POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Arrived

Geo-based Notificator App

Mestrado em Engenharia Informática e Computação Móvel

Luana de Paula Nalon

Leiria, Março de 2023

POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Arrived

Geo-based Notificator App

Mestrado em Engenharia Informática e Computação Móvel

Luana de Paula Nalon (2202299)

Trabalho de Projeto realizado sob a orientação da

Professora Doutora Sónia Maria Almeida da Luz

Leiria, Março de 2023

Originalidade e Direitos de Autor

O presente relatório de projeto é original, elaborado unicamente para este fim, tendo sido devidamente citados todos os autores cujos estudos e publicações contribuíram para a/o elaborar.

Reproduções parciais deste documento serão autorizadas na condição de que seja mencionada a Autora e feita referência ao ciclo de estudos no âmbito do qual a mesma foi realizado, a saber, Curso de Mestrado em Engenharia Informática – Computação Móvel, no ano letivo 2021/2022, da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria, Portugal, e, bem assim, à data das provas públicas que visaram a avaliação destes trabalhos.

Agradecimentos

Ao finalizar essa etapa tão importante da minha vida, não poderia deixar de expressar o meu profundo agradecimento à minha professora orientadora Sónia Luz por toda a disponibilidade, dedicação, orientação prestada e compreensão que sempre manifestou. Quero agradecer também a todos que, de alguma forma, contribuíram para a realização deste projeto. Em especial a minha mãe Geralda que sempre acreditou e lutou para que eu tivesse essa oportunidade, ao meu namorado, David, que esteve sempre presente a incentivar-me. Aos meus amigos e família que sempre me apoiaram nesta caminhada, a empresa VOID Software, que me deu a oportunidade de começar a minha carreira profissional nesta área e aos meus colegas de trabalho, que me aconselharam da melhor maneira para conseguir alcançar o objetivo final.

Resumo

Atualmente, os compromissos e obrigações da vida moderna podem frequentemente resultar em esquecimentos que comprometem negativamente a produtividade, a eficiência pessoal e até mesmo em situações de risco para a segurança pessoal. Essa dificuldade em lembrar de tarefas importantes ou até mesmo em notificar pessoas próximas sobre chegadas e saídas de locais específicos é um problema comum enfrentado por muitas pessoas.

Com a crescente dependência da tecnologia hoje em dia, a possibilidade de aproveitar ao máximo as novas ferramentas de localização é cada vez mais relevante. A tecnologia *geofencing*, em particular, oferece um grande potencial para melhorar a interação das pessoas com o ambiente e com aqueles ao seu redor. Além disso, diante da necessidade de gerir eficientemente as atividades diárias, as aplicações que utilizam *geofencing* podem ser uma solução eficaz para otimizar o controlo de tarefas relacionadas com a localização.

A ideia deste projeto surgiu através de uma proposta por parte da empresa VOID Software, na sequência da realização de um estágio profissional. O presente projeto propõe uma solução que aborda os problemas apresentados através do desenvolvimento de uma aplicação nativa iOS denominada Arrived, que utiliza a tecnologia *geofencing* para fornecer uma solução confiável e fácil de usar para criação de avisos e lembretes baseados em localização.

Através desta aplicação, os utilizadores poderão receber notificações relevantes quando chegam ou saem de um local específico, bem como enviar avisos para pessoas próximas. A aplicação também pretende resolver o problema de esquecimento de tarefas importantes e a falta de notificações relevantes com base em localização, proporcionando eficiência e produtividade no dia a dia das pessoas. Por fim, pretende-se com o presente relatório, refletir e explicar a solução desenvolvida, bem como os resultados obtidos, com o seu desenvolvimento.

Palavras-chave: *geofencing*, aplicação móvel, produtividade, notificação, avisos, lembretes

Abstract

Currently, the commitments and obligations of modern life can often result in forgetfulness that negatively impacts productivity, personal efficiency, and even personal safety risks. This difficulty in remembering important tasks or even notifying close people about arrivals and departures from specific locations is a common problem faced by many individuals.

With the growing dependence on technology nowadays, the possibility of making the most of new location-based tools is becoming increasingly relevant. Geofencing technology, in particular, offers great potential for improving people's interaction with the environment and those around them. Furthermore, given the need to efficiently manage daily activities, applications that use geofencing can be an effective solution to optimize the control of location-related tasks.

The idea for this project arose through a proposal from the company VOID Software, following a professional internship. This project proposes a solution that addresses the problems presented through the development of a native iOS application called Arrived, which uses geofencing technology to provide a reliable and easy-to-use solution for location-based alerts and reminders.

Through this application, users can receive relevant notifications when they arrive or leave a specific location, as well as send alerts to nearby individuals. The application also intends to solve the problem of forgetting important tasks and the lack of relevant location-based notifications, providing efficiency and productivity in people's daily lives. Finally, this report aims to reflect and explain the developed solution, as well as the results obtained from its development.

Keywords: geofencing, mobile application, productivity, notification, alerts, reminders

Índice

Lista de Figuras

Lista de tabelas

Lista de siglas e acrónimos

AES	Advanced Encryption Standard
API	Application Programming Interface
CAGR	Copound Annual Growth Rate
ESTG	Escola Superior de Tecnologia e Gestão
GIF	Graphics Interchange Format
GPS	Global Positioning System
IA	Inteligência Artificial
JSON	JavaScript Object Notation
MEI-CM	Mestrado em Engenharia Informática e Computação Móvel
MMS	Multimedia Messaging Service
MVC	Model-View-Controller
MVVM	Model-View-View-Model
QA	Quality Assurance
RFID	Radio Frequency Identification
SaaS	Software as a Service
SMS	Short Message Service
UI	User Interface
URL	Uniform Resource Locator
US	User Stories
UX	User Experience

1. Introdução

O presente relatório pretende demonstrar o trabalho desenvolvido no âmbito do Projeto de Mestrado em Engenharia Informática e Computação Móvel (MEI-CM) na Escola Superior de Tecnologia e Gestão (ESTG) do Politécnico de Leiria. Este projeto surgiu através de uma proposta realizada por parte da empresa VOID Software [1], na sequência da realização de um estágio profissional. Este projeto insere-se no tema de gestão *offline* de avisos com base em localização através do uso da tecnologia *geofencing* [2], com recurso a *Short Message Service* (SMS) e *frameworks* ligadas ao desenvolvimento *mobile* de aplicações iOS nativas, tendo como objetivo o desenvolvimento de uma aplicação móvel denominada Arrived.

Com o surgimento das tecnologias de localização, a possibilidade de criar aplicações que utilizam *geofencing* tem se tornado cada vez mais comum. A Arrived é uma aplicação de notificação com base nesta tecnologia que permite que o utilizador crie avisos que servirão para o lembrar de avisar pessoas próximas que chegou ou saiu de uma determinada localização recorrendo a SMS. O utilizador também poderá fazer um pedido de aviso a outro utilizador da Arrived, que ao receber esse pedido via SMS, poderá criar esse aviso no seu dispositivo, que passará a notificá-lo sempre que se aproximar ou sair da localização pedida. Assim ele poderá informar ao remetente desse pedido que já saiu ou chegou ao destino.

Outra vertente de notificações com base em localização permitida pela Arrived é a criação de lembretes para o próprio utilizador, para que ele nunca se esqueça das suas tarefas que dependem de localização, como ir buscar o jantar ao seu restaurante favorito, quando sair do trabalho, ou não se esquecer de comprar a medicação que a sua avó pediu ao se aproximar da farmácia. A gestão de todos os avisos criados é feita na aplicação de forma *offline* e de modo a não depender de Application Programming Interfaces (APIs) externas. Essas funcionalidades são apresentadas ao utilizador através de uma *interface* simples e atraente, para que a necessidade de pessoas com diferentes idades e necessidades seja alcançada.

Por fim, a aplicação Arrived surge no mercado para atender a uma necessidade bastante comum: a de comunicar a chegada ou saída de um local de forma rápida e eficaz ou até mesmo auxiliar na rotina que é por vezes exigente ao ajudar o utilizador a lembrar-se de

tarefas que necessitam ser realizadas em determinados locais. A nível profissional e pessoal, surgiu a oportunidade de explorar estes temas e os seus respetivos conceitos, os quais atualmente ainda não se encontram totalmente abordados no percurso académico, permitindo a aquisição de competências tecnológicas atuais e relevantes para o meu percurso profissional.

1.1. Enquadramento e problema

Um dos principais problemas que a Arrived pretende resolver é o de comunicação entre pessoas, principalmente quando necessitam informar umas às outras sobre terem chegado ou saído de uma determinada localização, ao reduzir a perda de tempo e aumentar a produtividade. Com a Arrived é possível que uma pessoa seja notificada via SMS ou iMessage quando a outra chegar ao local combinado, sem precisar fazer chamadas telefónicas, ou escrever mensagem explícita. Isso pode ser especialmente útil para diversos casos, tais como:

- Para ajudar na segurança de crianças e idosos: A aplicação pode ser utilizada para acompanhar a localização de crianças e idosos, emitindo avisos quando eles chegam ou saem de determinados locais, como escola, casa de familiares, ou até mesmo lares;
- Esquecimento de tarefas ou compromissos que precisam ser realizados em determinados locais: com a criação de lembretes baseados em localização, é possível que a aplicação notifique a pessoa quando ela estiver num determinado local, lembrando-a de uma tarefa ou compromisso que ela precisa realizar. Isso pode ser bastante útil para pessoas com uma agenda muito ocupada e que precisam de se lembrar de várias tarefas ao longo do dia;
- Coordenação em grupo: quando um grupo de pessoas precisa se encontrar num local específico, pode ser difícil coordenar a chegada de cada pessoa. Com a Arrived, uma pessoa pode enviar um aviso para outras pessoas quando ela chegar ao local de encontro. Isso torna a coordenação muito mais fácil e eficiente;
- Perda de objetos: se uma pessoa é propensa a perder objetos pessoais, como chaves ou carteira, a aplicação Arrived pode ajudar a lembrar de levá-los ao sair de casa ou de outros locais, emitindo avisos para lembrá-la.

1.2. Motivação e objetivos

O projeto Arrived tem dois objetivos:

1. Enquadramento teórico e levantamento do estado da arte relativamente a tecnologia *geofencing* e aplicações que, semelhante à aplicação Arrived, utilizam esta tecnologia;
2. Desenvolvimento de uma aplicação iOS nativa, que alavanque o potencial da tecnologia *geofencing* onde é possível a criação de avisos, pedidos de avisos e de lembretes com base em localização. Esta aplicação deve ter uma gestão *offline* e local dos mesmos, sem utilização de APIs, recorrendo a SMS e preferencialmente uso de *frameworks* nativas iOS que em conjunto irão permitir chegar à solução pretendida;

O resultado do estudo que implica o primeiro objetivo está descrito nos primeiros capítulos deste relatório e o segundo objetivo implica recorrer ao resultado do estudo previamente efetuado em conjunto com a ideia proposta pela entidade de acolhimento deste projeto, a empresa VOID Software, para possibilitar o desenvolvimento de uma solução, de modo a superar os desafios inerentes ao seu propósito.

Em suma, o objetivo da Arrived é ser uma solução inovadora para a comunicação de chegadas e partidas, bem como para a criação de lembretes pessoais baseados em localização e gestão local de todos os dados gerados. Com a utilização da tecnologia de *geofencing* e uma *interface* simples e agradável, a Arrived tem o intuito de oferecer uma experiência ao utilizador conveniente e intuitiva, ao permitir que a gestão dos seus avisos e lembretes seja feita diretamente da aplicação Arrived, sem a necessidade de alternar entre várias aplicações ou plataformas.

Por fim, é importante destacar que faz parte dos propósitos da Arrived garantir a privacidade e a segurança dos utilizadores, ao assegurar que os seus dados pessoais e informações de localização sejam armazenados e geridos de forma segura e protegida, sem a necessidade de depender de serviços externos, APIs ou até de uma conexão constante com a *internet*. Como podemos verificar, a Arrived é uma aplicação que promete facilitar a vida de muitas pessoas quando se trata de melhorar a produtividade e manter as pessoas próximas informadas da localização do utilizador.

1.3. Geofencing

Geofencing [2] é uma tecnologia que recorre ao Global Positioning System (GPS), Radio Frequency Identification (RFID), antenas de telemóveis e até mesmo sinais de Wi-fi

para definir limites geográficos. Este recurso permite a configuração de *triggers* para que quando um dispositivo entrar ou sair dos limites de um determinado perímetro, um alerta seja emitido. Os limites virtuais dessa cerca geográfica podem ser ativos ou passivos, sendo que os ativos exigem que o utilizador final ative os serviços de localização através de uma aplicação móvel. Já os passivos estão sempre ativos e trabalham em segundo plano; por isso, utilizam recursos como o Wi-fi e antenas de celulares, em vez de GPS ou RFID.

Esta tecnologia está ligada a diversas áreas, com variados usos práticos [3]. Por exemplo, quando um dispositivo móvel, que pertença a uma organização, seja este um portátil ou um *tablet*, se afastar uma certa distância do terreno da organização, um alerta poderá ser configurado por um administrador de rede para que quando isso acontecer ele possa desabilitar e apagar os arquivos confidenciais do dispositivo. Outra área onde se aplica frequentemente o uso de *geofencing*, é a do *marketing*, permitindo que um profissional possa usá-la numa loja de um centro comercial e enviar automaticamente um cupão digital para clientes que tenham a aplicação da loja, quando o dispositivo do cliente se aproximar de uma zona predefinida.

Outro exemplo onde o *geofencing* pode ser aplicado é na automação de casas ou empresas [4], com diversas possibilidades de utilização, como, por exemplo, uma pessoa que more numa zona bastante fria, poderá configurar o seu sistema de aquecimento para que este sistema seja ativado, quando se deslocar para casa e estiver a uma determinada distância, de modo a que no momento em que chegar a casa, a mesma já esteja numa temperatura considerada adequada.

Como podemos observar, o uso de *geofencing* é bastante predominante em diversas áreas e o seu uso vem crescendo nos últimos anos, considerando-se que a sua popularidade é devida à sua versatilidade. De acordo com um levantamento realizado no Markets And Markets [5], esse mercado movimentou US$458,3 milhões em 2016, e previa um aumento de 27.5% até 2022. Segundo o Technavio [5] o mercado de *geofencing* está projetado para crescer uma taxa de crescimento anual composta (CAGR) de 22,35% entre 2022 e 2027, o volume do mercado deverá aumentar em US$ 3.024,7 milhões, mas o seu crescimento depende de fatores de serviço, incluindo o aumento do *marketing* baseado em localização, aumento da procura por serviços de navegação interna e crescente procura por rastreamento de ativos.

Esta tecnologia foi fundamental para o funcionamento da aplicação Arrived porque permitiu desenvolver a sua principal funcionalidade que é a criação de avisos baseados em localização. Sempre que um aviso é criado, uma localização e um raio são definidos para a cerca geográfica e conforme as definições desse aviso, poderá ser lançada uma notificação ao utilizador ao chegar ou ao sair dessa cerca geográfica. O que poderá ser útil para diversas situações, desde um lembrete para avisar um familiar que chegou ao seu destino ou até mesmo para o próprio utilizador, para não se esquecer de comprar uma medicação quando passar perto da farmácia. No capítulo 5 será possível ver mais detalhes da aplicação de *geofencing* na implementação e funcionamento da aplicação Arrived.

1.4. Caracterização da entidade de acolhimento

Este projeto surgiu após nove meses de estágio profissional na empresa VOID Software [1] (ver Figura 1), por sugestão de um dos membros da administração. A caracterização que se segue foi efetuada com base em documentação interna da empresa.

Figura 1 - Logo VOID Software [1]

A VOID Software é uma empresa especializada no desenvolvimento de *software*, ativa desde 2006, e que desde então se dedica à criação de soluções adaptadas às necessidades específicas de cada desafio, com recurso a uma vasta gama de tecnologias de programação. Seguindo uma abordagem ágil centrada na personalização de cada produto conforme as necessidades individuais do cliente, privilegiando a experiência emocional do utilizador.

Com clientes localizados em diversas geografias e uma equipa de programadores experientes e dedicados, a VOID Software tem vindo a acrescentar valor nas mais variadas áreas de atuação, desde a área do *marketing*, ao sector contabilístico e financeiro, passando pelo desenvolvimento de sistemas de gestão de ativos digitais, e até à construção e manutenção de plataformas de gestão integrada de entidades públicas e privadas de grande dimensão.

Tirando partido da propriedade intelectual construída ao longo destes anos, com foco especial em áreas como a Inteligência Artificial (IA), Machine Learning e Blockchain, a

VOID Software tem vindo desde 2016 a apostar no desenvolvimento de produtos inovadores próprios ou em parceria, contando neste momento com participações em várias *startups* e consórcios de base tecnológica.

Na VOID Software são implementados serviços de *cloud*, Software as a Service (SaaS), integração de *hardware*, aplicações *web* e de dispositivos móveis. Empregam cerca de 60 colaboradores, com recursos adicionais variáveis disponíveis para fazer face às necessidades de cada caso particular, permitindo a mobilização de grupos multidisciplinares especializados em *User Interface* (UI) / *User Experience* (UX), desenvolvimento de *backend*, *frontend*, e controlo de qualidade.

Na Figura 2, é possível observar com mais detalhes estes grupos e como é feita toda a divisão dos recursos humanos através do organograma da empresa.

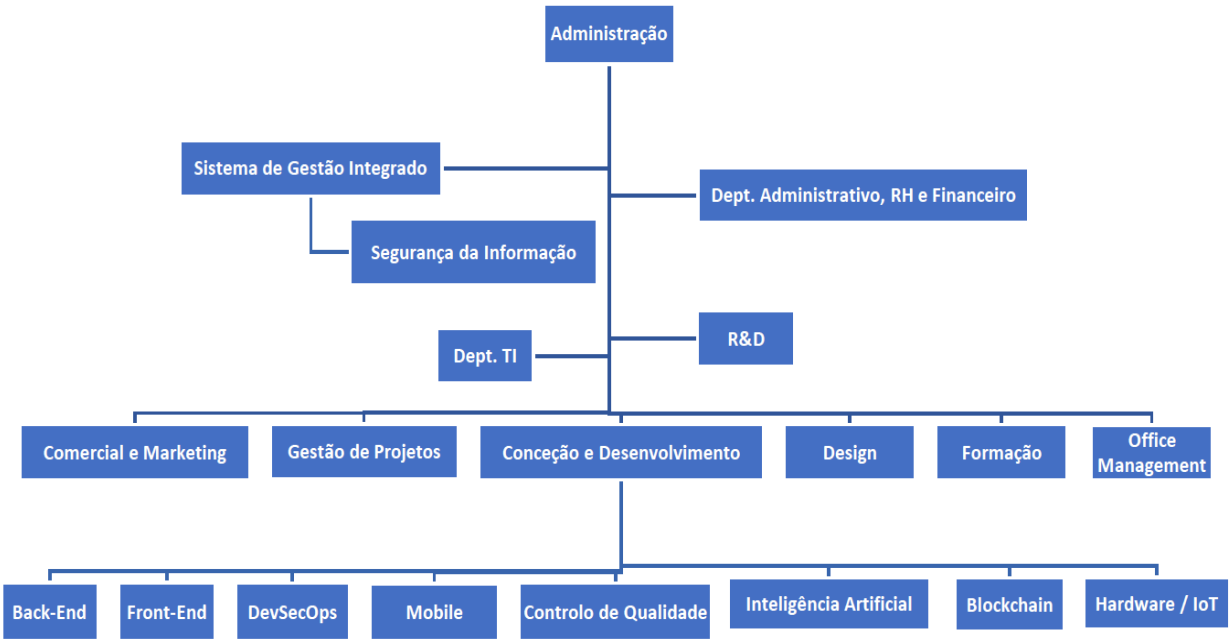

Figura 2 - Organograma da VOID Software

A missão da VOID Software é o desenvolvimento de produtos de *software* de alto nível para dar resposta aos novos desafios, utilizando equipas *agile* de alto desempenho que apresentam resultados de excelência e que desafiam diariamente os limites das possibilidades.

A VOID Software tem a visão de foco em mercados tecnologicamente amadurecidos, formação e retenção de recursos altamente qualificados em áreas tecnológicas relevantes,

parceria com a comunidade científica e contribuição para o desenvolvimento do ecossistema tecnológico e empreendedor local.

E por fim, os valores da empresa VOID Software são: colocar sempre as pessoas antes do negócio, compromisso dentro da equipa e perante os clientes, construção contínua do conhecimento através da experiência e orgulho e excelência em tudo o que fazem.

1.5. Organização do documento

O presente documento está organizado por capítulos que, de uma forma estruturada, descrevem os principais resultados obtidos ao longo do projeto. Apresenta-se, a seguir, uma breve descrição de cada capítulo:

- No Capítulo 2 - Trabalho relacionado, apresenta-se todo o estudo do trabalho relacionado a este projeto, com o foco em aplicações que, tal como a Arrived usam a tecnologia *geofencing* para o seu funcionamento;
- O Capítulo 3 - Especificações, foi escrito de forma a descrever toda a metodologia de desenvolvimento, análise de requisitos e respetivas *User Stories* (US) para chegar à solução Arrived;
- No Capítulo 4 - Estudo e configuração de tecnologias, são descritas todas as tecnologias fundamentais utilizadas no âmbito da implementação da aplicação Arrived;
- O Capítulo 5 - Arrived, contém as definições iniciais do ambiente de desenvolvimento, a apresentação da arquitetura da solução e implementação de cada funcionalidade desenvolvida com os seus respetivos testes;
- No Capítulo 6 - Conclusão, serão apresentados os objetivos alcançados, a aprendizagem obtida com a elaboração deste projeto, e ainda o estudo feito para o trabalho futuro a realizar.

2. Trabalho relacionado

Existem diversas aplicações que contam com a tecnologia *geofencing* para o seu funcionamento. Nos próximos subcapítulos será feita uma descrição das principais características analisadas para algumas dessas aplicações, tendo por base os requisitos inicialmente indicados para a aplicação Arrived. Adicionalmente será descrita uma comparação das funcionalidades analisadas com o que se pretende atingir ao desenvolver a aplicação Arrived.

2.1. Shortcuts

A aplicação Shortcuts [6] da Apple é uma ferramenta integrada ao sistema iOS que permite aos utilizadores a automação de tarefas, criação de fluxos de trabalho personalizados e o acesso rápido a funções comuns do dispositivo. É possível usar atalhos predefinidos pela aplicação ou criar personalizados (ver Figura 3 [6]) para diferentes tipos de tarefas, como enviar uma mensagem, abrir uma aplicação, criar uma nota, iniciar uma chamada telefónica ou até mesmo controlar dispositivos inteligentes em casa.

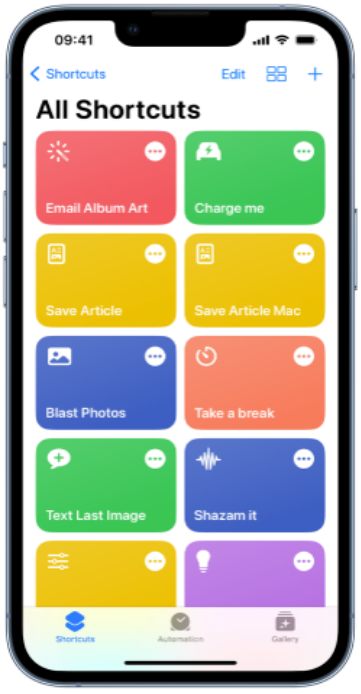

Figura 3 - Exemplos de atalhos criados na App Shortcuts [6]

Na aplicação Shortcuts, além da ação de criação de atalhos, é possível listá-los, reordená-los, editá-los, apagá-los, compartilhá-los, entre outras ações. Estes atalhos podem ser executados de diferentes formas, nomeadamente, pela própria aplicação Shortcuts onde é possível que o utilizador selecione o atalho que deseja executar, no ecrã inicial do dispositivo, via *widgets*, através do *apple watch*, através de outra aplicação e por meio de comandos de voz usando a assistente virtual Siri. Por exemplo, é possível criar um atalho personalizado que permita ao utilizador enviar uma mensagem de texto para um contacto específico dizendo "enviar mensagem".

Além dos atalhos, a aplicação Shortcuts oferece dois tipos de automações, a automação residencial e a pessoal (ver Figura 4 – adaptada de [6]). A automação residencial, fornece uma forma de executar ações da casa do utilizador com base em eventos como hora do dia, pessoas a chegar ou a sair, ou alguém a controlar um acessório, como podemos observar na segunda imagem da Figura 4 [6].

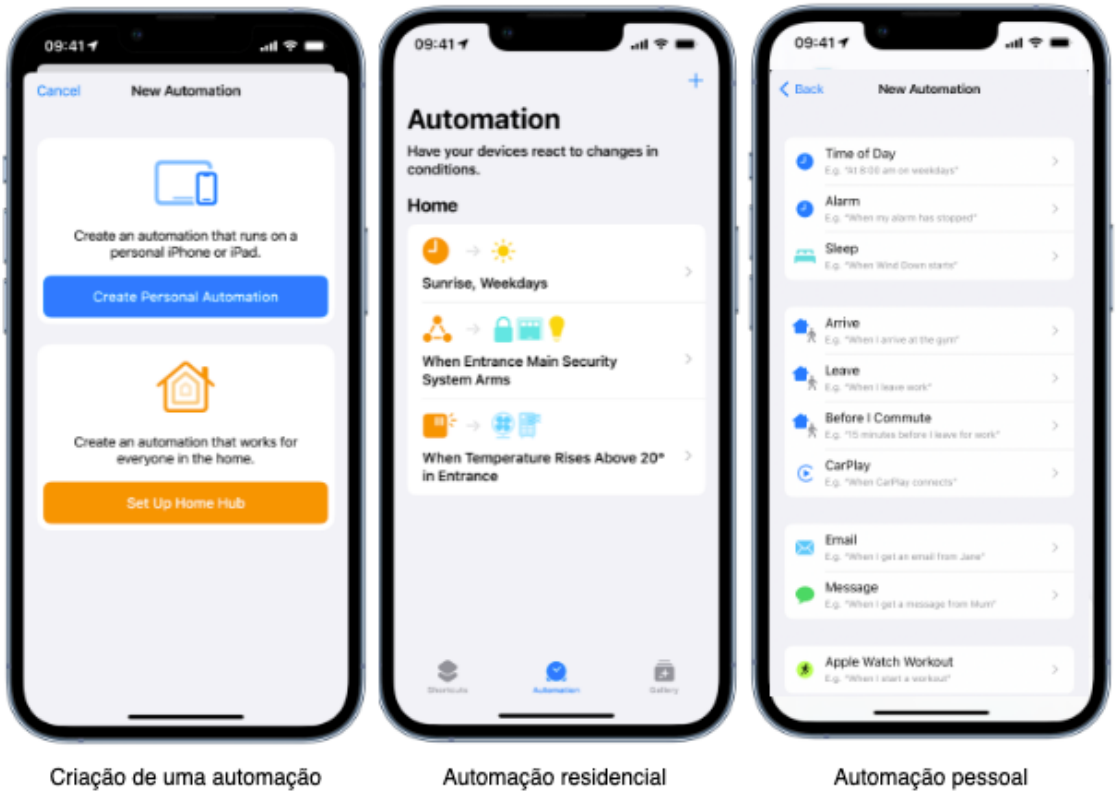

Figura 4 - Automações na aplicação Shortcuts [6]

Diferente de um atalho normal que requer execução manual, a automação residencial permite automatizar dispositivos de casa quando ocorre um evento. Usando ações, também

permite criar automação residencial avançada, com o uso de lógica para controlar os dispositivos, por exemplo, se a temperatura estiver acima de 21°C, o ar condicionado é ligado. Essa automação é sincronizada com o *hub* doméstico do utilizador através do iCloud.

As automações residenciais também podem ser convertidas num atalho, permitindo automatizar acessórios de casa com base em ações combinadas baseadas em lógica e dados de fontes externas. Após a criação, essas automações podem ser editadas, desativadas, ativadas ou apagadas.

Por outro lado, as automações pessoais fornecem uma maneira de agir com base em eventos como a hora do dia, chegar a um local ou abrir uma aplicação. Essa automação é semelhante a um atalho, no entanto, em vez de ser iniciada manualmente, é acionada por um evento. Essas automações podem tanto ser executadas automaticamente após o evento, como podem ser configuradas para serem executadas manualmente após o utilizador receber a notificação desse evento.

A configuração dessa automação, permite que o utilizador defina diferentes tipos de *triggers*. Como é possível observar na terceira imagem da Figura 4 [6], estes *triggers*, podem ser uma hora do dia, a receção de um *email* ou mensagem, carga de bateria, baseado em localização, entre outros. O utilizador tem a opção de escolher se essa automação é despoletada sempre ou num intervalo de tempo e definir a ação desta automação como, enviar mensagem, abrir uma aplicação, reproduzir uma música, fazer uma chamada, criar um alarme, entre outras.

Apesar da aplicação Shortcuts disponibilizar recursos que se assemelham aos disponibilizados na aplicação Arrived, é importante referir que a primeira grande diferença entre essas aplicações é o propósito de cada uma, enquanto a aplicação Arrived tem o foco na criação e gestão de avisos com base em localização, o foco da aplicação Shortcuts é a automação de tarefas e o acesso rápido a funções do dispositivo. O que pode ser bastante útil, para diversos contextos que contribuem bastante para a produtividade do utilizador.

Por outro lado, a configuração destas funcionalidades pode se tornar complexa, conforme o tipo de utilizador que as utiliza. O facto de a aplicação disponibilizar uma grande quantidade de atalhos e automações com diferentes formas de configurá-las, pode, por vezes, tornar-se um pouco complicado, principalmente quando o utilizador não tiver grande conhecimento do funcionamento do dispositivo em si, ou se estiver à procura de algo específico. Essas foram as impressões obtidas no primeiro contacto com a aplicação.

Após o estudo da aplicação Shortcuts através do seu manual [6], já foi possível compreender melhor o seu funcionamento e explorar mais a fundo algumas das funcionalidades disponibilizadas, para melhor efetuar uma análise comparativa entre as aplicações. Entre essas funcionalidades, a que mais se assemelha com a que a Arrived disponibiliza é o aviso com base em localização. Em ambas as aplicações é possível a criação desse tipo de aviso, tanto na chegada quanto na saída de um determinado limite de uma localização, efetuando uma ação quando esse gatilho acontece.

Na aplicação Arrived, esta ação é o envio de uma mensagem, já na aplicação Shortcuts essa ação poderá ser o envio de uma mensagem, ou outras ações especificadas anteriormente. Outra semelhança entre as aplicações é a possibilidade da criação de um lembrete para o próprio utilizador com base em localização, apesar dessa opção não estar tão explícita na aplicação Shortcuts, como está na Arrived.

Em relação a diferenças entre as aplicações, uma delas foi encontrada no momento da escolha da localização para o aviso, ambas as aplicações disponibilizam uma barra de pesquisa para este efeito e a visualização desta localização escolhida num mapa, mas apenas a Arrived disponibiliza também a opção de o utilizador selecionar esta localização através do mapa.

Outras diferenças encontradas entre as aplicações relacionadas a avisos baseados em localização é que ao contrário da aplicação Shortcuts, a Arrived disponibiliza: a funcionalidade para o utilizador efetuar o envio de pedidos de avisos a outros utilizadores da aplicação; a funcionalidade para o utilizador adicionar as suas localizações favoritas com nomes personalizados, para posteriormente usá-las na criação de um aviso; e a possibilidade de organização desses avisos conforme o tipo.

2.2. EgiGeoZone Geofence

A EgiGeoZone Geofence [7] é uma aplicação Android Geofencing, para *smartphone* ou *tablet* que usa GPS, Wi-fi ou dados móveis, para determinar a localização do utilizador e acionar diferentes ações quando ele entrar ou sair de zonas geográficas predefinidas (ver Figura 5 – adaptada de [7]). É possível a criação de até 100 zonas e para cada uma, o utilizador tem de definir no mínimo um nome, uma latitude, uma longitude e um raio mínimo de 50 metros [8].

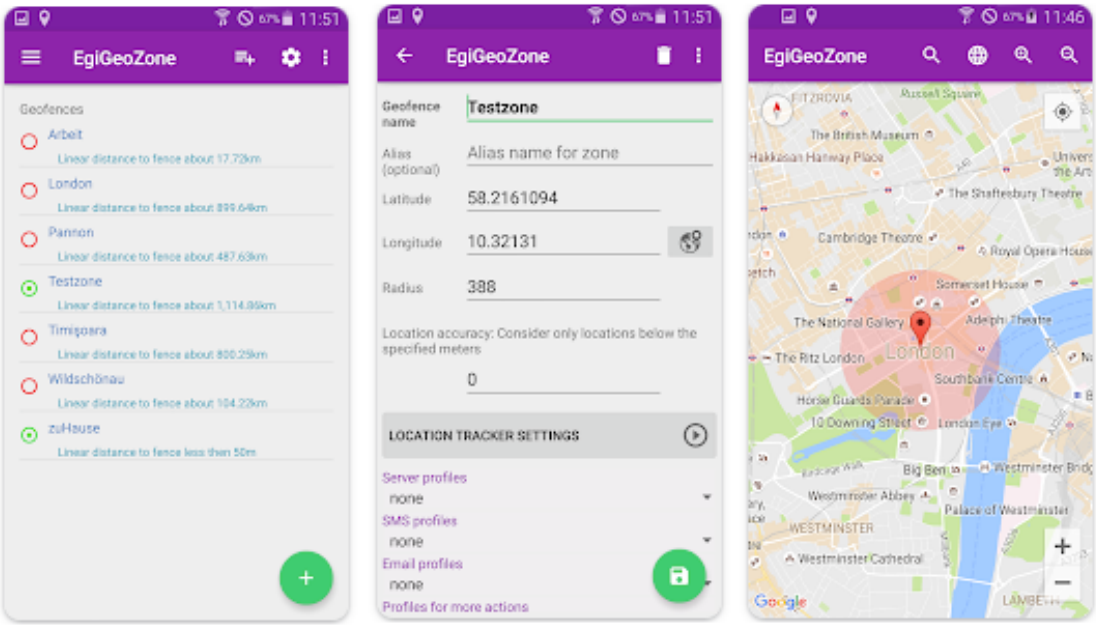

Figura 5 - Listagem e criação de uma zona geográfica na aplicação EgiGeoZone Geofence [7]

A aplicação EgiGeoZone permite acionar várias ações de forma automática ou simplesmente exibir uma notificação no ecrã para o utilizador acioná-las quando entra ou sai de uma zona. É possível que determinados eventos iniciem uma ação no *smartphone* do utilizador ou num servidor remoto. As ações disponibilizadas pela aplicação são: contactar um serviço de *geofencing* para controlar dispositivos de automação residencial, ou chamar os seus próprios *Uniform Resource Locators* (URL)s, enviar uma mensagem de texto, enviar *emails*, invocar tarefas "Tasker" [9], ativar ou desativar o Wi-fi, o som, ou o Bluetooth e rastreamento em tempo real.

Todas essas ações requerem uma configuração inicial. Na aplicação EgiGeoZone essas configurações são denominadas perfis (ver Figura 6 – adaptada de [7]), esses perfis podem ser associados a uma ou mais zonas, sendo executados quando ocorre um evento de entrada ou saída. Para que o utilizador se conecte a servidores de automação residencial é necessário ter alguma experiência em execução e configuração de *interfaces* com servidores (ver Figura 6 – adaptada de [7]), em alternativa podem optar por usar a aplicação para funções menos complexas como o envio de uma mensagem de texto ao sair ou entrar numa zona escolhida, que requer uma configuração mais simples.

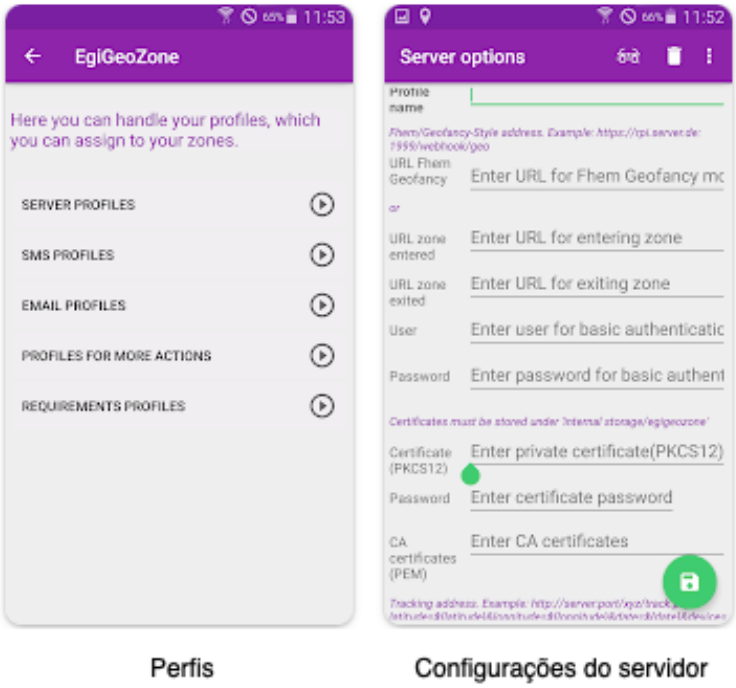

Figura 6 - Perfis e configurações de um servidor na aplicação EgiGeoZone Geofence [7]

É possível identificar possíveis aplicações onde essas ações possam ser executadas, como, abrir ou fechar a porta da garagem, ligar ou desligar o aquecimento ou as luzes, coordenar ações relacionadas com o carro. Por exemplo, se o motorista sair do trabalho, outra pessoa pode ser notificada automaticamente por SMS ou *e-mail*, permitindo que cheguem ao local da reunião a tempo. Ao sair da zona "casa", uma ação interna faz com que o Bluetooth seja ativado, permitindo que o telemóvel seja emparelhado com o alta-voz do carro. Quando o utilizador volta para casa, o Bluetooth é desligado novamente, economizando bateria. E por fim, ao chegar ao trabalho, pode-se fazer com que o EgiGeoZone desligue o som e ligue-o novamente ao sair.

Existem algumas semelhanças entre o EgiGeoZone e o Arrived, principalmente o uso de *geofencing*, o uso de notificações e o envio de SMS para notificar outros utilizadores relativamente a chegadas ou saídas de determinadas zonas e o facto de ambas as aplicações serem grátis. Já as principais diferenças são:

- A plataforma: enquanto o EgiGeoZone é uma aplicação disponibilizada para a plataforma Android, o Arrived é disponibilizado para iOS;
- O foco principal da aplicação: o Arrived tem o foco no envio de SMS e notificações para avisar utilizadores quando chegam ou saem de uma determinada localização predefinida, enquanto no EgiGeoZone existem outros tipos de ações que podem ser ativadas conforme o utilizador se aproxima ou se afasta de determinada localização;

- Funcionalidades: as diferenças encontradas entre funcionalidades vão ao encontro do propósito de cada a aplicação. A EgiGeoZone permite a configuração de diferentes perfis executados quando ocorre um evento de entrada ou saída, o que não é possível na Arrived. Já a Arrived, disponibiliza as funções para o utilizador efetuar o envio de pedidos de avisos a outros utilizadores da aplicação, do utilizador adicionar as suas localizações favoritas com nomes personalizados, para posteriormente usá-las na criação de um aviso e a possibilidade de organização desses avisos conforme o tipo. Essas funções não são disponibilizadas na aplicação EgiGeoZone;
- Complexidade na utilização: enquanto a Arrived fornece uma *interface* simples e intuitiva, com o mínimo de configurações necessárias para o utilizador tirar partido de todas as funcionalidades disponíveis, na EgiGeoZone todos as ações necessitam de configuração inicial e conforme a escolha do tipo de configuração pode-se tornar demasiado complexo para o utilizador.

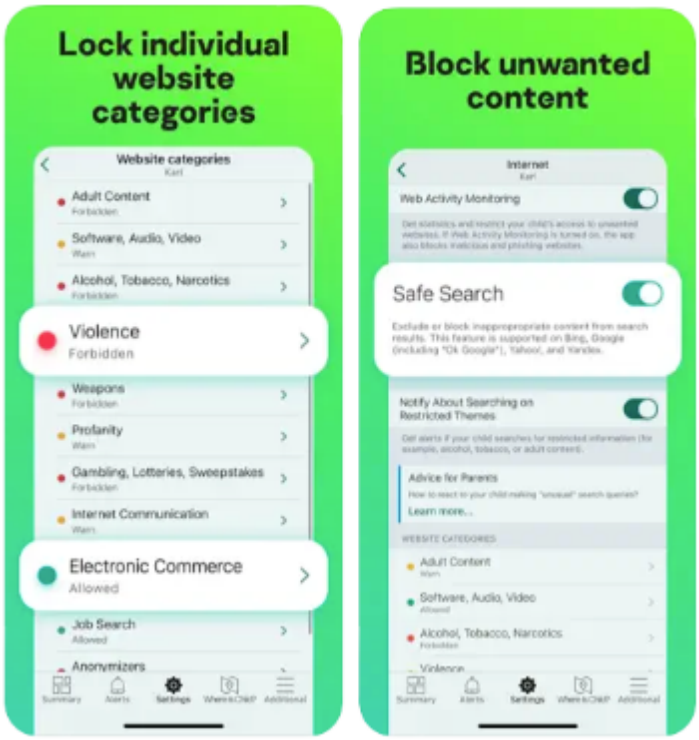

Figura 7 - Recursos de proteção a navegação a internet da aplicação Kaspersky Safe Kids with GPS [10]

2.3. Kaspersky Safe Kids with GPS

A aplicação Kaspersky Safe Kids with GPS [10], desenvolvida pela Kaspersky [11] conta com um perfil de utilização para crianças e um para os pais, ambos interligados através de uma conta My Kaspersky. Como se pode visualizar na Figura 7 [10], esta aplicação dispõe

de alguns recursos interessantes, tais como, ajudar os pais a protegerem as crianças dos perigos da navegação na Internet, como conteúdo para adultos ou outros programas e vídeos prejudiciais. Deste modo, oferece a possibilidade de bloquear esse conteúdo potencialmente prejudicial da *internet* e enviar notificações sobre ele.

Dispõe de informações sobre o nível da bateria do telefone dos filhos, monitorização das atividades no Facebook e chamadas telefónicas, permite a visualização de relatórios e gestão dos parâmetros da aplicação da criança, ajuda a gerir o acesso a jogos, aplicações inadequadas e permite gerir o tempo de ecrã por dispositivo. Na Figura 8 [10], é possível observar algumas das funcionalidades descritas.

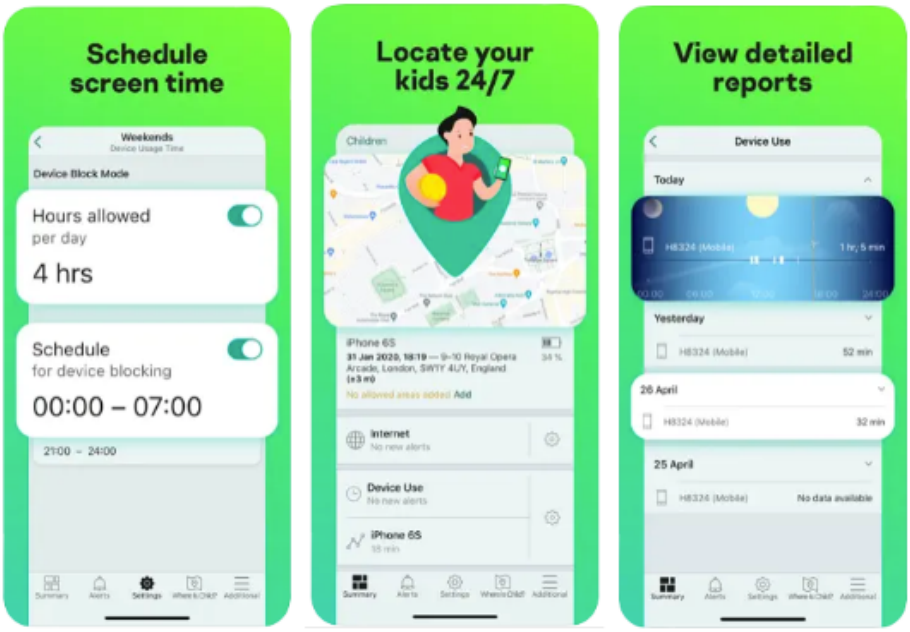

Figura 8 - Recursos de monitoramento de tempo de ecrã, localização e visualização de *reports* na aplicação Kaspersky Safe Kids with GPS [10]

Outros recursos importantes dessa aplicação e os que mais se assemelham à Arrived é o rastreio da localização dos filhos a qualquer momento com o uso do *geofencing* integrado (ver Figura 8 [10]), deste modo a aplicação informa se a criança saiu da área de segurança habitual. Isso é alcançado com a definição de uma área segura criada por uma cerca virtual no mapa que será posteriormente monitorizada pela aplicação, que por sua vez irá alertar o utilizador quando necessário.

A aplicação utiliza um rastreador GPS para localizar o paradeiro da criança num mapa *online* em tempo real, onde é possível rastrear todos os dispositivos que os filhos transportam com eles, além de disponibilizar partilha de conselhos de especialistas e dicas de psicólogos

infantis sobre tópicos *online*. O Kaspersky Safe Kids é grátis para iOS e Android, mas dispõe de algumas funcionalidades com o custo anual de 14,95 €.

Quando fazemos a comparação desta aplicação com a aplicação Arrived, podemos notar algumas semelhanças, ambas as aplicações são disponibilizadas para a plataforma iOS, fazem o uso de *geofence*, com a possibilidade da criação de uma cerca virtual numa determinada localização, onde o utilizador A é notificado quando o utilizador B sair dessa cerca. Embora a Arrived não tenha o foco em utilização de pais e filhos como a Kaspersky Safe Kids, esta pode ser facilmente utilizada para este efeito, mas não com tantos recursos relacionados a esse tema, como a Kaspersky Safe Kids.

Outra questão que não podemos deixar de considerar é que no caso do Kaspersky Safe Kids, só utilizadores do tipo pais podem definir essas cercas geográficas, para serem posteriormente notificados pela aplicação quando o utilizador filho saiu da mesma. Já no caso da Arrived, o utilizador A, pode definir essa cerca e pode escolher o utilizador B, que irá criar um aviso com base em localização para ele, mas ele não poderá fazer isso sem a permissão do utilizador B.

Outra diferença entre as aplicações é que a Arrived, não permite só a criação do aviso para quando o utilizador sai da cerca, mas também quando entra. Por fim, esse tipo de funcionalidade tal como as restantes é gratuita na Arrived, já na aplicação Kaspersky Safe Kids, essa funcionalidade está disponível apenas na versão paga da aplicação.

2.4. Checkmark 2

A aplicação Checkmark 2 [12], tem como objetivo principal a criação de lembretes baseados em localização com o uso de *geofencing*. É basicamente uma aplicação de lembrete inteligente que ajuda a organizar a rotina diária do utilizador ou uma agenda de negócios. É possível adicionar diversos locais, com descrição, tal como é possível observar na Figura 9 [12] e lembretes para concluir tarefas, como, por exemplo, no momento em que o utilizador passa pelo posto de combustível, a aplicação envia-lhe um lembrete para abastecer o seu carro.

Outros recursos disponíveis são, o agrupamento de locais semelhantes, por exemplo, três lojas mais visitadas ou cafés favoritos, a definição um horário específico para os lembretes baseados em localização, por exemplo, 15 minutos antes de sair do trabalho na quinta-feira à noite, o utilizador receberá uma notificação com um lembrete para dirigir até um supermercado para comprar alimentos em falta. É possível classificar as tarefas por

distância, reagendar tarefas e as tarefas do dia podem ser vistas com a aplicação em *background*. A aplicação Checkmark 2 está disponível apenas para dispositivos iOS e é uma aplicação paga, o seu custo total de 5,99 €.

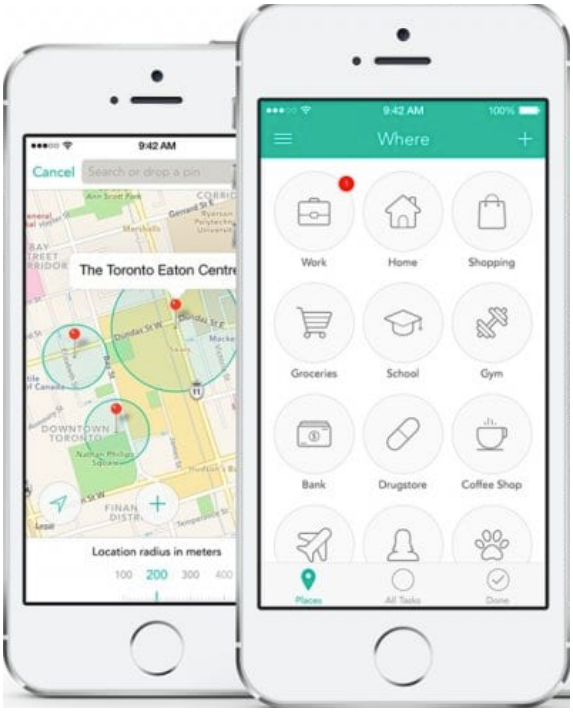

Figura 9 – Detalhes da aplicação Checkmark 2 [12]

Apesar de a aplicação Checkmark 2 e a Arrived terem diferentes focos, existem algumas semelhanças entre as aplicações, como a disponibilidade apenas para o sistema operativo iOS, o uso de *geofence*, a possibilidade de o utilizador criar lembretes para si mesmo com ação de notificá-lo quando se aproxima ou sai de uma determinada localização, do utilizador adicionar as suas localizações favoritas com nomes personalizados, para posteriormente usá-las na criação de um lembrete.

Relativamente às diferenças, podemos começar com o propósito de cada aplicação, enquanto a Checkmark 2 foca-se na organização pessoal e lembretes de tarefas com base em localização, a Arrived foca-se no envio e gestão de avisos e pedidos de avisos com base em localização. Outra diferença entre as aplicações é o custo associado, enquanto a Checkmark 2 tem um custo associado de utilização, a Arrived é totalmente gratuita.

E por fim, na aplicação Checkmark 2 as ações só envolvem um utilizador, que define as suas próprias ações e notificações, já na Arrived, cada ação que despoleta uma notificação pode envolver uma ou mais pessoas, não necessariamente utilizadores da aplicação. Permitindo avisar outras pessoas da chegada ou saída de determinadas localizações e envio de pedido de avisos para outros utilizadores.

2.5. Location Reminder

A Location Reminder [13], é uma aplicação gratuita de lembrete baseado em localização, desenvolvida para a plataforma Android. O seu sistema baseado em GPS inteligente envia lembretes de tarefas importantes conforme a localização atual e movimentos do utilizador, como sair ou entrar de um local predefinido. A aplicação irá notificar o utilizador quando o mesmo entrar numa zona geográfica que ele mesmo selecionou e até mesmo enviará um alerta de voz.

Os principais recursos desta aplicação são: a possibilidade de adicionar um local ao tocar manualmente num lugar específico do mapa disponibilizado pelo Google Maps [14] (ver Figura 10 [13]) ou selecionar o local com o uso do GPS. Para o lembrete baseado em localização, é possível configurar também a distância e o raio necessários. É possível adicionar e editar lembretes às listas de tarefas, escolher uma hora e um dia para o lembrete ou dias caso o utilizador queira que este se repita, e por fim, a aplicação faz a leitura de lembretes em texto e converte-os numa mensagem de voz.

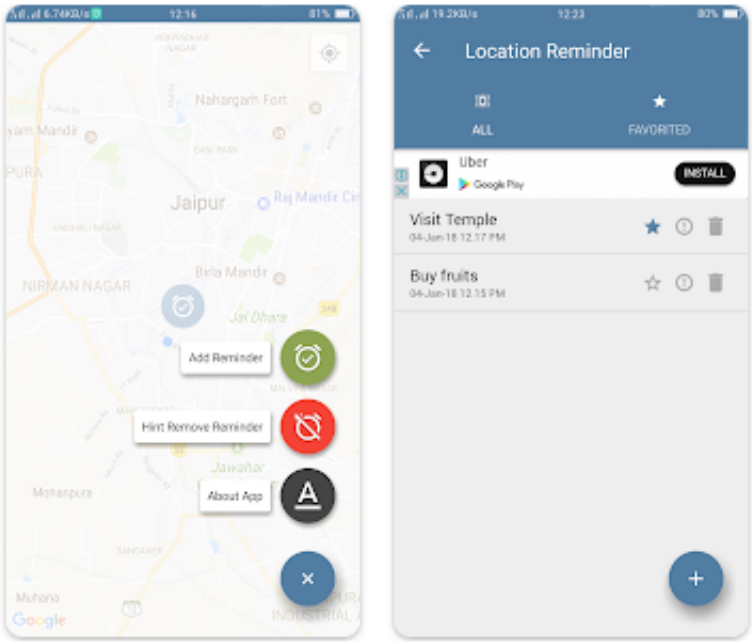

Figura 10 - Funcionalidades da aplicação Location Reminder [13]

A aplicação Location Reminder tem algumas semelhanças com a Arrived, ao começar pelo envio de notificações para o utilizador ao se aproximar de determinada localização, a possibilidade de selecionar um local no mapa de modo a definir uma localização para o lembrete, tal como a possibilidade de modificar a distância, raio e periodicidade desse

lembrete. Ambas as aplicações permitem a listagem desses lembretes, tal como a sua edição e remoção e por fim as duas aplicações são disponibilizadas gratuitamente.

Relativamente às diferenças temos uma diferença idêntica à identificada na aplicação Checkmark 2, a quantidade de utilizadores envolvidos em cada ação que despoleta uma notificação, ou seja, enquanto a Location Reminder é possível apenas um utilizador na Arrived podem ser um ou mais. A origem dessa diferença são os propósitos de cada uma das aplicações, enquanto a Location Reminder pretende enviar notificações de tarefas que o próprio utilizador definiu para si, a Arrived notifica um ou vários utilizadores quando outro chega ou sai de determinada localização.

Outras diferenças encontradas entre as aplicações são na criação de lembretes com base em localização, na aplicação Location Reminder é possível definir um intervalo de tempo para o utilizador ser notificado com esses lembretes, já no Arrived é possível escolher entre duas opções, que o utilizador receba sempre a notificação ou apenas uma vez. E por fim, enquanto a Arrived está disponível apenas para iOS a Location Reminder está disponível apenas para Android.

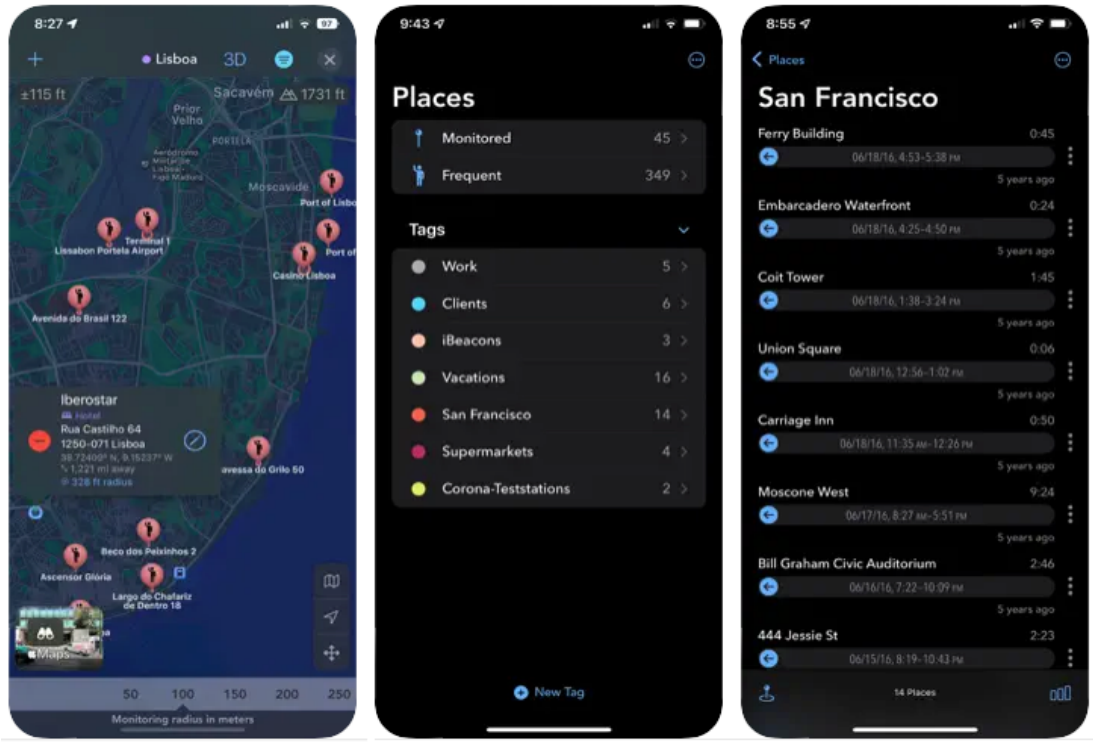

Figura 11 - Locais de interesse da aplicação Geofency - Time Tracking [15]

2.6. Geofency - Time Tracking

A Geofency - Time Tracking [15] é uma aplicação que permite a gravação automática de tempo baseada em localização, de modo a substituir o *check-in* e o *check-out* manuais.

Com o uso da tecnologia *geofencing*, assim que o utilizador entra ou sai de um local de interesse, os horários de chegada e partida são registados automaticamente (ver Figura 11 [15]). De modo a poupar a bateria do dispositivo, essa aplicação depende principalmente de dados do *smartphone* e Wi-fi e está disponível para dispositivos iOS, macOS, iPad e Apple Watch com um custo associado de 5,99 €.

O utilizador pode adicionar vários pontos de interesse diferentes, incluindo o seu trabalho, casa ou outros lugares da cidade e terá registos completos do tempo gasto num determinado lugar. Tal como é possível observar na Figura 11 [15] e Figura 12 [15], os principais recursos que a aplicação Geofency - Time Tracking disponibiliza são: o rastreamento de tempo baseado em localização automática 24h por dia; visualização do mapa no ecrã inteiro de modo que o utilizador selecione os seus pontos de interesse; o reconhecimento automático quando o utilizador entrar ou sair de uma geocerca; *tags* para agrupar lugares. Permite também estatísticas com resumos (únicos, diários, semanais e mensais), exportação de eventos para o calendário, exportação em formato compatível com o Excel, *widget* do dia, funcionamento em *background* e recurso *webhook* integrado onde é possível acionar um sistema de automação residencial assim que o utilizador chegar ou sair, por exemplo, acender as luzes ao voltar para casa.

Figura 12 - Funcionalidades da aplicação Geofency - Time Tracking [15]

Ao fazer uma comparação com a Arrived, podemos observar que, tal como em outras aplicações citadas anteriormente, a Geofency - Time Tracking conta com o uso de *geofencing* para a criação de cercas geográficas, onde a informação da localização do utilizador é o dado mais importante para que a aplicação possa efetuar uma ação. No caso desta aplicação esta ação é o registo do tempo que o utilizador esteve em determinada localização, diferentemente da Arrived, que apenas se interessa em saber se o utilizador chegou ou saiu de um determinado local predefinido para proceder ao envio de avisos.

Outros pontos que se assemelham à aplicação Arrived, são: a possibilidade de adicionar pontos de interesse através do mapa, no caso da Arrived este ponto é uma localização que o utilizador adiciona para posteriormente partilhar a sua chegada ou saída com outro utilizador, ou criar lembretes com base em localização para si. Ambas as aplicações são disponibilizadas para iOS e dispõem de funcionamento em *background,* mas diferente da aplicação Geofency - Time Tracking a Arrived é totalmente gratuita.

2.7. Análise comparativa

A análise de várias aplicações com recurso à tecnologia *geofencing*, permitiu identificar os pontos diferenciadores no que diz respeito aos objetivos propostos para a aplicação Arrived. Na Tabela 1 é feito um resumo das características e principais funcionalidades de cada aplicação analisada, em comparação direta com a aplicação Arrived. Os pontos alvo de comparação foram: os sistemas operativos para dispositivos móveis disponibilizados, a categoria, o custo, a informação se a aplicação contém anúncios, local onde o armazenamento de dados é feito, o ponto diferenciador de cada aplicação. E por fim, se a aplicação permite o envio de avisos para outros e lembretes com base em localização.

Comparativamente às aplicações analisadas, a aplicação Arrived diferencia-se por:

- Ter o foco na criação de avisos com base em localização;
- Armazenamento de dados completamente local, o que faz com a aplicação forneça aos utilizadores:
    - Maior segurança e privacidade: pois os dados estão armazenados nos seus dispositivos e não em servidores remotos que podem ser vulneráveis a ataques cibernéticos ou vulnerabilidades de segurança;
    - Não depender de uma conexão de internet para aceder ou atualizar os seus dados;
- Ter uma *interface* atrativa e intuitiva para todos os tipos de utilizadores.

Tabela 1 - Análise Comparativa - Características das aplicações com a tecnologia Geofencing

Características	Aplicações						
	Shortcuts	**EgiGeozone Geofence**	**Kaspersky Safe Kids with GPS**	**Checkmark 2**	**Location Reminder**	**Geofency - Time Tracking**	**Arrived**
Sistema operativo (Dispositivos móveis)	iOS	Android	iOS e Android	iOS	Android	iOS	iOS
Categoria	Produtividade	Utilitários	Utilitários	Produtividade	Produtividade	Utilitários	Utilitários
Custo	Gratuito	Gratuito	Gratuito e versão paga: 14,99 € por ano	5,99 €	Gratuito	5,99 €	Gratuito
Contém anúncios	Não	Não	Não	Não	Sim	Não	Não
Armazenamento de dados	Local e Nuvem	Local e Nuvem	Nuvem	Local e Nuvem	Local e Nuvem	Local e Nuvem	Local
Ponto diferenciador	Automações e atalhos	Ações com base em localização	Controlo parental	Lembretes baseados em localização	Lembretes baseados em localização	Registo de tempo baseado em localização	Envio de alertas com base em localização
Envio de alertas para outros com base em localização	Sim	Sim	Sim	Não	Não	Não	Sim
Lembretes com base em localização	Sim	Sim	Não	Sim	Sim	Não	Sim

É importante referir que os requisitos para o desenvolvimento da aplicação Arrived foram definidos antes do estudo e elaboração da análise comparativa descrita neste subcapítulo. Apesar disso, após ser feita esta análise e por se considerar que acrescentaria valor para a aplicação desenvolvida e vantagens face a algumas aplicações semelhantes, foi tomada a decisão de adicionar a funcionalidade que permite ao utilizador receber lembretes com base em localização.

3. Especificações

De forma a dar resposta a todos os objetivos apresentados anteriormente para a solução Arrived, apresenta-se neste capítulo, de forma mais detalhada, a sua metodologia de desenvolvimento e planeamento, os seus requisitos funcionais e não funcionais e, por fim, as suas respetivas *User Stories.*

3.1. Metodologia e planeamento

Este projeto está dividido em várias etapas que tentaram apresentar a necessidade de dominar os conceitos inerentes à utilização das tecnologias envolvidas e, posteriormente desenvolver uma solução à medida das necessidades identificadas. A primeira etapa iniciou-se com um levantamento do conceito de *geofences* e todo o estudo relacionado com aplicações que utilizam esse conceito, o qual foi sendo aplicado ao longo de todo o projeto. Na sequência dessas atividades foi iniciado o seu desenvolvimento.

De modo a compreender, e melhor acompanhar, o processo de desenvolvimento do projeto, foi criado um diagrama de Gantt apresentado na Figura 13 para a identificação de todas as etapas, desse processo, que decorreram ao longo de 77 semanas.

Nº	Descrição da tarefa	Duração
1	Execução do Projeto	77 semanas
2	Definição do tema do projeto	1 semana
3	Estudo do trabalho relacionado	4 semanas
4	Levantamento de requisitos para o desenvolvimento da aplicação	4 semanas
5	Tutoriais relacionados as tecnologias utilizadas no projeto	6 semanas
6	Desenvolvimento da aplicação	35 semanas
7	Refactor no desing da aplicação	16 semanas
8	Desenvolvimento de testes	10 semanas
9	Escrita do Relatório	45 semanas

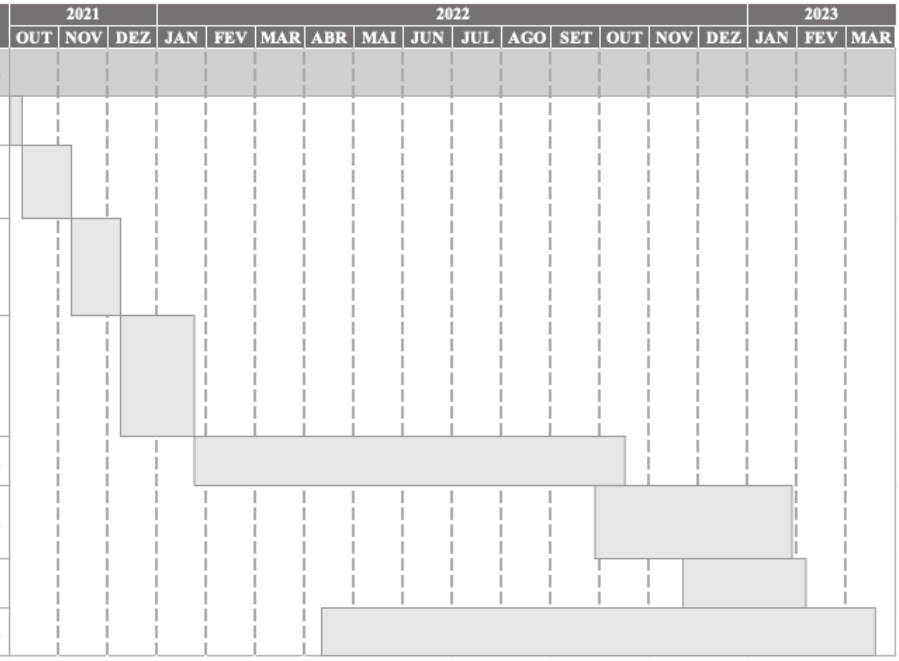

Figura 13 - Cronograma do processo de desenvolvimento do projeto

Para o desenvolvimento do projeto foi utilizada uma metodologia que tem como referência algumas metodologias ágeis, tais como:

- Scrum [16] – Eventos (*Sprints, sprint planning* e *review*);
- Kanban [17] – Todo o percurso do trabalho ao alcance do programador;
- *eXtreme Programming* [18] – Uso de *User Stories*.

A ferramenta Jira [17] foi utilizada como apoio para gerir e acompanhar todo o projeto através de um quadro de Kanban. De modo a auxiliar no controlo do fluxo do trabalho uma série de etapas foram adicionadas, tais como é possível visualizar na Figura 14 e na Figura 15, e que se descrevem de seguida.

1. *Todo* – tarefa a fazer, onde é criada a tarefa que também poderá ser designada de U*ser Story*;
2. *Blocked* – tarefa bloqueada por outra tarefa ou por outra dependência;
3. *In Progress* – tarefa que está a ser desenvolvida;
4. *In Review* – pedido de integração da tarefa, que está à espera de ser revisto;
5. Quality Assurance (QA) – tarefa ponta para ser testada;
6. *Done* – tarefa concluída.

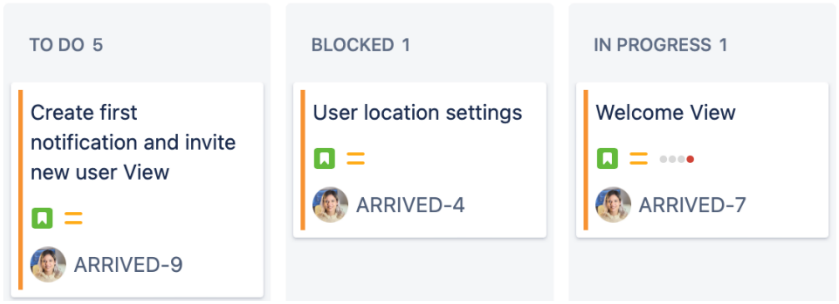

Figura 14 - Board do Jira - Todo, Blocked e In Progress

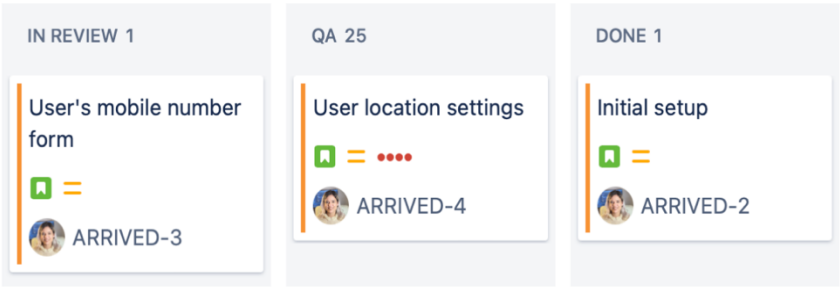

Figura 15 - Board do Jira - In Review, QA e Done

Uma *User Story* representa uma funcionalidade no desenvolvimento do projeto, com uma numeração, uma breve descrição dessa funcionalidade e consoante o seu grau de complexidade poderá conter sub-tarefas. Depois da criação de cada *User Story* no *backlog* estas foram sendo arrastadas para os seus respetivos estados, conforme as etapas da Figura 14 e Figura 15, foram sendo concluídas. De acordo com a Figura 16 podemos observar os detalhes da US 3 tais como: a sua descrição, a sua lista de sub-tarefas, o responsável pela sua implementação e, finalmente, o estado em que a mesma se encontrava, *In Progress*.

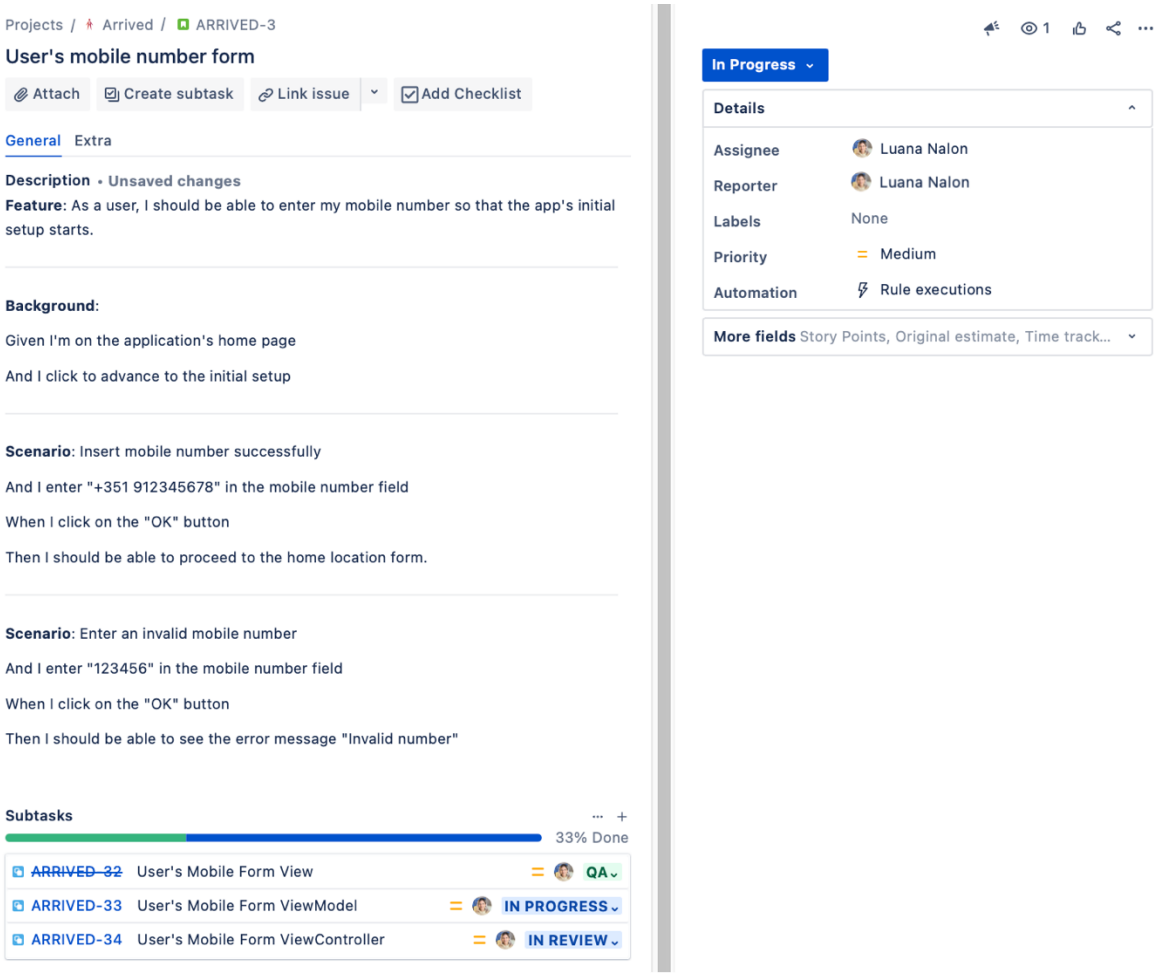

Figura 16 - Detalhes de uma User Story no Jira

A informação obtida para a criação e eventuais alterações de cada *User Story* foi sendo aprimorada ao longo da duração do projeto através de reuniões (*sprints* quinzenais), onde era discutido com a professora orientadora as funcionalidades e tarefas importantes para o seu desenvolvimento e melhoria, fazendo assim um *sprint planning* para cada quinzena. No fim dos *sprints* havia uma *review* de todo o trabalho feito de forma a adicionar ou adaptar novas tarefas no *backlog* para a próxima *sprint* caso fosse necessário.

Outra tarefa importante dos *sprints* quinzenais era a retrospetiva (*retrospective*) de como havia corrido cada tarefa, o que havia corrido bem, as dificuldades encontradas e o que poderia ser melhorado para o *sprint* seguinte. Ao longo deste processo de desenvolvimento, houve algumas reuniões também com os membros da VOID Software, nomeadamente com o responsável e cliente do projeto, representado pelo Marco Cova, o *tech lead* Hugo Larcher, a orientadora do projeto perante a empresa, Marta Carvalho, e o responsável pelo *design*, João Nascimento. Essas reuniões, foram bastante importantes para o sucesso do projeto, pois através delas, novas ideias foram discutidas e todas as dúvidas que surgiam eram esclarecidas.

3.2. Análise de requisitos

Nesta secção, irá ser descrito todo o processo de levantamento de requisitos, desde o primeiro contacto com o projeto na reunião de *kick off,* até à definição de todos os aspetos fundamentais para dar início à implementação do projeto Arrived.

3.2.1. Kick off

O termo *Kick off* [19], que em português significa "dar início" e é o termo utilizado para se referir a primeira reunião, quando um projeto é iniciado. É onde as ideias do projeto são apresentadas, onde é discutido o que é esperado alcançar com o seu desenvolvimento e quais serão as funções de cada participante. Portanto, é muito importante que essa reunião aconteça para que todos os envolvidos no desenvolvimento do projeto estejam na mesma página e evitando possíveis falhas de comunicação.

Como ponto de partida do projeto Arrived, houve uma *kick off,* com a equipa relacionada ao projeto. Nesta reunião, foi feito o levantamento inicial de requisitos, funcionais e não funcionais, e as dúvidas relacionadas a esses requisitos foram esclarecidas. A partir desse ponto, foi possível iniciar o desenvolvimento efetivo do projeto.

3.2.2. Levantamento de requisitos

Neste subcapítulo serão descritos todos os requisitos da aplicação Arrived, tanto os funcionais como os não funcionais. É importante salientar que, ao longo de todo o desenvolvimento deste projeto, alguns dos requisitos sofreram algumas alterações, de modo que a solução final tentasse alcançar todas as expectativas, tanto do cliente como do responsável pelo projeto e dos seus utilizadores finais.

3.2.3. Requisitos funcionais

Os requisitos funcionais da Arrived são:

1. Registo de dados do utilizador - Número de telemóvel e localização dos locais mais comuns para o mesmo (casa, trabalho, escola e outros);
2. Edição dos dados de localização do utilizador;
3. Capacidade de criação de um aviso com base numa localização, onde é possível determinar, um ou mais destinatários, uma cerca geográfica parametrizável em termos de distância, a indicação se é um aviso de chegada ou de saída dessa zona e a frequência que é necessário notificar o(s) destinatário(s) selecionado(s), se é apenas uma vez ou sempre;
4. Envio de uma SMS de modo a notificar o(s) destinatário(s) do aviso, que o utilizador chegou ou saiu de um determinado local adicionado por ele;
5. Criação de um pedido de aviso a outro utilizador, onde é possível definir todos os parâmetros citados no ponto 3 e uma mensagem customizada, com a diferença do destinatário que nesse caso será o utilizador que criou o pedido de aviso;
6. Capacidade de o utilizador criar um lembrete para si próprio quando entrar ou sair de uma determinada zona geográfica;
7. Três opções de escolha de localização para a criação de avisos e pedidos de avisos:
    a. 1ª opção: Através de uma barra de pesquisa, onde será possível a pesquisa tanto por morada como por coordenada geográfica;
    b. 2ª opção: Selecionar a localização desejada numa vista de mapa;
    c. 3ª opção: Selecionar uma das localizações definidas no ponto 1.
8. Editar avisos e pedidos de aviso;
9. Opção de o utilizador aceitar ou recusar um pedido de aviso;
10. Opção de o utilizador reenviar um pedido de aviso;
11. Opção de o utilizador executar ação de notificar o(s) destinatário(s) de um aviso, não só através da notificação recebida como também através da listagem de avisos;
12. Listar todos os tipos de aviso;
13. Apagar os pedidos de aviso;
14. Capacidade de o utilizador poder convidar outro utilizador para instalar a aplicação através do envio por SMS do *link* da aplicação na App Store.

3.2.4. Requisitos não funcionais

Os requisitos não funcionais da Arrived são:

1. Desenvolvimento *mobile*, mais especificamente para o sistema operacional iOS, com o uso da *framework* UIKit [20] como base, que por sua vez permite a criação e gestão de uma UI gráfica baseada em eventos, tanto para aplicações iOS que é o caso da Arrived como para aplicações tvOS;
2. De modo a promover a simplicidade, segurança e privacidade, efetuar o desenvolvimento da aplicação com o mínimo de dependências externas possíveis, tornando o seu funcionamento mais autónomo e seguro;
3. Para que haja um desenvolvimento prático, facilidade nos *update*s e clareza na manutenção do *software* para os programadores futuros, a preferência é dada à utilização de *frameworks* nativas de iOS, tais como:
    a. Core Data [21] – Base de dados local que permite a persistência local dos dados;
    b. Core Location [22] – *framework* que permite obter a localização geográfica e a orientação de um dispositivo;
    c. MapKit [23] – *framework* que permite a exibição de mapas ou imagens de satélite, a indicação de pontos de interesse, bem como a adição de informações de marcadores para as coordenadas do mapa;
    d. User Notifications [24] – *framework* que permite o envio de notificações direcionadas ao utilizador a partir de um servidor ou a possibilidade de geri-las localmente, que é o caso da Arrived;
    e. Message UI [25] – *framework* que possibilita a criação de uma interface para a escrita de mensagens de texto e e-mail, sem que o utilizador precise sair da aplicação.
4. Uso do conceito de *geofencing* [26], que permite monitorizar uma região para determinar quando o utilizador entra ou sai de uma área geográfica;
5. O desenvolvimento de uma aplicação simples de utilizar, para que alcance pessoas de diferentes faixas etárias.

3.3. User Stories

Com base na definição dos requisitos funcionais e não funcionais da aplicação Arrived, e para que o seu desenvolvimento fosse iniciado, a criação das seguintes *User Stories* foi efetuada:

1. *Initial Setup* – Criação do projeto e as suas configurações iniciais relativas ao ambiente de desenvolvimento e de testes;
2. *User's mobile number form* – Criação da vista para obter o número de telemóvel do utilizador;
3. *User Location Settings* – Criação da vista para obter as principais localizações do utilizador (casa, trabalho, escola e outras);
4. *Welcome Page* – Página de introdução para a aplicação Arrived, onde o utilizador poderá visualizar algumas dicas de utilização e começar a usar a aplicação;
5. *Create alert* – Criação de todos os três tipos de avisos permitidos pela aplicação (avisos para outro utilizador, avisos para o próprio utilizador e pedido de avisos para outros utilizadores);
6. *Reply to alert request* – Possibilidade de o utilizador responder a pedidos de avisos criados por outros utilizadores;
7. *List alerts* – Possibilidade de o utilizador visualizar a listagem de avisos e efetuar algumas ações relacionadas a essa listagem, nomeadamente, apagar, editar, reenviar pedidos de aviso e responder a avisos;
8. *Side Menu* - Menu lateral com as opções de editar as localizações definidas no ponto 3, convidar novo utilizador para instalar a aplicação, política de privacidade e termos e condições.

4. Estudo e configuração de tecnologias

Neste capítulo, todas as tecnologias usadas no âmbito da implementação da Arrived irão ser descritas. Iremos observar que a maioria das *frameworks* apresentadas pertencem à Apple, essa foi uma abordagem a ser tomada devido à aplicação Arrived ter sido desenvolvida para o sistema operativo iOS, o que faz com que a preferência em *frameworks* nativas facilite os *updates* futuros das mesmas e garanta uma boa experiência de utilizador, principalmente para os utilizadores habituais do iOS.

É importante referir que o estudo das tecnologias descritas nesse capítulo, foi fundamental para o desenvolvimento da aplicação Arrived. Pois o conhecimento adquirido, seja pela documentação ou através de tutoriais, permitiu tirar partido das suas principais vantagens, contribuindo consequentemente para a qualidade do produto final.

4.1. UIKit

A *framework* UIKit [27] permite a construção e gestão de uma UI gráfica orientada a eventos para uma aplicação tanto iOS como tvOS. Esta *framework* fornece: uma arquitetura de vista e janela para implementação de uma *interface*, a infraestrutura de manipulação de eventos para fornecer *Multi-Touch* e outros tipos de *inputs* para a aplicação, e o *loop* de execução principal necessário para gerir as interações entre o utilizador, o sistema e a aplicação. Além dos recursos citados, inclui suporte a animação, suporte a documentos, suporte a desenho e impressão, informações sobre o dispositivo atual, gestão e exibição de texto, suporte a pesquisa, suporte a acessibilidade, suporte a extensão de aplicação e gestão de recursos.

Os objetos fornecidos pela *framework* UIKit são usados para exibir o seu conteúdo no ecrã, interagir com esse conteúdo e gerir as interações com o sistema [28]. As aplicações contam com o UIKit para seu comportamento básico e o UIKit oferece diversas formas de personalizar esse comportamento de modo a atender necessidades do programador. Para cada aplicação criada, o Xcode fornece *templates* de projetos como pontos de partida, tal como podemos visualizar no exemplo da Figura 17, onde é apresentada a estrutura inicial da Arrived, que foi criada usando o *template Single View.* Os *templates* de projetos fornecem uma UI mínima para ser possível criar e executar o projeto imediatamente e ver os resultados num dispositivo ou no simulador.

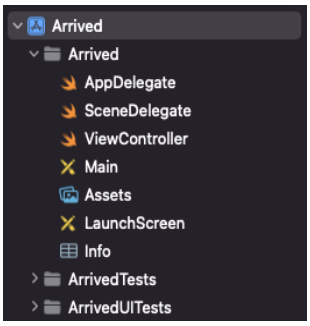

Figura 17 - Estrutura inicial de uma aplicação iOS single view no Xcode

Quando é feito o *build* de uma aplicação, o Xcode compila os *source* files e cria um *app bundle* para o projeto, que é um diretório estruturado contendo o código e os recursos associados à aplicação. Os recursos incluem: os *assets* de imagem, ficheiros de *storyboard* (vistas), ficheiros de *strings* e *metadados* da aplicação que dão suporte ao código. Apesar da importância da estrutura do *app bundle*, o Xcode tem já predefinido as localizações dos seus recursos. Todas as aplicações que usam UIKit precisam ter dois recursos principais:

6. *App icons*;
7. *Launch screen storyboard*.

O ícone da aplicação (ver Figura 18) é exibido através do sistema no ecrã inicial, nas configurações do dispositivo e em qualquer lugar onde haja necessidade de diferenciar a aplicação de qualquer outra. Por isto, é necessário fornecer várias versões de diferentes tamanhos deste ícone. Essas versões são disponibilizadas no recurso de imagem AppIcon do projeto Xcode.

Figura 18 - Ícone da aplicação Arrived

Para gerar os ícones para todos os tamanhos necessários, foi usada a aplicação *web* gratuita designada App Icon Generator [29], onde foi necessário fazer o *ulpload* da imagem que contém o ícone com a resolução 1024x1024 e selecionar os sistemas operativos desejados, que no caso da Arrived foi apenas o iOS.

O outro recurso importante é o ficheiro LaunchScreen (ver Figura 19), porque contém a UI inicial da aplicação, que poderá ser tanto uma versão simplificada da *interface* real como apenas um *splash screen*. Esse *launch screen* é exibido a partir da primeira iteração do

utilizador com a aplicação, ou seja, quando ele seleciona o ícone da aplicação é essa vista que o vai informar que a aplicação está a ser iniciada.

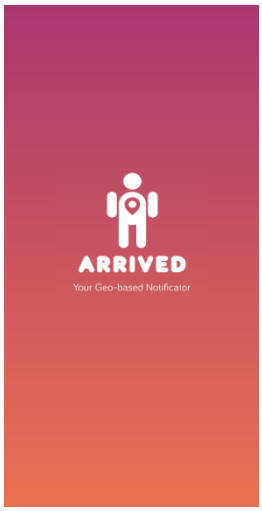

Figura 19 - Launch Screen da aplicação Arrived

Além disso, o *launch screen* também oferece cobertura para a aplicação enquanto esta inicia, para "distrair" o utilizador caso exista alguma verificação inicial que poderá ser necessária, nomeadamente para saber qual será a primeira vista real a ser apresentada ao utilizador na aplicação. Por fim, quando a aplicação está pronta, o sistema oculta o *launch screen* e revela a *interface* gráfica real da aplicação. O *launch screen* pode ser muito útil quando se trata de uma aplicação que não necessita ter alguma interação ou uma pré-configuração da vista inicial, visto que quando esta vista surge a aplicação ainda não foi carregada, ou seja, a hierarquia de vistas não existe.

Outro ficheiro muito importante para uma aplicação é o Info.plist, porque contém uma lista de informação de propriedades, que o sistema obtém sobre a configuração e os recursos da aplicação. Apesar do Xcode fornecer uma versão pré-configurada desse ficheiro em cada *template* de projeto, geralmente é necessária alguma alteração. Por exemplo, se a aplicação precisar de algum *hardware* específico ou usar *frameworks* de sistema específicas, poderá ser necessário adicionar informações relacionadas a esses recursos nesse ficheiro.

Uma alteração comum que poderá ser feita no Info.plist é a declaração dos requisitos de *hardware* e *software* da aplicação. É através desses requisitos que é feita a comunicação com o sistema para informar o que a aplicação precisa para ser executada. Por exemplo, na Arrived é necessário o uso do GPS de modo a verificar se a localização do dispositivo se aproximou de uma determinada localização, se esse requisito estiver especificado e o

dispositivo que for instalar a aplicação não tiver esse *hardware* a App Store impede que a aplicação seja instalada.

A estrutura de código fornecida pelo UIKit, produz muitos dos principais objetos da aplicação, incluindo aqueles que interagem com o sistema, que executam o *loop* de eventos principal da aplicação e que exibem o seu conteúdo no ecrã. A maioria desses objetos são usados tal como são fornecidos, com a possibilidade de incluir pequenas alterações. Sendo que para implementar qualquer aplicação é fundamental perceber quais os objetos a modificar bem como quando modificá-los.

Essa *framework* é baseada no padrão de *design* Model-View-Controller (MVC), em que há uma divisão de finalidade para cada objeto. Os objetos de modelo gerem os dados e a lógica de negócios da aplicação, os objetos da vista fornecem a representação visual de dados e, por fim, os objetos do controlador fazem a ligação entre o modelo e os objetos de vista, movendo dados entre eles em momentos adequados. Embora a UIKit seja baseada nesse padrão, na implementação da Arrived seguiu-se o padrão Model-View-View-Model (MVVM) que permitiu uma maior separação das funções de cada objeto, contribuindo na redução de código e lógica nos controladores das vistas, ao deixar nos controladores apenas a gestão de apresentação dos elementos de cada vista.

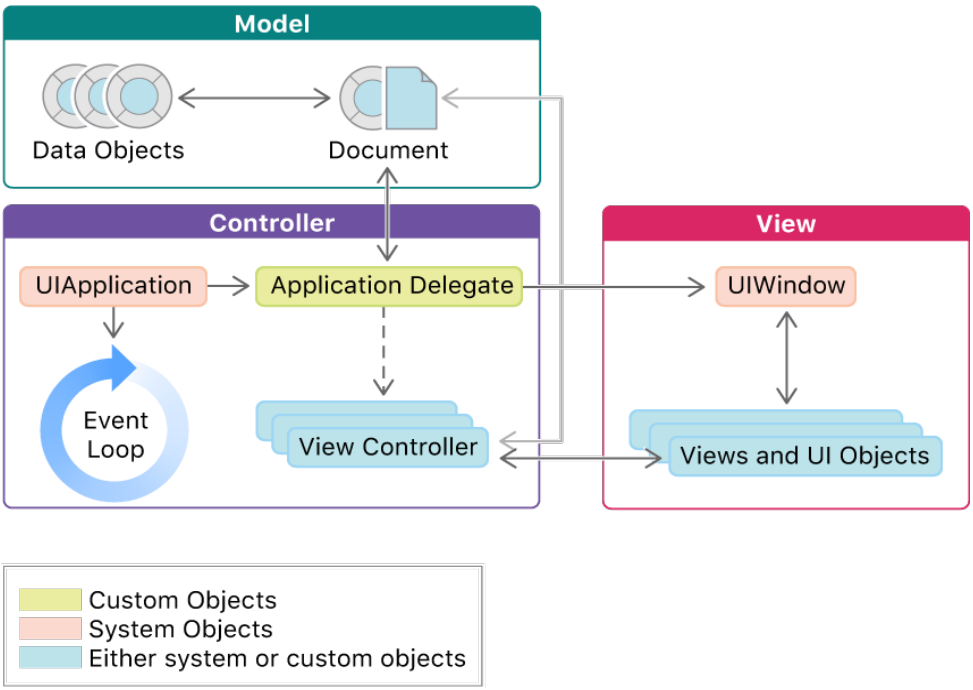

Figura 20 - Objetos principais de uma aplicação feita em UIKit [28]

De modo a compreender melhor a estrutura mais comum de uma aplicação feita em UIKit podemos analisar a Figura 20 [28]. Os objetos de modelo, que representam as estruturas de dados da aplicação desenvolvida, são fornecidos pelo programador. Embora o UIKit forneça a maioria dos objetos da vista, é possível definir vistas personalizadas para os dados, conforme necessário. São os controladores da vista e o *delegate* da aplicação [30] que coordenam a troca de dados entre os objetos de dados e as vistas do UIKit.

Muitos dos tipos básicos usados para definir os objetos do modelo da aplicação, são fornecidos através das *frameworks* UIKit e Foundation. Para efetuar a organização das estruturas de dados pertencentes a um ficheiro baseado em disco, o UIKit fornece um objeto UIDocument [31]. Já a *framework* Foundation define objetos básicos que representam *strings*, números, *arrays* e outros tipos de dados, sendo que muitos desses tipos também são fornecidos pela Swift Standard Library.

A maioria dos objetos do controlador e das camadas da vista são fornecidos pelo UIKit, como a definição da classe UIView [32], que usualmente é responsável por exibir o conteúdo da aplicação no ecrã. Já o objeto UIApplication [33] executa o *loop* de eventos principal e gere o ciclo de vida geral da aplicação.

Outro tópico fundamental relacionado com a UIKit é a proteção de dados do utilizador [34]. Na maioria dos dispositivos Apple existem dados pessoais que o utilizador não deseja expor a outras aplicações ou entidades externas. O utilizador poderá facilmente deixar de utilizar uma aplicação que aceda ou use dados de forma inadequada. Por isso, um programador deve aceder aos dados do utilizador ou do dispositivo apenas com o consentimento informado do utilizador, obtido conforme a lei aplicável, tomar as medidas apropriadas para proteger esses dados e garantir transparência sobre como serão usados.

As solicitações de acesso devem ser feitas somente quando a aplicação precisar dos dados, como, localização, contactos e fotos. Para isso é necessário inserir uma *string* que descreve o uso no ficheiro Info.plist, desta forma o sistema poderá apresentar ao utilizador o motivo pelo qual a aplicação precisa desse acesso. Na Arrived são feitas algumas solicitações desse tipo como podemos visualizar na Figura 21, onde temos as permissões necessárias para obter dados relacionados com a localização.

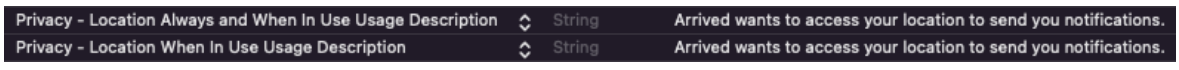

Figura 21 - Strings de descrição de uso de dados no fiheiro Info.plist

A transparência do processo sobre como os dados serão usados pode ser aplicada, por exemplo, quando a aplicação é enviada para a App Store, ao especificar um URL para a política ou declaração de privacidade como parte dos metadados do App Store Connect [35]. É importante garantir ao utilizador o controlo sobre os dados, proteger os dados recolhidos e respeitar as preferências do utilizador. E para que isso seja possível, o programador poderá fornecer configurações que permitam que o utilizador desabilite o acesso a informações confidenciais.

Para recursos protegidos do sistema, como localização, contactos e dados de saúde, o sistema operativo faz isso automaticamente através do menu "Privacidade" da aplicação de Configurações. O programador deve solicitar e usar a quantidade mínima de dados do utilizador ou do dispositivo necessários para realizar uma determinada tarefa. O acesso ou a recolha dos dados por motivos desnecessários, ou não óbvios, não deve ser feita apenas por se pensar que essa informação poderá ser útil no futuro.

Como pudemos verificar, a UIKit é umas das *frameworks* mais usadas para o desenvolvimento de iOS, tendo um vasto conjunto de recursos disponíveis. Ao longo deste relatório muitos desses recursos serão citados, devido o desenvolvimento da Arrived ter tirado partido de muitos deles.

4.2. Core Data

A *framework* Core Data [21] da Apple é usada para persistência local de dados para uso *offline*, para armazenar em *cache* os dados temporários e para adicionar a funcionalidade de *undo* numa aplicação iOS e num único dispositivo. Caso seja necessário sincronizar dados em vários dispositivos numa única conta do iCloud, o Core Data espelha automaticamente um esquema num *container* do CloudKit [36].

Através do editor de modelo de dados do Core Data, é possível definir os tipos e relacionamentos dos dados e gerar as respetivas definições de classe. O Core Data pode então gerir instâncias de objeto em *runtime* para fornecer os seguintes recursos:

- Persistência - De modo a facilitar a forma de guardar os dados de Swift e Objective-C sem ter de administrar uma base de dados diretamente, o Core Data abstrai os detalhes do mapeamento dos seus objetos para uma *store,* (ver Figura 22 – adaptada de [21]) que é um mecanismo de armazenamento de dados usado pelo Core Data para persistir os objetos do modelo da aplicação;

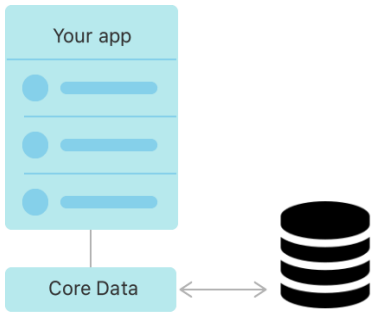

Figura 22 - Core Data (Persistência de dados) [21]

- *Undo* e *Redo* de alterações individuais ou em lote - De modo a facilitar a adição de suporte de *undo* e *redo* numa aplicação, o *undo manager* do Core Data rastreia as alterações e pode revertê-las individualmente, em grupos ou de uma só vez (ver Figura 23 [21]);

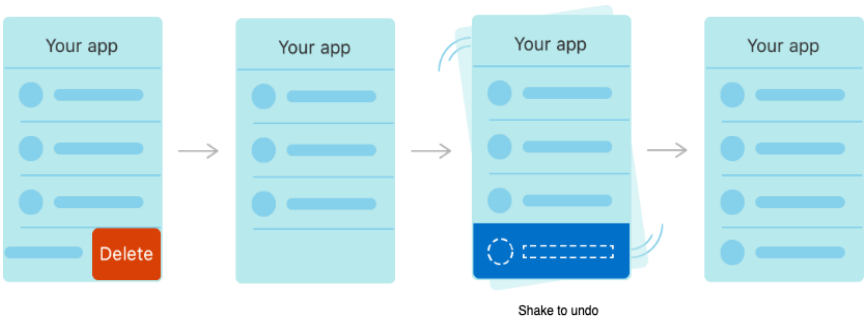

Figura 23 - Core Data (Ação Undo) [21]

- Tarefas de dados em *background* - Permite executar tarefas que envolvem dados que podem bloquear a UI, como o *parse* de JavaScript Object Notation (JSON) para objetos em *background*. É possível manter em *cache* ou armazenar os resultados para evitar demasiados pedidos ao servidor (ver Figura 24 [21]);

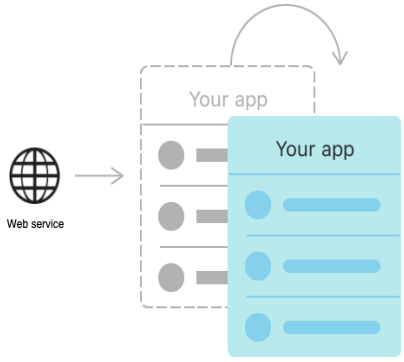

Figura 24 - Core Data (Tarefas de dados em background) [21]

- Sincronização da vista - O Core Data fornece *data sources* para vistas com UITableViews (tabela) [37] e para UICollectionViews (*layout* semelhante a uma grade) [38], isso ajuda a manter as vistas e os dados sincronizados na aplicação;
- Versionamento e migração - À medida que a aplicação evolui, o Core Data inclui mecanismos de controlo de versão do modelo de dados e migração de dados do utilizador.

Para a implementação do Core Data, o primeiro passo a ser tomado é a criação de um ficheiro de modelo de dados [39] (ver anexo A) para definir a estrutura dos objetos da aplicação, incluindo os seus tipos, propriedades e relacionamento como podemos visualizar na Figura 25.

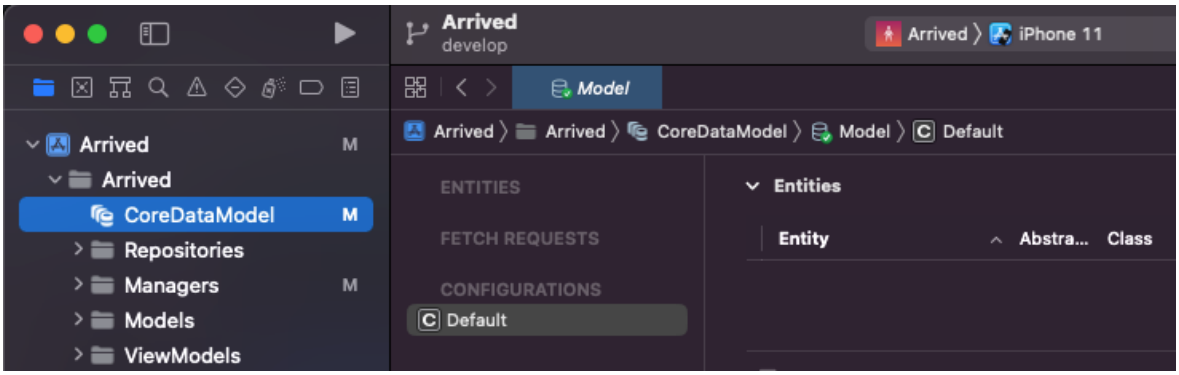

Figura 25 - Ficheiro do modelo de dados do Core Data no Xcode

Após a criação do modelo de dados já é possível criar as **Entities** [40] e configurá-las para modelar os objetos do modelo de dados da aplicação. Uma **Entity** descreve um objeto/modelo, incluindo o seu nome, atributos e relacionamentos [41]. Além dos campos obrigatórios *name* e *class name*, as entidades têm uma configuração padrão para o campo de geração de código obrigatório. Caso seja preciso adicionar herança, *constraints* únicas, controlo de versão ou outras informações opcionais, é necessário ser feita uma configuração. Caso contrário, é necessário apenas adicionar as propriedades que compõem a entidade criada, que foi o caso da Arrived. No Anexo B é possível obter mais detalhes da criação de uma **Entity**.

Uma entidade é composta por atributos e um atributo descreve uma propriedade de uma entidade [42]. É necessário especificar pelo menos o nome e o tipo de dados da propriedade, caso esta deva ser guardada no armazenamento persistente e se for necessário ter um valor quando for guardada. Também é possível, para alguns tipos de atributo, a opção de usar um

tipo escalar para representar o atributo nas classes geradas, bem como configurar o atributo para ter um valor *default* ou aplicar regras de validação de dados. Mais detalhes da configuração de atributos podem ser obtidos no Anexo C.

Após ter pelo menos duas entidades e os seus respetivos atributos definidos, caso haja necessidade, o próximo passo seria a configuração das relações [43] dos modelos da base de dados (ver Anexo D). A informação de como uma entidade afeta outra é obtida através de um relacionamento. Um relacionamento especifica pelo menos um nome, uma entidade de destino, uma regra de exclusão, um tipo de cardinalidade (*to one* ou *to many*) e configurações que indicam se o relacionamento deve ser salvo no armazenamento e se é necessário ter um valor quando salvo. E por fim, para cada relacionamento deve ser configurado um relacionamento inverso.

Apesar do modelo da entidade criada ficar logo disponível para manipulação no código após a sua criação, é possível gerar tanto automaticamente como manualmente as subclasses desses objetos que serão usados para a criação de instâncias das entidades. O Core Data dá essa possibilidade de gerar dois ficheiros para dar suporte à classe, um ficheiro da classe e um ficheiro de propriedades. O ficheiro da classe declara a classe como uma subclasse de NSManagedObject [44], classe base que implementa o comportamento para um objeto de modelo Core Data. Apesar desse passo ser opcional, é importante citá-lo, visto que na Arrived, optou-se por implementá-lo.

O ficheiro de propriedades declara uma extensão para conter as propriedades do tipo NSManaged que representam atributos e relacionamentos, os seus "assessores" e a funcionalidade auxiliar para encontrar instâncias deste tipo. O Core Data trata da geração de subclasses de objetos geridos, mas o programador assume o controlo quando precisa adicionar lógica ou editar propriedades. No Anexo E, é possível obter mais detalhes da geração desses ficheiros.

Após a criação do modelo de dados e configuração de suas relações, o próximo passo é o *setup* do **Core Data Stack** [45] (ver Anexo F), que permite a gestão e persistência da camada do modelo da aplicação. A *framework* Core Data contém um conjunto de classes que oferecem suporte colaborativo à camada de modelo de uma aplicação iOS (ver Figura 26):

- NSManagedObjectModel [46] - Uma classe que descreve os tipos da aplicação, incluindo as suas propriedades e relacionamentos. Essa classe é a representação

programática do ficheiro do modelo de dados gerado no início da configuração do Core Data. Esse modelo contém um ou mais objetos NSEntityDescription [47] que representam as entidades no esquema. Cada objeto NSEntityDescription possui objetos de descrição de propriedade que representam as propriedades da entidade no esquema;

- NSManagedObjectContext [48] - Uma classe que rastreia alterações em instâncias dos tipos da aplicação. Um contexto é um grupo de objetos de modelo relacionados que representam uma vista internamente consistente de um ou mais armazenamentos persistentes. As alterações nos objetos geridos permanecem na memória do contexto associado até que o Core Data salve esse contexto num ou mais *persistent stores*. Uma única instância de um objeto gerido existe num e apenas um contexto, mas várias cópias de um objeto podem existir em diferentes contextos. Portanto, um objeto é exclusivo para um contexto específico;

- NSPersistentStoreCoordinator [49] - Uma classe que guarda e procura instâncias dos tipos da aplicação nos *stores*. Esse coordenador usa o modelo para ajudar na comunicação entre contextos e *persistent stores*. As instâncias de NSManagedObjectContext usam um coordenador para guardar gráficos de objetos em armazenamento persistente e para recuperar informações de modelo. Um contexto não é totalmente funcional sem um coordenador, porque precisa dele para aceder a um modelo. O coordenador é projetado para apresentar uma fachada aos contextos dos objetos geridos de forma que um grupo de armazenamentos persistentes apareça como um armazenamento agregado. Um NSManagedObjectContext pode então criar um gráfico de objeto com base na união de todos os armazenamentos de dados que o coordenador abrange;

- NSPersistentContainer [50] - Uma classe usada para configurar o modelo, o contexto e o *store coordinator* simultaneamente. Esse objeto é basicamente um *container* que encapsula a pilha de dados principais da aplicação e simplifica a criação e a gestão dessa pilha de dados principais ao lidar com a criação do NSManagedObjectModel, NSPersistentStoreCoordinator e do NSManagedObjectContext.

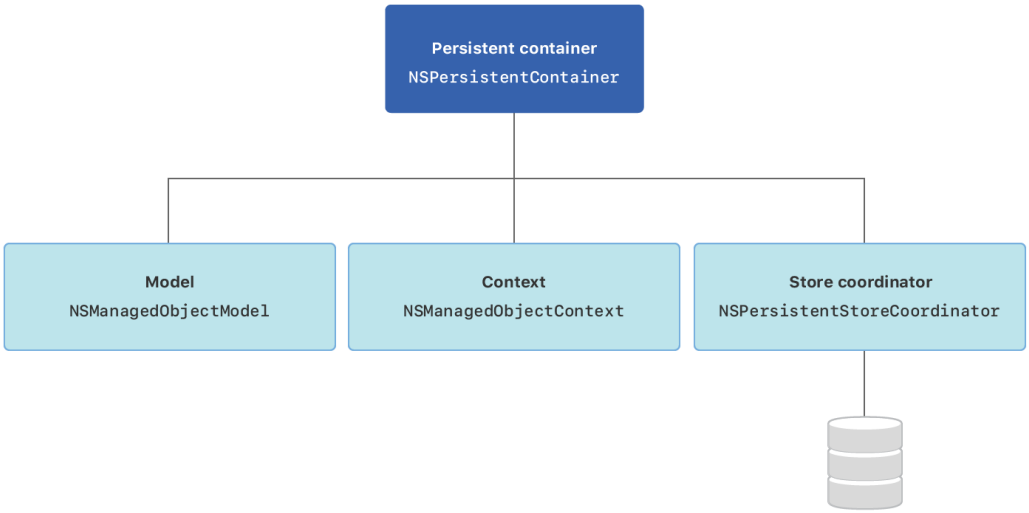

Figura 26 - Core Data Stack [45]

Quando se trabalha com Core Data, às vezes o código pode ser bastante detalhado e muitas vezes repetitivo, na maior parte dos casos as únicas coisas que mudam são os modelos consultados, portanto, o padrão Repository [51] é uma excelente opção para criar uma camada de abstração entre a camada de domínio e a camada de armazenamento. Este padrão, permite a facilidade em mediar entre a camada de domínio e a camada de dados [52]. No subcapítulo 5.3 é possível ver os detalhes da sua implementação na aplicação Arrived.

4.3. PhoneNumberKit

A *framework* PhoneNumberKit [53] é inspirada no libphonenumber [54] da Google com a função de analisar, formatar e validar números de telefones internacionais. As suas principais vantagens e funcionalidades são:

- Validar, normalizar e extrair os elementos de qualquer número de telefone em formato de *string*;
- Sintaxe do Swift e uma base de código legível e leve;
- Rapidez, com 1000 análises em aproximadamente 0,4 segundos;
- Ter os melhores *metadados* do projeto libphonenumber do Google;
- É totalmente testada para corresponder à precisão da implementação JavaScript do Google de libphonenumber;

- É construída para iOS e obtém automaticamente o código de região *default* do telefone;
- Contém um formatador editável **AsYouType** para UITextField [55];
- Converte códigos de países em nomes de países e vice-versa.

Para implementar a *framework* PhoneNumberKit, inicialmente é necessário instanciar um objeto denominado **PhoneNumberKit**, porque todas as interações com essa *framework* são feitas através desse objeto. O primeiro passo onde o PhoneNumberKit entra em ação é quando o utilizador preenche o formulário com o seu número de telemóvel na aplicação. O número do telemóvel do utilizador é obtido através de uma UITextField e para analisar a *string* obtida a função denominada parse da *framework* PhoneNumberKit é utilizada. O código de região é calculado automaticamente, mas pode ser substituído, se necessário. No caso da Arrived, essa definição foi mantida, porque o objetivo é aceitar números de todos os países.

O PhoneNumberKit faz automaticamente uma validação de tipo para garantir que o objeto criado seja válido, isso pode ser bastante caro em termos de desempenho e pode ser desativado, caso seja necessário. Ao ser implementado e testado na Arrived notou-se que o desempenho da aplicação não foi prejudicado, por esse motivo e por se considerar importante para a qualidade do *software* essa validação foi mantida. Esse método *parse*, no fim da validação devolve um objeto do tipo **PhoneNumber**. Os objetos **PhoneNumber** são estruturas Swift imutáveis e possuem as seguintes propriedades:

- *numberString;*
- *countryCode;*
- *nationalNumber;*
- *numberExtension;*
- *type (Mobile or Fixed).*

O PhoneNumberKit inclui também uma funcionalidade que permite analisar e validar uma grande quantidade de números de uma só vez, pois possui uma função de *parse* para *array* bastante eficaz. Sendo possível implementar as mesmas configurações da função de *parse* apresentada anteriormente e no fim da sua execução os números inválidos são

ignorados no *array* resultante. Embora seja interessante, essa funcionalidade não se adequou às necessidades de implementação da Arrived.

Outra funcionalidade importante do PhoneNumberKit e que é usada na implementação da Arrived, é a **AsYouTypeFormatter**, tal como o nome sugere, permite a formatação do texto no momento em que é inserido, e para a implementar, apenas foi preciso substituir o tipo do campo de texto de UITextField para PhoneNumberTextField e definir o **PhoneNumberKit** na opção Module do *storyboard* que contém esse campo, tal como podemos visualizar na Figura 27 que mostra o separador *Identity Inspector* do *storyboard* no Xcode com as respetivas alterações.

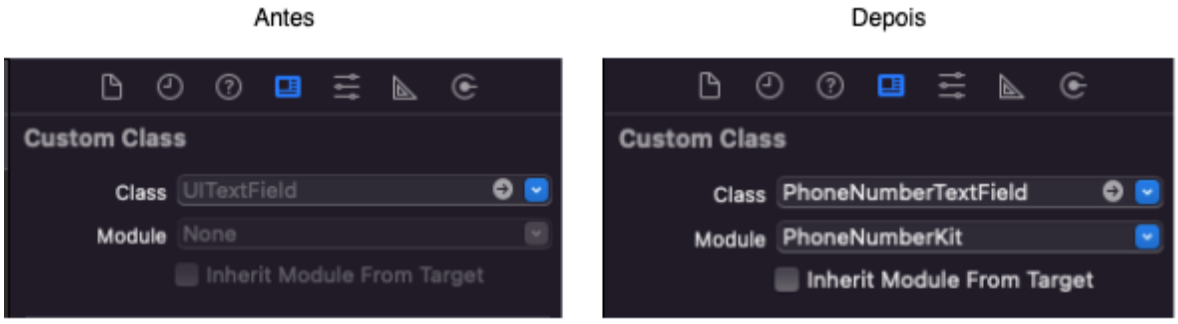

Figura 27 - PhoneNumberKit - AsYouType - UITextfield antes e depois

A UI do PhoneNumberTextField pode ser personalizada de várias formas com as seguintes definições:

- **withFlag**: exibe o código do país para a região atual e bandeira do país do lado esquerdo do campo de texto e com o seu tamanho definido com base no tamanho do texto;
- **withExamplePlaceholder**: que mostra no *placeholder* do campo um número de exemplo para a *currentRegion*;
- **withPrefix**: permite que o prefixo do código do país seja inserido e removido automaticamente ao editar as alterações do campo.

Tal como podemos visualizar na Figura 28, todas as três configurações foram ativadas na implementação da aplicação Arrived.

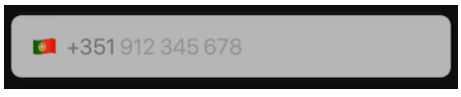

Figura 28 - Configurações da UI do PhoneNumberTextField

O PhoneNumberTextField formata automaticamente os números de telefone e oferece ao utilizador recursos completos de edição. Mas também é possível personalizar, ao usar o PartialFormatter diretamente. O código de região *default* é calculado automaticamente, mas pode ser substituído se necessário bem como os métodos de *parse* descritos inicialmente. No caso da Arrived essa configuração manteve-se como *default*, para serem aceites números de todos os países.

Como pudemos analisar, o PhoneNumberKit dispõe de uma panóplia de funcionalidades que facilitam a implementação desse tipo de validação. Além de todas as funcionalidades e configurações já descritas, é possível também consultar países para obter o indicativo ou obter um determinado país através de um indicativo. E por fim o programador pode aceder aos metadados que alimentam o PhoneNumberKit. Isso permite o desenvolvimento de quaisquer comportamentos, porque estes podem ser implementados no próprio PhoneNumberKit.

4.4. MapKit

A *framework* da Apple denominada MapKit [23] permite a exibição de imagens de mapas ou satélites, o destaque de pontos de interesse e a possibilidade de determinar informações de marcadores para coordenadas do mapa. Essa *framework* pode ser usada para diversos propósitos, tais como:

- Integrar mapas diretamente em vistas de uma aplicação iOS;
- Adicionar anotações e sobreposições a um mapa para destacar pontos de interesse;
- Fornecer *autocomplete* ou sugestões para a pesquisa de um destino ou ponto de interesse, de forma a auxiliar o utilizador.

A Arrived, tira partido de todos os três propósitos do MapKit, onde o utilizador tem a possibilidade de adicionar uma localização tanto para as suas moradas principais: casa, trabalho e escola, como para as localizações adicionadas no momento da criação de um aviso ou pedido de aviso. Para alcançar esse objetivo a Arrived conta com uma vista do mapa denominada MKMapView [56] (ver Figura 29). A MKMapView apesar de ter a palavra *view* no nome, não se limita apenas a essa característica, por ser uma *interface* de mapa incorporável, bastante semelhante à que a aplicação Maps [57] fornece.

Figura 29 - Vista de mapa da framework MapKit

Usada para exibir informações e manipular o conteúdo do mapa de uma aplicação a classe MKMapView, como podemos ver na Figura 29, fornece a possibilidade de centralizar o mapa numa determinada coordenada, especificar o tamanho da área que o utilizador deseja exibir e adicionar anotações no mapa com informações personalizadas. Ao se inicializar a classe MKMapView, é especificada a região inicial para que esse mapa seja exibido e essa região é definida por um ponto central e uma distância horizontal e vertical, como uma extensão que define quanto do mapa deve ser visível e como é definido o nível de zoom.

Na Arrived essa região foi definida com o ponto central na localização atual do utilizador, essa localização é obtida através da *framework* **Core Location** [22], que será detalhada no subcapítulo 4.5 e com a distância horizontal e vertical de 500 metros. Apesar dessa definição ter de ser efetuada inicialmente e programaticamente, a classe MKMapView suporta muitas interações *standard* para alterar a posição e o nível de zoom do mapa e a vista do mapa suporta gestos como o movimento em pinça para aumentar e diminuir o *zoom* e *drag* para navegar pelo mapa. O suporte para esses gestos é habilitado por *default*, mas também pode ser desabilitado usando as propriedades **isScrollEnabled** [58] e **isZoomEnabled** [59].

As informações sobre o comportamento da visualização do mapa são possíveis de obter ao fornecer um *delegate* [60]. A MKMapView invoca os métodos do seu *delegate* personalizado para informá-la sobre as alterações no *status* do mapa e para coordenar a exibição de anotações personalizadas. O *delegate* pode ser qualquer objeto na aplicação, desde que esteja conforme o protocolo MKMapViewDelegate [61]. Na Arrived esse protocolo foi implementado para gerir as vistas das anotações do mapa, mais

especificamente com o método que devolve uma instância de **MKAnnotationView** [62] sendo a representação visual de cada anotação do mapa.

Como foi referido anteriormente, num primeiro momento a vista do mapa é exibida nas configurações iniciais do utilizador, que por sua vez poderá indicar as suas localizações principais ao fazer o *tap* no mapa na localização desejada. Essa ação permite a criação de uma anotação na vista do mapa, essa anotação é do tipo **MKPointAnnotation** [63] que representa os dados de um local aplicado a um ponto específico num mapa. Esses dados são, especificamente as coordenadas geográficas desse ponto e um título, que no caso da Arrived é o nome da configuração (casa, trabalho, escola e outra).

Através dessa anotação é criada a MKAnnotationView com os dados especificados e com o ícone da Arrived, tal como podemos visualizar na Figura 30. Ao fazer um *tap* por cima do ícone é possível visualizar o título desta anotação.

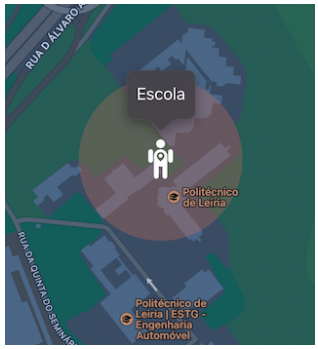

Figura 30 - Título de uma MKAnnotationView na vista do MapKit

Outra forma que o utilizador tem para adicionar as suas localizações no mapa é efetuar a pesquisa através de uma *search bar,* onde são mostradas sugestões baseadas no que é inserido, tal como podemos visualizar na Figura 31. O MapKit permite essa facilidade de pesquisa através de três objetos:

1. O objeto utilitário denominado **MKLocalSearch** [64], que inicia pesquisas baseadas em mapas com moradas e pontos de interesse e processa os resultados obtidos. Após a conclusão da solicitação, o objeto entrega os resultados ao **completionHandler** [65] escolhido, que por sua vez poderá devolver uma resposta com a listagem de localizações sugeridas ou um objeto do tipo **Error** [66], caso haja algum problema ao obter essa resposta;
2. O **MKLocalSearchCompleter** [67] é o objeto responsável por gerar a lista de localizações com base na *string* fornecida através da pesquisa. Esse objeto é usado para recuperar sugestões de preenchimento automático para os próprios controlos de

pesquisa baseados em mapa. À medida que o utilizador digita o texto, a sequência de texto atual é alimentada no *search completer*, que fornece possíveis resultados correspondente a locais ou pontos de interesse. Quando o valor da propriedade **queryFragment** [68] desse objeto é atualizado uma pesquisa é iniciada. Essa propriedade pode ser atualizada em tempo real à medida que o utilizador digita novos caracteres num campo de texto porque o objeto *completer* aguarda um curto período de tempo para que a *string* de consulta se estabilize. Quando as modificações na *string* de consulta são interrompidas, o *completer* inicia uma nova pesquisa e devolve os resultados ao seu representante como uma matriz de objetos MKLocalSearchCompletion [69];

3. O objeto **MKLocalSearchCompletion**, corresponde a uma *string* totalmente formada que completa uma *string* parcial, ou seja, não é criada instância dessa classe diretamente porque primeiramente o MKLocalSearchCompleter é usado para iniciar uma pesquisa baseada num conjunto de *strings* parciais. E esse objeto por sua vez, armazena as correspondências na sua propriedade *results*. E por fim, é possível recuperar quaisquer objetos MKLocalSearchCompletion dessa propriedade e exibir os termos de pesquisa na interface.

Figura 31 - Pesquisa de localização feita com apoio do MapKit

A listagem de localizações sugeridas através das pesquisas são apresentadas através de uma **UITableView** [37] sendo uma vista de tabela usada para apresentar os dados (ver Figura 31). O valor obtido quando o utilizador seleciona um dos resultados é do tipo MKLocalSearchCompletion e após obtê-lo é criada uma instância do objeto **Request** [70] do MKLocalSearch que recebe o valor obtido como *input*. Esse objeto representa os parâmetros a serem usados na pesquisa de pontos de interesse no mapa e o mesmo é utilizado para instanciar o próprio MKLocalSearch.

O MKLocalSearch, por sua vez, tal como especificado mais acima neste capítulo, permite iniciar a pesquisa que devolve uma lista dos pontos de interesse no mapa e dessa lista apenas o primeiro *item* é escolhido e, desse *item,* são obtidas as propriedades coordenadas geográficas e nome. Tal como na ação de *tap* na vista do mapa, ao obter essa

informação através da pesquisa, é criada uma anotação no mapa com as mesmas informações, nome da localização e coordenadas geográficas.

Por fim, mas não menos importante, outra característica importante do MapKit é a possibilidade de disponibilizar rotas de transporte público no Maps, que pode ser usado também para complementar as rotas fornecidas na aplicação. Por exemplo, se a aplicação fornecer apenas rotas para viagens de metropolitano, o Maps pode ser usado para fornecer rotas a pé, de e para estações de metropolitano. Apesar de ser uma característica interessante, a mesma não se aplicou no contexto da Arrived.

4.5. Core Location

A *framework* Core Location da Apple [22] produz serviços que determinam a localização geográfica, altitude e orientação de um dispositivo, ou a sua posição em relação a um dispositivo iBeacon [71] próximo. A estrutura recolhe dados usando todos os componentes disponíveis no dispositivo, incluindo Wi-fi, GPS, Bluetooth, magnetómetro, barómetro e *cellular hardware*.

Para configurar, iniciar e parar os serviços Core Location, são usadas instâncias da classe **CLLocationManager** [72] que oferece suporte às seguintes atividades relacionadas à localização:

- Atualizações de localização padrão e significativas - Permite acompanhar grandes ou pequenas alterações na localização atual do utilizador com um grau de precisão configurável;
- Monitorização da região - Permite monitorizar regiões distintas de interesse e gerir eventos de localização quando o utilizador entrar ou sair dessas regiões;
- Alcance do *beacon* - Permite localizar e detetar *beacons* próximos;
- Bússola - Permite reportar as mudanças de rumo da bússola a bordo.

Para usar os serviços de localização, a aplicação solicita autorização e o sistema solicita que o utilizador conceda ou negue a solicitação, tal como podemos visualizar na Figura 32. Através da aplicação Configurações os utilizadores podem alterar as configurações do serviço de localização a qualquer momento, e isso poderá afetar tanto aplicações individuais como o dispositivo como um todo. A Arrived executa o Core Location e através dessa implementação recebe eventos, incluindo alterações de autorização.

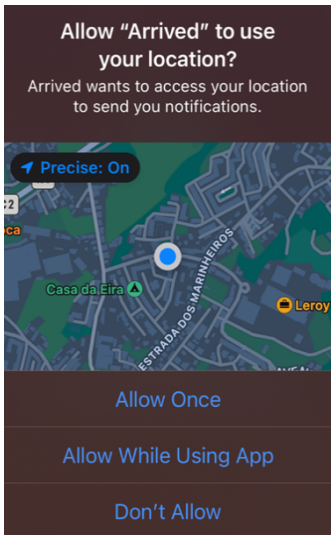

Figura 32 - Solicitação de permissão para os serviços de localização do Core Location

Para adicionar esses serviços de localização a uma aplicação, o **CLLocationManager** é criado, e o seu *delegate* é implementado sendo decidido o modo de autorização que a aplicação requer. Quando a aplicação é executada, este verifica se o dispositivo oferece suporte a serviços de localização, configura e inicia os serviços de localização desejados e solicita autorização para receber a localização do utilizador. Após autorizado, a aplicação recebe eventos de localização para os serviços de localização principal solicitados.

É importante determinar se o dispositivo oferece suporte a serviços de localização. Os serviços de localização principal exigem *hardware* que pode não estar presente em todos os dispositivos. Após essas verificações a instância da classe **CLLocationManager** é criada. Na Arrived foi criada a classe **ArrivedLocationManager**, que por sua vez, tem a função de criar a instância do **CLLocationManager** e implementar as funções relativas a si, bem como o seu *delegate*.

A **ArrivedLocationManager** é instanciada no arranque da aplicação, mais especificamente no AppDelegate [30], para gerir algumas interações com o sistema. Na criação da instância do **CLLocationManager**, o seu *delegate* é atribuído antes de iniciar qualquer serviço de localização. Tal como os métodos relacionados ao serviço de localização, é igualmente importante a implementação dos métodos relacionados a falhas no *delegate* quando os serviços de localização não estiverem disponíveis no dispositivo.

Ao tentar iniciar um serviço indisponível, o objeto **CLLocationManager** invocará um dos métodos relacionados a falhas do seu *delegate*. E isso permite que seja exibida uma mensagem de erro com essa informação ao utilizador, para alertá-lo desse impedimento.

Após essas ações são invocados os métodos apropriados para as ações que o *location manager* irá executar. No caso da Arrived, corresponde à solicitação de permissão do utilizador para usar os serviços de localização independentemente se a aplicação estiver em uso e à configuração para a aplicação receber atualizações de localização ao ser executada também em *background*.

Na Arrived, todas as propriedades associadas a esse serviço foram configuradas com precisão. O Core Location gere o consumo de recursos, desligando alguns componentes de *hardware* quando não é necessário. Por exemplo, ao definir a precisão desejada para eventos de localização para um quilómetro, permite ao *location manager* a flexibilidade de desligar o *hardware* de GPS e contar apenas com o Wi-fi ou *cell radios*, o que pode reduzir o consumo de recursos de *hardware* significativamente.

Em relação às funcionalidades usadas do CLLocationManager, além das citadas anteriormente que permitem a inicialização do serviço temos a possibilidade de indicar o momento em que se inicia a geração de atualizações que informam a localização atual do dispositivo, o que faz com que seja possível usar esse recurso apenas quando realmente for necessário.

A implementação da Arrived tirou partido não só dos métodos já existentes do Core Location, mas também de alguns dos seus recursos, e através deles foi implementado o método **getPlace**, que apesar de não ser um método nativo do **CLLocationManager**, encaixa-se nessa categoria por utilizar recursos do mesmo. Esses recursos começam pelo parâmetro recebido por essa função que é do tipo **CLLocation** [73] que representa uma coordenada geográfica com informações de precisão e *timestamp*. Esse método tem a função de transformar essa localização recebida num **CLPlacemark** [74], sendo basicamente uma descrição amigável de uma coordenada geográfica e essa descrição normalmente contém o nome do local, o endereço e outras informações relevantes.

Para obter esse **CLPlacemark**, é inicialmente criada uma instância da *interface* **CLGeocoder** [75] usada para a conversão entre coordenadas geográficas e nomes de lugares. Esse objeto obtido através dessa instância é um geocodificador e é de captura única que funciona com um serviço baseado em rede para pesquisar informações de *placemark* para o valor de coordenada especificado. Para usar um objeto geocodificador, após criado, apenas é necessário invocar um dos seus métodos de *forward* ou *reverse geocoding* para iniciar a solicitação.

As solicitações de *forward geocoding* tratam um endereço legível pelo utilizador e encontram o valor correspondente de *latitude* e *longitude* e essas solicitações também podem devolver informações adicionais sobre o local especificado, como um ponto de interesse ou construção. Já as solicitações de *reverse geocoding*, funcionam de forma quase contrária, ao obter um endereço legível através de uma coordenada geográfica.

Na Arrived foi efetuada uma solicitação desse tipo e para isso foi utilizado um método do **CLGeocoder** chamado **reverseGeocodeLocation** [76] que por sua vez recebe a localização do tipo **CLLocation** que foi especificada inicialmente e recebida no método **getPlace** e através dessa localização consegue encontrar um endereço legível pelo utilizador. Para obter esse endereço, esse método envia os dados de localização especificados para o servidor de *geocoding* de forma assíncrona e quando a solicitação é concluída, o geocodificador executa o *completion handler* fornecido na *main thread*. Caso seja bem-sucedido o método devolve um objeto do tipo **CLPlacemark** e caso a solicitação seja cancelada ou haja um erro na obtenção das informações do *placemark*, esse parâmetro será nulo. E se for o caso de um erro, o mesmo é devolvido com o motivo do *placemark* não ter sido devolvido.

Para ambos os tipos de solicitação, os resultados são devolvidos usando um objeto **CLPlacemark**. Mas no caso de solicitações de *forward geocoding*, vários *placemarkers* podem ser devolvidos se as informações fornecidas resultarem em vários locais possíveis.

Essas solicitações de *geocoding* são bastante úteis, mas devem ser usadas com alguma cautela porque são limitadas por taxa para cada aplicação, portanto, fazer muitas solicitações num curto período pode fazer com que algumas das solicitações falhem. Quando a taxa máxima é excedida, o geocodificador devolve um erro com essa informação. No desenvolvimento da Arrived, esse ponto foi considerado, pois o seu uso é feito em apenas dois momentos:

1. No momento em que o utilizador configura as suas moradas principais e opta por selecionar através do mapa. Quando é confirmada essa localização, esse método é invocado sento mostrado um *dialog* com a descrição dessa localização, tal como podemos visualizar na Figura 33;

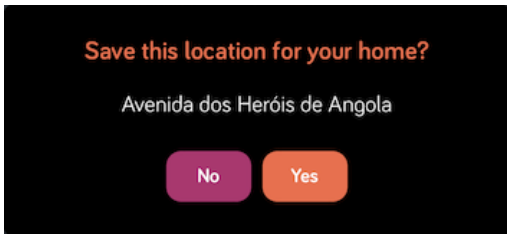

Figura 33 - Confirmação de localização obtida por reverse geocoding

2. No momento em que um aviso ou um pedido de aviso é criado, onde o utilizador tem a possibilidade de selecionar a localização desse aviso através da seleção da localização no mapa, tal como no ponto 1.

Por fim, em relação aos métodos do CLLocationManagerDelegate que foram usados na Arrived temos:

- O método **didChangeAuthorization** [77], que informa ao *delegate* quando a aplicação cria o *location manager* e quando o status de autorização é alterado. No caso da Arrived, se o *status* é autorizado o método **startUpdatingLocation** é invocado;

- O método **didUpdateLocations** [78], que informa ao *delegate* que novos dados de localização estão disponíveis. No caso da Arrived, obtém essa localização atual do utilizador e armazena nas UserDefaults [79] do dispositivo, que será posteriormente usada nas vistas dos mapas como localização inicial, tal como foi detalhado no subcapítulo 4.4;

4.6. User Notifications

A *framework* User Notifications [24] da Apple, permite o envio de notificações para o dispositivo do utilizador que podem ser geridas tanto localmente como a partir de um servidor. Essas notificações transmitem informações importantes aos utilizadores da aplicação, mesmo que esta não esteja a ser executada. Por exemplo, quando o utilizador recebe um lembrete através de uma notificação para não se esquecer de realizar uma tarefa definida por ele anteriormente, ou também podem indicar à aplicação para efetuar o *download* de informações e atualizar a sua *interface*. Essas notificações podem exibir tanto um alerta, como reproduzir um som ou marcar o ícone do aplicativo.

Para as notificações locais a aplicação cria o conteúdo da notificação e uma condição é especificada, como hora ou local, que aciona a entrega da notificação. Para notificações remotas, é necessário ter um servidor que gera notificações *push* e o serviço Apple Push

Notification [80] lida com a entrega dessas notificações aos dispositivos do utilizador (ver Figura 34). De modo geral, esta *framework* permite:

- A definição dos tipos de notificações compatíveis com a aplicação;
- A definição de quaisquer ações personalizadas associadas aos tipos de notificação definidas na aplicação;
- O agendamento de notificações locais para entrega;
- O processamento de notificações já entregues;
- Dar resposta às ações selecionadas pelo utilizador.

Figura 34 – Comparação entre Notificação Local e Notificação Push

Apesar da *framework* User Notification permitir não só o envio de notificações locais como também notificações remotas, o foco deste subcapítulo será nas notificações locais, por serem o tipo de notificações usadas no desenvolvimento da aplicação Arrived. No entanto, alguns dos passos que serão descritos sobre a implementação servem para ambos os tipos de notificação.

O primeiro passo para a implementação das notificações, é a solicitação de permissão para exibir os alertas, reproduzir sons ou marcar o ícone da aplicação em resposta a uma

notificação [81] (ver Figura 35). Essas ações acontecem quando a aplicação está em *background* ou fechada, o que permite informar ao utilizador que a aplicação tem informações relevantes para serem visualizadas. Como o utilizador pode considerar as interações baseadas em notificação disruptivas, é necessário ter permissão para usá-las.

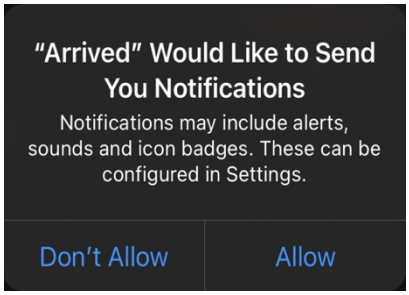

Figura 35 - Solicitação de permissão para o envio de notificações

Um ponto importante a ter em atenção é que mesmo que o utilizador autorize o recebimento de notificações, ele poderá sempre alterar essa permissão nas configurações do dispositivo. Por este motivo sempre que for necessário a criação de uma notificação, é importante que a verificação do status de permissão seja feita antes, e essa *framework* fornece mecanismos para que isso seja possível.

O próximo passo é a criação de uma notificação [82]. Existem três opções que servem para determinar quando a aplicação entrega a notificação ao utilizador:

1. Intervalo de tempo – Notificação baseada num determinado intervalo de tempo, é usada para agendar temporizadores e pode ser configurado para ser enviado só uma vez ou repetir após o tempo de intervalo acabar;
2. Calendário – Notificação baseada numa determinada data e hora, caso só a hora seja definida na criação do alerta, a notificação será enviada no horário especificado;
3. Localização – A notificação baseada em localização é despoletada quando o utilizador entra ou sai de uma região. Internamente o iOS utiliza *geofences* para monitorizar essas regiões (para mais informações sobre esse tema consulte o subcapítulo 1.3).

Todas as opções usadas para gerar um *trigger* de notificação, permite que a opção de repetir ou não o seu envio seja definido no momento da sua criação. Este subcapítulo será dedicado às notificações despoletadas por localização, por se tratar do *trigger* de notificação implementado na aplicação Arrived. Para a criação desse tipo de *trigger* de notificação é necessário a permissão do utilizador para obter a localização do dispositivo, isso é possível

através da *framework* Core Location (para mais detalhes dessa funcionalidade consulte o subcapítulo 4.5).

Após a autorização de localização ser concedida a notificação poderá ser criada. E para isso os seguintes parâmetros são necessários:

- *NotificationContent* - Conteúdo da notificação:
    - *title* – Título da notificação;
    - *body* – Mensagem da notificação;
    - *categoryIdentify* – Identificador para a categoria da notificação, é usado para determinar as ações apropriadas a serem exibidas para a notificação, este parâmetro será detalhado mais à frente neste subcapítulo;
    - *threadIdentifier* – O identificador exclusivo da *thread* relacionada a esta solicitação de notificação. É usado para agrupar visivelmente as notificações.
- *CircularRegion* (Região circular / *geofence*):
    - *location* – Coordenadas geográficas do centro da região circular;
    - *radius* – Raio da região circular definido em metros;
    - *notifyOnEntry – Flag* que indica se a notificação é para ser envida quando o dispositivo se aproximar da região definida;
    - *notifyOnExit – Flag* que indica se a notificação é para ser enviada quando o dispositivo se afastar da região definida.
- *Localization Notification Trigger* – Como o nome diz, é o acionador da notificação e além de um identificador único, recebe os parâmetros anteriores, o conteúdo e a região da notificação.

Após a criação da notificação, a sua entrega é feita após a ação definida para esta se concretizar, ou a aproximação ou o afastamento do dispositivo com base na sua região circular, na figura Figura 36 podemos ver essa notificação recebida. Apesar de todas as etapas de implementação para este alerta ainda não terem acabado, para um melhor entendimento dessa tecnologia, foi adotada a abordagem da explicação de cada parte conforme as etapas de utilização.

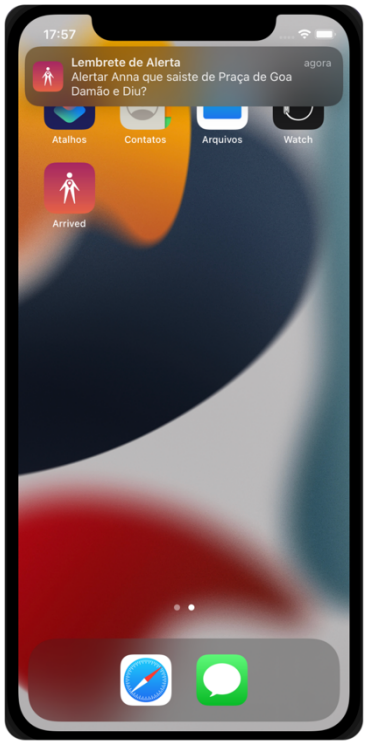

Figura 36 - Notificação da Arrived recebida através da *framework* User Notification

Posto isso, como podemos observar na Figura 37, ao fazer *scroll down* na notificação apresentada, um botão com a descrição "Alertar" surge, e esse botão corresponde a uma ação da notificação, sendo possível haver outros botões com outras ações associadas, ou até nenhum, já no caso da Arrived apenas um foi necessário. Para a criação dessa ação de notificação, além de um identificador que identifica exclusivamente a ação, os seguintes parâmetros precisaram ser indicados:

- *Title* – Representa o texto apresentado no botão da notificação;
- *Options* – Representa o comportamento associado à ação que poderão ser uma ou mais das seguintes opções:
    - *authenticationRequired* - Apenas executa a ação se o dispositivo estiver desbloqueado. Caso o dispositivo esteja bloqueado, o sistema solicitará que o utilizador o desbloqueie;
    - *destructive* - Executa uma tarefa destrutiva;
    - *foreground* - Esta opção faz com que aplicação seja iniciada em primeiro plano. Apenas esta opção foi usada no desenvolvimento da Arrived.

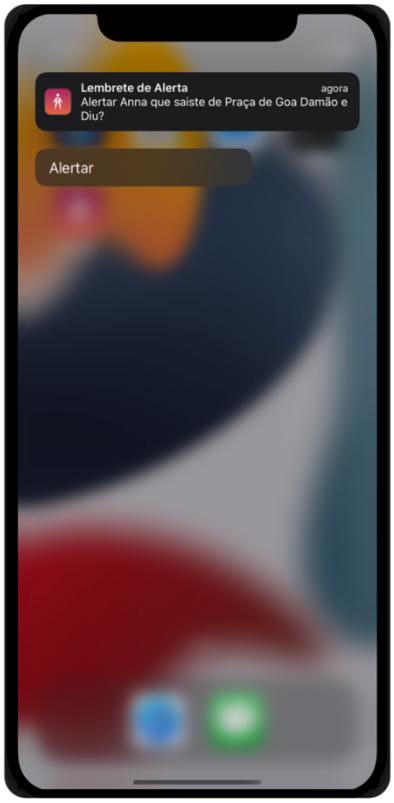

Figura 37 - Ação de uma notificação da Arrived recebida através da *framework* User Notification

Relativamente à criação das categorias de notificação, através delas é possível definir o tipo de notificações que uma aplicação poderá receber. Para definir uma categoria, para além do identificador usado para identificá-la exclusivamente, os seguintes parâmetros também são necessários:

- *actions* – Ação ou ações definidas anteriormente;
- *intentIdentifiers* – Informam ao sistema que a notificação está relacionada a uma solicitação feita pela Siri;
- *options* – Indicam como lidar com as notificações associadas a essas categorias.

No caso da Arrived, tanto o parâmetro *intentIdentifiers* como *options* não foram preenchidos por não serem precisos no âmbito da sua implementação. Por fim, após a definição das categorias, as mesmas são atribuídas ao sistema de notificação que passará a conhecer as suas categorias e respetivas ações. Tal como foi citado anteriormente essa categoria é atribuída no momento da criação da notificação através do parâmetro categoryIdentify.

4.7. Message UI

A *framework* Message UI [83] da Apple, fornece *view controllers* próprios para apresentar *interfaces* de mensagens de *e-mail* e SMS. Essas *interfaces* são usadas para que o utilizador tenha acesso a esses serviços sem que tenha a necessidade de sair da aplicação. Essa *interface* é apresentada na aplicação ao utilizador de forma modal permitindo-lhe a opção de personalizar o conteúdo da mensagem antes de enviar ou até cancelar o seu envio (ver Figura 38). O objeto *delegate* do controlador lida com a ação efetuada pelo utilizador após a *interface* ser fechada.

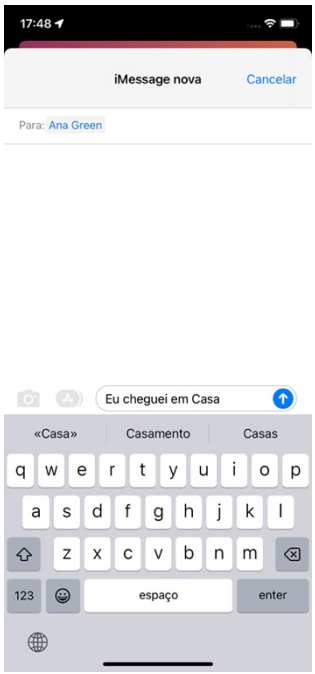

Figura 38 - Interface do envio de mensagens de texto da framework Message UI

Apesar dessa *framework* disponibilizar *interfaces* não só o para o envio de SMS, mas também para o envio de *email*, apenas a *interface* do envio de SMS foi utilizada no desenvolvimento da Arrived. A Message Composition Interface é basicamente um *view controller* padrão que permite que o utilizador escreva e envie não só SMS como também Multimedia Messaging Service (MMS) e iMessage. Antes dessa interface ser apresentada pela aplicação que a implementa, alguns campos precisam ser preenchidos, como a mensagem a ser enviada e os seus destinatários. Após ser apresentada, o utilizador poderá efetuar qualquer alteração a esses campos antes do envio. Essa *interface* não garante o envio da mensagem, porque o utilizador poderá sempre cancelar o seu envio, mas caso ele opte por enviar a aplicação nativa iOS Messages assume a responsabilidade do envio.

É importante frisar que antes de invocar essa *interface*, tem de haver uma verificação se o dispositivo está configurado para o envio de mensagens e caso não esteja, o utilizador deverá ser notificado, sendo este método de verificação disponibilizado pela *framework*. Depois da verificação o **MFMessageComposeViewController** [84] é criado, os campos das mensagens são preenchidos e esse controlador é invocado tal como qualquer outro da aplicação. Essa vista não é fechada automaticamente, mas sim conforme a ação do utilizador, após a ação o **MFMessageComposeViewController** envia o resultado da operação de volta para o controlador que o invocou, com a informação se o envio foi feito com sucesso, se foi com insucesso ou se foi cancelado.

Na Arrived essa *framework* é crucial para comunicação entre utilizadores, visto que a aplicação não dispõe de uma API. O envio de SMS é usado nas seguintes situações:

- Quando o utilizador deseja convidar um amigo para instalar a aplicação, poderá fazê-lo através da opção "Convidar Amigos", através dessa opção é criada uma mensagem personalizada com o *link* da aplicação para a AppStore;

- Quando o utilizador cria um pedido de aviso para outro utilizador, é criada uma mensagem consoante o seu pedido, com todas as informações necessárias para o aviso ser criado pelo outro utilizador. Essas informações vão através de um URL com a identificação da aplicação, e ao selecioná-lo a aplicação é aberta, já com esse pedido de aviso preenchido, e o utilizador só precisará aceitá-lo ou recusá-lo;

- Após a criação de um aviso, quando o utilizador se aproxima ou se afasta de uma determinada localização determinada pelo aviso criado, o mesmo recebe uma notificação, para lembrá-lo de avisar o destinatário desse aviso, ao selecionar essa notificação, a mensagem é preenchida com o aviso e o(s) respetivo(s) destinatário(s), o utilizador apenas necessita de confirmar o seu envio.

4.8. Contacts

Uma das principais funções de um dispositivo móvel é o armazenamento de informações dos contactos de um utilizador, no caso do iOS essas informações são obtidas através da *framework* Contacts [85] da Apple. Apesar dessa *framework* permitir não só a leitura como também a edição dos contactos, o foco desse subcapítulo será na leitura dos contactos, por ser essa a base das funcionalidades implementadas na aplicação Arrived.

Um dos principais objetos dessa *framework* é o representado pela classe contacto **(CNContact)** [86], esse objeto é imutável e nele contém as propriedades de contacto, como o nome, a imagem ou os números de telefone. Tal como alguns dos outros serviços citados neste capítulo, para poder aceder aos contactos de um dispositivo, essa *framework* precisa que o utilizador lhe dê acesso a essa informação, por isso a execução da funcionalidade que será descrita a seguir só será possível, caso o utilizador conceda essa permissão que podemos ver na Figura 39.

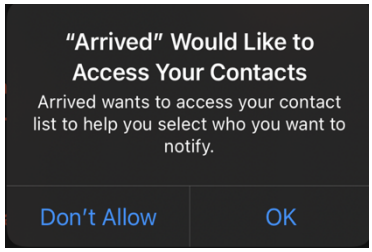

Figura 39 - Solicitação de permissão para aceder aos contactos

Umas das funcionalidades fornecidas pela *framework* Contacts é a pesquisa de contactos através do armazenamento de contactos **(CNContactStore)**, que basicamente é a base de dados de contactos do utilizador. O armazenamento de contactos encapsula todas as operações de entrada e saída e é responsável por pesquisar e guardar contactos e grupos. Várias formas de restringir os contactos devolvidos numa pesquisa são fornecidos pela *framework*, como predicados predefinidos e a propriedade denominada **keysToFetch**. Caso existam contactos com contas diferentes que representam a mesma pessoa, eles serão vinculados automaticamente e a pesquisa devolverá apenas um contacto com toda a informação.

Essa funcionalidade é usada na Arrived no âmbito de localizar mais detalhes de um contacto através de um pedido de aviso, sendo esse pedido de aviso recebido através de um SMS. A informação do aviso a ser criado juntamente com o contacto de quem o enviou é armazenada num URL, e a informação relativa ao contacto é apenas o seu número de telefone. Tal como podemos ver na Figura 40, para mostrar o pedido de aviso devidamente identificado, foi necessário obter mais dados do contacto e esses dados foram obtidos através da pesquisa de contactos desta *framework*, o que permitiu obter não só o nome, como também a foto do contacto. Obviamente que esse contacto tinha de estar presente na lista de contactos do dispositivo, caso contrário não seria possível obter os restantes dados, nesse caso seria apresentado apenas o número de telefone.

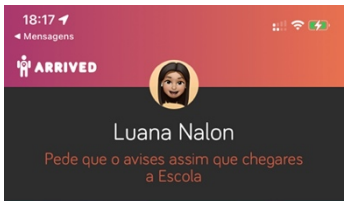

Figura 40 - Informação de um contacto no pedido de aviso

Outro ponto a ter em atenção é que um utilizador poderá ter contactos na conta local do seu dispositivo ou contas de servidor configuradas para sincronizar contactos. Cada conta tem pelo menos um *container* de contactos e um contacto pode estar em apenas um *container*. Um grupo é um conjunto de contactos dentro de um *container* e apesar de nem todas as contas suportarem grupos, algumas suportam até subgrupos.

Uma conta do iCloud tem apenas um *container* e pode ter muitos grupos, mas nenhum subgrupo. Por outro lado, uma conta do Exchange não tem grupos, mas pode ter vários *containers* representando pastas do Exchange. Esses *containers* foram considerados na implementação da funcionalidade de pesquisa da *framework* Contacts na Arrived.

Outro componente importante da *framework* Contacts, usada na Arrived é o **CNContactPickerViewController** [87], porque fornece uma *interface* que permite que o utilizador selecione um ou mais contactos da lista de contactos apresentada no controlador de visualização de contactos (ver Figura 41). Nesse caso não é necessário a autorização de acesso aos contactos do utilizador porque a aplicação só terá acesso após a seleção final do utilizador.

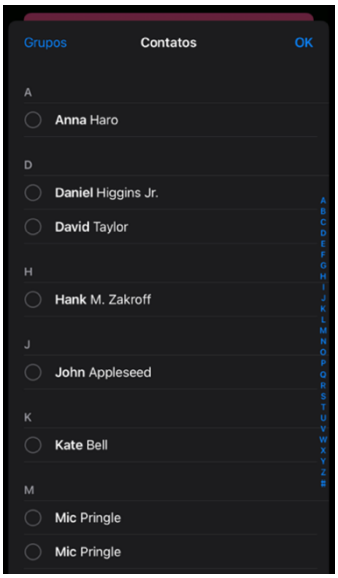

Figura 41 - Vista do CNContactPicker

Tal como na funcionalidade anterior, nessa funcionalidade também existem predicados predefinidos que permitem controlar a seleção do contacto pelo utilizador sendo que essa definição só terá efeito no momento que a exibição seja apresentada. No caso da Arrived o predicado foi definido para ser possível apenas a seleção de contactos que tenham um ou mais números de telefone, visto que a objetivo final da seleção desses contactos é o envio de SMS com os avisos predefinidos.

Para capturar as ações do utilizador na *interface* da seleção de contactos temos o componente **CNContactPickerDelegate** [88], que, por sua vez, permite a implementação de métodos para responder aos eventos do utilizador na seleção de contactos, sejam eles o cancelamento dessa seleção ou a obtenção dos contactos selecionados.

Por fim, tanto na seleção como na pesquisa dos contactos, o objeto ou objetos devolvidos são sempre no formato citado anteriormente, o **CNContact**, e para facilitar a manipulação dos dados obtidos desses contactos, na Arrived foi implementado um *parse* desse modelo para o modelo de domínio dos contactos, o que facilitou bastante a apresentação e persistência desses dados na aplicação.

5. Arrived

Neste capítulo será apresentada toda a estrutura que compõe a aplicação Arrived, nomeadamente a sua arquitetura, bem como todo o seu processo de desenvolvimento, que se inicia nas definições iniciais do projeto, implementação do padrão Repository e implementação de todas as suas funcionalidades. Por fim serão apresentados os testes desenvolvidos, bem como os resultados obtidos.

5.1. Arquitetura

A Arrived é uma aplicação desenvolvida para *smartphones* iOS, que corram uma versão superior ou igual a 15.0. A linguagem de implementação usada foi o Swift, seguindo principalmente o padrão de arquitetura MVVM, que propõe a separação da lógica de apresentação da lógica de negócio. Alguns outros padrões de arquitetura foram considerados na implementação da Arrived, nomeadamente, o Repository, que permitiu a criação de uma camada de abstração entre a camada de domínio e a camada de armazenamento e o Dependency Injection, que permitiu a redução da dependência entre componentes.

Conforme representado na Figura 42, podemos ver a estrutura da Arrived, onde o ponto central é a representação da aplicação iOS, e ao seu redor temos o utilizador, que utiliza a aplicação para criar lembretes, avisos, pedidos de avisos e receber avisos de outros utilizadores. Podemos observar também as *frameworks* usadas na sua implementação, inclusive a *framework* Core Data, usada para a persistência dos dados e a aplicação de mensagens da Apple, que por sua vez é usada para o envio de SMS com avisos e pedidos de avisos da Arrived. No capítulo 4 é possível obter mais detalhes da implementação de cada uma das *frameworks* usadas.

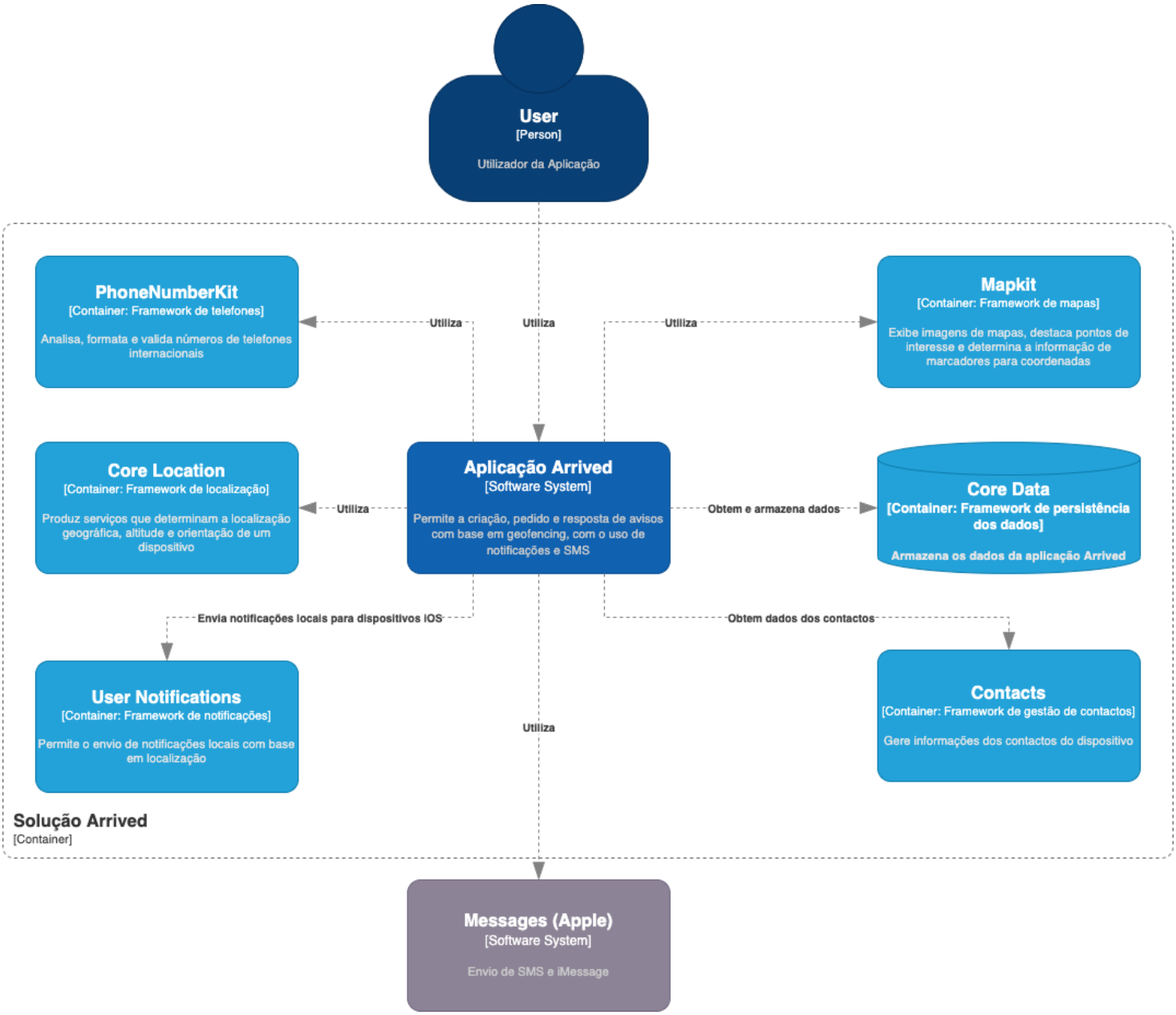

Figura 42 - Diagrama da Solução Arrived (Diagrama de Contentores - C4 nível 2)

Ao expandimos o contentor da aplicação Arrived, representado pela Figura 43 é possível ver com mais detalhe a estrutura implementada através do padrão MVVM. O uso desse padrão faz com que os dados apresentados na vista (*View*) através do *ViewController* se encontrem no *ViewModel e o ViewModel* assume a responsabilidade de obter e preparar os dados que serão apresentados na vista. Esses dados são alcançados recorrendo à persistência local permitida através do Core Data. Os dados armazenados, são obtidos e persistidos utilizando o respetivo *Repository* que utiliza o Modelo de Domínio para obter os dados do *ViewModel* e a Entidade/Modelo para persistir e ler esses dados localmente. É possível ver

mais detalhes de implementação dos repositórios, modelos de domínio e de dados no subcapítulo 5.3.

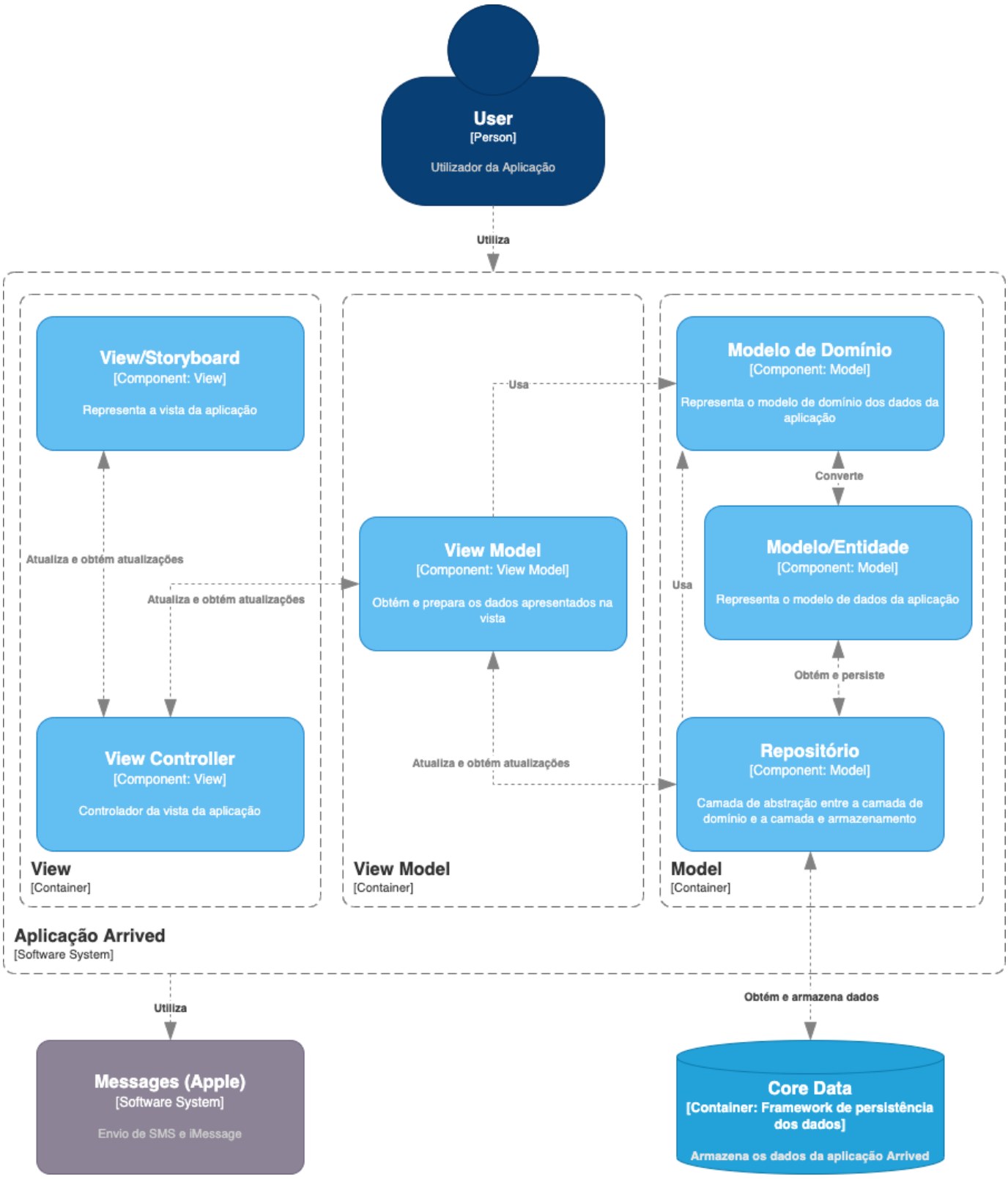

Figura 43 - Diagrama da Aplicação Arrived (Diagrama de componentes - C4 nível 3)

A estrutura de dados definida na Arrived é bastante simples, tal como podemos observar na Figura 44. Esta estrutura é composta por três modelos, o *Alert*, o *Location* e o *Contact*, sendo o Alert o modelo principal e, tal como podemos analisar na imagem, tem uma relação de *many to many* com o modelo *Contact*, ou seja, um *Alert* pode ser criado para mais de um contato e um contacto, poderá estar presente em um ou mais avisos. A outra relação do Alert é com a entidade Location, que por sua vez é uma relação *many to one*, onde um aviso poderá apenas ter uma localização associada, mas uma localização poderá estar presente em mais do que um aviso.

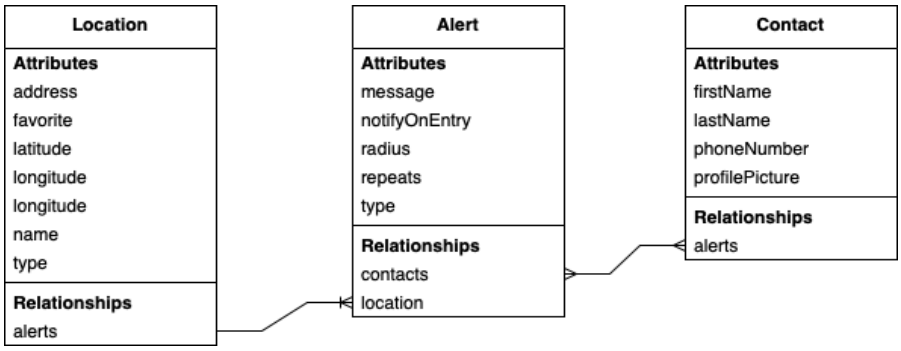

Figura 44 - Diagrama do modelo de dados do Core Data

5.2. Definições iniciais

A implementação de uma aplicação, independente da plataforma, necessita de definições iniciais. A maioria dessas definições, são obtidas através dos requisitos não funcionais, e no caso da Arrived estão listados no subcapítulo 3.2. Quando se trata de uma aplicação iOS, como a Arrived, essas definições são feitas logo após a criação do projeto no Xcode e podem ser alteradas no futuro caso seja necessário. Não existe uma ordem certa para essas configurações, mas vamos começar com o nome da organização no documento do projeto, que no caso da Arrived, por se tratar de um projeto desenvolvido não só para a VOID Software, mas também através da VOID Software, o nome da organização foi definido como "VOID Software SA".

O outro ponto importante que poderá variar conforme a empresa/autor do projeto é o cabeçalho de todos os ficheiros gerados no projeto, e na Arrived, pelos mesmos motivos da configuração anterior, foi necessário acrescentar a descrição do copyright, que neste caso em específico foi, "*Copyright © 2023 VOID Software SA. All rights reserved*.". Para

adicionar esse cabeçalho é adicionado à raiz do projeto, um ficheiro denominado IDETemplateMacros e ao criar esse ficheiro com o respetivo texto do *copyright*, todos os ficheiros criados futuramente no projeto irão conter esse texto no seu cabeçalho.

Outra configuração importante é a criação dos ficheiros de tradução, denominados Localizables, esses ficheiros permitem que a aplicação ofereça suporte a vários idiomas [89]. Apesar dessa configuração ser opcional, no caso da Arrived esta foi necessária por se tratar de uma aplicação que suporta dois idiomas, o português – PT e o inglês – USA. A configuração desses ficheiros é bastante simples, basicamente ao criar o ficheiro Localizable, o Xcode reconhecerá automaticamente que se trata de um ficheiro de traduções.

Ao selecionar os idiomas escolhidos, é criado um ficheiro Localizable para cada um deles. O conteúdo desse ficheiro é representado como chave e valor, sendo cada valor representado através de uma *string* e são essas chaves que substituirão as *strings* no código. As chaves são declaradas com o mesmo nome em ambos os ficheiros, o que muda é o conteúdo das *strings*, sendo que cada uma é escrita no seu respetivo idioma. Por fim, quando essa *string* for necessária em algum lugar no código, por exemplo, para ser apresentada em alguma vista, é necessário apenas invocá-la através da sua chave predefinida.

Por fim, quando a aplicação estiver a ser usada, os seus textos serão apresentados conforme o idioma e região definidos no dispositivo. Outra configuração que foi feita no contexto da Arrived foi a do Core Data. Apesar de esta configuração não ter sido feita logo no início do *setup* do projeto, por questões de abordagens de implementação, poderia estar neste subcapítulo, porque de acordo com o projeto poderá fazer sentido já iniciar a sua implementação com essa configuração feita. No subcapítulo 4.2 é possível obter mais detalhes de toda essa configuração.

5.3. Implementação do padrão Repository

Para ser possível a compreensão da estrutura implementada na Arrived, e de como essa estrutura contribui para a redução de código, foi criado um repositório genérico com a função de realizar as mesmas operações em qualquer objeto de modelo Core Data. O passo inicial foi a especificação do protocolo denominado **Repository**. Esse protocolo conta com o elemento que representa a entidade gerida pelo repositório e alguns métodos que representam as operações mais utilizadas como *get*, *create* e *delete*.

As classes que implementam este protocolo, independentemente de estarem a trabalhar com Core Data, Realm [90], UserDefaults [79] ou um serviço de API, terão que especificar o modelo/entidade com o qual trabalham. E por fim, a ideia principal é que os métodos devolvam resultados que consistem numa entidade para criação, uma listagem de resultados para uma consulta e um booleano se a exclusão foi bem-sucedida.

O próximo passo foi a implementação concreta do repositório Core Data genérico, que tem a função de lidar com os objetos de modelo do Core Data, a classe desse repositório, implementa o protocolo **Repository** e ao ser instanciada recebe a entidade genérica que poderá representar qualquer modelo de dados do Core Data. Este repositório conta com instância da classe **CoreDataStack**, o que faz com que o repositório possa aceder aos seus parâmetros e executar as operações necessárias no Core Data. Ao implementar o protocolo **Repository**, este repositório implementa os métodos especificados anteriormente, (*get, create e delete*) e cada um é responsável por devolver o resultado da sua execução.

É importante a separação entre os modelos de domínio dos modelos de armazenamento, porque os modelos de domínio não devem se preocupar com nenhum aspeto de como os dados são armazenados no Core Data. Ao ter isso em consideração, muito esforço de desenvolvimento futuro poderá ser poupado. Caso seja preciso, por exemplo, a divisão de um modelo em modelos menores para melhorar o desempenho da consulta do Core Data, ou talvez a implementação de outra *framework* de persistência como o Realm em substituição do Core Data. Deste modo, a camada de domínio não deve interagir diretamente com a persistência de dados.

Por fim, para lidar com o objeto de domínio foi implementado um repositório concreto para cada um. Esse repositório, contém a instância do repositório genérico e recebe a entidade que irá lidar. Neste repositório é feita a implementação dos métodos que irão de facto efetuar as operações de persistência no seu respetivo modelo de dados. Para os restantes modelos desenvolvidos na Arrived foi feita a mesma implementação descrita neste subcapítulo, tendo em conta as necessidades de cada modelo.

5.4. Implementação das funcionalidades

Neste subcapítulo, serão apresentados os detalhes de implementação de cada funcionalidade da aplicação Arrived, nomeadamente a recolha de dados do utilizador, a criação de um aviso, criação de um lembrete com base em localização, a criação de um

pedido de aviso, a resposta de um pedido de aviso, a listagem dos diferentes tipos de avisos e as opções do menu lateral da aplicação.

5.4.1. Recolha de dados do utilizador

Quando a aplicação Arrived é instalada, e o utilizador acede à aplicação pela primeira vez, terá de indicar alguns dados, que serão usados para a criação de avisos no futuro. Para isso, foi criada uma vista principal com a indicação dos dois tipos de dados que serão recolhidos:

1. Um formulário de registo do número de telemóvel, que podemos observar na Figura 45;

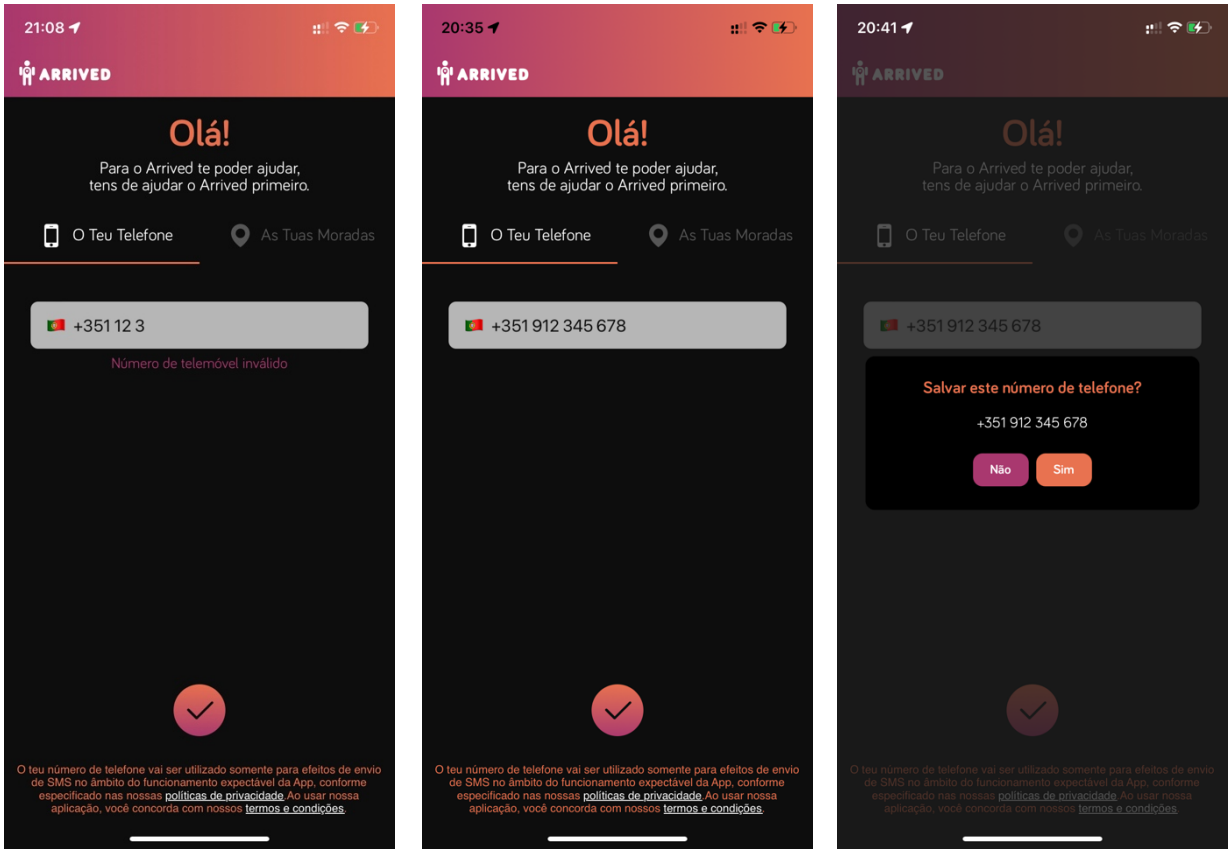

Figura 45 - Formulário de recolha do número de telemóvel do utilizador

2. Um formulário para a indicação da localização das moradas principais do utilizador: casa, trabalho, escola e outros como podemos observar na Figura 46.

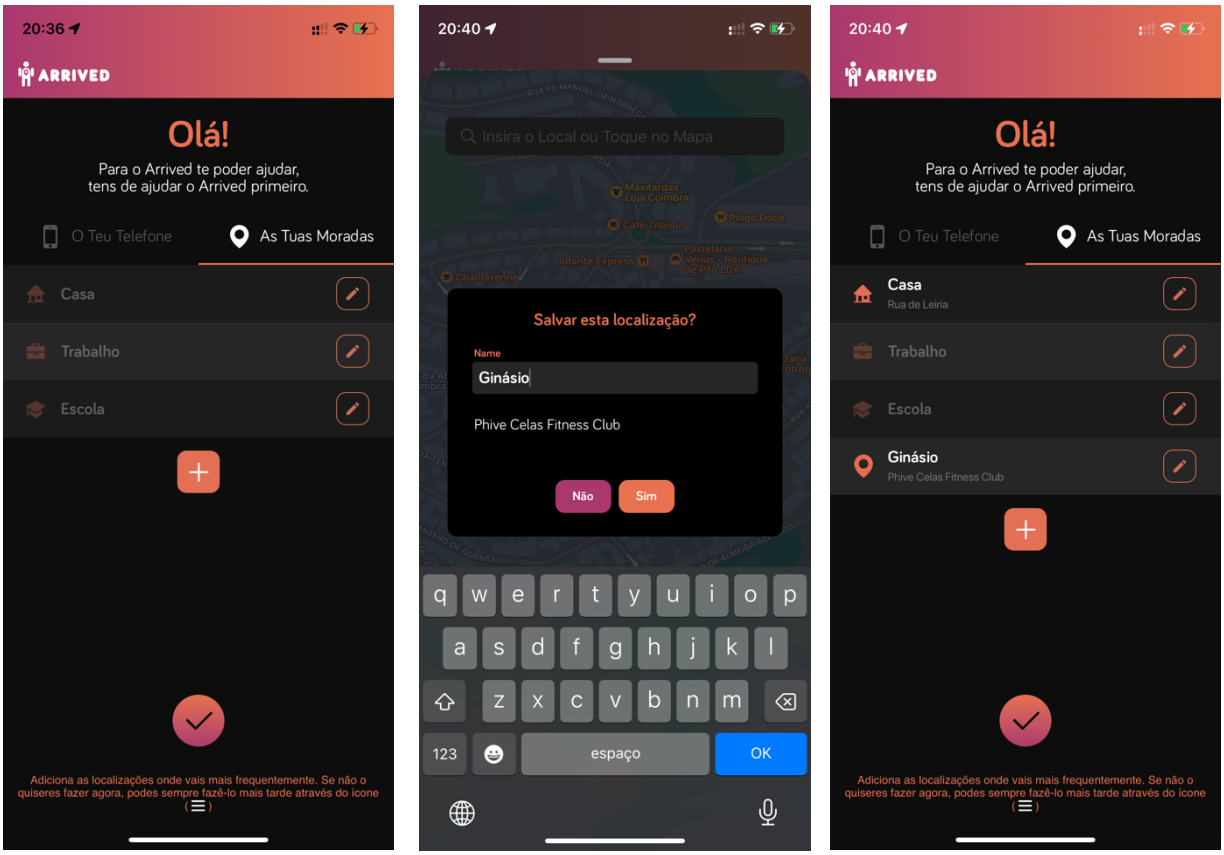

Figura 46 - Setup das localizações principais do utilizador

A navegação entre os dois tipos de formulários é feita, não só pelas opções que podemos ver na parte de cima da vista principal, mas também, através do botão de confirmação que aparece por baixo. A primeira vista apresentada ao utilizador é o formulário para a recolha do seu contacto, e onde constam também os *links* para a política de privacidade e termos e condições (ver Figura 45). O campo de contacto é de caráter obrigatório e o objetivo da recolha do número de telemóvel do utilizador é para que no futuro ele possa utilizar uma das funcionalidades da Arrived, que permite a criação de pedidos de aviso para outro utilizador. Para esse pedido ser enviado é necessário que o seu contacto conste no pedido. Na secção 5.4.2.2 podemos ver mais detalhes dessa funcionalidade.

Um dos objetivos da Arrived é alcançar o máximo de pessoas a nível mundial, por isso, este formulário aceita números de telefone de todos os países. E de modo a facilitar a validação desses números, e melhorar experiência do utilizador, optou-se por utilizar a *framework* PhoneNumberKit, apresentada com mais detalhes no subcapítulo 4.3. Tal como pudemos observar na Figura 45, a experiência do utilizador é melhorada no sentido de inicialmente ter o código da região pré-preenchido automaticamente, a bandeira do país e a

formatação desse contacto sendo feita conforme ele insere o número, indicando imediatamente se se trata de um número válido ou não.

Numa primeira abordagem, tanto a validação do contacto, como a visualização da bandeira de cada país ao lado do campo, começaram a ser implementadas através de um *Enum*. Este *Enum*, continha o código do país e a quantidade de dígitos aceitável para cada um, e existia a imagem com a bandeira de cada país no ficheiro de *Assets* da aplicação. Cada imagem tinha o mesmo nome do código do respetivo país, listado no *Enum,* para que quando o utilizador inserisse o código do país a bandeira fosse mostrada na vista.

Ao notar o aumento da complexidade dessa validação e as possíveis falhas que poderiam ocorrer, uma pesquisa foi feita para descobrir uma solução que resolvesse esse problema e como pudemos ver no subcapítulo 4.3, a solução encontrada foi a *framework* PhoneNumberKit. Por fim, o número de telemóvel obtido é persistido nas UserDefaults [91] do dispositivo para depois ser possível aceder sempre que for necessário.

Acerca das localizações principais do utilizador, não há obrigatoriedade de preenchimento para nenhuma delas, e caso o utilizador não queira preenchê-las inicialmente, tem a opção de saltar essa parte ao selecionar o botão de confirmação da vista principal de configuração (ver Figura 46). Na parte inferior desta vista, também é possível visualizar a seguinte dica dada ao utilizador acerca disso: “Adiciona as localizações onde vais mais frequentemente. Se não o quiseres fazer agora, podes sempre fazê-lo mais tarde através do ícone do menu lateral”.

Caso o utilizador opte por criar uma localização, tal como podemos visualizar na Figura 47, a vista de criação é apresentada já com um marcador no mapa com o logotipo da Arrived, representando a localização atual do dispositivo. E o utilizador poderá definir esta localização de duas formas diferentes:

1. Através da barra de pesquisa - O utilizador poderá efetuar uma busca pela localização, tal como foi detalhado no subcapítulo 4.4. A *framework* MapKit usada nesta implementação permite a sugestão de localizações e pontos de interesse que o utilizador poderá escolher com base na sua pesquisa. Ao selecionar uma das opções sugeridas, como no ponto anterior, a vista do mapa e o marcador são redirecionados para a localização que representa a opção escolhida.
2. Através da seleção da localização na vista do mapa – O utilizador poderá optar por selecionar a localização desejada diretamente no mapa e para alcançar essa

localização poderá utilizar os recursos que a própria vista proporciona, como o *zoom in*, *zoom out* e selecionar e arrastar para conseguir visualizar a localização desejada. Ao selecionar a localização desejada, tal como nos pontos anteriores o marcador estará presente para marcar o local escolhido.

Ao guardar a localização escolhida, tal como podemos observar na Figura 46 e Figura 47, é exibido um *dialog* de confirmação com a descrição desta localização. Caso seja uma localização diferente das localizações *default* (casa, trabalho ou escola) o utilizador poderá inserir um nome para esta nova localização (ver Figura 46). Para confirmar a nova localização, o utilizador apenas tem de selecionar a opção "sim" ou caso queira cancelar, a opção "não". Caso o utilizador confirme, essa localização é persistida no dispositivo através do Core Data (ver subcapítulo 4.2) como uma das localizações favoritas do utilizador. Após a inserção da localização, esta passa da cor cinza para branca para indicar que foi definida, tal como podemos ver na Figura 46.

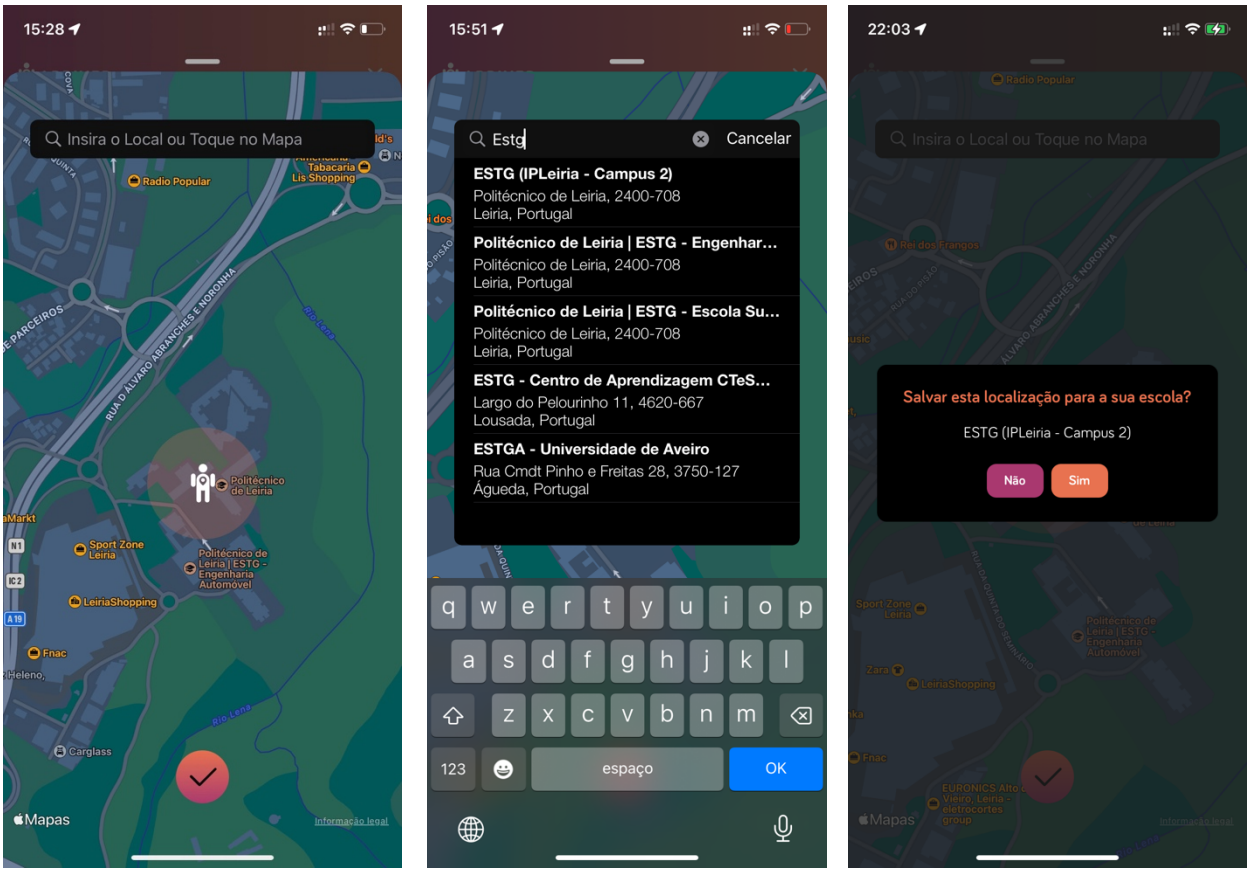

Figura 47 - Definição da localização da escola do utilizador

Para cada localização persistida as seguintes informações são obtidas:

- *type* – poderá ser *home, work, school* ou *other*;

- coordenadas geográficas – (*latitude* e *longitude*);
- *name* – no caso das localizações *default* é o mesmo que o tipo (casa, trabalho ou escola) e no caso de outra localização é o nome dado pelo utilizador;
- *address* – poderá ser uma morada ou algum ponto de interesse que identifique a localização;
- *favorite* – indicação se a localização é favorita ou não. As localizações definidas nas configurações iniciais são todas declaradas como favoritas, mas na criação de um aviso, que irá ser descrito no subcapítulo 5.4.2, poderá não ser.

A ação do utilizador confirmar a inserção das suas localizações principais ou a escolha de saltar essa parte, faz com que a indicação em como completou as suas configurações iniciais seja guardada através de um valor booleano nas *UserDefaults* do dispositivo. Permitindo assim, que na próxima vez que o utilizador abra a aplicação essa informação não seja solicitada novamente.

Por fim, o utilizador é reencaminhado para a vista de *welcome* da aplicação. Tal como podemos observar na Figura 48, nessa vista tem duas opções disponíveis, poderá selecionar a opção para criar o seu primeiro aviso, funcionalidade que veremos com mais detalhes no subcapítulo 5.4.2, ou poderá convidar um amigo para instalar a aplicação, funcionalidade que irá ser explicada na secção 5.4.7.2. De modo a contribuir para experiência de utilização, e semelhante ao apresentado na vista da Figura 46, são apresentadas ao utilizador, algumas dicas de utilização da aplicação, tais como:

- Dica 1 – "Para eliminares um aviso ou lembrete, arrasta-o para a esquerda e pressiona o ícone do caixote de lixo";
- Dica 2 – "Para fazer com que um aviso seja ativado sempre, liga o interruptor";
- Dica 3 – "Convida as pessoas que te são próximas a instalar também o Arrived, para que possam comunicar mais facilmente. Podes sempre fazê-lo mais tarde, no ícone do menu lateral".

A vista de *welcome* é apresentada sempre que o utilizador entre na aplicação e não tenha nenhum aviso criado, caso contrário visualizará a vista principal da aplicação, que consiste na sua listagem de avisos. No subcapítulo 5.4.6 é possível ver mais detalhes sobre esta vista.

Figura 48 - Vista de welcome da aplicação Arrived

5.4.2. Criação de avisos

Esta funcionalidade é uma das mais importantes da aplicação Arrived, porque contempla o seu objetivo principal, a criação de avisos com base em localização. Tal como foi explicado no subcapítulo anterior, essa opção é apresentada ao utilizador logo após a configuração inicial dos seus dados e está sempre presente, tanto na vista de *welcome* como na vista principal da Arrived. Neste subcapítulo serão apresentados os três tipos de avisos que a aplicação disponibiliza:

1. aviso, que dá a possibilidade de o utilizador avisar uma pessoa conhecida que chegou ou saiu de uma determinada localização;
2. pedido de aviso, que dá a possibilidade de o utilizador enviar um pedido de aviso a outro utilizador;
3. lembrete, que dá a possibilidade de o utilizador criar lembretes para si com base em localização.

5.4.2.1. Criação de um aviso

Para a criação de um aviso, tal como podemos observar na Figura 49, o utilizador precisa inserir alguns dados, nomeadamente:

- A frequência que esse aviso será enviado, se é sempre ou apenas uma vez;
- A indicação de quando a notificação para o envio do aviso deve ser enviada, se é quando o utilizador se aproximar ou afastar da região que será indicada;
- A seleção de um ou mais contactos da sua agenda que receberão a mensagem deste aviso;
- A localização e o raio, para definir a zona geográfica onde esse aviso irá disparar.

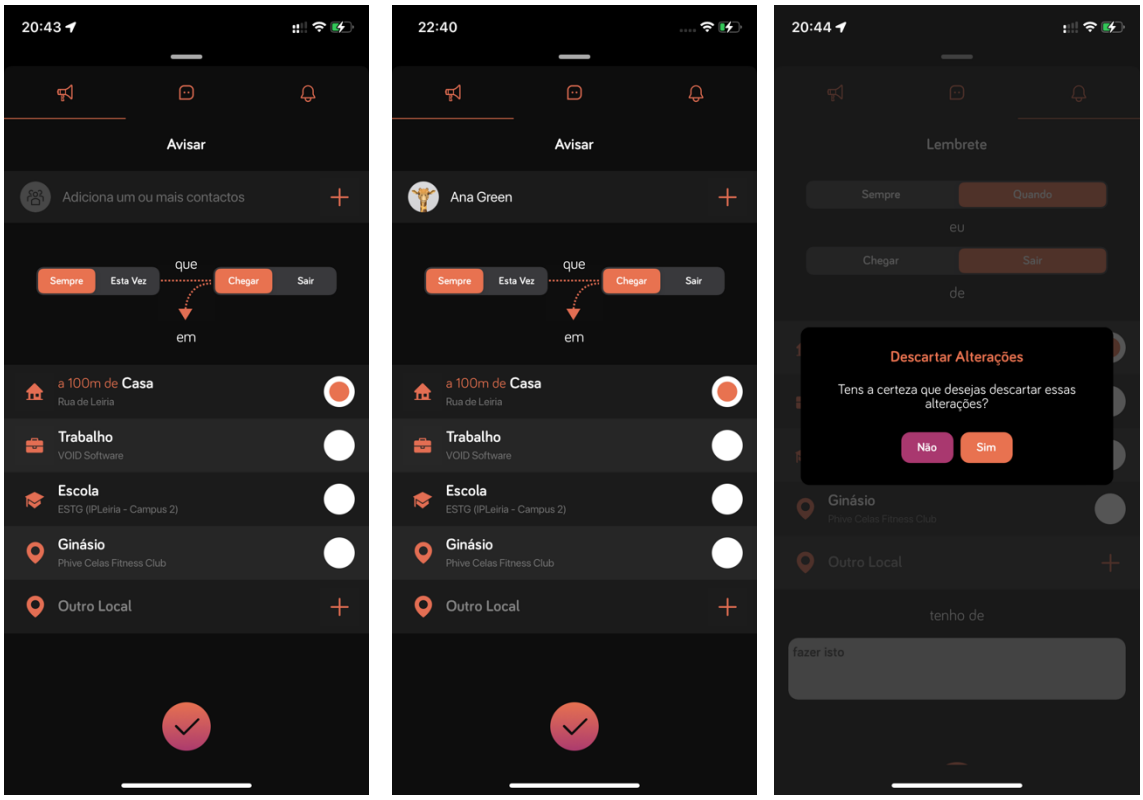

Figura 49 - Criação de um aviso na aplicação Arrived

Para obter os contactos que receberão este aviso, tal como foi demonstrado no subcapítulo 4.8, foi utilizada a *framework* Contacts. Na primeira imagem da Figura 50, podemos visualizar um exemplo após o utilizador escolher três contactos para o aviso. Após essa seleção, caso o utilizador queira remover alguns dos contactos escolhidos, apenas tem de os selecionar, para conseguir visualizá-los (ver segunda imagem da Figura 50), podendo assim remover individualmente cada contacto, usando a ação de *swipe* para a esquerda, ou selecionando o botão do caixote do lixo. Caso pretenda remover todos de uma vez, basta

efetuar a ação de *swipe* por cima da listagem, como apresentado na terceira imagem da Figura 50.

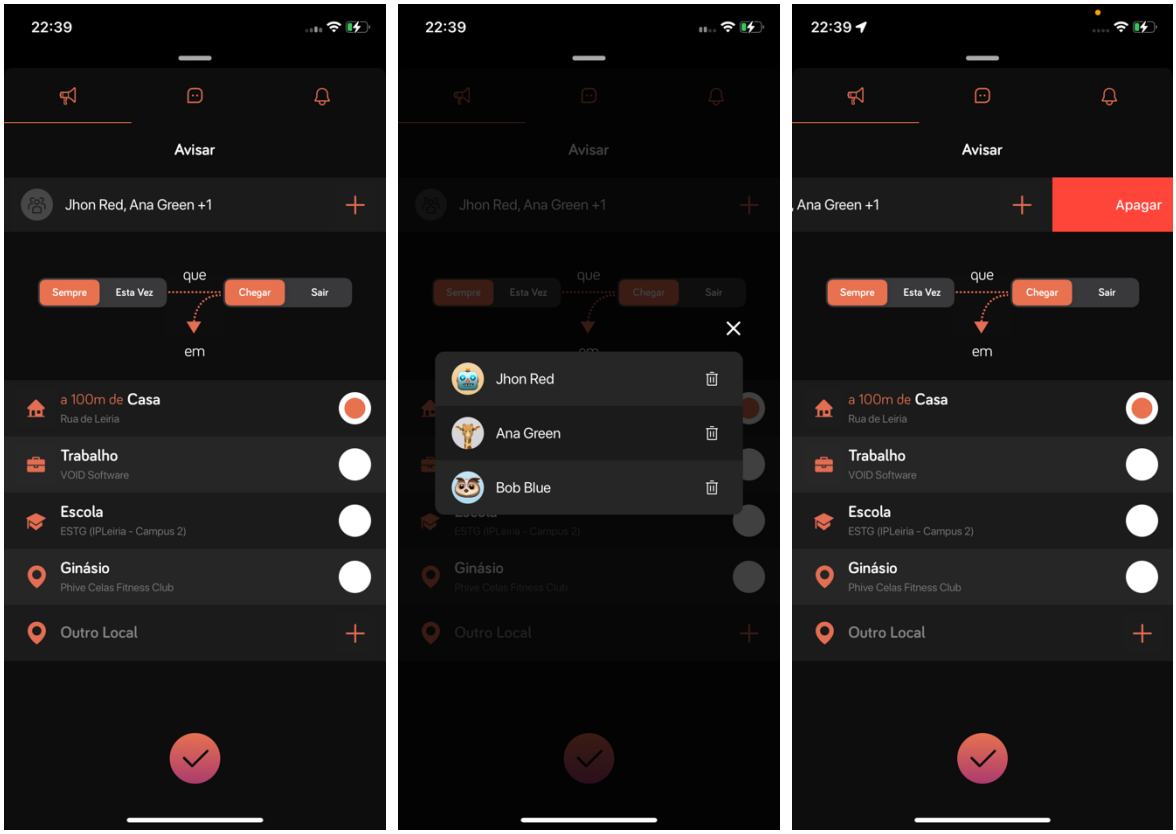

Figura 50 - Inserção de contactos na criação de um aviso na aplicação Arrived

Relativamente à localização do aviso, como podemos ver na Figura 49 e na Figura 51, o utilizador poderá indicá-la de duas formas diferentes: através da seleção de uma das suas localizações favoritas (casa, trabalho, escola ou outra); através da opção "Outro Local", que lhe permite a escolha da localização tanto pela seleção na vista do mapa; como através da barra de pesquisa, tal como foi descrito nas configurações iniciais do utilizador no subcapítulo 5.4.1.

Relativamente ao raio da zona geográfica, tal como podemos ver na primeira imagem da Figura 51, terá um valor inicial de 100 metros, mas o utilizador tem a possibilidade de escolher outro valor entre 1 e 1000 metros através do *slider*. Mesmo que o utilizador tenha escolhido uma das localizações favoritas para o aviso, poderá editar o raio desta localização através da vista do mapa. Com a diferença que neste caso a seleção do mapa estará desativada, e a barra de pesquisa escondida (ver segunda imagem da Figura 51). Isso acontece para evitar alterações nas localizações favoritas feitas por engano. A opção de

edição das localizações estará disponível no menu lateral e será apresentada na secção 5.4.7.1.

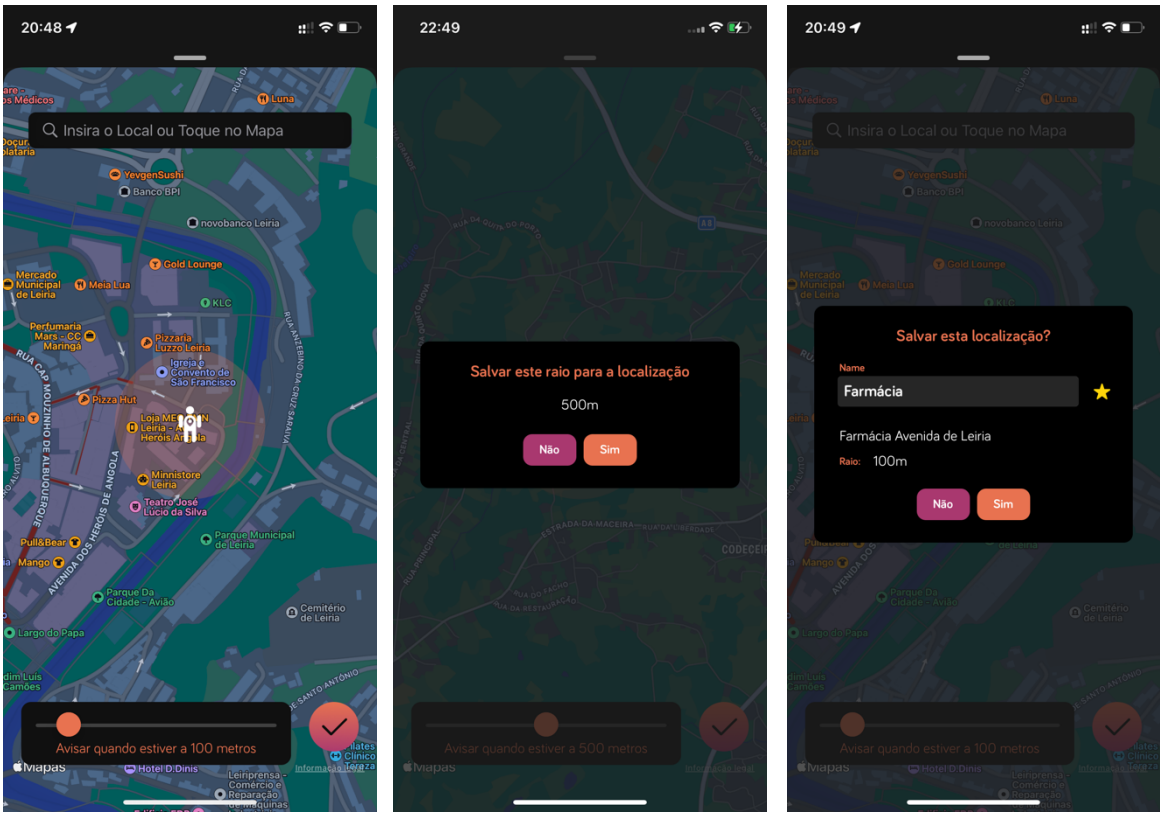

Figura 51 - Configuração da localização e do raio de um aviso na aplicação Arrived

Após o utilizador selecionar o botão para guardar a localização e o raio, é apresentado um *dialog* de confirmação com essa informação. Como podemos observar na Figura 51, esse *dialog* tem variações. Caso o utilizador esteja apenas a editar o raio, por se tratar de uma localização favorita, a confirmação é apenas do raio. Caso se trate de uma nova localização, poderá não só indicar o nome desta localização como também a informação sobre se pretende guardá-la nos favoritos através do ícone da estrela. Ao confirmar esta localização e/ou raio, o utilizador é redirecionado para a vista da criação do aviso com essa informação preenchida.

O utilizador poderá, então, confirmar a criação do aviso ou cancelar ao fechar a sua vista. Caso feche a vista é apresentado um *dialog* de confirmação (Figura 49), caso confirme, o aviso é criado já com a mensagem que será enviada ao destinatário do aviso. Esta mensagem é produzida através dos dados obtidos do preenchimento dos campos anteriores, mas poderá conter algumas variações, conforme as opções selecionadas do aviso. Por exemplo, caso o utilizador tenha escolhido as seguintes opções: avisar ao seu contacto, Ana, sempre que chegar, à Rua de Leiria, a mensagem será: “Eu cheguei à Rua de Leiria”.

Por fim, a notificação local deste aviso também é criada com base nos dados recolhidos, para que, posteriormente, o utilizador seja notificado quando se aproximar ou se afastar da localização definida. Além da criação da notificação local, ao confirmar a criação do aviso, os dados são persistidos no dispositivo através do Core Data para que posteriormente o utilizador possa gerir os seus avisos. O utilizador então é redirecionado para a vista principal da Arrived, onde é possível visualizar o aviso criado na sua listagem de avisos.

5.4.2.2. Criação de um lembrete com base em localização

A criação de um lembrete com base em localização é bastante semelhante à criação de um aviso. Como podemos observar na Figura 52, para a criação do lembrete, o utilizador precisa indicar os seguintes dados:

- A frequência com que a notificação desse lembrete será apresentada, se é sempre ou apenas uma vez;
- A indicação de quando a notificação para este lembrete deve ser apresentada, se é quando o utilizador se aproximar ou se afastar da região que será indicada;
- A localização e o raio, para definir a zona geográfica onde a notificação deste lembrete irá disparar;
- A mensagem que a notificação do lembrete irá exibir ao utilizador.

As diferenças entre o lembrete e o aviso são:

- Ao contrário do aviso, o lembrete não possui contactos, porque o seu propósito é a criação de notificação para o próprio utilizador;
- Ao contrário do aviso, não existe uma ação para a notificação do lembrete, porque o intuito desta notificação é apenas lembrar o utilizador de uma tarefa com base na sua localização. A seleção desta notificação apenas abre a aplicação normalmente;
- Na criação do lembrete o utilizador deve indicar a mensagem que deseja visualizar quando receber a notificação. Diferente do aviso, onde a mensagem que o utilizador recebe na notificação é para lembrá-lo de avisar outro(s) utilizador(es) que chegou ou saiu de algum lugar.

Um exemplo de um caso de uso de um lembrete com base em localização, é o utilizador ser notificado para não se esquecer de comprar a sua medicação quando se aproximar a uma certa distância da localização de uma farmácia.

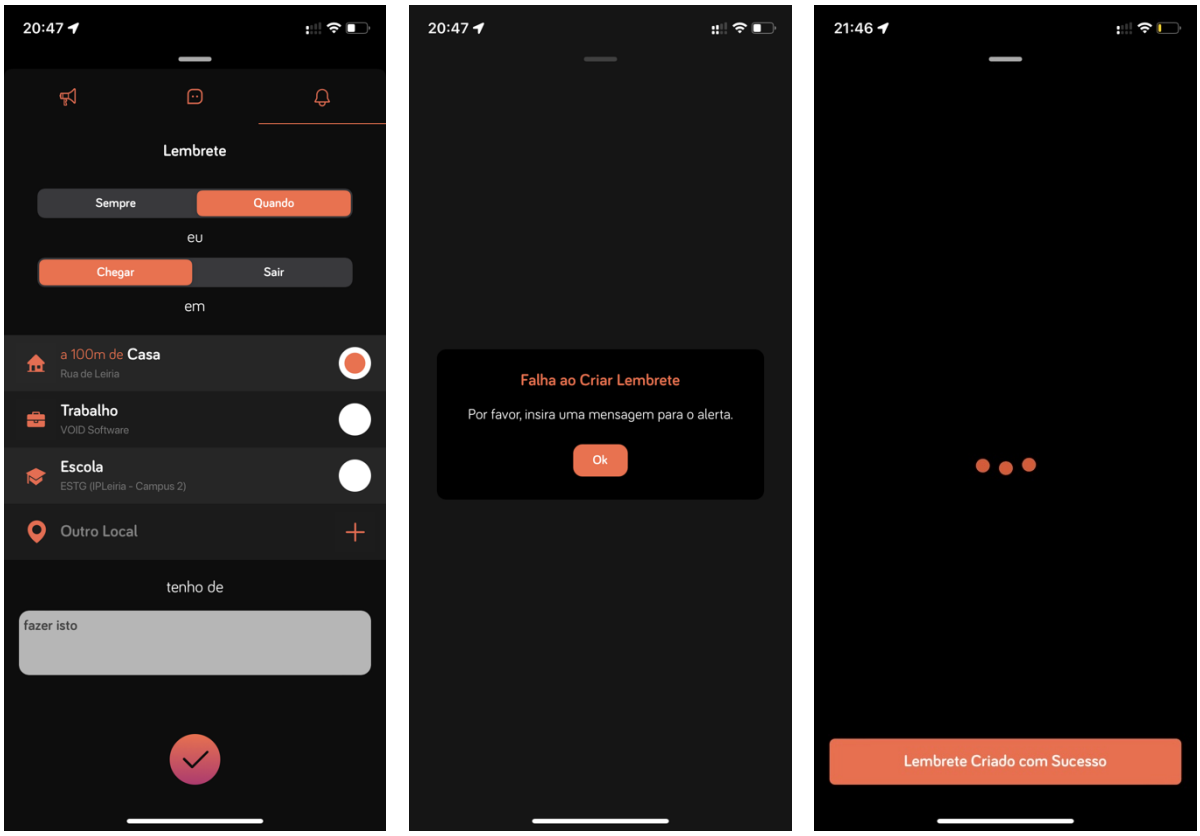

Figura 52 - Criação de um aviso do tipo lembrete na aplicação Arrived

5.4.2.3. Criação de um pedido de aviso

A criação de um pedido de aviso, tem um processo semelhante à criação de um aviso, o que difere os dois, é que o pedido de aviso é para ser criado no dispositivo do remetente do pedido, para que ele possa avisar ao destinatário que chegou ou saiu de uma determinada localização. Tal como podemos observar na Figura 53, os dados necessários para a sua criação são:

- A frequência com que esse aviso será enviado, se é sempre ou apenas uma vez;
- A indicação de quando é que a notificação para o aviso deve ser enviada, se é quando o remetente se aproximar ou se afastar da região que será indicada;
- A seleção de um ou mais contactos da sua agenda que receberão este pedido de aviso;
- A localização e o raio, para definir a zona geográfica onde esse aviso irá disparar;
- Uma mensagem, para acompanhar este pedido de aviso, que o destinatário irá receber.

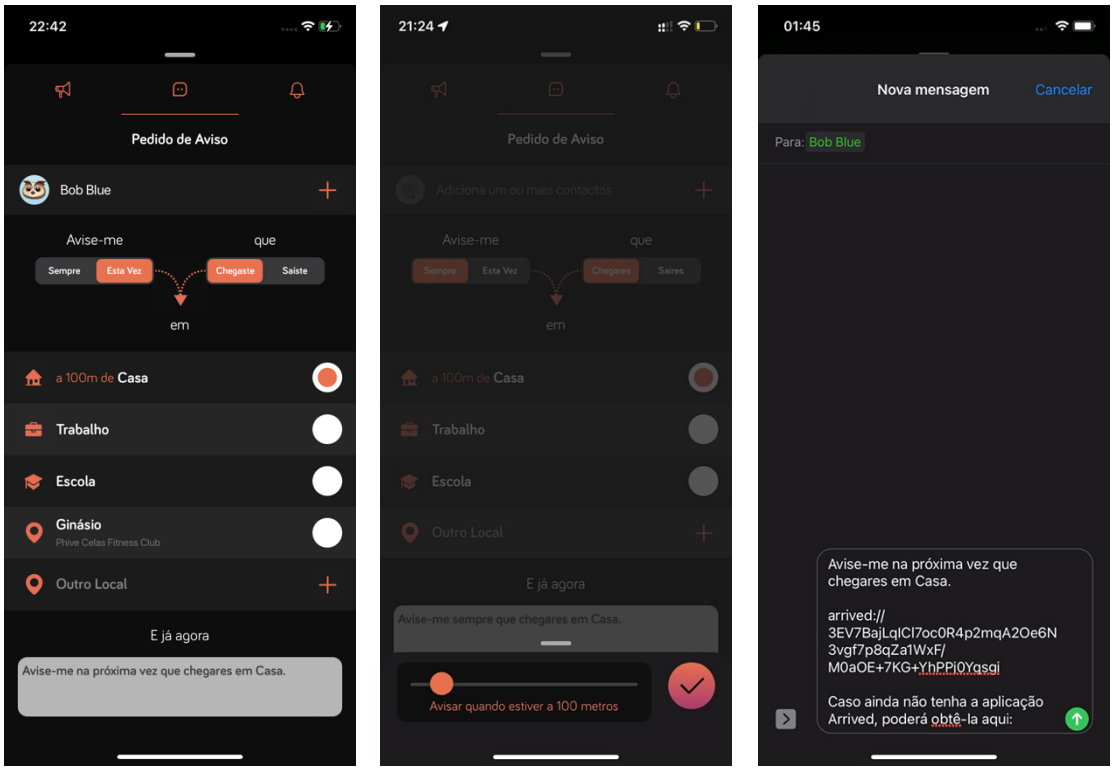

Figura 53 - Criação de um pedido de aviso na aplicação Arrived

A maioria dos dados são recolhidos da mesma forma que são recolhidos na criação de um aviso. As diferenças estão:

- No campo da mensagem, que apesar de ser um parâmetro presente em todos os tipos de aviso, apenas no lembrete e no pedido de aviso, o utilizador poderá editá-la. Como podemos ver na Figura 53, esta mensagem é inicialmente criada com base nos restantes dados indicados, dando a possibilidade de o utilizador acrescentar ou retirar alguma informação se assim o entender. A mensagem terá componentes de grande importância na criação desse pedido, como veremos mais a frente neste subcapítulo;
- No campo do(s) contacto(s), que neste caso se trata de quem irá receber um pedido para a criação de um aviso;
- No campo da localização, diferente dos tipos de avisos anteriores, no caso de o utilizador escolher uma das opções de localização favoritas dos tipos: casa, trabalho ou escola. Por exemplo, na localização de casa, ele estará a referir-se à localização da casa de quem irá receber o pedido de aviso, e não à da sua própria casa. Na Figura 53, é possível observar as seguintes diferenças relativas a isso:
    - diferente dos avisos anteriores, na criação de um pedido de aviso a descrição desse tipo de localização não aparece por baixo do seu nome, mesmo que o

utilizador já tenha definido a localização da sua casa nas configurações iniciais;

- diferente dos avisos anteriores, apesar de não conseguir editar as localizações através dessa vista, quando o utilizador seleciona uma localização desse tipo, consegue visualizá-las no mapa. Neste caso, como podemos ver na Figura 53, ao selecionar uma localização desse tipo, o utilizador apenas tem a opção de alterar o raio desta localização, porque a mesma só irá ser definida no dispositivo de quem receber este pedido.

Pôr fim, ao concluir a criação do pedido de aviso, esse pedido é persistido no dispositivo e a *interface* de envio de mensagens é apresentada ao utilizador já com o(s) destinatário(s) e o corpo da mensagem do pedido preenchido, como podemos ver na Figura 53. Caso o utilizador crie um pedido de aviso com as seguintes opções: avisar sempre, quando chegar, à Rua de Leiria ao seu contacto, Ana, a mensagem *default* será "Avisa-me sempre que chegares à Rua de Leiria".

Como foi dito anteriormente, essa mensagem tem um propósito bastante importante, além de informar ao destinatário sobre o pedido de aviso, também contém dois URLs, como podemos ver na Figura 53. O primeiro URL contém toda a informação necessária para a criação do aviso, mas só funciona se o destinatário do pedido tiver a aplicação instalada no seu dispositivo. Já o segundo é justamente para o caso do utilizador que receber esse pedido e ainda não tenha instalado a aplicação, ao selecionar esse segundo URL ele será redirecionado para a página da Arrived na Apple Store e poderá efetuar a sua instalação, para posteriormente efetuar a criação de um aviso.

O primeiro URL foi definido através de um URL *scheme* [92], que fornece uma forma de referenciar recursos de uma aplicação. Para a proteção do conteúdo deste URL, antes da sua adição ao corpo da mensagem, o seu conteúdo é encriptado (obter mais detalhes no subcapítulo 5.4.5). Quando o utilizador seleciona o URL personalizado dentro do SMS, após o seu conteúdo ser desencriptado, a aplicação é iniciada num contexto especificado. Para melhor explicar a informação que o primeiro URL contém, "arrived://351912345678/113/39.71623215497757/-8.85682708045416/100", vamos continuar com o exemplo, usado para a criação de um pedido de aviso, e vamos dividir o seu conteúdo pelos seus componentes de *path*. Esses componentes estão separados pelo caractere "/" e o objetivo de cada um é:

- arrived - Nome do *scheme* personalizado do URL, definido no ficheiro de configuração **Info.plist** da aplicação Arrived, basicamente é o que indica ao sistema operativo qual aplicação abrir, quando o utilizador seleciona o URL;
- 351912345678 – Base do URL e onde começa o conteúdo com a informação contida no URL referente ao contexto da Arrived. Essa informação é o número de telemóvel do utilizador que envia o pedido de aviso;
- 113 – Este componente contém três informações diferentes do pedido de aviso a ser criado, sendo que cada número representa uma informação:
    - O primeiro é a indicação se o aviso é para enviar sempre ou apenas uma vez, sendo que o 1 representa "sempre" e 0 representa "apenas uma vez";
    - O segundo é a indicação se o aviso é para enviar quando o utilizador chegar ou sair da localização, sendo que 1 representa "chegar" e 0 representa "sair";
    - O terceiro representa o tipo de localização do aviso, sendo 0 a representar a localização de casa, 1 do trabalho, 2 da escola e 3 para outra localização;
- 39.71623215497757 e -8.85682708045416 – Esses componentes representam a latitude e longitude da localização do aviso. Esses componentes só são inseridos no URL caso a localização seja do tipo 3, porque caso não seja desse tipo, a latitude e longitude que serão usadas para criar o aviso serão obtidas através das localizações principais do utilizador que receberá esse pedido de aviso.
- 100 – Este componente representa o raio definido no pedido de aviso;

Após o utilizador confirmar o envio da mensagem com o pedido de aviso, os dados desse pedido são persistidos localmente na aplicação através do Core Data e o(s) destinatário(s), recebe(m) o SMS com o pedido. Por fim, uma mensagem de sucesso ou erro (caso algo corra mal), é apresentada ao utilizador, redirecionando-o para a vista da listagem de avisos e, tal como nos avisos anteriores, poderá visualizar o pedido de aviso criado com os restantes. No subcapítulo 5.4.4, será descrito o que acontece quando o utilizador seleciona o primeiro URL do SMS enviado com o pedido de aviso.

5.4.3. Notificação de um aviso ou lembrete

Ao se tratar de uma notificação de um aviso, quando o utilizador é notificado, e seleciona a notificação ou a sua opção "Alertar", é reencaminhado para a aplicação Arrived, que por sua vez apresentará a *interface* da aplicação nativa de mensagens do dispositivo, já com o(s) destinatário(s) do aviso e o corpo da mensagem preenchidos. O utilizador terá apenas de

selecionar o botão de enviar. É importante ressaltar que antes do envio, o utilizador poderá não só editar o conteúdo, como também cancelar o seu envio.

Caso o serviço de iMessage [93] do iPhone quer do remetente como do(s) destinatário(s) esteja ativo, o envio é feito por este meio, caso contrário o envio é feito com o recurso a SMS. Ao se tratar de um lembrete com base em localização, tal como foi explicado anteriormente, o utilizador será lembrado da sua tarefa através da mensagem da notificação recebida e ao selecionar esta notificação será reencaminhado normalmente para a aplicação.

Embora existam APIs que procedam ao envio de SMS automaticamente, como o SMS Messaging API do Twilio [94] ou a Messages API do Vonage [95], é importante dizer que a implementação de APIs desse género iam contra alguns dos requisitos iniciais da Arrived, nomeadamente: a implementação de uma aplicação sem recursos de APIs externas, ser o mais *offline* possível e que permita priorizar e tirar partido da utilização de *frameworks* nativas iOS. Considerando que o envio de um SMS é uma ação extremamente sensível, e pelo facto da Apple não permitir o seu envio automaticamente através da *framework* MessageUI, o seu envio é feito sempre com a confirmação do utilizador.

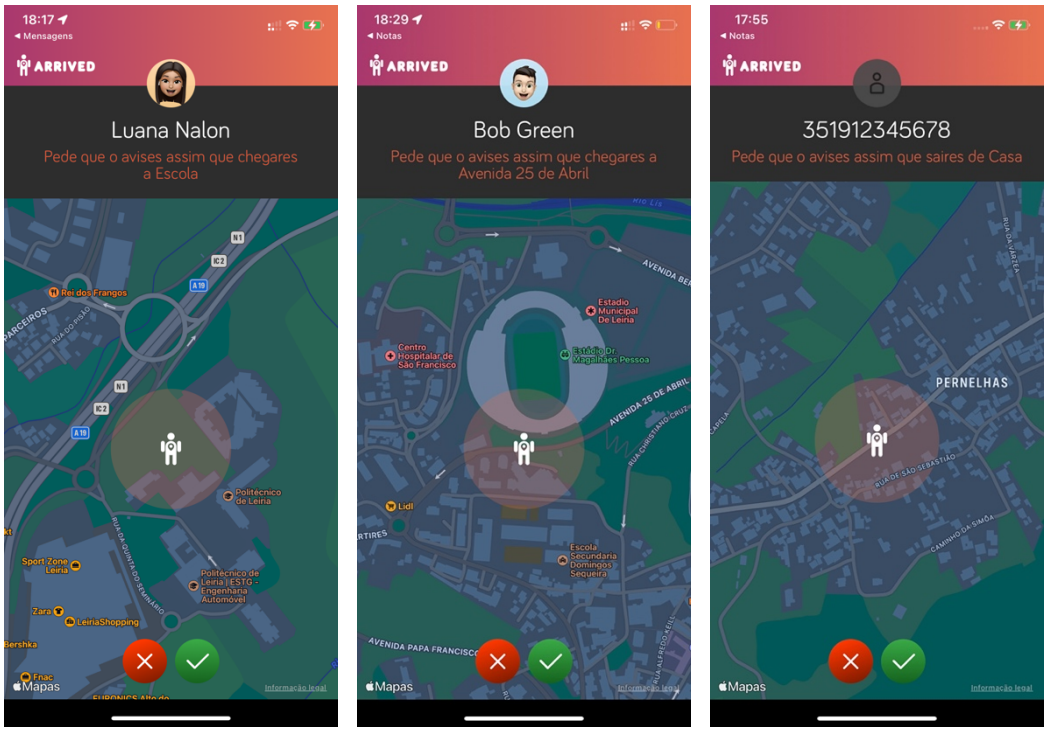

Figura 54 - Resposta a um pedido de aviso na aplicação Arrived

5.4.4. Resposta de um pedido de aviso

Essa funcionalidade é utilizada sempre que o utilizador recebe um pedido de aviso de outro utilizador da Arrived. Ao selecionar o primeiro URL do pedido de aviso recebido através do SMS, após o conteúdo desse URL ser desencriptado, o utilizador é reencaminhado para a aplicação, e através do conteúdo do URL é obtida a informação necessária para a criação do aviso. Como podemos ver na Figura 54, a primeira vista que o utilizador visualiza é a de confirmação, onde terá de indicar se aceita ou não esse pedido de criação de aviso.

Para a criação dessa vista foi necessário obter a informação do contacto que constava no URL, e como explicado no subcapítulo 5.4.3, essa informação é apenas o número de telemóvel do utilizador que enviou o pedido. Mas como podemos ver na Figura 54, existem mais detalhes a serem apresentados na vista, como a foto e o nome do contacto. Foi possível obter essa informação com o auxílio da *framework* Contacts através de uma pesquisa efetuada entre os contactos do utilizador. No subcapítulo 4.8 é possível obter mais detalhes dessa implementação.

Caso o utilizador tenha este contacto guardado na sua agenda de contactos do dispositivo, é possível obter tanto a sua foto (caso exista) como o seu nome, porque visualmente essa forma é mais intuitiva para mostrar a identificação de quem pede a criação do aviso. Caso o utilizador não tenha este contacto no seu dispositivo, tal como podemos visualizar na terceira imagem da Figura 54, apenas o número de telemóvel será mostrado no pedido. Outra informação que podemos observar nesse pedido é a mensagem apresentada, essa mensagem é criada conforme as informações obtidas do URL.

No exemplo da primeira imagem da Figura 54, temos um pedido de aviso com as seguintes características: avisar quando chegar a 150 metros da escola ao seu contacto, Luana Nalon, e o URL selecionado seria: “arrived://351912345679/012/150”. Neste caso a mensagem, com o nome do contacto será “Luana Nalon pede que o avises sempre que chegares à Escola”. Podemos observar algumas diferenças entre este URL e o URL do exemplo da secção 5.4.2.3, além do número de telemóvel e o raio, temos o terceiro componente do URL “012” com o último número igual a dois, o que significa que a localização do aviso é do tipo predefinido “*school*” e por isso não temos os dois componentes que representam a latitude e longitude, porque esses dados serão obtidos através da localização da escola do utilizador.

É importante que todos os utilizadores que instalem a aplicação e tenham interesse em usufruir das suas funcionalidades, procedam à configuração inicial dos seus dados, nomeadamente, número de telemóvel e localizações das suas moradas principais. Pois, só assim poderão tirar partido por completo de todas as funcionalidades. Neste caso, se o utilizador que recebeu esse pedido de aviso, não tivesse essas configurações feitas, pelo menos o número de telemóvel e a localização da escola, não poderia responder a esse aviso até ter esses dados definidos.

Por exemplo, no caso do utilizador ter o número de telemóvel definido e não ter a localização da escola, um *dialog* ser-lhe-ia apresentado (ver Figura 55), com uma mensagem a informá-lo sobre os dados em falta e com a opção para reencaminhá-lo para a vista de configuração da localização (ver Figura 47). Após definir a localização em falta, é novamente reencaminhado para a vista da resposta do aviso, para poder aceitá-lo ou recusá-lo.

Figura 55 - Dialogs com mensagens relacionadas a resposta de um pedido de aviso na aplicação Arrived

Caso o utilizador tenha tudo bem definido, além dos dados do contacto e a respetiva mensagem, essa vista terá um marcador no mapa com o ícone da Arrived, por cima da localização da sua escola (ver Figura 54). Essa localização é obtida de formas diferentes, conforme o seu tipo. Caso essa localização seja uma localização favorita, como no exemplo dado, e o utilizador já tenha preenchido os dados dessa localização inicialmente (conforme detalhado no subcapítulo 5.4.1), esta será obtida através de uma pesquisa nos dados persistidos no dispositivo.

No entanto, se a localização for do tipo *other*, e o URL tenha a informação da latitude e longitude, é usado o método de *reverse geocoding* implementado com componentes da *framework* Core Location (ver subcapítulo 4.5). Através desse método, como podemos observar na terceira imagem da Figura 54, é possível obter o nome dessa localização, que

poderá ser uma rua, um largo, um ponto de interesse, entre outros. E com todos esses dados reunidos, a criação desse aviso será possível.

Caso o utilizador recuse esse pedido de aviso, ao confirmar a sua decisão através do *dialog* de confirmação (ver Figura 55), será reencaminhado para a vista inicial da aplicação. Caso aceite esse pedido, um novo aviso é criado (de acordo com o apresentado na secção 5.4.2.1), com os dados obtidos através do URL. Por fim, o utilizador é reencaminhado para a vista principal, já com o seu novo aviso entre os restantes.

5.4.5. Encrypt & Decrypt do URL do pedido de aviso

Como já foi referido na secção 5.4.2.3, quando o utilizador envia um pedido de aviso para outro utilizador, este pedido contém uma mensagem e dois URLs, o URL da aplicação Arrived na App Store e o URL local para a aplicação, que contém a informação para criação do aviso no dispositivo que recebeu este pedido de aviso.

De modo a fornecer segurança e proteção adicional para o conteúdo do URL local, foi utilizado o algoritmo criptográfico Advanced Encryption Standard (AES) [96] para encriptar o conteúdo do URL. Esta encriptação foi desenvolvida através da *framework* CryptoKit [97]. Para realizar a encriptação do conteúdo do URL local com AES, a classe **SymmetricKey** do CryptoKit foi utilizada para gerar uma chave simétrica de 256 bits, que está a ser utilizada tanto para a encriptação quanto para a desencriptação do conteúdo do URL.

Quando o utilizador cria um pedido de aviso para outro utilizador, o conteúdo do URL que vai na mensagem é encriptado através da classe **AES.GCM** do CryptoKit, gerando assim um valor *combined* [98], este valor contém os dados criptografados, um *nonce* e uma *tag* de autenticação que ajuda a garantir a integridade dos dados. Como podemos observar na Figura 56, para permitir que o valor *combined* do conteúdo do URL fosse enviado para outro utilizador, foi necessário converter este valor numa *string*. Para isso, foi utilizada a codificação Base64 para transformar o valor de *combined* numa *string* que poderia ser enviada via SMS.

Quando o outro utilizador recebe o URL com o valor *combined* via SMS, ao abrir a aplicação através desse URL, a mesma chave simétrica é usada para desencriptar o conteúdo do URL. Para isso, a classe **AES.GCM** do CryptoKit foi utilizada novamente para realizar a desencriptação do valor de *combined* e obter o valor original do conteúdo do URL. Com esta implementação adicional de segurança, foi possível garantir a confidencialidade e a

integridade dos dados transmitidos. E o utilizador poderá proceder à resposta ao pedido do aviso, como foi descrito no subcapítulo 5.4.4.

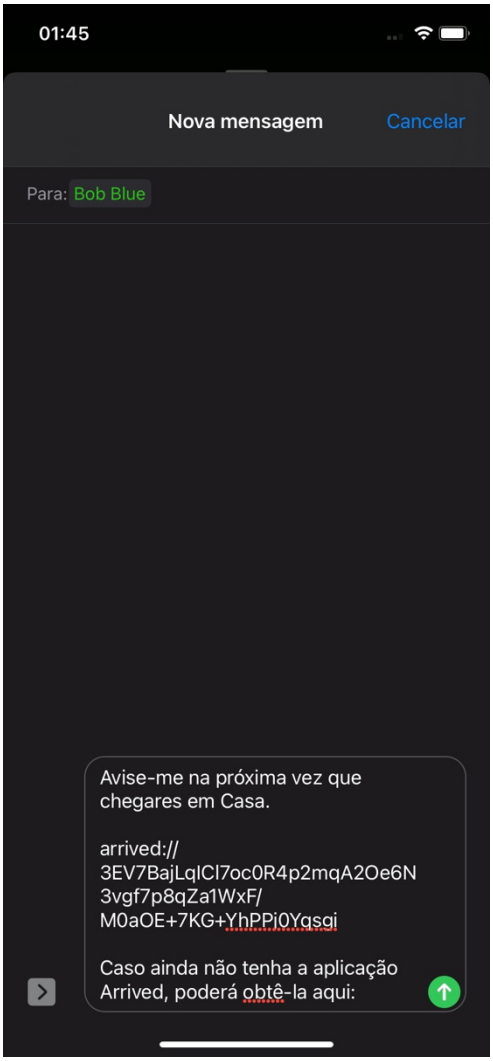

Figura 56 - URL de um pedido de aviso com os dados encriptados através do algoritmo AES

5.4.6. Listagem de avisos

Como já foi referido nos subcapítulos anteriores, a listagem de avisos representa a vista principal da aplicação, quando o utilizador já tem um ou mais avisos criados. Caso contrário, ele visualizará a vista de *welcome* (ver Figura 48). Como podemos verificar na Figura 57 e na Figura 58, na listagem de avisos o utilizador tem quatro opções de visualização: a listagem de todos os tipos de avisos, os avisos para outros, os pedidos de avisos e os lembretes.

Na Figura 57, é possível ver um exemplo onde existem avisos de todos os tipos criados. Conforme o tipo de aviso, é possível visualizar algumas opções de ações que o utilizador poderá executar em cada um. A primeira ação comum a todos os tipos de aviso é o apagar, ação que pode ser realizada ao fazer um gesto de *swipe* para o lado esquerdo, por cima do

aviso. Outras duas ações, comuns a todos os tipos, é a de visualização e edição. Para isso o utilizador precisa apenas selecionar o aviso pretendido, e, por questões de *design*, no caso dos lembretes o utilizador também pode selecioná-lo através do ícone de lápis.

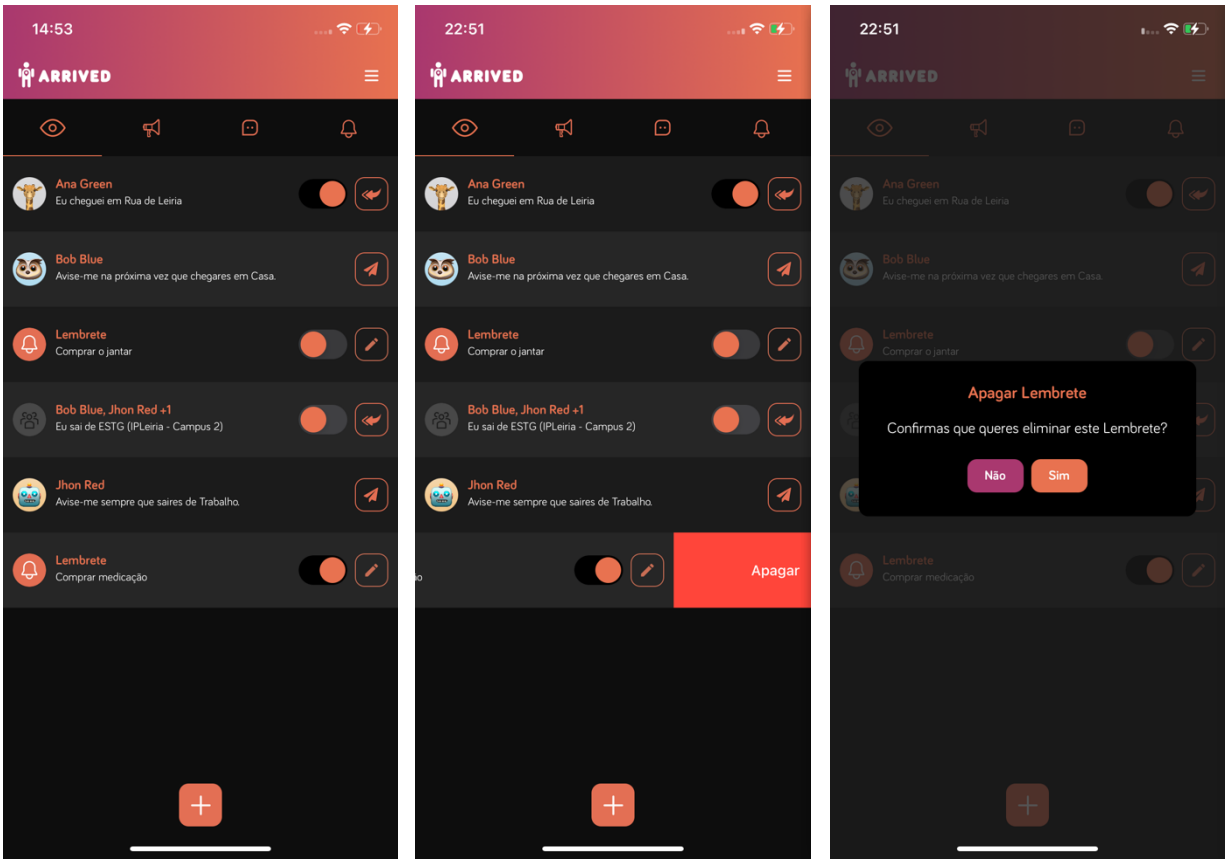

Figura 57 - Arrived Listagem e remoção de avisos na aplicação Arrived

A outra opção é o *switch* (ver Figura 58), onde o utilizador tem a possibilidade de fazer uma alteração nos seus avisos, ou nos seus lembretes. Caso o *switch* esteja ativo o aviso está configurado para avisar sempre e caso esteja desativo está configurado para avisar apenas uma vez. Ao alterar essa opção, o utilizador visualizará um *dialog* (ver Figura 59) e após a confirmação, além de ser feito um *update* a este aviso na base de dados, a notificação local existente é apagada e uma nova notificação é criada com a nova frequência de envio. Se a alteração for para o aviso ser enviado sempre, o utilizador passará a ser notificado sempre que se aproximar ou sair da região conforme esteja especificado no aviso. E caso o utilizador desative a opção de avisar sempre, a notificação de aviso será enviada apenas uma vez ou mais uma vez, caso esta já tenha sido enviada antes.

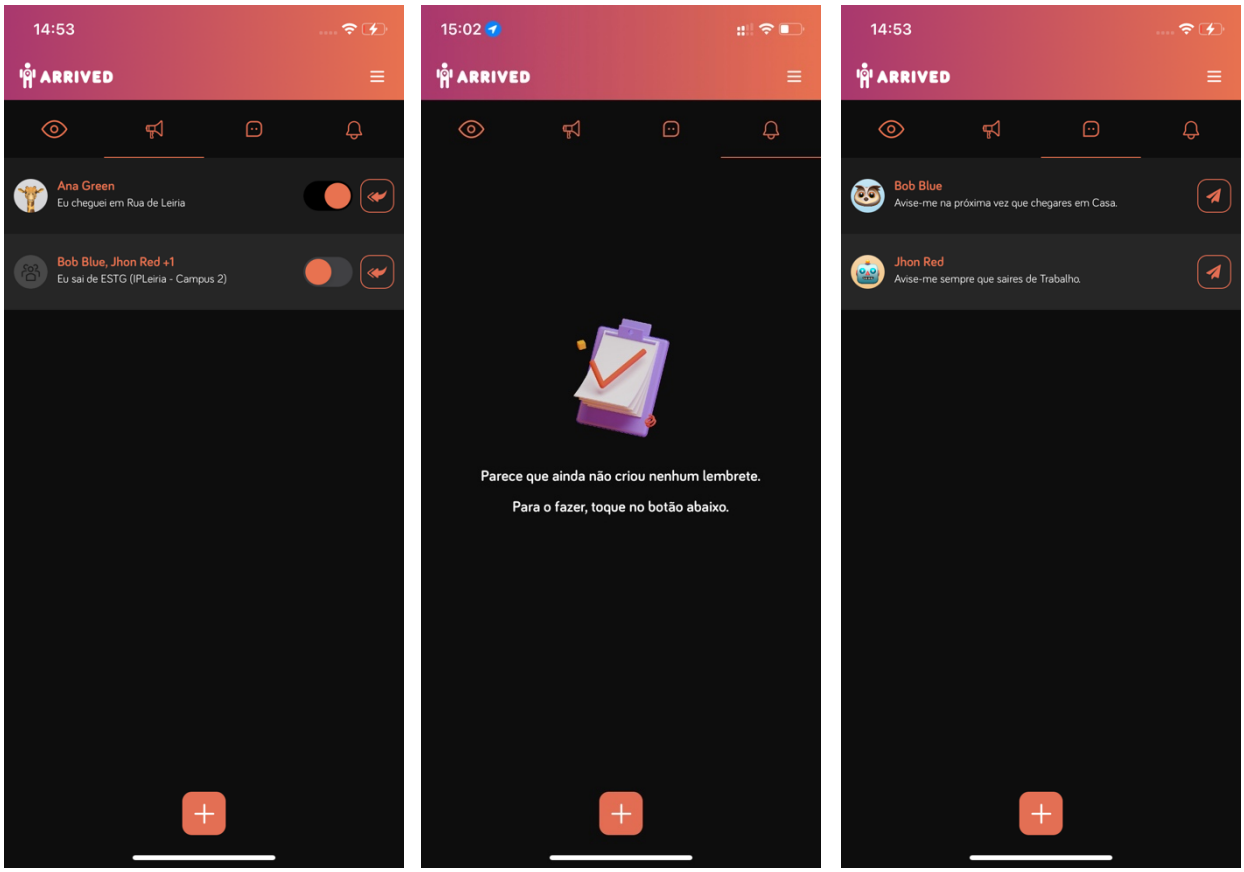

Figura 58 - Opções para listar tipos diferentes de avisos na aplicação

Outra opção, representada por um botão com o ícone de um avião de papel (ver Figura 58), está presente apenas nos pedidos de aviso. Esta opção oferece a possibilidade de o utilizador reenviar esse pedido ao(s) seu(s) destinatário(s). Ao selecioná-la, após a confirmação, o utilizador é reencaminhado para a *interface* do envio de SMS já com o(s) destinatário(s) e o corpo da mensagem preenchidos com os mesmos dados do pedido já enviado, e assim poderá repetir o seu envio.

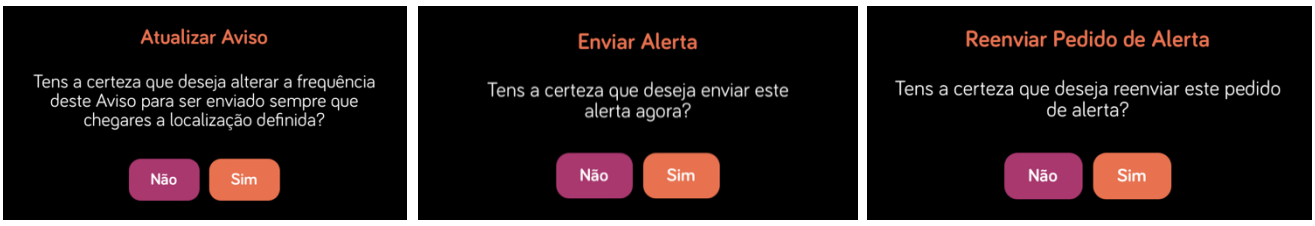

Figura 59 - Dialogs com mensagens relativas a ações de avisos na aplicação Arrived

Por fim, existe a opção com o ícone de duas setas (Figura 58). Esta opção está presente nos avisos para outras pessoas. Ao selecionar essa opção, após a confirmação, o utilizador é reencaminhado para a *interface* do envio de SMS, com todos os dados preenchidos e poderá

informar o(s) destinatário(s) do aviso, que chegou ou saiu de uma determinada região, conforme as características do aviso. Essa opção pode ser usada, caso o utilizador esteja com a aplicação aberta quando chega ou sai de uma determinada região e lembrar-se de avisar o outro utilizador, ou caso o utilizador se esqueça de avisá-lo através da notificação e queira avisar posteriormente.

5.4.7. Menu lateral

O menu lateral, localizado no canto superior direito da página inicial, é representado pelo ícone "hambúrguer" (ver Figura 60). Este menu é composto por quatro opções: a opção de visualizar, atualizar, apagar e adicionar as localizações favoritas do utilizador, a opção de enviar um convite para a instalação da aplicação Arrived, a opção de visualização da política de privacidade e a opção de visualização dos termos e condições. Neste subcapítulo iremos explicar com mais detalhes essas opções.

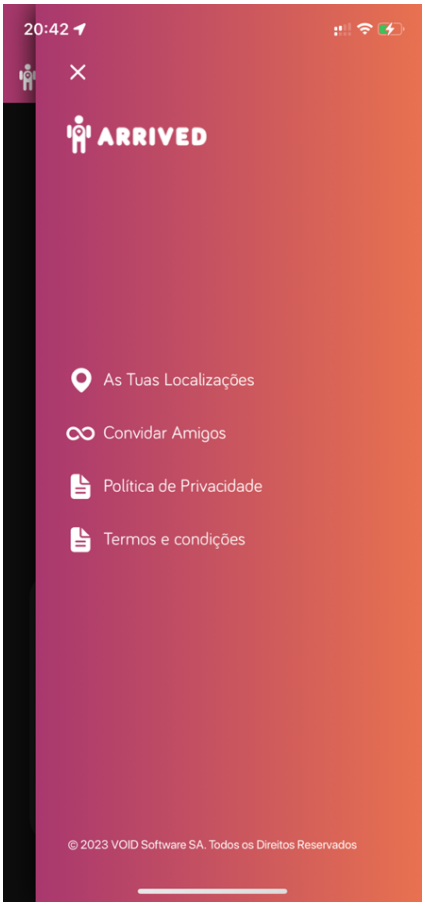

Figura 60 - Menu lateral da aplicação Arrived

5.4.7.1. Visualização, adição, remoção e edição das localizações favoritas do utilizador

Ao selecionar esta opção, como podemos ver na Figura 61, o utilizador é reencaminhado para uma vista semelhante à de recolha inicial dos dados do utilizador, e poderá visualizar, adicionar, apagar e editar as suas localizações favoritas. O funcionamento desta vista também é semelhante à de recolha inicial dos dados (descrito com mais detalhes no subcapítulo 5.4.1), a diferença, além da parte superior da vista, é que neste caso, quando o utilizador acede a esta opção, já poderá ter avisos criados.

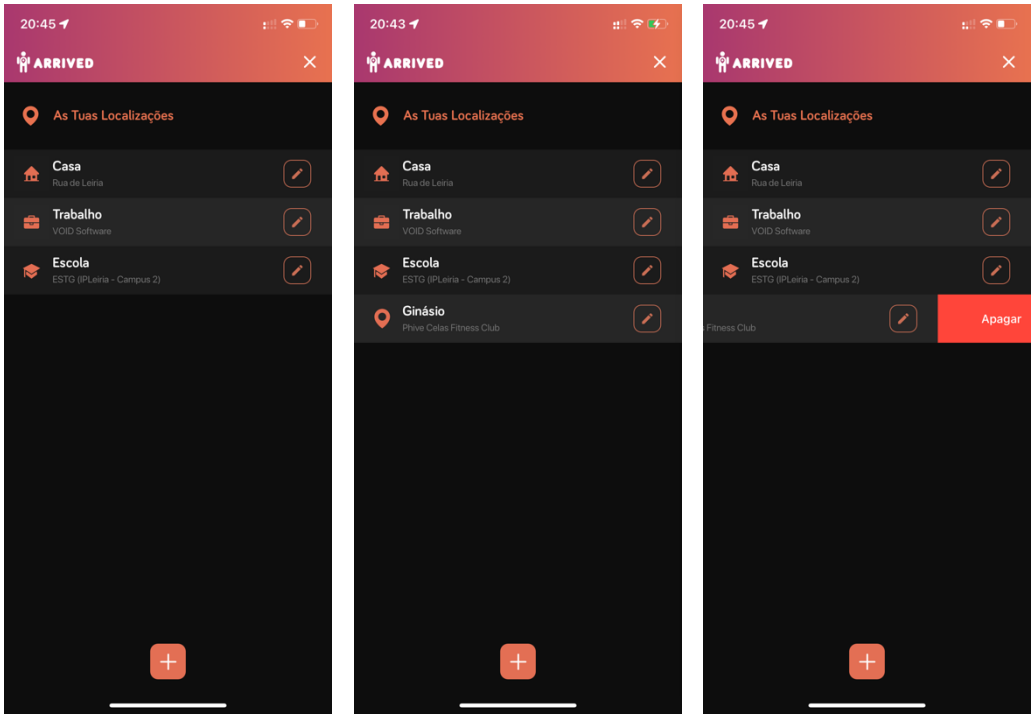

Figura 61 - Listagem de localizações favoritas do utilizador da aplicação Arrived

É importante citar que no caso de o utilizador já ter avisos criados anteriormente, ao executar operações do tipo apagar ou editar, no desenvolvimento desta funcionalidade, foi crucial ter em consideração os avisos criados que envolvem estas localizações. No caso da tentativa de apagar uma localização que esteja presente em um ou mais avisos, como podemos ver na Figura 62, o utilizador receberá uma mensagem de erro.

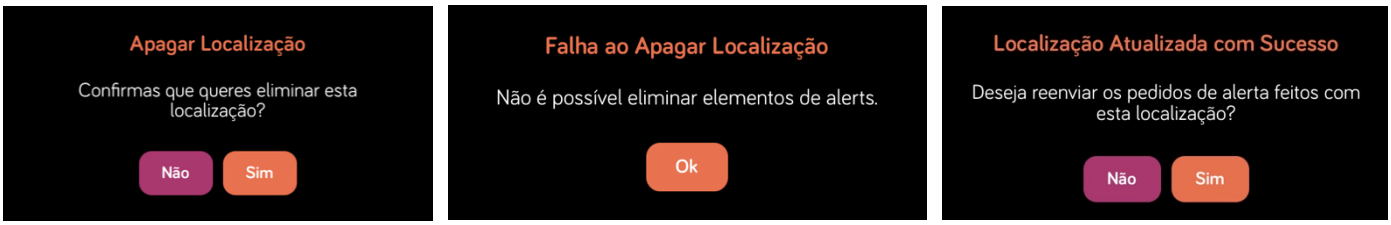

Figura 62 - Dialogs com mensagens relacionadas a operações em localizações favoritas do utilizador

No caso da edição de uma localização presente em um ou mais avisos para outros utilizadores ou lembretes, os mesmos são atualizados, com a nova localização. Ao se tratar de uma localização presente em um ou mais pedidos de avisos, através do *dialog* da Figura 62, é questionado ao utilizador se deseja reenviar esses pedidos de avisos, com a nova localização. Ao confirmar, o utilizador é redirecionado para a *interface* do envio de mensagens com o(s) contacto(s) e o conteúdo do pedido atualizado, e poderá proceder ao seu reenvio.

5.4.7.2. Convite de instalação da aplicação

Ao selecionar essa opção, como podemos visualizar na Figura 63, o utilizador visualizará a *interface* do controlador de partilha nativo da Apple [99], e poderá efetuar a partilha do *link* da aplicação Arrived na App Store através das suas aplicações de mensagem. Além do *link*, o convite contém uma mensagem de apresentação: "Olá! Já ouviste falar da Arrived? É uma aplicação baseada na localização que permite criar avisos para ti ou notificar alguém quando chegares ou saíres de uma localização específica. É super fácil de usar e muito útil para a tua rotina diária. Deves definitivamente experimentar!! Faça o *download* aqui: link_da_applicação".

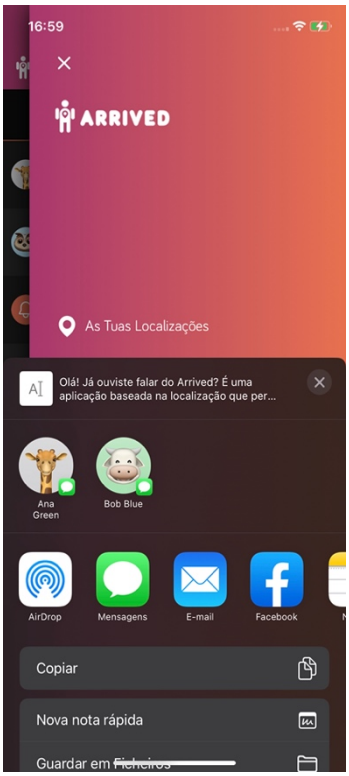

Figura 63 - Convite de instalação da aplicação Arrived

Quando o destinatário do convite seleciona o URL contido na mensagem, é redirecionado para a aplicação Arrived na App Store e poderá proceder à sua instalação. Até à data da entrega deste relatório, a publicação da aplicação na App Store, não estava finalizada, por este motivo o convite não contém o *link* da aplicação.

5.4.7.3. Política de Privacidade e Termos e condições

As duas últimas opções do menu lateral, contêm a informação da política de privacidade e termos e condições da aplicação Arrived (ver Figura 64). Uma única vista foi criada para exibir os dos tipos de informação separadamente, conforme a seleção do utilizador. O *link* para essas informações também pode ser encontrado na vista de configuração inicial dos dados do utilizador.

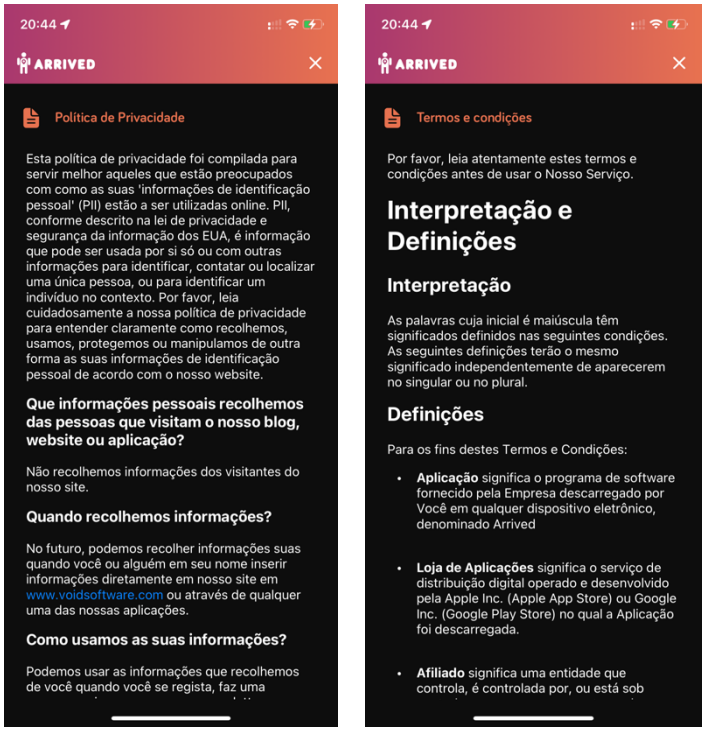

Figura 64 - Política de Privacidade e Termos e Condições da aplicação Arrived

5.5. Refactor no design da aplicação

A ideia da aplicação Arrived, surgiu de um dos membros da VOID Software, e já existia antes de se tornar um projeto de mestrado em parceria com a empresa. Por este motivo, já existia uma proposta de *design* anterior a que foi apresentada neste subcapítulo. Inicialmente a implementação da aplicação começou a ser feita com base no *design* anterior, mas ao longo desse processo, algumas funcionalidades foram acrescentadas e as ideias iniciais foram

sendo amadurecidas. Tornando esta implementação um desafio grande de adaptação face aos requisitos atuais da aplicação, que favorecesse também a experiência do utilizador.

Para resolver este desafio, foi proposto pelo responsável pelo *design*, ser feito um novo *design,* com *inputs* obtidos através dos testes manuais realizados por mim, que pudesse dar solução ao problema inicial considerando as novas funcionalidades, sem mudar o propósito da aplicação, mas que, em simultâneo, fosse atual e fizesse sentido para melhorar a UX. Essa decisão foi fundamental para alcançar o resultado obtido, mas é importante descrever o que o seu desenvolvimento implicou ao nível de trabalho.

A primeira mudança, aconteceu na vista de configurações iniciais da aplicação, onde o utilizador insere o seu número de telemóvel e as suas localizações favoritas. Tal como se pode ver na Figura 65, anteriormente o estado e navegação entre essas configurações, era apresentado na parte superior da vista. Isso aumentou a complexidade e consequentemente o tamanho do código do controlador dessa vista. Para resolver esse problema, numa primeira abordagem, optou-se por criar uma hierarquia de controladores pais e filhos, de modo a separar em duas vistas e controladores diferentes, a recolha do contacto e das localizações favoritas, mantendo a vista principal apenas para gerir os estados e apresentar os controladores filhos conforme a seleção do utilizador.

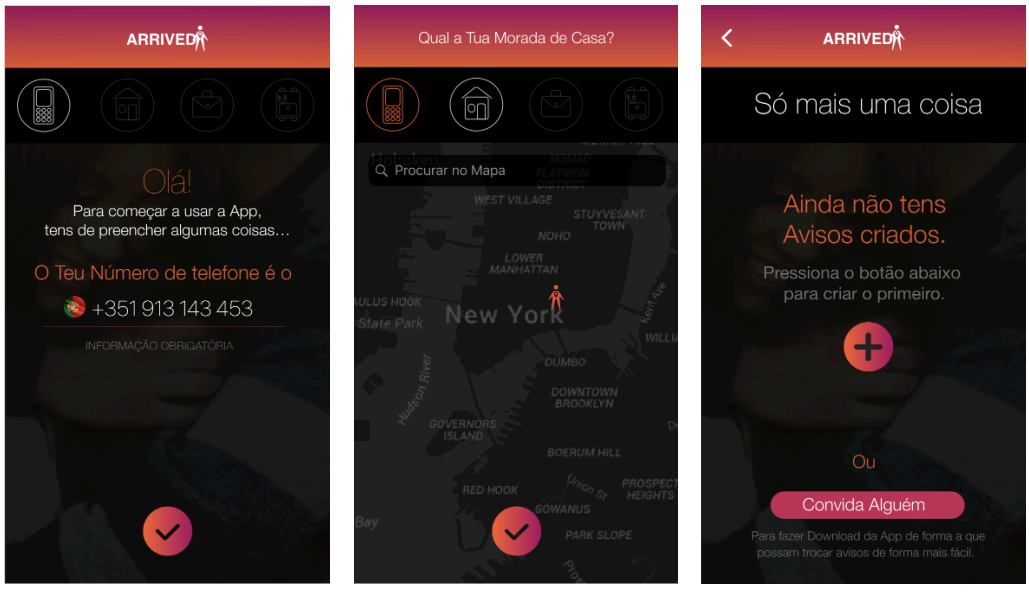

Figura 65 - Configuração inicial do utilizador no design antigo da aplicação Arrived

Já a abordagem atual da funcionalidade apresentada no subcapítulo 5.4.1, utiliza uma única vista e controlador, com a parte superior comum às duas configurações, mas tira

partido de uma tabela (UITableView [37]), que permite separar a lógica e o conteúdo de cada secção em células/componentes que, por sua vez, preenchem a tabela conforme o que é preciso apresentar. Ou seja, quando o utilizador seleciona a primeira secção, para preencher o seu contacto, é apresentado na tabela uma célula que contém o campo do número de telefone. Por outro lado, quando seleciona a configuração das localizações, a tabela é preenchida com as células que contêm a informação de cada localização. Essa alteração contribuiu para mais flexibilidade na implementação e uma navegação mais fluida entre as opções.

Outra mudança significativa que envolve não só essa funcionalidade, mas também funcionalidades ligadas aos avisos e atualização de dados de localização (ver Figura 66), foi a reutilização da vista e controlador do mapa. Anteriormente essa vista e o seu código associado, eram replicados nos controladores, conforme a necessidade da sua utilização, o que acabou por trazer alguma complexidade, repetição e aumento de código nos controladores que possuíam um mapa. Na implementação atual existe apenas uma vista do mapa com o seu controlador, que é apresentado separadamente do controlador que o invoca para tratar das operações que envolvem o mapa.

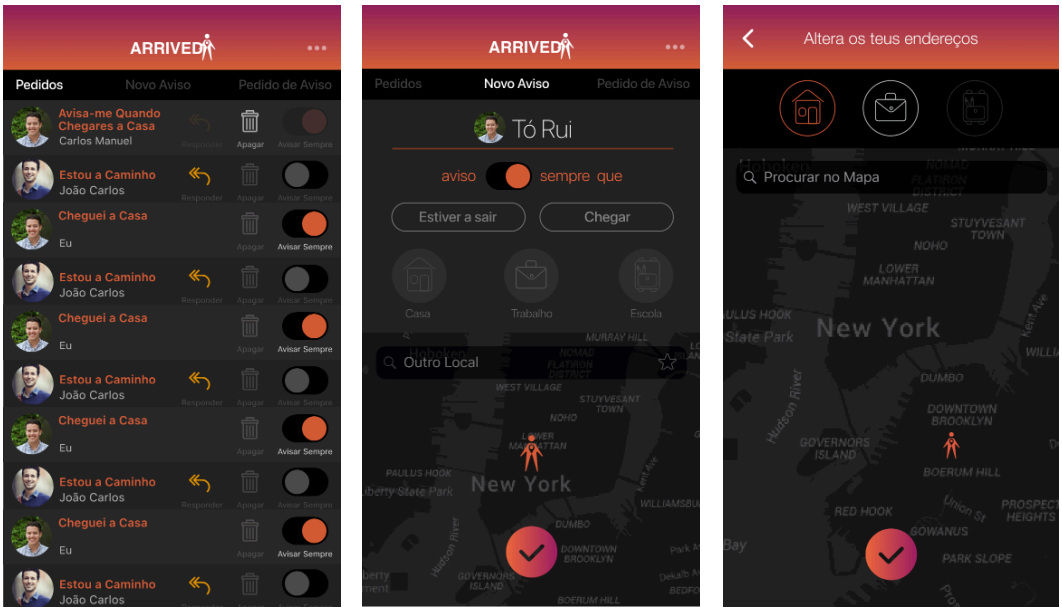

Figura 66 - Design antigo da listagem e criação de avisos e atualização das localizações principais da aplicação Arrived

Outra mudança ocorrida, foi a criação de um menu lateral, que na primeira versão não existia. Tal como é possível observar na Figura 66, a opção dada pelo botão apresentado no canto superior direito da aplicação, abria apenas a vista para edição das localizações favoritas

do utilizador. É possível notar, também, algumas diferenças nesta vista, nomeadamente a possibilidade de o utilizador adicionar mais localizações além das principais (casa, trabalho e escola), opção inexistente no *design* anterior.

Relativamente à página inicial da aplicação, que contém a listagem de avisos, é possível notar algumas alterações. Anteriormente, não havia forma de o utilizador filtrar os avisos conforme o seu tipo e as opções de criação dos avisos faziam parte do menu superior. Na versão atual o menu superior foi utilizado apenas para a listagem dos diferentes tipos de avisos, com a opção de criação dos avisos sempre disponível através de um botão localizado na parte inferior da vista principal.

Por fim, a vista da criação de avisos também sofreu algumas alterações, de modo a acomodar todos os parâmetros dos diferentes tipos de avisos. Nesta vista foi utilizada a mesma abordagem da vista de configurações iniciais do utilizador, com o uso de uma tabela preenchida com componentes separados, conforme a seleção do utilizador. O que ajudou a reduzir a complexidade, e otimizou a forma de acesso a essa funcionalidade, tornando mais simples a interação do utilizador com a aplicação.

5.6. Testes

Neste subcapítulo, serão apresentados os testes realizados na aplicação Arrived, bem como alguns resultados obtidos. Os testes foram divididos em três categorias: testes unitários, testes de UI e testes manuais. Para o desenvolvimento dos testes unitários e de UI foram criadas versões *mocks* dos repositórios de cada modelo de dados, de modo a criar um ambiente de testes completamente independente de dados reais da aplicação. Ambos os tipos de testes foram desenvolvidos com o uso da ferramenta XCTest [100], fornecida pelo Xcode.

Os testes unitários foram realizados com o objetivo garantir que cada componente da aplicação funcionasse corretamente e isoladamente. Esses testes foram sendo criados no fim de cada *task* implementada. Antes da integração de cada *branch* novo para o *branch* principal do repositório da aplicação, os testes eram executados para validar o funcionamento tanto da nova *task* como do que já havia sido previamente desenvolvido. Os testes unitários permitiram garantir o funcionamento correto do código e a deteção, e respetiva correção, de *bugs* ao longo de todo o desenvolvimento.

Relativamente aos testes de UI, foram criados vários casos de teste, que permitiram verificar o comportamento das vistas da aplicação em diferentes situações. Por exemplo, no

formulário do contacto do utilizador, foi possível desenvolver cenários para validar o comportamento da vista quando é inserido um número inválido, o que permitiu validar se a devida mensagem de erro era apresentada. Ou, quando um número válido era inserido, foi possível validar se a aplicação reencaminhava o utilizador para a vista correta.

Outro exemplo de teste de UI realizado, foi na vista do formulário de criação de um lembrete. Na criação de um lembrete sem inserir o campo da mensagem, era feita a verificação se uma mensagem de erro era apresentada, ou, caso a mensagem para a criação do lembrete fosse indicada, foi validado se uma mensagem de sucesso era apresentada. Com esses testes foi possível corrigir alguns comportamentos não esperados em algumas vistas da aplicação.

Além dos testes unitários e de UI, foram realizados testes manuais para garantir que a aplicação funcionasse corretamente em cenários reais de uso. Foram realizados testes em diversos cenários, nomeadamente:

- A criação de avisos para locais favoritos, como casa e trabalho, de modo a avisar outra pessoa que chegou ou saiu dessas localizações. Esses testes foram feitos com as possíveis formas de configuração de um aviso:
    - Quando o utilizador se aproximou ou se afastou da localização pretendida;
    - Com as duas frequências de envio, sempre ou apenas uma vez;
    - Com alterações feitas em parâmetros do aviso. Por exemplo, a localização, para testar se a notificação do aviso era recebida na nova localização.
- A criação de um pedido de aviso para outra pessoa. Esse teste foi feito com uma pessoa que viajou para outro país, de modo a validar o funcionamento desta funcionalidade em distâncias maiores. O pedido de aviso foi enviado a esta pessoa, sendo esse pedido aceite e criado no seu dispositivo. O objetivo era validar se essa pessoa receberia a notificação quando chegasse ao hotel do país de destino, e com a ação dessa notificação, conseguisse enviar a mensagem de confirmação ao remetente do pedido. O teste foi bem sucedido, e a mensagem de confirmação foi enviada;
- A criação de lembretes com base em localização. Foi criado um lembrete para quando houvesse uma aproximação de 100 metros de uma certa farmácia, uma notificação fosse recebida com o intuito de lembrar o utilizador de comprar a medicação X;

- Apagar um aviso. Foi realizado o teste para quando um aviso era apagado, para verificar se a sua notificação não era enviada depois dessa operação.

Os resultados obtidos nos testes manuais foram muito importantes para análise do comportamento da aplicação e considerados bastante satisfatórios, porque todos tiveram o resultado esperado. Foram identificados poucos ajustes necessários. Esses ajustes foram apenas relacionados à UX, como a adição de algumas mensagens de sucesso em falta, quando algumas operações fossem bem-sucedidas.

Um dos requisitos da aplicação é ser o mais *offline* possível, por este motivo, foram feitos alguns testes relativos à dependência de conectividade da aplicação com a *internet*. De modo a validar exatamente quais as funcionalidades que necessitam de conectividade para o seu funcionamento. O resultado desses testes mostrou que a única funcionalidade realmente dependente de conexão com a *internet*, é a criação e atualização de uma localização. Porque a *framework* MapKit requer conexão para efetuar o *download* dos mapas, para obter informações necessárias para renderizá-los na vista, bem como para obter as sugestões de localizações devolvidas na barra de pesquisa.

Esta restrição não impediu o funcionamento das funcionalidades principais da aplicação, como a visualização e criação de avisos, lembretes e pedidos de avisos, porque foi possível selecionar localizações previamente definidas para os criar. Relativamente ao recebimento das notificações quando o utilizador se aproxima ou se afasta de uma localização, a única diferença poderia ser notada na precisão da localização do dispositivo, porque a *framework* Core Location utiliza recursos como Wi-fi e torres de celular para obter a localização do dispositivo.

No caso de não haver conexão com a *internet*, a localização é obtida apenas com o recurso do GPS. Neste teste foi criado um aviso com um raio de 100 metros, o seu funcionamento foi como esperado, ao aproximar o dispositivo do local especificado, uma notificação foi lançada, não sendo possível notar nenhuma diferença acerca da precisão da localização.

Por fim, é importante notar que a combinação de testes unitários, de UI e manuais realizados, foi eficaz na identificação de problemas que puderam ser corrigidos, aumentando a qualidade da aplicação desenvolvida.

6. Conclusão

O objetivo deste projeto foi estudar a tecnologia *geofencing* e aplicações que utilizam esta tecnologia, bem como desenvolver uma aplicação iOS nativa que permitisse a criação de avisos, pedidos de avisos e lembretes com base em localização com o uso da tecnologia *geofencing*. A aplicação também teria de ser projetada para gerir todos os avisos *offline* e localmente, sem a utilização de APIs externas, e com o uso preferencial de *frameworks* nativas do iOS.

Para concluir este projeto, foram enfrentados diversos desafios, nomeadamente a necessidade de adquirir as competências necessárias para trabalhar com o desenvolvimento de aplicações nativas iOS e com as tecnologias envolventes, bem como a disciplina necessária para equilibrar o trabalho e o desenvolvimento do projeto em horário pós-laboral. A aquisição destas competências permitiu não só a criação de um produto de qualidade, mas também o desenvolvimento pessoal do percurso profissional na área do desenvolvimento de aplicações iOS e crescimento pessoal. Através do desenvolvimento da aplicação, foi possível explorar conceitos relacionados com a gestão de avisos baseados em localização e adquirir competências tecnológicas relevantes para o mercado atual.

No que se refere aos objetivos propostos para a aplicação Arrived, podemos afirmar que estes foram alcançados. A solução desenvolvida não só cumpre os requisitos iniciais definidos pelo cliente, mas ultrapassa-os ao mesmo tempo que se mantém focada na sua finalidade principal. Foi criada uma aplicação simples e intuitiva que utiliza a tecnologia de *geofencing* para a criação de lembretes para o utilizador, avisos e pedidos de avisos para outras pessoas, com base na localização e recorrendo a SMS, permitindo a gestão de todos os avisos de forma *offline*, sem necessidade de recorrer à APIs externas.

Em síntese, o projeto Arrived apresenta uma solução prática e inovadora para o problema de comunicação e coordenação entre pessoas relativamente a localizações específicas. A aplicação oferece uma *interface* simples e atraente que atende às necessidades de diferentes utilizadores, desde a segurança de crianças e idosos até lembretes relacionados com lugares específicos. Este projeto pode ainda contribuir para o avanço do conhecimento académico na área da tecnologia de *geofencing* e aplicações móveis.

6.1. Trabalho futuro

No que diz respeito ao trabalho futuro, a publicação da aplicação Arrived na App Store será uma prioridade. Adicionalmente, é possível que se desenvolva uma versão Android para atender a um público mais amplo. Outra adição importante para a aplicação iOS seria a possibilidade de o utilizador gerir as ações relacionadas com a aplicação através do Apple Watch. Além disso, a inclusão da opção de sincronização dos dados da aplicação com o CloudKit seria também uma grande mais-valia. Tais melhorias seriam importantes para garantir a competitividade da aplicação e atender às necessidades dos utilizadores, ao mesmo tempo expandir as possibilidades de desenvolvimento futuro.

Referências Bibliográficas

[1] "VOID Software :: the software specialists." https://void.pt/ (accessed Mar. 16, 2023).

[2] "What is geo-fencing (geofencing)? - Definition from WhatIs.com." https://www.techtarget.com/whatis/definition/geofencing (accessed Sep. 13, 2022).

[3] M. S. I M Zin, M. A. Sazali, A. A. M Isa, and M. S. M Isa, "Development of Auto-Notification Application for Mobile Device using Geofencing Technique 169," vol. 7, no. 2, pp. 2180–1843.

[4] "O que é geofencing? Como funciona? Veja 3 exemplos práticos." https://maplink.global/blog/o-que-e-geofencing/ (accessed Sep. 13, 2022).

[5] "Geofencing Market Size, Share and Global Market Forecast to 2022 | MarketsandMarkets." https://www.marketsandmarkets.com/Market-Reports/geofencing-market-209129830.html (accessed Sep. 13, 2022).

[6] "Shortcuts User Guide – Apple Support (UK)." https://support.apple.com/en-gb/guide/shortcuts/welcome/ios (accessed Mar. 07, 2023).

[7] "EgiGeoZone Geofence – Apps no Google Play." https://play.google.com/store/apps/details?id=de.egi.geofence.geozone&hl=pt_PT&gl=US (accessed Oct. 25, 2021).

[8] "Manual." https://www.egigeozone.de/manual/default_en.html#ZoneAnlegen (accessed Mar. 11, 2023).

[9] "Tasker – Apps no Google Play." https://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm&hl=pt_PT&gl=US (accessed Nov. 04, 2021).

[10] "Kaspersky Safe Kids: Proteja as suas crianças." https://www.kaspersky.pt/safe-kids (accessed Oct. 25, 2021).

[11] "Sobre nós | Kaspersky." https://www.kaspersky.pt/about (accessed Mar. 12, 2023).

[12] "Checkmark 2 on the App Store." https://apps.apple.com/us/app/checkmark-

2/id825863849 (accessed Oct. 25, 2021).

[13] "Location Reminder – Apps no Google Play." https://play.google.com/store/apps/details?id=com.dvtech.memento&hl=pt_PT&gl=US (accessed Nov. 01, 2021).

[14] "Google Maps – Apps no Google Play." https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=pt_PT&gl=US (accessed Nov. 12, 2021).

[15] "Geofency | Time Tracking on the App Store." https://apps.apple.com/us/app/geofency-time-tracking/id615538630 (accessed Oct. 25, 2021).

[16] "What Is Scrum: A Guide to the Most Popular Agile Framework." https://www.scrumalliance.org/about-scrum (accessed Jul. 20, 2022).

[17] "Kanban - A brief introduction | Atlassian." https://www.atlassian.com/agile/kanban (accessed Jul. 20, 2022).

[18] "Extreme Programming." https://www.cs.usfca.edu/~parrt/course/601/lectures/xp.html (accessed Jul. 20, 2022).

[19] "Kick off: o que é, benefícios e como fazer essa reunião?" https://rockcontent.com/br/blog/kick-off/ (accessed Sep. 11, 2022).

[20] "UIKit | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/ (accessed Nov. 17, 2021).

[21] "Core Data | Apple Developer Documentation." https://developer.apple.com/documentation/coredata (accessed Nov. 15, 2021).

[22] "Core Location | Apple Developer Documentation." https://developer.apple.com/documentation/corelocation (accessed Nov. 15, 2021).

[23] "MapKit | Apple Developer Documentation." https://developer.apple.com/documentation/mapkit/ (accessed Nov. 15, 2021).

[24] "User Notifications | Apple Developer Documentation."

https://developer.apple.com/documentation/usernotifications (accessed Nov. 15, 2021).

[25] "Message UI | Apple Developer Documentation." https://developer.apple.com/documentation/messageui (accessed Nov. 15, 2021).

[26] "Monitoring the User's Proximity to Geographic Regions | Apple Developer Documentation." https://developer.apple.com/documentation/corelocation/monitoring_the_user_s_proximity_to_geographic_regions (accessed Nov. 17, 2021).

[27] "UIKit | Apple Developer Documentation." https://developer.apple.com/documentation/uikit (accessed Apr. 26, 2022).

[28] "About App Development with UIKit | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/about_app_development_with_uikit (accessed Apr. 26, 2022).

[29] "App Icon Generator." https://appicon.co/ (accessed Apr. 27, 2022).

[30] "UIApplicationDelegate | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/uiapplicationdelegate (accessed Apr. 02, 2022).

[31] "UIDocument | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/uidocument (accessed Apr. 28, 2022).

[32] "UIView | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/uiview (accessed Apr. 28, 2022).

[33] "UIApplication | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/uiapplication (accessed Apr. 06, 2022).

[34] "Protecting the User's Privacy | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy (accessed Apr. 28, 2022).

[35] "App Store Connect workflow - App Store Connect Help." https://help.apple.com/app-store-connect/#/dev300c2c5bf (accessed Jul. 31, 2022).

[36] "CloudKit - iCloud - Apple Developer." https://developer.apple.com/icloud/cloudkit/ (accessed Jul. 31, 2022).

[37] "UITableView | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/uitableview (accessed Mar. 27, 2022).

[38] "UICollectionView | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/uicollectionview (accessed Apr. 19, 2022).

[39] "Creating a Core Data Model | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/creating_a_core_data_model (accessed Apr. 09, 2022).

[40] "Entity | Apple Developer Documentation." https://developer.apple.com/documentation/realitykit/entity (accessed Apr. 14, 2022).

[41] "Configuring Entities | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/modeling_data/configuring_entities (accessed Apr. 19, 2022).

[42] "Configuring Attributes | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/modeling_data/configuring_attributes (accessed Apr. 21, 2022).

[43] "Configuring Relationships | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/modeling_data/configuring_relationships (accessed Aug. 04, 2022).

[44] "NSManagedObject | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/nsmanagedobject (accessed Mar. 18, 2023).

[45] "Core Data Stack | Apple Developer Documentation."

https://developer.apple.com/documentation/coredata/core_data_stack (accessed Apr. 11, 2022).

[46] "NSManagedObjectModel | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/nsmanagedobjectmodel (accessed Apr. 11, 2022).

[47] "NSEntityDescription | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/nsentitydescription (accessed Apr. 11, 2022).

[48] "NSManagedObjectContext | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/nsmanagedobjectcontext (accessed Apr. 11, 2022).

[49] "NSPersistentStoreCoordinator | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/nspersistentstorecoordinator (accessed Apr. 11, 2022).

[50] "NSPersistentContainer | Apple Developer Documentation." https://developer.apple.com/documentation/coredata/nspersistentcontainer (accessed Apr. 11, 2022).

[51] "Repository - Architectural Pattern - Software-Pattern.org." http://software-pattern.org/Repository (accessed Sep. 05, 2022).

[52] "Repository pattern using Core Data and Swift - UserDesk." https://www.userdesk.io/blog/repository-pattern-using-core-data-and-swift/ (accessed Apr. 14, 2022).

[53] "GitHub - marmelroy/PhoneNumberKit: A Swift framework for parsing, formatting and validating international phone numbers. Inspired by Google's libphonenumber." https://github.com/marmelroy/PhoneNumberKit (accessed Apr. 06, 2022).

[54] "GitHub - google/libphonenumber: Google's common Java, C++ and JavaScript library for parsing, formatting, and validating international phone numbers." https://github.com/google/libphonenumber (accessed Apr. 06, 2022).

[55] "UITextField | Apple Developer Documentation."

https://developer.apple.com/documentation/uikit/uitextfield (accessed Apr. 07, 2022).

[56] "MKMapView | Apple Developer Documentation." https://developer.apple.com/documentation/mapkit/mkmapview (accessed Mar. 23, 2022).

[57] "Maps - Apple." https://www.apple.com/maps/ (accessed Mar. 23, 2022).

[58] "isScrollEnabled | Apple Developer Documentation." https://developer.apple.com/documentation/mapkit/mkmapview/1451998-isscrollenabled (accessed Mar. 24, 2022).

[59] "isZoomEnabled | Apple Developer Documentation." https://developer.apple.com/documentation/mapkit/mkmapview/1452577-iszoomenabled (accessed Mar. 24, 2022).

[60] "delegate | Apple Developer Documentation." https://developer.apple.com/documentation/uikit/uiapplication/1622936-delegate (accessed Mar. 24, 2022).

[61] "MKMapViewDelegate | Apple Developer Documentation." https://developer.apple.com/documentation/mapkit/mkmapviewdelegate (accessed Mar. 24, 2022).

[62] "MKAnnotationView | Apple Developer Documentation." https://developer.apple.com/documentation/mapkit/mkannotationview (accessed Mar. 26, 2022).

[63] "MKPointAnnotation | Apple Developer Documentation." https://developer.apple.com/documentation/mapkit/mkpointannotation (accessed Mar. 26, 2022).

[64] "MKLocalSearch | Apple Developer Documentation." https://developer.apple.com/documentation/mapkit/mklocalsearch (accessed Mar. 22, 2022).

[65] "completionHandler | Apple Developer Documentation." https://developer.apple.com/documentation/corespotlight/cssearchquery/1649312-

completionhandler (accessed Mar. 27, 2022).

[66] “Error | Apple Developer Documentation.” https://developer.apple.com/documentation/swift/error (accessed Mar. 27, 2022).

[67] “MKLocalSearchCompleter | Apple Developer Documentation.” https://developer.apple.com/documentation/mapkit/mklocalsearchcompleter (accessed Mar. 27, 2022).

[68] “queryFragment | Apple Developer Documentation.” https://developer.apple.com/documentation/mapkit/mklocalsearchcompleter/1452555-queryfragment (accessed Mar. 29, 2022).

[69] “MKLocalSearchCompletion | Apple Developer Documentation.” https://developer.apple.com/documentation/mapkit/mklocalsearchcompletion (accessed Mar. 27, 2022).

[70] “MKLocalSearch.Request | Apple Developer Documentation.” https://developer.apple.com/documentation/mapkit/mklocalsearch/request (accessed Mar. 29, 2022).

[71] “iBeacon - Apple Developer.” https://developer.apple.com/ibeacon/ (accessed Mar. 30, 2022).

[72] “CLLocationManager | Apple Developer Documentation.” https://developer.apple.com/documentation/corelocation/cllocationmanager (accessed Apr. 02, 2022).

[73] “CLLocation | Apple Developer Documentation.” https://developer.apple.com/documentation/corelocation/cllocation (accessed Apr. 02, 2022).

[74] “CLPlacemark | Apple Developer Documentation.” https://developer.apple.com/documentation/corelocation/clplacemark (accessed Apr. 02, 2022).

[75] “CLGeocoder | Apple Developer Documentation.” https://developer.apple.com/documentation/corelocation/clgeocoder (accessed Apr. 05, 2022).

[76] "reverseGeocodeLocation(_:completionHandler:) | Apple Developer Documentation." https://developer.apple.com/documentation/corelocation/clgeocoder/1423621-reversegeocodelocation (accessed Apr. 05, 2022).

[77] "locationManagerDidChangeAuthorization(_:) | Apple Developer Documentation." https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/3563956-locationmanagerdidchangeauthoriz (accessed Apr. 02, 2022).

[78] "locationManager(_:didUpdateLocations:) | Apple Developer Documentation." https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423615-locationmanager (accessed Apr. 02, 2022).

[79] "UserDefaults | Apple Developer Documentation." https://developer.apple.com/documentation/foundation/userdefaults (accessed Apr. 02, 2022).

[80] "Registering Your App with APNs | Apple Developer Documentation." https://developer.apple.com/documentation/usernotifications/registering_your_app_with_apns (accessed Jul. 25, 2022).

[81] "Asking Permission to Use Notifications | Apple Developer Documentation." https://developer.apple.com/documentation/usernotifications/asking_permission_to_use_notifications (accessed Jul. 23, 2022).

[82] "Local Notifications: Getting Started | raywenderlich.com." https://www.raywenderlich.com/21458686-local-notifications-getting-started (accessed Jul. 25, 2022).

[83] "Message UI | Apple Developer Documentation." https://developer.apple.com/documentation/messageui (accessed Oct. 25, 2021).

[84] "MFMessageComposeViewController | Apple Developer Documentation." https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontroller (accessed Jul. 22, 2022).

[85] "Contacts | Apple Developer Documentation." https://developer.apple.com/documentation/contacts (accessed Jul. 26, 2022).

[86] “CNContact | Apple Developer Documentation.” https://developer.apple.com/documentation/contacts/cncontact (accessed Jul. 31, 2022).

[87] “CNContactPickerViewController | Apple Developer Documentation.” https://developer.apple.com/documentation/contactsui/cncontactpickerviewcontroller (accessed Jul. 31, 2022).

[88] “CNContactPickerDelegate | Apple Developer Documentation.” https://developer.apple.com/documentation/contactsui/cncontactpickerdelegate (accessed Jul. 31, 2022).

[89] “App Localization in Swift — iOS Swift Guide | by Baptiste Montagliani | The Startup | Medium.” https://medium.com/swlh/app-localization-in-swift-ios-swift-guide-baa2c2e4298e (accessed Sep. 11, 2022).

[90] “MongoDB Realm | MongoDB.” https://www.mongodb.com/realm/lp?utm_source=google&utm_campaign=gs_footprint_row_search_brand_realm_atlas_product_desktop&utm_term=realm&utm_medium=cpc_paid_search&utm_ad=p&utm_ad_campaign_id=11303420057&adgroup=118482427679&gclid=CjwKCAjwu_mSBhAYEiwA5BBmf7XGW2s5xfBKfRT5-D-3cBPmPUc_x2fOACoPmYG1hckZXxk_8kKYFRoC7UMQAvD_BwE (accessed Apr. 19, 2022).

[91] “UserDefaults | Apple Developer Documentation.” https://developer.apple.com/documentation/foundation/userdefaults (accessed Apr. 14, 2022).

[92] “Defining a custom URL scheme for your app | Apple Developer Documentation.” https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app (accessed Sep. 03, 2022).

[93] “Qual é a diferença entre iMessage e SMS/MMS? - Suporte Apple (PT).” https://support.apple.com/pt-pt/HT207006 (accessed Sep. 03, 2022).

[94] “SMS Messaging API | Twilio.” https://www.twilio.com/pt-br/sms (accessed Sep. 03, 2022).

[95] “Messages API | Vonage.” https://www.vonage.com/communications-apis/messages/ (accessed Sep. 03, 2022).

[96] “Advanced Encryption Standard (AES) - GeeksforGeeks.” https://www.geeksforgeeks.org/advanced-encryption-standard-aes/ (accessed Mar. 31, 2023).

[97] “AES | Apple Developer Documentation.” https://developer.apple.com/documentation/cryptokit/aes (accessed Mar. 31, 2023).

[98] “combined | Apple Developer Documentation.” https://developer.apple.com/documentation/cryptokit/aes/gcm/sealedbox/combined (accessed Mar. 31, 2023).

[99] “UIActivityViewController | Apple Developer Documentation.” https://developer.apple.com/documentation/uikit/uiactivityviewcontroller (accessed Mar. 25, 2023).

[100] “XCTest | Apple Developer Documentation.” https://developer.apple.com/documentation/xctest (accessed Mar. 26, 2023).

[101] “NSNumber | Apple Developer Documentation.” https://developer.apple.com/documentation/foundation/nsnumber (accessed Apr. 21, 2022).

[102] “Setting Up a Core Data Stack | Apple Developer Documentation.” https://developer.apple.com/documentation/coredata/setting_up_a_core_data_stack (accessed Apr. 11, 2022).

[103] “View Controllers | Apple Developer Documentation.” https://developer.apple.com/documentation/uikit/view_controllers (accessed Apr. 11, 2022).

[104] “Dependency Injection - Creational Pattern - Software-Pattern.org.” http://software-pattern.org/Dependency-Injection (accessed Sep. 05, 2022).

Anexos

- Anexo A – Criação do ficheiro de modelo de dados no Core Data
- Anexo B – Criação de uma Entity no Core Data
- Anexo C – Configuração dos atributos de uma entidade no Core Data
- Anexo D – Criação de um relacionamento entre modelos do Core Data
- Anexo E – Geração das subclasses dos modelos das entidades no Core Data
- Anexo F – Configuração do Core Data Stack

Anexo A – Criação do ficheiro de modelo de dados no Core Data

O ficheiro de modelo de dados do Core Data pode ser adicionado na criação do projeto Xcode ao selecionar a opção *Use Core Data* como podemos ver na Figura 67, ou pode ser adicionado a um projeto existente, como foi o caso da Arrived (ver Figura 68).

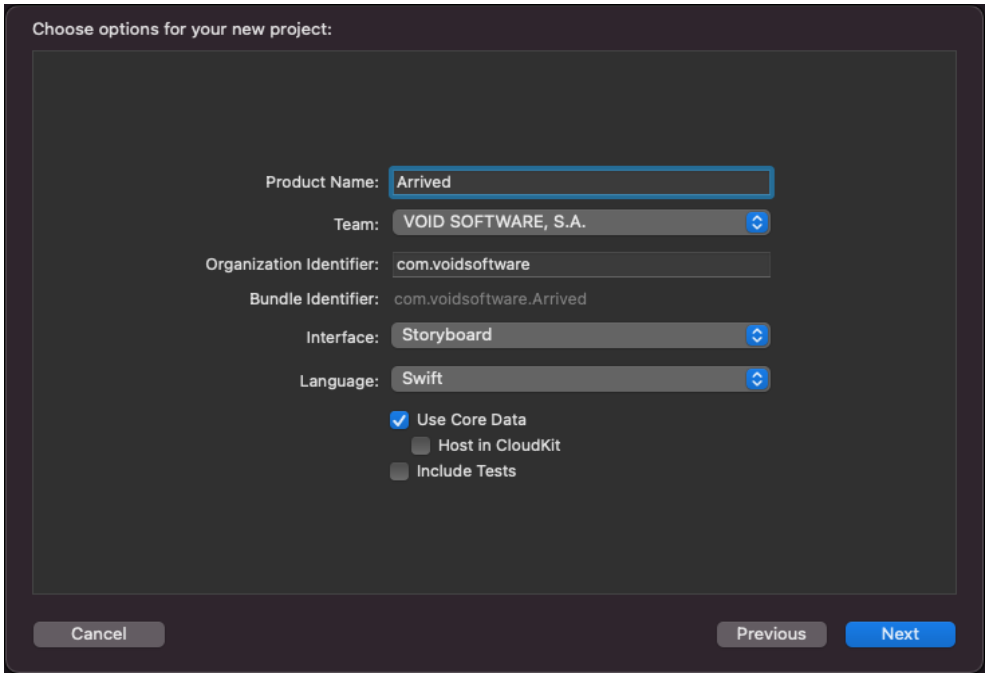

Figura 67 - Criação de um projeto no Xcode com o Modelo do Core Data

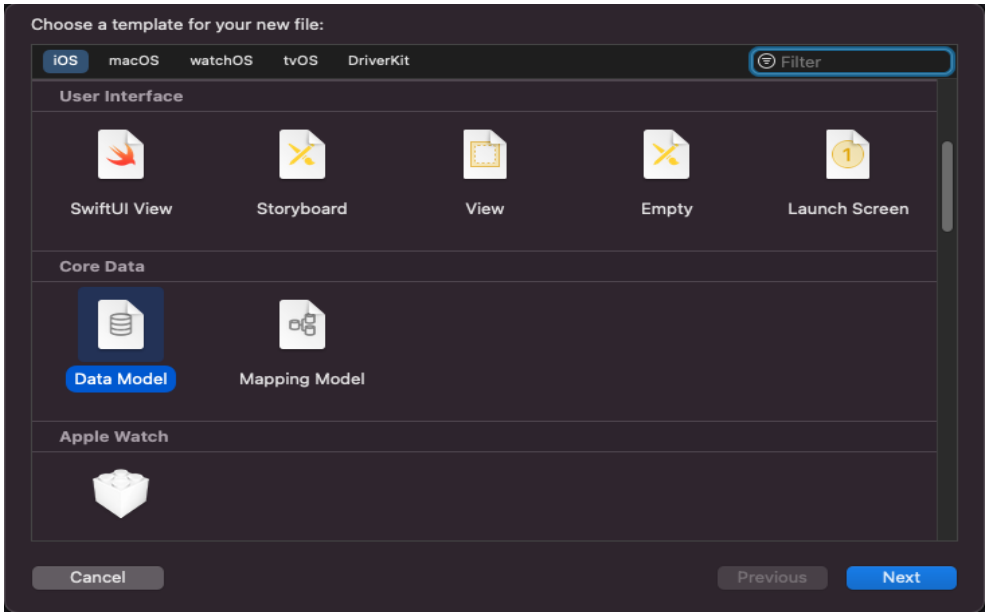

Figura 68 - Criação de um modelo do Core Data num projeto Xcode existente

Anexo B – Criação de uma Entity no Core Data

As entidades do Core Data são criadas no ficheiro de modelo de dados e para isso apenas são necessários dois passos (ver Figura 69):

1. Clicar em *Add Entity* na parte inferior da área do editor, e dessa ação surgirá na listagem de entidades uma entidade com o *placeholder* chamado **Entity**;

2. Depois é necessário fazer um duplo clique na entidade recém-adicionada e renomeá-la. Esta etapa atualiza o nome da entidade e o nome da classe, visíveis no *Data Model inspector*.

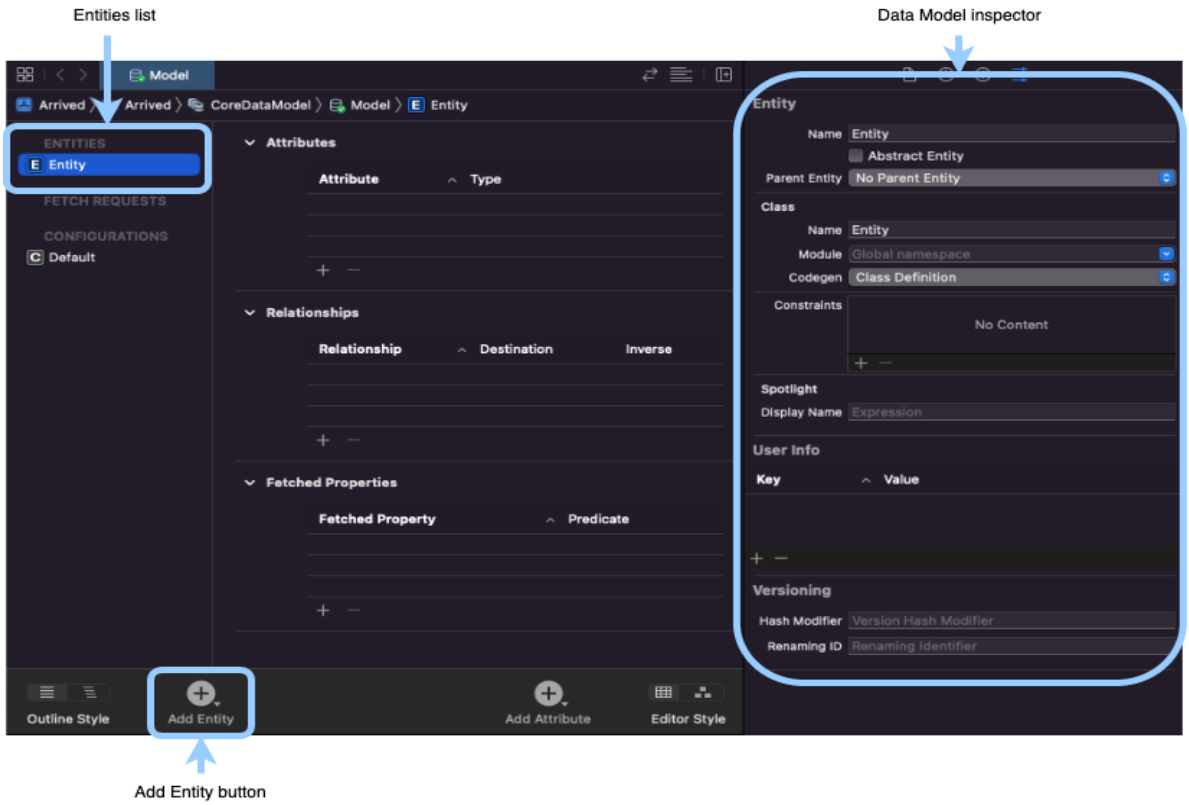

Figura 69 - Criação de uma Entity do Core Data no Xcode

Anexo C – Configuração dos atributos de uma entidade no Core Data

Para explicar como é feita a configuração dos atributos de uma entidade, irá ser usada para exemplo umas das entidades criadas na Arrived, a entidade **Location** e para isso os seguintes passos foram executados (ver Figura 70):

1. Criação de um atributo – A entidade **Location** foi selecionada, e ao clicar em *Add attribute* na parte inferior da área do editor, um novo atributo com o *placeholder* chamado *atribute* e com o tipo *Undefined* surgiu na lista de atributos;

2. Renomeação do atributo criado - Na lista de atributos, apenas foi necessário fazer duplo clique no atributo e renomeá-lo, neste exemplo o atributo foi renomeado para **latitude**;

3. Escolha do tipo de dados do atributo - Na lista Atributos, como podemos ver na Figura 71, apenas foi necessário clicar em *Undefined* e escolher o tipo de dados do atributo no menu *Type*, que no caso do atributo **latitude** ficou definido como Double.

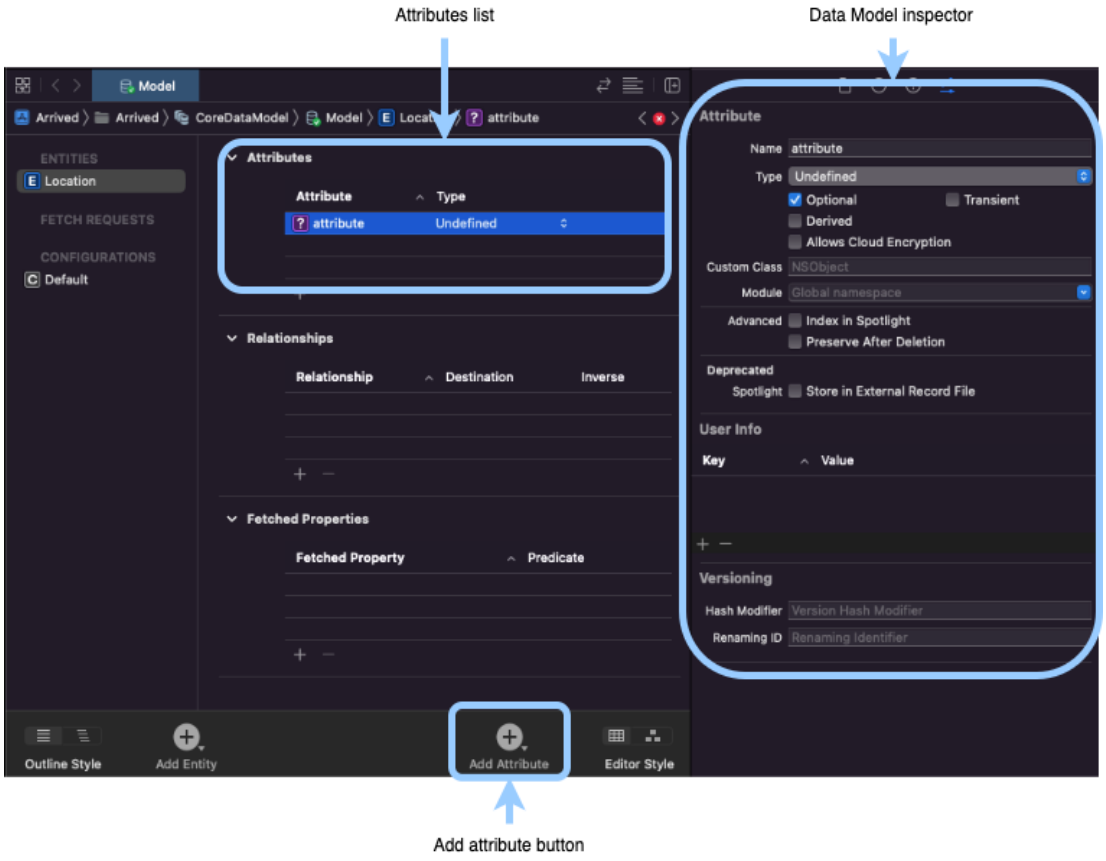

Figura 70 - Criação de um atributo de uma Entity do Core Data no Xcode

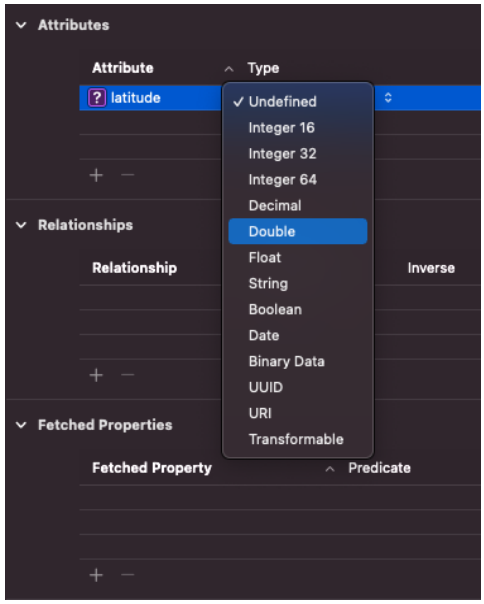

Figura 71 – Definição do tipo de dados de um atributo do Core Data no Xcode

Para configurar os atributos bastou um clique em cima do atributo a configurar, sendo os seus detalhes apresentados no Data Model Inspector, tal como podemos ver na Figura 72. Neste exemplo temos o atributo **latitude** do tipo Double, mas conforme o tipo do atributo, alguns dos parâmetros a configurar poderão ser diferentes.

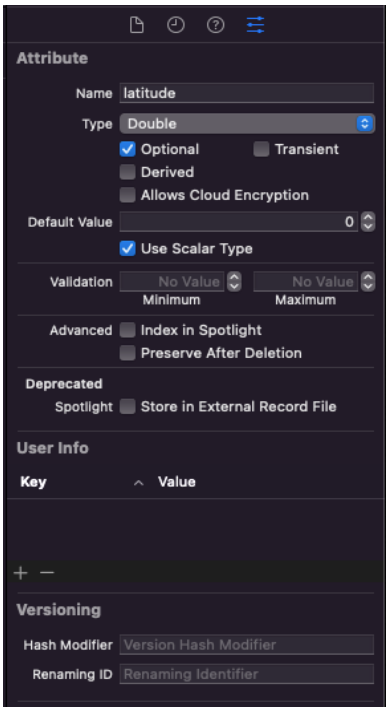

Figura 72 – Detalhes de um atributo a configurar de uma entidade do Core Data no Data Model inspector do Xcode

E os parâmetros que podem ser configurados através do Data Model Inspector são:

- *Attribute Type* – O tipo de dados do atributo;
- *Optional* – Especifica se caso o atributo precise ter um valor ou não quando for guardado;
- *Transient* – Se um atributo for *transient* então não será guardado no armazenamento persistente, mas pode ser útil para armazenarem temporariamente valores calculados ou derivados;
- *Allows Cloud Encryption* – Para armazenamentos persistentes gravados numa base de dados CloudKit, esta opção instruí o sistema a criptografar o valor do atributo antes de enviá-lo ao iCloud;
- *Default Value* – Permite fornecer um valor *default* para grande parte dos tipos;
- *Use Scalar Type* - Para alguns tipos, permite a escolha entre representações escalares e não escalares durante a geração de código. Por exemplo, para um *Double*, marcar a caixa de seleção *Use Scalar* produz um *Double*, enquanto desmarcá-la produz um NSNumber [101];
- *Validation* – Dá a opção de definir regras de validação, como, por exemplo, valores mínimo e máximo para um tipo numérico;
- *Index in Spotlight* - Adiciona o campo ao índice *Spotlight* para instâncias criadas a partir desta entidade;
- *Preserve After Deletion* - Inclui o atributo na marca de exclusão desta entidade. Quando o rastreamento de histórico persistente é ativado e um objeto é excluído através de um contexto, o Core Data regista um marcador de identificação, conhecido como a sua marca de exclusão, na transação relevante;
- *User Info* – Permite o armazenamento de qualquer informação específica da aplicação relacionada ao atributo através de um dicionário;
- *Versioning Renaming ID* – Caso um atributo seja renomeado entre as versões do modelo, permite a definição de um identificador de renomeação no novo modelo para o nome do atributo correspondente no modelo anterior.

Anexo D – Criação de um relacionamento de modelos do Core Data

Para adicionar um relacionamento entre dois modelos do Core Data apenas dois passos precisam ser executados:

1. Selecionar a opção gráfica do editor do modelo para ser possível visualizar todas as entidades da aplicação, tal como podemos observar na Figura 73;

2. Fazer um *control-drag* de uma entidade para outra para criar um par de relacionamentos. Essa ação fará com que uma seta apareça entre as entidades (ver Figura 73) para indicar a criação de um relacionamento com o nome *newRelationship* em cada entidade.

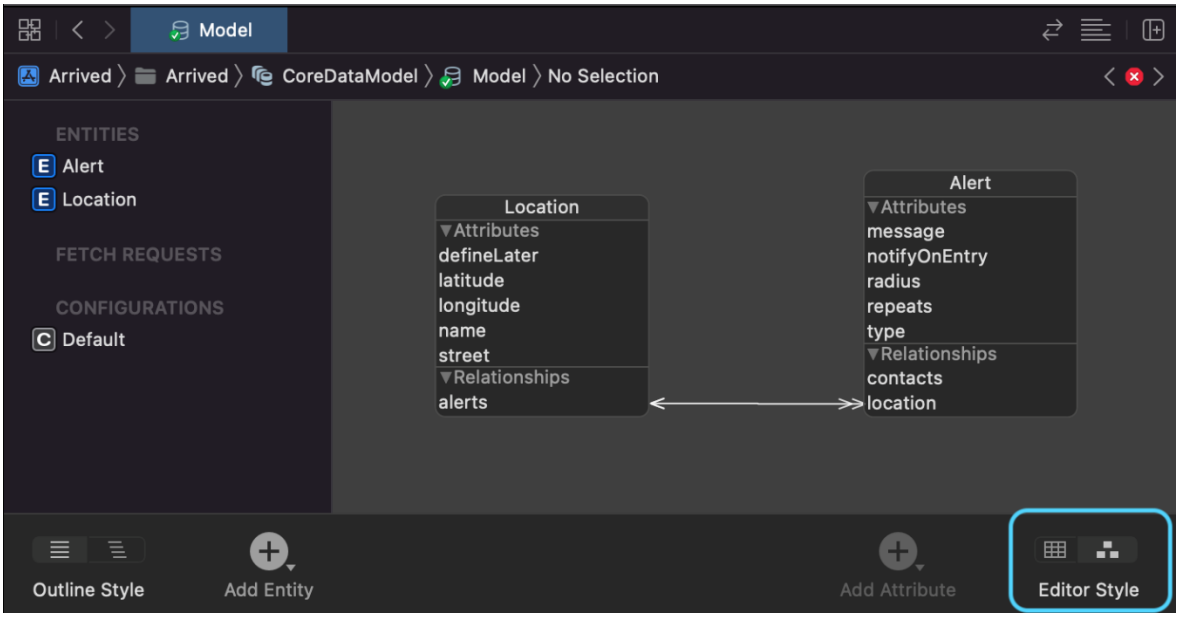

Figura 73 - Opção de mudança de estilo de visualização e relações entre entidades do Core Data

Após a criação das relações é necessário configurar cada relacionamento e para explicar como essa configuração é feita, será usado o exemplo de uma relação entre duas entidades implementadas na Arrived a entidade **Location**, já citada anteriormente, e a entidade **Alert**. Tal como podemos visualizar na Figura 74, para introduzir essa configuração foi preciso executar os seguintes passos:

1. Alterar a visualização do editor para o estilo de tabela para editar uma entidade de cada vez;

2. Abrir o Data Model Inspector, (menu mostrado na lateral direita da Figura 74);

3. Selecionar a entidade de origem na lista de entidade de origem e a seguir selecionar o novo relacionamento na lista de relacionamentos. No Data Model Inspector é necessário configurar os parâmetros *name*, *destination*, *inverse*, *delete rule*, e *cardinality type* e indicar se é *transient* ou *optional*;

4. Selecionar a entidade de destino na lista de entidades e a seguir selecionar o novo relacionamento na lista de relacionamentos, e configurar os mesmos parâmetros citados no ponto 3.

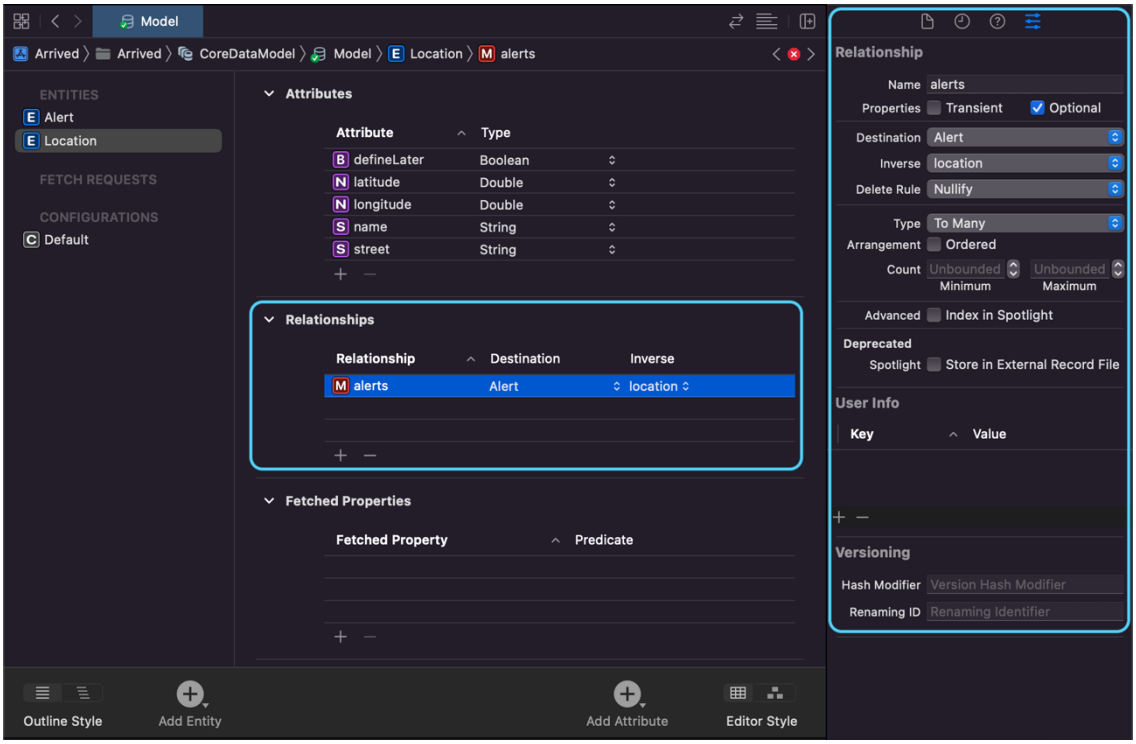

Figura 74 - Configuração de ralações do Core Data no Xcode

Este exemplo, mostra a relação entre um aviso e a localização em que este é despoletado. Enquanto um aviso só tem uma localização, uma localização poderá estar presente em vários avisos, configurando uma relação *one to many*. Para uma melhor compreensão dos parâmetros configurados no menu Data Model Inspector, abaixo teremos uma breve descrição de cada um:

- *Transient* – Esses relacionamentos não são salvos no armazenamento persistente, mas podem ser úteis para armazenar temporariamente valores calculados;

- *Optional* - Esses relacionamentos não precisam ter instâncias do seu tipo de destino, já um obrigatório deve apontar para uma ou mais instâncias no tipo de destino;

- *Destination* – Cada relacionamento aponta de uma entidade de origem para uma de destino;

- *Inverse* – Esses relacionamentos inversos permitem que o Core Data propague alterações em ambas as direções quando uma instância do tipo de origem ou de destino é alterada;

- *Delete Rule* – Essa regra descreve como as alterações se propagam entre os relacionamentos quando uma instância de origem é excluída do Core Data e para isso é necessário a escolha de uma das seguintes opções:

    - *No action* – Opção que exclui a instância do objeto de origem, mas mantém referências a esta, em qualquer instância do objeto de destino;

    - *Nullify* – Opção que exclui a instância do objeto de origem e anula as referências a esta, em qualquer instância do objeto de destino;

    - *Cascade* – Opção que exclui a instância do objeto de origem e, todas as instâncias do objeto de destino;

    - *Deny* – Opção que exclui o objeto de origem só se não apontar para nenhuma instância do objeto de destino.

- *Cardinality Type* - Regra que especifica o tipo do relacionamento como sendo *to one* ou *to many*. Os relacionamentos *to one* conectam a origem a uma única instância do tipo de destino, já o *to many* conectam a origem a um conjunto mutável do tipo de destino e opcionalmente as seguintes opções podem ser configuradas:

    - *Arrangement* – Opção que indica que o relacionamento tem uma ordenação inerente permitindo gerar um conjunto mutável ordenado;

    - *Count* – Opção que permite a indicação de limites superiores e inferiores do número de instâncias de destino.

Anexo E – Geração das subclasses dos modelos das entidades do Core Data

Para selecionar a opção de geração das subclasses dos modelos das entidades, apenas foi preciso selecionar uma entidade na lista de entidades e selecionar a opção no menu Codegen do Data Model Inspector. No caso da Arrived a opção *Manual/None* foi selecionada. Essa opção foi escolhida porque dá a liberdade para editar as propriedades da subclasse do objeto

gerido, por exemplo, caso seja preciso alterar modificadores de acesso e adicionar métodos adicionais ou lógica de negócios.

Já a opção *Class definition* é indicada quando não é preciso editar as propriedades ou a funcionalidade da subclasse do objeto gerido e os ficheiros de propriedades que o Core Data gera. E por fim, a opção *Category/Extension* é escolhida para adicionar métodos adicionais ou lógica de negócios na subclasse do objeto gerido, permitindo o controlo total do ficheiro da classe, enquanto continua a gerar automaticamente o ficheiro de propriedades para mantê-lo atualizado com o editor de modelo.

Anexo F – Configuração do Core Data Stack

Para configurar as classes que gerem e persistem os objetos da aplicação [102], o primeiro passo é feito ao inicializar o NSPersistentContainer. Geralmente, o Core Data é inicializado durante a inicialização da aplicação, fazendo com que o NSPersistentContainer seja instanciado no *delegate* da aplicação [30].

De forma a evitar problemas de concorrência, que poderão surgir no futuro, na implementação da Arrived optou-se por ao invés de declarar a instância do NSPersistentContainer no *delegate* da aplicação. Essa instância foi declarada numa classe de instância partilhada, denominada CoreDataStack que, por sua vez, foi criada não só para acomodar o *persistent container* mas também a instância do NSManagedObjectContext. Este é responsável pela execução das operações de armazenamento e consulta de dados do Core Data e, por fim, a função *save*, responsável por guardar as operações feitas no Core Data, que melhora o desempenho da persistência, guardando o contexto somente quando houver alterações.

Depois de criado, o *persistent container* mantém referências às instâncias do modelo e *store coordinator* nas suas propriedades *managedObjectModel*, *viewContext* e *persistStoreCoordinator*, respetivamente. Tornando possível a passagem de uma referência ao *container* para a interface do utilizador, ou seja, bastando apenas adicionar o *import* do Core Data no *root* View Controller [103] da aplicação, e criar uma variável para manter uma referência ao *persistent container*.

Apesar de ser possível em ambas as opções de instanciação do *persistent container* aceder a essa instância em qualquer controlador da aplicação, na Arrived essa implementação foi feita de forma a se adaptar aos padrões de arquitetura MVVM, Repository

[51] e Dependency Injection [104]. Tal como foi dito anteriormente, é possível passar o valor do *persistent container* para o *root* ViewController e os restantes controladores que necessitarem da mesma, mas geralmente é uma mais-valia a criação de uma camada de abstração entre a camada de negócios e a camada de armazenamento, por isso, essa atribuição passou a ser feita primeiramente no(s) repositório(s) do(s) modelo(s) de dados usado(s) no ViewModel de cada ViewController da aplicação através de Dependency Injection.