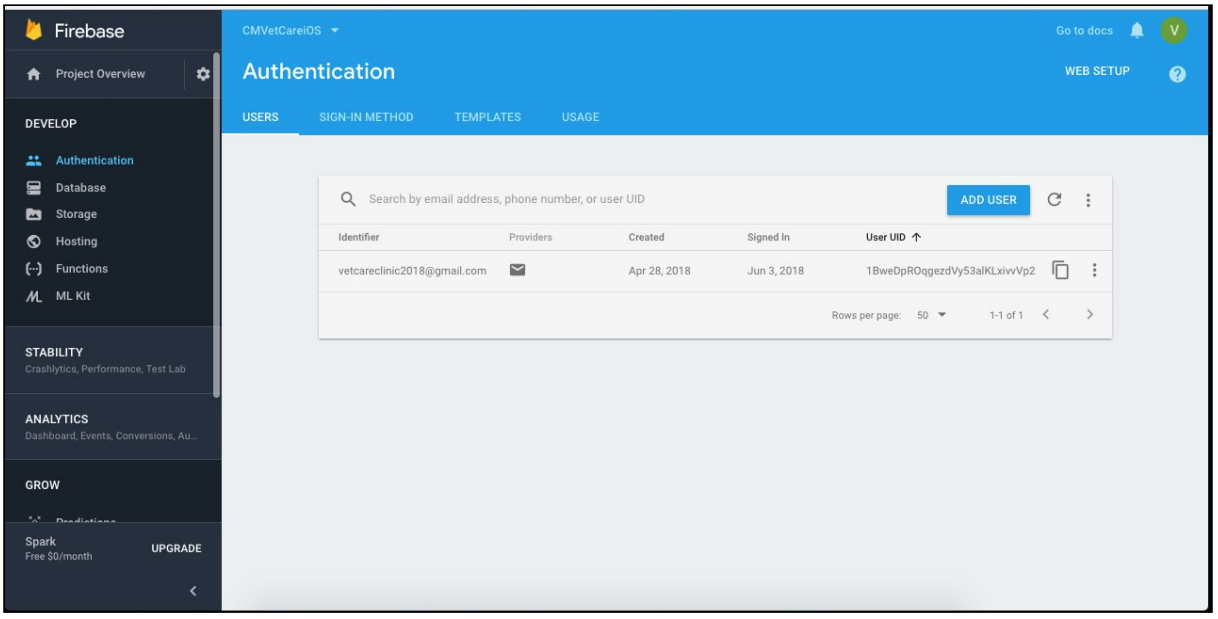

Fig. 11: Autenticação no Firebase

Persistência

Relativamente à persistência, esta parte foi garantida através do uso do *Core Data*. Enquanto que no módulo de Android foi usada a plataforma Firebase para autenticação e armazenamento dos dados, no módulo de iOS, o Firebase foi apenas usado para autenticação dos utilizadores. Por sugestão do professor e sendo esta uma sugestão de extrema importância pois levou ao uso e aprendizagem de novas ferramentas, bem como a à introdução de um conceito de persistência que não existia com o Firebase, foi optado o uso do *Core Data* levando à existência de uma base de dados local enquanto que com o Firebase na aplicação em Android, esta era remota.

Acesso a API

Como se pode observar no diagrama, foi feito o acesso e uso de duas API da Google: Google Calendar e Google Drive.

Hoje em dia o uso da Internet e das tecnologias existentes, encontra-se num crescimento rápido e em grande escala. Desta maneira e como uma forma de haver um fácil e rápido acesso, a partir de qualquer local, dos vários funcionários da clínica veterinária a diversas informações, implementou-se o acesso ao Google Calendar para o armazenamento de tarefas a realizar e o acesso à Google Drive onde estarão dispostos documentos de alta ou documentos com conclusões acerca de exames realizados como por exemplo, análises ao sangue. Deste modo, em qualquer altura e em qualquer lugar, desde que haja acesso à Internet, qualquer funcionário da clínica veterinária consegue ter acesso a estas informações, seja para consulta ou possíveis alterações, não estando as mesmas

apenas associadas a armazenamento local de ficheiros ou armazenamento das tarefas no calendário do telemóvel.

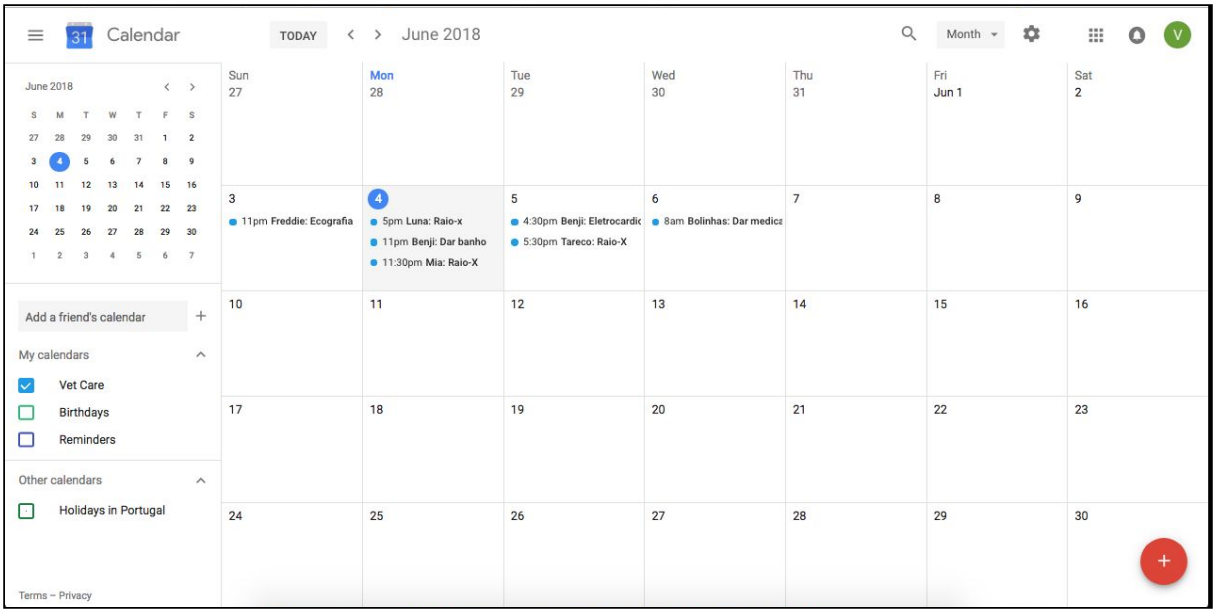

Fig. 12: Google Calendar

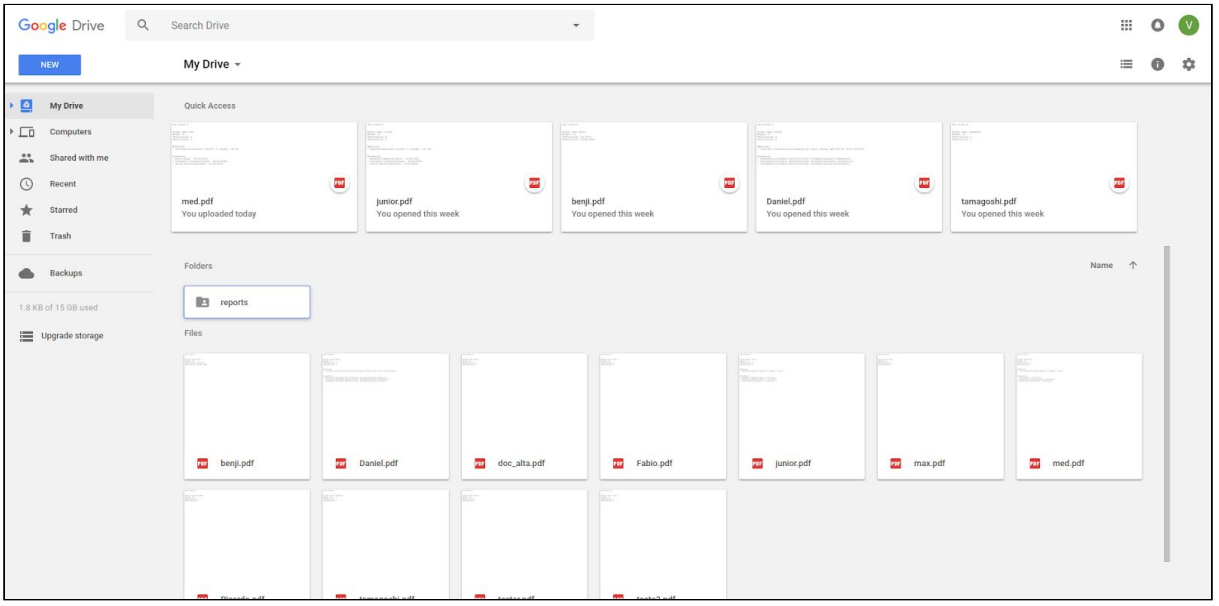

Fig. 13: Google Drive

Mecanismos de Notificações

Notificações são uma boa maneira de manter o utilizador a par do que a aplicação vai realizando. No VetCare, consideramos que uma boa maneira de manter o utilizador a par de algumas ações que acontecem na aplicação, seria através de *Alerts*.

Relativamente às tarefas a realizar, com o uso da API do Google Calendar, encontra-se implementado a recepção de notificações quando está a chegar a hora de

realizar determinada tarefa. Esta era sem dúvida uma das funcionalidades que tinha que ser implementada pois o objectivo principal da aplicação era ajudar na realização dos cuidados aos animais internados, e foi implementada com sucesso.

Outro tipo de notificações que se encontra implementado, é o aparecimento de um *UIAlertController* (bastante comum em iOS) no centro da UI do *View Controller* em questão assim que se identifica um animal. Esta funcionalidade será descrita no capítulo acerca da solução implementada. Este tipo de *Alert* é ainda mostrado em outras situações.

Fig. 14: Notificações do Calendário

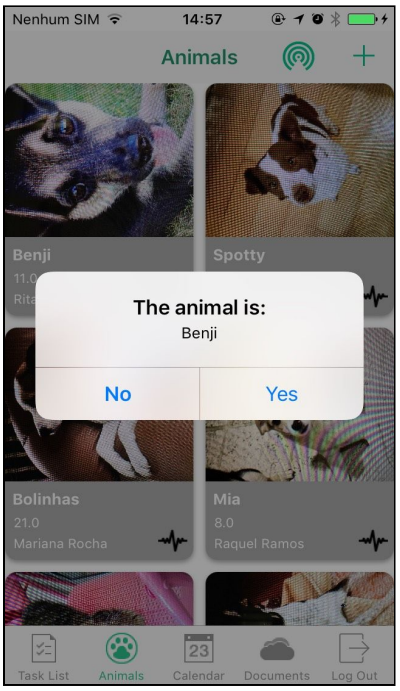

Fig. 15: UIAlertController

5 Solução Implementada

Organização do Código

Quando se desenvolve um projeto, todo o código deve estar organizado. Não só por uma questão de organização mas sim porque se por exemplo, o projeto não for desenvolvido apenas individualmente, uma boa organização do código será vital para que outras pessoas envolvidas no projeto, consigam encontrar os ficheiros que pretendem.

Desta maneira, os ficheiros *Swift* foram agrupados em diretórios consoante a sua funcionalidade e o "local" onde se encontram. A organização do código pode ser observada na seguinte imagem. Relativamente aos diretórios de código:

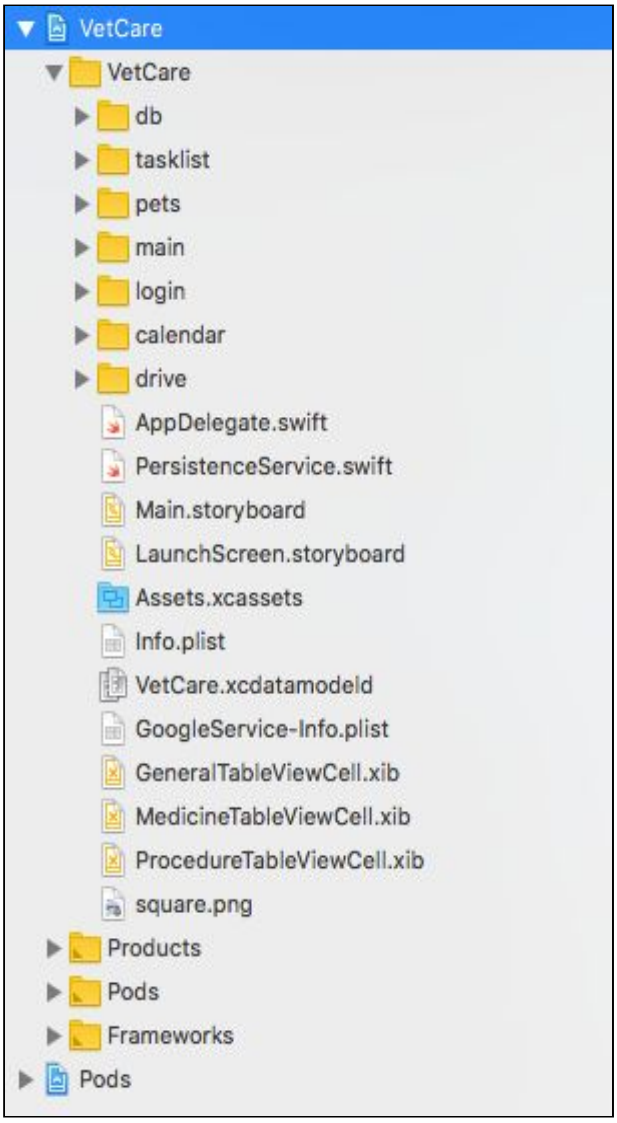

Fig. 16: Organização do Projeto

- **db**: diretório onde se encontram definidas as várias entidades do projeto e suas propriedades;
- **tasklist**: diretório que diz respeito ao *View Controller* inicial que aparece ao utilizador quando inicia sessão na aplicação e ficheiro para leitura do QR code e check da tarefa correspondente;
- **pets**: diretório que contém todos os *View Controllers* respetivos aos animais nomeadamente o *View Controller* do perfil de cada um, entre outros;
- **main**: diretório que contém o ficheiro principal onde é definida a *UITabBarController* que irá estar presente em todos os *View Controllers* da aplicação;
- **login**: diretório que contém o *View Controller* para login e o ficheiro que efetua o logout, ou seja, o *View Controller* que diz respeito ao ecrã inicial da aplicação onde é feito o login da clínica veterinária e o código desenvolvido para logout;
- **calendar**: diretório onde se encontra o *View Controller* que tem definido o calendário (sincronizado com o Google Calendar) com todos os eventos, sejam do presente dia ou futuros;
- **drive**: diretório que contém o ficheiro onde é feito o acesso à Google Drive.

Para além dos diretórios, na figura anterior é ainda possível observar-se alguns ficheiros e diretórios importantes, e que devem ser referidos, para a aplicação desenvolvida:

- **AppDelegate.swift**: ficheiro a partir de onde a aplicação é iniciada. No caso da aplicação VetCare, é ainda o ficheiro onde se inicializa e configura a autenticação com o Firebase;
- **PersistenceService.swift**: ficheiro que lida com a persistência e serve de suporte ao uso do Core Data;
- **Main.storyboard**: ficheiro correspondente ao *Storyboard* da aplicação, ou seja, todas as UI desenvolvidas para a aplicação que formam a "história" da mesma;
- **Assets.xcassets**: diretório que contém os ícones usados na aplicação bem como o ícone da aplicação gerado para vários tamanhos sendo assim adaptado para vários dispositivos móveis com diferentes tamanhos de ecrã;
- **Info.plist**: ficheiro que contém uma lista de configurações para a execução da aplicação;
- **VetCare.xcdatamodeld**: ficheiro correspondente ao modelo dos dados;
- **Pods**: bibliotecas usadas durante o desenvolvimento do projeto. Este tópico será abordado posteriormente no relatório.

Funcionalidades Implementadas

Desde o dia da escolha do projeto, que foi definida a história da aplicação e as funcionalidades a implementar. Com a ajuda de ambos os professores, algumas funcionalidades foram sofrendo alterações de modo a ficarem melhores para a experiência do utilizador seja em termos visuais ou em termos técnicos.

A primeira funcionalidade implementada foi o Login. Isto é possível realizar-se com a autenticação disponibilizada pelo Firebase em que são definidas credenciais válidas que podem ser usadas para efetuar a autenticação.

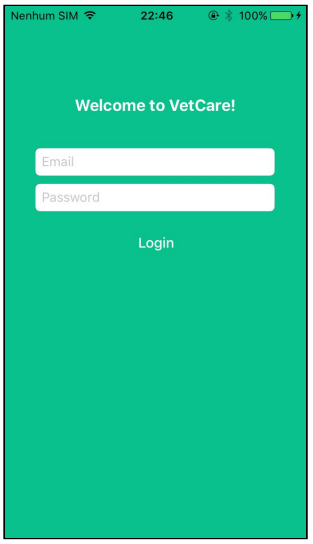

Fig. 17: Ecrã de Login

Após o utilizador fazer o login, o primeiro *View Controller* a ser disposto ao mesmo, será a *Task List*. Esta lista de tarefas irá conter todas as tarefas do dia que determinado funcionário deverá realizar. A cada tarefa estará associada uma hora, um animal e a tarefa a realizar. Após a realização da tarefa, o funcionário em questão que a realizou, dispõe de um leitor de QR code implementado que ao ler-se um QR code associado a um animal (QR code que contém como informação o nome do animal cuja tarefa vai ser realizada), dará um "check" na tarefa associada a esse mesmo animal. Para que o utilizador aceda a esse leitor, existe um ícone na *Navigation Bar* (cuja imagem é um QR Code) que o utilizador deverá premir, levando a que o leitor seja inicializado. Para se atualizar a lista de tarefas, como esta tem que fazer acesso à API do Google Calendar para atualização das tarefas, o utilizador terá que fazer um pequeno deslize com o seu dedo em direcção ao fundo do ecrã do dispositivo móvel, levando a aparecer um ícone de *refresh* (implementação de um *UIRefreshControl*) cuja ação associada a este movimento, é um novo acesso à API para ir buscar os eventos atualizados.

Fig. 18: Task List

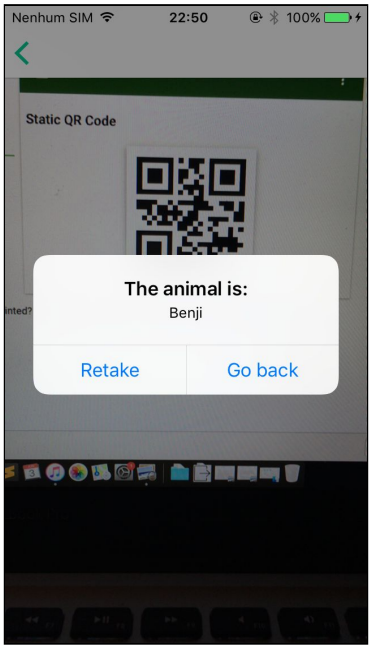

Fig. 19: Leitor QR Code

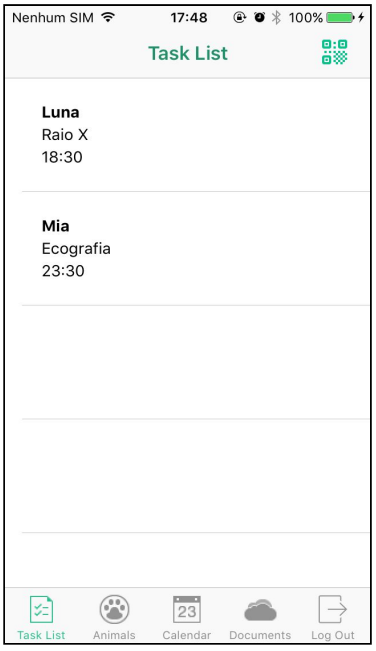

Fig. 20: Alerter

Uma das principais e mais importantes funcionalidades implementadas foi a rápida identificação dos animais. Esta funcionalidade foi implementada recorrendo a *beacons* BLE (Bluetooth Low Energy). Foi assumido que quando o animal dá entrada na clínica veterinária, é-lhe fornecida uma coleira com um *beacon* incorporado na sua coleira que contém como informação interna um UUID. Assim, sempre que um funcionário pretender identificar um animal rapidamente, basta aproximar-se com o seu dispositivo móvel, pressionar o botão para leitura de beacons (assinalado na figura 22) e será apresentado um *UIAlertController* no centro da UI com a identificação do animal. Esta funcionalidade foi simulada com um telemóvel a simular um *beacon* BLE pois não foi possível a utilização de um *beacon* real. Sendo assim, num telemóvel foi instalada uma pequena aplicação que simula um beacon: “Beacon Simulator” por Vincent Hiribarren. Para que isto seja possível, ao animal tem que lhe ser atribuído o UUID do *beacon* que irá usar na coleira para que seja possível associar o UUID com o nome do animal.

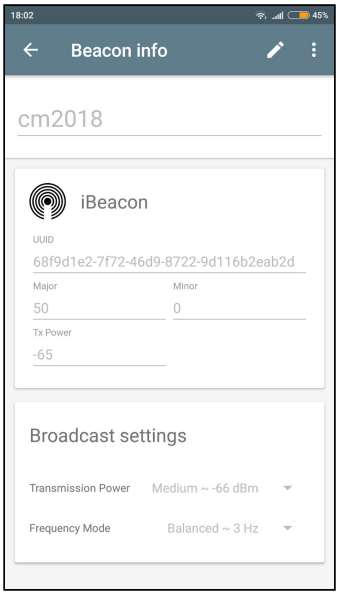

Fig. 21: Simulador Beacon BLE

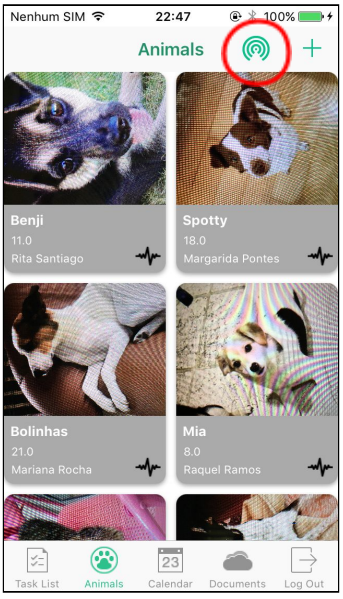

Fig. 22: Ícone para leitura do Beacon

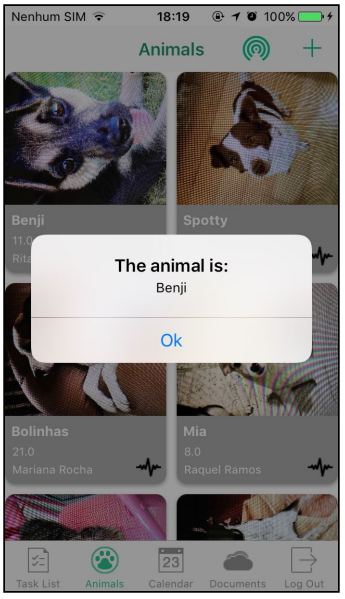

Fig. 23: UIAlertController

Com a *Tab Bar,* é possível navegar entre todos os *View Controllers* implementados. O próximo *View Controller* que vamos analisar é o dos *Animals*. Este *View Controller* é possivelmente um dos mais importantes pois contém toda a informação relativa aos animais internados. Ao navegar para o *View Controller* dos animais (*Animals* na *Tab Bar*), é apresentado uma *UICollectionView* com todos os animais e algumas informações. Ao carregar-se num *card*, o utilizador é redireccionado para o perfil do animal em questão. Caso no *card* seja pressionado o ícone dos sinais vitais, o utilizador é redireccionado para um *View Controller* que apresenta um gráfico dos sinais vitais do animal nas últimas 24 horas com duas variáveis: batimentos cardíacos e temperatura.

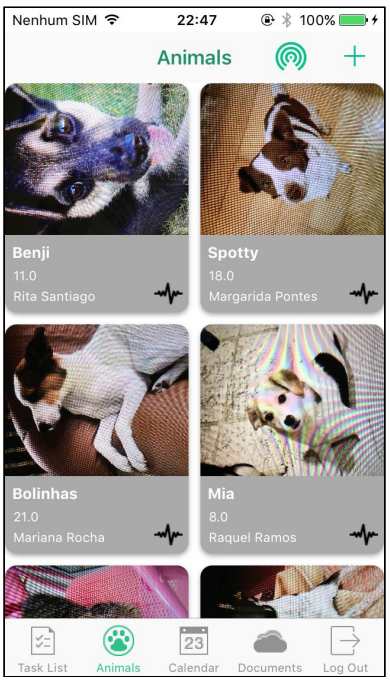

Fig. 24: Lista de Animais

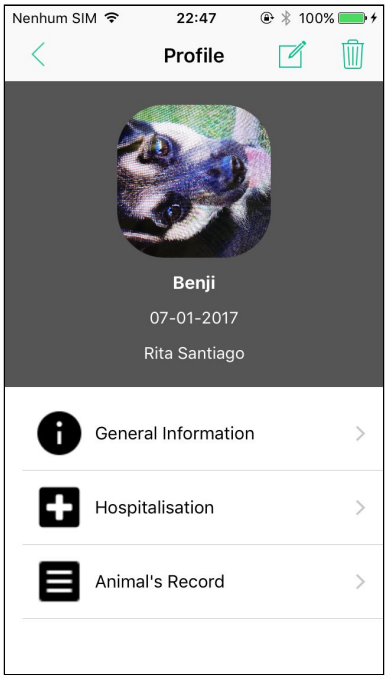

Fig. 25: Perfil do Animal

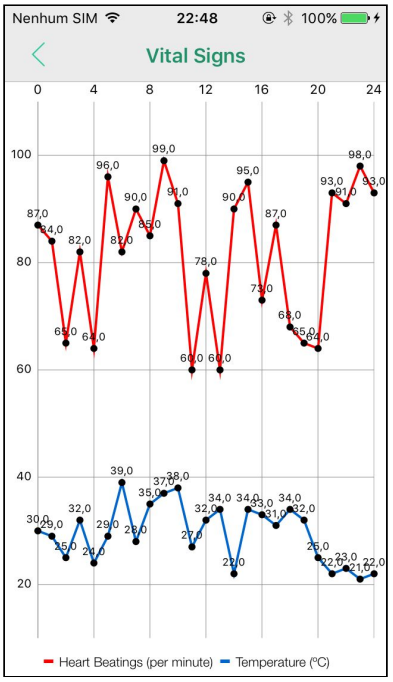

Fig. 26: Sinais Vitais

Dentro do perfil de cada animal, podem ser observadas 3 categorias principais: **General Information**, **Hospitalisation** e **Animal's Record**. O *View Controller* respectivo à General Information dispõe de todas as informações básicas acerca do animal em questão.

Fig. 27: Informação Geral

Ainda dentro do perfil é possível aceder-se ao *View Controller* da Hospitalisation. Neste *View Controller* é possível observar-se 3 segmentos: **General**, **Medication** e **Procedures**. No segmento *General*, encontram-se informações respectivas ao dia em que o animal deu entrada na clínica veterinária. No segmento *Medication* encontra-se informação acerca de medicamentos que o animal se encontra a tomar. E no último segmento, o segmento dos *Procedures*, encontram-se todos os procedimentos já realizados pelo animal desde que este foi internado. Estes segmentos foram implementados através de um *UISegmentedControl*.

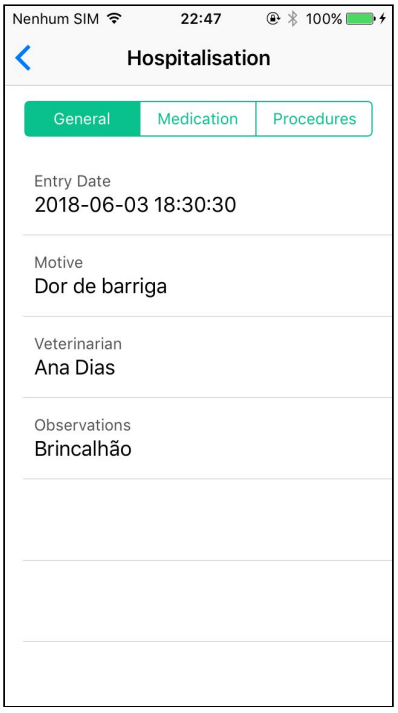

Fig. 28: Informação de Entrada

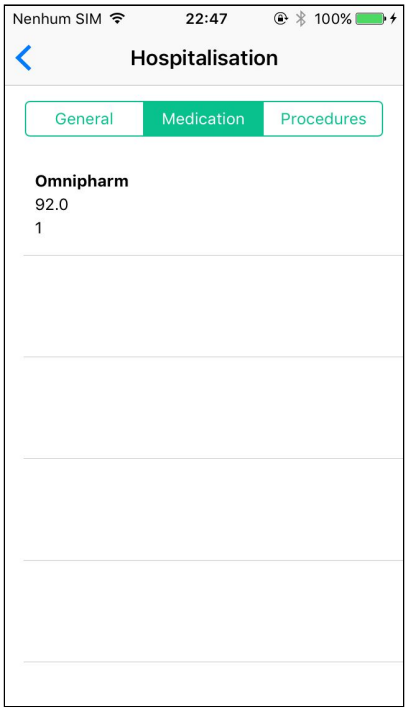

Fig. 29: Medicação

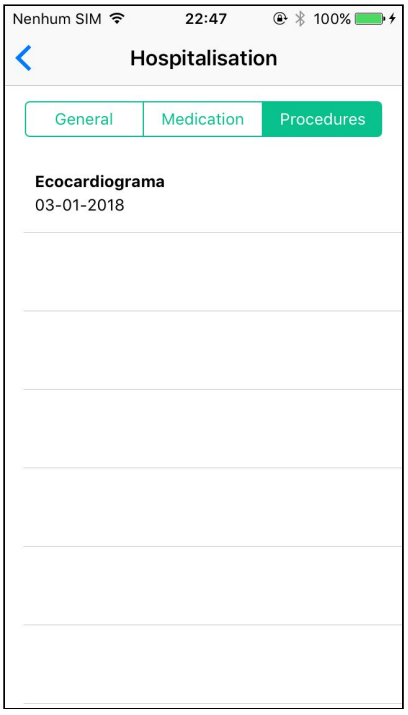

Fig. 30: Procedimentos

Voltando ao perfil do animal, é possível ainda ver-se mais informação acerca do mesmo, pressionando a secção *Animal's Record*. Este clique levará o utilizador para o *View Controller* onde se encontra disposta informação acerca de consultas prévias do animal e nomeadamente que procedimentos foram realizados nessas consultas.

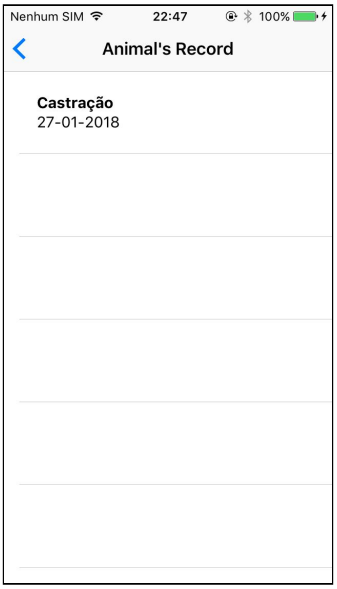

Fig. 31: Histórico

No perfil do animal, o utilizador tem ainda ao seu dispor dois botões: um botão com um ícone de escrita de mensagem (comum em iOS) e um botão cujo ícone é um caixote do lixo. Ambos aparecem na *Navigation Bar* do *View Controller* em questão. Quando o primeiro é premido, encaminha o utilizador para um novo *View Controller* onde pode ser escrita uma mensagem. Este botão e este novo *View Controller* trazem ao utilizador a possibilidade de poder mandar mensagens para os donos dos animais (sem ter que inserir o número de telemóvel do dono pois esta informação é obtida diretamente do *Core Data*) para ir dando actualizações do estado do animal ao dono ou para por exemplo, avisar após um procedimento, que este correu bem e o animal se encontra melhor. Esta funcionalidade passa pela aplicação de Mensagens da Apple. A mensagem para o dono é escrita no *View Controller* e ao premir o botão de enviar, é aberta a aplicação de Mensagens default do iOS, já com a mensagem escrita proveniente da aplicação e o número de telemóvel proveniente do *Core Data* (número de telemóvel do dono do animal em questão).

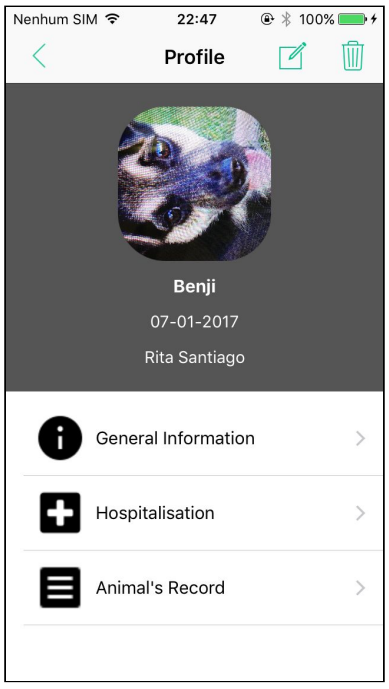

Fig. 32: Perfil do Animal

Fig. 33: Envio de Mensagem

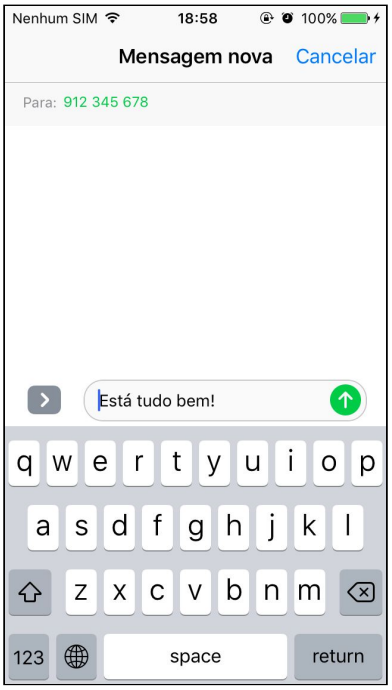

Fig. 34: Aplicação das Mensagens do iOS

Relativamente ao segundo botão que já foi referido e que tem o ícone de um caixote do lixo, este botão servirá para dar alta a um animal. Ora portanto, assim que este botão é premido, o utilizador irá ser reencaminhado para um *View Controller* onde deverá preencher algumas informações: o peso com que o animal saiu da clínica (pois quando os animais estão internados e se encontram a tomar medicação e a realizarem procedimentos, existe tendência para uma mudança significativa no peso), o nome do veterinário que deu a alta e possíveis observações. Assim que neste *View Controller* for premido o botão “Generate PDF” que se encontra a meio da UI, será gerado um documento PDF com o nome do animal, as informações preenchidas no *View Controller* e os medicamentos administrados e procedimentos realizados durante o internamento. Este documento será armazenado na Google Drive.

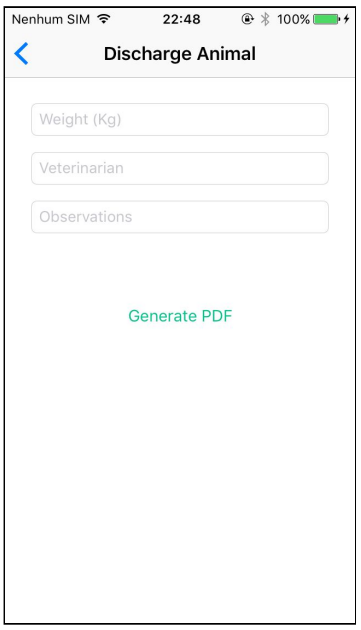

Fig. 35: Informação para o PDF

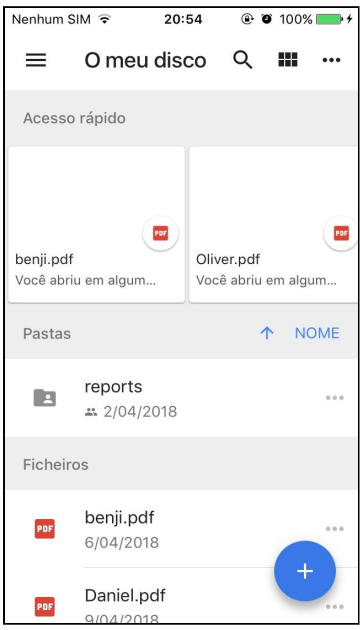

Fig. 36: Google Drive

Para além de todas as funcionalidades ligadas aos animais individualmente, na lista inicial onde estão dispostos todos os animais internados, é possível observar-se um botão no campo superior direito com o ícone de "+" cuja funcionalidade do mesmo é adicionar um novo animal. Quando se efetua um clique neste botão, o utilizador é reencaminhado para um novo *View Controller* com vários campos *TextField* que terão que ser preenchidos de modo a um novo animal ser adicionado à lista de animais internados. Para além do preenchimento dos campos, é também necessário tirar-se uma foto ao animal, foto esta que será disposta na lista de animais e no perfil de cada animal. Após o preenchimento dos campos e a captura da foto, o utilizador tem um botão no canto superior direito da UI, na *Navigation Bar*, com o texto "Save" sendo que este botão ao ser premido levará o utilizador a adicionar o novo animal. Para além das informações básicas sobre o mesmo, é ainda necessário preencher: o motivo pelo qual o animal foi internado, que médico o internou e observações do animal como por exemplo se pode morder ou tem alguma alergia, ou outra observação importante que deve ser estritamente necessário os funcionários da clínica saberem.

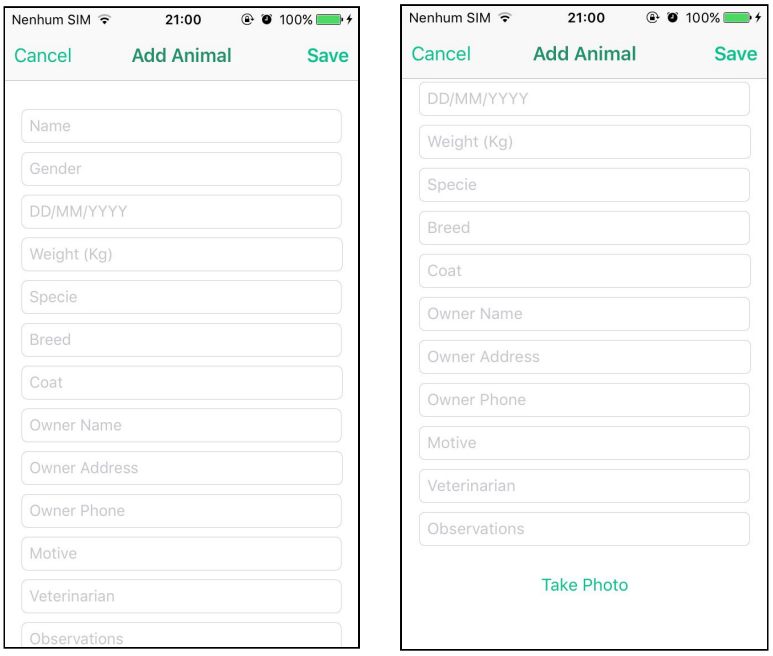

Fig. 37 e 38: Adicionar Animal

Voltando à *Tab Bar*, irá ser agora abordado um novo *View Controller*: *Calendar*. O utilizador pode navegar para este *View Controller* carregando no item "Calendar" na *Tab Bar*. Este *View Controller* é composto por um calendário com indicação do mês e ano por cima do mesmo e por baixo do calendário é possível observar-se uma *UITableView* com as tarefas a realizar no dia em que o utilizador clicar. Neste calendário é possível observar-se as tarefas do presente dia, tarefas de dias anteriores e futuras tarefas. O calendário disposto encontra-se ainda sincronizado com o Google Calendar sendo que quem possuir os dados da conta associada à clínica veterinária, consegue consultar em qualquer local e em qualquer altura (desde que tenha acesso à Internet), todas as tarefas associadas à clínica seja pela aplicação ou pelo Google Calendar.

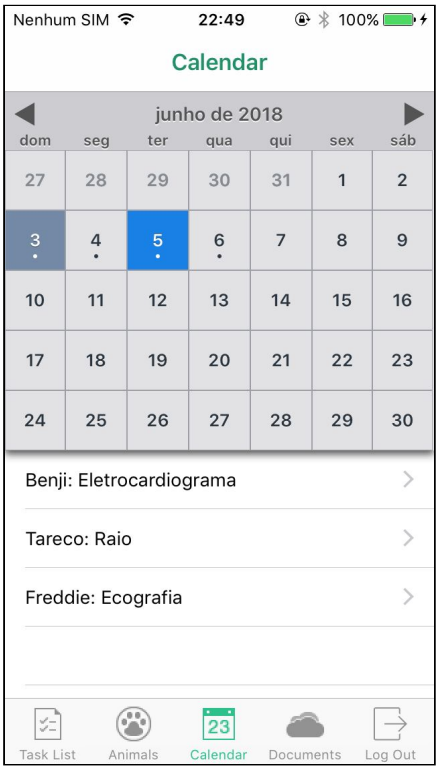

Fig. 39: Calendário

Na *Tab Bar*, existe um item denominado *Documents* e com o ícone da Cloud. Este item tem como objetivo que, ao ser pressionado, o utilizador seja reencaminhado para a Google Drive onde estão guardados os mais variados documentos acerca dos animais que atualmente se encontram internados ou já estiveram.

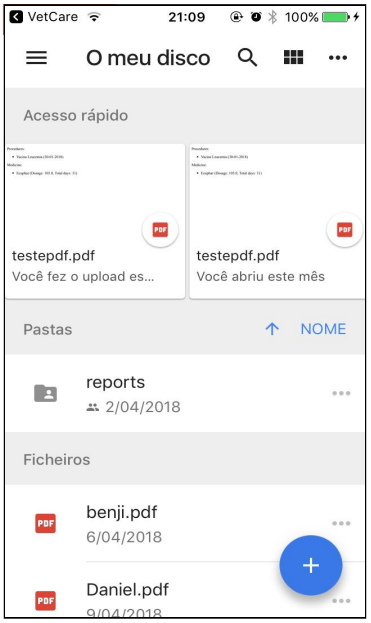

Fig. 40: Acesso à Drive pelo VetCare

Na *Tab Bar* existe ainda a possibilidade de fazer logout levando o utilizador a ser reencaminhado para o ecrã de login da aplicação.

Uma outra funcionalidade implementada relativamente ao login e logout através do Firebase, foi a possibilidade de a sessão do utilizador poder ficar guardada em vez de este andar sempre a inserir as suas credenciais. Se o utilizador não terminar sessão, o seu ID de sessão permanecerá guardado. Sempre que a aplicação é iniciada, é verificado se existe alguma sessão já iniciada: caso exista, a aplicação passa logo para o *View Controller* da *Task List*. Caso não haja sessão iniciada, o *View Controller* que será demonstrado ao utilizador, é o do Login.

Pods

Foram usadas diversas bibliotecas durante o desenvolvimento do projeto.

Ao contrário do que foi feito em Android, em iOS as bibliotecas foram sendo adicionadas ao projeto através do Podfile. As bibliotecas usadas foram:

- MBCalendarKit: usada para visualização do calendário no *View Controller* do Calendar e inserção de eventos no mesmo através da sincronização de eventos com o Google Calendar;
- Charts: usada para a geração dos gráficos que contém os valores dos sinais vitais os animais;
- TPPDF: usada para a geração dos PDF's com pedaços de texto.

O link para qualquer uma destas bibliotecas encontra-se no capítulo das Referências.

Core Data

O *Core Data* foi o mecanismo usado para a implementação da base de dados na aplicação VetCare. Ao contrário do que foi feito no módulo de Android, em que foi usado o Firebase para armazenar os dados da aplicação, no módulo de iOS optou-se pelo uso do *Core Data* como sugestão do professor e usando assim uma base de dados local em vez de ser remota levando ao conceito de persistência que se encontra agora presente na nossa aplicação.

Core Data é o M de MVC, ou seja, corresponde à camada do *Model* na aplicação. O *Core Data* em si não é uma base de dados, é muito mais que isso. É também conhecido por "*object graph manager*" pois um *object graph* é uma coleção de objetos que se conectam/relacionam com os outros e é o *Core Data* quem vai gerir essas relações e vai gerir ainda os ciclos de vida dos objectos.

Inicialmente foram definidas todas as entidades, seus atributos e o tipo dos atributos como foram especificados por alto no capítulo da Arquitetura do projeto. Como o *Core Data* permite estabelecer relações entre entidades, foram definidas relações *One-To-One* e *One-To-Many*. As relações foram as seguintes:

- **Animal** ↔ **Owner**: *One-To-One* considerando que cada dono apenas tem um animal;
- **Animal** ↔ **Internment**: *One-To-One* pois o registo de internamento é apenas um por animal enquanto estes se encontram internados;
- **Animal** ↔ **Medicine**: *One-To-Many* considerando que os animais podem tomar mais do que um medicamento;
- **Animal** ↔ **Historic**: *One-To-One* pois cada animal contém um histórico de procedimentos que já realizou na clínica veterinária;
- **Animal** ↔ **Procedure**: *One-To-Many* pois o animal pode realizar vários procedimentos enquanto se encontra internado.

Para que o ficheiro *AppDelegate.swift* não ficasse extenso, foi desenvolvido um ficheiro *Swift* com o nome *PersistenceService.swift* que lida com a persistência do *Core Data*. O *NSPersistentContainer* neste ficheiro consiste num conjunto de objectos que facilitam o armazenamento e a recuperação de informação do *Core Data*. Mesmo que se faça um *fetch* a determinados dados, se os mesmos forem armazenados num array, o array ficará com os dados armazenados em memória mas se a aplicação for fechada antes de se fazer um *PersistenceService.saveContext()*, estando o conceito de persistência aqui presente, os dados não irão estar presentes quando se voltar a iniciar a aplicação.

Para se aceder à base de dados, isto é feito através de um *NSFetchRequest* à entidade sobre a qual se pretende pesquisar dados. Após isso, guardam-se os dados por exemplo num array e é com esse array que se vão mostrar os dados por exemplo dos animais na *UICollectionView* definida no *View Controller* do item "Animals" na *Tab Bar*. Uma parte importante do Core Data é que por exemplo quando se adiciona um novo animal, é preciso voltar a fazer um *NSFetchRequest* ao mesmo pois a visualização de novos dados não é instantânea sendo esta parte uma desvantagem. O uso do *Core Data* na aplicação é essencialmente feito a 3 níveis:

- Visualização de dados por exemplo relativamente aos animais;
- Adição de dados quando se adiciona um novo animal que foi internado;
- Eliminação de dados quando um animal tem alta da clínica veterinária.

Algumas diferenças que se podem agora evidenciar em relação ao Firebase e ao *Core Data* são as seguintes:

- O *Core Data* é uma boa ferramenta para o armazenamento de dados podendo criar relações entre os mesmos, o que não acontece com o Firebase apesar de este último ser vantajoso quando se pretende que vários utilizadores tenham acesso aos mesmos dados sendo que assim, estes dados são armazenados online;
- O *Core Data* armazena objetos com características específicas e estes objetos podem ser relacionados o que é similar a uma base de dados SQL enquanto que no Firebase quer com o Firestore ou RTDB (Realtime Database), estas são bases de dados não relacionais sendo que o acesso a relações entre objetos se torna mais complicado quando se pretende aceder a informação.

Como conclusão desta comparação, achamos que esta comparação acaba por não ser "apropriada" pois ambas as ferramentas trazem diferentes vantagens e desvantagens para as aplicações móveis. Dependendo muito do propósito da aplicação, pode ser escolhido o Firebase ou o *Core Data* (no caso de ser uma aplicação para iOS).

O conceito de persistência foi pedido pelo professor como funcionalidade indispensável da aplicação e consideramos que esta funcionalidade foi implementada com sucesso.

6 Conclusão

Os objetivos do projeto foram concluídos com sucesso. Todas as funcionalidades pensadas inicialmente e também outras que foram sugeridas pelo professor posteriormente, ficaram implementadas.

Foi um trabalho que envolveu bastante pesquisa e bastante aprendizagem de novos conhecimentos num curto espaço de tempo mas que no fim revelou ser muito gratificante para ambos os elementos do grupo de trabalho pois concluímos que aprendemos bastante. Alguns dos maiores problemas foi a inclusão da API do Google Calendar e da Google Drive pois muita da documentação está desatualizada ou não se encontra muito explícita e esta documentação não ajudou muito levando a um elevado número de horas necessário para se conseguir implementar algo com estas API pois não se encontrava muita informação atual sobre as mesmas. Outro dos grandes problemas foi a geração dos documentos PDF pois não foi fácil ter o acesso à Google Drive para armazenamento. Outro problema sentido foi através do QR Code conseguir dar uma tarefa por concluída e conseguir transmitir para o Google Calendar a sincronização em como determinada tarefa já tinha sido concluída. Por último, o uso do Core Data foi também uma dificuldade algo elevada mas ultrapassada com sucesso. A utilização de uma ferramenta como o Core Data a que o grupo não estava habituado, ferramenta esta que era desconhecida, e que permite aplicar o conceito de persistência como o Firebase não permite, é uma mais valia para uma aplicação. De um modo geral, foi um projeto trabalhoso mas muito gratificante no final e que levou a uma grande aprendizagem de várias tecnologias.

A unidade curricular de Computação Móvel na componente de iOS tem uma boa organização, os conteúdos são bem leccionados e o grau de satisfação dos elementos do grupo com a unidade curricular é bastante elevada.

7 Referências e Recursos

Human Interface Guidelines
Google Drive API
Google Calendar API
Charts
MBCalendarKit
TPPDF

Summary of project resources:
Project resources for the iOS module:

- Code repository: http://code.ua.pt/projects/cm2018_vetcare_ios