Mestrado em Informática e Sistemas

Cat|Mobile

Relatório de Estágio apresentado para a obtenção do grau de Mestre em Informática e Sistemas
Especialização em Desenvolvimento de Software

Autor
Diogo Renato Tavares Gomes

Orientador
José Fernando Fachada Rosado
Professor do Departamento de Engenharia Informática e Sistemas
Instituto Superior de Engenharia de Coimbra

Supervisor
Daniel Ribeiro Margarido
Dognædis

Coimbra, Dezembro, 2018

Resumo

O presente documento representa o trabalho realizado por Diogo Renato Tavares Gomes, durante o estágio incluído no Mestrado em Informática e Sistemas, ramo de Especialização em Desenvolvimento de Software, do Instituto Superior de Engenharia de Coimbra, na Dognædis, uma empresa de cibersegurança do grupo Prosegur. O estágio teve como objectivo principal a criação de um sistema de envio de alertas para dispositivos móveis de modo a permitir, à equipa de operadores, receber e responder a alertas mesmo estando fora da empresa. De entre diversas funcionalidades, o sistema desenvolvido combina o envio de notificações destinadas a operadores específicos com capacidade de partilha de dispositivos móveis entre operadores, mantendo inexistência de dados persistidos na aplicação móvel do dispositivo. Todos os dados usados pela aplicação são obtidos do servidor em tempo real. Os operadores, destinatários das notificações, são determinados por escalonamento pré-definido por um administrador. A aplicação de *backend*, que dá suporte à aplicação móvel, serve também para o administrador gerir os operadores e supervisionar as respostas aos alertas, podendo também consultar o volume de alertas e históricos de actividade no sistema.

Palavras-chave:
Mobilidade; Escalonamento; Cibersegurança; Alertas; Notificações Push móveis.

Abstract

This document represents the work done by Diogo Renato Tavares Gomes, during the internship included in the Master in Informatics and Systems, Specialization in Software Development branch, of the Instituto Superior de Engenharia de Coimbra, at Dognædis, a cybersecurity company of the Prosegur group. The main objective of the internship was to create a mobile alarm system to allow operators to receive and respond to alerts even when outside the company. Among several features, the developed system combines the sending of notifications to specific operators with the ability to share mobile devices between operators, while maintaining the lack of persistent data in the mobile application on the device. All data used by the application is retrieved from the server in real time. The operators, recipients of the notifications, are determined by pre-defined scheduling by an administrator. The backend application, which supports the mobile application, also allow the administrator to manage the operators and to supervise the responses to the alerts, and also to watch the volume of alerts and history of activity in the system.

Key words:
Mobility; Scheduling; Cibersecurity; Alerts; Mobile push notifications.

Agradecimentos

Agradeço ao Professor Doutor José Fernando Fachada Rosado por todo o apoio e disponibilidade que ofereceu enquanto meu orientador. Igualmente grato para com o Engenheiro Daniel Ribeiro Margarido que, como supervisor na empresa, desempenhou o seu papel com grande rigor, tendo sido um excelente anfitrião na empresa e sempre disponível para as diligências necessárias ao longo do projecto. Um agradecimento também a Catarina Martins, do departamento de Recursos Humanos, pelo apoio na logística inerente ao estágio e também pela sua constante disponibilidade no esclarecimento de questões diversas. Ao Engenheiro Gonçalo Amaro, agradecimentos pela proposta de estágio e apoio no esclarecimento de todas as questões. Agradecimentos também à restante equipa Dognædis, pelo excelente acolhimento, actividades de integração e pelo excelente ambiente de trabalho que proporcionou ao longo dos 7 meses de estágio. Agradeço ainda a todos os docentes do ISEC, que de alguma forma contribuíram para o meu *upgrade* de conhecimento enquanto aluno do Mestrado em Informática e Sistemas (MIS).

Dedicatória

Este trabalho é dedicado a toda a minha família, em especial à minha mãe Maria Tavares, ao meu pai Casimiro Gomes, à minha irmã Sara, à minha avó Arminda da Silva, aos meus avós do Vale do Milho, José Gomes e Maria Lopes, à minha companheira Cláudia Alexandre e família Alexandre, restantes amigos e colegas, por todo o apoio, paciência, presença e compreensão durante este percurso.

Conteúdo

Lista de Tabelas

Lista de Figuras

Definições e Acrónimos

AIDL - *Android Interface Definition Language* - em português, linguagem de descrição de interface para Android, é uma linguagem de programação utilizada para descrever a interface dos componentes de aplicações para Android.

API - *Application Programming Interface* - em português, interface de programação de aplicações, é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicações que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar os seus serviços.

CERT - *Computer Emergency Readiness Team* - em português, Equipa de Resposta a Emergências Informáticas, foi formada em Novembro de 1988 por Defense Advanced Research Projects Agency (DARPA) depois da Internet ter sofrido um incidente provocado por um *worm*. Actualmente, o CERT tem foco em falhas de segurança e incidentes de *denial-of-service*, providenciando alertas e instruções para lidar com incidentes assim como medidas de prevenção. O CERT também é responsável por se envolver em campanhas de consciencialização e conduzir pesquisas para aumentar a segurança de sistemas.

CFG - *Control Flow Graph* - em português, grafo de fluxo de controle é uma representação que usa notação de grafo para descrever todos os caminhos que podem ser executados por um programa de computador.

Cron O *cron* é um utilitário de Linux que permite agendar comandos ou *scripts* para correr automaticamente no servidor a determinada hora e data.

CSIRT - *Computer Security Incident Response Team* - é um grupo técnico responsável por resolver incidentes relacionados com segurança informática. Pode ser um serviço prestado por uma empresa especializada ou uma unidade da própria empresa.

CSS - *Cascading Style Sheets* - é um mecanismo para adicionar estilo (cores, fontes, espaçamento, etc.) a um documento para Web.

DEIS - *Departamento de Engenharia Informática e Sistemas* - é um dos departamentos do ISEC que, desde 1989, se dedica à formação, investigação, desenvolvimento e prestação de serviços na área da Engenharia Informática.

DOM - *Document Object Model* - em português, Modelo de Documento por Objectos, é uma convenção multi-plataforma e independente de linguagem para representação e interacção com objectos em documentos HTML, XHTML e XML.

DRF - *Django REST framework* - é uma poderosa *framework* para construir Web APIs em Django [1].

endpoint Um endpoint de uma API é o Uniform Resource Locator (URL) onde o serviço pode ser acedido por uma aplicação cliente.

ER - *Entidade Relacionamento* - em português, entidade relacionamento (modelo ER), é um modelo de dados para descrever os dados ou aspectos de informação de um domínio de negócio ou seus requisitos de processo, de uma maneira abstracta com objectivo de ser implementada em base de dados relacional.

FCM - *Firebase Cloud Messaging* - anteriormente conhecido como Google Cloud Messaging, é uma solução em *cloud*, multi-plataforma de mensagens e notificações para aplicativos Android, iOS e Web, de uso gratuito.

GCM - *Google Cloud Messaging* - é a antiga plataforma de mensagens para dispositivos móveis da Google que foi recentemente substituída pelo Firebase Cloud Messaging.

GIL - *Global Interpreter Lock* - em português, bloqueio de intérprete global é um mecanismo usado em linguagens de programação interpretadas, para sincronizar a execução de *scripts* para que apenas um *script* nativo possa ser executado de cada vez.

GUI - *Graphical User Interface* - em português, interface gráfica do utilizador, é um tipo de interface do utilizador que permite a interacção com dispositivos digitais por meio de elementos gráficos como ícones e outros indicadores visuais, em contraste com a interface de linha de comando.

HTML - *Hypertext Markup Language* - em português, Linguagem de Marcação de Hiper-Texto, é o componente mais básico da web. Serve para descrever e definir o conteúdo e a aparência de uma página web. Além do HTML, em geral outras tecnologias são usadas para descrever a apresentação/aparência (CSS) ou funcionalidade (JavaScript) das páginas Web.

HTTP - *Hypertext Transfer Protocol* - em português Protocolo de Transferência de Hipertexto, é um protocolo de comunicação utilizado em sistemas de informação, distribuídos e colaborativos. Este protocolo é a base para a comunicação de dados da *World Wide Web*.

HTTPS - *Hyper Text Transfer Protocol Secure* - em português, protocolo de transferência de hipertexto seguro, é uma implementação do protocolo HTTP sobre uma camada adicional de segurança que utiliza o protocolo SSL/TLS. Essa camada adicional permite que os dados sejam transmitidos por meio de uma conexão criptografada e que se verifique a autenticidade do servidor e do cliente por meio de certificados digitais.

IDE - *Integrated Development Environment* - em português, Ambiente de Desenvolvimento Integrado, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objectivo de agilizar este processo..

IETF - *Internet Engineering Task Force* - é um grupo informal internacional aberto, composto de técnicos, agências, fabricantes, fornecedores e pesquisadores, que se ocupa do desenvolvimento e promoção de standards para Internet, em estreita cooperação com o World Wide Web Consortium e ISO/IEC, em particular TCP/IP e o conjunto de protocolos Internet. O IETF tem como missão identificar e propor soluções a questões/problemas relacionados com a utilização da Internet, além de propor padronização das tecnologias e protocolos envolvidos.

IP - *Internet Protocol* - em português, Protocolo de Internet, é um protocolo de comunicação usado entre todas as máquinas de redes informáticas para encaminhamento dos dados.

IPC - *Inter-Process Communication* - em português, comunicação entre processos, é o grupo de mecanismos que permite aos processos transferirem informação entre si.

IPN - *Instituto Pedro Nunes* - criado em 1991, na sequência de uma iniciativa da Universidade de Coimbra, o Instituto Pedro Nunes - Associação para a Inovação e Desenvolvimento em Ciência e Tecnologia (IPN), é uma associação sem fins lucrativos que promove a inovação na área científica e tecnológica.

ISEC - *Instituto Superior de Engenharia de Coimbra* - é uma escola portuguesa de ensino superior politécnico, integrada no Instituto Politécnico de Coimbra. Considera-se um centro de transmissão e difusão de cultura, ciência e tecnologia, cabendo-lhe ministrar a preparação para o exercício de actividades profissionais no domínio da engenharia e promover o desenvolvimento da região em que se insere.

IT - *Information Technology* - em português, Tecnologia da Informação é conjunto de todas as actividades e soluções providas por recursos informáticos que visam a produção, o armazenamento, a transmissão, o acesso, a segurança e o uso das informações.

JS - *JavaScript* - é uma linguagem de programação interpretada. Foi originalmente implementada como parte dos navegadores web para que *scripts* pudessem ser executados do lado do cliente e interagissem com o utilizador sem a necessidade deste *script* passar

pelo servidor, controlando o navegador, realizando comunicação assíncrona e alterando o conteúdo do documento exibido.

JSON - *JavaScript Object Notation* - é um formato de troca de dados simples e rápida (*parsing*) de dados entre sistemas, independente de linguagem de programação, derivado do JavaScript, de formato compacto e padrão aberto independente.

LAN - *Local Area Network* - em português, rede de área local, em informática, consiste numa rede utilizada para interconexão de computadores e outros equipamentos informáticos cuja finalidade é a troca de dados dentro de uma área delimitada geograficamente.

MIS - *Mestrado em Informática e Sistemas* - curso do ISEC que tem por objectivo formar Mestres em Informática e em Sistemas capazes de exercerem a sua actividade profissional com um elevado nível de competência técnica, científica e profissional em cada uma das áreas de especialização proposta.

MIT - *Massachusetts Institute of Technology* - é uma licença onde qualquer pessoa que obtém uma cópia do software e seus arquivos de documentação associados pode lidar com eles sem restrição, incluindo sem limitação dos direitos a usar, copiar, modificar, publicar, distribuir ou vender cópias do software. As condições impostas para tal são apenas de manter o aviso de copyright e uma cópia da licença em todas as cópias do software.

MSS - *Managed Security Services* - em português, serviços de gestão de segurança, representa o segmento de mercado de empresas que, monitorizam ou fazem gestão de ambientes de TI remotamente, através do uso compartilhado de Centros de Operação de Segurança (SOC - Security Operation Center).

OOP - *Object-oriented programming* - em português, Programação orientada a objectos, é um paradigma de programação baseado no conceito de "objectos", que podem conter dados na forma de campos, também conhecidos como atributos, e códigos, na forma de procedimentos, também conhecidos como métodos.

ORM - *Object-relational mapping* - em português, Mapeamento objeto-relacional, é uma técnica de desenvolvimento utilizada para reduzir a dificuldade de conjugar a programação orientada aos objetos com a utilização de base de dados relacionais. As tabelas da base de dados são representadas através de classes e os registos de cada tabela são representados como instâncias das classes correspondentes.

Pull Ao contrário de Push, o Pull consiste no envio de informação por parte do servidor em resposta a um pedido do cliente.

Push Notificações Push, também chamadas Push de servidor, são envios de informação, com iniciativa no servidor, sem que tenha havido um pedido do cliente.

R&D - *Research And Development* - em português, desenvolvimento ou investigação e desenvolvimento é um termo que tem um significado comercial importante, independente da associação tradicional tecnológica da pesquisa e desenvolvimento.

REST - *Representational State Transfer* - em português, Transferência de Estado Representativo, é um tipo de arquitectura que define um conjunto de restrições e propriedades baseados em HTTP. Os *Web Services*, que obedecem à arquitectura REST, fornecem interoperabilidade entre sistemas de computadores na Internet.

RTIR - *Request Tracker for Incident Response* [2] - ferramenta de gestão de incidentes, analisada no Capítulo 3.

SaaS - *Software as a Service* - em português, software como serviço, é uma forma de distribuição e comercialização de software. No modelo SaaS, o fornecedor do software responsabiliza-se por toda a estrutura necessária à disponibilização do sistema (servidores, conectividade, cuidados com segurança da informação), e o cliente utiliza o software via Internet, eventualmente pagando um valor pelo serviço.

SDK - *Software Development Kit* - em português, Kit de desenvolvimento de software, é tipicamente um conjunto de ferramentas de desenvolvimento de software que permite a criação de aplicações para um certo pacote de software, *framework*, plataforma de *hardware*, sistema de computador, etc.

SGBD - *Sistema de Gestão de Base de Dados* - é o *software* responsável pela gestão de bases de dados. O principal objectivo é retirar da aplicação cliente a responsabilidade de gerir o acesso, a persistência, a manipulação e a organização dos dados.

SMS - *Short Message Service* - em português, serviço de mensagens curtas, é um serviço disponível em telemóveis digitais que permite o envio de mensagens curtas (até 160 caracteres) entre estes equipamentos e entre outros dispositivos móveis, e até entre telefones fixos, conhecidas popularmente como mensagens de texto.

SOC - *Security Operation Center* - em português, Centro de Operações de Segurança, é um termo genérico que descreve parte ou a totalidade de uma plataforma cujo objectivo é prestar serviços de detecção e reacção a incidentes de segurança.

TCP - *Transmission Control Protocol* - em português, Protocolo de Controle de Transmissão, é um dos protocolos sob os quais assenta a Internet. O TCP é um protocolo de nível da camada de transporte (camada 4) do Modelo OSI, sobre o qual que se assentam a maioria dos serviços informáticos em rede..

TTL - *Time To Live* - apesar de ser um termo também usado nas redes TCP/IP, no contexto em que é usado neste documento, TTL tem outro significado e representa literalmente o tempo útil de vida, neste caso, de uma mensagem FCM, tal como referido em [3].

UML - *Unified Modeling Language* - em português, Linguagem de Modelagem Unificada , é uma linguagem-padrão para a elaboração da estrutura de projectos de software. Esta linguagem pode ser usada para a visualização, especificação, construção e documentação de artefactos que façam uso de sistemas complexos de software.

URL - *Uniform Resource Locator* - em português, Localizador Uniforme de Recursos, é endereço de rede, no qual se encontra algum recurso informático.

VPN - *Virtual Private Network* - em português, rede de comunicações privada, é uma rede virtual construída sobre uma rede de comunicações pública (como por exemplo, a Internet) em que a conexão é segura e criptografada, que pode ser considerada como um túnel, entre o computador pessoal e um servidor operado pelo serviço VPN, podendo dar acesso a recursos protegidos numa rede interna.

W3C - *World Wide Web Consortium* - é a principal organização de padronização da World Wide Web. Consiste em um consórcio internacional com quase 400 membros, agrega empresas, órgãos governamentais e organizações independentes com a finalidade de estabelecer padrões para a criação e a interpretação de conteúdos para a Web..

WAN - *Wide Area Network* - em português, rede de longa distância ou rede de área alargada é uma rede de computadores que abrange uma grande área geográfica, geralmente um país ou continente.

XMPP - *Extensible Messaging and Presence Protocol* - conhecido anteriormente como *Jabber*, é um protocolo aberto, extensível, baseado em XML, para sistemas de mensagens instantâneas, desenvolvido originalmente para mensagens instantâneas e informação de presença formalizado pelo Internet Engineering Task Force (IETF).

Capítulo 1

Introdução

A presente dissertação descreve o trabalho desenvolvido na unidade curricular Estágio que é parte integrante do MIS no ramo de Desenvolvimento de Software do Departamento de Engenharia Informática e Sistemas (DEIS) do Instituto Superior de Engenharia de Coimbra (ISEC) no ano lectivo 2017/2018. O estágio, com duração de 1200 horas, teve lugar na empresa Dognædis com vista à análise, desenho e implementação do projecto Cat|Mobile. Este projecto visa ser uma solução para a notificação de alertas em dispositivos móveis enviados a partir da infraestrutura local da empresa e que inclui funcionalidades de pré-tratamento de alertas com distinção de operadores, entre outras. Neste capítulo é apresentada a empresa Dognædis como entidade acolhedora e é feito o enquadramento relativamente ao protejo indicando os principais objectivos e, finalmente, a descrição da estrutura do documento.

1.1 Entidade Acolhedora

A Dognædis, entidade acolhedora do estágio, é uma empresa localizada em Coimbra, com escritórios no Reino-Unido, que tem a cibersegurança como foco nas suas actividades. Actualmente, fazendo parte do grupo Prosegur desde 2016 [4], tem vindo a alargar a sua carteira de clientes e o seu leque de serviços. A Dognædis foi fundada em 2010 por um grupo de quatro engenheiros da Universidade de Coimbra: Francisco Rente, Hugo Trovão, Mário Zenha Rela, e Sérgio Alves. Esta mesma equipa, fez durante cinco anos, parte de um Computer Security Incident Response Team (CSIRT) de nome CERT-IPN localizado no Instituto Pedro Nunes (IPN) [5], onde fez grande sucesso e ficou bastante conceituada, antes de a actual Dognædis ser criada, como empresa privada. A Dognædis preza por estar na vanguarda das tecnologias de segurança e por se dedicar ao fornecimento de segurança aos seus clientes através das suas excelentes soluções inovadoras. Actualmente com certificação ISO/IEC 27001 - Information Security Management Systems, a Dognædis está preparada para lidar com informação segura, consolidando um conjunto de práticas orientadas à segurança na manipulação de informação [6].

1.2 Âmbito

O Estágio Curricular teve lugar no departamento de Research And Development (R&D) da Dognædis. O departamento de R&D é responsável por encontrar e desenvolver soluções para apoio aos serviços comercializados pela empresa. A Dognædis dispõe de vários serviços no âmbito da cibersegurança, que proporciona aos seus clientes. Managed Security Services (MSS) é um desses serviços. O serviço de MSS é realizado pelo Security Operation Center (SOC) e consiste em vigilância e monitorização das infraestruturas informáticas dos clientes. O SOC é composto por uma equipa de operadores que efectuam monitorização contínua das redes e sistemas informáticos dos clientes e efectuam resposta a alarmes oriundos de sistemas de alarmística gerados por actividade suspeita detectada nas redes informáticas. No âmbito desse serviço do SOC, os contratos estabelecidos entre os clientes e a Dognædis especificam em que períodos, diariamente, o serviço deve ser prestado, levando eventualmente a casos em que o serviço de resposta a alarmes tenha de estar disponível 24/7 para determinados clientes.

Para viabilizar a disponibilidade contínua do serviço de monitorização e resposta a alarmes, surgiu a necessidade de implementar uma solução que permita dar continuidade ao serviço em qualquer circunstância, isto é, mesmo com a equipa total ou parcialmente fora das instalações físicas da empresa, em mobilidade.

Este relatório tem o propósito de relatar o desenvolvimento da solução para resolver esse problema, atravessando todas as fases, desde a análise até à implementação e validação, com testes.

1.3 Objectivos

Os objectivos gerais podem ser repartidos em duas secções, o estágio e o projecto desenvolvido durante o estágio. Os primeiros correspondem à experiência e conhecimento adquirido durante o estágio na empresa e os segundos ao projecto em si, nomeadamente às suas expectativas e à sua realização.

Estágio - O principal objectivo do estágio na empresa é a aquisição de experiência de carácter sócio-profissional através da familiarização com metodologias e processos usados no ambiente de trabalho assim como o contacto com as ferramentas de desenvolvimento de software usadas na empresa.

Projecto - O objectivo principal do projecto Cat|Mobile é a criação de uma solução de resposta a alarmes para a equipa de SOC da Dognædis poder trabalhar em mobilidade, no exterior, usando os equipamentos e tecnologias que tem disponíveis no inventário de equipamento móvel da empresa. Esta solução, composta por uma aplicação móvel e um serviço de *backend*, permite avisar, em tempo real, os operadores, dos alertas gerados em clientes da

empresa. Anteriormente à existência da solução aqui apresentada, os operadores apenas podiam receber e gerir os alertas quando se encontram presencialmente, junto ao posto de trabalho, localmente na empresa, ou simplesmente serem notificados através de Short Message Service (SMS) quando se encontram em mobilidade, contudo recebendo apenas uma pequena parte da informação. Com a solução Cat|Mobile, os operadores podem receber os alertas, dentro e fora da empresa, através de notificações no smartphone, e gerir o tratamento desses alertas sem necessitar de estar junto do posto de trabalho fixo. Estas notificações facilitam a análise de uma eventual necessidade de tratamento do alerta, mantendo a mobilidade quando se encontram fora da empresa. A solução permite ao operador ver cada alerta apresentado no seu dispositivo móvel e decidir se é necessário interromper a mobilidade em que se encontra para dar resposta a um alerta que necessite de intervenção com recurso ao posto de trabalho fixo. Para analisar o alerta, a solução Cat|Mobile oferece ao operador funcionalidades de pesquisa, que lhe permitem visualizar dados de outros alertas já ocorridos de modo a apoiar na clarificação do alerta e assim decidir se o alerta necessita de uma análise profunda, tendo de ser tratado num posto de trabalho fixo ou se é algo simples que pode ser resolvido apenas com os recursos disponíveis em mobilidade.

Inerente ao objectivo principal da solução de alarmística, pretende-se que o Cat|Mobile funcione também como ferramenta de gestão para que a chefia da equipa de SOC possa designar operadores responsáveis pelo tratamento de alertas a cada dia e seguindo um escalonamento pré definido de operadores disponíveis de forma a que todos os alertas sejam tratados com resiliência e dentro dos prazos previstos para cada cliente. Em suma, a equipa de SOC espera da solução Cat|Mobile, as seguintes funcionalidades principais:

- Envio de alertas para dispositivos móveis;
- Resposta a alertas a partir de dispositivos móveis;
- Escalonamento de operadores para resposta a alertas;
- Intercâmbio de dispositivos móveis;

Outras funcionalidades, esperadas mas menos prioritárias:

- Criação de *tickets* em sistema externo
- Análise estatística das respostas a alertas.

1.4 Contribuição

Com o sistema Cat|Mobile em produção, a equipa SOC fica provida dos meios necessários para oferecer o serviço de alarmística 24/7 com maior comodidade para os operadores, podendo optimizar o tempo de trabalho fora da empresa. Em termos de recursos, a equipa também fica mais folgada pois os operadores vão poder intercambiar livremente os dispositivos móveis disponíveis na empresa. Vão passar a ter muito mais informação quando se encontram fora da

empresa. Passam a conseguir filtrar e pesquisar alertas fora da empresa. Passam a ter capacidade de classificar alertas sem necessitarem de um posto de trabalho fixo. Passa a existir um processo onde é definida a responsabilidade de quem está a resolver cada alerta, facilitando bastante a coordenação da equipa. Em termos administrativos, o gestor de equipa passa a dispor de uma ferramenta que lhe permite agendar turnos de operadores com comodidade e também pode acompanhar o estado dos alertas e detectar eventuais *botlenecks* e pontos de falha durante o desempenho de cada elemento da equipa.

1.5 Estrutura do Documento

Os capítulos da dissertação estão dispostos da seguinte forma:

Introdução - No primeiro capítulo, é apresentado o estágio e a empresa onde este decorre, assim como o departamento onde foi o estagiário foi inserido. São definidos os objectivos, apresentada a contribuição e a estrutura do documento.

Análise de Requisitos - No segundo capítulo são analisadas as funcionalidades esperadas do produto final para a enumeração de requisitos.

Estado da Arte - No terceiro capítulo é feito o estudo do estado da arte onde são comparadas soluções existentes que cumpram alguns dos requisitos enumerados.

Metodologia de Desenvolvimento - No quarto capítulo, é exposta a metodologia empregue no processo de desenvolvimento. Por fim, são apresentadas as ferramentas e tecnologias que foram usadas no projecto.

Planeamento - No quinto capítulo são descritas as fases do projecto, enumerados os riscos e as formas de os mitigar, concluindo com o planeamento.

Arquitectura - O sexto capítulo é sobre a arquitectura, onde é feito o enquadramento e apresentada a arquitectura externa do sistema global. De seguida é apresentada a arquitectura interna do componente servidor e da aplicação móvel, onde são enumerados os módulos a desenvolver para cumprir com os requisitos esperados.

Implementação - No sétimo é descrita a implementação, onde são abordados todos os módulos que compõem a arquitectura do servidor, da aplicação móvel e a forma como se relacionam entre si.

Testes e Qualidade - O oitavo capítulo relata os testes efectuados e a forma como foi validado o produto final do projecto.

Conclusão - Por fim, no nono capítulo, é feita uma reflexão geral sobre o projecto. São enumerados os objectivos atingidos, não atingidos e deixadas sugestões para trabalho futuro.

Capítulo 2

Análise de Requisitos

Para a realização fora da empresa de funções de alarmística da equipa SOC, a solução pretendida deve preencher um conjunto extenso de requisitos levantados a partir das funcionalidades principais esperadas pelas partes interessadas no projecto. Para encontrar esses requisitos, foi necessário conhecer detalhes importantes de funcionamento do serviço de alarmística de modo a colmatar um máximo de funcionalidades até agora impossíveis de realizar fora da empresa, em mobilidade. Tendo em conta que o sistema pretendido é composto por multi-utilizador em acesso concorrencial, foi também necessário deduzir alguns dos requisitos de forma a manter a integridade dos dados e das comunicações durante a utilização do Cat|Mobile. Assim, foi necessário entender que informação deve ser enviada aos operadores em mobilidade para que estes possam analisar cada alerta o melhor possível e quais as operações que estes necessitam de ter disponíveis no dispositivo móvel para dar o máximo de tratamento de alerta possível sem recorrer a outras ferramentas apenas disponíveis num posto de trabalho fixo ou semi-móvel (computador portátil). Assim, foi necessário identificar quais os dados dos alertas que são necessários para que os operadores consigam analisar e iniciar o tratamento de alertas, fora da empresa, apenas com um dispositivo móvel. Para o tratamento dos alertas, foi também necessário identificar quais as operações que necessitam ter na aplicação móvel para poderem fazer um máximo de tratamento possível sem recorrer a outras ferramentas externas. Ferramentas essas mais especializadas, que apenas se encontram disponíveis num posto de trabalho fixo ou semi-móvel (computador portátil).

Para melhor entender o problema e fazer levantamento de requisitos, foram realizadas reuniões com as partes interessadas no Cat|Mobile. Nessas reuniões foram explicadas as circunstâncias em que os alertas são gerados e de que forma estes são tratados pelos operadores da equipa SOC. Os vários requisitos levantados durante as reuniões, serão descritos nas secções seguintes deste capítulo. Entretanto, fruto dessas mesmas reuniões, destaca-se a descrição do processo normal de um alerta, essencial para perceber o âmbito do projecto.

Relativamente ao início do processo de tratamento de alertas, ficou especificado que os alertas a enviar aos operadores teriam como origem, ferramentas informáticas controladas pela

empresa. Estas ferramentas, podem ser aplicações ou simples scripts que fazem parte do sistema de alarmística existente. O sistema de alarmística existente é composto por uma série de componentes que tratam a informação através de um pipeline de aplicações, desde o momento em que é detectada uma anomalia ou actividade suspeita durante o processo de monitorização do sistema informático de um cliente, até ao momento de ser gerado um alerta a ser tratado por um operador. Assim, um evento gerado durante a monitorização, antes de se tornar em alerta, passa por ferramentas de diagnóstico e de correlação que efectuam pré-processamento automático visando filtrar o máximo possível de falsos-alertas e, se possível, adicionar aos alertas informação complementar para a investigação manual a realizar pelos operadores do SOC durante o tratamento. Assim, é na da entrega dos alertas aos operadores, que se inicia o âmbito da solução a implementar e que inspirou o nome do projecto Cat|Mobile, que tem por base o comando de catalogar "cat" e a função canalizadora "pipe" que encaminha o alerta para um dispositivo móvel. De salientar, que toda a análise automática a realizar sobre os alertas terá de estar feita antes de chegar ao Cat|Mobile pois a solução pretendida apenas deverá fornecer os meios necessários para o operador realizar investigação superficial, tratamento rápido de alertas que não necessitem de investigação profunda ou simplesmente reconhecer o alerta e informar a equipa SOC do início de tratamento. Por investigação superficial, entenda-se, pesquisa e consulta de outros alertas ocorridos anteriormente. Investigação profunda refere-se à utilização de meios e ferramentas apenas disponíveis em posto de trabalho fixo ou semi-móvel. O tratamento rápido pode consistir na conclusão do alerta por resolução imediata decidida pelo operador, e cujo motivo ficará registado na resposta ao alerta. A opção de reconhecimento serve simplesmente para que a equipa SOC tenha conhecimento de que o alerta está a ser tratado pelo operador, pois caso contrário teria de passar para outro operador disponível de modo a que cada alerta não fique por tratar e possa cumprir o prazo previsto de resolução.

Para a gestão de multi-utilizador concorrencial será necessário desenvolver uma solução que permita registar os actos de cada utilizador no sistema, sejam estes espontâneos ou deliberados. Isto é, fazer o registo de actos ou confirmações que permitam não repúdio do utilizador. Assim como também conhecer inequivocamente o modo de contacto instantâneo de cada operador para que as comunicações sejam sempre dirigidas à pessoa pretendida através do seu dispositivo móvel e não dirigidas a outra pessoa com quem poderia ter sido partilhado o dispositivo móvel depois de este ter sido usado por outra pessoa.

Na abordagem inicial de requisitos, cedo se percebeu que para implementar a mobilidade esperada da solução, esta teria de ser composta por dois componentes que se complementariam num sistema, tal como representado na Figura 2.1. Um componente móvel algures na rede pública Wide Area Network (WAN) e um componente fixo de apoio localizado na rede Local Area Network (LAN) da empresa. A componente móvel, uma aplicação para dispositivos com Android, por ser esse o tipo de equipamento disponível na equipa SOC. O componente fixo, um servidor localizado na empresa, ao alcance das ferramentas de emissão de alertas, para receber os alertas e intermediá-los com a aplicação móvel através de uma Application Programming

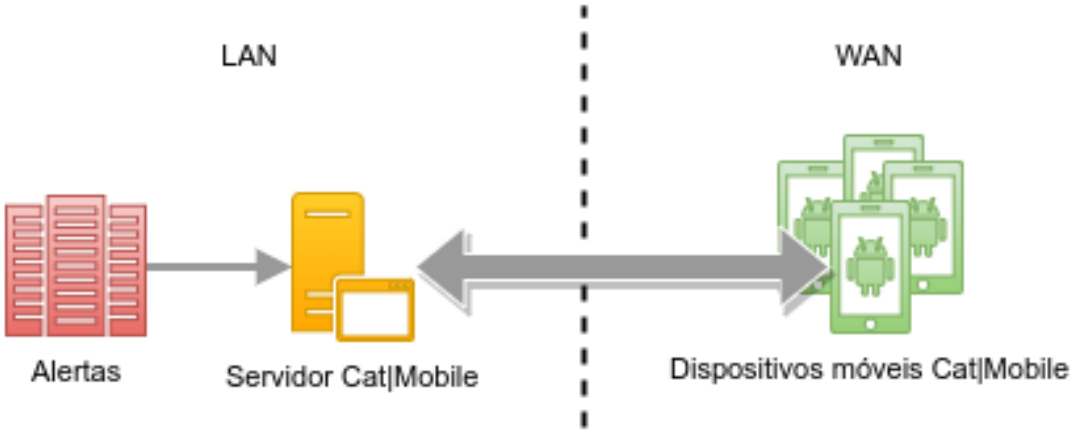


Figura 2.1: Representação inicial do Sistema.

Interface (API). Havendo esta separação física entre componentes, os requisitos da solução podem ser repartidos entre eles, facilitando a sua distinção nas fases seguintes do projecto.

A cada requisito está atribuída uma classificação para distinguir o nível de prioridade a dar a esse requisito durante a fase de implementação. Esses níveis de classificação têm o seguinte significado:

- **Must have** - Requisito de implementação imperativa pois dele dependem as funcionalidades principais esperadas da solução.
- **Should have** - Requisito importante em termos de expansibilidade mas não necessário para cumprir com as funcionalidades principais esperadas da solução.
- **Nice to have** - Requisito que faz parte de um conjunto não fechado de funcionalidades interessantes e possíveis de implementar, estando satisfeitos os outros níveis de classificação.

2.1 Requisitos Funcionais

Em engenharia de software, um requisito funcional define uma função de um sistema de software ou seu componente. O requisito funcional representa o que o software faz, em termos de tarefas e serviços [7]. No caso do Cat|Mobile, os requisitos estão repartidos pelos dois principais componentes, o Servidor e a Aplicação Móvel.

2.1.1 Requisitos Funcionais do Servidor

Na Tabela 2.1 estão representados os Requisitos do Servidor com a respectiva classificação de prioridade.
De seguida são detalhados cada um dos Requisitos Funcionais do Servidor.

Tabela 2.1: Requisitos Funcionais do Servidor

Requisitos Funcionais do Servidor	**Classificação**
RF01S - Recepção de Alertas Externos	Must have
RF02S - Fornecimento de Dados à Aplicação Móvel	Must have
RF03S - Notificações Resilientes	Must have
RF04S - Turnos de Operadores	Must have
RF05S - Escalonamento *OnCall*	Must have
RF06S - Backoffice de Administração	Must have
RF07S - Intercâmbio de Dispositivos Móveis	Should have
RF08S - Seguimento das Notificações	Should have
RF09S - Timeline do Tratamento de Alertas	Should have
RF10S - Dashboards	Should have
RF11S - Estatísticas	Nice to have

Recepção de Alertas Externos No Cat|Mobile, os alertas, são os dados que se pretendem transmitir aos operadores. A função do sistema é intermediar esses dados que estão a ser gerados por ferramentas externas e encaminhá-los para os dispositivos móveis dos operadores. A recepção dos alertas vindos das origens externas é, por esse motivo, crucial. As origens dos alertas, aliás as ferramentas que os geram, têm formas próprias de os disponibilizar. Existem ferramentas que podem executar scripts com base em eventos e outras ferramentas que apenas disponibilizam dados através de uma API. A forma de obter os alertas é diferente nestes dois tipos de ferramentas. As primeiras têm capacidade de enviar os alertas para o Cat|Mobile, enquanto que as segundas necessitam que o Cat|Mobile consulte-as para obter os alertas.

Fornecimento de Dados à Aplicação Móvel Sendo o Cat|Mobile composto por um servidor de apoio a uma aplicação móvel, é imprescindível a existência de uma API que permita essa comunicação entre os dois componentes. É através desta API que a aplicação móvel poderá comunicar com o servidor. De facto, todas as funcionalidades, que requerem comunicação entre os dois componentes, dependem desta API.

Notificações Resilientes Para os alertas, transmitidos pelo Cat|Mobile, chegarem aos dispositivos móveis dos operadores, é necessário fornecer um meio de transporte e um sistema que assegure a entrega e verifique a recepção dos dados. Cada alerta terá pelo menos uma notificação associada. Por notificação entende-se que é uma tentativa de fazer chegar um

alerta a um operador. Caso a notificação não seja recebida ou não seja vista pelo operador destinatário, dentro de um determinado prazo, então o sistema deve tentar uma nova notificação com outro operador predefinido.

Turnos de Operadores Para receber as notificações e tratar os alertas, devem existir operadores disponíveis. De modo a que se possam atribuir responsabilidades no tratamento de alertas, prevê-se que haja a cada dia, um operador responsável ou, usando o termo próprio da equipa de SOC, o operador que está *OnCall*. Para gerir os turnos de operadores que vão estar Oncall a cada dia, o Cat|Mobile deve ter a capacidade de inscrever operadores num calendário. Calendário este que será consultado no momento em que o sistema terá de enviar uma notificação para entregar o alerta ao operador responsável no momento.

Escalonamento *OnCall* Cada alerta tem de ser tratado imperativamente dentro de um prazo estabelecido em contrato com o cliente. Por isso e não excluindo a hipótese de que o operador responsável pode não estar disponível para o tratamento de um alerta em dado momento, tem de existir um sistema que atribua a responsabilidade a outro operador, seguindo uma escalada de responsabilidade pré-definida. Este sistema de escalonamento será consultado, pelo sistema de notificação de alertas, para determinar qual o operador seguinte a quem terá de enviar uma nova notificação, quando uma determinada notificação não atinja um estado esperado de reconhecimento dentro de um prazo pré-estabelecido.

Backoffice de Administração Para configurar as diversas variáveis de que o Cat|Mobile necessita e para gerir os dados das diversas entidades armazenadas, deve existir um Graphical User Interface (GUI) que permita a um administrador ter acesso ao Cat|Mobile. Através desta GUI, o gestor da equipa de SOC irá poder registar todos os dados usados pelo Cat|Mobile, com excepção dos alertas, pois esses são fornecidos por aplicações externas. Os dados geridos através da GUI são relativos aos operadores, turnos e origens de alertas. Relativamente aos operadores e aos turnos, estes são constituídos pelos membros da equipa de SOC, com respectivo registo de turnos em calendário e os escalonamentos inerentes aos turnos. Relativamente às aplicações externas ou seja as origens de alertas, estas devem estar registadas para identificar a origem de cada alerta, ou seja, a ferramenta ou sistema externo que deu origem ao alerta. Para além de campos de identificação e autenticação, cada origem deverá ter definição para períodos de inactividade diários e semanais. As notificações de alerta devem ser omitidas se os alertas oriundos da origem tiverem sido emitidos dentro do período de inactividade pré-definido.

Intercâmbio de Dispositivos Móveis A possibilidade de a equipa de SOC poder usar dispositivos móveis partilhados entre operadores, é algo bastante importante para o funcionamento da equipa e para a gestão de recursos de *hardware* da empresa.

Seguimento de Notificações O seguimento de notificações é uma expansão do requisito de

notificações resilientes, onde se pretende fazer registo de todas as alterações de estado das notificações sob a forma de timeline para ter esses dados disponíveis em análises posteriores.

Timeline no Tratamento de Alertas Para futura análise estatística de resultados, é necessário registar todos os eventos relativos ao tratamento dos alertas.

Dashboard Na GUI de administração seria interessante poder visualizar na página inicial, uma dashboard com alguns gráficos de resumo de actividade do Cat|Mobile, nomeadamente, quantidade de alertas recebidos ao longo do tempo, por cliente, etc.

Estatísticas Para efeitos de análise estatística por parte do administrador como gestor de equipa, seria interessante obter leituras de valores estatísticos de determinadas métricas por períodos de tempo, tais como, tempo médio de resposta a alerta, tempo máximo, mínimo, etc.

2.1.2 Requisitos Funcionais da Aplicação Móvel

Na Tabela 2.2 estão representados os Requisitos Funcionais da Aplicação Móvel com a respectiva classificação de prioridade.

Tabela 2.2: Requisitos Funcionais da Aplicação Móvel

Requisitos Funcionais da Aplicação Móvel	**Classificação**
RF01AM - Tratamento de Notificações	Must have
RF02AM - Tratamento de Alertas	Must have
RF03AM - Autenticação	Must have
RF04AM - Criação de *tickets*	Should have
RF05AM - Dashboard	Nice to have

De seguida os detalhes de cada um dos Requisitos Funcionais da Aplicação Móvel.

Tratamento de Notificações A aplicação móvel terá de ser capaz de receber as notificações que são enviadas ao operador. Para além da capacidade de recepção, a aplicação deverá poder enviar confirmação de recepção quando a notificação chega ao dispositivo e também quando a notificação é visualizada pelo operador. Cada um destes estados por onde passa a notificação, deve ser transmitido ao servidor de forma transparente para o utilizador, sem necessitar da intervenção dele, para o servidor ter conhecimento do estado em que se encontra a notificação e poder decidir sobre a necessidade de reiniciar a

notificação do alerta com outro operador. Com o registo destes eventos no servidor, é possível ao gestor conhecer os factos sobre as notificações enviadas e, através do requisito de Seguimento de Notificações, obter o "não repúdio"dos operadores responsáveis no momento.

Tratamento de Alertas Relativamente aos alertas enviados através das notificações, é também necessário enviar feedback ao servidor sobre as interacções directas ou indirectas com estes. Inicialmente, quando o alerta é visualizado, é enviada a confirmação de visualização. Nesta altura, o prazo de resposta ao alerta ainda se encontra a escoar e continuará nesse estado até o operador ser explícito quanto ao início do tratamento do alerta. Para isso, no momento da visualização do alerta, o operador deve ter opção de iniciar o tratamento ou de renunciá-lo. Conforme a resposta dada pelo operador, ou da ausência dessa resposta, o servidor decide se nomeia outro operador responsável pelo alerta e desencadeia outra notificação ou se dá por concluído o processo inicial de entrega de alerta e inicia o processo seguinte de tratamento de alerta.

Autenticação A autenticação do operador na aplicação móvel é um requisito crucial da aplicação pois, para além de restringir o acesso aos dados, é o que permite identificar o operador perante o servidor do Cat|Mobile e que vai permitir contactá-lo sempre que for necessário enviar uma notificação dirigida a ele.

Criação de *tickets* Os *tickets* são registos de incidentes relativos a alertas que são criados no RTIR [2], a ferramenta externa para gestão de *tickets* usada na empresa. A opção de criação de *ticket* deve estar disponível durante o tratamento do alerta. Uma vez activada esta opção, a resposta dada ao alerta, durante o seu tratamento, é usada simultaneamente para responder ao alerta e para criar um *ticket* no RTIR.

Dashboard Uma dashboard disponível na aplicação é algo interessante para o operador obter informação geral acerca do sistema ou de estatísticas acerca do trabalho que tem realizado. Este requisito não é fundamental ao sistema mas pode-se considerar trivial de implementar e bastante expansível, uma vez implementados os requisitos relativos ao registo em formato de timeline de dados de notificações e de alertas.

2.2 Requisitos Não Funcionais

Requisitos não-funcionais são os requisitos relacionados com o uso da aplicação em termos de desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenção e tecnologias envolvidas [7]. Nesta secção são apresentados os requisitos não funcionais para o servidor e para a aplicação móvel.

2.2.1 Requisitos Não Funcionais do Servidor

Na Tabela 2.3 estão enumerados os requisitos não funcionais do servidor, classificados por prioridade. De seguida, são detalhados os requisitos.

Tabela 2.3: Requisitos Não Funcionais do Servidor

Requisitos Não Funcionais do Servidor	**Classificação**
RNF01S - Persistência dos dados	Must have
RNF02S - Resiliência nas notificações	Must have
RNF03S - Segurança nas comunicações	Must have
RNF04S - Disponibilidade	Should have

Persistência dos dados Todos os dados usados pelo Cat|Mobile no servidor e criados através dos requisitos funcionais, devem ser persistidos em suporte físico e possíveis de usar mesmo que o hardware seja reiniciado.

Resiliência nas notificações Os alertas devem imperativamente chegar a um operador responsável. Para isso, as notificações devem ser transmitidas insistentemente no sentido de tentar chegar a um operador disponível.

Segurança nas comunicações As comunicações através dos diversos pontos de entrada e saída do Cat|Mobile devem estar protegidas de modo a assegurar a segurança da informação. Isto é, nomeadamente em termos de confidencialidade, integridade e autenticidade.

Disponibilidade Todos os serviços do sistema devem manter a sua disponibilidade durante a utilização normal do sistema.

2.2.2 Requisitos Não Funcionais da Aplicação Móvel

Na Tabela 2.4 estão os requisitos não funcionais da aplicação móvel, classificados por prioridade. De seguida, a descrição em detalhe dos requisitos.

Não persistência dos dados A persistência de dados na aplicação móvel deve ser inexistente. Todos os dados para o funcionamento da aplicação devem ser obtidos do servidor no momento em que são necessários e nunca devem ser guardados na memória não volátil do dispositivo. Este requisito é imposição directa das partes interessadas no Cat|Mobile.

Segurança Todas as comunicações com dados confidenciais devem passar por canal seguro. Os dados que tenham de passar por canal não seguro, devem ser não confidenciais.

Tabela 2.4: Requisitos Não Funcionais da Aplicação Móvel

Requisitos Não Funcionais da Aplicação Móvel	**Classificação**
RNF01AM - Não persistência dos dados	Must have
RNF02AM - Segurança	Must have
RNF03AM - Usabilidade	Should have
RNF04AM - Poupança de bateria	Should have
RNF05AM - Poupança de dados	Nice to have

Usabilidade O interface de utilização deve ser agradável e intuitivo.

Poupança de bateria Dado que a bateria é um recurso precioso do dispositivo móvel. Sempre que possível, devem ser usadas opções que permitam maior poupança.

Poupança de dados Dado que os dados em dispositivos móveis têm, na maioria das situações, os seus custos contabilizados, pretende-se que se opte por soluções que usem a rede de dados, o menos possível.

2.3 Conclusão

Neste capítulo foram enumerados os requisitos do projecto, com base nos objectivos indicados pelas partes interessadas. As indicações base deram origem aos requisitos necessários no desenvolvimento para atingir os objectivos. Requisitos esses que serão refinados em tarefas durante as etapas seguintes do projecto.

Capítulo 3

Estado da Arte

O Estado da Arte relativo ao projecto Cat|Mobile tem a sua análise subdividida em dois aspectos principais. A gestão dos alertas é o aspecto relativo aos dados que a solução pretende gerir. O envio de notificações móveis é outro aspecto que o autor pretende ter analisado.

3.1 Tecnologia e ferramentas para gestão de alertas

Dentro do âmbito da solução pretendida, foi feita uma pesquisa de eventuais produtos, já existentes no mercado, que fossem ao encontro dos requisitos iniciais do Cat|Mobile. Previamente ao Cat|Mobile, era usado na empresa um sistema baseado em SMS para notificar operadores em mobilidade dos alertas a ser gerados. No entanto, o sistema de SMS resolve apenas o problema de notificar o operador dos alertas. Um sistema de SMS por GSM, tem funcionalidade de recibos de entrega, mas o recibo de entrega é apenas do dispositivo e não garante que o utilizador recebeu, leu e compreendeu realmente o alerta. Num cenário em que o operador responsável abandona momentaneamente o telemóvel e nesse curto espaço de tempo recebe um alerta por SMS, mas logo a seguir, o telemóvel se desliga por falta de bateria, não existe garantia de que o operador tenha lido o alerta, apesar de nos sistemas haver indicação de SMS entregue. Por e-mail, é possível fazer chegar notificações de alerta e fazer gestão dessa informação. No entanto existe uma falha de segurança ao manter o conteúdo dos e-mails armazenados localmente no dispositivo móvel.
Na tabela 3.1 é apresentada uma comparação de funcionalidades entre o tratamento de alertas local, na empresa, e algumas formas de tentar obter essas mesmas funcionalidades com tecnologias comuns, nomeadamente protocolos de e-mail e o SMS. As funcionalidades usadas como critério de comparação são as seguintes:

- **Notificações**: Capacidade de enviar mensagens instantâneas dirigidas a uma ou mais pessoas de um grupo.
- **Gestão**: Capacidade de organizar os alertas, podendo fazer pesquisas, filtros, e responder para dar seguimento aos alertas.

- **Escalonamento**: Capacidade de alterar os destinatários responsáveis pela resposta a alertas conforme uma hierarquia de responsabilidade pré definida em calendário.
- **Não repúdio**: Capacidade de garantir que determinada pessoa tomou conhecimento, não o podendo negar.
- **Seguro**: Informação transmitida de forma segura entre a origem e destino. Dados não armazenados localmente. Não passagem de dados sensíveis (não encriptados) por terceiros.
- **Móvel**: Capacidade de ser transportável e usável mesmo de pé, sem estar sentado.
- **Agregação**: Capacidade de juntar alertas ou notificações, por características e poder fazer análise estatística sobre os dados agregados

Tabela 3.1: Comparação de tecnologias para a gestão de alertas

	Local	**POP3**	**IMAP**	**SMS**	**Cat|Mobile**
Notificações	✗	✓	✓	✓	✓
Gestão	✓	✓	✓	✗	✓
Escalonamento	✓	✗	✗	✗	✓
Não repúdio	✓	✓	✓	✗	✓
Seguro	✓	✗	✗	✗	✓
Móvel	✗	✓	✓	✓	✓
Agregação	✓	✗	✗	✗	✓

Relativamente a ferramentas orientadas para objectivos semelhantes, foi feita uma pesquisa de produtos e elaborado um quadro de comparação. A solução pretendida com o Cat|Mobile tem algumas semelhanças com outros sistemas, geralmente conhecidos por gestores de *tickets* ou aplicações de suporte Helpdesk. Alguns destes sistemas existem na forma de aplicações livres e Open-Source, nomeadamente para a gestão de incidentes. No entanto o foco principal, da solução pretendida, não é a gestão a incidentes, mas sim fazer proxy de alertas através de notificações para uma aplicação móvel que permita a interacção do utilizador para posterior actualização de estado de *tickets* já existentes ou criar *tickets* novos se um alerta o justificar. Neste caso, um eventual software de gestão de incidentes assume o papel de origem de alertas no sistema Cat|Mobile, que por sua vez ficará encarregue de fazer notificações ao operador responsável pela resolução, através da aplicação móvel. Neste âmbito, foram analisados alguns produtos existentes no mercado.

O Request Tracker for Incident Response (RTIR) [2] é um projecto *open source* de um sistema para tratamento de incidentes, orientado para equipas de segurança informática. É usado por todo mundo em dezenas de Computer Emergency Readiness Team (CERT) e CSIRT e é usado também na Dognædis. De facto este sistema é um dos componentes que estará integrado com o Cat|Mobile. Como se pode ver na Figura 3.1, o RTIR tem um workflow desenhado especificamente para reportar e investigar incidentes. Para se poder usar, dispõe de interface Web com níveis de permissão por grupo de utilizadores e de API para realização remota de tarefas.

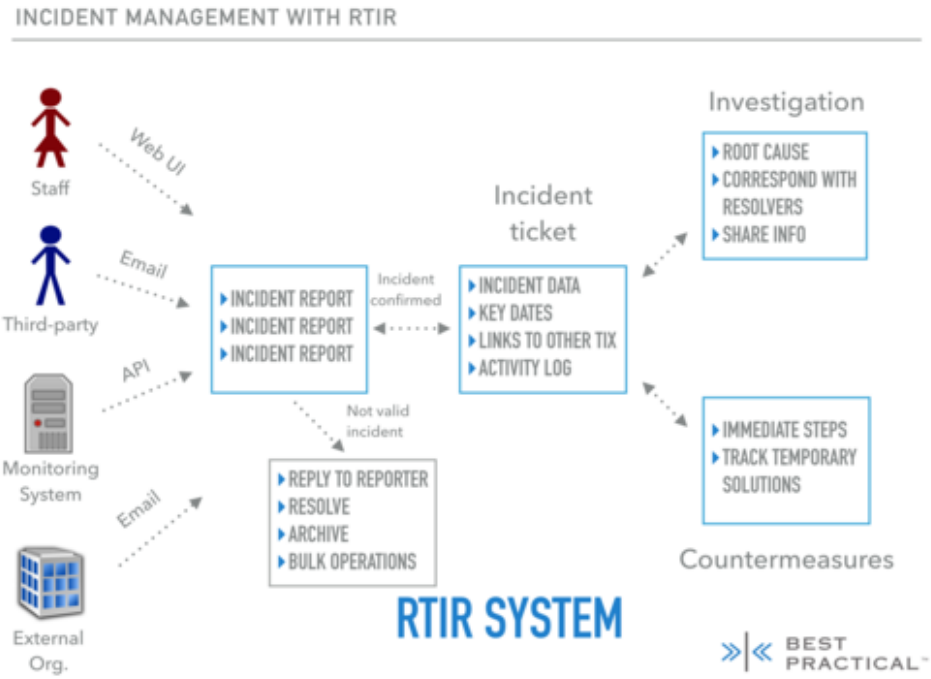


Figura 3.1: Workflow da gestão de incidentes no RTIR.

O Freshdesk [8] é uma solução de software baseado na *Cloud* para apoio no serviço ao cliente. Dispõe de um grande número de funcionalidades automáticas para agilizar procedimentos. Com diversos módulos orientados para departamentos específicos das organizações, como por exemplo para o departamento de Recursos Humanos [8].

O ngDesk [9] é um serviço de gestão de tickets inteiramente gratuito. Está disponível apenas na *Cloud* em servidores próprios da empresa que o mantém. Não tem o código fonte disponível nem é instalável em servidores privados. Apesar de ter todas as suas funcionalidades disponíveis em interface web (tanto de administração como de utilização operacional), dispõe também de uma aplicação móvel que permite aos operadores interagir com a aplicação [9].

O C-Desk [10] é uma ferramenta completa para organizações gerirem os pedidos aos seus serviços, não só no departamento de Information Technology (IT) mas também em qualquer outro departamento da organização. Assim, conforme se pode ver na Figura 3.2, o C-Desk não é apenas uma aplicação de *Helpdesk* para IT, mas um completo sistema de *Helpdesk* para toda a organização. [11].

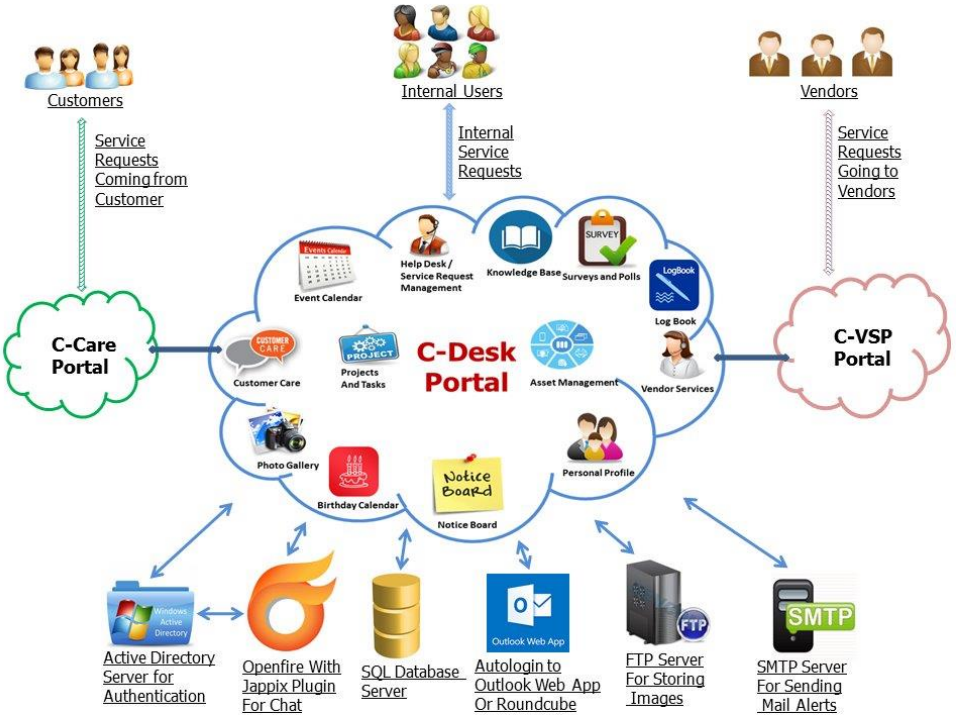


Figura 3.2: Diagrama de Arquitectura do C-Desk.

Na tabela 3.2 pode ser vista a comparação entre produtos analisados e o Cat|Mobile usando como critérios as seguintes funcionalidades:

- ***Tickets*** para capacidade de gestão de dados, alteração de estado, atribuição de colaborador, etc.
- **API server** para a capacidade de receber dados através de pedidos HTTP como forma de gerar de alertas no sistema.
- **API client** para gerar alertas no sistema através de consulta a API externa.
- **Móvel** para a existência de uma aplicação móvel disponível para interacção com o sistema.
- **Segurança** como garantia de que as transacções de dados são seguras e de que não existem dados sensíveis a passar por sistemas de terceiros.
- **OnCall** representa a capacidade de agendamento de colaboradores operacionais por turnos e com escalonamento de responsabilidades.

- **Servidor** representa a capacidade de o sistema poder ser instalado em servidor próprio e não estar restrito a um serviço Cloud externo à empresa.
- **Grátis** significa que o produto não traz custos acrescidos à empresa, para além dos custos de desenvolvimento e manutenção do sistema.

Tabela 3.2: Comparação de produtos para a gestão de alertas

	RTIR	**C-Desk**	**Freshdesk**	**ngDesk**	**Cat|Mobile**
Tickets	✓	✓	✓	✓	✓
API server	✓	✓	✗	✗	✓
API client	✗	✓	✗	✗	✓
Móvel	✗	✓	✓	✓	✓
Segurança	✓	✗	✗	✗	✓
OnCall	✗	✗	✗	✓	✓
Servidor	✓	✗	✗	✗	✓
Grátis	✓	✓	✗	✓	✓

3.1.1 Conclusão

Todos estes produtos fazem parte do mercado das aplicações de gestão de incidentes. Os produtos deste tipo implementam algumas das funcionalidades pretendidas na solução, permitindo, por exemplo, o uso de aplicações móveis que interagem com os dados do produto em servidor. Estes produtos, para além de serem pagos, consistem, na maioria, em serviços Software as a Service (SaaS) na Cloud, não havendo modalidade de instalação local. No caso da empresa Dognædis, a segurança da informação está sempre em primeiro plano, por isso, os dados não devem poder circular livremente por serviços de terceiros, sobretudo quando se tratam de dados de clientes com os quais se têm acordos de confidencialidade. Por esse motivo, tem de ser descartado logo à partida, qualquer produto ou serviço, com o qual tenha de haver partilha de informação. Não obstante esta restrição, esses produtos são interessantes do ponto de vista de desenvolvimento do Cat|Mobile, pois podem ser usados como fonte de conhecimento, tirando partido do vasto desenvolvimento, para inspirar funcionalidades relevantes que possam vir a ser implementadas neste projecto.

3.2 Tecnologia de envio de notificações

As comunicações entre o servidor e os dispositivos móveis são um requisito que necessita de escolha no tipo de tecnologia a usar. Para fazer essa escolha, foi feito um estudo do estado da arte relativamente às opções existentes para este tipo de requisito. De entre as várias tecnologias encontradas, foram escolhidas as que mais se enquadram no sistema a implementar.

As aplicações Web foram inicialmente desenvolvidas em torno de um modelo cliente-servidor, onde o cliente é sempre quem inicia as transacções, com pedidos ao servidor. Assim, não existia um mecanismo que permitisse ao servidor enviar dados ao cliente sem que este tivesse feito um pedido. Para resolver este problema, foi desenvolvida a técnica de Long Polling, que simula indisponibilidade do servidor para manter aberta uma ligação Hypertext Transfer Protocol (HTTP). A técnica é iniciada com o cliente, que faz um pedido inicial ao servidor e o servidor mantém o pedido em espera até ter dados disponíveis. Logo que o servidor tiver os dados prontos, envia a resposta do pedido inicial. Assim que o cliente recebe a resposta, imediatamente inicia um novo pedido e o ciclo repete-se. Esta solução, representada pela Figura 3.3 de facto, emula pedidos com iniciativa no servidor. Contudo, é uma solução com vários pontos de falha, nomeadamente quando o cliente interrompe a ligação momentaneamente, por exemplo por passar de WiFi para 4G causando interrupção do pedido, levando o servidor a sofrer eventuais respostas perdidas [12].

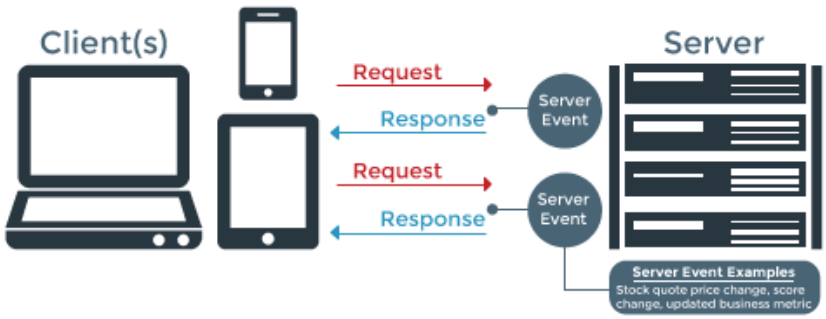


Figura 3.3: Diagrama de HTTP Long Polling.

Surgida após Long Polling como solução para pedidos com origem no servidor em modelos cliente-servidor, *WebSockets* é uma tecnologia bastante mais evoluída e fiável que tira partido do conceito de *sockets* para Web. A técnica consiste em permitir comunicação bidireccional por canais *full-duplex* sobre um único *socket*. Em informática, um *socket*, é um canal aberto a transacções. Conforme representado na Figura 3.4, o que *WebSockets* faz é manter um *socket* Transmission Control Protocol (TCP) aberto no cliente Web, para onde o servidor pode enviar dados sempre que necessário. *WebSockets* é, de facto, um protocolo e uma API que têm vindo a ser normalizadas pelo IETF e World Wide Web Consortium (W3C) [13]. Actualmente,

WebSockets é tido como o estado da arte em termos de aplicações *real-time* para Web, contudo também tem as suas limitações. No caso de uso para comunicações com uma aplicação móvel, uma ligação permanente de *WebSocket* pode impedir que o dispositivo móvel entre em modo de poupança energética e mesmo que o aparelho consiga entrar nesso modo, a ligação seria interrompida, não sendo portanto, a tecnologia mais indicada para uso permanente de aplicações *real-time* em dispositivos móveis.

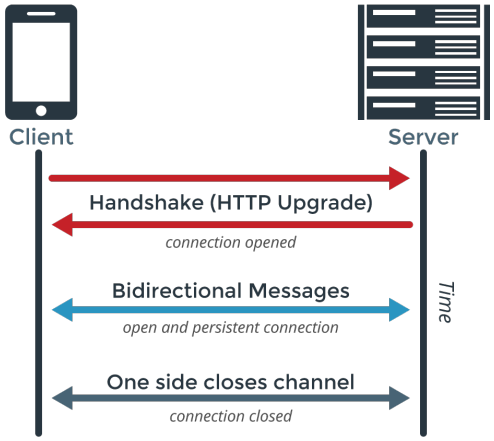


Figura 3.4: Diagrama de WebSockets.

Firebase Cloud Messaging (FCM) é uma plataforma pertencente à Google, conhecida anteriormente por Google Cloud Messaging (GCM). Esta plataforma permite uma ligação robusta e eficiente em termos energéticos, entre um servidor e dispositivos móveis ou Web. Permite enviar mensagens direccionadas a dispositivos específicos ou a dispositivos agrupados [14].

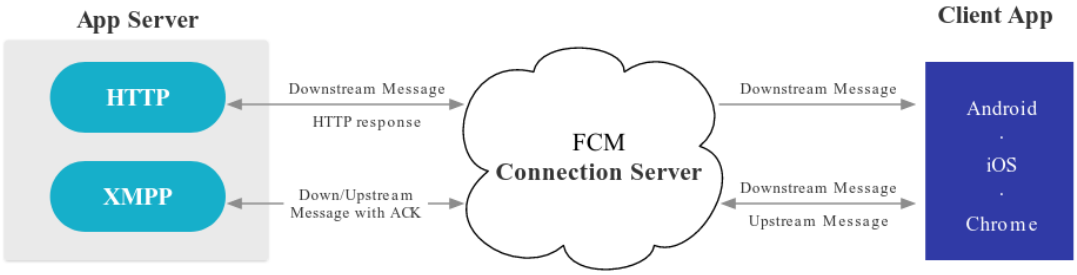


Figura 3.5: Diagrama de Firebase Cloud Messaging.

Conforme representado, na Figura 3.5, o FCM funciona como ponto intermédio de distribuição de mensagens, em que o servidor envia as mensagens para o FCM, com um ou mais clientes como destinatários e o FCM se encarrega de entregar as mensagens nesses clientes. No caso, de clientes em dispositivos Android, é necessário que as aplicações Google Play ou Google Services estejam instaladas e a funcionar. De facto a tecnologia por trás de FCM é o

protocolo Extensible Messaging and Presence Protocol (XMPP) usado como meio de transporte para o envio de mensagens instantâneas em dispositivos móveis [15].

Na tabela 3.3 está representada uma comparação entre as tecnologias de mensagens instantâneas. Para comparar as tecnologias, foram definidos alguns factores importantes na decisão, aos quais se atribuíram pesos de importância. A maior soma de pesos dos factores de decisão, dita qual a tecnologia escolhida para as mensagens instantâneas.

- **Terceiros** - Define se depende das infraestruturas de terceiros para poder funcionar;
- **VPN** - Define se necessita de VPN permanente para funcionar;
- **Texto** - Define se a notificação pode conter texto livre e legível;
- **App ini** - Define se a aplicação móvel tem de estar inicializada no dispositivo;
- **Push** - Define se o envio das notificações é assíncrono com iniciativa do servidor (*push*) ou se é síncrono, com iniciativa da aplicação móvel (Pull).

Relativamente à conotação positiva ou negativa dada a cada factor de importância:

- **Terceiros** ✓= Não depende de terceiros ✗= Depende de terceiros ;
- **VPN** ✓= Não necessita de VPN ✗= Necessita de VPN;
- **Texto** ✓= Pode enviar texto livre ✗= Não pode enviar texto livre;
- **App ini** ✓= Não necessita de aplicação iniciada ✗= Necessita de aplicação iniciada;
- ***Push*** ✓= Mensagens assíncronas ✗= Mensagens síncronas.

Tabela 3.3: Comparação de tecnologias para mensagens instantâneas

	Terceiros	**VPN**	**Texto**	**App ini**	**Push**	**Total**
	5	10	10	10	5	pontos
Long Polling	✓	✗	✓	✗	✗	15
WebSockets	✓	✗	✓	✗	✓	20
FCM	✗	✓	✗	✓	✓	25

3.2.1 Conclusão

A escolha da tecnologia para envio de mensagens ficou determinada pela Tabela 3.3. O Firebase Cloud Messaging (FCM) foi a tecnologia de mensagens instantâneas sugerida às partes interessadas que deram a aprovação do seu uso, com a condição de que não haja passagem de informação confidencial por essa plataforma da Google.

Capítulo 4

Metodologia

Neste capítulo é descrita a metodologia usada durante o desenvolvimento do projecto e enumeradas as ferramentas e tecnologias adoptadas para a realização do projecto.

4.1 Preparação

Previamente ao inicio do desenvolvimento do projecto, houve um primeiro contacto com as ferramentas de implementação a usar durante o desenvolvimento, com o objectivo de obter as bases de conhecimento necessárias ao desempenho das tarefas que iriam surgir. Ainda durante essa fase de pré-desenvolvimento, teve lugar uma reunião com as partes interessadas no projecto. Nessa reunião foi esclarecido o problema que se pretende resolver e transmitida a visão das partes interessadas, sobre o produto que esperam obter. Foram indicadas as funcionalidades que são mais prioritárias e as que são menos prioritárias, em termos de funcionalidades possíveis de implementar durante o tempo disponível para o desenvolvimento do projecto. Nessa reunião, o autor teve oportunidade de colocar questões sobre a visão explanada, de fazer troca de impressões, apresentar sugestões e delinear uma previsão de objectivos possíveis dentro do tempo disponível. Tendo como objectivos iniciais do projecto, os referidos no Capítulo 1, a fase seguinte foi a realização da análise de requisitos com intuito de refinar os objectivos em requisitos implementáveis.

4.2 Processo de Desenvolvimento

Com a lista de requisitos pronta, foi desenhada uma arquitectura da solução. Durante o desenho, o autor apercebeu-se que a visão inicial, apesar de objectiva, deu origem a requisitos que envolviam bastante complexidade. A complexidade desses requisitos iria certamente originar dependências de novos requisitos com maior nível de objectividade. Estes novos requisitos poderiam ser descobertos em sessões de *brainstorming* em que vários analistas poderiam prever um conjunto de situações que revelaria esses novos requisitos. Contudo, sendo a equipa de

desenvolvimento constituída por um único elemento, o autor do trabalho, seria impossível realizar eventuais sessões de *brainstorming*. Então a solução alternativa encontrada passaria por descobrir os novos requisitos à medida que fossem feitos testes aos requisitos já implementados. O comportamento dos testes durante a implementação iria revelar as lacunas, sobretudo de arquitectura, mas também da implementação e eventualmente impor novos requisitos de modo a colmatar essas falhas. Por este motivo, a partir da fase de desenho e arquitectura inicial, foi colocado em prática um método com base no desenvolvimento iterativo e incremental. Este método não teria como objectivo, a criação de um produto de forma incremental por etapas, mas sim, a descoberta incremental de requisitos de baixo nível necessários à implementação dos requisitos iniciais de mais alto nível. Desta forma foi possível desenvolver o produto sem demasiadas intervenções das partes interessadas, pois conhecendo a visão que estas tinham à partida do produto, foi possível, com este método, deduzir bastantes novos requisitos que se revelaram pertinentes ou logicamente necessários para satisfazer os requisitos iniciais. Assim, o método de desenvolvimento representado na Figura 4.1, é composto por três fases principais. Inicialmente, pela fase de pré-desenvolvimento de preparação onde foi feita a análise de requisitos inicial e o desenho inicial da arquitectura, seguida do ciclo de desenvolvimento com várias iterações com duração de uma semana, onde os requisitos são implementados, testados e refinados até à conclusão da implementação de todos os requisitos ou até à aproximação do final do tempo disponível. Por fim, na última fase, após o desenvolvimento, o sistema é instalado em ambiente de produção onde são realizados testes finais de validação antes do sistema ser colocado em produção para ser usado em cenário real de utilização. Devido à limitação do tempo disponível para a implementação do projecto, todo o processo de desenvolvimento é realizado em modo ágil, ou seja, com foco na produtividade e não tanto em documentação ou artefactos que documentem o processo. De facto todos os novos requisitos que surgiram durante o processo, foram acrescentados como detalhes dos requisitos iniciais, detalhes esses descritos no Capítulo 6. Por sua vez, os detalhes foram convertidos em tarefas a serem implementadas conforme descritas no Capítulo 7.

4.2.1 Pré-desenvolvimento

Na fase de pré-desenvolvimento teve lugar a Análise de Requisitos, onde foram levantados os requisitos iniciais, a partir da lista de objectivos definidos em reunião com as partes interessadas. Com base nos requisitos, foi feito um desenho inicial da solução com a respectiva arquitectura. Nesta fase o desenho e arquitectura estavam ainda bastante vagos e longe do resultado final por se desconhecer ainda todos os contornos que iriam exigir os requisitos, na sua concretização prática. Contudo, os artefactos obtidos foram suficientes para dar inicio às actividades de pesquisa e desenvolvimento iterativo da fase seguinte.

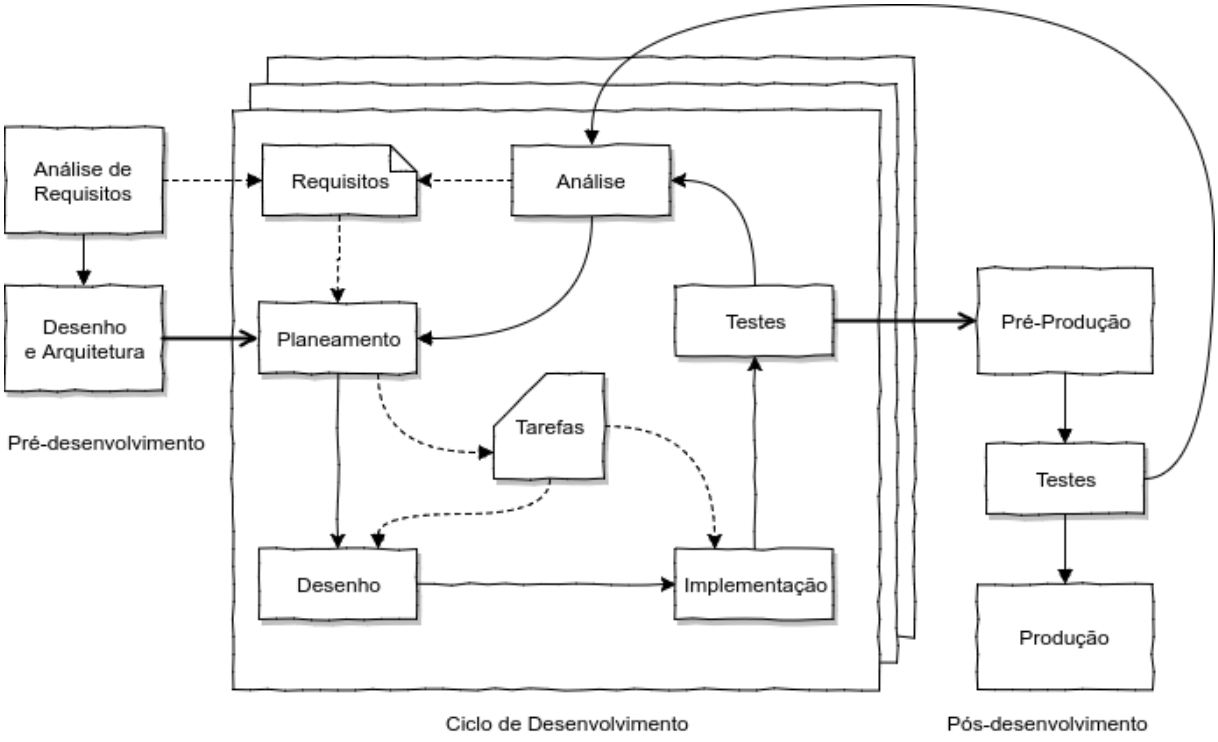


Figura 4.1: Diagrama do processo de desenvolvimento.

4.2.2 Ciclo de Desenvolvimento

A fase de desenvolvimento é composta por um ciclo de actividades que se repete semana após semana ao longo do período destinado à implementação do projecto. Este ciclo, iterativo incremental, tem por objectivo gerar requisitos implementados que se acumulam no produto em desenvolvimento enquanto que gera também, em *by-product* a revelação de novos requisitos, refinados de requisitos existentes e que alimentam as iterações seguintes. O processo deste ciclo é iniciado com o planeamento, onde são escolhidos os requisitos de implementação prioritária. Neste momento são escolhidos os requisitos mais críticos, aqueles dos quais a solução desenhada depende mais, para serem desenvolvidos em primeiro. Ainda dentro do planeamento, são definidas as tarefas necessárias para implementar um ou mais desses requisitos de modo a prever uma semana preenchida de trabalho. De seguida é desenhada a forma de implementar essas tarefas antes de passar à implementação, onde são, de facto, implementadas as tarefas que pretendem implementar o requisito. Após a implementação, são realizados testes com o intuito de verificar se a implementação cumpre, de facto, os requisitos. Do resultado dos testes, é feita uma análise de onde se retiram as ilações necessárias que eventualmente darão origem a alterações na lista de requisitos ou na arquitectura, sempre com o objectivo supremo de cumprir com todos os requisitos que completem em termos práticos os requisitos iniciais.

4.2.3 Pós-desenvolvimento

A fase final do desenvolvimento é composta pela instalação do produto desenvolvido no servidor, em condições idênticas às usadas em produção, para ser testado em ambiente pré-produção antes de ser colocado em produção, sujeito a condições reais de funcionamento. Eventualmente, durante os testes de pré-produção, pode ser necessário voltar ao ciclo de desenvolvimento para corrigir alguma falha inesperada.

4.3 Ferramentas e Tecnologias Adoptadas

Nesta secção são indicadas as ferramentas escolhidas para a implementação do projecto, de acordo com a fase do projecto em que foram usadas.

4.3.1 Preparação do Projecto

Durante todo o desenvolvimento do projecto foi usado um computador de secretária fornecido pela empresa, com sistema operativo Linux, distribuição Ubunto e ambiente gráfico Xfce.

4.3.2 Análise de Requisitos

Para analisar os requisitos e apontamentos em geral, foi usado o editor de texto online Google Docs, por este permitir guardar os documentos automaticamente na *cloud* e fazer histórico de revisões com funcionalidades de reversão.

4.3.3 Estado da Arte

Para o estudo do estado da arte foi usado o *browser* Mozilla FireFox por ser um dos melhores *browsers* do momento e conter um bom gestor de sites favoritos.

4.3.4 Arquitectura

Para a criação de diagramas de definição de arquitectura, foi usado o Draw.io [16], ferramenta online, que permite guardar directamente na *cloud* e que possui as mais diversas figuras para usar em diagramas de informática. Com essa ferramenta, foram criados diagramas Unified Modeling Language (UML) para representar a arquitectura da solução, através de fluxogramas e diagramas de fluxo.

4.3.5 Desenvolvimento

Durante a fase de desenvolvimento, foram usadas as seguintes ferramentas:

PyCharm [17] - Integrated Development Environment (IDE) de programação em Python;

PostgreSQL [18] - Sistema de Gestão de Base de Dados (SGBD) para a aplicação de servidor.

AndroidStudio [19] - IDE de programação Java para Android;

Git - Sistema de controle de versões distribuído;

Django [20] - *Framework* grátis de código aberto, para desenvolvimento Web em Python, onde o foco é criação rápida de aplicações sem necessidade de "reinventar a roda". Na Figura 4.2 está representada a arquitectura do Django no contexto em que será usada no projecto. Foi escolhida a versão 2 devido ao fim de suporte previsto para versões anteriores a partir de 2020;

REST - Para criar interfaces API para sistemas distribuídos;

JSON - Para a representação de objectos a transmitir entre sistemas distribuídos.

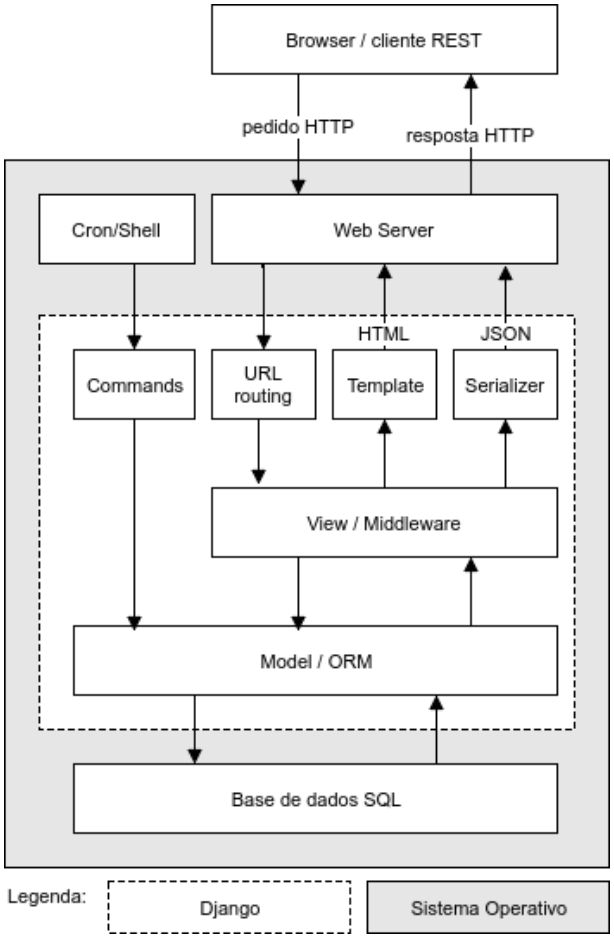


Figura 4.2: Arquitectura Django.

4.3.6 Testes

Para testar o produto durante e depois do desenvolvimento, foram usadas as seguintes ferramentas:

unittest [21] - Framework de testes para Python, incluída no Django;

Postman [22] - Utilitário para testes a API REST;

Apium [23] - Framework de testes para multi-plataformas, que permite testar simultaneamente a aplicação de servidor e a aplicação no dispositivo móvel;

Nginx [24] - Serviço HTTP/HTTPS para a aplicação de servidor;

uWSGI [25] - Serviço aplicacional interpretador de Python para a aplicação de servidor.

4.3.7 Relatório

Para a escrita do presente relatório, foram usadas as seguintes ferramentas:

Draw.io [16] - Ferramenta de desenho on-line, usada para criação de diagramas diversos;

StarUML [26] - Ferramenta de UML Para criação de diagramas de modelos Entidade Relacionamento (ER);

Texmaker [27] - Editor de LaTeX, para a composição das páginas do documento.

4.4 Conclusão

Neste capítulo foi descrita a metodologia usada no processo de desenvolvimento deste projecto. A complexidade desconhecida à partida do produto a desenvolver e o facto de existir apenas um elemento na equipa de desenvolvimento, foram factores de dificuldade num processo que poderia ser mais simples, com uma equipa mais numerosa. Assim, apesar de mais moroso, o método baseado em "*trial and error*"[28] revelou-se, de facto, a única forma encontrada de resolver alguns problemas, nomeadamente aqueles que só iriam surgir durante uma utilização real do produto, apesar de poderem estar, aparentemente, cumpridos os requisitos iniciais. Terminando o capítulo, foram também apresentadas as ferramentas e tecnologias escolhidas e usadas ao longo das fases do projecto.

Capítulo 5

Planeamento

Neste capítulo, são descritas as fases principais do projecto e apresentada a sua cronologia. São também enumerados os riscos previstos durante o desenvolvimento e as formas encontradas de os reduzir.

5.1 Fases do Projecto

Conforme referido na Proposta de Estágio no Anexo D, o estágio será repartido pelas seguintes fases:

- **T1** - Estudo de Plataformas - Nesta fase são estudadas as ferramentas a usar, analisados os componentes que interagem com o projecto e feitos estudos sobre soluções existentes no mercado.
- **T2** - Levantamento de Requisitos - Nesta fase são enumerados requisitos a partir dos objectivos do projecto, feito o seu planeamento e preparado o desenvolvimento através de uma arquitectura orientada à solução pretendida.

  T2.1 - Esclarecimento do problema a resolver;

  T2.2 - Enumeração de requisitos;

  T2.3 - Planeamento do projecto;

  T2.4 - Arquitectura da solução.

- **T3** - Desenvolvimento - Nesta fase são implementados os requisitos funcionais levantados no Capítulo 2 e enumerados nas Tabelas 2.1 e 2.2.
- **T4** - Testes - Nesta fase são testados e validados os requisitos e objectivos iniciais do projecto.
- **T5** - Relatório - Escrita do Relatório de Estágio.

5.2 Riscos

Com o planeamento definido, foram analisados os pontos susceptíveis de maiores dificuldades, e atribuídas classificações de acordo com a Tabela 5.1

Tabela 5.1: Classificação de Risco

Risco	**Probabilidade**	**Impacto**
CR1	Baixa	Baixo
CR2	Alta	Baixo
CR3	Baixa	Alto
CR4	Alta	Alto

Sendo CR1 a classificação de menor risco e CR4, a mais alta, deve ser dada prioridade à mitigação das situações de risco mais elevado.

R1 (CR4) - Desconhecimento da linguagem de programação - Este risco é altamente provável pois é um facto que, no início do projecto, o autor desconhecia a linguagem Python e a framework Django. Este risco tem alto impacto, pois um dos objectivos do estágio é que este seja desenvolvido usando as ferramentas tipicamente usadas na empresa para este tipo de projecto. Este risco mitiga-se investindo algum tempo a aprender o básico da linguagem com recurso a documentação e fóruns *online*.

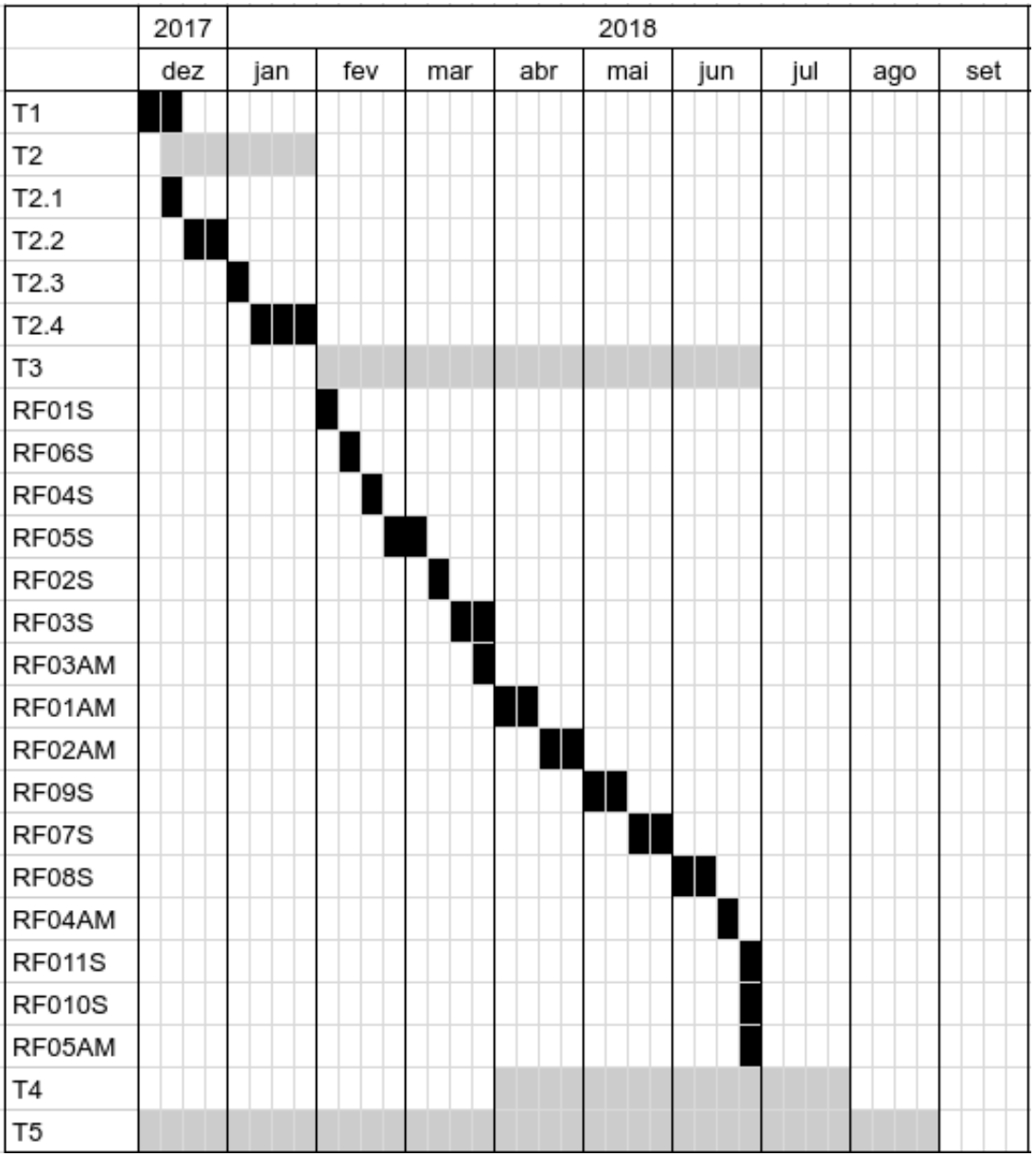
R2 (CR3) - Tecnologias de terceiros - Eventuais falhas no uso de tecnologias de terceiros são um risco pouco provável por serem tecnologias bastante testadas e documentadas. Contudo, se alguma destas tecnologias não produzir os resultados previstos, pode ser necessário alterar a arquitectura inicial, o que pode causar atrasos no desenvolvimento do projecto. Como forma de mitigar este tipo de risco, vão realizar-se com prioridade máxima, protótipos funcionais, reaproveitáveis, para casos de uso que usem tecnologias de terceiros.

R3 (CR1) - Recursos de terceiros - Eventuais atrasos na entrega de recursos por parte de terceiros, tais como hardware, infraestruturas ou chaves para VPN podem causar atrasos em algumas fases do projecto, embora não se preveja que possa ser motivo de fracasso do projecto. Para prevenir este tipo, vão se efectuar os pedidos de recursos com algum tempo de antecedência ao momento em que serão necessários.

5.3 Planeamento

Na Tabela 5.2 está representado o espaço temporal reservado às tarefas. A execução das tarefas foi planeada tendo em conta as eventuais dependências entre elas. Na primeira coluna é reservada às fases do projecto, conforme referidas na Secção 5.1. A fase T3 é composta pelo desenvolvimento dos requisitos funcionais do projecto, referidos na Secção 2.1. A fase T4, referente aos testes e validação, ocorreu em paralelo com a fase T3 a partir do momento em que o projecto em desenvolvimento já permitia ter algumas funcionalidades testadas. A fase T5, referente ao Relatório de Estágio, teve lugar ao longo de todo o projecto.

Tabela 5.2: Planeamento do Projecto



5.4 Conclusão

O planeamento é uma fase importante do projecto, pois consiste na estimativa do tempo necessário a cada tarefa e na gestão do tempo disponível para as tarefas previstas. Os riscos são componentes do planeamento a ter em atenção pois podem comprometer todo o projecto, tendo sido por isso imperativo encontrar formas de mitigação para não comprometer o sucesso do projecto.

Capítulo 6

Arquitectura

Neste capítulo, são apresentadas as propostas de arquitectura para o sistema, de acordo com os requisitos especificados no Capítulo 2.

6.1 Enquadramento

De modo a entender o enquadramento do sistema a desenvolver, está representada na Figura 6.1, a infraestrutura com os componentes que intervêm no sistema.

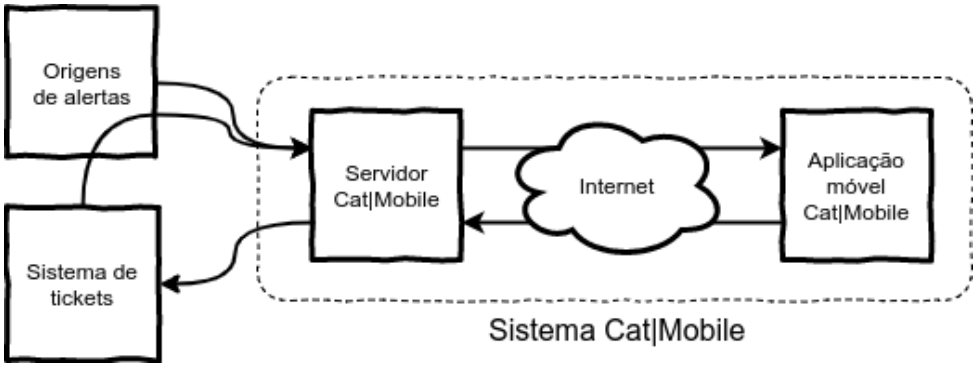


Figura 6.1: Diagrama de Arquitectura de alto nível do sistema.

O sistema Cat|Mobile, representado pela área delimitada a tracejado na Figura 6.1, é composto por um servidor e por uma aplicação móvel que comunica com o servidor através da Internet. Fora da área a tracejado encontram-se representadas as origens de alertas que alimentam o servidor e o sistema externo de *tickets* usado pelo servidor para fazer registos externos de *tickets*.

Origens de alertas As origens de alertas representam as diversas ferramentas externas que fornecem os alertas ao Cat|Mobile. Estas origens podem ser divididas em 2 grupos, as que enviam os alertas por iniciativa própria e as que disponibilizam os alertas apenas por consulta.

Sistema de *tickets* O sistema de *tickets* é a ferramenta externa utilizada pelo Cat|Mobile para quando os operadores decidem criar um *ticket* durante o tratamento dos alertas. A particularidade do sistema de *tickets* é que este também pode ser uma origem de alertas. Por exemplo, quando um *ticket* é criado por iniciativa de um cliente, esse *ticket* é convertido em alerta que o Cat|Mobile deve transmitir, para ser tratado por um operador. Neste caso, ao ser tratado o alerta, o Cat|Mobile terá de notificar o sistema de *tickets* das alterações de estado, nomeadamente quando se iniciar e terminar o tratamento desse *ticket*.

Servidor Cat|Mobile Para receber e enviar os alertas aos operadores, um servidor com todo o software necessário, deverá estar instalado em rede com acesso às origens de alertas. Esse servidor deverá poder comunicar com a aplicação móvel instalada em dispositivos com ligação à Internet.

Aplicação Móvel Cat|Mobile A aplicação móvel será a interface que os operadores dispõem para usar o Cat|Mobile. A aplicação só funciona se existir ligação à Internet pois todos os dados são transmitidos em tempo real pelo servidor e nunca vão ficar persistidos localmente no dispositivo móvel.

6.2 Desenho e Especificação da Arquitectura Externa

De forma a entender a estrutura geral do sistema, do ponto de vista externo às aplicações, é apresentada a arquitectura externa do sistema, na Figura 6.2. A arquitectura externa é composta por componentes localizados em duas áreas distintas. No exterior da empresa, localizam-se os múltiplos dispositivos móveis com a aplicação Cat|Mobile instalada e que é usada pelos operadores. Fora da empresa, na Cloud, encontra-se também o sistema de distribuição de mensagens FCM. No interior da empresa, encontra-se o servidor onde estará instalado o serviço Cat|Mobile, usado pelo administrador. Também no interior da empresa, encontram-se os componentes externos do serviço, usados pelo Cat|Mobile.

Interior da Empresa - No interior da empresa, fica localizado o Servidor e a rede local onde se encontram componentes capazes de comunicar com o servidor. Na rede local encontram-se componentes externos não controlados pelo Cat|Mobile, nomeadamente as origens de alertas, o sistema de tickets e os administradores da aplicação que interagem através de browser Web.

Servidor - O servidor Cat|Mobile é um sistema composto pela aplicação, pelo serviço HTTP, cron, Base de Dados e interface de rede, sendo estes os componentes internos do servidor, com relevância para a arquitectura do sistema. HTTP Server - Dentro do servidor, o serviço HTTP serve para os acessos Web e para a API Representational State Transfer (REST).

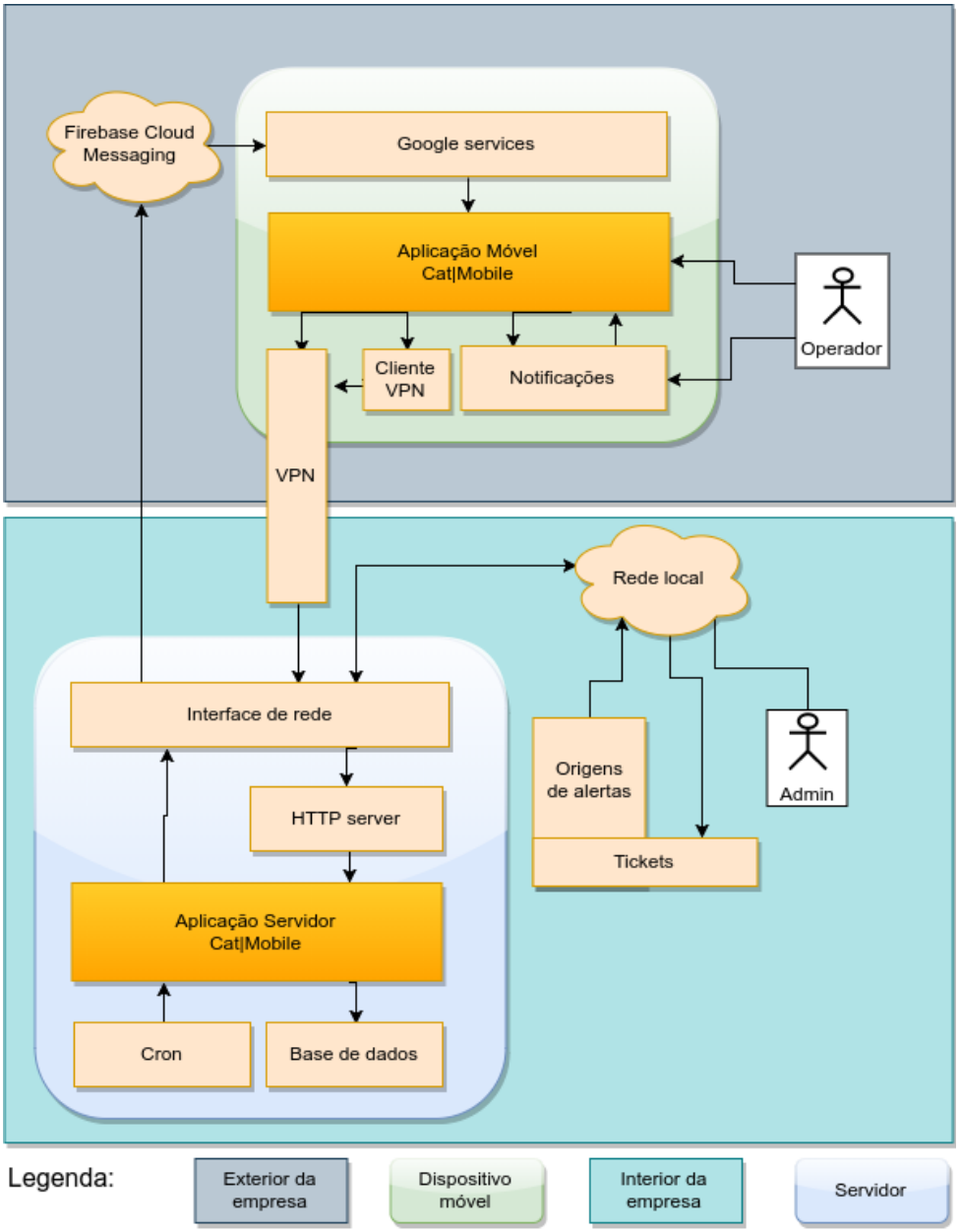


Figura 6.2: Diagrama de Arquitectura Externa do Sistema.

Interface de rede - Para o servidor poder comunicar com o exterior, necessita da interface de rede. Essa interface dispõe apenas de endereço Internet Protocol (IP) na rede interna destinada aos servidores da empresa. Todos os acessos ao servidor e acessos vindos dele, são determinados pela Firewall da empresa. Assim, o servidor pode fazer pedidos à Internet por iniciativa própria, mas só aceita pedidos vindos da rede local ou vindos através da Virtual Private Network (VPN).

Base de dados - A persistência dos dados em disco é assegurada por pelo SGBD. Esse sistema apenas estará disponível localmente no servidor, estando barrado o acesso por rede à base

de dados. Eventualmente, este sistema pode vir a ser movido para o exterior, neste caso ficando com acesso por rede.

Cron - Para os alertas vindos de origens com consulta, é necessário que essas consultas sejam agendadas no cron. Este componente pode vir a ser necessário para outras funcionalidades que se levantem durante o desenvolvimento da solução.

VPN - O servidor apenas tem comunicação com o exterior através de VPN. Essa VPN é estabelecida entre um router na infraestrutura de rede da empresa e o cliente de VPN instalado no dispositivo móvel para que a aplicação móvel tenha acesso ao servidor.

Firebase Cloud Messaging (FCM) - Plataforma Google para distribuição de mensagens em dispositivos móveis.

Dispositivo Móvel - O dispositivo móvel é o dispositivo usado pelos operadores, onde corre a aplicação móvel Cat|Mobile que interage com o servidor através da Internet.

Cliente VPN - Para que a aplicação móvel Cat|Mobile possa comunicar com o servidor, é necessário ter acesso à VPN onde o servidor se encontra disponível. Para estabelecer a VPN no dispositivo móvel é necessária uma aplicação de cliente para VPN.

Notificações - O Sistema Operativo Android dispõe de uma API para a apresentação de notificações ao utilizador. As notificações são criadas através da aplicação ou serviço que está a correr no dispositivo. Essas mesmas notificações permitem ao utilizador interagir com aplicação quando este reage à notificação. Por exemplo ao clicar na notificação, com determinado identificador, a aplicação é iniciada e é apresentado ao utilizador, o elemento identificado.

6.3 Arquitectura Interna

Nesta secção é descrito o desenho e especificação da arquitectura interna do servidor e da aplicação móvel.

6.3.1 Arquitectura Interna da Aplicação em Servidor

A componente de servidor do Cat|Mobile é responsável pela gestão de operadores, turnos, dispositivos móveis, monitorização de notificações e também pelo serviço de suporte de dados para a aplicação móvel. Este componente aplicacional do Sistema depende de componentes externos para algumas das funcionalidades, nomeadamente de FCM para as notificações móveis e de Cron para a execução agendada de processos. Na Figura 6.3 estão representados os módulos que compõem a componente aplicacional com as respetivas interligações. A descrição

dos módulos, apresentada de seguida, é composta pela descrição dos objetivos do módulo e da interligações com outros módulos.

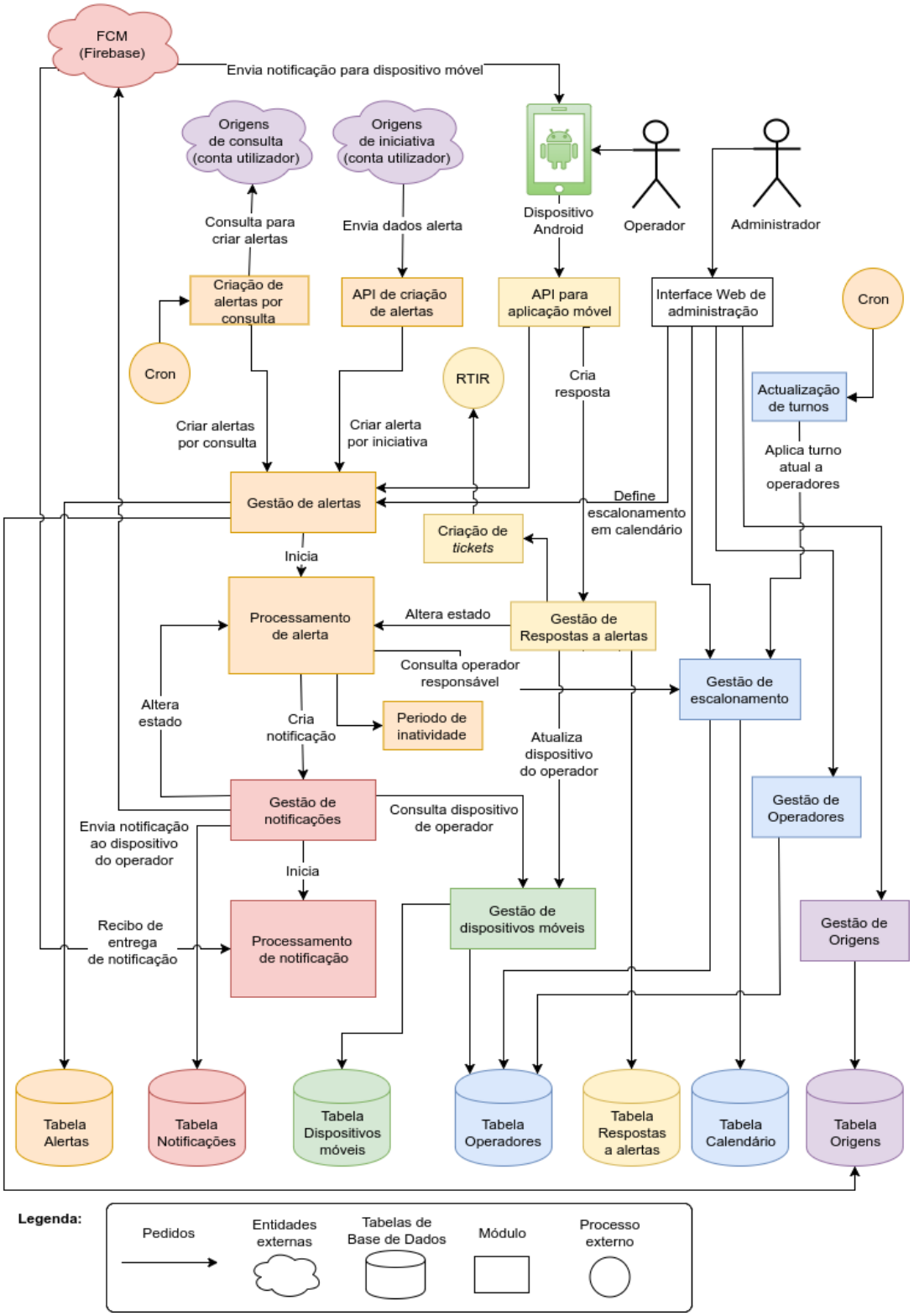


Figura 6.3: Diagrama de Arquitectura Interna da Aplicação em Servidor.

Gestão de Utilizadores

Os utilizadores Cat|Mobile são todas as entidades que interagem com o sistema para consulta ou manipulação de dados. Sejam operadores, origens ou administradores, estando autenticados no sistema, é possível manter registo inequívoco das ações de cada utilizador. Na Figura 6.4 está representado o sistema que compõe a gestão de utilizadores. Esta gestão é composta pela Gestão de Operadores e pela Gestão de Origens, ambas acessíveis através do Interface de Administração. Com os utilizadores agregados em grupos como representado na Tabela 6.1, é possível definir permissões por tipo de utilizador para aceder a funcionalidades específicas. Estas permissões permitem distinguir os utilizadores que podem criar alertas (Origens) dos utilizadores que podem responder aos alertas (Operadores).

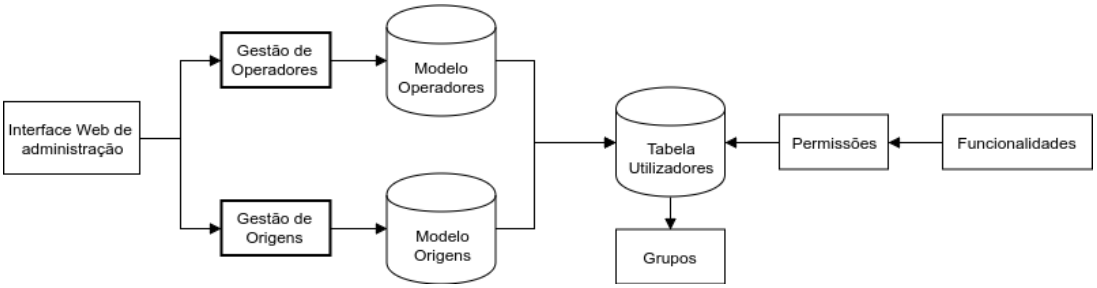


Figura 6.4: Diagrama da Gestão de Utilizadores.

Tabela 6.1: Grupos de Utilizadores.

Grupo	Descrição
Operadores	Grupo de utilizadores que apenas tem acesso ao Cat|Mobile através da aplicação móvel. A aplicação móvel é utilizada para dar resposta a alertas
Origens	Grupo para as ferramentas externas de geração de alertas que podem criar alertas no Cat|Mobile. Tendo um utilizador distinto para cada ferramenta, para além de identificar a ferramenta, é possível revogar o acesso dessa ferramenta no caso de uma eventual necessidade de segurança.
Administradores	Grupo de utilizadores com acesso ao interface Web de administração do Cat|Mobile

Gestão de Alertas

Os alertas são criados através da API de criação de alertas ou do cron que executa a tarefa de criar alertas por consulta externa. Ambas as formas de criação de alertas são usadas por contas de utilizadores do grupo Origens. Nos dados do alerta é enviada a identificação do cliente onde foi gerado o alerta. Ao ser criado um alerta no Cat|Mobile, este é registado em base de dados relacional que associa o alerta à sua origem e ao cliente. Simultaneamente é registada a data e hora da criação do alerta. Caso o alerta tenha o RTIR como origem, deve ser enviado o identificador do *ticket* no RTIR para prevenir a criação de alertas duplicados. Conforme representado na Figura 6.5, para além dos módulos de criação de alertas, também os módulos de *Interface de Administração* e *API para Aplicação Móvel* interagem com a *Gestão de Alertas*, contudo, apenas para consulta.

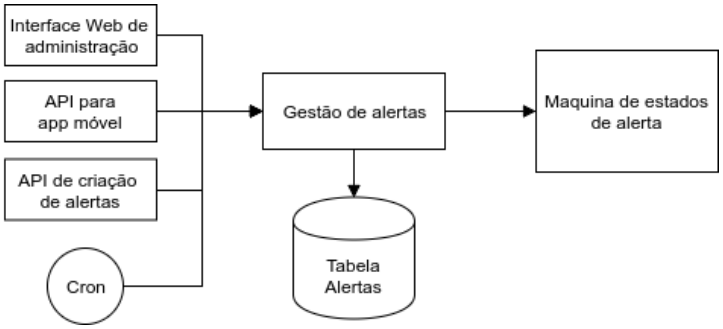


Figura 6.5: Diagrama da Gestão de Alertas.

API para Criação de Alertas

Esta API permite a criação de Alertas no sistema. Conforme representado na Figura 6.6, a criação de alertas necessita de uma conta de utilizador do grupo de Origens. Isto significa que as credenciais usadas por operadores não são válidas para criar alertas. As ferramentas de geração de alertas que têm capacidade de executar tarefas por sua própria iniciativa, usam credenciais próprias que as identificam como sendo uma origem de alertas no Cat|Mobile. Para além da autenticação, para a criação de um alerta são necessários os dados referidos pela Tabela 6.2.

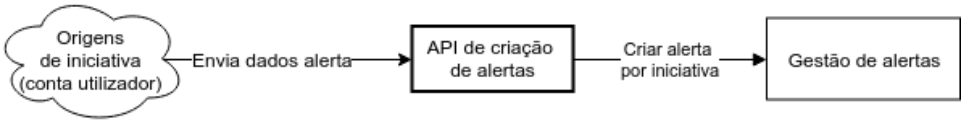


Figura 6.6: Diagrama da API para Criação de Alertas.

Tabela 6.2: Dados para a criação de alertas.

Campo	Descrição
Título	Uma expressão curta e legível que permite distinguir os diferentes alertas entre si
Descrição	A descrição detalhada o suficiente para esclarecer o operador da situação do alerta
Severidade	Um valor numérico pré-estabelecido que quantifica a prioridade que o operador deve dar ao tratamento do alerta. Este valor é também usado pelo sistema para estabelecer o tempo limite disponível para o início do tratamento do alertas
Cliente	A identificação do cliente a que se refere o alerta. Cada ferramenta de origem de alertas pode gerar alertas de vários clientes

Criação de Alertas por Consulta Externa

Existem ferramentas de origens de alertas que não têm capacidade de iniciativa própria de enviar os alertas ao Cat|Mobile, mas que dispõem de API para consulta. Para esses casos a iniciativa de obter alertas terá de vir do Cat|Mobile, que o fará em intervalos de tempo regulares, definidos no *cron* do sistema operativo. Conforme visível na Figura 6.7, continua a ser necessário identificar a origem do alerta, contudo não serão usadas credenciais de acesso à API do Cat|Mobile, mas sim credenciais próprias de acesso à API da ferramenta externa que estarão associadas a uma origem de alertas do Cat|Mobile.

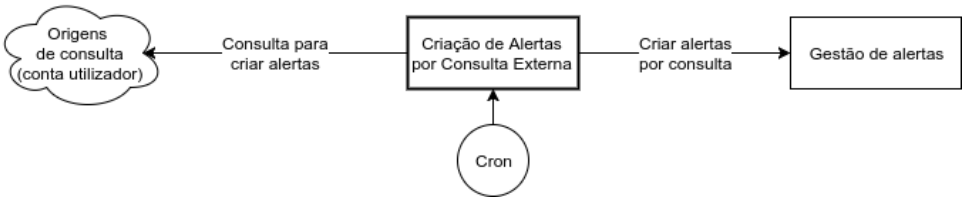


Figura 6.7: Diagrama da Criação de Alertas por Consulta Externa.

Gestão de Notificações

As notificações são mensagens enviadas aos operadores para os informar da existência de alertas que necessitam de ser tratados. Na Figura 6.8 está representado o sistema que compõe este módulo. Existem dois tipos de notificação:

Principal - Notificação que é enviada ao operador responsável. Esta notificação é mais ruidosa e permanece por determinado tempo ou até ser respondida, de forma idêntica à de uma chamada telefónica.

Secundária - São notificações enviadas aos outros operadores dentro do grupo Oncall. Estas notificações são silenciosas e apenas informativas da existência de um novo alerta no Cat|Mobile.

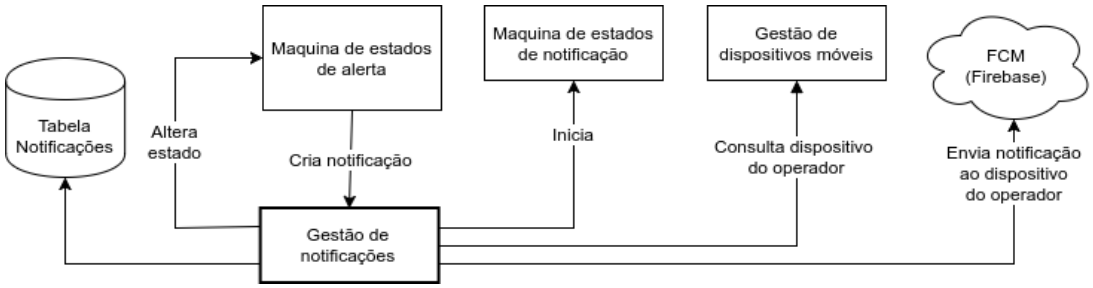


Figura 6.8: Diagrama da Gestão de Notificações.

Para contactar os operadores, as notificações são enviadas aos dispositivos móveis que estes usam. Na Tabela 6.3 é descrita a estrutura relacional das notificações com outras entidades do sistema.

Tabela 6.3: Estrutura relacional de uma notificação.

Campo	**Descrição**
Operador	Indentificação do operador a quem é enviada a notificação
Dispositivo	Identificação do dispositivo usado pelo operador no presente momento
Tipo	Identificação do alerta a que se refere a notificação
Data	Instante em que a notificação é criada

Gestão de Escalonamento

As notificações de alertas são sempre enviadas aos operadores responsáveis. Conforme se pode ver na Figura 6.9, os operadores responsáveis diariamente são definidos pelo administrador no calendário. Por regra definida pela equipa SOC, cada alerta tem um único operador responsável. No entanto o operador responsável pode ser um de entre vários disponíveis a dado momento. O administrador é quem define os operadores disponíveis a cada momento e qual a ordem pela qual estes devem ser notificados em caso de escalonamento do operador responsável.

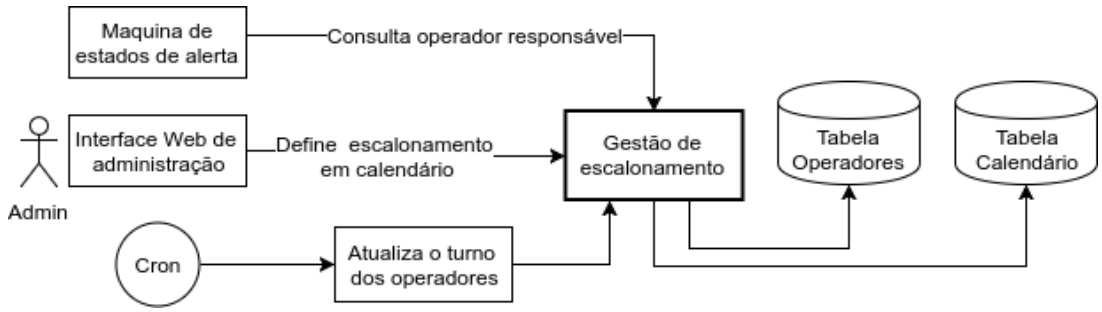


Figura 6.9: Diagrama da Gestão de Escalonamento.

Actualização de Turnos

O operador responsável pela resposta a alertas é o primeiro operador de uma tabela de escalonamento constituída por outros operadores da equipa *Oncall*. Essa equipa é pré definida em calendário pelo administrador. A entrada em vigor das equipas definidas em calendário acontece com o decorrer do tempo dependendo apenas do cron para tornar efetivos os turnos, tal como representado pela Figura 6.9.

Gestão de Dispositivos Móveis

Conforme representado pela Figura 6.10, ao ser usada por um operador, a aplicação móvel CatlMobile envia ao servidor dados identificadores do dispositivo em que está a ser executada. Com esses dados, o servidor regista o dispositivo em base de dados relacional e relaciona o dispositivo com o operador que fez o login para assim ficar registado qual o dispositivo que está atualmente a ser usado pelo operador. Sempre que um novo dispositivo seja usado por um operador, é actualizada a relação que define o dispositivo atual do operador mantendo o registo do dispositivo anterior. O registo de todos os dispositivos permite ao administrador a qualquer momento consultar por quem já foi usado cada dispositivo.

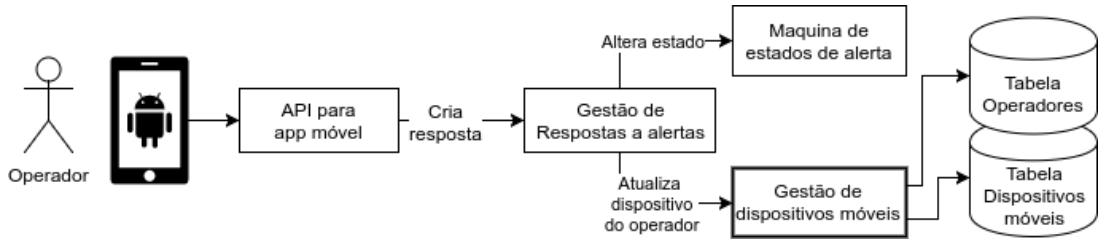


Figura 6.10: Diagrama da Gestão de Dispositivos.

Processamento de Alertas

No momento em que é recebido um alerta, o Cat|Mobile inicia o processo representado pela Figura 6.11. Esse processo começa por selecionar o operador que fica definido como responsável pelo alerta. Essa seleção de operador é feita através da consulta da tabela de escalonamento. O Sistema permanece no estado de seleção de operador enquanto não houver um operador a selecionar. Assim que é selecionado o operador responsável, tenta fazer o envio da notificação. Caso o processo de envio de notificação seja concluído com sucesso, o sistema fica a aguardar pela confirmação de entrega da notificação na aplicação móvel. Recebida a confirmação de entrega da notificação na aplicação móvel, que entretanto já deve ter surgido no dispositivo móvel, o sistema passa a aguardar a resposta do operador. Resposta essa que será de aceitação ou de rejeição de tratamento do alerta. Com a aceitação, o sistema passa para o estado de tratamento do alerta e o sistema fica a aguardar pela sua conclusão. Uma vez concluído, é terminado o processo do alerta. Caso o processo de envio da notificação não seja concluído com sucesso, ou caso expire o tempo designado para a visualização do alerta na aplicação móvel, ou caso não seja aceite ou expire o limite de tempo para aceitação do alerta, o sistema volta a selecionar outro operador responsável para repetir todo o processo com um novo operador.

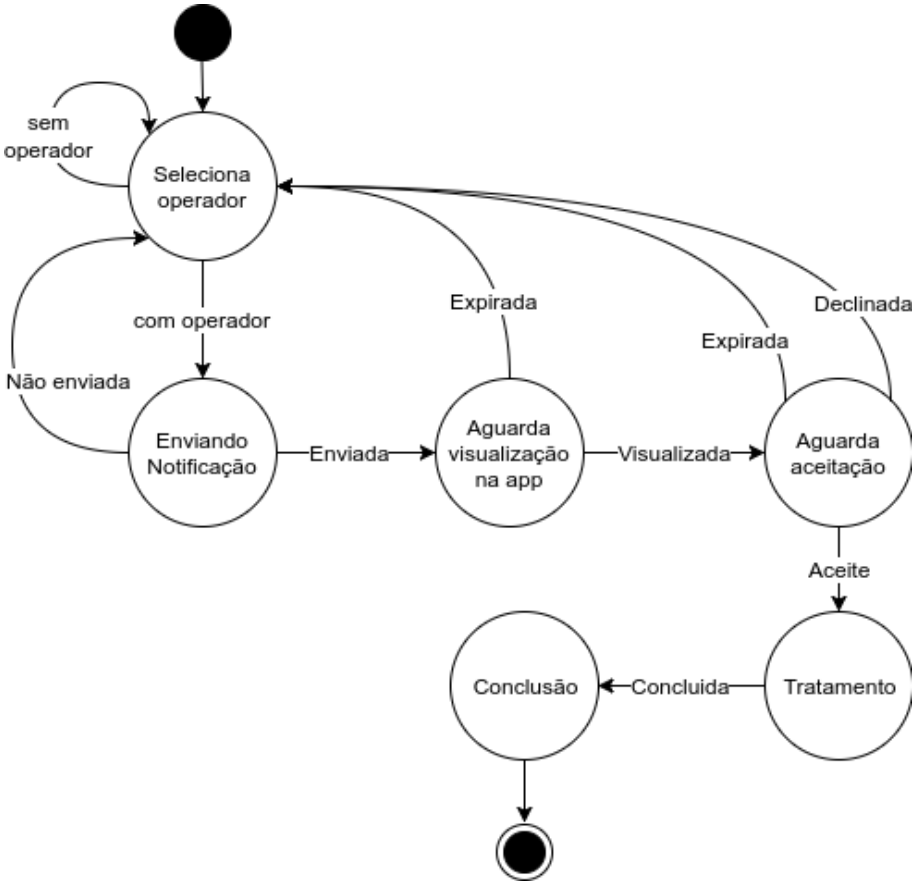


Figura 6.11: Diagrama de Estados para o Processamento de Alertas.

Processamento de Notificações

O processo de envio de notificação representado na Figura 6.12 é iniciado durante o processamento de alertas. Quando o processamento de alertas já tem selecionado um operador responsável e pretende enviar a notificação ao operador, é iniciado o processo de envio de notificação. Como as notificações são enviadas ao operador através do seu dispositivo móvel, este processo começa por selecionar o dispositivo móvel do operador. O dispositivo móvel do operador é definido quando o operador faz login na aplicação móvel. Nesse momento o sistema toma conhecimento do dispositivo móvel do operador e mesmo que o operador não esteja a usar a aplicação Cat|Mobile, o sistema sabe que pode enviar notificações a esse dispositivo porque este está na posse do operador. O problema de um operador "A" ficar sem dispositivo surge quando um outro operador "B" faz login no dispositivo que era do operador "A". Nesse momento o dispositivo que era do operador "A" passa a pertencer ao operador "B" e o operador "A" fica sem dispositivo perante o Cat|Mobile. Para mitigar este problema o sistema mantém-se no estado de selecção de dispositivo durante um determinado tempo para dar hipótese ao operador responsável de fazer login num dispositivo que passará a ser o seu. Durante este estado, ficam em aberto soluções a implementar futuramente que permitam contactar o operador responsável de algum modo para o alertar da falta de dispositivo móvel para receber notificações. Caso o operador já tenha um dispositivo associado a si, o processo faz o envio da notificação e fica a aguardar a confirmação de entrega durante um determinado tempo. Caso o dispositivo esteja desligado ou não tenha acesso à Internet, a notificação não será entregue dentro do tempo esperado. No caso de expirar o tempo de espera pela confirmação de entrega da notificação, o sistema volta à seleção de dispositivo, dispositivo esse que pode ter mudado entretanto e volta a tentar a notificação. Permanece neste ciclo enquanto não expirar o tempo total reservado à notificação. Caso expire o tempo de espera por um dispositivo funcional ao qual seja possível entregar a notificação, termina o processo de notificação para este operador e é devolvida uma resposta negativa ao processamento de alerta para que selecione outro operador a quem enviar notificação.

Período de Inatividade

Dado que o Cat|Mobile é um sistema para apoio à alarmística para quando os operadores se encontram fora da localização física da empresa, quando os operadores se encontram na empresa, são usadas as ferramentas habituais para tratamento da alarmística. Durante o horário em que os operadores estão na empresa, o Cat|Mobile deve permanecer inativo, sem enviar notificações de alertas aos operadores. Para gerir esses períodos de inatividade, o administrador deve poder configurar para cada dia da semana, as horas em que o sistema deve estar inativo. Esta tabela de inatividade é para ser usada pelo sistema de processamento de alertas antes de decidir o início do processo.

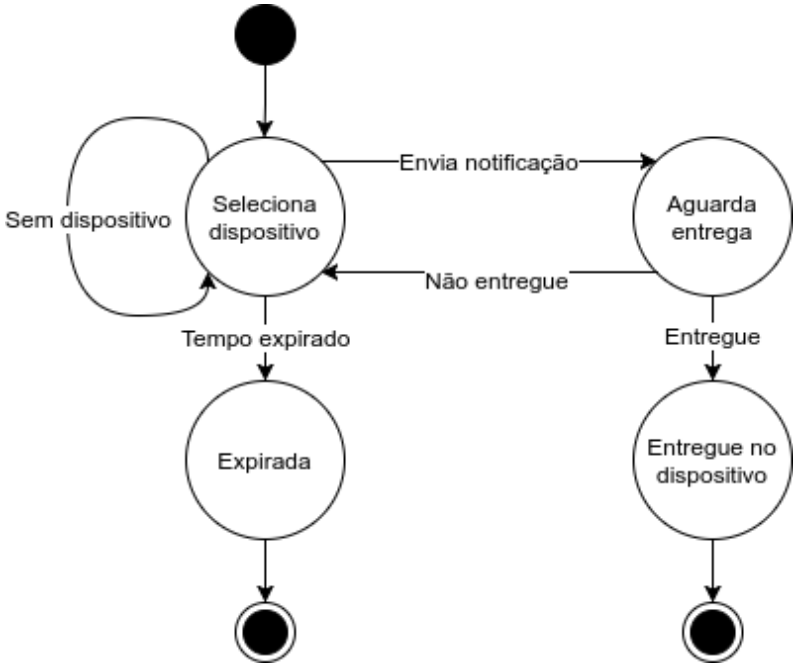


Figura 6.12: Diagrama de Estados para o Processamento de Notificações.

Gestão de Resposta a Alertas

As respostas aos alertas são todas as respostas dadas manualmente pelo operador e também as automáticas dadas pela aplicação móvel. As respostas automáticas são dadas pela aplicação quando o operador visualiza o alerta e que representam o recibo de leitura, útil ao administrador como sendo de não repúdio do operador. Na Tabela 6.4 estão descritas as possíveis respostas manuais dadas pelo operador ao alerta.

Tabela 6.4: Tipos possíveis de resposta a Alertas

Resposta	**Descrição**
Aceitar	O operador inicia o tratamento do alerta
Declinar	O operador informa que não pode iniciar o tratamento do alerta
Responder	O operador escreve um texto de resposta que fica registado junto com o alerta, podendo ser lido sempre que o alerta é consultado. O operador pode criar várias respostas durante o tratamento do alerta. Na resposta o operador tem ainda duas opções: • Opção para criar um ticket na ferramenta externa de tickets, usando o texto da resposta como descrição do ticket a criar; • Opção para dar como concluído o tratamento do alerta.

Criação de *tickets*

Os *tickets* são registos de incidentes no RTIR[2]. Como referido no Capítulo 3, o RTIR é a ferramenta de gestão de *tickets* usada na empresa. O RTIR é, simultaneamente origem de alertas e destino para os *tickets* criados pelos operadores durante o tratamento de alertas na aplicação móvel Cat|Mobile. Assim sendo, a funcionalidade de criação de *tickets* deve ser tida em conta durante a criação de alertas para que os *tickets* criados pelo Cat|Mobile não criem novos alertas, de modo circular. A criação de *ticket* é uma opção possível durante a resposta escrita a um alerta cujo tratamento já está iniciado. O RTIR tem uma API que é acessível através de credenciais de utilizador próprias da aplicação. Quando o operador ativa a opção de criação de *ticket*, o Cat|Mobile usa as credenciais para criação de *tickets*, associadas ao operador, para criar o *ticket* no RTIR.

API para Aplicação Móvel

Dado que um dos requisitos de segurança especifica que todos os dados partilhados com terceiros não devem conter informação confidencial, ao usar um serviço externo para o envio de notificações, o conteúdo dessas notificações tem obrigatoriamente de ser não confidencial. Assim, em acordo com as partes interessadas, ficou decidido que os dados a enviar por notificação seriam apenas numéricos sem significado algum confidencial, apenas interpretáveis pelos operadores e impossível de descrever o conteúdo do alerta ou da identidade do cliente gerador, mesmo para os operadores, sem o auxílio da aplicação móvel e respectiva autenticação. Assim, para aceder ao conteúdo do alerta referido na notificação, o operador não tem outra forma senão a de usar a aplicação móvel, com a devida autenticação. A aplicação, por sua vez, acede ao servidor com autorização do operador e consulta o alerta referido na notificação usando para isso o identificador numérico que faz parte dos dados enviados na notificação e que apenas identificam o alerta perante o Cat|Mobile, não comprometendo informação confidencial. Para processar esse e outros pedidos da aplicação móvel, o servidor Cat|Mobile dispõe dos serviços em API descritos na Tabela 6.5.

Interface Web de Administração

Para administrar o Cat|Mobile, o sistema dispõe de um interface gráfico para Web acessível pelos utilizadores do grupo de administração mediante as respetivas credenciais de acesso.

Dashboard principal do Interface de administração Para visualizar o estado geral do Cat|Mobile, com as seguintes informações:

- Tabela de escalonamento de operadores em vigor e da seguinte;
- Gráfico com volume de alertas recebidos por dia durante os últimos 30 dias;
- Listagem do último alerta recebido por cada cliente;
- Gráfico com volume de alertas recebidos por cliente.

Tabela 6.5: Serviços para API de aplicação móvel

Entidade	**Serviços**
Login	Para autenticar os operadores. Neste pedido são enviadas as credenciais introduzidas manualmente pelo operador no formulário de login da aplicação e os dados identificadores do dispositivo móvel adicionados automaticamente ao pedido pela própria aplicação e que são usados pelo servidor para identificar o dispositivo móvel usado pelo operador. De notar que, caso esse mesmo dispositivo estiver a ser usado por outro operador, o outro operador deixa de ter um dispositivo móvel associado. Em resposta ao pedido, caso as credenciais fornecidas sejam válidas, é devolvido um *token* para ser usado durante a sua sessão de utilização da aplicação e também alguma informação interessante ao operador tal como datas de início e de fim do seu turno, data e hora da última vez que fez login.
Alertas	Para tratamento dos alertas. Para cada alerta individual: • Consulta dos dados completos do alerta com toda a informação necessária ao operador para dar início ao seu tratamento; • Consulta da lista de respostas dadas ao alerta pelo operador responsável; • Pedido de resposta a alerta: – Aceitar ou declinar; – Resposta escrita, com ou sem criação de ticket externo; Lista de alertas: • Todos - todos os alertas com tratamento concluído e por concluir • Operador - os alertas em que o operador é responsável; • Concluídos - os alertas já concluídos; • Oncall - os alertas por iniciar; • Ongoing - os alertas já iniciados mas não concluídos; • Missed - os alertas que o operador, como responsável, não atendeu.

Interface para Gestão de alertas Para visualizar e pesquisar informação de alertas através de:

- Listagem de alertas pesquisável através de filtros por data, cliente, operador responsável, origem e texto livre. Com possibilidade de eliminar alertas da lista principal, movendo-os para uma lista de arquivo, podendo também reverter o arquivamento. Cada alerta da lista tem ligação para uma página de visualização do alerta;
- Visualização de alerta para aceder a toda a informação relacionada com o alerta, nomeadamente:
  - Dados de criação do alerta;
  - Operador responsável;
  - Notificações enviadas;

  - Lista de mudanças de estado das notificações, com respectivas datas;
  - Lista de mudanças de estado do alerta, com respectivas datas;

Interface para Gestão de operadores Para visualizar, criar, editar ou eliminar contas de operadores.

- Na visualização, estão disponíveis as seguintes informações:
  - Listagens de alertas onde o operador terá sido responsável pelo tratamento;
  - Listagem de eventuais alertas que o operador, como responsável, terá não atendido;
  - Listagem de respostas dadas a alertas;
  - Dispositivo móvel em utilização, com ligação para uma página de gestão desse dispositivo;
  - Datas e escalonamento do turno que estiver em vigor.
- Criar e editar os dados do operador, nomeadamente, credenciais de acesso ao Cat|Mobile e credenciais de acesso ao RTIR.

Interface para Gestão de origens Para visualizar, criar, editar ou eliminar contas que dão acesso às ferramentas de origem de alertas poderem criar alertas no Cat|Mobile

- Na criação ou edição de contas de origem são definidas as credenciais de acesso ao Cat|Mobile. Existem dois tipos de ferramentas externas como origem de alertas, as ferramentas que criam alertas no Cat|Mobile por iniciativa própria e as ferramentas que apenas permitem consulta dos alertas de que dispõem. Caso a ferramenta de origem em causa seja do tipo consulta, a conta Cat|Mobile para essa ferramenta deve dispor das credenciais próprias para fazer o acesso externo. Esse acesso é essencial para fazer a consulta da ferramenta e assim obter os alertas para criar no Cat|Mobile em nome dessa origem. Caso a ferramenta externa de origem de alertas tenha capacidade de enviar os alertas por iniciativa própria ao Cat|Mobile, então deve ser criada uma conta simples de origem de alertas e fornecidas as credenciais de acesso para serem configuradas na ferramenta externa, assim como também os passos necessários e a estrutura das mensagens de alerta a ser enviadas à API de criação de alertas do Cat|Mobile.

6.3.2 Arquitectura Interna da Aplicação Móvel

A aplicação móvel Cat|Mobile é o interface dos operadores para tratar os alertas remotamente, através de um dispositivo Android. Os módulos que compõem a aplicação estão representados na Figura 6.13 e serão descritos nesta secção.

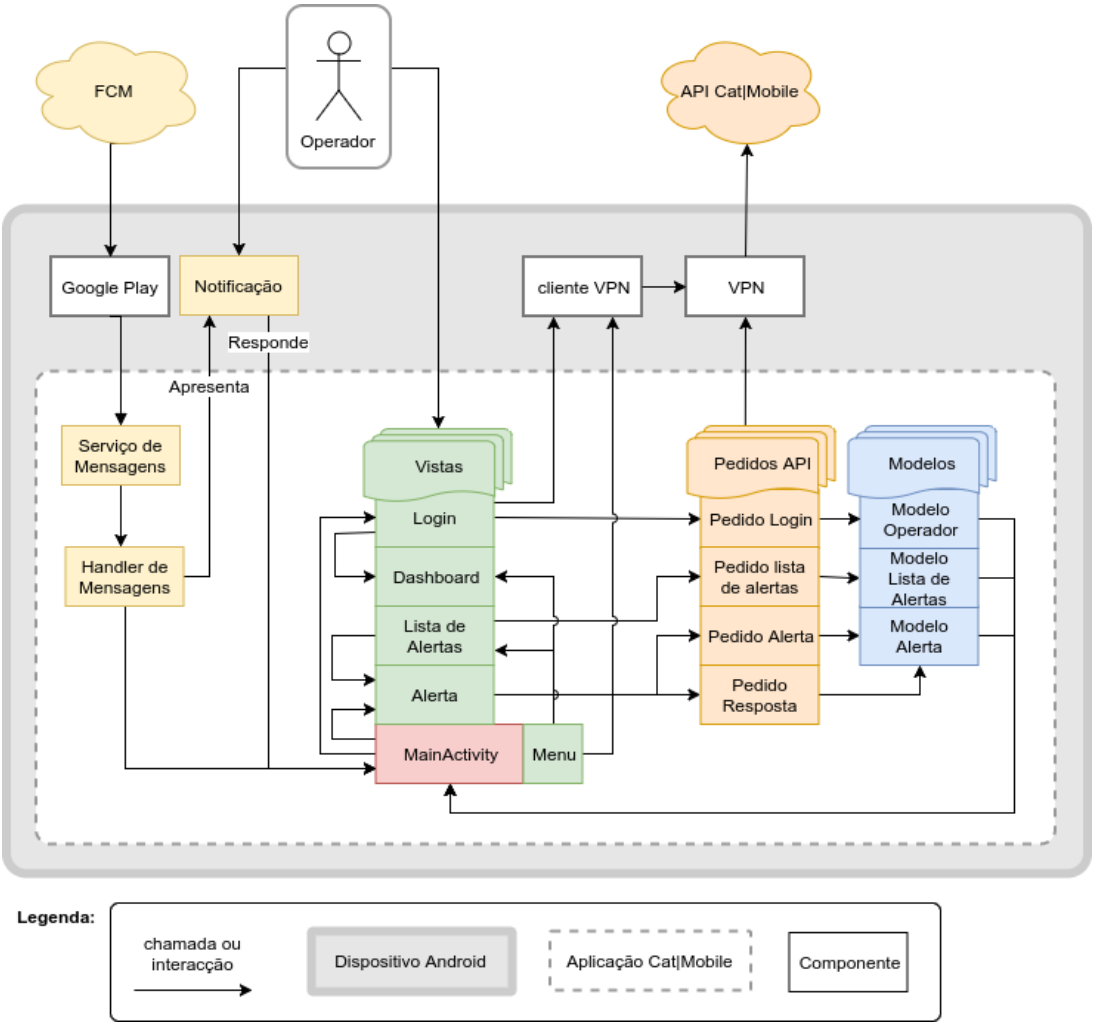


Figura 6.13: Diagrama de Arquitectura Interna da Aplicação Móvel.

Serviço de Mensagens

O *Serviço de Mensagens* é o componente que recebe as mensagens que chegam através do FCM. A arquitectura de alto nível deste processo está representada na Figura 6.14. As mensagens enviadas pelo servidor, destinadas ao dispositivo móvel, passam pelo FCM e são recebidas no dispositivo pelo componente Google Play Services que deverá estar instalado. O *Serviço de Mensagens* da aplicação móvel Cat|Mobile recebe as mensagens transportadas pelos serviços da Google e envia-as para o *Handler de Mensagens*, para serem filtradas, processadas e encaminhadas.

Handler de Mensagens

Conforme se pode ver na Figura 6.14, o objectivo do *Handler de Mensagens* é de triar as mensagens recebidas através do *Serviço de Mensagens*. Como tal, o *Handler de Mensagens* separa

as mensagens por categorias:

- Mensagens de primeira responsabilidade;
- Mensagens de tópicos (onde se encontram subscritos os operadores de segunda e terceira responsabilidade);
- Mensagens de actualização de turnos.

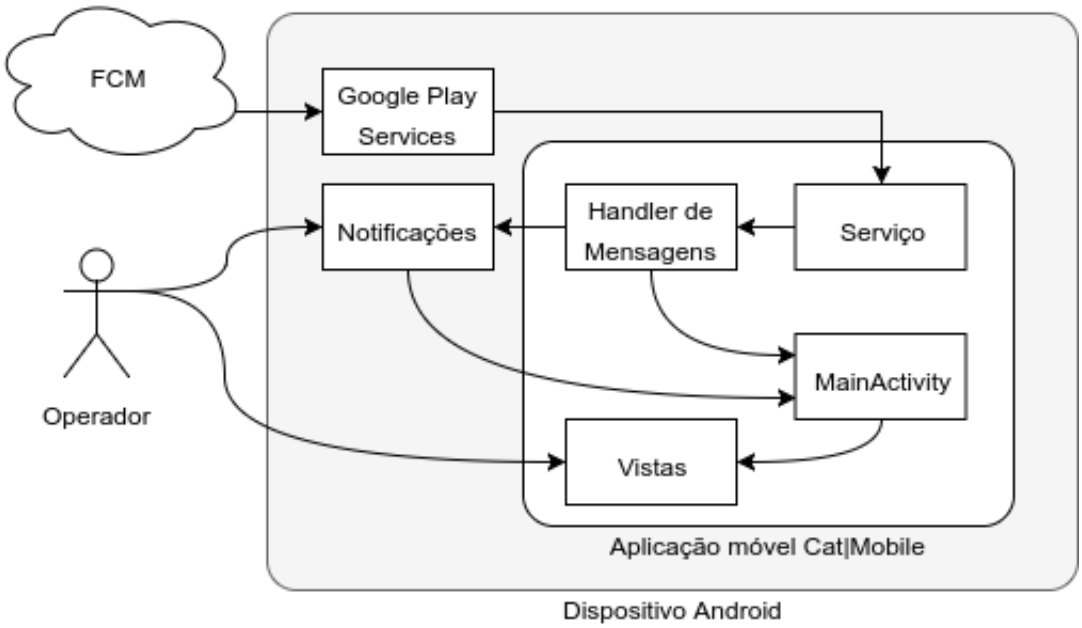


Figura 6.14: Diagrama de Arquitectura do processamento de mensagens FCM na Aplicação Móvel.

As mensagens de turnos são enviadas directamente para a *MainActivity* para actualizar os dados do operador. Com as mensagens de tópicos são criadas notificações silenciosas, tipo SMS. Com as mensagens de primeira responsabilidade, são criadas notificações ruidosas, tipo chamada telefónica, que necessitam de ser atendidas para parar. As notificações são criadas com possibilidade de interacção do operador. O operador pode interagir com as notificações e essa interacção é processada pela *MainActivity*. No caso da notificação vinda da mensagem de tópico, ao clicar nela, o operador será chamado a fazer *login*, para ver o alerta na aplicação. No caso da notificação de primeira responsabilidade, o operador tem opção de clicar para aceitar ou para recusar, em ambos os casos, também necessita de se autenticar e, finalmente, confirmar a resposta, ao ver o alerta.

Login

A vista *Login* para o operador se autenticar na aplicação é composta por um formulário onde o operador preenche o seu nome de utilizador e palavra passe. Para validar as credenciais, a aplicação necessita de comunicar com a API do servidor, conforme representado na Figura 6.15.

Como o servidor não tem acesso directo a partir da rede WAN, apenas por VPN, o módulo *Login* encarrega-se de activar a VPN através do cliente para VPN, instalado no dispositivo Android. Uma vez estabelecida a VPN, as credenciais são validadas através do módulo *Pedido Login*. Estando o *Login* válido, a aplicação é iniciada com o módulo *Dashboard*.

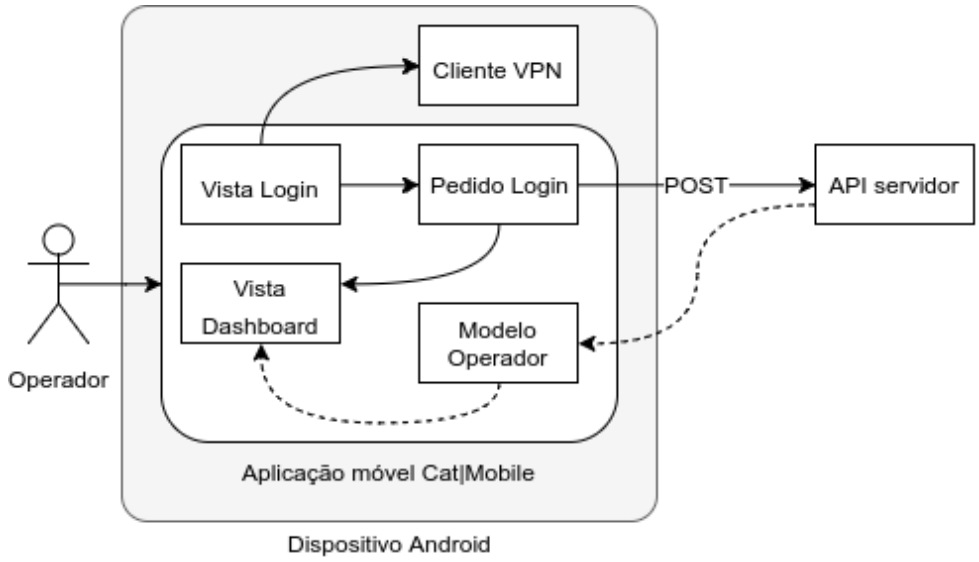


Figura 6.15: Diagrama de Arquitectura do Login da Aplicação Móvel.

Dashboard

Na *Dashboard* da aplicação são apresentadas informações diversas sobre o estado da aplicação, alertas recentes e o turno do operador, com data de início e data de fim. Informa também a data e hora do último *login* efectuado pelo operador.

Main Activity

A *MainActivity* é a base de todas as vistas, todos os dados que são transferidos entre vistas, passam pela *MainActivity*. Se o *Handler de Mensagens* receber uma mensagem com actualização do turno no operador, a *MainActivity* actualiza os dados do turno respectivo na instância do *Modelo de Operador* e refresca a vista onde esses dados são apresentados.

Lista de Alertas

A *Lista de Alertas* é uma vista onde são visíveis todos os alertas do Cat|Mobile que estão por concluir. Esta lista dispõe de parâmetros que permitem ao operador filtrar a lista de modo a facilitar a pesquisa por informação útil durante o tratamento de alertas. Esses parâmetros incluem, o cliente, a origem, o nível de severidade, o operador responsável e uma sequência de palavras. Conforme representado na Figura 6.16, a vista *Lista de alertas*, obtém os dados

a apresentar através do módulo *Pedido Lista de Alertas*. A lista de alertas é composta por alertas resumidos, isto é, apresentam apenas alguns campos, nomeadamente, o título, a origem, o cliente, a data, o nível de severidade, o estado e o nome do operador responsável. Os restantes dados do alerta são apresentados quando o operador selecciona um dos alertas da lista. Nesse momento é chamada a vista *Alerta* para apresentar o alerta por completo.

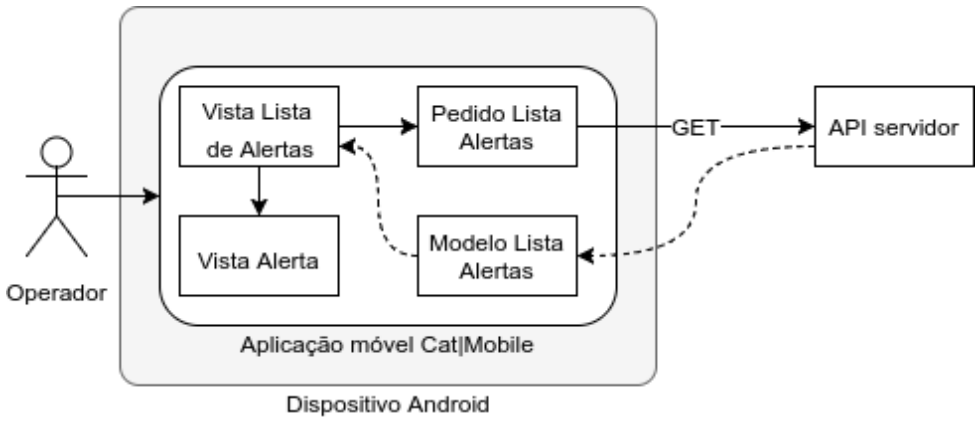


Figura 6.16: Diagrama de Arquitectura da Lista de Alertas da Aplicação Móvel.

Alerta

A vista de *Alerta* é iniciada sempre que se pretende visualizar e interagir com um alerta. Conforme representado na Figura 6.17, esta vista usa o módulo *Pedido Alerta* para solicitar ao servidor todos os dados do alerta. Estes dados incluem também a lista completa de eventuais respostas que o alerta possa já ter associadas. Na vista do alerta, o operador pode consultar os

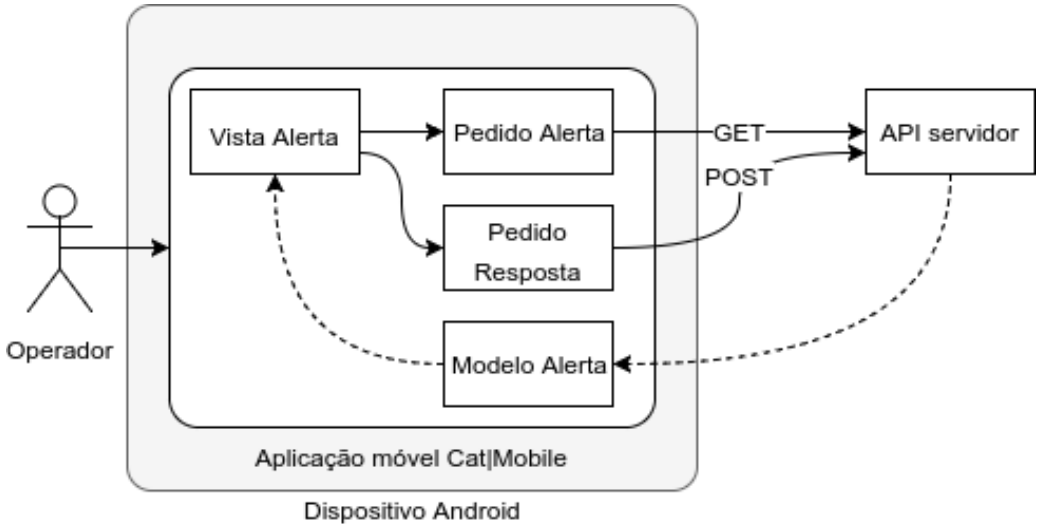


Figura 6.17: Diagrama de Arquitectura da vista de Alerta da Aplicação Móvel.

dados do alerta, o estado em que este se encontra e, caso seja o operador responsável, pode criar respostas ao alerta. Essas respostas são criadas no servidor através do módulo *Pedido Resposta*.

Menu

Conforme representado na Figura 6.18, o menu dá acesso directo a várias vistas, nomeadamente ao *Dashboard* e a várias *Lista de alertas* pré-filtradas. Também permite sair da aplicação. Ao sair invoca o fecho da VPN através do cliente VPN. O menu encontra-se acessível a partir de todas as vistas da aplicação, incluindo aquelas que não têm acesso directo a partir do menu, como é o caso da vista *Alerta*.

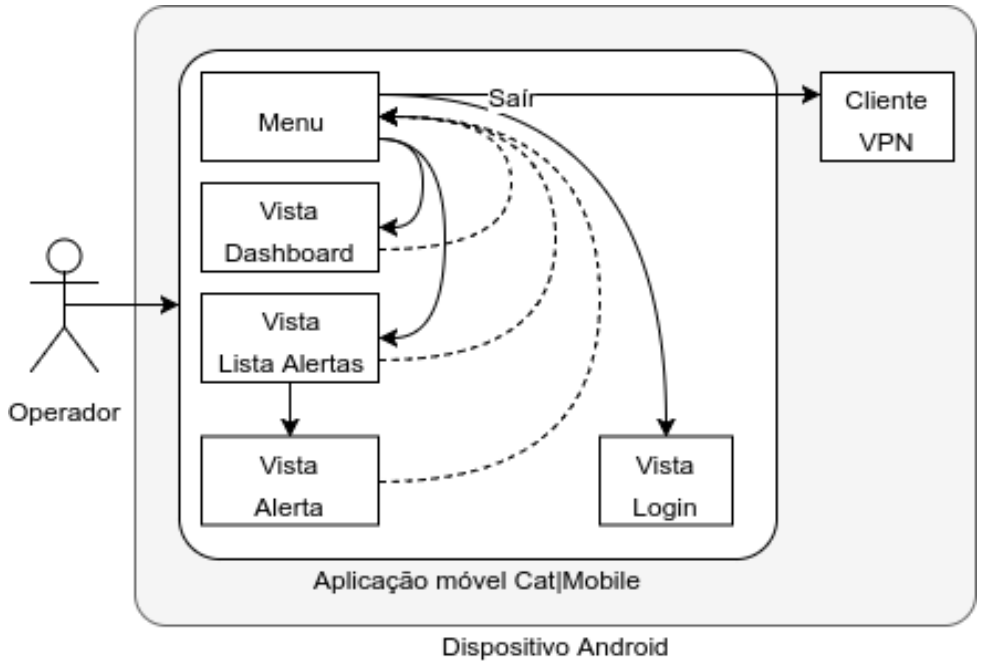


Figura 6.18: Diagrama de Arquitectura do Menu da Aplicação Móvel.

Pedido Login

O *Pedido Login* é usado pelo módulo vista *Login* para validar as credenciais introduzidas no formulário respectivo. O pedido é dirigido ao *endpoint* **/api/v1/token** com o verbo POST em que, no *Body* são enviadas, para além das credenciais, também o identificador do dispositivo móvel, para que o servidor reconheça o dispositivo e actualize o dispositivo actual do operador se assim for necessário. A resposta enviada pelo servidor é guardada numa instância de *Modelo de Operador*.

Pedido Lista de Alertas

O *Pedido Lista de Alertas*, que serve o módulo *Lista de Alertas*, é direccionado ao *endpoint* **/api/v1/alerts** com o verbo GET, com ou sem parâmetros, para obter uma lista de alertas, com ou sem filtro. O resultado do pedido é armazenado numa instância do *Modelo Lista de Alertas*.

Pedido Alerta

O *Pedido Alerta*, que serve o módulo vista *Alerta*, é direccionado ao *endpoint* **/api/v1/alerts/(id)** com o verbo GET e o ID do alerta para obter os dados do alerta. Os dados do alerta ficam guardados numa instância do *Modelo Alerta*

Pedido Resposta

O *Pedido Resposta* é usado no módulo *Alerta* para o operador responsável dar resposta ao alerta. Isto é, se aceita o declina o alerta e outras respostas possíveis. Este pedido é direccionado ao *endpoint* **/api/v1/alerts/(id)/replies/** com o verbo POST para criar a resposta associada ao alerta identificado pelo seu ID. Depois de criada a resposta ao alerta, o servidor devolve todo o alerta actualizado que fica guardado na instância de *Modelo Alerta*. O servidor devolve todo o alerta, porque a resposta criada pelo operador pode ter alterado o estado do processo do alerta levando a que alguns campos possam ter sido também alterados.

Modelo Operador

O *Modelo Operador* guarda na sua instância, a resposta do servidor ao pedido de *Login*, nomeadamente, o *token*, o turno e a momento do último *login*. O token será usado em todos os restantes pedidos durante a sessão do operador. Os restantes dados existem somente para informação ao operador.

Modelo Lista de Alertas

O *Modelo Lista de Alerta*s guarda a lista de alertas resumidos obtidos através do *Pedido de Lista de Alertas* ao servidor. Este modelo consiste em uma lista de instâncias de *Modelo Alerta* com apenas alguns campos preenchidos, nomeadamente, o título, a origem, o cliente, a data, o nível de severidade, o estado e o nome do operador responsável.

Modelo Alerta

O *Modelo Alerta* guarda na sua instância, a resposta do *Pedido Alerta* obtida do servidor. Nesta instância são guardados todos os dados do alerta e ainda a lista de eventuais respostas associadas ao alerta. A lista de respostas é constituída por instâncias de *Modelo Resposta*.

Modelo Resposta

O *Modelo Resposta* guarda cada instância de resposta associada a alerta. Estas respostas são obtidas do servidor através do *Pedido Alerta* que, para além do alerta, também obtém a lista de respostas associadas.

6.4 Conclusão

Neste capítulo foi detalhada a arquitectura da aplicação de servidor e da aplicação móvel, numa primeira abordagem tendo em conta o enquadramento em que se encontrá depois de implementada, em produção e, na segunda abordagem relativamente à parte interna, que depois foi desenvolvida. Esta perspectiva das aplicações antes da implementação é bastante importante no sentido em que será o molde a seguir para que a implementação resulte como previsto.

Capítulo 7

Implementação

Neste capítulo é descrita a implementação do Cat|Mobile. O capítulo está subdividido em duas partes. Primeiro é descrita a implementação da componente Servidor e, por fim, a componente Aplicação Móvel.

7.1 Implementação do Servidor

Terminado o desenho e especificação da arquitectura do Servidor, pode-se passar à implementação dos módulos que o compõem. O mapa de correspondência entre os requisitos e os módulos que os implementam pode ser consultado no Apêndice A.

7.1.1 Gestão de Utilizadores

A gestão de utilizadores consiste nas funcionalidades de registar, editar e listar utilizadores do sistema. Os utilizadores do sistema são os operadores, origens e administrador. Os operadores são os utilizadores que acedem ao sistema através da aplicação móvel, as origens são as contas de acesso à criação de alertas, usadas pelas ferramentas de origem de alertas e o administrador é o utilizador que acede ao interface de gestão Web. Para este módulo, as tarefas a implementar são as seguintes:

- Grupos de utilizador, para Origens e de Operadores;
- Credenciais de consulta a ferramentas externas de alertas.

Os grupos de utilizador são uma funcionalidade nativa do Django, que permite categorizar utilizadores e aplicar permissões ao grupo em vez de a cada utilizador individualmente [29]. Geralmente as definições de grupos são persistidas em base de dados, durante a configuração da aplicação Django, através do interface de administração nativo do Django [30]. No entanto, não está prevista a utilização desse interface de administração nativo no Cat|Mobile durante a utilização do sistema. Apenas eventualmente para criação de novos administradores ou para

eliminar dados definitivamente. Por outro lado, como se pretende que a aplicação esteja pronta a funcionar, pré-configurada a partir do momento que for instalada num servidor, a configuração inicial de definições como estas, terá de ser feita programaticamente pela própria aplicação. Para isso utiliza-se uma funcionalidade do Django para a persistência de parâmetros de inicialização chamada Migrations. Esta funcionalidade é também usada para inicializar o próprio Django quando instalado pela primeira vez, numa base de dados vazia. As Migrations são a forma de o Django propagar, no esquema da base de dados, as alterações aos modelos da aplicação, por exemplo, para adicionar campos ou eliminar um modelo [31]. Para além de registar alterações no esquema da base de dados, as Migrations permitem também executar código arbitrário através da classe RunPython [32]. Aproveitando o facto de que as Migrations fazem parte dos passos de instalação da aplicação no Servidor, ao usar RunPython durante as Migrations, abre-se a possibilidade de fazer todas as pré-configurações necessárias para obter o estado inicial desejado do Cat|Mobile. Neste caso, RunPython será usado nas Migrations para a criação dos grupos de utilizador, Origens e Operadores.

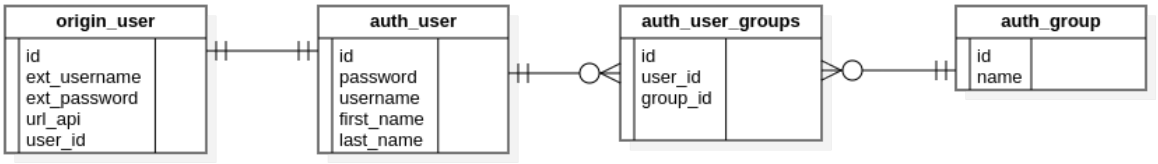


Figura 7.1: Diagrama de Modelo ER dos utilizadores do grupo Origens.

Os campos de credenciais que se pretendem guardar junto do dados das Origens, são simples campos de texto, para depois serem usados nos pedidos às APIs das ferramentas externas. Conforme se pode ver na Figura 7.1, o Django dispõe nativamente de alguns campos associados ao utilizador, tais como *first_name* e *last_name* que são comuns a todos os utilizadores. Para se adicionar campos extra, tem de se criar uma nova classe *Model* [33] com esses campos e, nessa classe, criar uma relação de 1 para 1 com o modelo de User [34] nativo. Com esta técnica, denominada *Profile* [35], o Object-relational mapping (ORM) do Django permite fazer *queries* aos novos campos com grande facilidade a partir dos objectos da classe User que tenham essa nova classe instanciada [36]. Para instanciar a nova classe sempre que uma nova Origem for instanciada e assim poder guardar as credenciais extra já referidas, é usada outra funcionalidade do Django, chamada *Signals*. O Django inclui um emissor de sinais que permite notificar classes desacopladas quando determinados eventos ocorrem durante a execução da aplicação [37]. Os sinais do Django são a forma encontrada para instanciar a nova classe aos utilizadores do grupo Origem. Para isso a aplicação deve ficar à escuta do sinal de criação de utilizadores do grupo Origens. No momento da criação de um utilizador, este ainda não pertence a grupo algum, por isso o sinal que identifica a criação de um utilizador Origem, será o sinal de quando

o utilizador é adicionado ao grupo Origem e não o sinal de quando o utilizador é criado, pois isso iria adicionar os campos extra a todos os utilizadores mesmo fora do grupo Origens. O sinal que deverá ser escutado é o sinal post_save à classe User, que é o momento após o objecto ter sido modificado devido a, por exemplo, ter sido adicionado a um grupo. No momento do processamento do sinal, é ainda verificado se o utilizador em causa pertence ao grupo Origens, e somente se for o caso, então, instância a nova classe com os campos extra, conforme pretendido.

7.1.2 Gestão de alertas

O módulo para gestão de alertas consiste na implementação do modelo para os alertas. Esse modelo é usado para a persistência dos dados de cada alerta e respectivos relacionamentos que este tem com outras entidades do Cat|Mobile. Assim, para este módulo foi implementada a seguinte tarefa:

- Modelo de Alerta.

A classe de persistência para os alertas, representada na Figura 7.2, é a classe usada para instanciar cada alerta criado no sistema pela API de criação de alertas ou pela tarefa cron de criação de alertas. Ambas as formas de criação de alerta estão associadas a um utilizador do grupo de origens, cuja referência ficará registada na instância com a designação origin_id. O cliente onde ocorreu o alerta é uma informação enviada pela origem, no corpo do alerta, que também ficará registada com a designação client. Ambos o cliente e a origem são dados fixos, nunca alteráveis durante o tempo de vida do alerta. No entanto o campo de referência ao operador responsável pelo alerta operator_id, esse é variável durante o processamento do alerta, enquanto o sistema andar a procurar um operador disponível.

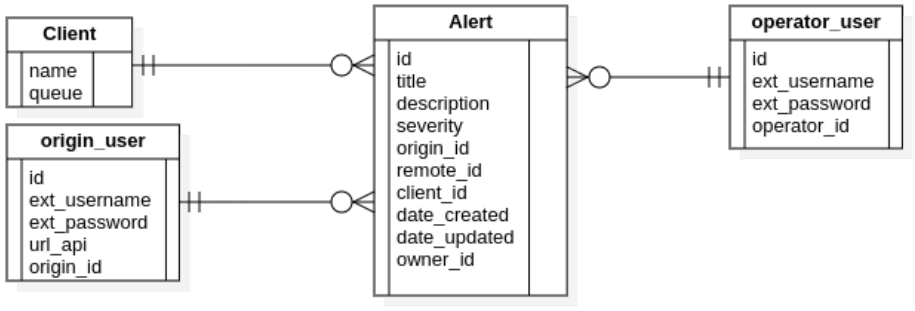


Figura 7.2: Diagrama de Modelo ER de Alerta.

7.1.3 API para criação de alertas

O módulo de API para criação de alertas consiste na implementação de uma API REST para que ferramentas externas ao sistema possam criar alertas em formato HTTP. Como a criação de alertas consiste na instanciação da classe Alert, a API terá o foco nessa classe. Este módulo teve as seguintes tarefas implementadas:

- API REST;
- Autenticação;
- Permissão;
- endpoints da API.

A implementação da API REST consiste na implementação dos verbos HTTP escolhidos para API e todo tratamento associado a cada verbo, tal como a definição de campos de *Header* e *Body* dos pedidos, assim como os HTTP status codes [38] enviados junto das respostas, nos variados casos que possam ocorrer. Para acelerar a implementação da API, foi usada a Django REST framework (DRF) [1]. Esta *framework* possui várias classes implementadas para os diversos casos de uso que pode ter uma API REST. O programador apenas tem de escolher a(s) classe(s) que mais se adequam aos seus requisitos e usar polimorfismo estendendo essas classes, sobrepondo os métodos dos verbos HTTP que se pretendem personalizar. Para o caso da criação de alertas que se pretende implementar, a classe DRF mais indicada seria *CreateAPIView* [39]. Esta classe, implementa somente o verbo POST que, numa primeira análise, surge como ideal para a funcionalidade única de criar alertas, que se se pretende dar às ferramentas de origem de alertas. Contudo, pretende-se obter uma outra funcionalidade do *endpoint* dos alertas, pois este *endpoint* será partilhado com a API da Aplicação Móvel, descrita na Secção 7.1.14. A API da Aplicação Móvel irá usar este *endpoint*, não para criar alertas, mas para filtrar listas de alertas. Por este motivo, será necessário ter também implementado o verbo GET para o conseguir. Felizmente que DRF dispõe da classe *ListCreateAPIView* que implementa este requisito. No fundo, a classe *ListCreateAPIView* é uma classe constituída por *Mixins* [40], que implementa em simultâneo, os métodos das classes *CreateAPIView*, já aqui referida, e da classe *ListModelMixin* que será descrita na Secção 7.1.14 durante a implementação da filtragem de alertas.

Para definir o modelo de dados dos objectos a enviar à API, DRF dispõe de classes para serialização baseadas nas classes *Model* do Django que, por sua vez, são usadas como ORM do próprio. A classe *ModelSerializer* de DRF faz a conversão direta das classes *Model* do Django, mapeando os campos das classes em atributos REST. Para alterar os campos disponíveis em REST, basta criar uma nova classe com herança de *ModelSerializer* e sobrepor a indicação de quais os campos incluídos [41]. Apesar da classe *Alert* possuir numerosos campos, no caso da criação de alertas, pretende-se limitar os campos possíveis de definir através dos pedidos apenas

aos referidos na Tabela 6.2, ou seja, *Título*, *Descrição*, *Severidade* e *Cliente*.

Para identificar a origem de cada alerta, é necessário que as ferramentas de Origens sejam autenticadas em todos os pedido que fazem à API. Para isso, a DRF dispõe de uma técnica de autenticação baseada em tokens [42]. A *TokenAuthentication* é uma classe DRF que aceita o POST de *username* e *password* de utilizadores existentes e devolve o respectivo *token*. Esse *token* serve para ser usado posteriormente em novos pedidos que necessitam de autenticação. O *token* é usado no campo *Authorization* do *Header* no pedido HTTP e permite ao servidor identificar a que utilizador pertence o *token*, sabendo assim de quem se trata.

Para garantir que apenas os utilizadores do grupo Origens criam alertas através da API, foi necessário fazer override do método post da *CreateAPIView* [39], anteriormente referida. Nesse método, a lógica para dar permissão de criação do alerta, consiste em criar o alerta apenas se o utilizador que faz o pedido, pertence ao grupo Origens. Caso contrário, devolve o HTTP status *401 Unauthorized*.

A criação dos *endpoints* da API para criação de alertas consiste na definição do URL para onde vão ser enviados os pedidos vindos das ferramentas externas de origem de alertas e também do *endpoint* para obtenção do *token* de autenticação. Como boa prática na construção de Web APIs, a estrutura do URL consiste em referir o recurso usado e a versão da API. A versão da API permite que, em futuras versões da API, se possa garantir retro-compatibilidade com as versões anteriores, mantendo-as em actividade. Para fazer a ligação entre os URL e os métodos da API, o Django dispõe do *URL Dispatcher* [43]. Este sistema consiste, basicamente, no mapeamento das *strings* que definem o URL com as *Views* que respondem ao URL. As *Views*, no caso do Django, são os processamentos de pedidos HTTP. Estas podem ser simples funções directas ou classes estruturadas, chamadas *Class-based views*, com diversos métodos para responder a vários tipos de pedido num mesmo URL [44]. No caso da API para criação de alertas, a *View* usada é a que foi definida no parágrafo de Implementação da API. A API definitiva encontra-se detalhada na Tabela 7.1.

Tabela 7.1: Endpoints da API para Criação de Alertas.

URL	**Método**	**Descrição e *URL Dispatcher***
/api/v1/token	POST	Obter *token* de autenticação.
path('api/v1/token/', GetToken.as_view()		
/api/v1/alerts	POST	Criar alertas.
path('/api/v1/alerts', ApiListCreateAlertsView.as_view()		

7.1.4 Criação de alertas por consulta externa

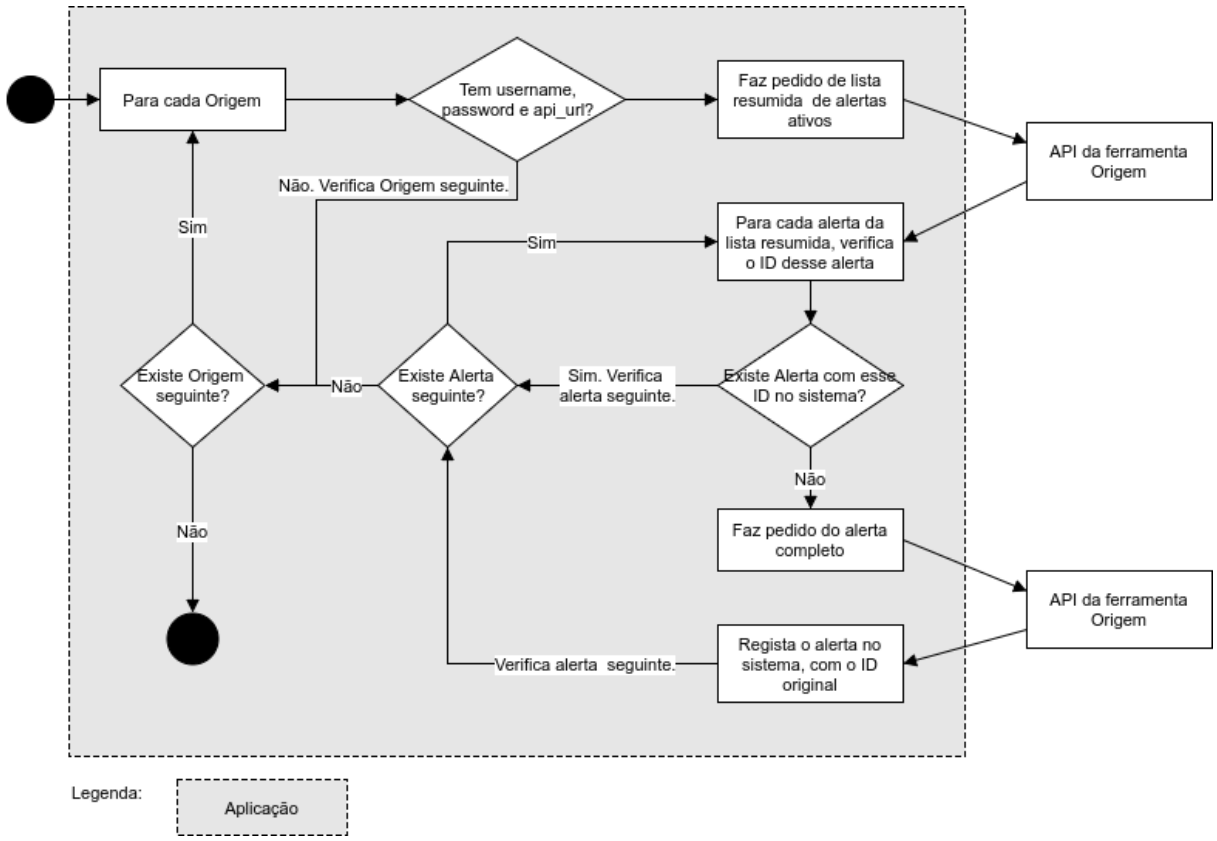


Figura 7.3: Fluxograma da criação de alertas por consulta externa.

A criação de alertas através da consulta ao RTIR é um processo iniciado regularmente pelo cron. O processo, representado pela Figura 7.3, consiste em executar pedidos à API do RTIR com o objectivo de obter os novos alertas disponibilizados. Para isso, são iteradas todas as origens registadas no sistema e consideradas apenas as que têm definidos os campos *username_ext*, *password_ext* e *url_api*, pois somente essas obtêm os seus alertas por consulta ao RTIR. As origens que não têm esses campos definidos, são origens que fazem a criação de alertas por iniciativa própria e não têm API de consulta. Para concretizar o módulo, foi necessário implementar as seguintes tarefas:

- Obtenção de alertas do RTIR;
- Tarefa para o cron.

Para cada uma das origens de consulta, é então, feito um primeiro pedido para obter uma lista sumária de alertas e onde é verificado se o alerta já existe no Cat|Mobile. Para isso é usada a biblioteca *RTIR4REST* [45], que dispõe do método *get_all_new_open_tickets_idlist* para obter uma lista dos IDs de todos os novos alertas no RTIR. Essa lista é então usada para filtrar pelo campo *remote_id* dos alertas, quais os que ainda não existem no Cat|Mobile. Para os que ainda

não existem, é feito um segundo pedido pelo ID do alerta novo para obter todos os dados do alerta e criar o alerta no Cat|Mobile com todos os mesmos atributos também usados pela API de criação de alertas. Durante o registo do alerta é também registado o ID do alerta na ferramenta externa, no campo *remote_id*, para além dos restantes campos, para evitar repetições de alertas, quando o cron executar o processo de novo.

Para a execução do processo através do cron, é necessário configurar o crontab, com a regularidade com que se pretende executar o comando e também indicar qual o comando a executar. A regularidade fica ao critério do administrador, no entanto, o comando que executa o processo tem de estar pré-definido. Para executar métodos da aplicação, o Django dispõe da funcionalidade *Custom Management Commands* [46]. Desta forma é possível invocar um método da aplicação Django a partir do cron ou mesmo da linha de comandos do sistema operativo e assim interagir com os dados da aplicação enquanto ela está a ser executada.

7.1.5 Gestão de notificações

A Gestão de notificações é o módulo onde está definido o modelo de persistência das notificações. Como referido no Capítulo 2, notificações são as mensagens enviadas pelo servidor aos dispositivos móveis com informação sobre alertas que devem ser tratados pelos operadores responsáveis. A persistência destas notificações é essencial para manter histórico das notificações enviadas e do seu estado de recepção. Este módulo necessitou da seguinte tarefa implementada:

- Modelo para notificações.

Para persistir cada notificação, é necessário um modelo que interligue as várias entidades intervenientes. Este modelo, representado pela tabela *Notification* da Figura 7.4, contém o alerta que se pretende notificar, o operador destinatário, o dispositivo móvel em uso pelo operador, o turno em que se encontra, data de criação e o tempo máximo dado ao operador para responder à notificação. Alguns destes dados, nomeadamente o turno e o dispositivo móvel são redundantes

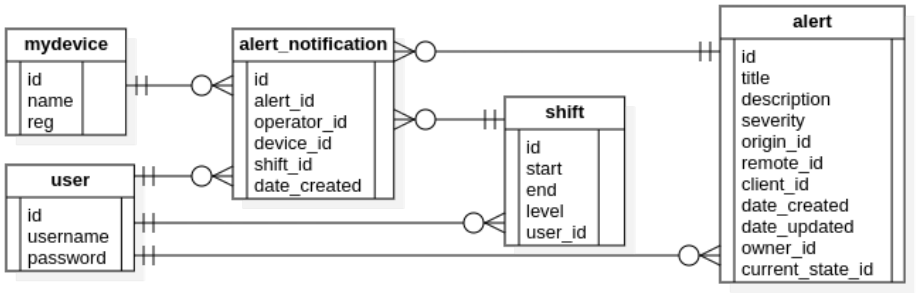


Figura 7.4: Diagrama de Modelo ER das Notificações.

no momento em que é registada a notificação, pois podem ser consultados através da relação que

estes têm com a tabela User, no entanto, estes dados são temporários, pois tanto o turno como o dispositivo são variáveis. Portanto, para manter o registo histórico dos factos é necessário registar quais os dados a ser usados naquele momento.

7.1.6 Gestão de Escalonamento

O objectivo do módulo Gestão de Escalonamento é permitir a definição do operador destinatário como parte do processo de envio de notificações. A escolha do operador destinatário de uma notificação passa por ter turnos registados onde estão definidos os operadores responsáveis a cada momento. Estes turnos são consultados pelo processo de escolha do operador destinatário a quem enviar cada notificação. Este registo consiste em ter definido diariamente, um conjunto de operadores, disponíveis para tratar alertas numa escala de responsabilidade. Para isso deverá ser criada uma classe que terá funcionalidade de agenda, em que são definidos operadores escalonados para determinados períodos de dias. Essa agenda será preenchida pelo administrador através do interface de gestão e usada pelo Gestor de Notificações para escolher o destinatário de cada notificação. Para este módulo, foi necessário implementar a seguinte tarefa:

- Modelo para turnos.

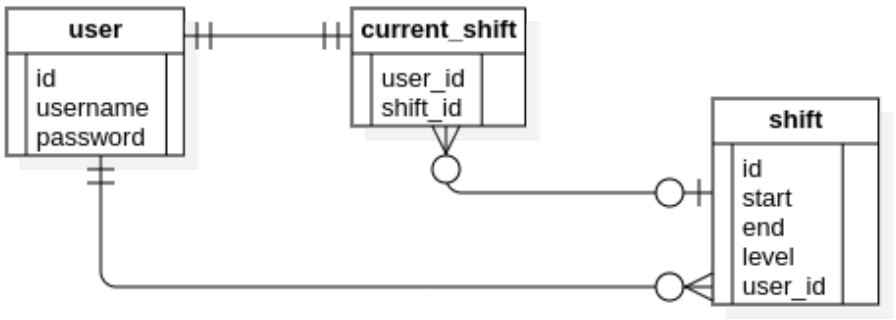


Figura 7.5: Diagrama de Modelo ER dos Turnos de Operadores.

O modelo que persiste os turnos de operadores é uma estrutura onde é definido o período do turno, com data inicial e data final, a escala de responsabilidade e o operador em questão. Para fazer o agendamento de turnos, estes devem ficar persistidos em tabela própria da base de dados. O turno actual de cada operador será o turno em que a data actual esteja compreendida entre a data inicial e final. Para evitar ocupar a base de dados com consultas morosas, foi também criada uma tabela ligada ao operador onde fica registado o seu turno actual, indexando rapidamente os dados do turno sem ter de percorrer toda a tabela, sempre que se quiser saber o turno actual do operador. Essa tabela, referida na Figura 7.5 é criada só para os utilizadores do

grupo Operadores usando a funcionalidade *Signals* do Django [37], já referida anteriormente em 7.1.1 no parágrafo *Definição de campos para credenciais*.

7.1.7 Actualização de Turnos

Para actualizar o turno actual de cada operador, terá de ser implementado um método que verifique se o turno actual ainda está em vigor e, caso contrário, consulte a base de dados em busca de um outro eventual turno desse operador, que inclua a data actual. Dado que os turnos têm resolução diária, voltamos a recorrer ao *cron*, como referido na Secção 7.1.4, desta vez para executar o método de actualização de turnos diariamente, conforme representado pela Figura 7.6. Este módulo necessitou das seguintes tarefas implementadas:

- Actualização de turnos;
- Comando para tarefa de cron para actualização de turnos;
- Envio de mensagem móvel com actualização de turno.

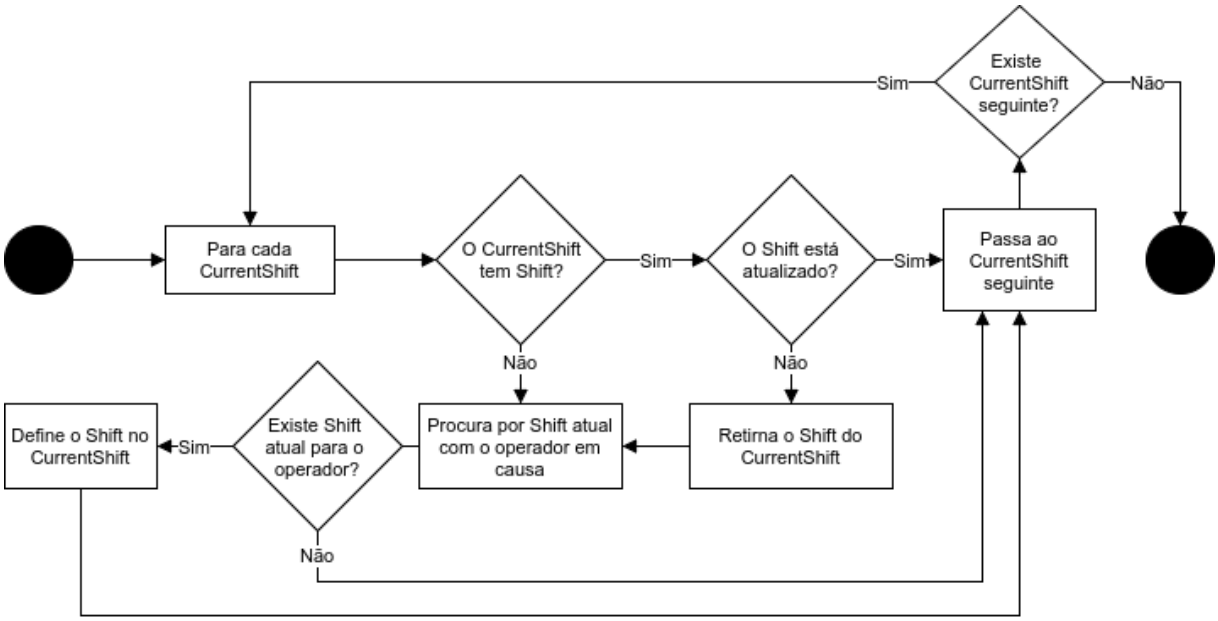


Figura 7.6: Fluxograma da Actualização dos Turnos de Operadores.

Como cada operador tem o seu turno actual numa instância de CurrentShift, a forma mais eficiente será de percorrer directamente todas as instâncias de CurrentShift, em vez de percorrer todos os operadores para obter o respectivo turno, pois evita-se a correlação de uma tabela.

Para criar o comando executável a partir do cron e que executa o actualização de turnos, foi utilizada a técnica descrita na Secção 7.1.4, para a criação de alertas com o cron.

A tarefa de enviar mensagem móvel com actualização de turno, por usar a mesma tecnologia que o envio de notificações, estará descrita na Secção 7.1.10, no parágrafo sobre o método *update_shift*.

7.1.8 Gestão de Dispositivos Móveis

Como referido anteriormente, no capítulo Análise em 2, os dispositivos móveis usados pelos operadores, podem ser partilhados. Para isso, o módulo Gestão de Dispositivos Móveis comporta um modelo de persistência para dispositivos móveis e uma tabela associada a cada operador com definição de dispositivo actual, de forma idêntica à usada em 7.1.6 para registo do turno actual. Este módulo necessitou das seguintes tarefa implementadas:

- Modelo para dispositivos móveis;
- Partilha de dispositivos.

A classe *MyDevice*, representada na Figura 7.7, regista os dados identificadores de cada dispositivo para que este possa ser contactado no envio de notificações. Para permitir o conceito de *pool*. Neste conceito, existe um conjunto de elementos onde alguns dos quais estão a ser usados. Para isso, foi criada a classe auxiliar *CurrentMyDevice* que regista o dispositivo actual de cada operador e que é actualizada sempre que o dispositivo é utilizado por outro operador.

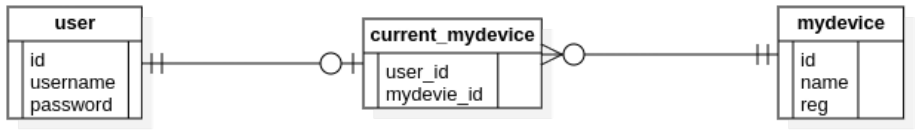


Figura 7.7: Diagrama de Modelo ER dos Dispositivos Móveis dos Operadores.

Para haver partilha de dispositivos, tem de ser possível actualizar o dispositivo actual de cada operador. Essa actualização deverá ser feita quando o operador usa a aplicação móvel no dispositivo, isto é, quando estiver autenticado na aplicação. Como representado na Figura 7.8, percebe-se que, quando um dispositivo troca de operador, este deve passar a ser o dispositivo actual do novo operador enquanto que o operador anterior ficará sem dispositivo actual.

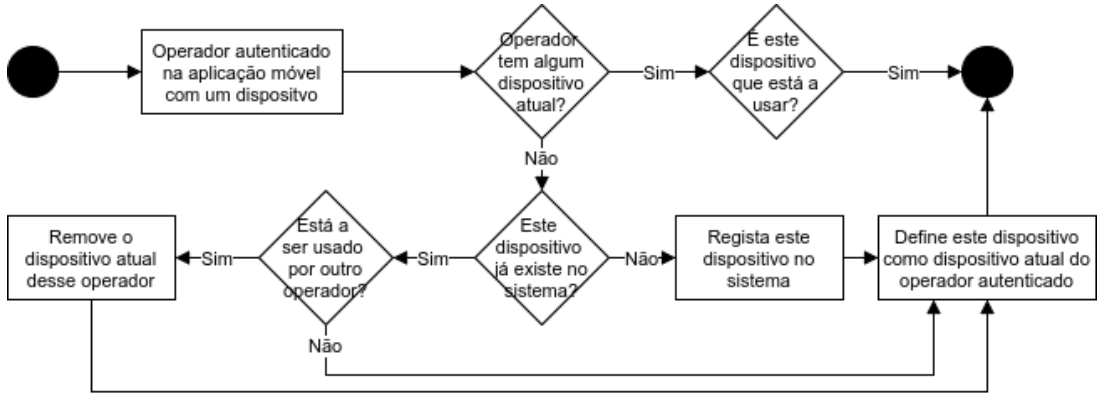


Figura 7.8: Fluxograma da Actualização dos Dispositivos Móveis dos Operadores.

7.1.9 Processamento de Alertas

O *Processamento de Alertas* é o módulo que tem por objectivo encontrar um operador disponível para ficar como responsável do tratamento de cada novo alerta recebido. Desde o momento em que o alerta chega ao sistema até o seu tratamento ser dado por concluído, o alerta passa por vários estados. Esses estados, já referidos na Figura 6.11, possuem lógica própria em cada uma das suas transições. O estado *Enviando Notificação* corresponde à invocação do módulo *Processamento de Notificações* que, por sua vez, também é composto por um mecanismo de estados próprio. Assim, o *Processamento de Alertas* é iniciado sempre que um novo alerta é criado no sistema. Para iniciar o processo usou-se a funcionalidade *Signals* do Django [37], descrita na Secção 7.1.1. O *signal* ficou associado ao evento de persistência do alerta em base de dados. Como esse evento de persistência faz parte de outros processos que ficam a aguardar resposta, como é o caso dos pedidos HTTP, a intercalação com o processo de alertas e respectiva máquina de estados iria colocar outros processo em espera até que terminasse a máquina de estados. Ora, isso seria insustentável pois daria *timeout* em todos os pedidos HTTP. Para resolver esse problema, a instanciação da máquina de estados, durante a criação de alertas, é feita numa nova *thread* independente da *thread* onde está a ser criado o alerta. Assim, tal como representado na Figura 7.9, o processo de criação de alerta pode continuar como previsto e

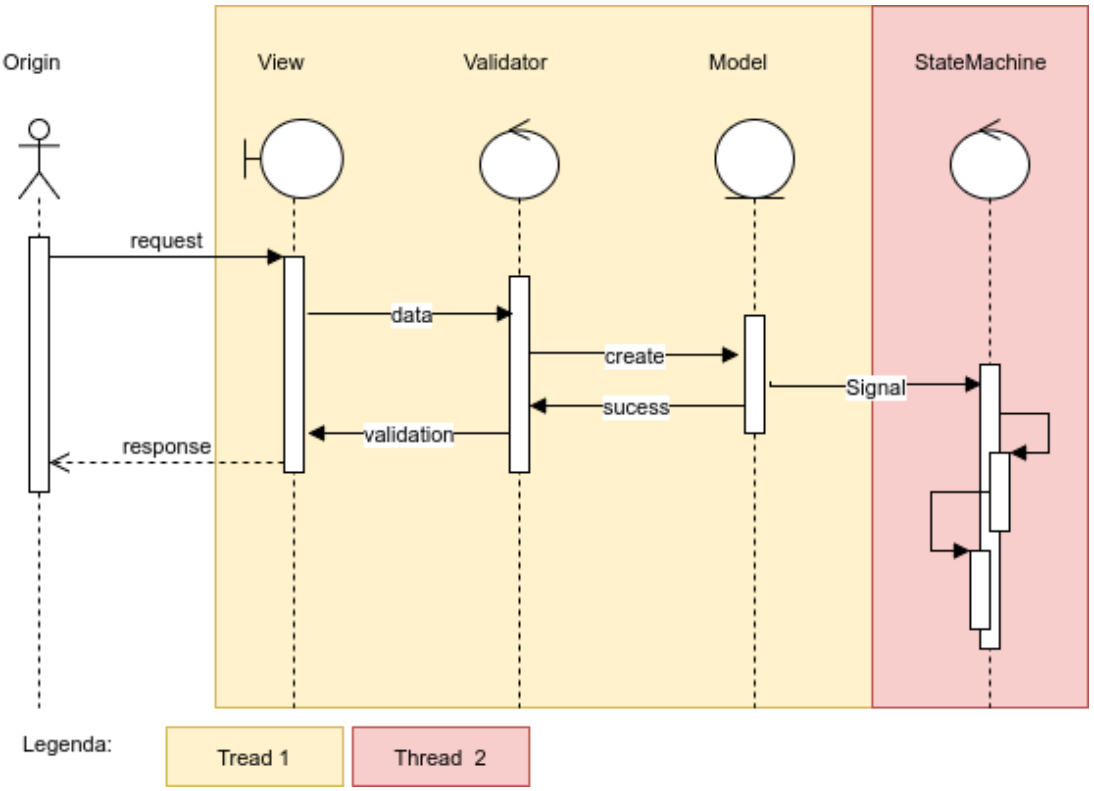


Figura 7.9: Diagrama de Sequência da criação de alertas e de inicialização de máquina de estados.

devolver respostas dentro do tempo esperado, enquanto que a máquina de estados segue a sua execução de forma independente e podendo demorar todo o tempo de que necessita. Para este módulos, foi necessário implementar as seguintes tarefas:

- Máquina de estados de alerta;
- Estado *Selecciona Operador*;
- Estado *Enviando Notificação*;
- Estado *Aguarda visualização na app*;
- Estado *Aguarda aceitação*;
- Estado *Tratamento*;
- Estado *Conclusão*.

A máquina de estados é um processo em que cada estado executado fica a aguardar por um resultado que permita avançar para o estado seguinte, seguindo um caminho com opções pré-definidas. Em caso de interrupção inesperada do sistema, é necessário que a máquina mantenha o estado em que se encontrava antes da interrupção, sem reiniciar. O reiniciar da máquina de estados de alerta, iria retomar o processo, desde o primeiro estado, com o re-envio da primeira notificação, provavelmente causando confusão na equipa de operadores. Para resolver este problema e para manter a consistência da máquina, é necessário persistir os estados da máquina de modo que esta possa ficar pausada em cada estado com residência em armazenamento física e não apenas em memória volátil. Esta persistência de estados permite também manter histórico de todas as fases do tratamento dos alertas, permitindo ao administrador, posteriormente, fazer análises do tempo decorrido entre alterações de estado. O modelo de persistência dos estados de alerta está representado na Figura 7.10.

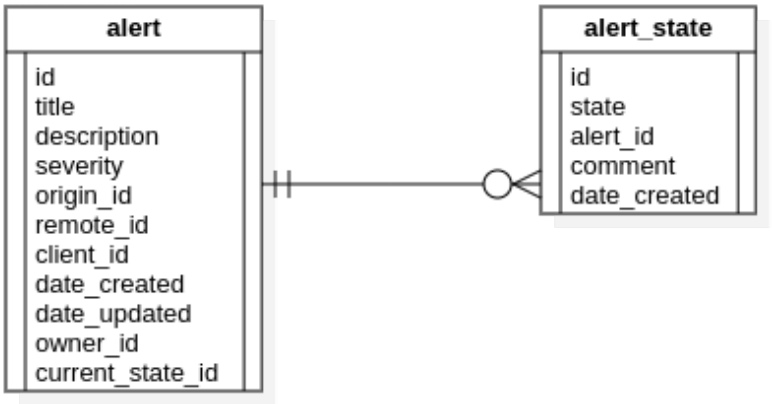


Figura 7.10: Diagrama de Modelo ER de Estados do Alerta.

Para a implementação da máquina de estados, foi usada uma classe abstracta de máquina de

estados com base em *statemachine.py* e classe testadora *statemachine_test.py* de David Mertz [47]. A implementação de David Mertz não inclui a persistência de estados que se pretende nesta solução. Para isso, foi criada uma classe de modelo adicional a ser persistida com a *string* de designação de cada estado. Esta designação é usada para identificar e persistir o estado a cada nova alteração. Assim, o novo modelo de persistência de estados, como representado na Figura 7.10, regista o estado actual do alerta, com a data em que entrou nesse estado e um campo de comentário para adicionar informação adicional sobre o motivo da mudança de estado, útil para o administrador supervisionar o histórico de eventos. Para actualizar o estado da máquina e retomar o seu estado a partir da base de dados, a máquina é sempre inicializada com a instância do alerta que se pretende processar. Durante a inicialização, representada na Figura 7.11 , é obtida a designação de estado do alerta que se encontra no respectivo campo do alerta. Com essa designação identificadora, a máquina coloca-se no estado respectivo e pronta a executar a lógica necessária para a alteração de estado.

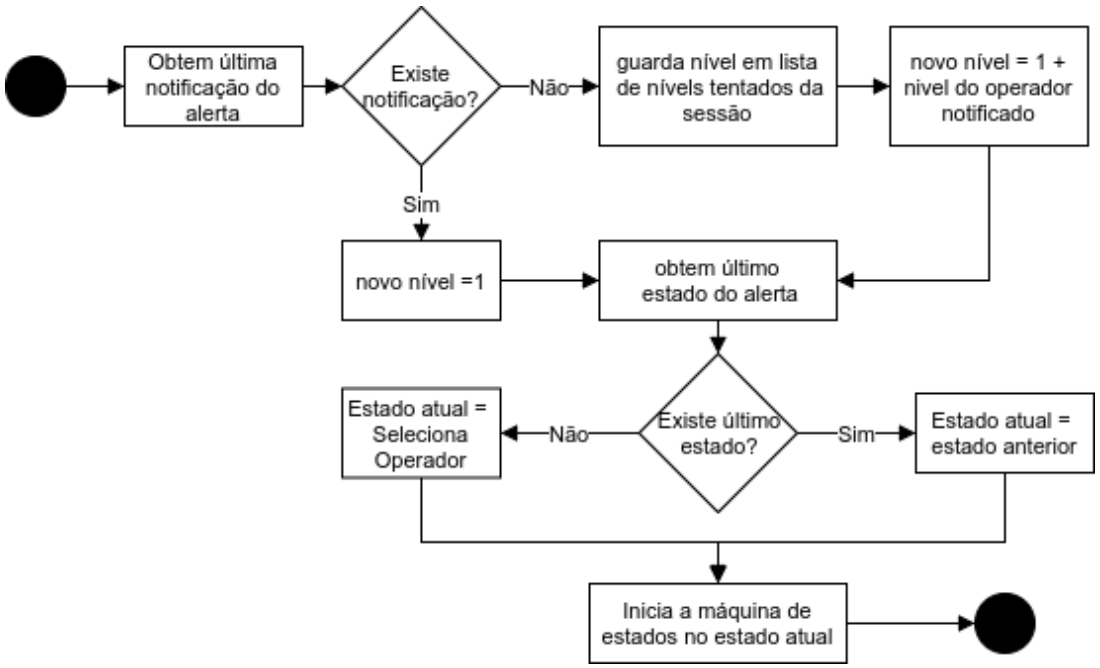


Figura 7.11: Fluxograma de inicialização da máquina de estados de alerta.

O estado *Selecciona Operador* é o estado inicial da máquina de estados. Neste estado, conforme representado na Figura 7.12, o sistema tenta contactar um dos operadores disponíveis, começando pelo que tiver mais alto nível de responsabilidade. O contacto com o operador, consiste em enviar-lhe uma notificação para o dispositivo móvel, durante o estado seguinte, no processamento de notificações. Do processamento de notificações espera-se uma resposta positiva ou negativa para, de acordo com a resposta, passa para o estado seguinte ou regressa ao estado *Selecciona Operador*, desta vez tentando o contacto com outro operador. No en-

tanto, caso o operador não tenha um dispositivo móvel, o processo de envio de notificações é ignorado pois não haveria um dispositivo móvel para onde enviar a notificação. Nesse caso, a máquina regressa ao mesmo estado, ficando registado como mudança de estado apesar de voltar ao mesmo, mas para registar o motivo de se ter conseguido notificar o operador. Se o estado *Selecciona Operador* estiver a ser visitado após tentativa falhada de notificação, o operador a tentar contactar será o operador de nível de responsabilidade seguinte ao que se tentou contactar antes. Se o nível incrementado for superior a 3, o sistema desiste de tentar contactar operadores durante a sessão e só o volta a fazer quando o sistema decidir reiniciar o processo, isto é, quando um determinado evento leve a re-processar os alertas que tenham ficado neste estado. Entretanto, pode ficar resolvido algum dos problemas que tenha impedido o alerta de ser tratado por um operador responsável, nomeadamente, por passar a haver operadores em turno, ou operador com dispositivo móvel, etc.

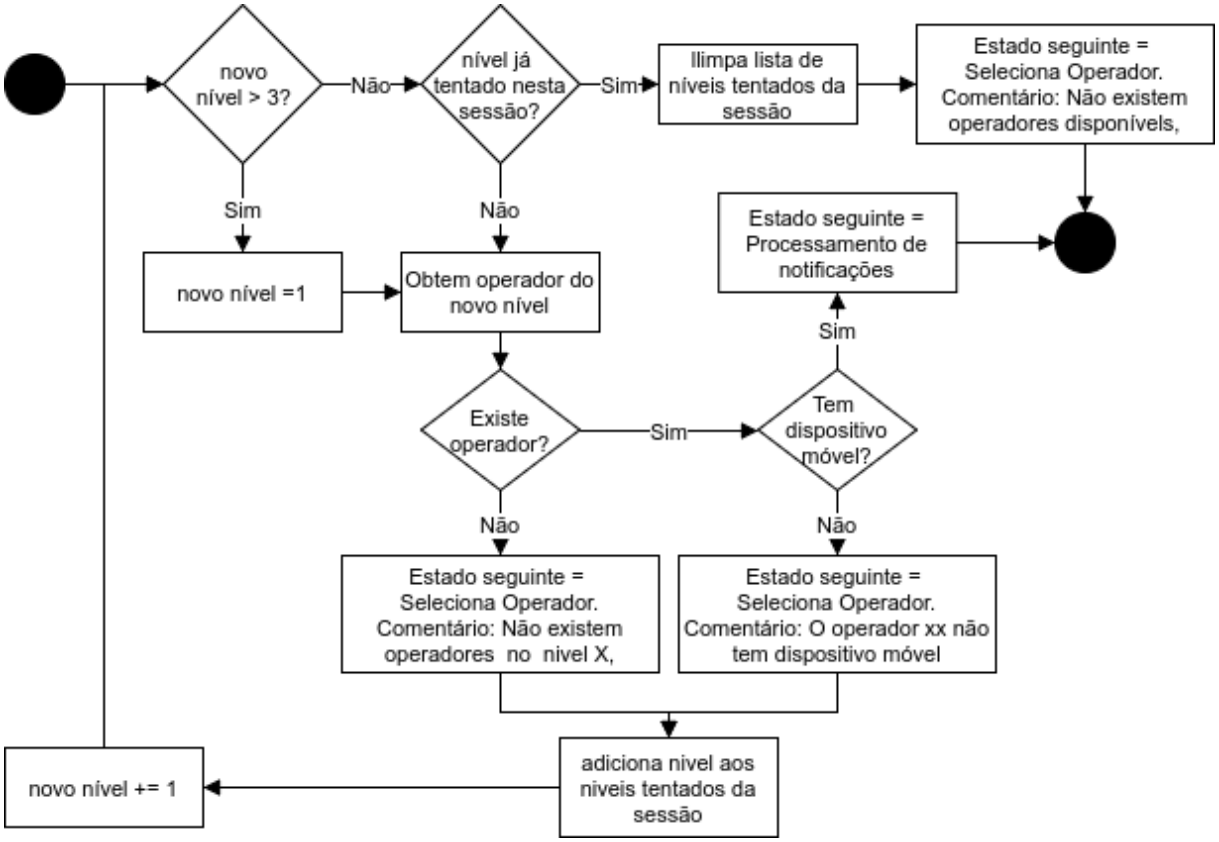


Figura 7.12: Fluxograma do estado Escolha de Operador.

O estado *Enviando Notificação* é um processo individual, com um ciclo de vida próprio. O processo de notificações, cuja descrição se encontra na Secção 7.1.10, é responsável por alterar o estado do processo de alerta. Em caso de sucesso, altera para o estado *Aguarda Visualização*. Em caso de insucesso da notificação, regressa ao estado *Selecciona Operador*.

No estado *Aguarda Visualização*, o sistema já enviou, com sucesso a notificação ao operador e aguarda que ele visualize o alerta. Como a notificação é apenas um aviso de que existe um alerta novo, para o operador ver o alerta, ele tem de clicar na notificação que aparece no dispositivo móvel. Como representado na Figura 7.13, ao clicar na notificação para a abrir, é invocada a aplicação Cat|Mobile, onde o operador terá de se autenticar caso a aplicação estivesse fechada. Automaticamente a aplicação pede o alerta da notificação ao servidor e o alerta é apresentado ao operador. Entretanto, durante o pedido do alerta ao servidor, foi dada como confirmada a visualização do alerta, fazendo com que o sistema passe para o estado seguinte.

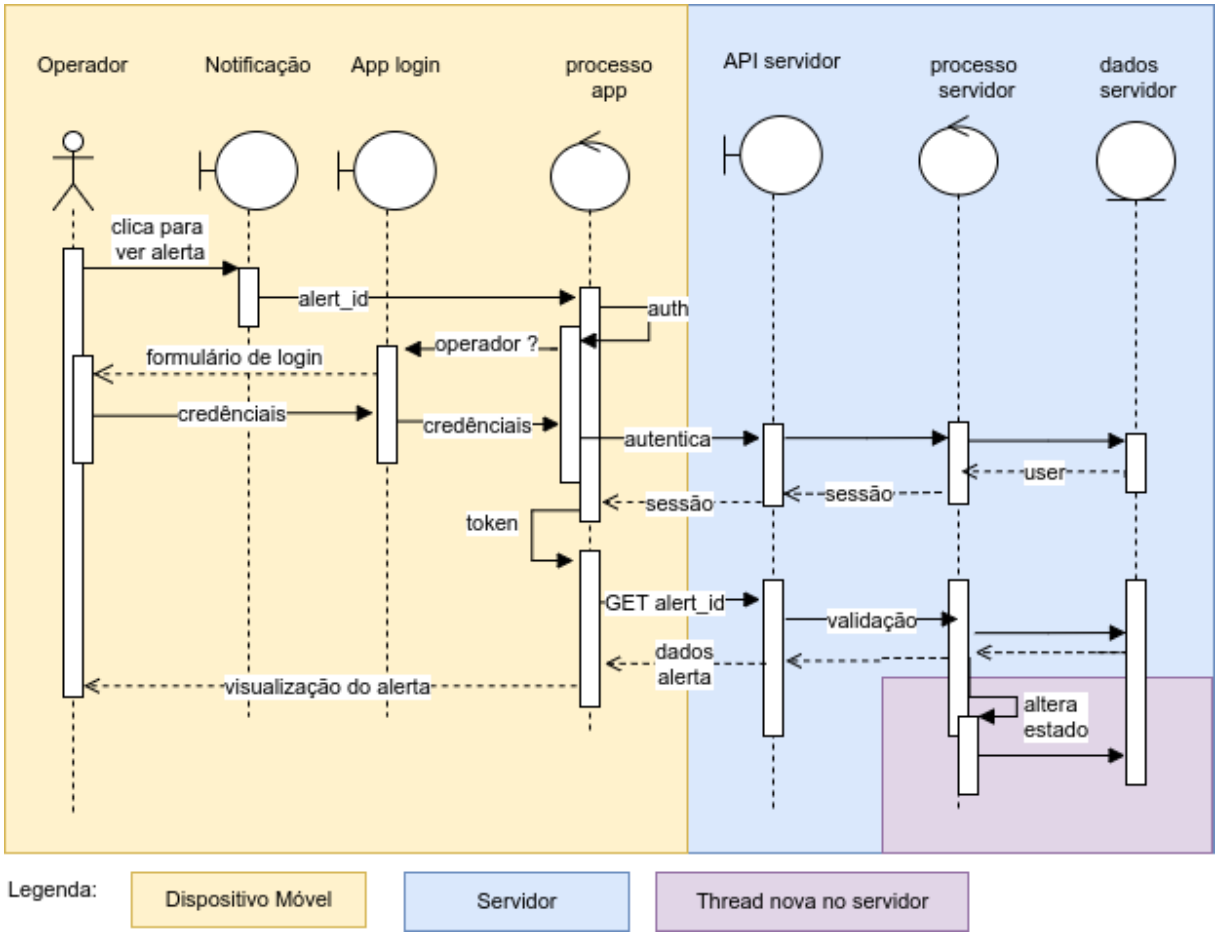


Figura 7.13: Diagrama de Sequência para a visualização de alerta.

No estado *Aguarda Aceitação* o alerta já está visualizado pelo operador que recebeu a notificação, resta aguardar que o operador diga que aceita iniciar o tratamento do alerta. O tempo de espera pela aceitação é pré-definido e indexado ao nível de severidade do alerta. Conforme representado na Figura 7.14, se o sistema não receber resposta do operador dentro do tempo esperado, o sistema assume que o alerta foi recusado e volta ao estado *Selecciona Operador*, para

outra tentativa. Ainda dentro do tempo de resposta, o operador tem hipótese de recusar o alerta, dessa forma poupando algum tempo desnecessário. De notar que será ignorada a visualização do alerta ou resposta de qualquer outro operador, que não o responsável pelo alerta no momento em que é enviada a notificação, pois a transição para o estado de *Tratamento* apenas é possível ao operador seleccionado, que tenha aceite o alerta dentro da janela de tempo pré-definida. Esta medida é imperativa para impedir interferências e garantir a integridade do processo.

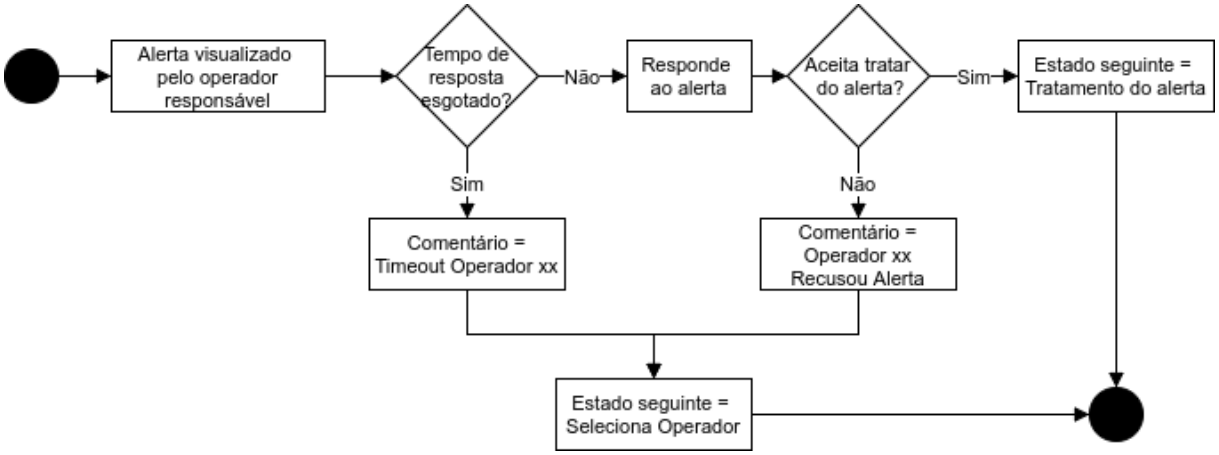


Figura 7.14: Fluxograma do estado Aguarda Aceitação.

O alerta entra no estado de *Tratamento* quando o operador notificado aceita o alerta. Este estado não tem restrições temporais para passar para o estado seguinte e serve somente para informar a equipa de que o alerta já se encontra a ser tratado. Durante este estado, o operador pode continuar a dar respostas ao alerta, com o intuito de ir documentando o tratamento do alerta e a sua evolução. Para passar para o estado seguinte, basta o operador, na aplicação móvel, activar a opção de conclusão que aparece no formulário de resposta.

O estado *Conclusão* é alcançado quando o operador responde ao alerta com a opção de conclusão activada. Este estado serve para identificar que o tratamento do alerta já foi dado por concluído.

7.1.10 Processamento de notificações

Como referido na Secção 7.1.9, o processamento de notificações é activado, quando o processamento de alertas entra no estado *Enviando Notificação*. Neste momento, está identificado o operador destinatário e o dispositivo móvel para onde será enviada a notificação. Neste processo, tenta-se certificar que a notificação tenha sido entregue no dispositivo mas não garante que o alerta tenha sido visto. A garantia visualização do alerta é posterior ao processamento

de notificações e faz parte do processamento de alertas. Para gerir o processamento de notificações foi implementada a máquina de estados representada na Figura 6.8. Para análise de resultados, rastreamento de ocorrências e também para o funcionamento do processamento de alertas, todos as notificações devem ter os seus estados persistidos juntamente com os seus dados relacionados. Contudo, a persistência não serve para retomar estados anteriores, como no processamento de alertas. Dado que os intervalos de tempo, em que ocorre a notificação, são bastante curtos, todo o processo pode residir em memória volátil. Por outro lado, se o processo for interrompido inadvertidamente, o máximo que pode acontecer é um operador receber notificações duplicadas, mas que apontam para um mesmo alerta, e dado que o estado do alerta é que garante a sua consistência, não existe o perigo de concorrência do alerta, em que mais do que um operador tentaria responder ao mesmo alerta. Neste módulo, foram implementadas as tarefas seguintes:

- Modelo de persistência para estados de notificações;
- Máquina de estados;
- Estado *Selecciona Dispositivo*;
- Estado *Aguarda Entrega*;
- Estado *Expirada*;
- Estado *Entregue no Dispositivo*.

O modelo de persistência do estado das notificações serve para registar o estado e o momento de cada transição de estado das notificações. Conforme representado na Figura 7.15, o modelo é composto pela identificação do estado, da identificação da notificação, da data de criação e da resposta obtida pela notificação.

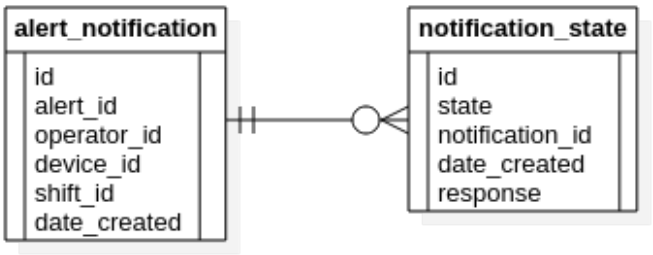


Figura 7.15: Diagrama Modelo ER dos Estados de Notificações.

A máquina de estados de notificações foi construída com a mesma base da máquina de estados de alertas. A sua inicialização é feita com a notificação como parâmetro, para dela obter todos os dados necessários para construir a mensagem e enviar essa mensagem a um dispositivo móvel. Apesar do dispositivo já estar definido quando o processo se inicia, *Selecciona*

Dispositivo é um estado que, por ser visitado em diferentes momentos do processo, pode vir a ser necessário, caso o dispositivo inicial deixe de ser o dispositivo actual do operador. Embora seja uma hipótese remota, ela existe, pelo que este estado serve para evitar que a notificação seja enviada a outro operador que não o operador designado como destinatário. Assim, a lógica deste estado é simplesmente verificar se o operador tem um dispositivo actual para enviar a notificação, entrando no estado *Aguarda Entrega* e, caso não tenha, aguardar a quinta parte do tempo total disponível para enviar a notificação para voltar a tentar. Este ciclo permanece em execução até expirar o tempo total para enviar a notificação, e entrar no estado *Expirada*.

O estado *Aguarda entrega* formaliza a entrega da notificação e aguarda pela confirmação da entrega. Neste estado, o dispositivo móvel está definido. Resta construir a mensagem e fazer o seu envio. Construída a mensagem, resta fazer os seu envio. Conforme o estudo do estado da arte referido na Secção 3.2 determinou-se que a solução mais adequada aos requisitos, para o envio de mensagens móveis, seria o serviço FCM. Para administrar o FCM, a Google disponibiliza um Software Development Kit (SDK) em Python [48], ideal para usar integrar na aplicação Django que se está a desenvolver. Conforme definido na Secção 6.3.1, os dados partilhados com terceiros, neste caso para o envio de notificações, não devem conter informação confidencial. Contudo a mensagem enviada deve conter dados que a aplicação possa usar para ter acesso à informação concreta do alerta e também um indicador de severidade do alerta como informação não crítica em termos de confidencialidade mas útil ao operador que recebe a notificação. Com estes pressupostos, o conteúdo da mensagem deve ser constituído apenas pelo ID do alerta e pelo valor numérico da severidade. Para além do conteúdo visível da mensagem, é também necessário preencher atributos específicos para Android [49], tais como o Time To Live (TTL) que define o tempo máximo que o FCM deve aguardar pelo dispositivo *online*, no caso de estar *offline* naquele instante, e também o parâmetro *clickAction* para definir que *activity*, do dispositivo Android, deve ser invocada quando o utilizador clica na notificação [3]. Para integrar o FCM na aplicação Django, foram seguidas as instruções indicadas em [50] e foi criada uma classe para instanciar o FCM, com a sua inicialização conforme [51] e também criados métodos necessários para os vários tipos de mensagens que se pretende enviar através desta plataforma. A inicialização do FCM consiste em obter o *token* de autenticação para acesso ao serviço através do envio de um ficheiro de configuração em formato JavaScript Object Notation (JSON) com as credenciais necessárias. Esse ficheiro de configuração é gerado automaticamente na plataforma on-line de gestão do FCM [52] e tem de estar presente no projecto Django para poder ser usado pelo SDK. A classe *FirebaseAdmin* foi criada para a administração do FCM. Esta classe é instanciada com uma notificação como parâmetro e contém os seguintes métodos implementados para enviar mensagens através do FCM:

Método *send_msg* - Este método envia a mensagem com base no alerta e no dispositivo referido na notificação. O resultado do método é a resposta da API do FCM. Essa resposta é

positiva caso o dispositivo tenha sido reconhecido pelo FCM, ou negativa em caso contrário. Como se pode ver na Figura 7.16, **os pedidos à API do SDK são síncronos, pelo que a resposta não certifica a entrega da mensagem no dispositivo, mas somente a entrega no FCM**, que irá tentar entregar no dispositivo móvel quando este estiver *on-line*. Este último passo de entrega é, no entanto, assíncrono e demora um tempo indeterminado. Apesar de um dos parâmetros da mensagem ser o TTL [3], o momento exacto em que a mensagem é entregue depende da disponibilidade do dispositivo, nomeadamente se tem rede ou por estar a reiniciar.

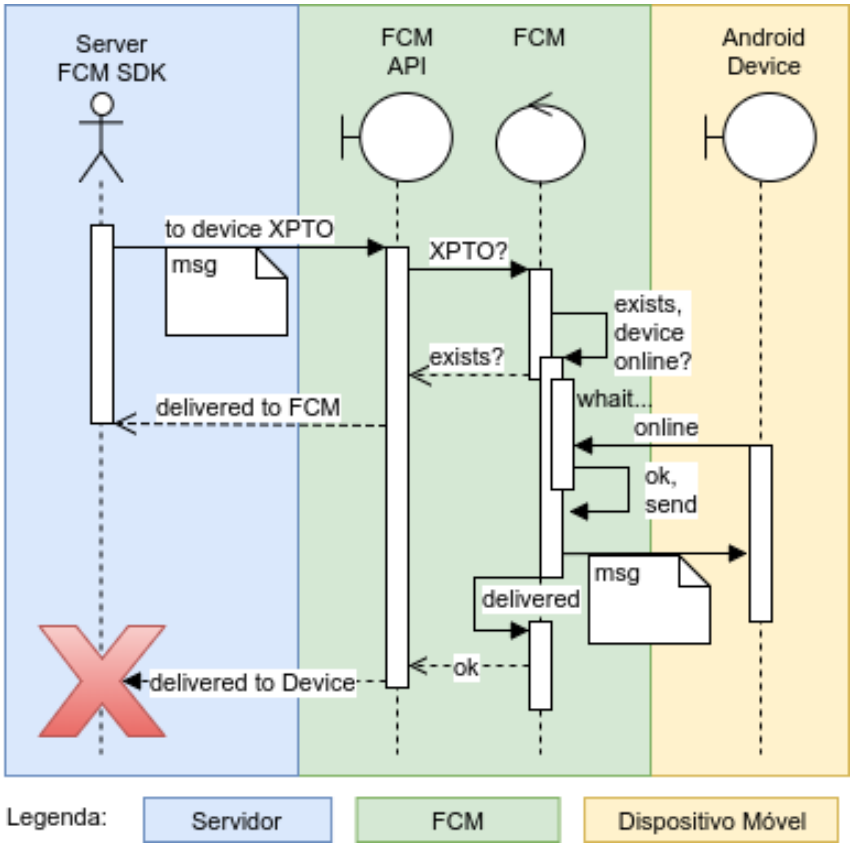


Figura 7.16: Diagrama de Sequência do envio de uma mensagem através da API do FCM.

- **For Android:** If you'd like to be notified when the application successfully receives a message, you can use `delivery_receipt_requested` functionality following the guidelines. This requires you to setup an XMPP server.

Figura 7.17: Anotação sobre os recibos de entrega FCM para Android.

De facto como se pode ver no texto da Figura 7.17, retirada de [53], a confirmação de entrega de mensagens no dispositivo móvel é uma funcionalidade exclusiva da implementação de FCM para XMPP . Para receber a confirmação de uma mensagem entregue

no dispositivo móvel, e não apenas no FCM, é necessário implementar uma ligação ao servidor XMPP do FCM e enviar as mensagens através desse serviço [54], em vez de ser através da API como se faz com o SDK. O envio de mensagens e recepção de recibos de entrega, com a implementação da ligação ao servidor XMPP está representado na Figura 7.18. A diferença entre o uso de XMPP e o uso da API é que os pedidos através de XMPP são assíncronos. Isto é, os recibos de entrega não chegam sequencialmente após o envio de cada mensagem. O recibo de uma mensagem enviada no instante T=0 pode chegar depois do recibo de uma mensagem enviada em T=1. Cabe à aplicação que gere o envio das mensagens, gerir também a correspondência entre mensagens e recibos. Para isso, torna-se necessário adicionar um campo extra ao modelo de persistência de notificações, para registar o ID atribuído pelo FCM à mensagem. Esse ID é atribuído quando o FCM recebe uma mensagem e valida o seu destinatário. Esse ID é devolvido rapidamente após o envio da mensagem, ou, no caso do destinatário não ser reconhecido, um erro em vez do ID. O formato da resposta do FCM é a mesma obtida através dos pedidos à API, seguindo a abordagem inicial com recurso ao SDK. No entanto, os IDs que de apenas garantem a entrega no FCM na abordagem anterior, agora servem para identificar os recibos de entrega no momento em que estes chegam à aplicação através de XMPP.

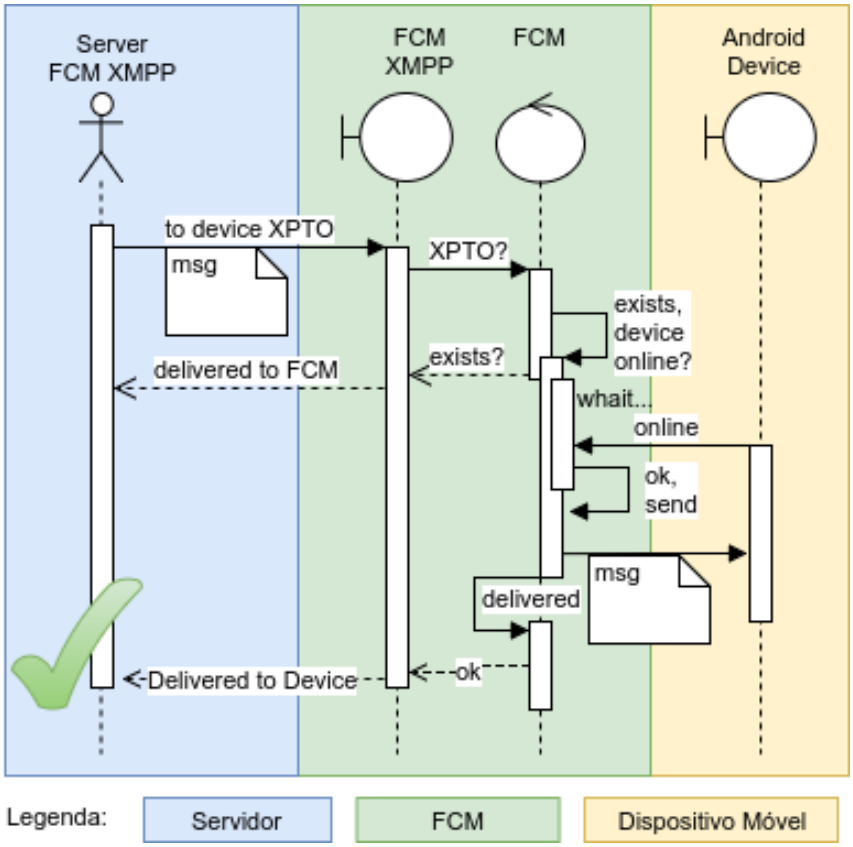


Figura 7.18: Diagrama de Sequência do envio de uma mensagem através de XMPP para FCM.

Para testar a funcionalidade de ligação a servidor XMPP e fazer a sua implementação segundo as instruções do FCM [54], foi necessário integrar uma biblioteca cliente de XMPP na aplicação Django. O autor escolheu a SleekXMPP [55] por estar bem documentada, ter uma grande comunidade de suporte e estar actualizada para a versão 3+ do Python em uso no projecto. O teste desta tecnologia consistiu em implementar pedidos de envio de mensagens para o servidor XMPP seguindo os exemplos disponíveis em [56] e fazendo o tratamento das respostas referidas em [57]. Os testes revelaram-se promissores, contudo ainda é necessário um investimento considerável em termos de tempo para implementar um conjunto demasiado grande de funcionalidades com equivalência ao SDK para a API já existente. Devido ao tempo limitado à realização do projecto, os recibos de entrega terão de ficar fora do projecto actual, mas fica a sugestão como trabalho futuro para evolução do Cat|Mobile.

Método *subscribe_topic* - Este método envia subscrição para tópicos de mensagens. Estes tópicos correspondem aos níveis de responsabilidade dos turnos dos operadores. Todos os operadores subscritos num tópico recebem a mesma notificação, pois esta é enviada em *Multicast* para o tópico. O nível de maior responsabilidade está excluído dos canais pois a notificação é enviada directamente ao dispositivo do operador e para mais nenhum outro operador. As notificações enviadas para os canais subscritos são meramente informativas e não para serem respondidas pelos operadores. Contudo, essas notificações também dão acesso ao conteúdo do alerta, mas não interferem no processo de alerta referido em 7.1.9 ou no processo de notificação referido em 7.1.10.

Método *unsubscribe_topic* - O método *unsubscribe* é necessário para desinscrever operadores, aliás os seus dispositivos móveis, de tópicos e deixarem de receber notificações quando estes saem da equipa *OnCall* ou mudam para um turno de primeira responsabilidade, ou trocam de dispositivo móvel. Sempre que o operador troca de dispositivo móvel durante um turno, todos os tópicos devem ser desinscritos do dispositivo anterior e inscritos no seguinte.

Método *update_shift* - Este método envia mensagens invisíveis para os operadores terem o seu turno actualizado na aplicação móvel a qualquer momento e não apenas quando fazem login. Se a indicação de turno for apenas enviada quando o operador faz login na aplicação, poderia acontecer que o turno fosse alterado durante a utilização da aplicação por parte do operador, sem que ele se apercebesse da alteração. Assim este método garante uma actualização assíncrona desse dado, onde o operador apenas vê na aplicação que o seu turno foi actualizado mas sem notificação como no caso dos alertas. Este tipo de mensagem sem notificação é possível porque o que transforma a mensagem em notificação é o serviço de processamento de mensagens da aplicação. Pelo que este tem um filtro para distinguir as mensagens são para aparecer em notificações das mensagens que são para

actualizar dados do sistema, como é o caso desta.

No estado *Expirada*, o processamento da notificação é terminado sem sucesso devido a *timeout* na entrega, devido ao operador não ter um dispositivo móvel activo reconhecido pelo FCM para enviar a notificação. Neste estado, o processo de notificação regressa ao processo de alerta no estado *Selecciona Operador* para recomeçar o processo com outro operador.

Se o processo atinge o estado *Entregue no Dispositivo*, significa que a notificação foi enviada para o FCM, mas não garante que tenha sido entregue no dispositivo, devido à limitação da API em não emitir recibos de entrega, a não ser por XMPP, como referido no parágrafo anterior. Contudo, ao entrar no processamento de alerta em estado *Aguarda Visualização*, é iniciado um contador de tempo disponível para o operador aceitar o alerta.

7.1.11 Período de Inactividade

O *Período de Inactividade* é uma funcionalidade que foi adicionada já perto do final do desenvolvimento do projecto e que, desta forma, demonstra alguma eficiência da arquitectura, permitindo o acrescento de funcionalidades ao intercalar funcionalidades existentes sem grandes alterações estruturais. Neste caso, foi solicitado que existisse um período diário sem notificações, correspondente ao período em que os operadores estão na empresa e não necessitam de receber alertas pelo dispositivo móvel. Este módulo necessita da seguinte tarefa implementada:

- Modelo de período de inactividade.

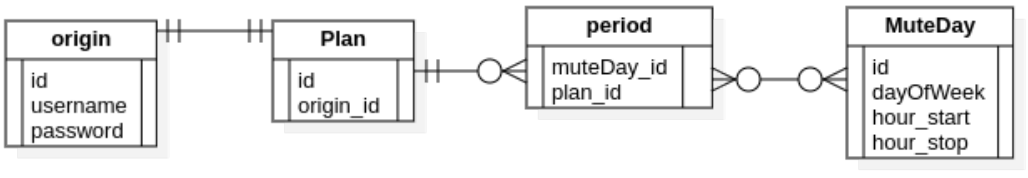


Figura 7.19: Diagrama de Modelo ER da persistência de períodos de inactividade.

Para persistir os períodos de inactividade, foi criado o modelo de persistência representado pela Figura 7.19 onde o administrador pode registar os períodos de inactividade, organizados em planos. De seguida foi adicionada uma cláusula na pré-inicialização do processamento de alertas que verifica se em dado momento o plano da origem do alerta permite o envio de notificações.

7.1.12 Gestão de Respostas a Alertas

As respostas aos alertas são todas as interacções, directas ou indirectas, que o operador tem com o alerta. Das respostas alerta possíveis existem as que são dadas explicitamente pelo operador, por exemplo, quando aceita o alerta, ou quando conclui o tratamento do alerta e existem as respostas que são dadas um pouco prevarsivamente, sem o operador se dar conta. Estas respostas pervasivas ou automáticas, ocorrem, por exemplo quando o operador visualiza o alerta, ou quando não responde ao alerta dentro da janela de tempo prevista. Uma resposta não dada a um alerta por parte de um operador responsável é, de facto, registada como uma resposta negativa no sistema. Só assim se poderia persistir o histórico de alterações de estado do alerta e dos seus respectivos motivos. Para a gestão de respostas, foi necessário implementar as seguintes tarefas:

- Modelo de resposta;
- API para respostas.

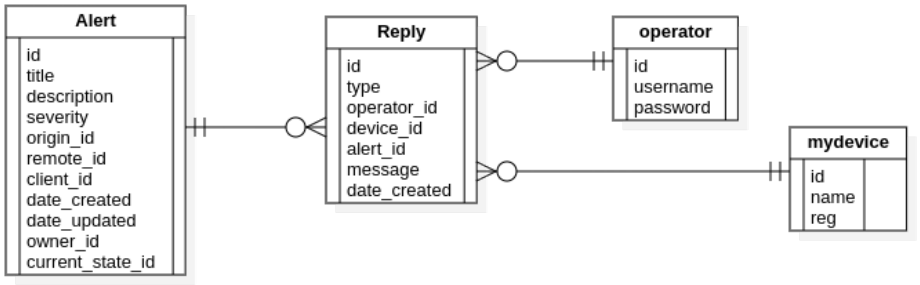


Figura 7.20: Diagrama de Modelo ER da persistência de respostas a alerta.

Com o conceito de resposta a alertas esclarecido, pode-se passar ao modelo de persistência representado na Figura 7.20. Este modelo é composto por numa nova classe *Reply* que se associa à classe *Alert* já existente. Na classe *Reply* é guardado o campo *Type*, para identificar o tipo de resposta, para além da ligação ao alerta a que pertence a resposta, o operador, autor da resposta e do dispositivo móvel onde foi criada a resposta. Os tipos de resposta possíveis são automaticamente preenchidos pela aplicação móvel e têm a sua confirmação verificada no servidor. Estes tipos de resposta interagem directamente com alguns dos estados do processamento de alerta e do processamento de notificações:

Delivered - Esta resposta é enviada durante o estado *Aguarda Entrega* do processamento de notificações onde termina favoravelmente esse processo e faz retomar o processamento de alertas no estado *Aguarda Visualização*. A resposta *Received* é enviada pela aplicação quando recebe uma notificação. É portanto, o recibo de entrega da notificação. No entanto, esta resposta nem sempre é enviada, porque é necessário VPN activa para dar acesso de rede ao servidor. Como a VPN é activada apenas quando o operador efectua

o login na aplicação, é somente possível enviar recibos quando o operador está a usar a aplicação. De notar que, para a funcionalidade de recibos poder ser usada com rigor, é necessário ainda algum trabalho futuro de desenvolvimento da solução, por exemplo, com a implementação de ligação ao servidor XMPP do FCM.

Received - Durante *Aguarda Visualização* do processamento de alerta, é aguardada a resposta *Received* que é criada automaticamente, pelo servidor quando recebe na API, o pedido da aplicação para obter o alerta. Neste momento, há garantia de que o alerta foi visto na aplicação por vontade do operador, pois foi necessário clicar na notificação e seguir o processo determinado para aceder ao alerta, que passa obrigatoriamente por estar autenticado na aplicação. Esta resposta faz com que o processamento de alerta passe para o estado *Aguarda Aceitação*.

Decline - Durante o estado *Aguarda Aceitação*, o alerta permite uma de duas respostas possíveis ao operador responsável (e somente a esse operador): Aceita ou Declina. A resposta *Decline* é uma resposta que pode ser dada manualmente pelo operador, enviada através da API, ou que é assumida automaticamente pelo servidor em caso de terminado o tempo disponível para resposta. Em ambos os casos, o processamento de alerta é reiniciado no estado *Selecciona Operador*.

Take - Ainda durante o estado *Aguarda Aceitação*, se o operador der resposta positiva de aceitação do alerta, é enviada a resposta *Take* ao servidor que, por sua vez, alerta o processamento de alerta para o estado *Tratamento*.

OnGoing - Durante o estado *Tratamento*, o operador pode continuar a dar respostas com o intuito de ir documentando o tratamento do alerta. Essas respostas são do tipo *OnGoing* e não alteram o estado do processamento.

Conclusion - Durante o estado *Tratamento*, as respostas *OnGoing* têm uma opção para concluir o tratamento que transformam a resposta em *Conclusion*. Essa resposta altera o processamento de alerta para o estado *Conclusão*.

Para a aplicação móvel poder dar respostas a alertas, é necessário uma API, com o respectivo *endpoint* configurado. A resposta a alertas é um dos *endpoints* da API do módulo *API para Aplicação Móvel* cuja implementação se encontra descrita na Secção 7.1.14.

7.1.13 Criação de Tickets

Criar um *ticket* é uma opção durante as respostas do tipo *OnGoing*. Ao dar uma resposta ao tratamento do alerta, o operador tem opção de criar um *ticket* no RTIR, com a mensagem escrita na resposta. Esta opção indica ao servidor que deve fazer um POST na API do RTIR. Esse *ticket*,

apesar de ser criado pelo servidor, deverá estar autenticado como sendo do operador que deu a resposta ao alerta. Como o sistema foi desenhado com a possibilidade de haver vários sistemas de *tickets*, onde cada um desses sistemas corresponde a uma origem de *tickets* por consulta, a criação do *ticket* também tem de referir qual desses sistemas deve ser usado para criar o *ticket*. Partindo do princípio que cada operador irá trabalhar apenas num desses eventuais sistemas de tickets, faz sentido persistir essa informação nos dados do operador para escolher o *endpoint* correcto na altura de fazer o POST de criação do *ticket*. As tarefas necessárias à implementação são as seguintes:

- Modelo para as credenciais de operador para acesso ao RTIR;
- Modelo para o *endpoint* do RTIR;
- Pedido para criação de ticket na API do RTIR.

A forma de guardar as credenciais para criação de *tickets* nos operadores é idêntica à forma de como se guardaram as credenciais de acesso ao RTIR para as contas de utilizador correspondentes à instância do RTIR a usar. Através da técnica de criar *Profiles* de utilizador [35], que já se encontrava a associar campos extra aos utilizadores do grupo Origens em 7.1.1, vai-se criar uma nova classe de persistência, desta vez para os operadores como representado na Figura 7.21.

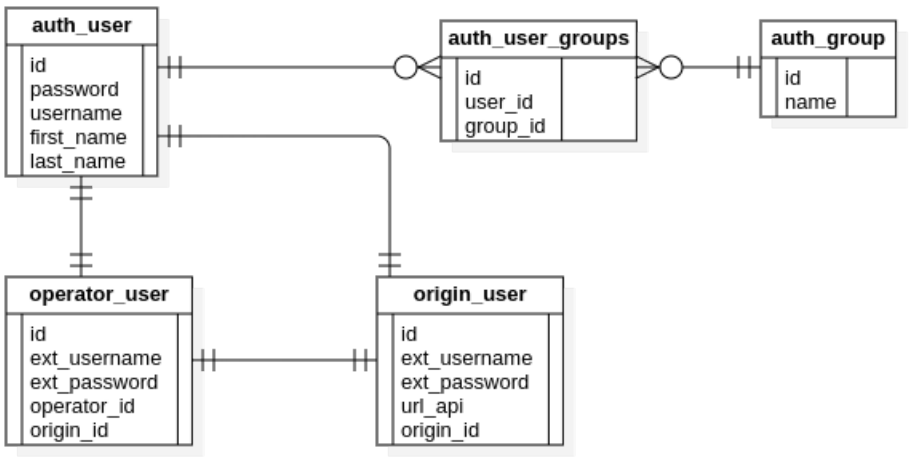


Figura 7.21: Diagrama de Modelo ER para dados de acesso à criação de *tickets*.

A mesma classe que associa as credenciais de acesso ao operador, vai também guardar a referência ao utilizador origem que detêm o URL da API onde pertencem as credenciais.

Para criar o ticket são necessários os dados do *ticket*, as credenciais para a API externa de *tickets*, o URL do *endpoint* dessa API e as credenciais de acesso. Para criar o conteúdo do ticket

é usado o conteúdo da resposta ao alerta, que terá sido escrita pelo operador com esse intuito. As credenciais a usar são as credenciais para tickets associadas ao operador como representado na Figura 7.21. O *endpoint* a usar é obtido através da Origem de alertas a que estão associadas as credenciais, também representado na Figura 7.21. É também necessário identificar o cliente a que se refere o alerta. A identificação dos clientes, RTIR, é constituída por uma designação própria que corresponde a uma determinada fila de *tickets*. Para poder usar essa designação, foi necessário criar um modelo de persistência do cliente conforme está na Figura 7.22. De notar que, até este ponto da implementação, o cliente ainda não dispunha um modelo próprio. A sua referência era apenas pelo simples nome. Para não ter de fazer alteraçõs profundas no código já desenvolvido, o autor optou por manter o nome do cliente como campo único que identifica o cliente.

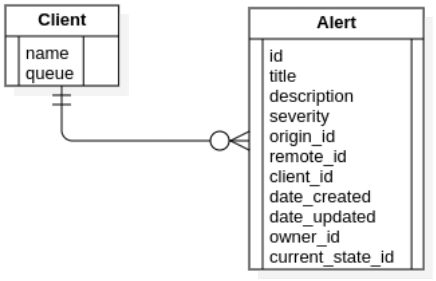


Figura 7.22: Diagrama de Modelo ER da tabela Cliente.

Reunidos todos os elementos, resta fazer o pedido de criação do *ticket* na API. Para tal, foi usada a biblioteca *RTIR4REST* [45] que disponibiliza o método *create_ticket* para criar *tickets* no RTIR. Uma vez criado o *ticket*, é necessário actualizar o campo *remote_id* do alerta com o ID que é devolvido na resposta ao pedido de criação to *ticket*, para impedir que esse *ticket* dê origem a um novo alerta.

7.1.14 API para Aplicação Móvel

Os dados que a aplicação móvel obtém e com os quais interage, são todos transmitidos através da API do servidor. Para aceder a essa API, é necessário autenticação da mesma forma que as origens necessitam para criar alertas, mas, neste caso apenas para o grupo dos operadores, pois só esses utilizadores estão autorizados a usar as funções orientadas para a aplicação móvel. Para obter o *token* de autenticação, o *endpoint* será o mesmo do já usado pelas origens. Contudo, para além de credenciais será necessário enviar identificação do dispositivo móvel e também porque a resposta será aproveitada para enviar mais informação para a aplicação, que será usada assim que é iniciada. Ao enviar estes dados extra junto com o *token* evitam-se pedidos extra que iriam

causar atrasos na abertura da aplicação. Para interagir com os alertas, o *endpoint* também é o mesmo, só os verbos mudam. Como a aplicação móvel não tem o propósito de criar alertas, o verbo POST não deve estar permitido para os operadores. No entanto o verbo GET deve estar disponível, tanto para obter listas filtradas ou não de alertas, como para obter um alerta individual. Necessita ainda de poder criar respostas a alertas e de receber todas as respostas de cada alerta individual. Como está previsto na aplicação móvel que o alerta, ao ser visualizado individualmente, contenha também todas as suas repostas visualizadas juntamente com os dados do alerta. Este último requisito de receber todas as respostas de cada alerta individual não fará sentido estar implementado num *endpoint*, mas sim usar-se o somente o *endpoint* do alerta individual e enviar junto com o alerta, a lista de todas as respostas. Desta forma, um único pedido satisfaz os dois requisitos, resultando em rapidez e eficácia na transmissão de dados. Na Tabela 7.2 encontram-se resumidos os *endpoints* da API. De seguida é descrita a utilidade de cada *endpoints* na aplicação móvel.

Tabela 7.2: Endpoints da API para Aplicação Móvel.

URL	**Método**	**Descrição e *URL Dispatcher***
/api/v1/token	POST	Obter o *token* + extras
path('api/v1/token/', GetToken.as_view()		
/api/v1/alerts/(id)/replies/	POST	Criar resposta
path('/api/v1/alerts/<int:pk>/replies/', ApiCreateAlertReplyView.as_view()		
/api/v1/alerts/	GET	Filtrar alertas.
path('/api/v1/alerts', ApiListCreateAlertsView.as_view()		
/api/v1/alerts/(id)	GET	Obter alertas.
path('/api/v1/alerts/<int:pk>', ApiViewAlertView.as_view()		

Obter o *token* + extras - Para se autenticar com credenciais, identificar o dispositivo móvel e obter dados de apresentação, tais como turno actual e outros dados que possam surgir em novos requisitos. Este URL aponta para a classe *GetToken* que já está a ser usada para fornecer o *token* às origens de alertas. No entanto o método implementado para processar o POST, diferencia a resposta, conforme o grupo a que pertence o utilizador autenticado. Caso o utilizador pertença ao grupo Origens, devolve apenas o *token*, como já referido na Secção 7.1.3. Caso pertença ao grupo Operadores, devolve o *token* do operador, o turno actual do operador,a identificação do dispositivo móvel perante o servidor e ainda a data e hora do último *login* do operador. Todos estes dados são devolvidos em formato JSON, gerados através de classes *Serializer* do DRF [41].

Criar resposta - Para responder a um alerta. Este URL está mapeado com a classe *ApiCreateAlertReplyView* que descende de *CreateAPIView* onde apenas está implementado o método para o verbo POST. Esta classe, começa por identificar o alerta através do seu ID enviado como parâmetro no URL, para depois, com os dados enviados no *Body* do POST, criar uma instância de *Reply* para associar como resposta ao alerta. Durante a criação da resposta, é verificada a existência da opção de *Conclusão* que determina a entrada no estado Conclusão do processo do alerta referida na Secção 7.1.9, ou a opção *Ticket* que despoleta a criação de um *ticket* na ferramenta externa de *tickets*, cujo processo é referido na Secção 7.1.13.

Obter alerta - Para obter um alerta indicado por uma notificação, ou seleccionado da lista de alertas. Junto com o alerta devem ser enviadas todas as respostas se existirem. Para obter um determinado alerta através da API é necessário indicar o ID do alerta no pedido. A classe *ApiViewAlertView*, que está ligada ao URL deste *endpoint*, descende de *RetrieveAPIView*, uma classe que apenas disponibiliza o método para GET. Esta classe está pré-programada para obter a instância Alert indicada pelo ID enviado no URL. Uma vez, obtida a instância pretendida, preparam-se os dados a enviar em formato JSON, através da classe *AlertSerializer*. A classe *AlertSerializer* envia todos os campos do alerta, sem restrições e ainda acrescenta à resposta a lista de respostas que o alerta possa ter eventualmente.

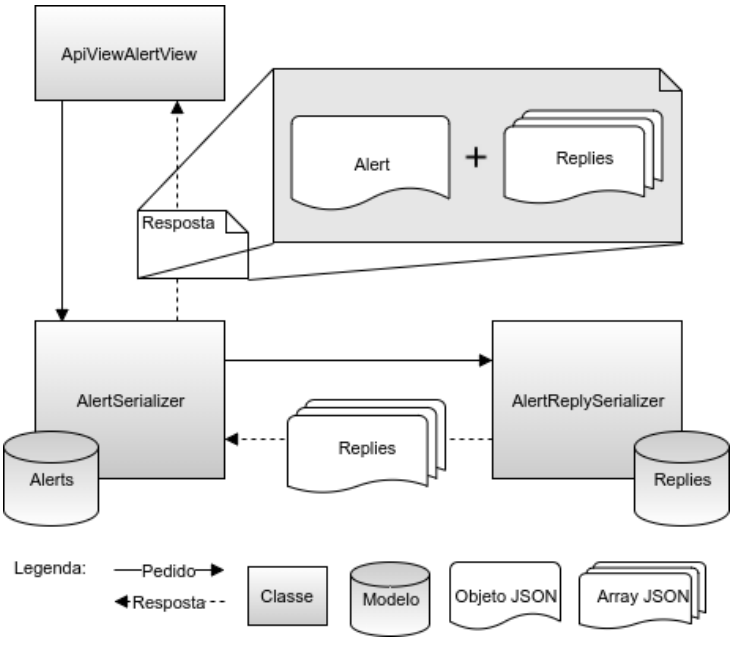


Figura 7.23: Diagrama de serialização de alertas para API móvel.

Como se pode ver na Figura 7.23, para enviar também as respostas, é invocada a classe *AlertReplySerializer* que trata de devolver a lista serializada em JSON de todas as respos-

tas desse alerta que será incluída nos dados do alerta a serem devolvidos em resposta ao pedido do alerta.

Filtrar alertas - Obter listas de alertas resumidos filtrados por campos tais como, cliente, origem, severidade, etc. Este URL corresponde à classe *ListCreateAPIView*. Esta classe já fora referida em 7.1.3 por ser partilhada com o URL de criação de alertas devido às boas práticas *REST* que se pretendem usar. Esta classe tem capacidade para processar POST e GET em alertas. Contudo, durante o método que trata o POST, são permitidos apenas utilizadores do grupo Origens enquanto que o método para GET está limitado a utilizadores do grupo Operadores. No caso da filtragem de alertas, é usado o método GET para obter uma lista de alertas. Essa lista pode ser filtrada através do envio de parâmetros no URL, correspondentes a campos dos alertas. Se o pedido for efectuado ao URL sem parâmetros de filtragem, é devolvida a lista completa de alertas. Os parâmetros possíveis de usar na filtragem são determinados pelo atributo *search_fields* [58]. Para devolver uma lista filtrada basta aplicar o método *filter_queryset*, que verifica automaticamente a existência de um ou mais campos em *search_fields* e usa-os para filtrar o *queryset* inicial [59]. Uma vez obtidas as instâncias pretendidas, resta serializa-las em JSON para as devolver. No entanto, pretende-se que a lista seja de alertas em formato reduzido, isto é sem a descrição do alerta, apenas com o título, origem, data, severidade,estado e nome do operador responsável. Para isto, recorreu-se à classe *AlertListSerializer* e limitaram-se os campos a disponibilizar indicando-os no atributo fields [60].

7.1.15 Interface de Administração

O *Interface de Administração* é um interface gráfico Web acessível através de *browser* para Internet. O Django, como framework de desenvolvimento em Python orientado para a Web, sobretudo, dispõe de todos as tecnologias necessárias para facilitar o trabalho do programador em desenvolver projectos para Web. No caso do Cat|Mobile, será necessário implementar vários formulários, para o Login, para registo de origens e operadores e também para filtrar listagens. Será também necessário implementar listagens, com colunas e paginação. Para apresentação do interface Web, o Django dispõe de um motor de templates que gera Hypertext Markup Language (HTML) a partir de modelos de templates pré-definidos. Estes templates recebem os dados a apresentar, vindos de *Views*, que por sua vez, geram dados a partir de classes *Model*. Ao receber os dados em bruto, os templates distribuem esses dados por áreas pré-determinadas, criando páginas dinâmicas a partir de modelos estáticos. Esses templates são construídos com uma mistura de HTML e linguagem *Markup* própria do motor de templates e que permite efectuar processos de baixa complexidade, tal como ciclos e substituição de variáveis, entre outros[61]. Para os formulários, o Django dispõe das classes *Form* [62] e *ModelForm* [63] que facilitam a criação de formulários com base em classes de modelo existentes. Para não gastar demasiado

tempo com o aspecto gráfico das páginas HTML, o autor decidiu usar um projecto *opensource* de *dashboard* genérico para projectos de administração Web. Como base de construção para os templates Django, foi usado o projecto *ModularAdmin* [64]. O *ModularAdmin* tem licença Massachusetts Institute of Technology (MIT), que dá permissão de uso livre. Este projecto implementa um *dashboard* de administração genérico, com diversas páginas HTML construídas para os mais diversos fins. Inclui bibliotecas JavaScript (JS) para a construção de gráficos estatísticos, listagens, menus, formulário de login, estilos tipográficos, etc. Tudo com respectivos Cascading Style Sheets (CSS) pré-definidos e preparados para *mobile*. A adaptação para templates de Django requer uma análise prévia onde se identificam as partes em comum entre as diversas páginas, para com elas, criar bases de template a ser reusadas pelas páginas finais do template. É ainda necessário preparar os *placeholders* para os dados das *Views* e também para os formulários Django. Para implementar o Interface de Administração, foi necessário implementar as seguintes tarefas:

- Página de Login
- Auto-Logout
- Dashboard principal
- Menu
- Agenda
- Gestão de Alertas
- Gestão de Operadores
- Gestão de Origens
- Gestão de Dispositivos

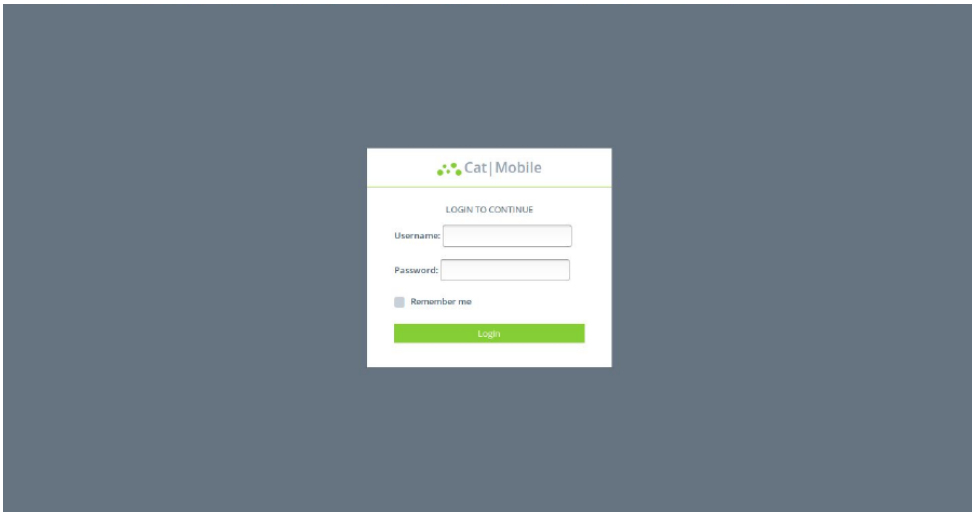


Figura 7.24: Página de Login do Interface de Administração.

A página de Login, conforme se pode ver na Figura 7.24, é composta por um formulário

com campos de username e password. Esta página é a única de acesso livre. Todas as outras páginas são redireccionadas para esta em caso de falta de sessão activa. A sessão só é iniciada para os utilizadores administradores. Utilizadores do grupo origens ou operadores, não têm acesso ao interior do interface de administração. Para implementação do formulário de login é usada a classe *LoginView* do Django que implementa a permissão de acesso a administradores com a classe *ManagerLoginForm* derivada de *AuthenticationForm* [65].

Como forma de segurança, implementou-se um o logout automático, que termina a sessão ao fim de 5 minutos sem actividade, seja de teclado ou de rato. Para isso, sem querer "reinventar a roda", instalou-se e configurou-se o módulo *django-session-security* [66].

A página *Dashboard*, visível na Figura 7.25 é a página inicial do interface de administração. Aqui o administrador pode visualizar o estado do sistema com informações resumidas dos dados armazenados. Assim, esta página é composta por um quadro que apresenta os turnos dos operadores OnCall organizados por níveis de responsabilidade.. Ao clicar no operador OnCall, é apresentado o calendário/agenda onde o período do turno é visível na forma de tabela. Caso não haja operador OnCall em algum nível, é apresentado um botão que liga ao formulário de registo de turno pré-preenchido. Ao lado do quadro de turno, a Dashboard apresenta um gráfico estatístico que representa o volume de alertas diários dos últimos 30 dias. Em baixo do quadro de turnos e do gráfico de alertas, surge uma lista dos clientes que receberam alertas recentes em que cada linha da lista é composta pelo alerta recebido desse cliente, que liga à página do alerta. Ao lado desta lista, um gráfico circular que representa o volume de alertas por cliente até ao momento.

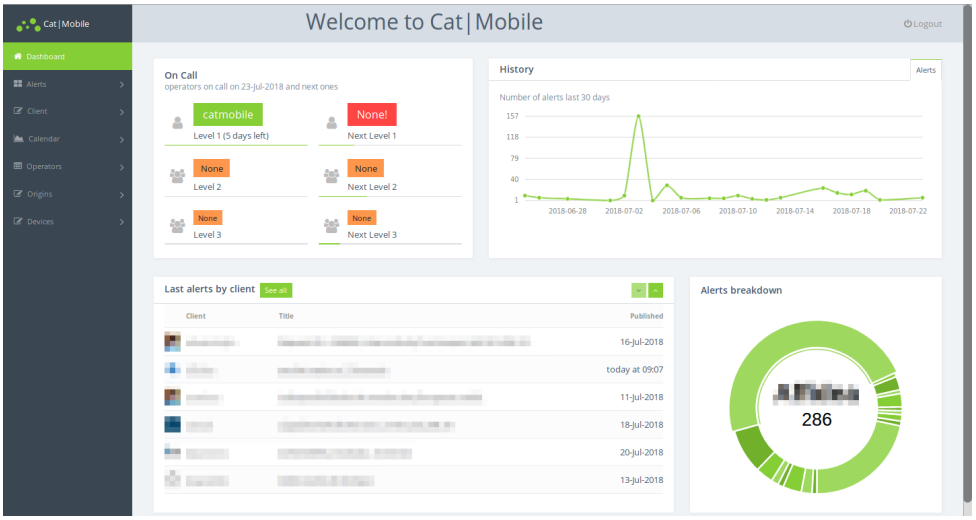


Figura 7.25: Dashboard principal do Interface de Administração.

O menu, é um elemento HTML que é visível em todas as páginas no interior do interface de administração. Este menu tem ligações para as seguintes páginas: Dashboard, lista de alertas, registo de origens, lista de origens, registo de clientes, lista de clientes, registo de operadores, lista de operadores, registo de turnos, calendário/agenda e lista de dispositivos móveis. No topo do menu, tem a opção para sair.

A funcionalidade de gestão de turnos pretendida tem muito em comum com funcionalidades de gestão de eventos em calendário, ou seja uma agenda. Como tal foi usado um módulo de calendário e agenda para Django com possibilidade de adaptação para a gestão de turnos pretendida. O módulo *Scheduler* [67], implementa mais funcionalidades do que as pretendidas para o requisito inicial a que se destina. No entanto, as funcionalidades extra, podem vir a ser aproveitadas em requisitos futuros, dos quais ainda não tenha surgido a necessidade, mas que provavelmente venham a surgir, pela similaridade do problema que este módulo pretende resolver com as características do requisito inicial. O modelo de dados, definido pelo módulo *Scheduler*, é constituído por eventos que pertencem a calendários. Os eventos, para além das datas de início e de fim, possuem ainda, atributos tais como título, descrição, data de criação, autor e informação sobre a eventual repetição do evento. No caso da utilização que se pretende dar ao módulo, é necessário que os eventos registem o atributo de utilizador a que se destina o turno e o atributo de nível de prioridade respectivo, para além dos atributos já existentes na classe *Event*. Para associar as instâncias *Event* com as instâncias *Shift* dos turnos, foi usada a funcionalidade de relacionamento de eventos com objectos genéricos disponível no módulo *Scheduler* [67]. Para criar os eventos do tipo turno (em que o evento tem um utilizador e um nível associados) é necessário um formulário onde o administrador especifica as datas de início e fim, o utilizador do turno e o nível de prioridade. O código desse formulário fica responsável por criar a instância de *Event* com as datas indicadas, e criar o *EventRelation* com o turno *Shift* respectivo. Os *Forms* Django são objectos da classe *Form* constituídos por atributos [62]. No caso de *forms* com origem em classes do tipo *Model*, os atributos de base do *Form* são os atributos da classe *Model* a que está associado. Estes atributos podem ser incluídos ou excluídos do *Form* à vontade do programador. Podem também ser criado atributos exclusivos do *Form* que durante a validação podem dar origem a atributos do *Model*. No caso do requisito a implementar, a classe *Event* é o *Model* do *form*. No entanto, também devem aparecer os campos para utilizador e nível de prioridade (que não fazem parte da classe do *Model*). Esses campos são processados pelo método *form_valid* [68], após a submissão do formulário, onde será criado o respectivo *EventRelation* que liga o turno *Shift* ao evento. Para a data de início de data de fim, seria útil um *"date picker"*. Para isso foi necessário incluir um módulo JS que implementa o "*date picker*". A inclusão de módulos JS nos templates Django pode ser feita de duas maneiras. Directamente nos ficheiros de template ou através de *Widgets* [69]. Para usar módulos JS em campos de formulários Django, é necessário fazer a inclusão dos ficheiros JS na declaração dos templates HTML. Isso pode ser feito através de HTML puro ou com o comando *static* dos tem-

plates Django. Depois, é necessário alterar manualmente os atributos dos campos do formulário de forma a identificá-los e configurá-los para serem usados pelo pretendido módulo JS. Isto implica que o formulário Django seja decomposto e os seus diversos campos sejam declarados um a um no template HTML para serem renderizados. Desta forma abdica-se da funcionalidade de renderização completa e sistematizada de formulários disponibilizada pelo Django, ficando o programador com mais controlo, mas também inteiramente responsável por essa renderização. Esta abordagem é mais susceptível a eventuais falhas durante a programação devido ao aumento da complexidade dos templates assim como aumenta o custo do desenvolvimento por implicar a repetição de código a cada reutilização de módulos JS. Em alternativa a essa abordagem mais laboriosa, existem os *Widgets* que são componentes disponibilizados pelo Django para apresentar campos de formulários [69]. Em cada *Widget* está definida a localização e implementação do módulo JS. Para utilizar *Widgets*, o programador apenas tem de indicar a utilização do *widget* por determinado(s) campo(s), na classe que define o formulário. A renderização do JS nos templates HTML fica a cargo do *Widget* [70]. Apesar da curva de aprendizagem mais acentuada com esta funcionalidade do Django, os *Widgets* possibilitam um controlo centralizado de módulos JS e também permitem maior facilidade no reaproveitamento ao longo dos diversos formulários da aplicação sem ter fazer alterações manuais nos ficheiros de template. De notar, que a utilização de *widgets* de terceiros pode levar à ocorrência de problemas. Por vezes os *widgets* utilizam bibliotecas de JS que são incluídas nos seus módulos. Devido a inclusão manual de bibliotecas nos templates ou mesmo pela instalação de outros *widgets*, pode acontecer que uma mesma biblioteca seja incluída no Document Object Model (DOM) mais do que uma vez, fazendo com que a função onde é declarada a biblioteca apareça múltiplas vezes. Ora, como o DOM é executado *Top-Down*, a biblioteca tem de ser instanciada antes de poder ser usada, mas ao ser re-instanciada após a eventual utilização desta, num *script*, o que acontece é que ela deixa de estar reconhecida, para ser usada pelos *scripts* anteriores. Para resolver este problema, poderia-se alterar o código do *widget* para não incluir determinada biblioteca. No entanto como não é recomendável a alteração do código de módulos de terceiros (pois pode causar problemas em actualizações futuras), a solução ideal passa por criar um novo *widget* que tenha, como parente, o *widget* original e nele sobrepor as definições de "media" (onde são declarados os ficheiros JS a ser usados [70]) e assim fazer na aplicação, a gestão dos scripts e bibliotecas a ser incluídos no DOM. Para além do formulário para criação directa de turnos, foram também modificados os templates HTML de calendário incluídos no módulo *Scheduler* de modo a obter o formulário com datas pré-preenchidas ao clicar sobre um dia do calendário.

A gestão de alertas, visível na Figura 7.26, é constituída por funcionalidades de listagem, filtragem e operações de publicação ou despublicação. As operações de publicação e despublicação equivalem a eliminar e recuperar virtualmente os alertas, mantendo-os assim sempre em base de dados, apenas omitindo a sua visibilidade.

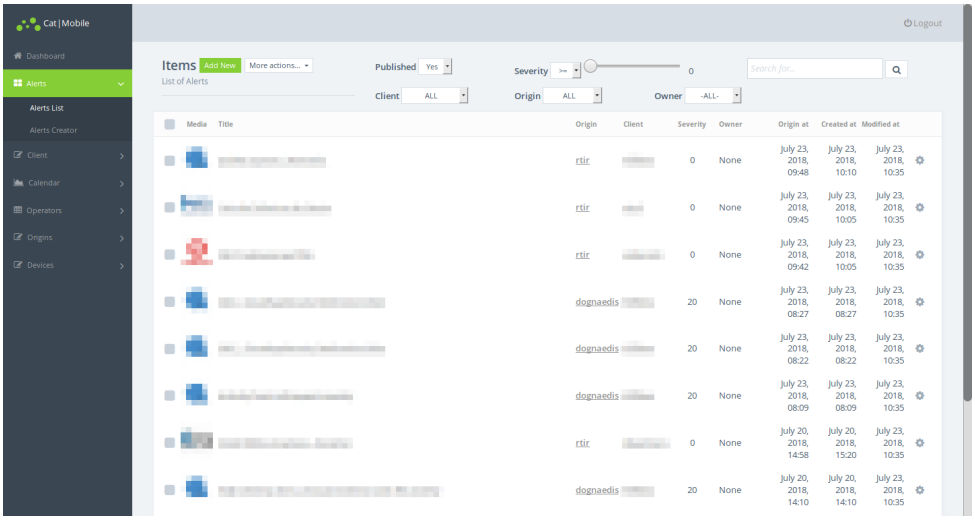


Figura 7.26: Gestão de Alertas no Interface de Administração.

Os Alertas são listados através das *class-views* Django *ListViews* que permitem iterar os objectos de determinada classe com bastante facilidade para o programador. A lista com os objectos é transferida da view para o template definido na *ListView*, onde deve ser iterado com o HTML formatado a gosto para cada campo dos objectos. Para que a listagem possa ser repartida por várias páginas navegáveis, é necessário implementar paginação. Mais uma vez, o Django dispõe dessa funcionalidade já implementada nas *ListViews*, facilitando muito o trabalho do programador. Para isso, basta definir o campo *paginate_by*=10 para definir páginas de 10 objectos cada, por exemplo. Depois, no template, fica disponível um novo objecto *page* que terá de ser incluído no HTML para apresentar a navegação entre páginas através de *anchors* que fazem pedidos GET à *ListView* com indicação da amplitude de páginas a apresentar. Para que possam ser realizadas acções sobre a lista de alertas, é necessária a inclusão de um formulário HTML por onde possam ser feitos pedidos POST à *ListView*. Para que as ListView possam implementar *Forms* (normalmente associadas a *FormView*) é necessário que tenham herança de *FormMixin* (*Mixins* são uma forma de usar interfaces de Object-oriented programming (OOP) em Python). Desta forma, para além das funcionalidades de *ListView*, podemos ter também funcionalidades de *Forms*, tudo na mesma view. Para as funcionalidades de *Form*, é necessário ainda indicar a classe de *Form*, que também será necessário criar. Para escolher e identificar os alertas da lista pode-se usar a funcionalidade de tratamento de POST na *View* que responsável pelo *Form*. A função que recebe o request POST dispõe do método *getlist*, que aceita uma *string* como argumento. O método *getlist* server para filtrar os dados recebidos com atributo *"name"* dos *Inputs* de *Form* do HTML. Para usar este método, é necessário que o template HTML tenha identificado cada Alerta com o seu ID e acrescentada uma *tag* INPUT do tipo *CheckBox* que permite escolher os elementos de uma lista, enviando-os, devidamente identificados ao método *getlist* do tratamento do *Form*. Dentro da classe dos *Forms* existe o método *clean* que permite

acesso aos dados do *form* antes de este ser enviado à *view* para tratamento. Este método deve ser usado apenas para um primeiro passo na validação dos dados e nunca para fazer alterações no modelo pois estes ainda não se encontram totalmente validados.

Quanto à filtragem de alertas, esta é possível através dos campos severidade, publicado, operador responsável, cliente, origem e campo de texto livre. Para implementar a filtragem usou-se GET por ser o verbo HTTP mais adequado ao uso pretendido pois são feitas apenas consultas de dados. O uso do GET permite também portabilidade de listas de Alertas, pois cada lista é produzida em conformidade com os parâmetros do URL. Cada alerta da listagem inclui ligação para a página do alerta. A página do alerta é composta pelos dados do alerta, pelo estado actual e pelos histórico de alterações de estados e de notificações. Todos os elementos que constam nessas listas, tais como operadores, dispositivos móveis, clientes e origens, possuem ligação para as respectivas páginas de dados detalhados.

A gestão de operadores é constituída pelo formulário de registo do operador, da lista de operadores e da página relativa a cada operador. O formulário de registo inclui os campos de *username* e *password* a ser usados no *login* da aplicação móvel. Inclui também os campos de username e password externos para criar tickets no RTIR. A lista de operadores inclui a ligação para a página de cada operador. A página de operador inclui um cabeçalho onde são visíveis os dados do operador, nomeadamente, o seu turno e dispositivo móvel actual. O turno inclui a ligação para o turno na agenda global. O dispositivo inclui ligação para a página dedicada ao dispositivo onde é apresentado o seu histórico de utilização. Para além do cabeçalho, como histórico de actividade do operador, uma lista das mais recentes alterações de estado no processamento de alertas em que o operador interviu e em que cada alerta tem ligação para a página do alerta.

A gestão de origens, de modo idêntico à gestão de operadores, é composta pelo formulário de registo da origem, da lista de origens e da página relativa a cada origem. O formulário de registo inclui os campos de *username* e *password* a ser usados nas ferramentas externas de origem de alertas. Inclui também os dados de acesso ao RTIR se a origem for desse tipo. A lista de origens, inclui uma ligação para a página de cada origem. A página da origem inclui a lista de alertas recentes emitidos por ela em que cada alerta tem ligação para a página do alerta.

A gestão de dispositivos é composta por uma lista de dispositivos apenas. Não existe formulário de registo de dispositivos porque estes são automaticamente registados pela aplicação móvel quando esta é usada num novo dispositivo. Cada dispositivo da lista possui ligação para a página do dispositivo. A página do dispositivo é composta pelos dados identificadores do dispositivo e pelo operador que eventualmente estará a usar o dispositivo actualmente. Possui também um histórico de alterações de operador que tenham usado o dispositivo.

7.2 Implementação da Aplicação Móvel

Passando à componente móvel do sistema, com o desenho e especificação da arquitectura da aplicação móvel terminado, dá-se inicio à implementação dos módulos que o compõem. O mapa de correspondência entre os requisitos e os módulos que os implementam pode ser consultado no Apêndice A.

7.2.1 Serviço de Mensagens

O serviço que recebe as mensagens enviadas pelo FCM é um serviço usado pelo Google Play Services que está a correr em segundo plano no sistema Android. As tarefas implementadas para o serviço de mensagens são:

- Firebase
- FirebaseMessagingService
- AndroidManifest

O Firebase foi adicionado ao projecto, seguindo as instruções da documentação online [71]. Para implementar o serviço responsável por receber as mensagens FCM, foi criada a classe *MyFirebaseMessagingService* conforme indicado na documentação [72]. Esta classe recebe as mensagens enviadas pelo FCM, através do método *onMessageReceived* e processa-as através da classe *AlertNotificationHandler*.

Para o serviço ficar reconhecido no sistema e poder ser iniciado automaticamente pelo Android, foi necessário configura-lo no ficheiro *AndroidManifest* conforme as instruções da documentação [71]. De notar que caso o sistema Android instalado no dispositivo contenha algum limitador de inicialização automática de aplicações, este deverá estar configurado para permitir a aplicação Cat|Mobile.

7.2.2 Handler de Mensagens

O Handler de Mensagens, é uma classe invocada pelo serviço para encaminhar as mensagens conforme o seu tipo. As tarefas implementadas neste módulo são as seguintes:

- AlertNotificationHandler
- NotificationUtils

O Handler das mensagens faz a distinção entre mensagens conforme representado na Figura 7.27. Se a mensagem for de um alerta, será para criar uma notificação no sistema Android. Caso não seja de um alerta, é uma actualização de dados, por parte do servidor, que será enviada para a *MainActivity*. Para a criação de notificações, o alerta contido na mensagem é distinguido

por nível. Nível 1 significa que é uma notificação para o operador responsável, logo será uma notificação ruidosa que só é interrompida quando o operador responder ou quando expirar um determinado tempo. Caso o alerta não seja de nível 1, a notificação é silenciosa e permanece na lista de notificações até ser visualizada. Antes de ser apresentada a notificação, no caso de ser de nível 1, o ecrã deve estar desbloqueado para que a notificação possa estar visível para o operador responder.

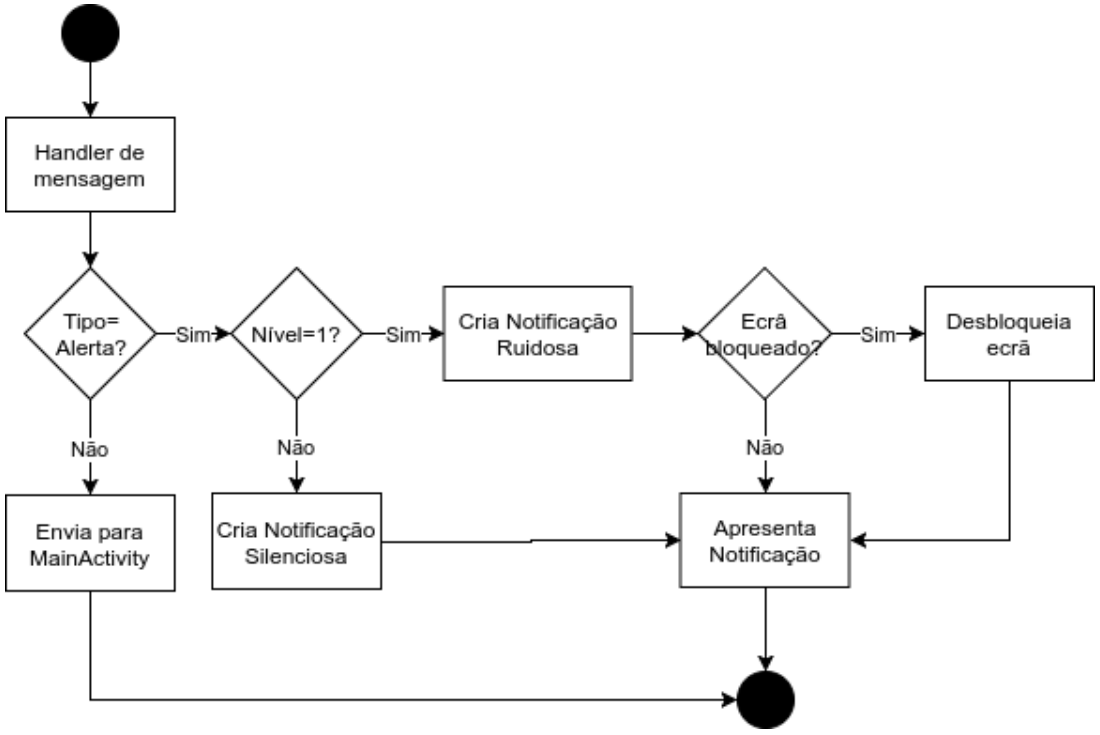


Figura 7.27: Fluxograma do Handler de mensagens.

A classe *NotificationUtils* foi criada para dar apoio a *AlertNotificationHandler* com as tarefas de criação dos diferentes tipos de notificação e respectivos acessórios tais como emissão de sons, imagens, desbloqueio de ecrã, etc. Relativamente à criação das notificações, é de salientar que todas as notificações incluem, na sua composição, uma chamada à *MainActivity*. Esta chamada, que é tecnicamente um *Intent* de Android [73], pode enviar comandos para a *MainActivity*. No caso das notificações de primeira responsabilidade, é apresentada ao operador, uma notificação com opções para aceitar ou rejeitar, como pode ser visto na Figura 7.28. No caso de outras notificações, não é enviado nenhum comando de reposta quando a notificação é selecionada para visualizar o alerta.

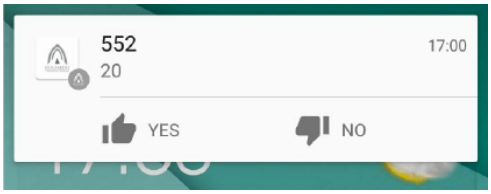


Figura 7.28: Notificação de alerta de primeira responsabilidade.

7.2.3 MainActivity

MainActivity é a classe responsável pela interligação de todos os módulos da aplicação. Esta classe deriva da classe *Activiy* do Android [74]. Sempre que é iniciada, mesmo antes do *Login*, a aplicação começa sempre por instanciar esta classe. É ela que determina se o operador tem a sessão válida quando pretende visualizar um alerta referido numa notificação. O menu da aplicação é instanciado nesta classe. As tarefas implementadas para *MainActivity* são:

- BroadcastReceiver
- Menu
- Navegação
- Controlo de VPN
- Auto Logout
- Auto restart

Para receber os *Intents* com as mensagens enviadas pelo serviço através de *AlertNotificationHandler* é necessário instanciar o *BroadcastReceiver*. No método *onReceive* do *BroadcastReceiver* faz-se a triagem das mensagens, para, no caso das notificações, serem apresentados os respectivos alertas e actualizações do *Modelo Operador* no caso de mensagens de actualização de turno.

O menu consiste numa *NavigationView*, representada na Figura 7.29, instanciada na *MainActivity* onde, para além das opções de menu, também são apresentadas informações do operador, tais como o turno e nome de utilizador.

Para responder à selecção de opções do menu, é implementado o método *onNavigationItemSelected* do interface *NavigationView.OnNavigationItemSelectedListener*. No menu, cada opção corresponde a um Fragment diferente, com excepção da opção de *Logout*. Assim, a resposta de cada opção é, basicamente a instanciação do *Fragment* respectivo. No caso da opção de *Logout*, essa consiste em eliminar os dados e sessão do utilizador, interromper a VPN e iniciar a *LoginActivity* apresentando a *Vista Login*.

As várias vistas da aplicação, são todas implementadas em *Fragments*[75], com excepção da *Vista Login*. A navegação dentro da aplicação consiste em poder retroceder ao *Fragment* anterior quando o utilizador selecciona o *BackButon* do dispositivo [76]. Para implementar essa funcionalidade, foi usado o *FragmentManager* do Android [77] para registar na sua pilha cada

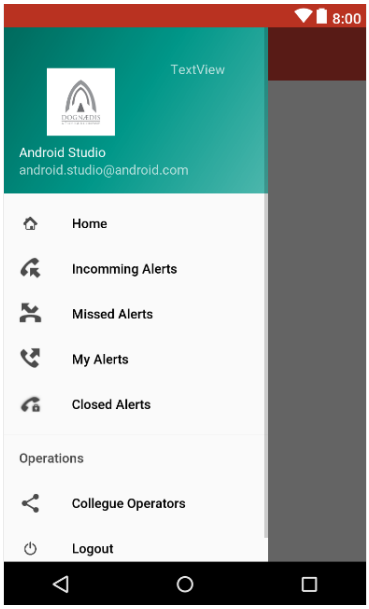


Figura 7.29: Menu da aplicação móvel.

Fragment instanciado. De seguida, com *override* do método *onBackPressed* de *Activity* chama-se o método *popBackStack* do *FragmentManager* para aparecer o *Fragment* anterior da pilha, quando o utilizador usa o *BackButton* do dispositivo.

O controlo da VPN consiste em usar o cliente de VPN instalado no sistema, para activar ou desactivar a VPN. O cliente VPN necessário é obrigatoriamente a aplicação *OpenVPN for Android* [78] pois assim foi determinado pela empresa. Partindo do principio que o cliente VPN tem um perfil configurado e a funcionar, existe a possibilidade da aplicação poder iniciar e terminar a VPN sempre que necessário. Activar antes do *Login*, desactivar depois do *Logout*. Para isso pode-se tirar partido do interface Android Interface Definition Language (AIDL) disponível no *OpenVPN for Android*. O AIDL permite que as aplicações Android disponibilizem algumas funcionalidades a outras aplicações Android, através de Inter-Process Communication (IPC). Neste caso, o *OpenVPN for Android* dispõe das funcionalidades pretendidas através de AIDL, por isso, resta implementar o interface *IOpenVPNAPIService*[79] na nossa aplicação e usar os métodos do interface para iniciar a VPN (com o perfil pré-configurado) ou parar a VPN quando necessário.

O Auto Logout é uma funcionalidade que visa, simultaneamente, a segurança e a poupança de recursos. Consiste em contar 5 minutos de inactividade na aplicação por parte do utilizador. Terminado o tempo, a aplicação faz *Logout* ao utilizador e, consequentemente, interrompe a VPN. Desta forma, previne-se o uso inadvertido da aplicação por terceiros e reduz-se o tempo desnecessário de uso da VPN. Este contador de tempo é implementado em uma *thread* separada da principal, onde, na principal, o contador é reiniciado sempre que o utilizador interage com a aplicação, através do método *onUserInteraction* de *Activity*.

Durante a utilização da aplicação, é possível surgirem erros inesperados que levem a aplicação a terminar abruptamente. Caso isso aconteça, a aplicação deixaria de receber alertas, pois o serviço que recebe as notificações estaria parado. Para mitigar este problema, surge a necessidade de implementar um mecanismo de *auto-restart* da aplicação em caso de erros inesperados. Os erros inesperados são, em linguagem Java, excepções não tratadas, que são lidadas através do método *uncaughtException* de *UncaughtExceptionHandler* [80]. Para poder implementar o método *uncaughtException* com código que permite o auto-restart, foi criada a classe *MyExceptionHandler* que implementa o interface *Thread.UncaughtExceptionHandler* e definida no *onCreate* de MainActivity com *Thread.setDefaultUncaughtExceptionHandler*. A implementação do *restart* da aplicação durante da intercepção do *crash* consiste em criar um novo *Intent* com a *MainActivity* para relança-la antes de terminar a anterior, causadora do erro fatal.

7.2.4 Vista Login

O *Login* é a vista da aplicação que aparece visível quando o utilizador não está autenticado. Nela, existe um formulário onde o operador insere o seu nome de utilizador e palavra passe para se autenticar. Para implementar a *Vista Login*, foram implementadas as seguintes tarefas:

- Formulário de autenticação
- LoginActivity

Conforme se pode ver na Figura 7.30, o formulário de login inclui os campos *username* (aqui preenchido com o nome catmobile), *password* e os botões *sign in* para validar. Por requisito de segurança, nenhum dos campo é guardado como predefinição no sistema, obrigando sempre o operador a preencher os campos necessários.

LoginActivity é a *Activity* que gere o processo de *Login*. Este processo é simples. Consiste apenas em enviar as credenciais através do *Pedido Login* ao servidor, e aguardar pelo seu resultado. Caso seja um resultado positivo, inicia a *MainActivity*, com a *Vista Dashboard* caso contrário, apresenta o resultado no campo *Log output* do formulário de autenticação.

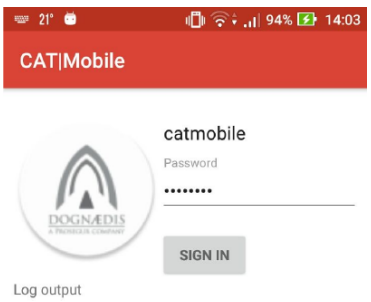


Figura 7.30: Login na aplicação móvel.

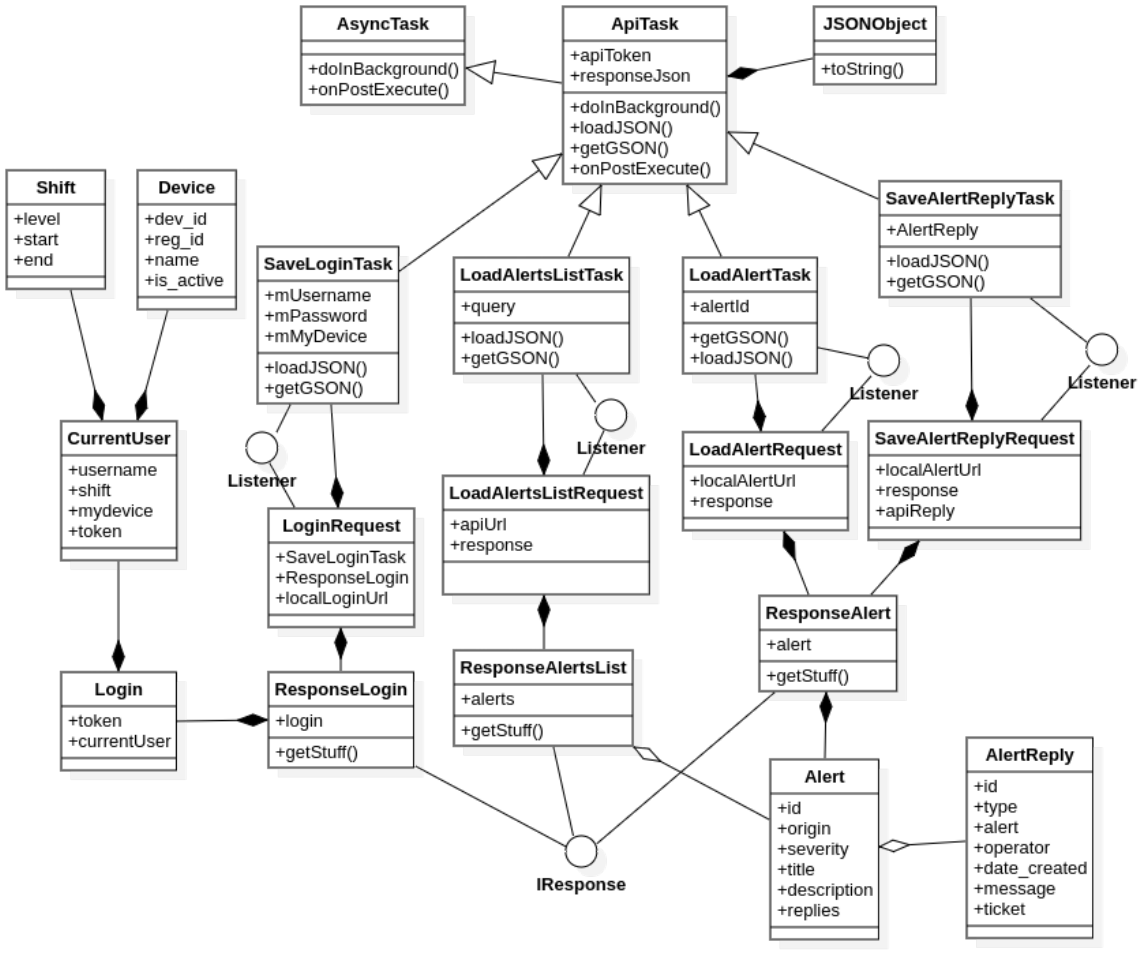


Figura 7.31: Diagrama de Classes dos pedidos HTTP da aplicação móvel ao servidor.

7.2.5 Pedido Login

O *Pedido Login* é usado pela *LoginActivity* para validar credenciais no servidor. O pedido ao servidor, implementado por este módulo tem, na sua estrutura, vários componentes que podem ser reaproveitados por outros módulos que também fazem pedidos ao servidor. A Figura 7.31 representa o diagrama de classes que implementa essa estrutura. As tarefas implementadas são:

- ApiTask
- SaveLoginTask
- ResponseLogin
- LoginRequest

A classe *ApiTask* é uma classe abstracta que implementa o método *doInBackground* da classe super *AsyncTask* para chamar o método *loadJSON* implementado na classe inferior *SaveLoginTask*, para obter a resposta em formato JSON no atributo *responseJson* e devolver o método *getGSON* da classe inferior *SaveLoginTask* aplicado ao *responseJson*.

A classe *SaveLoginTask* é a concretização da classe abstracta *ApiTask* onde são implementados métodos *loadJSON* e *getGSON* que são específicos do pedido a implementar. No método loadJSON é especificado o *endpoint* e o verbo HTTP a usar no pedido. No método *getGSON* é feita a conversão do objecto JSON para um objeto da classe *ResponseLogin*. Para fazer a conversão entre JSON e objectos Java é usada a biblioteca *Gson* [81].

ResponseLogin é a classe modelo que armazena em formato objeto Java, a reposta ao pedido em JSON, conforme especificado pela biblioteca Gson. Esta classe é composta por uma instância de classe *Login* e pelo método *getStuff* que devolve o conteúdo de todos objectos que implementam *IResponse*. A classe Login é, de facto, o conteúdo da resposta do pedido. Este contem o *token* que, será usado para autenticar os restantes pedidos pedidos, e uma instância de *CurrentUser*. A classe *CurrentUser* contém os dados do operador, nomeadamente, o turno em classe *Shift* e o dispositivo móvel em classe *Device*. O objeto da classe *CurrentUser* terá eventualmente actualizado o objeto *Shift*, caso a aplicação receba do servidor, uma actualização de turno.

A classe *LoginRequest* é que permite a instanciação do *Pedido de Login*. Ela é instanciada quando o utilizador clica no botão de login. Nesta classe é instanciada a *SaveLoginTask* com o endpoint para *login* no servidor, *username* e *password*. Uma vez criado o pedido, o sistema envia-o com uma chamada síncrona da qual fica a espera da resposta ou de eventual *timeout*. A resposta é obtida na *LoginActivity* através do método *onLoaded* ou *onError* do interface *SaveLoginTask.Listener*.

7.2.6 Vista Dashboard

A *Vista Dashboard* é a vista inicial da aplicação depois de autenticado o utilizador. Nela é são visíveis dados de informação resumida sobre o sistema relativo ao operador autenticado. As informações visíveis actualmente são apenas a data e hora do último *Login* do operador, contudo mais informações poderão vir a constar nesta vista, ficando para trabalho futuro.

Para apresentar a data e hora do último Login do operador, usou-se a resposta do servidor ao Pedido de Login para enviar essa informação.

7.2.7 Vista Lista de Alertas

A Lista de Alertas é uma vista composta pela lista de alertas resumidos obtida do servidor. Esta lista tem uma aparência inspirada pela aplicação Gmail para Android, que desliza na vertical e permite a selecção de vários alertas em simultâneo, para, através de uma única resposta, res-

ponder a vários alertas em simultâneo. Esta lista possui ainda, opções de filtro para os campos, severidade, cliente, origem e campo de texto livre Ao seleccionar um alerta da lista, é executada a *Vista Alerta* para visualizar o alerta seleccionado.

A Figura 7.32 representa a lista de alertas. Nesta vista, são visíveis os alertas na sua forma resumida. O circulo colorido contem a primeira letra da origem do alerta e a cor representa o nível de severidade. O texto central visível do alerta resumido é composto pela origem do alerta, título e resumo. Na lateral direita, aparece a hora do alerta no caso de ser no dia corrente, ou a data caso seja anterior. Aparece também o nome do operador responsável, caso exista, assim como a estrela amarela que representa a atribuição do operador responsável. Por fim, o número de respostas já dadas ao alerta. No topo da listagem, estão os filtros, nomeadamente, filtro de origem com opção das origens existentes na listagem, filtro por data, com opção de próprio dia, semana ou mês e um *slider* para filtrar os alertas por severidade. Existe ainda a opção de pesquisar por texto livre. Todos estes filtros são executados na própria aplicação, sobres os alertas obtidos a partir do *Pedido Lista de Alertas*.

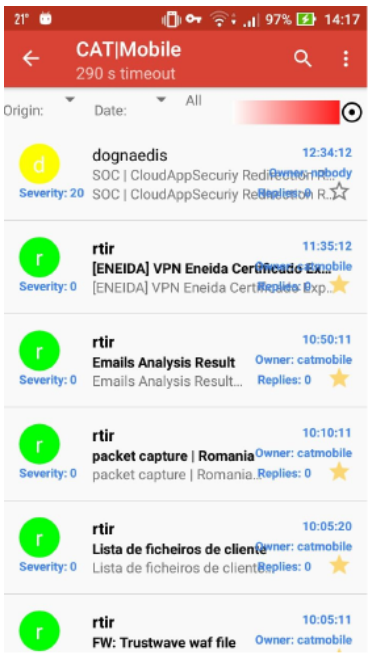


Figura 7.32: Vista da Lista de Alertas na aplicação móvel.

7.2.8 Pedido Lista de Alertas

O *Pedido Lista de Alertas* consiste no pedido feito ao servidor da lista de alertas resumidos. Este pedido, iniciado por *AlertsListFragment* usa a mesma estrutura dos outros pedidos ao servidor, conforme se pode ver na Figura 7.31. As tarefas implementadas para o pedido são:

- LoadAlertsListTask
- ResponseAlertsList
- LoadAlertsListRequest

A classe *LoadAlertsListTask* implementa no método *loadJSON* o pedido ao servidor com o verbo GET, ao *endpoint /api/v1/alerts/* definido na Secção 6.3.1. A resposta em formato JSON desse pedido, é convertida por Gson em objeto Java da classe *ResponseAlertsList*.

A classe *ResponseAlertsList* é usada para instanciar a resposta do pedido HTTP de lista de alertas em objeto Java. Esta classe é composta por uma lista de objectos da classe Alert onde são instanciados cada um dos alertas. Apesar da classe Alert conter todos os atributos que definem um alerta completo, no caso da lista de alertas, apenas são preenchido os campos usados na listagem, ficando vazios os restantes. Ao fazer o pedido dos alertas na sua forma reduzida, está-se a satisfazer o requisito de poupança de recursos, neste caso simultaneamente de dados e do tempo necessário à transferência desses dados. De notar que um alerta completo é composto, para além dos dados próprios que o caracterizam, também pela lista de respostas que pode ter eventualmente. Estes dados, não sendo úteis na listagem dos alertas, não fazia sentido estarem a ser transferidos, usando recursos desnecessários.

A classe *LoadAlertsListRequest* é instanciada em *AlertsListFragment* para iniciar o pedido através de *LoadAlertsListTask* e criar a lista de alertas, com a resposta obtida.

7.2.9 Vista Alerta

Estando os alertas visíveis na listagem de alertas de forma resumida, o operador selecciona um alerta da lista para visualizar o alerta na integra. Para aceder ao alerta completo, incluindo as suas respostas, optou-se por fazer o pedido ao servidor dos dados desse alerta somente quando o operador pretende visualizar o alerta. As tarefas implementadas para a Vista Alerta são:

- AlertFragment
- Resposta a alerta

A visualização do alerta, implementada em *AlertFragment*, como se pode ver no exemplo da Figura 7.33 consiste na apresentação dos dados do alerta, da lista de eventuais respostas e do botão de resposta ao alerta. A lista de respostas é deslizante na vertical e apresenta toda a informação relativa a cada resposta, nomeadamente autor, descrição e o tipo de resposta, como referido na Secção 7.1.12,

O botão para resposta ao alerta, no canto inferior direito da Figura 7.33, faz surgir o formulário de resposta. O formulário apenas é visível se o operador actual for o operador responsável

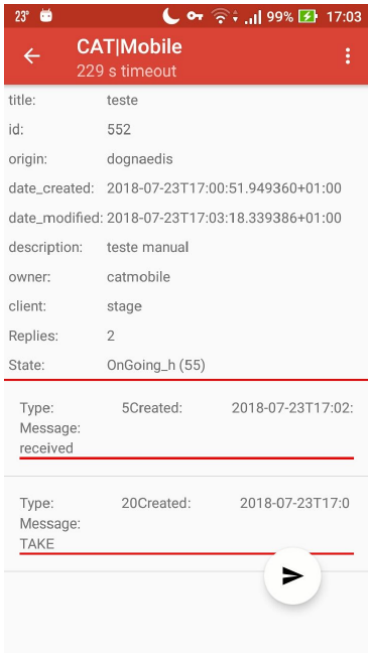


Figura 7.33: Vista do Alerta na aplicação móvel.

pelo alerta, caso contrário, aparece uma mensagem a informar que não pode responder ao alerta. Caso seja a primeira resposta ao alerta, o operador tem opção de aceitar, conforme a Figura 7.34 ou declinar, conforme a Figura 7.35. Caso não seja a primeira resposta ao alerta, a reposta, conforme a Figura 7.36, tem de conter texto descritivo para ser válida e tem opção de criar *ticket* e de conclusão.

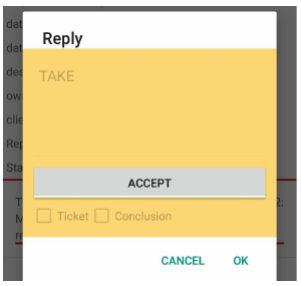


Figura 7.34: Aceitação de Alerta na aplicação móvel.

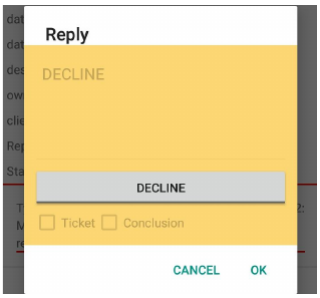


Figura 7.35: Declinação de Alerta na aplicação móvel.

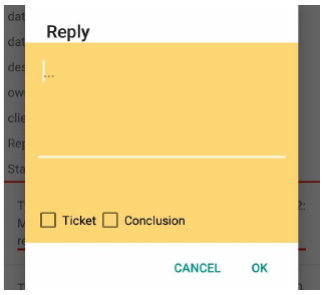


Figura 7.36: Resposta a Alerta na aplicação móvel.

7.2.10 Pedido Alerta

O *Pedido Alerta* consiste no pedido feito ao servidor de um determinado alerta. Este pedido pode vir da selecção de um alerta da lista de alertas ou de uma notificação recebida. De uma

forma ou de outra, o alerta pedido é sempre identificado pelo seu atributo "id". Em resposta ao pedido, o servidor devolve um objeto em formato JSON com todos os atributos do alerta, incluindo a lista de eventuais respostas que o alerta possa ter. A estrutura de classes deste pedido, é idêntica à dos pedidos anteriores, como se pode ver na Figura 7.31. As tarefas implementadas para o Pedido Alerta são:

- LoadAlertTask
- ResponseAlert
- LoadAlertRequest

A classe *LoadAlertTask* executa um pedido GET ao *endpoint /api/v1/alerts/(id)* referido na Secção 6.3.1, com especificação do valor "id"relativo ao alerta que pretende obter. A resposta do servidor, em formato JSON é convertida em objeto Java da classe *ResponseAlert* através de Gson.

A classe *ResponseAlert* instância as respostas dos pedidos de *LoadAlertTask*. Esta classe é composta por uma instância da classe *Alert*. A classe Alert, por sua vez contém todos os atributos que descrevem o alerta, incluindo uma lista de *AlertReply*. A classe *AlertReply* armazena todos os atributos de cada resposta que um alerta possa ter.

Para pedir o alerta ao servidor, a classe *LoadAlertRequest* é instanciada na lista de alertas ou na *MainActivity* por causa de uma notificação recebida. Em ambos os casos, o alerta como resposta do pedido é usado na instância de *AlertFragment* para visualizar o alerta.

7.2.11 Pedido Resposta

O *Pedido Resposta* consiste no pedido de resposta dada pelo operador responsável ao alerta. Esta opção, que aparece disponível na *Vista Alerta*, é executada depois de preenchidos e validados os campos do formulário de resposta. Depois do enviado ao servidor, é devolvido o alerta na integra, com a lista de respostas actualizada e eventualmente com o estado do alerta também actualizado se a resposta tiver alterado o estado do alerta, para actualizar a informação visível na *Vista Alerta*. As tarefas implementadas para o pedido de resposta a alerta, são:

- SaveAlertReplyTask
- SaveAlertReplyRequest

De modo similar aos pedidos anteriores e como representado na Figura 7.31, o método *loadJSON* da classe *SaveAlertReplyTask* faz o POST ao *endpoint /api/v1/alerts/(id)/replies/*, indicando no *endpoint* o *ID* do alerta ao qual se pretende dar a resposta, tal como referido na Secção 6.3.1.

A classe *SaveAlertReplyRequest* é instanciada pelo *alertDialogBuilder* de *AlertFragment* quando o utilizador clica em "OK"no formulário de resposta ao alerta. A resposta do servidor, que consiste em um objeto Alert completo, substitui o alerta visível no *AlertFragment*.

Capítulo 8

Testes e Qualidade

Neste capítulo é relatada a fase de testes do projecto. Esta fase teve início durante a implementação do projecto, numa altura em que já estavam implementadas funcionalidades possíveis de testar. Os testes de software realizados em simultâneo com a implementação são chamados de Testes de Regressão [82]. Estes testes, efectuados de forma automatizada, são uma ferramenta muito útil durante o desenvolvimento de software pois permitem verificar se funcionalidades implementadas anteriormente continuam a funcionar depois de implementadas novas funcionalidades que interferentes. Os testes tiveram também a utilidade de salientar problemas imprevistos que necessitaram de resolução, nomeadamente, dentro do processamento de alertas e de notificações.

8.1 Infraestrutura

Para efectuar os testes automatizados e manuais, durante a implementação do projecto, foi usado o modo de servidor local disponível no Django em conjunto com as funcionalidades de testes unitários e de integração disponíveis na *framework*. Uma vez terminada a implementação da maioria dos requisitos, para a solução ser usada em ambiente real, foi configurado o servidor Linux onde o sistema permanecerá em produção. A distribuição Linux escolhida para o sistema operativo foi Debian, versão 9.0, por ser a distribuição preferida pela empresa para este tipo de projectos e ser a versão Debian *stable* do momento [83]. A distribuição Debian preza pelo seu gestor de pacotes que, ao instalar aplicações a partir de repositórios oficiais, instala também configurações e *scripts* de gestão, para além da própria aplicação, o que agiliza bastante os procedimentos de instalação. Nesse servidor foram instalados e configurados todos os componentes necessários ao sistema desenvolvido, nomeadamente:

- Servidor HTTP
- Servidor Aplicacional
- Sistema de Arranque
- Base de Dados

- Agendamento de Tarefas

O serviço de HTTP é a única forma de entrada de dados no Cat|Mobile. Este serviço é usado pelo Interface de Administração Web e pela API REST. Para instalar o servidor, usou-se o Nginx por ser o serviço usado na empresa neste tipo de projectos, facilitando a manutenção futura. Para que toda a comunicação com o servidor seja segura, o Nginx foi configurado com Hyper Text Transfer Protocol Secure (HTTPS). Assim, tanto o interface Web como a API REST têm garantida a encriptação de todos os dados que passam via HTTPS. Para interligar a aplicação Django com o servidor HTTP, foi configurado o serviço aplicacional uWSGI. Este serviço, também ele, usado na empresa para outros projectos, é parte integrante do Nginx, pelo que a sua configuração é bastante amigável.

O uWSGI é um serviço que interpreta código Python em *single-thread*, por *default*, pois não tem o Global Interpreter Lock (GIL) activo [84]. Dado que a aplicação desenvolvida tira partido de *multi-thread*, foi necessário invocar a opção que activa o GIL para permitir o uso de *threads*, *enable-threads = true*. A situação que levou à descoberta deste comportamento em relação a *threads*, foi um problema difícil de detectar pois a aplicação continuava a correr mesmo em *single-thread*, apenas os resultados não coincidiam com os esperados durante os testes. De acordo com [85], o uso de *threads* em Python é desaconselhado por ser um factor de degradação de desempenho, podendo mesmo a vir a ser um *bottleneck* do sistema. Por esta razão e por não haver tempo disponível no âmbito do estágio, o autor deixa para trabalho futuro o eventual *upgrade* de tecnologia para um sistema alternativo ao uso de *threads*, caso a aplicação venha a sofrer degradação de desempenho devido ao uso do GIL.

Com o serviço aplicacional configurado, resta configurar o seu auto-arranque no sistema. Para isso foi criado um *script* que permite a gestão da configuração uWSGI criada para o Cat|Mobile. O auto-arranque do *script* é gerido pelo *Systemctl* que é um dos componentes do Systemd [86]. O Systemd é o sistema de gestão de serviços actualmente usado no Debian e que veio substituir o Init[87] usado até à versão 7.0 (Wheezy) [88].

O motor de base de dados escolhido foi PostgreSQL por também ser o SGBD de escolha na empresa, permitindo uma futura possível integração da base de dados da aplicação com o sistema de base de dados existente na empresa, usufruindo assim da manutenção regular e de sistemas de *backup* existentes.

O agendamento de tarefas do qual o Cat|Mobile depende para tarefas assíncronas implementadas no sistema, foi efectuado com o registo de comandos em *crontab* afim de serem executados periodicamente.

8.2 Web e API

O Interface de Administração Web e a API REST constituem os meios possíveis de alterar os dados e funcionamento da aplicação de servidor. Ao testar estes dois componentes, estão-se a testar todas as funcionalidades do lado do servidor do Cat|Mobile. Os testes efectuados foram, basicamente, sobre a utilização normal das funcionalidades e algumas possíveis más utilizações de forma a testar também a robustez e segurança da aplicação. Uma parte destes testes foi realizada manualmente, através da simulação de utilização por parte do administrador no interface Web e também da ferramenta Postman [22], utilizada para testar a API REST. Alguns testes, mais recorrentes, foram automatizados em código, usando funcionalidades do Django para esse efeito [89]. Para automatizar a simulação da utilização do Interface de Administração Web, foi usada a biblioteca Selenium para Python [90], que permite efectuar testes do tipo *black-box* a elementos HTML.

8.3 Aplicação Móvel

A aplicação móvel foi testada manualmente pelo autor e também automaticamente com recurso à ferramenta de testes Appium [23]. O Appium é uma ferramenta que permite integrar os testes de uma aplicação móvel com os testes de uma aplicação de servidor. Funciona como um orquestrador que coordena acções em simultâneo no dispositivo móvel e na aplicação de servidor. As acções possíveis de realizar no dispositivo móvel, com o Appium, são do tipo *black-box*[91] em que é possível aceder a todas as funcionalidades do dispositivo, para além da aplicação alvo dos testes. Ou seja, é possível interagir com as notificações, que se encontram fora da aplicação, usar o teclado do dispositivo para preencher formulários, etc. No entanto, como não acede ao código-fonte da aplicação móvel, não verifica, por exemplo, o estado das variáveis. Esse tipo de testes realiza-se através de com JUnit[92]. Contudo, é uma ferramenta muito poderosa pois permite efectuar testes com sequência de acções que interagem tanto com o servidor como com o dispositivo móvel. Pode inclusivamente testar a aplicação em vários dispositivos móveis em simultâneo e concorrencialmente com distinção das acções realizadas em cada dispositivo, podendo os dispositivos serem reais ou virtuais, por emulador.

8.4 Processamento de alertas e notificações

As máquinas de estados do processamento de alertas e notificações são componentes muito importantes no sistema pois deles depende grande parte do sucesso do projecto. São estes processos que garantem o objectivo principal do projecto que é fazer chegar a notificação do alerta ao operador responsável ou ao seu substituto, o mais rapidamente possível. Estes processos usam grande parte dos componentes anteriormente testados, mas isso só não basta, tem de ser

testados também os algoritmos que fazem variar os estados das máquinas de estados. Para isso foi necessário encontrar todas as combinações possíveis de variação de estados que possam ocorrer.

Cada variação de estado dentro do processo, equivale a um caminho de teste. Para melhor organizar todos os caminhos possíveis, transformaram-se as máquinas de estados em Control Flow Graph (CFG). O grafo obtido dessa transformação inclui a máquina de estados do processamento de alertas representado pela Figura 6.11 e a máquina de estados do processamento de notificações representado pela Figura 6.12. Cada estado dessas máquinas deu origem a um nó do CFG. Foram acrescentados alguns nós para que, nos casos em que um estado pode voltar a si próprio, houvesse um nó intermédio que identificasse o motivo dessa alteração de estado. Assim, o CFG obtido está representado na Figura 8.1. Aos nós do grafo, foram atribuídas as seguintes designações de modo a obter uma "história"através de cada caminho de teste:

1. Selecciona operador
2. Sem operador
3. Inicia processo de notificação
4. Selecciona dispositivo
5. Sem dispositivo
6. Notificação expirada
7. Aguarda entrega
8. Não entregue
9. Entregue
10. Aguarda visualização
11. Expirada
12. Aguarda aceitação
13. Declinada
14. Tratamento
15. Conclusão

Recorrendo à aplicação online *Graph Coverage Web Application* [93] foram gerados caminhos de teste baseados em *Prime Path Coverage* [94]. Este tipo de teste de cobertura consiste em gerar todas as combinações possíveis de caminhos entre o primeiro e último nó, com eventuais ciclos a serem percorridos, uma vez no máximo, em cada caminho.

Numa primeira abordagem, considerados todos os nós existentes, foram obtidos 52 caminhos. Numa segunda abordagem, foram procurados os caminhos impossíveis de percorrer, denominados *infeasible sub paths* [95] .

Relativamente ao *sub-path* [4,5,4], a passagem do nó 4 para o 5 ocorre num cenário em que o operador seleccionado não tem dispositivo móvel associado. Isso pode acontecer por dois motivos, nunca fez Login na aplicação móvel ou outro operador fez um Login mais recente num dispositivo anteriormente associado a ele. Como consequência, volta ao nó 4. Neste momento, o sistema está preparado para poder aguardar um pequeno período de tempo para dar

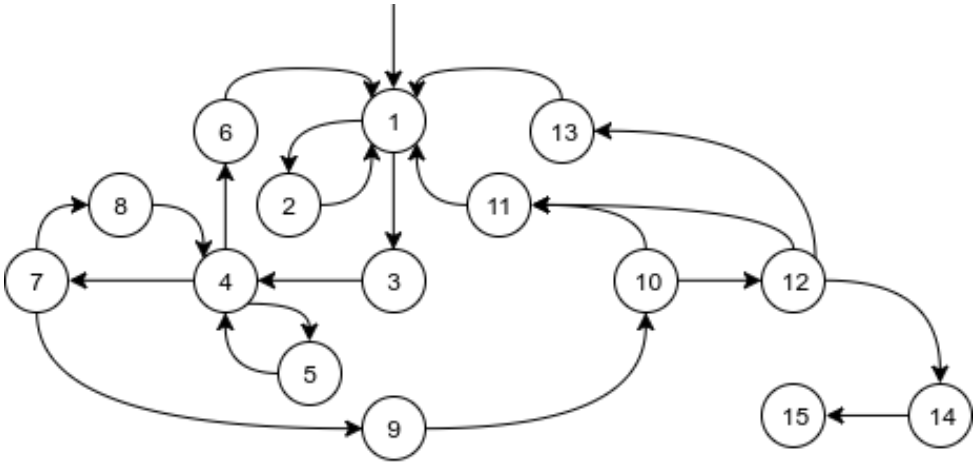


Figura 8.1: Control Flow Graph do processamento de alertas e notificiações.

oportunidade do operador sem dispositivo ser contactado, por meio alternativo, no sentido de ser solicitado a fazer Login na aplicação móvel de modo a ficar com dispositivo associado e, assim, poder receber a notificação que lhe é destinada. Contudo, actualmente, como o mecanismo de contacto alternativo não se encontra implementado (tendo ficado para trabalho futuro), o tempo de espera no nó 4 é nulo, pelo que não existe hipótese do operador sem dispositivo poder recuperar a notificação perdida, que corresponderia à passagem para o nó 7. Por este motivo, o *sub-path* [4,5,4,7] passa a ser um *infeasible sub paths*.

No nó 7, o sistema está preparado para aguardar o recibo de entrega da notificação. Como referido na Secção 7.1.10, idealmente, o recibo seria da entrega no dispositivo móvel mas, como a implementação actual não permite essa funcionalidade, o recibo é apenas da entrega no FCM. Ora, como todos os dispositivos móveis que usam a aplicação Cat|Mobile estão registados no FCM (dado que o registo é automático) esta confirmação de entrega é irrelevante e apenas existe para no futuro poder ser implementada como recibdo de entrega no dispositivo. Assim, a eventual falha de entrega, correspondente à passagem para o nó 8, nunca chega a acontecer enquanto os recibos forem de entrega no FCM e não do dispositivo. Por este motivo, o *sub-path* [7,8,4] é também um *infeasible sub paths*.

Não tendo sido encontrados mais *infeasible sub paths*, foi recalculada a *Prime Path Coverage*, desta vez tendo em conta os *infeasible sub paths* [4,5,4,7] e [7,8,4]. O resultado obtido, de 28 caminhos, pode ser consultado no Apêndice B.

Cada um desses testes corresponde a uma execução do processamento de alerta e notificação. Com a execução destes testes, em modo *black-box*, está a ser testada a quase totalidade do sistema, desde o registo de operadores passando pelo registo de origem de alertas, registo de turnos e escalonamento, registo de dispositivo através da utilização da aplicação móvel, recepção de alertas, emissão de notificações até à resposta a alertas. Através de todas estas tarefas, é testado todo o ciclo de vida de um alerta e funcionalidades acessórias.

O teste número 22 da lista, por exemplo, é composto pela seguinte sequência de nós da Figura 8.1: [1, 2, 1, 2, 1, 3, 4, 7, 9, 10, 12, 14, 15], que corresponde à sequência de nós representado na Tabela 8.1. Neste teste pode-se verificar, por exemplo, que o operador é seleccionado 3 vezes, correspondendo à selecção final do operador de terceira responsabilidade, testando, assim todo o processo de escalonamento dos operadores.

Tabela 8.1: Sequência de nós do teste 22 de *Prime Path Coverage*.

Ordem	Nó	Descrição
1	1	Seleciona operador
2	2	Sem operador
3	1	Seleciona operador
4	2	Sem operador
5	1	Seleciona operador
6	3	Inicia processo de notificação
7	4	Selecciona dispositivo
8	7	Aguarda entrega da notificação
9	9	Entregue
10	10	Aguarda visualização do alerta
11	12	Aguarda aceitação do alerta
12	14	Tratamento
13	15	Conclusão

Com todos os testes transcritos para sequências legíveis de estados, foi possível realizar todos os testes de *Prime Path Coverage* e assim cobrir todas situações possíveis que possam ocorrer com este sistema, garantindo que o sistema se encontra preparado para entrar em produção com a confiança desejada.

8.5 Conclusão

Neste capítulo foram demonstrados os testes efectuados ao produto final do projecto de acordo com os requisitos iniciais. Com estes testes ficou demonstrada a qualidade obtida com a implementação do projecto e a confiança que se pode esperar da solução.

Capítulo 9

Conclusão

Com este capítulo, dá-se por concluído o Relatório de Estágio, onde o autor tentou usar o espaço que teve disponível para descrever, com algum detalhe, as situações que entendeu terem conteúdo mais relevante, dentro da estrutura de base típica deste tipo de documento.

9.1 Objectivos cumpridos

Na opinião do autor, o estágio foi considerado muito interessante tendo em conta o desafio que lhe foi colocado com a proposta de estágio que esteve na origem do projecto. A complexidade do desafio foi bastante cativante e bastante útil a oportunidade de contactar com tecnologias que lhe eram estranhas e com as quais ficou familiarizado e motivado a poder continuar a usá-las profissionalmente. Continuando com a opinião, o autor entende que o estágio foi concluído com sucesso, os objectivos principais foram atingidos, grande parte das funcionalidades foram implementadas e foram criadas bases sólidas para *upgrades* futuros. Apesar de, por altura da escrita deste documento, ainda não ter sido usado em ambiente real, prevê-se que, pelos testes realizados, o sistema responda conforme esperado. Alguns detalhes acessórios ficaram por implementar devido ao rígido *deadline* de conclusão do projecto, contudo, a arquitectura com que foi desenvolvida a solução demonstrou, durante o desenvolvimento, a relativa facilidade com que podem ser adicionadas novas funcionalidades, acrescentando ou modificando lógica às existentes.

9.2 Objectivos por cumprir

Devido à limitação temporal para o desenvolvimento do projecto e a eventuais deslizes de tempo em tarefas mais morosas, existem alguns detalhes do produto que, no resultado final, ficaram em falta ou que poderiam ser melhorados, com um pouco mais de tempo investido, pois não necessitam de intervenção profunda para ficarem concluídos. São estes, os seguintes:

- Interface de Administração - Dados estatísticos relativos a tempos de resposta a alertas;

- Processamento de notificações - Recibo de entrega no dispositivo;
- Aplicação móvel - *Dashboard* com mais informação;
- Aplicação móvel - Lista de Alertas, melhoramento dos filtros;
- Aplicação móvel - Melhoramento da usabilidade.

9.3 Trabalho futuro

Durante o desenvolvimento do projecto, foram encontradas possíveis novas funcionalidades que, sendo pertinentes, poderiam enriquecer a usabilidade, desempenho e manutenção do produto em futuras versões. Estas seriam:

- Processo de aviso para quando o operador responsável não dispõe do dispositivo móvel actual, da empresa, registado no sistema, impossibilitando-lhe o envio de notificações por parte do sistema. Por exemplo, enviar o aviso através de SMS para o telemóvel pessoal;
- Cliente de XMPP para FCM, para assim obter os recibos de entrega de notificações no dispositivo móvel. Ou, melhor ainda, implementação do envio de mensagens através de um servidor XMPP próprio, descartando por completo a necessidade de plataformas de terceiros;
- Serviço para recepção de *logs* da aplicação móvel, por exemplo para receber excepções não tratadas, de forma a poder usar informação dos próprios utilizadores para corrigir *bugs* na aplicação;
- Sistema proprietário, independente para actualizações automáticas da aplicação móvel;
- Sistema alternativo ao uso de *threads* na aplicação de servidor, devido a eventuais problemas que possam surgir com o uso de *threads* em Python, conforme referido na Secção 8.1.

Referências

[1] Django REST framework web site. http://www.django-rest-framework.org/, . Acedido em: 2018-03-20.

[2] RTIR best practical web site. https://bestpractical.com/rtir/, . Acedido em: 2017-12-15.

[3] Android-specific fields fcm documentation guides. https://firebase.google.com/docs/cloud-messaging/admin/send-messages#android-specific_fields, . Accessed: 2018-04-15.

[4] Dognædis sapotek. https://tek.sapo.pt/noticias/negocios/artigos/prosegur-compra-portuguesa-dognaedis, . Acedido em: 2018-07-15.

[5] IPN website. https://www.ipn.pt/, . Acedido em: 2018-07-15.

[6] Dognædis linkedin. https://www.linkedin.com/company/dognaedis/, . Acedido em: 2018-07-15.

[7] C. Vazquez and G. Simões. *Engenharia de Requisitos: Software Orientado ao Negócio*. Editora Brasport Livros Técnicos e Digitais, 1ª edition, 2016. ISBN:978-85-7452-790-1.

[8] Freshdesk web site. https://freshdesk.com/features. Acedido em: 2017-12-11.

[9] ngDesk web site. https://www.ngdesk.com/. Acedido em: 2017-12-11.

[10] C-Desk web site. http://www.cdesk.in/internal.aspx, . Acedido em: 2017-12-11.

[11] C-Desk service desk. http://www.cdesk.in/Service_Desk/Service_Desk.aspx, . Acedido em: 2017-12-15.

[12] What is HTTP Long Polling? pubnub blog. https://www.pubnub.com/blog/2014-12-01-http-long-polling/, . Acedido em: 2018-01-05.

[13] The WebSocket Protocol ieft tools. https://tools.ietf.org/html/rfc6455, . Acedido em: 2018-01-05.

[14] Firebase Cloud Messaging firebase products. https://firebase.google.com/products/cloud-messaging/, . Acedido em: 2018-03-15.

[15] GCM Cloud Connection Server (XMPP) google services. http://www.androiddocs.com/google/gcm/ccs.html, . Acedido em: 2018-06-14.

[16] Draw.io web site. https://www.draw.io/, . Acedido em: 2018-02-15.

[17] PyCharm web site. https://www.jetbrains.com/pycharm/, . Accessed: 2017-12-06.

[18] PostgreSQL project web site. https://www.postgresql.org/, . Accessed: 2017-12-10.

[19] Android Studio web site. https://developer.android.com/studio/, . Accessed: 2018-04-02.

[20] Django Project web site. https://www.djangoproject.com/, . Acedido em: 2017-12-15.

[21] Unit testing framework python documentadion. https://docs.python.org/3/library/unittest.html#module-unittest, . Accessed: 2018-07-10.

[22] Postman postman web site. https://www.getpostman.com/products, . Accessed: 2018-05-02.

[23] Appium web site. http://appium.io/, . Acedido em: 2018-05-08.

[24] Nginx web site. https://www.nginx.com/, . Accessed: 2018-07-02.

[25] The uWSGI project web site. https://uwsgi-docs.readthedocs.io/en/latest/, . Accessed: 2018-07-02.

[26] StarUML web site. http://www.staruml.io/, . Acedido em: 2018-02-15.

[27] Texmaker web site. http://www.xm1math.net/texmaker/, . Accessed: 2018-07-02.

[28] Trial and error wikipedia. https://en.wikipedia.org/wiki/Trial_and_error. Accessed: 2018-08-05.

[29] Django Groups django documentation. https://docs.djangoproject.com/en/2.0/topics/auth/default/#groups, . Acedido em: 2018-01-10.

[30] Django admin site django documentation. https://docs.djangoproject.com/en/2.1/ref/contrib/admin/, . Acedido em: 2017-12-16.

[31] Django Migratons django documentation. https://docs.djangoproject.com/en/2.0/topics/migrations/, . Acedido em: 2018-03-23.

[32] Django RunPython django documentation. https://docs.djangoproject.com/en/2.1/ref/migration-operations/#runpython, . Acedido em: 2017-12-15.

[33] Django Models django documentation. https://docs.djangoproject.com/ko/2.0/topics/db/models/, . Acedido em: 2017-12-15.

[34] Django User django documentation. https://docs.djangoproject.com/en/2.1/ref/contrib/auth/#user-model, . Acedido em: 2018-01-15.

[35] Django User Profile how to add user profile to django admin. https://simpleisbetterthancomplex.com/tutorial/2016/11/23/how-to-add-user-profile-to-django-admin.html, . Acedido em: 2018-01-20.

[36] Django Queries django documentation. https://docs.djangoproject.com/en/2.0/topics/db/queries/, . Acedido em: 2018-02-15.

[37] Django Signals django documentation. https://simpleisbetterthancomplex.com/tutorial/2016/11/23/how-to-add-user-profile-to-django-admin.html, . Acedido em: 2018-02-03.

[38] HTTP Status Code Registry web site. https://www.ietf.org/assignments/http-status-codes/http-status-codes.xml. Acedido em: 2018-04-6.

[39] CreateAPIView django rest framework api guide. http://www.django-rest-framework.org/api-guide/generic-views/#createapiview, . Acedido em: 2018-03-22.

[40] Mixins and Python ian lewis web site. https://www.ianlewis.org/en/mixins-and-python, . Acedido em: 2018-03-15.

[41] ModelSerializer django rest framework api guide. http://www.django-rest-framework.org/api-guide/serializers/#modelserializer, . Acedido em: 2018-04-11.

[42] TokenAuthentication django rest framework api guide. http://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication, . Acedido em: 2018-04-25.

[43] URL dispatcher django documentation. https://docs.djangoproject.com/ko/2.0/topics/http/urls/, . Acedido em: 2017-12-15.

[44] Class-based views django documentation. https://docs.djangoproject.com/ko/2.0/topics/class-based-views/, . Acedido em: 2017-12-20.

[45] RTIR4REST github. https://github.com/BikerDroid/RTIR4REST, . Acedido em: 2017-02-15.

[46] Custom Management Commands django documentation. http://https://docs.djangoproject.com/en/2.1/howto/custom-management-commands/, . Acedido em: 2018-05-06.

[47] Using state machines algorithms and programming approaches in python. https://www.ibm.com/developerworks/library/l-python-state/index.html, . Acedido em: 2018-03-25.

[48] Firebase Admin Python SDK documentation reference. https://firebase.google.com/docs/reference/admin/python/, . Acedido em: 2018-03-01.

[49] Android-specific field firebase guides. https://firebase.google.com/docs/cloud-messaging/admin/send-messages#android-specific_fields, . Acedido em: 2018-03-10.

[50] Add Firebase to your app firebase guides. https://firebase.google.com/docs/admin/setup#add_firebase_to_your_app, . Acedido em: 2018-03-02.

[51] Firebase Admin Python SDK firebase guides. https://firebase.google.com/docs/admin/setup#initialize_the_sdk, . Acedido em: 2018-03-03.

[52] Firebase Console web site. https://firebase.google.com/docs/admin/setup#initialize_the_sdk, . Acedido em: 2018-03-01.

[53] Lifetime of a message firebase guides. https://firebase.google.com/docs/cloud-messaging/concept-options#lifetime, . Acedido em: 2018-03-25.

[54] Firebase Cloud Messaging XMPP Protocol firebase documentation reference. https://firebase.google.com/docs/cloud-messaging/xmpp-server-ref, . Acedido em: 2018-04-10.

[55] SleekXMPP web site. http://sleekxmpp.com/, . Acedido em: 2018-04-15.

[56] Build App Server Send Requests firebase guides. https://firebase.google.com/docs/cloud-messaging/send-message, . Acedido em: 2018-04-16.

[57] Interpreting an upstream XMPP message firebase documentation reference. https://firebase.google.com/docs/cloud-messaging/xmpp-server-ref#interpret-upstream, . Acedido em: 2018-04-16.

[58] SearchFilter django rest framework api guide. http://www.django-rest-framework.org/api-guide/filtering/#searchfilter, . Acedido em: 2018-03-16.

[59] Custom generic filtering django rest framework api guide. http://www.django-rest-framework.org/api-guide/filtering/#custom-generic-filtering, . Acedido em: 2018-02-05.

[60] Serializer fields django rest framework api guide. http://www.django-rest-framework.org/api-guide/fields/, . Acedido em: 2018-02-25.

[61] Templates django documentation. https://docs.djangoproject.com/en/2.0/topics/templates/, . Acedido em: 2018-02-03.

[62] Form django documentation. https://docs.djangoproject.com/en/2.0/ref/forms/api/#django.forms.Form, . Acedido em: 2018-03-15.

[63] ModelForm django documentation. https://docs.djangoproject.com/en/2.0/topics/forms/modelforms/, . Acedido em: 2018-03-16.

[64] ModularAdmin license. https://github.com/modularcode/modular-admin-html/blob/master/LICENSE. Acedido em: 2018-05-03.

[65] Customizing authentication in Django django documentation. https://docs.djangoproject.com/en/2.0/topics/auth/customizing/, . Acedido em: 2018-02-01.

[66] django-session-security github. https://github.com/yourlabs/django-session-security, . Acedido em: 2018-02-01.

[67] Django Scheduler github. https://github.com/llazzaro/django-scheduler, . Acedido em: 2018-04-15.

[68] Django Form handling with class-based views django documentation. https://docs.djangoproject.com/en/2.0/topics/class-based-views/generic-editing/, . Acedido em: 2018-02-05.

[69] Django Widgets django documentation. https://docs.djangoproject.com/en/2.0/ref/forms/widgets/, . Acedido em: 2018-02-06.

[70] Form Assets (the Media class) django documentation. https://docs.djangoproject.com/en/2.0/topics/forms/media/#topics-forms-media, . Acedido em: 2018-02-06.

[71] Add Firebase to Your Android Project firebase guides. https://firebase.google.com/docs/android/setup, . Acedido em: 2018-04-16.

[72] FirebaseMessagingService firebase reference. https://firebase.google.com/docs/reference/android/com/google/firebase/messaging/FirebaseMessagingService, . Acedido em: 2018-05-17.

[73] Intents and Intent Filters android documentation guides. https://developer.android.com/guide/components/intents-filters, . Acedido em: 2018-04-07.

[74] Activities android documentation guides. https://developer.android.com/guide/components/activities/, . Acedido em: 2018-04-06.

[75] Fragments android documentation guides. https://developer.android.com/guide/components/fragments, . Acedido em: 2018-04-08.

[76] Providing proper Back navigation android documentation guides. https://developer.android.com/training/implementing-navigation/temporal, . Acedido em: 2018-04-27.

[77] FragmentManager android documentation reference. https://developer.android.com/reference/android/support/v4/app/FragmentManager, . Acedido em: 2018-04-09.

[78] OpenVPN for Android google play. https://play.google.com/store/apps/details?id=de.blinkt.openvpn&hl=en_US, . Acedido em: 2018-05-15.

[79] IOpenVPNAPIService ics-openvpn github. https://github.com/schwabe/ics-openvpn/blob/master/main/src/main/aidl/de/blinkt/openvpn/api/IOpenVPNAPIService.aidl. Acedido em: 2018-05-17.

[80] UncaughtExceptionHandler java platform, standard edition 7 api specification. https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.UncaughtExceptionHandler.html, . Acedido em: 2018-06-08.

[81] Gson github project. https://github.com/google/gson. Acedido em: 2018-04-15.

[82] M. Pezze and M. Young. *Software Testing and Analysis: Process, Principles and Techniques*. Wiley, 2008. ISBN 9780471455936. URL https://books.google.pt/books?id=mjEiAQAAIAAJ.

[83] Debian Releases debian.org. https://www.debian.org/releases/, . Accessed: 2018-08-02.

[84] Things to know (best practices and "issues") READ IT !!! uwsgi docs. https://uwsgi-docs.readthedocs.io/en/latest/ThingsToKnow.html?highlight=gilk. Accessed: 2018-07-04.

[85] GlobalInterpreterLock python wiki. https://wiki.python.org/moin/GlobalInterpreterLock, . Accessed: 2018-07-03.

[86] How To Use Systemctl to Manage Systemd Services and Units digitalocean community tutorials. https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units, . Accessed: 2018-07-04.

[87] Systemd Replaces Init in Linux tecmint. https://www.tecmint.com/systemd-replaces-init-in-linux/, . Accessed: 2018-07-02.

[88] Init debian.org wiki. https://wiki.debian.org/Init, . Accessed: 2018-08-02.

[89] Testing in Django django documentation. https://docs.djangoproject.com/en/2.0/topics/testing/, . Accessed: 2018-04-05.

[90] Selenium with Python web site documentation. https://selenium-python.readthedocs.io/, . Accessed: 2018-06-02.

[91] Black-box testing wikipedia. https://en.wikipedia.org/wiki/Black-box_testing, . Accessed: 2018-07-10.

[92] JUnit4 rules with Android Test android training. https://developer.android.com/training/testing/junit-rules, . Accessed: 2018-07-10.

[93] Graph Coverage Web Application web site. https://cs.gmu.edu:8443/offutt/coverage/GraphCoverage, . Accessed: 2018-08-10.

[94] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2016. ISBN 9781316773123. URL https://books.google.pt/books?id=58LeDQAAQBAJ.

[95] Infeasible paths in the context of data flow based testing criteria article. http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-65002006000200006, . Accessed: 2018-08-10.

Parte II

Apêndices

Apêndice A

Mapas de Requisitos

Nos mapas de requisitos podem ver-se as correspondências entre os módulos de arquitectura e os requisitos funcionais.

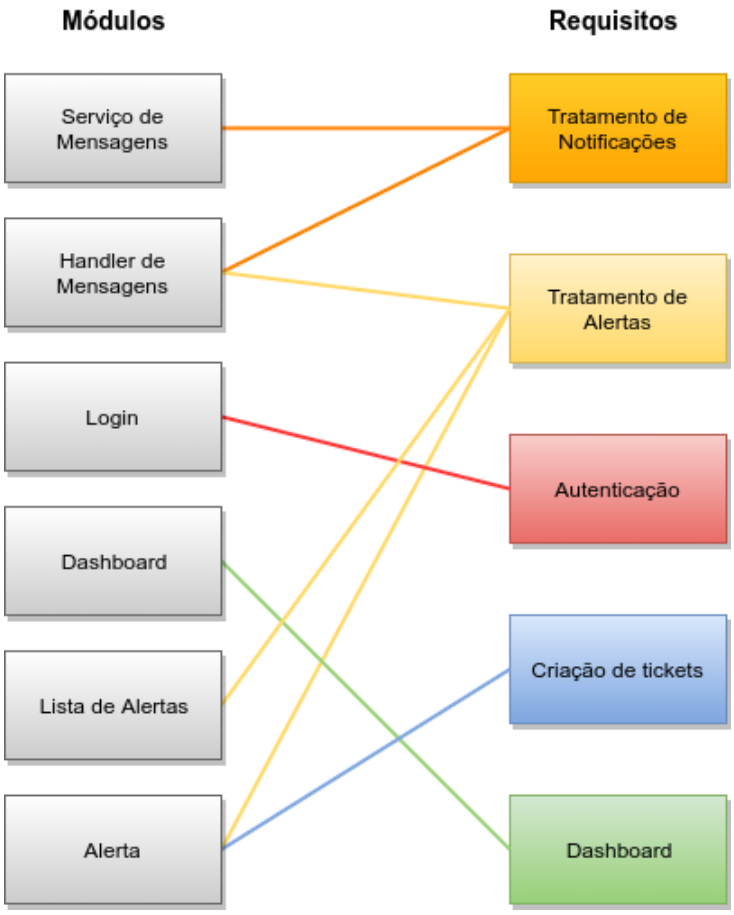


Figura A.1: Mapa de correspondência entre requisitos e módulos da aplicação móvel.

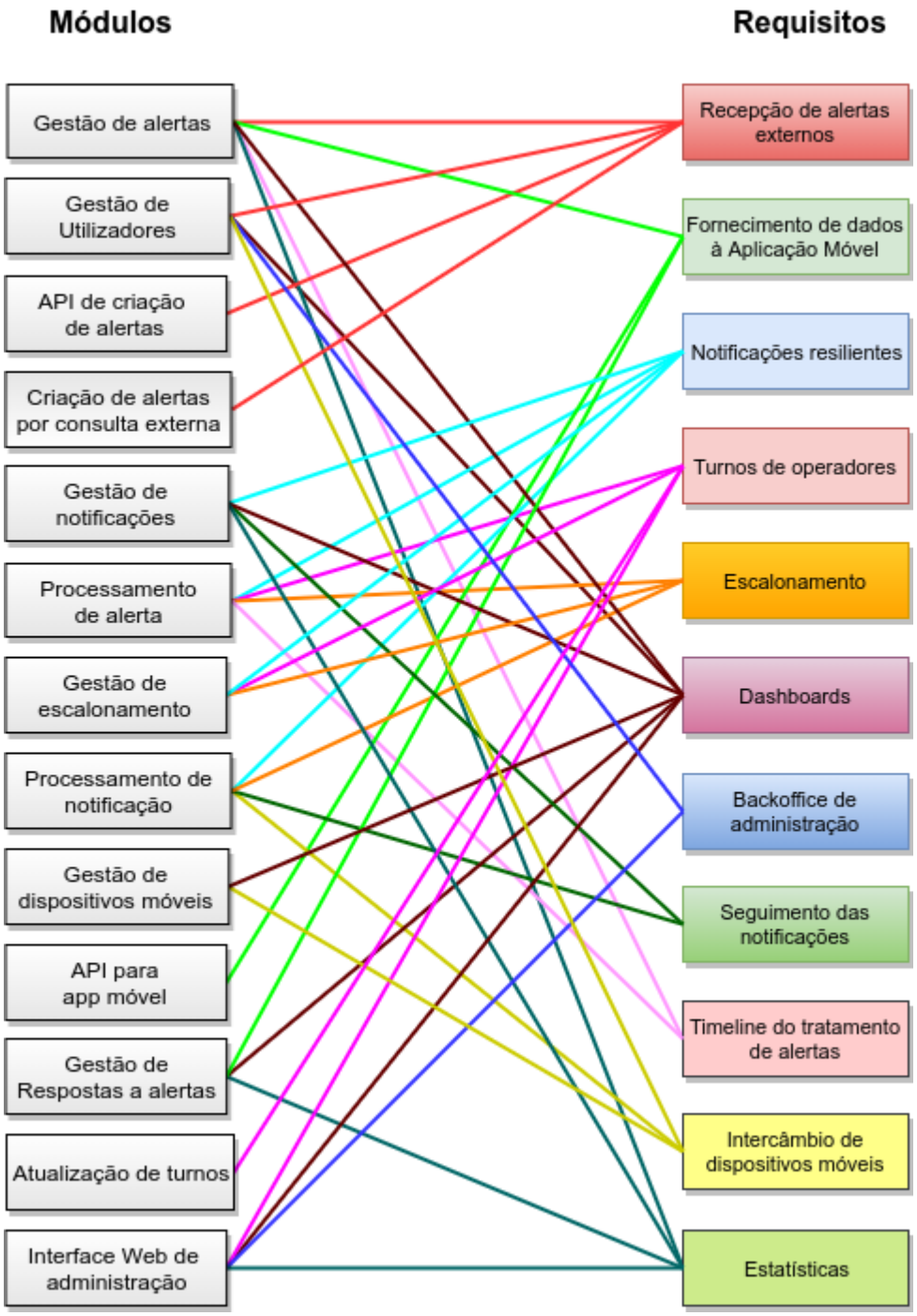


Figura A.2: Mapa de correspondência entre requisitos e módulos do servidor.

Apêndice B

Prime Path Coverage

Lista de Requisitos de Teste para Prime Path Coverage.

1. (1,3,4,6,1,3,4,7,9,10,12,11,1,3,4,7,9,10,12,14,15)
2. (1,3,4,6,1,3,4,7,9,10,12,13,1,3,4,7,9,10,12,14,15)
3. (1,2,1,3,4,7,9,10,12,11,1,3,4,7,9,10,12,14,15)
4. (1,2,1,3,4,7,9,10,12,13,1,3,4,7,9,10,12,14,15)
5. (1,3,4,7,9,10,12,13,1,2,1,3,4,7,9,10,12,14,15)
6. (1,3,4,7,9,10,12,11,1,2,1,3,4,7,9,10,12,14,15)
7. (1,3,4,7,9,10,11,1,3,4,7,9,10,12,13,1,3,4,7,9,10,12,14,15)
8. (1,3,4,7,9,10,11,1,3,4,7,9,10,12,11,1,3,4,7,9,10,12,14,15)
9. (1,3,4,7,9,10,12,13,1,3,4,7,9,10,11,1,3,4,7,9,10,12,14,15)
10. (1,3,4,7,9,10,12,13,1,3,4,7,9,10,12,13,1,3,4,7,9,10,12,14,15)
11. (1,3,4,7,9,10,12,13,1,3,4,7,9,10,12,11,1,3,4,7,9,10,12,14,15)
12. (1,3,4,7,9,10,12,13,1,3,4,6,1,3,4,7,9,10,12,14,15)
13. (1,3,4,7,9,10,12,11,1,3,4,6,1,3,4,7,9,10,12,14,15)
14. (1,3,4,6,1,3,4,7,9,10,11,1,3,4,7,9,10,12,14,15)
15. (1,2,1,3,4,7,9,10,11,1,3,4,7,9,10,12,14,15)
16. (1,3,4,7,9,10,11,1,2,1,3,4,7,9,10,12,14,15)
17. (1,3,4,7,9,10,11,1,3,4,7,9,10,11,1,3,4,7,9,10,12,14,15)
18. (1,3,4,7,9,10,11,1,3,4,6,1,3,4,7,9,10,12,14,15)
19. (1,3,4,6,1,3,4,6,1,3,4,7,9,10,12,14,15)
20. (1,3,4,5,4,6,1,3,4,7,9,10,12,14,15)
21. (1,2,1,3,4,6,1,3,4,7,9,10,12,14,15)
22. (1,2,1,2,1,3,4,7,9,10,12,14,15)
23. (1,3,4,7,9,10,12,13,1,3,4,5,4,6,1,3,4,7,9,10,12,14,15)
24. (1,3,4,7,9,10,12,11,1,3,4,5,4,6,1,3,4,7,9,10,12,14,15)
25. (1,3,4,7,9,10,11,1,3,4,5,4,6,1,3,4,7,9,10,12,14,15)
26. (1,3,4,6,1,3,4,5,4,6,1,3,4,7,9,10,12,14,15)

27. (1,2,1,3,4,5,4,6,1,3,4,7,9,10,12,14,15)
28. (1,3,4,5,4,5,4,6,1,3,4,7,9,10,12,14,15)

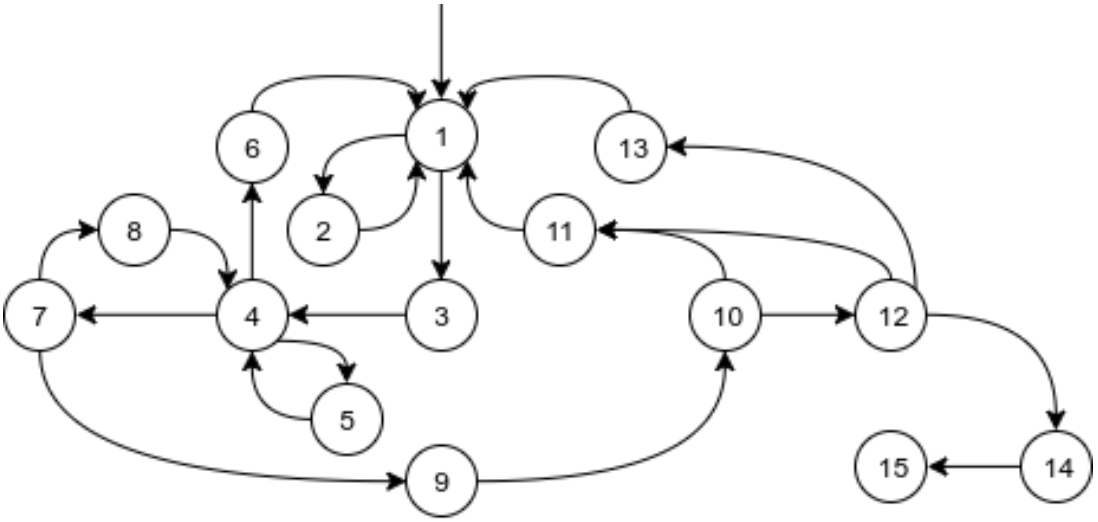


Figura B.1: Control Flow Graph do processamento de alertas e notificiações.

Apêndice C

Manual Cat|Mobile

O documento que se segue é o manual de utilização do sistema Cat|Mobile.

Cat|Mobile

Manual de utilização

Introdução

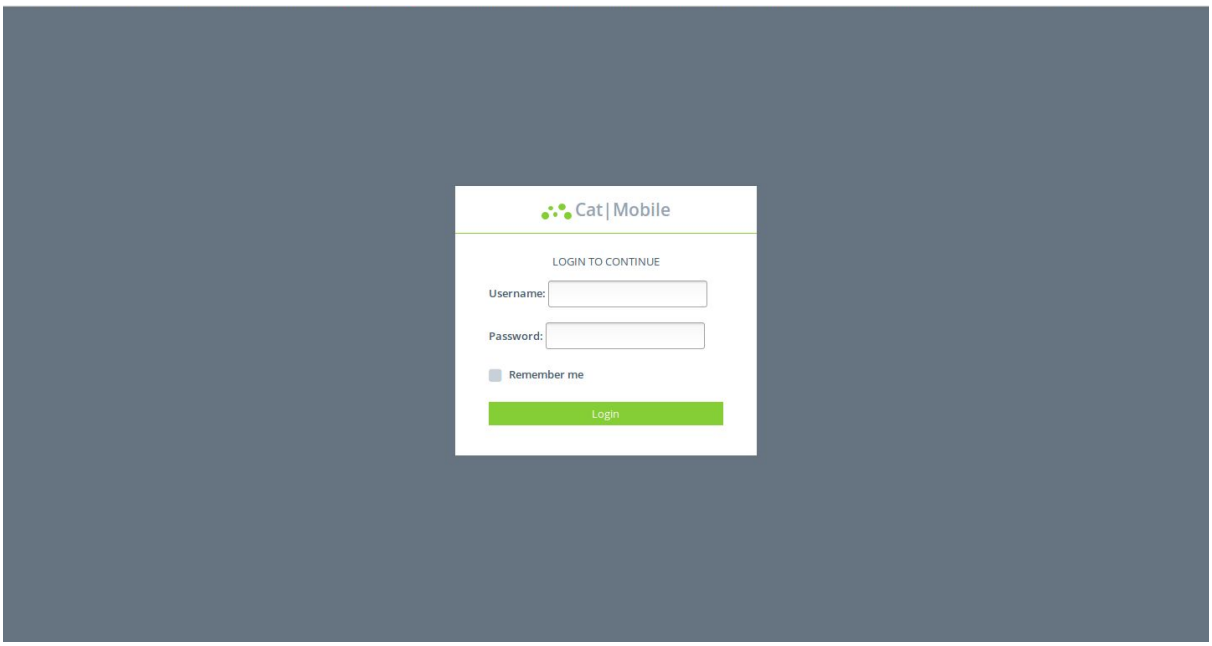
Cat|Mobile é um sistema composto por servidor e aplicação móvel, destinado a ser usado pela equipa de SOC da Dognaedis, para envio de alertas. A principal função da aplicação móvel é receber os alertas que são enviados pelo servidor. O servidor tem um calendário de turnos onde estão definidos os operadores On-Call responsáveis pelos alertas a cada momento. Os alertas são enviados a todos os operadores mas só o(s) operador(es) On-Call recebem notificações de alertas, que devem responder. As respostas aos alertas consistem, simplesmente, em clicar para aceitar (ou rejeitar) cada alerta, com mensagem de acompanhamento facultativa, através da aplicação móvel. A aplicação móvel é instalável em sistemas Android e requer ligação à Internet para receber as notificações de alertas. Os dispositivos podem ser usados por vários operadores alternadamente, mas para o operador começar a receber notificações tem de efetuar login na aplicação a seu turno, pelo menos uma vez, no dispositivo onde pretende receber as notificações. Para fazer login na aplicação é necessário que o operador tenha uma VPN, no dispositivo, estabelecida com ligação ao servidor. Para gerir os operadores e respetivos turnos On-Call, existe também uma aplicação de administração para a Web. Neste documento estão descritas as funcionalidades em geral do sistema e em particular do interface de administração do Cat|Mobile e da respetiva aplicação móvel.

Administração

A administração do Cat|Mobile consiste nas seguintes funcionalidades principais:

- Gestão de operadores
- Gestão de turnos
- Gestão de origens
- Gestão de clientes

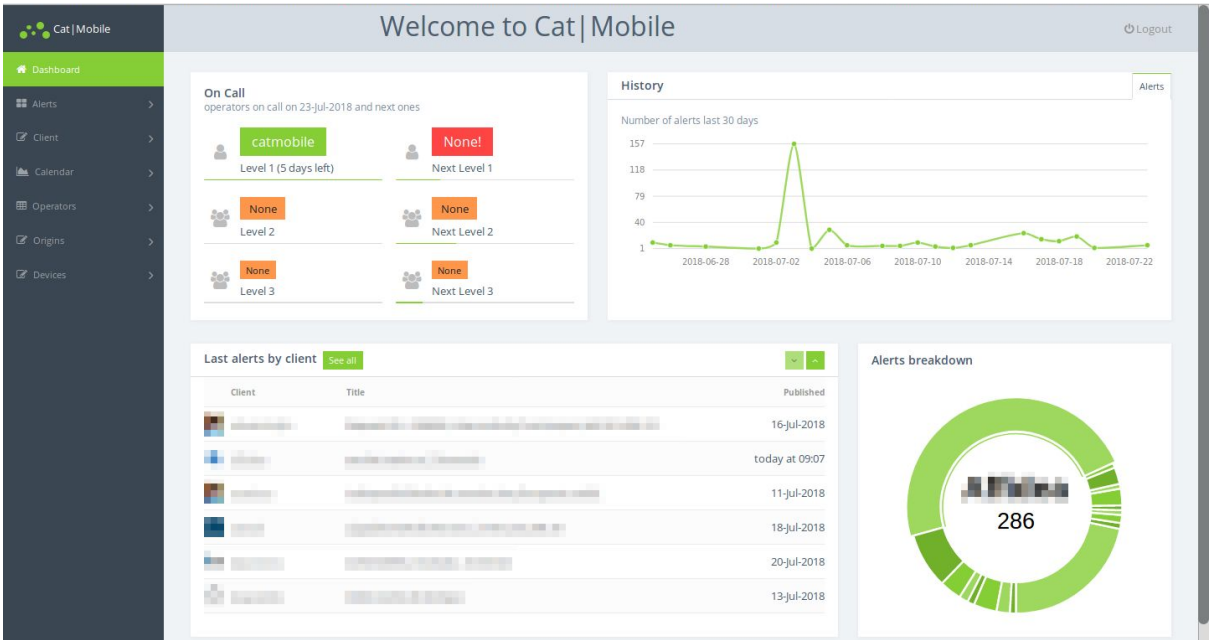
A área de admininstração está disponível em https://catmobile.dognaedis.com
Username: _____
Password: _____
Após o login efetuado, fica visível a área de administração em que a sua página inicial é constituída pela dashboard geral e um menu lateral de funcionalidades.
Nota: A sessão de utilizador será terminada automaticamente ao fim de 5 minutos de inatividade.



Dashboard

A dashboard da página principal encontra-se na raíz do URL e é a primeira opção do menu lateral de funcionalidades. Na dashboard estão distribuídos vários painéis com as seguintes informações:

- **On Call** - Apresenta o estado atual e o estado seguinte do escalonamento para os operadores de serviço. Cada um destes níveis é um atalho clicável para o calendário onde se encontra a funcionalidade de gestão de turnos. **Nota**: O escalonamento está dividido em 3 partições. O operador em On-Duty 1 é o primeiro a ser chamado a responder aos alertas. Os seguintes só são chamados se os anteriores não responderem dentro do prazo previsto.
- **History** - Apresenta o gráfico de volume de alertas diários ao longo do tempo.
- **Last alerts by client** - Apresenta a lista dos últimos alertas por cliente.
- **Alerts breakdown** - Apresenta um gráfico circular com a distribuição total dos alertas por cliente.



Menu lateral

O menu lateral de funcionalidades encontra-se disponível permanentemente durante a sessão de utilizador. O menu é constituído por opções distribuídas em árvore, tendo as seguintes secções principais:

- Alerts
- Clients
- Calendar
- Operators
- Origins
- Devices

Alerts

A secção Alerts do menu lateral, é constituída pela lista de alertas. A lista de alertas apresenta os alertas listados por ordem de data de criação. A lista é paginada e cada página é acessível através do navegador de páginas disponível na base da lista. Cada linha representa um alerta e tem a seguinte estrutura:

- Checkbox de selecção de alerta para ação;
- Logotipo do cliente da empresa que gerou o alerta;
- Título do alerta;
- Origem (ferramenta que emitiu o alerta);
- Designação do cliente que gerou o alerta;
- Nível de severidade do alerta;
- Operador designado como owner do alerta;
- Data de criação do alerta na sua origem;
- Data de criação do alerta no Cat|Mobile;
- Data da última atualização ou alteração de estado do alerta no Cat|Mobile.

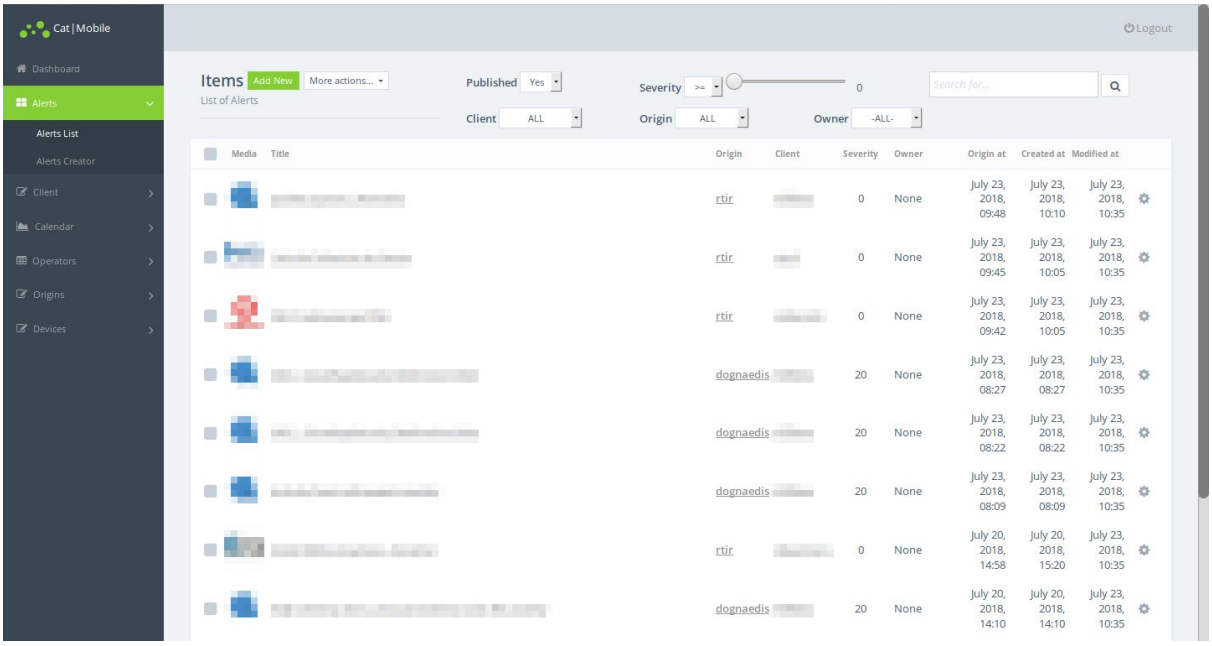
No topo da lista estão disponíveis ações a efetuar sobre os alertas selecionados e opções que permitem filtrar a lista. As ações disponíveis são:

- Unpublish - Para despublicar o alerta da lista (mas manter em arquivo);
- Publish - Para trazer de volta à lista principal, um alerta arquivado.

As opções de filtro disponíveis são:

- Published - Yes/No - Para filtrar por alertas publicados ou não;
- Client - Para filtrar por cliente;
- Severity - Para filtrar por nível de severidade;
- Origin - Para filtrar por ferramenta de origem;
- Owner - Para filtrar por operador designado como owner de alertas;
- Search - Para pesquisa livre de palavra;

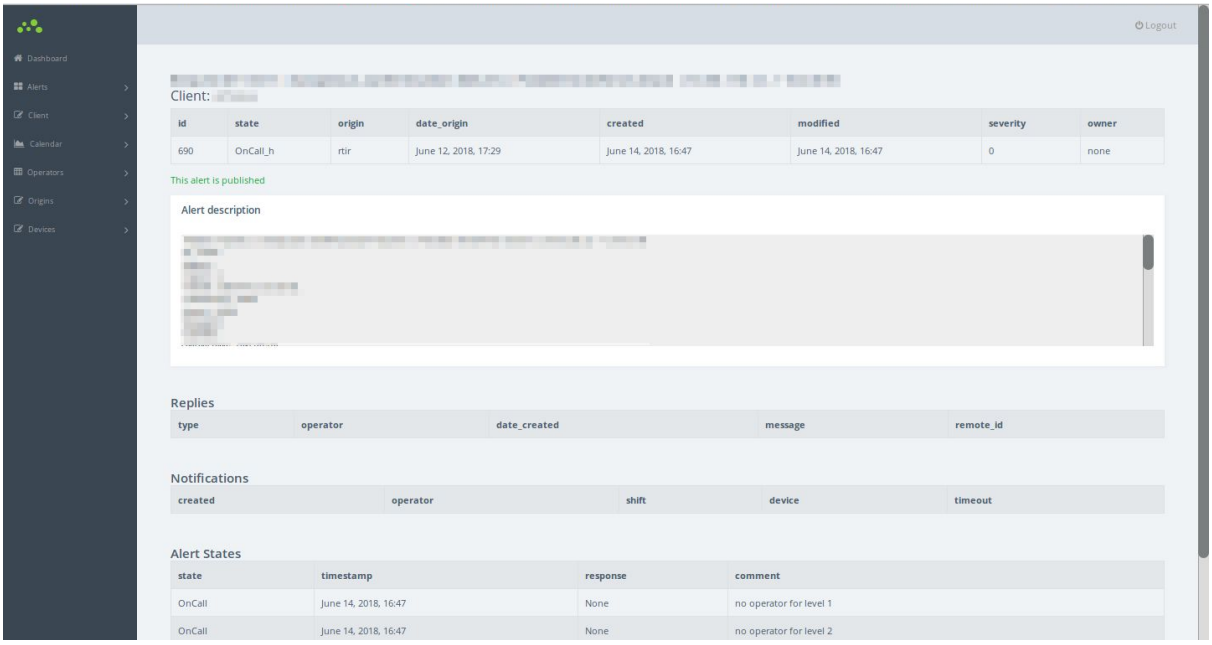
Para executar o filtro, o utilizador deve clicar na “lupa” representada pelo ícone na opção de Search.



Ao clicar no título de um alerta, é apresentada uma página sobre esse alerta. Nessa página são visíveis todos os dados do alerta e um histórico de notificações enviadas a operadores para o tratamento do alerta. Para além da lista de notificações, existe também a lista de respostas dadas por operadores ao alerta.

- Replies - respostas dadas ao alerta. As respostas podem ser as seguintes:
    - Received - esta é uma resposta automática, enviada pela aplicação móvel quando o operador está autenticado e visualiza o conteúdo do alerta.
    - Accepted - é quando o operador indica que vai dar início ao tratamento do alerta.
    - Refused - quando o operador indica que não pode dar início ao tratamento do alerta.
    - Conclusion - quando o operador dá por concluído o tratamento do alerta.
- Notifications - é a lista de notificações enviadas pelo servidor a operadores, seguindo o escalonamento previsto.
- Alert states - é a lista de alterações de estado por onde cada alerta passa.
    - OnCall - O primeiro estado de um alerta é OnCall, onde o sistema decide qual o operador a quem enviar notificação.

- Notification - Se encontrar operador em OnCall, o estado seguinte é Notification que é onde é enviada a notificação. No estado de notificação, o sistema aguarda por uma resposta do operador durante o tempo apresentado na coluna 'timeout'. Caso o operador não responda com Accepted dentro do tempo limite, o sistema volta ao estado OnCall para ir buscar o operador seguinte na lista de escalonamento.
- OnGoing - é o estado onde o alerta se encontra em tratamento.
- Conclusion - é o estado em que o tratamento foi dado por concluído.



Clients

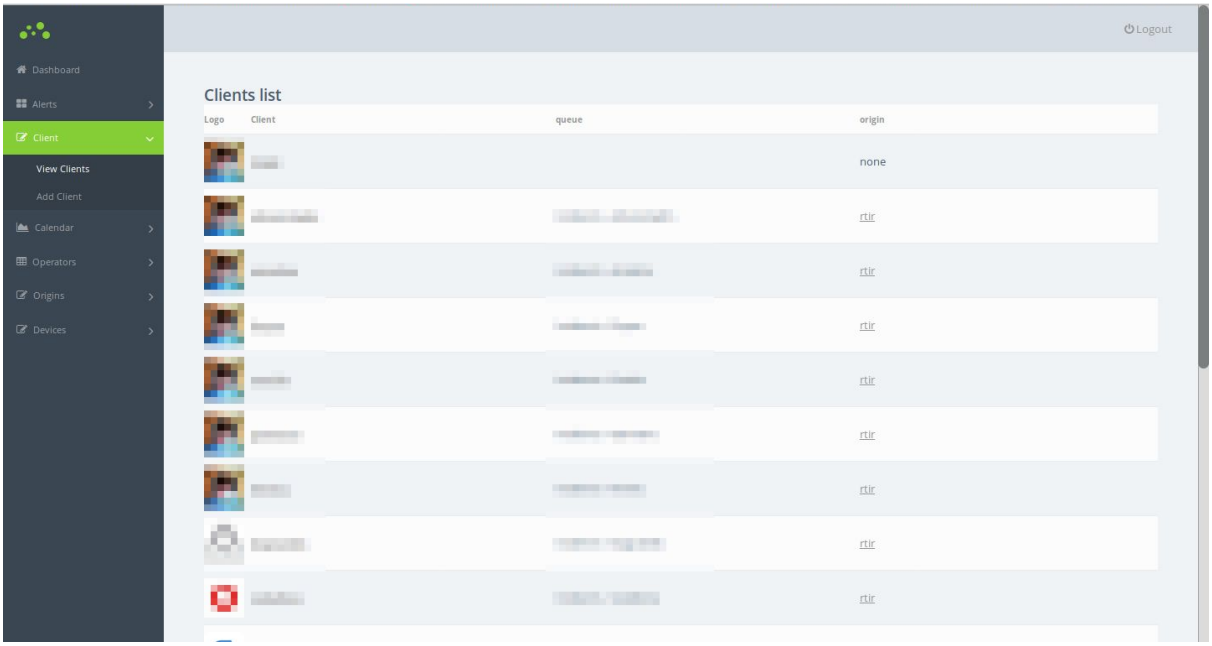
A secção de Clients do menu lateral é dedicada aos clientes da empresa onde são gerados os alertas do Cat|Mobile. Nesta seção existem duas opções:

- View clients - Apresenta a lista de clientes
- Add client - Adiciona cliente

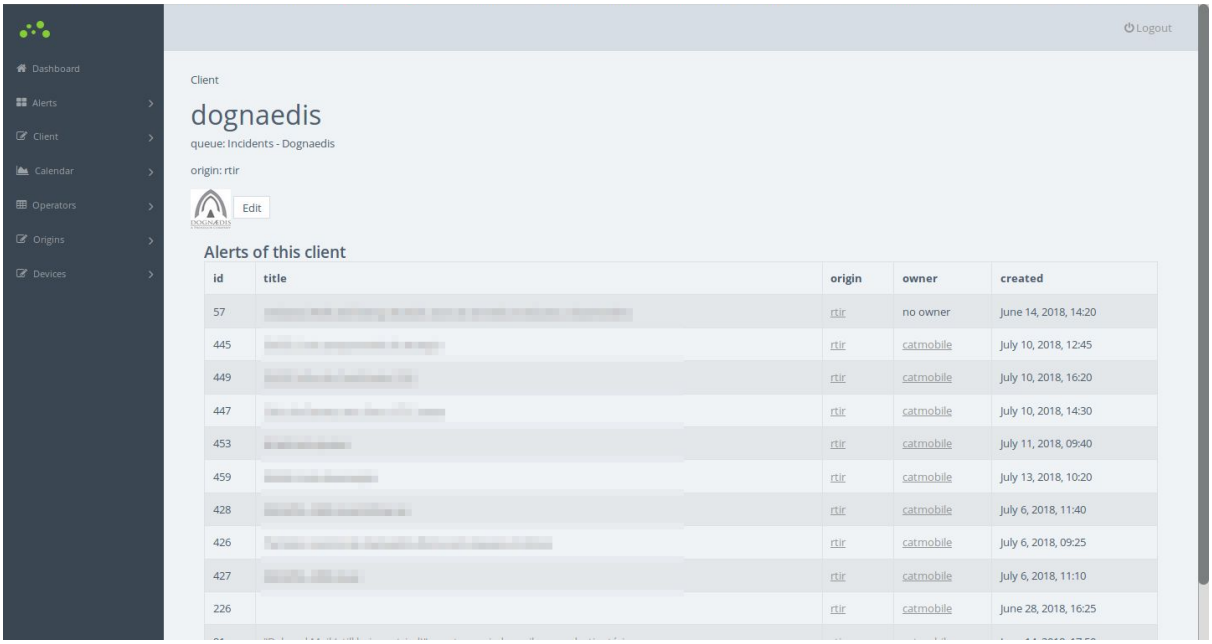
Nota: Os clientes novos podem ser adicionados manualmente, mas também são adicionados automaticamente sempre que surge um alerta de um cliente desconhecido, ou uma nova queue no sistema de tickets. No entanto, na criação automática de clientes, o cliente fica com um logotipo genérico, sendo necessária a intervenção do administrador para definir um logotipo específico.

A opção de View clients apresenta a lista de clientes estruturada da seguinte forma:

- Logotipo do cliente (ou logotipo genérico caso o cliente ainda não tenha o logotipo definido pelo administrador);
- Queue, que indica a designação usada pelo sistema de tickets para identificar o cliente;
- Origin, que indica qual o sistema de tickets associado ao cliente.



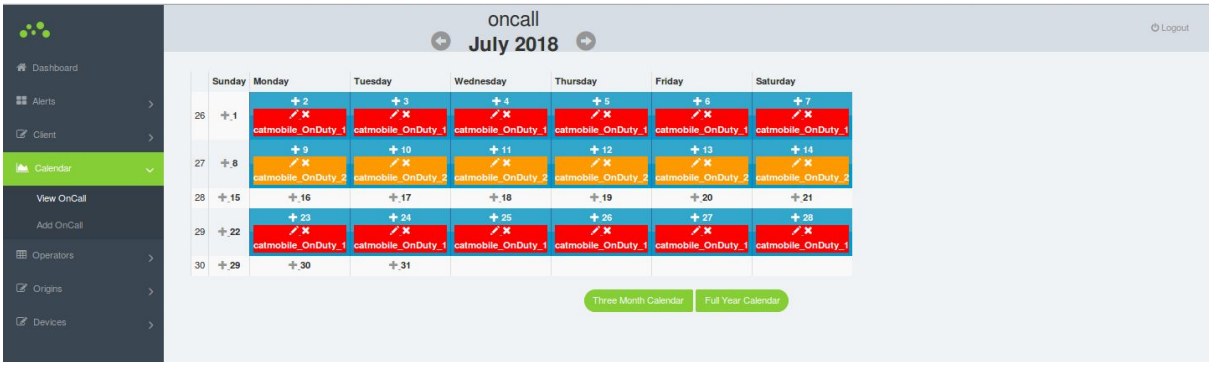
Para visualizar a mini-dashboard dedicada ao cliente, deve-se clicar na designação do cliente na lista de clientes ou no respetivo logotipo.
A mini-dashboard associada a cada cliente apresenta os dados do cliente usados pelo Cat|Mobile, com opção para editar e uma lista de alertas gerados pelo cliente.
A opção editar permite alterar os dados do cliente, nomeadamente o nome, a designação da queue no sistema de tickets, o ficheiro de imagem de logotipo e o sistema de tickets associado ao cliente.



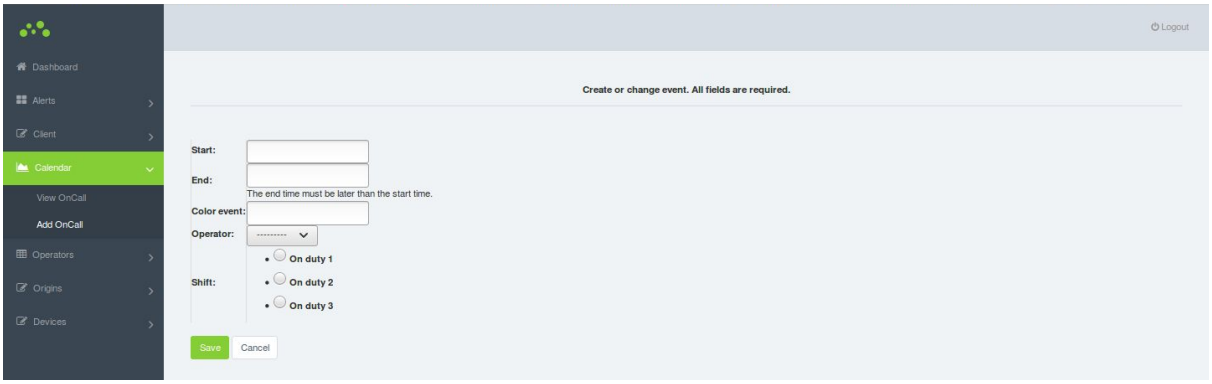
Nota: Como os alertas são identificados pelo nome do cliente, a sua alteração pode significar a desconexão de alertas com cliente, e posteriormente, levar à criação automática de um novo cliente com o nome antigo, pelo simples motivo de ser criado um novo cliente sempre que é recebido um alerta associado a cliente com nome desconhecido.
A opção Add client permite adicionar um cliente de forma manual, especificando o seu nome, logotipo e designação da queue usada pelo sistema de tickets.

Calendar

O Calendar serve para fazer a gestão de turnos On-Call. Esta seção tem duas opções, View On-Call para visualizar o calendário onde também é possível editar, e Add On-Call para adicionar turnos de operadores On-Call no calendário. Esta opção é redundante pois também está implementada na visualização do calendário.
Na opção View On-Call é apresentado o calendário. O calendário tem, por omissão, o modo mensal pré-definido em que é visível o mês atual. É possível alterar a visualização do calendário para trimestral ou anual clicando nas opções visíveis na base do calendário. Para alterar o mês visível, existem setas no topo do calendário que permitem “navegar” pelos meses.

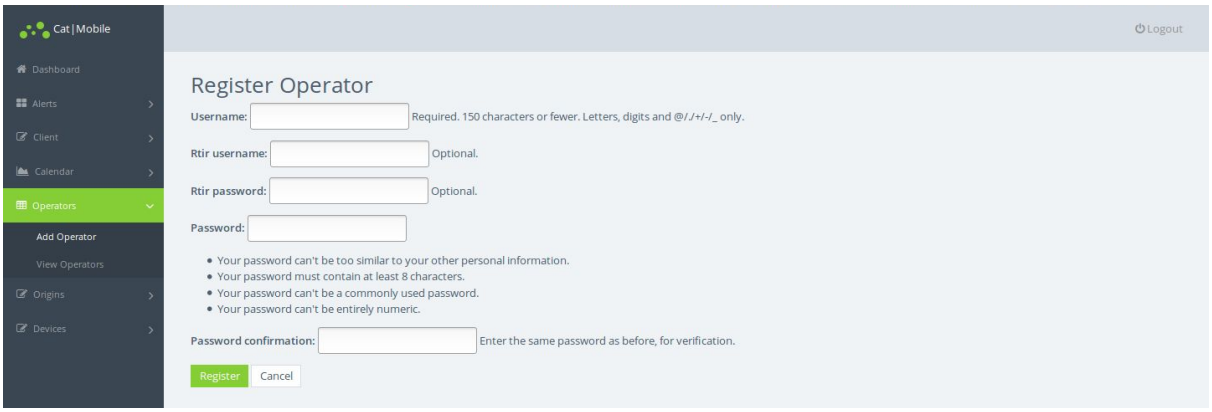


Na visualização mensal do calendário, o calendário é apresentado como um tradicional calendário, com as semanas por linhas, a iniciar no Domingo e terminar no Sábado. Cada dia é representado por uma célula onde são apresentados os turnos para esse dia. Caso um dia não tenha ainda os turnos todos preenchidos é possível adicionar um turno ao dia em questão diretamente no calendário. Ao clicar no sinal “+” de determinado dia, aparece um formulário para adicionar o turno, pré-preenchido com a data inicial do dia escolhido e data final de 5 dias posteriores. Estas datas são alteráveis durante a edição do formulário. Para terminar o registo de um turno, resta escolher o operador da lista de operadores disponíveis e o shift de “on-duty” (1,2 ou 3). Ao clicar em “Save” o sistema irá validar o registo fazendo verificação de não sobreposição de turnos. A opção de Add On-Call permite adicionar um turno sem ser pelo calendário. O formulário é o mesmo, apenas não aparece pré-preenchido como os formulários iniciados pelo calendário.



Operators

Operadores são os utilizadores da aplicação móvel Cat|Mobile. Na secção Operators estão as opções Add operator e View operators.
Add operator é a opção para registar operadores na base de dados do sistema. O registo de operadores consiste na criação da conta de operador em que são definidos o username e password para poder fazer login na aplicação móvel. Nos campos extra Rtir-username e Rtir-password são definidas as credenciais do sistema de tickets para situações em que a resposta a alertas inclua a criação de ticket no RTir.



View operators apenas apresenta a lista de operadores em que o nome de operador é um atalho clicável para a página do operador. A página do operador consiste num mini-dashboard onde são visíveis os dados do operador e um gráfico com o volume de alertas diário que lhe foram destinados ao longo do tempo. Nessa mesma página, são ainda visíveis duas listas, a lista de alertas destinados ao operador e a lista de respostas dadas aos alertas. As respostas a alertas consiste nas alterações de estado dos alertas. Cada alerta passa por vários estados desde que surge no Cat|Mobile até ser dado como resolvido. Os estados que um alerta pode tomar são os seguintes:

- Received - Significa que o texto do alerta foi visualizado na aplicação, pelo operador.
- Take - Significa que o operador deu início ao tratamento do alerta.
- Denied - Significa que o operador não está disponível para o tratamento do alerta.
- Conclusion - Significa que o operador deu por terminado o tratamento do alerta.

Origins

As origens são contas de acesso ao Cat|Mobile com as quais as ferramentas externas podem criar alertas através da API. As origens são também as origens dos alertas criados no sistema. Os alertas são obtidos das origens de duas formas. Podem ser enviados por iniciativa da ferramenta externa ou podem ser obtidos por consulta ao sistema de tickets. As credenciais RTir são para o caso das origens que necessitam de consulta externa. Para o caso de ferramentas externas, as credenciais são necessárias para usar a API do Cat|Mobile para criação de alertas através de Post.
Para a origem criar alertas através da API são necessários dois Posts, um primeiro para obter o token, onde é usado o username e password e o Post seguinte onde é usado o token para enviar um alerta em formato Json. De seguida é apresentado um exemplo desses dois Posts em Python:

Para obter o token:

```
import requests
url = "http://catmobile.dognaedis.com/api/get-token/"
payload = "{\n  \"username\": \"________\",\n  \"password\":
\"________\"\n}"
headers = {
      'Content-Type': "application/json"
      }
response = requests.request("POST", url, data=payload,
headers=headers)
print(response.text)
```

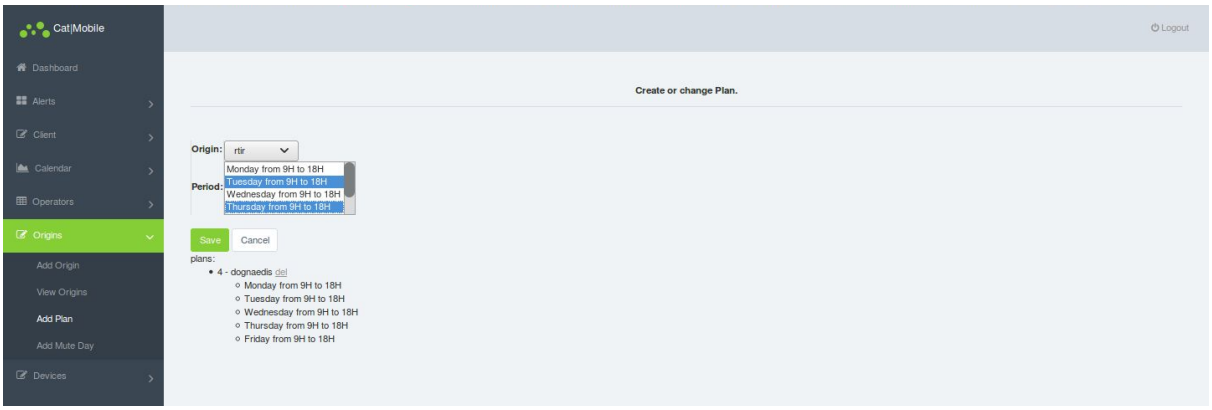
Para criar o alerta, com o token obtido:

```
import requests
url = "http://catmobile.dognaedis.com/api/"
payload = "{\n \"title\": \"teste\",\n \"description\": \"staging
tests\",\n \"severity\":20,\n \"client\":\"stage\"\n}"
headers = {
'Authorization': "Token
_____________________________________",
'Content-Type': "application/json"
}
response = requests.request("POST", url, data=payload,
headers=headers)
print(response.text)
```

As origens podem ter associado um Plan. Um plano é um conjunto de Mute days, ou seja de períodos diários onde se pretende que a origem não faça envio de notificações quando são criados alertas. Para criar um plano, é necessário ter criado previamente os Mutes days.

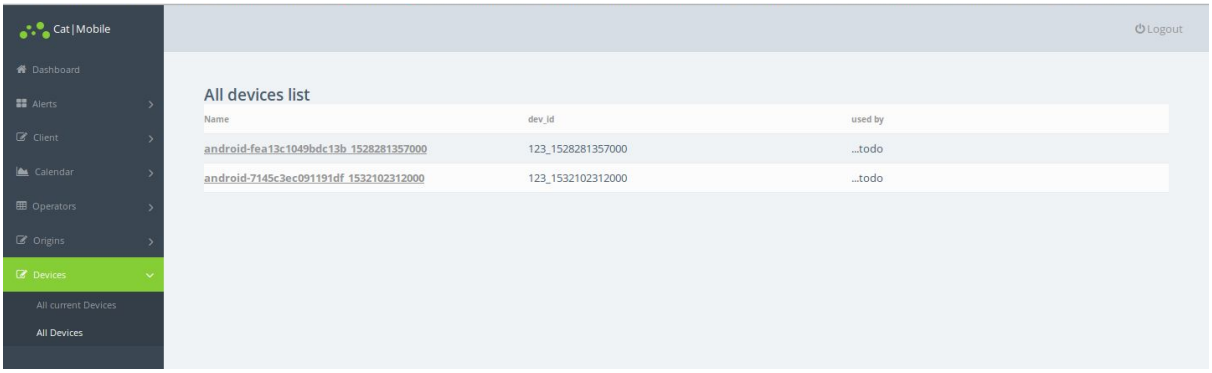
Os Mute days são criados na opção Add Mute day. Começa-se por escolher o dia da semana e depois a hora de início e a hora de fim. Podem criar-se diversos Mute days com várias combinações de horários se assim se desejar.

Esses Mute days vão estar disponíveis em Add Plan para criar um plano associado a uma origem. O processo é simples, o administrador escolhe a origem de entre as disponíveis e depois, o(s) período(s) que se pretendem aplicar à origem. Podem selecionar-se vários em simultâneo com o auxílio da tecla shift.

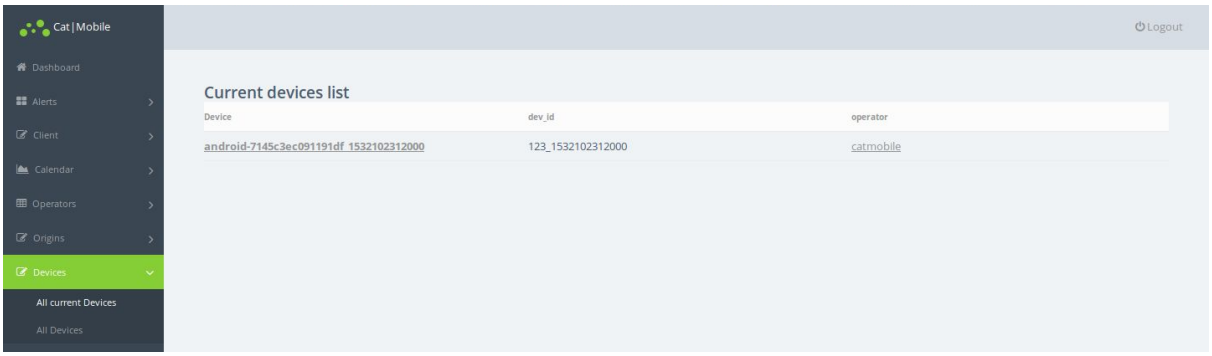


Devices

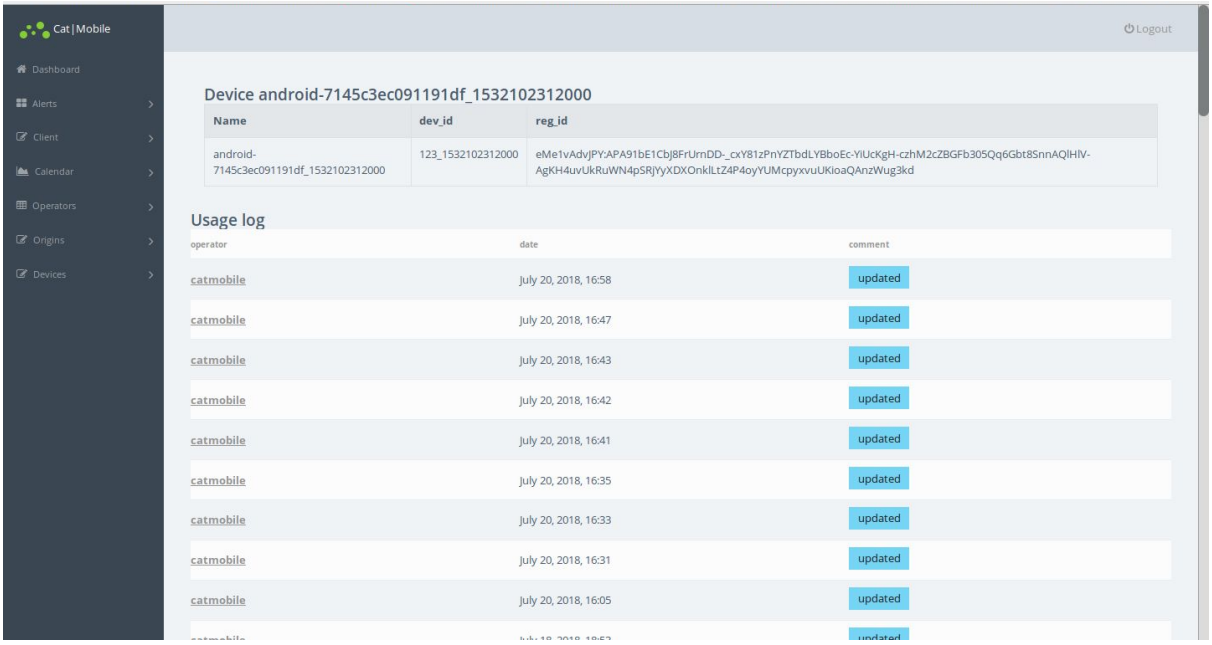
As devices referidas no menu lateral são os dispositivos moveis usados pelos operadores. Apesar do Cat|Mobile permitir intercâmbio de dispositivos móveis entre operadores, o administrador não é chamado a gerir essas trocas, pois o processo é transparente . Para que o sistema identifique o dispositivo alocado ao operador, basta que o operador faça login na aplicação móvel com o dispositivo que está a usar, assim que faz uma troca de dispositivo. Esta seção existe no apenas no menu do Cat|Mobile para fazer o tracking dos dispositivos e verificar que dispositivo está a usar cada operador, que operadores já usaram o dispositivo e as data de troca dos dispositivos .
A opção All Devices lista todas os dispositivos reconhecidos pelo sistema, ou seja, todos os dispositivos já utilizados, pelo menos uma vez.



A opção All Current Devices lista os dispositivos que estão atualmente alocados a operadores.



Ao clicar num dispositivo da lista, aparece o mini-dashboard do dispositivo. Nessa página são visíveis os dados que identificam o dispositivo no sistema e a lista de uso do dispositivo, nomeadamente todos os logins efetuados no Cat|Mobile e trocas de dispositivo sempre identificados por operador e com a data do evento.



Operações

Introdução

O objetivo principal da aplicação móvel Cat|Mobile é de proporcionar aos operadores da equipa SOC um meio de resposta rápida aos alertas, para quando estão oncall mesmo quando estão em mobilidade. Em geral, Oncall consiste em determinar qual ou quais os operadores responsáveis pela resposta aos alertas, mesmo fora da empresa. A cada momento, deve existir um operador responsável pela resposta a alertas. Caso esse operador não responda ao alerta dentro de um determinado prazo, outro operador será chamado a responder ao alerta. O administrador Cat|Mobile é quem faz a definição do escalonamento de operadores responsáveis. Todos os operadores do grupo OnCall recebem notificações de alertas. Quando as notificações são enviada pelo servidor, embora dirigidas a um operador, tecnicamente são enviadas a um dispositivo móvel. Para o sistema

poder identificar o dispositivo usado atualmente pelo operador, o operador tem de fazer login na aplicação, pelo menos uma vez. Os operadores podem trocar de dispositivos móveis sempre que quiserem e podem mesmo trocar de dispositivo entre si. Não é necessário ter a aplicação aberta para receber as notificações, mas quando uma notificação é aberta, a aplicação é iniciada e o operador tem de fazer login para aceder à informação do alerta. Devido ao facto de estarmos a usar uma plataforma de terceiros para transmitir a notificação, a notificação é suposta conter informação reduzida e inútil para terceiros. A informação transmitida na notificação é apenas o ID do alerta no Cat|Mobile e um número que corresponde à severidade do alerta. A informação do alerta propriamente dito, apenas se encontra disponível ao operador depois de autenticado na aplicação.

Conceitos

- **Grupo Oncall** - é um conjunto de operadores responsáveis por responder aos alertas dentro de um período de tempo definido pelo administrador.
- **Escalonamento** - Dentro do grupo Oncall, apenas um operador é responsável por responder a dado alerta a cada momento. Caso esse operador não responda ao alerta dentro de um prazo estabelecido, outro operador será designado responsável, seguindo o escalonamento pré definido pelo administrador.
- **Notificações** - A informação enviada nas notificações consiste no ID e severidade do alerta, ambos os dados numéricos. Para ver o alerta referido pela notificação, é necessário fazer login na aplicação. As notificações são enviadas através de Google Play services e podem ser de dois tipos:
    - **Intensas** - Aviso idêntico ao de uma chamada telefónica, enviadas apenas ao operador responsável.
    - **Calmas** - Aviso idêntico ao de um e-mail recebido, enviada para todos os operadores do grupo.
- **Google Play** - Este componente tem de estar a funcionar no dispositivo móvel. No entanto, não é necessário ter uma conta Google configurada no dispositivo.
- **Auto boot** - Se existir algum sistema de restrições de auto-boot de aplicações a correr no dispositivo, é necessário desbloquear ou dar-lhe permissão para a aplicação Cat|Mobile poder iniciar-se automaticamente.
- **Dispositivo móvel** - Os operadores podem trocar e fazer intercâmbio de dispositivos móveis, mas para o Cat|Mobile contactar o operador através do dispositivo, o operador tem de fazer login na aplicação pelo menos uma vez.
- **Login** - Para fazer login na aplicação, são necessárias credenciais, fornecidas pelo administrador.
- **VPN** - Para a aplicação móvel comunicar com o servidor, é necessário ter a VPN estabelecida. O cliente de VPN a usar no Android tem de ser obrigatoriamente o OpenVPN for Android devido ao formato das configurações de OpenVPN fornecidas pelo departamento de IT na empresa. O OpenVPN for Android tem de estar instalado no dispositivo porque a aplicação Cat|Mobile está integrada com este cliente de modo a ligar e desligar a VPN sempre que necessário. A ligação configurada deve ter o nome de “dognaedis” (ou outro a designar, no futuro).

Manual da aplicação móvel

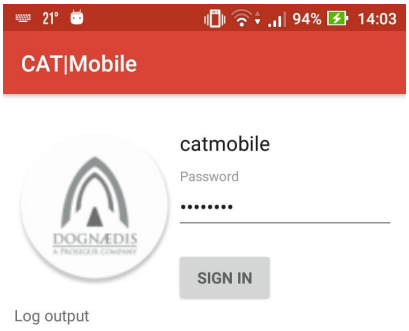
A aplicação móvel é constituído por um interface de utilizador e por um serviço que recebe as notificações.

Interface de utilizador

De seguida são apresentadas e descritas as funcionalidades disponíveis através do interface de utilizador.

Login

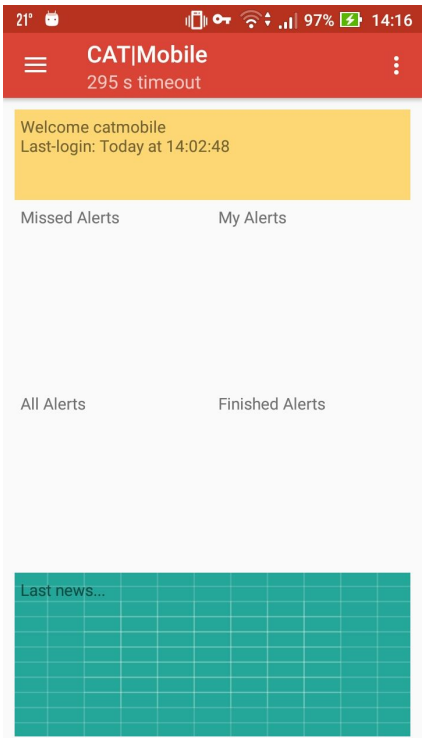
Para aceder à aplicação onde estão os dados dos alertas e poder responder aos alertas, é necessário preencher as credenciais de operador no formulário de Login. Preenchidas as credenciais, ao clicar em SIGN IN, a aplicação tenta estabelecer a VPN através da app OpenVPN for Android. Este processo demora alguns segundos. O estado da ligação da VPN pode ser acompanhado através da app OpenVPN for Android que é iniciada em background. Caso o Login não funcione à primeira vez, verificar se é devido a falta de VPN.



Dashboard

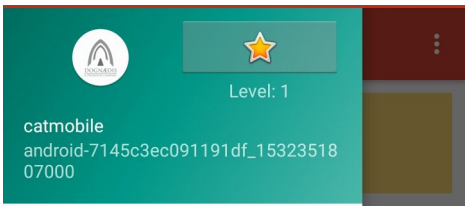
Na página inicial da aplicação é constituída pelos seguintes elementos:

- Tempo de timeout em segundos para o logout automático;
- Last Login: data e hora do último login efetuado pelo operador;
- Missed Alerts - resumo dos últimos alertas perdidos (TODO);
- My Alerts - resumo dos últimos alertas do operador (TODO);
- All Alerts - resumo de todos os últimos alertas (TODO);
- Finished Alerts - resumo dos últimos alertas terminados (TODO);
- Last news - Notícias diversas enviadas pelo servidor (TODO);
- Menu lateral - Menu que aparece ao clicar no ícone com 3 traços horizontais. 

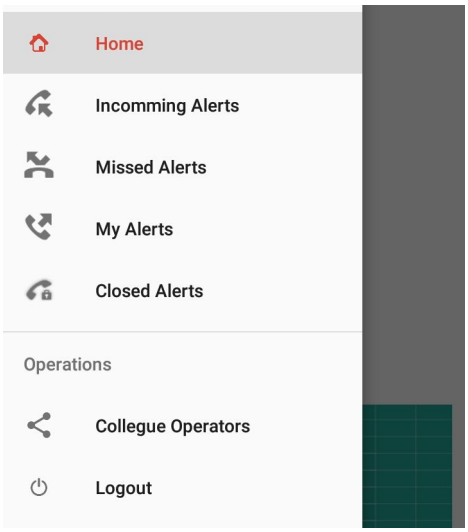


Menu Lateral

O menu lateral é constituído pelos seguintes elementos:

- Cabeçalho - Mostra informação sobre o operador.
    - Nome do operador;
    - Nome do dispositivo;
    - OnDuty: N (1-3) - clicar apresenta as datas do turno do operador;
    - 

- Opções de menu;
    - Home - Dashboard
    - Incoming Alerts - Lista de todos os alertas
    - Missed Alertas - Lista de alertas falhados
    - My Alertas - Lista de alertas do operador
    - Closed Alerts - Lista de alertas concluídos
    - Collegue Operators - Lista de operadores em shift
    - Logout - Para terminar a sessão de utilizador na aplicação.



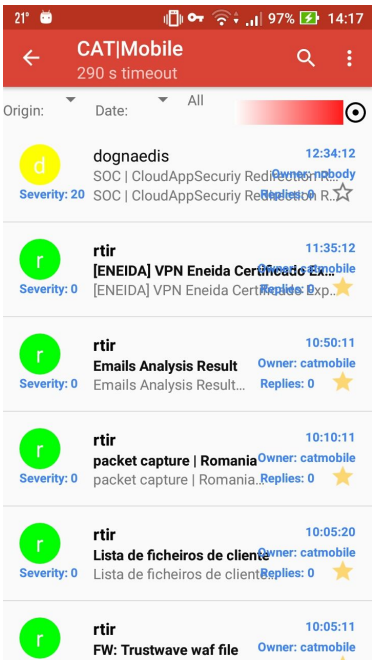
Logout

O logout é uma opção disponível no menu lateral da aplicação. Contudo se existir 5 minutos de inatividade, a aplicação faz logout automático. De uma forma ou de outra, o logout desliga a também a VPN.

Incoming Alerts

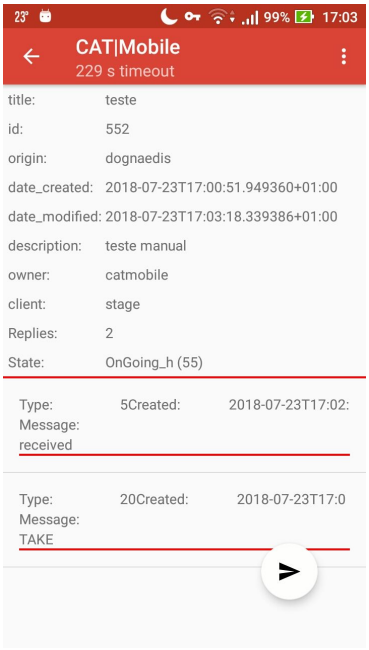
A lista de alertas é composta pelos seguintes elementos:

- Filtro
  - Origin - filtrar por origem de alertas;
  - Date - filtrar por alertas do próprio dia, de ontem, de há 7 dias e de há 1 mês;
  - Severity - filtrar por severidade;
- Alertas
  - Cor conforme a severidade (verde = 0, amarelo = 20, vermelho = 40)
  - Origem - origem do alerta
  - Título - do alerta
  - Hora de criação do alerta
  - Owner - operador owner, se existir
  - Replies - número de respostas ao alerta



Alerta

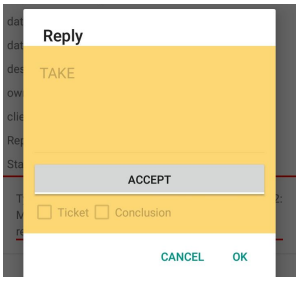
A visualização de um alerta consiste na apresentação dos dados do alerta. Para além dos dados do alerta, são também visíveis as eventuais respostas que este já tenha tido e o botão para efetuar respostas.



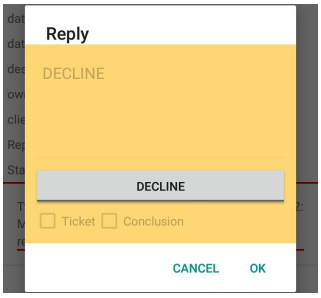
Os alertas são acedidos através da lista de alertas ou através das notificações.

- **Respostas** - Só pode responder a um alerta, o operador designado como responsável por esse alerta, chamado de “owner”.

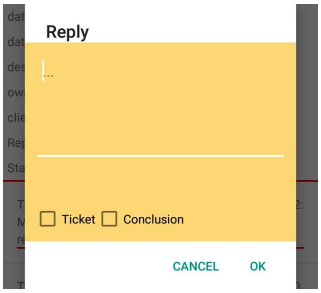
- **Aceitar** - Indica ao servidor, o início de tratamento do alerta pelo operador responsável.



- **Recusar** - Indica ao servidor que tem usar o escalonamento para designar outro operador como responsável.

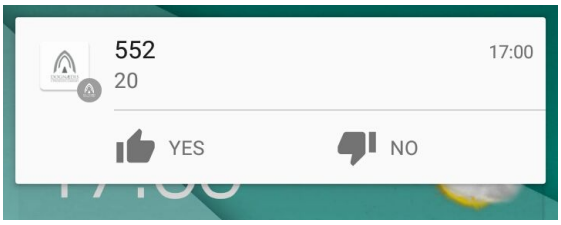


- **Reply** - Uma vez aceite, durante a resposta a um alerta, o operador tem a opção de criar no RTir um ticket com base na resposta. Para terminar o tratamento do alerta, basta selecionar a opção Conclusion durante a resposta. A primeira resposta que um alerta pode ter é sempre de aceitar ou recusar o alerta.



Notificações

Ao clicar numa notificação, a aplicação tenta abrir o alerta referido na notificação. Caso o operador tenha o login efetuado, o alerta é apresentado de imediato. Caso contrário, surge o formulário e só depois do login validado é que o alerta é apresentado.



Na barra de notificações, a notificação de alerta tem a seguinte configuração:

Em ambas as apresentações da notificação, o utilizador tem opção para aceitar (yes) ou rejeitar (no). Estas opções funcionam como atalhos para a resposta pretendida. O alerta continua a necessitar de sessão iniciada na aplicação para poder ser visto ou dar resposta.

Apêndice D

Proposta de Estágio

De seguida é apresentado o documento da proposta de estágio entregue pela empresa.

PROPOSTA DE ESTÁGIO

Título da Proposta: CAT | Mobile

SUMÁRIO

Este estágio propõe o desenvolvimento de uma plataforma mobile para recepção de alarmística, capaz de agregar dados de várias fontes, permitindo ao operacional de segurança on-Call a sua visualização sobre uma aplicação mobile.
Para tal, esta deverá ser capaz de agregar em backoffice dados de alarmística de diferentes fontes, já existentes na empresa e permitir a fácil adição de outras novas que surjam. Deverá também, apresentar os dados e informação de investigações ao operador de forma simples e intuitiva. Tal irá permitir ao operador uma melhor e mais eficaz capacidade de acção e monitorização quando on-Call.
O sistema deverá também obedecer a certas normas de segurança, de modo a minimizar a quantidade de informação disponível no dispositivo. Permitir a filtragem de alertas por severidade e a criação de tickets na plataforma de ticketing.

ÂMBITO

A Dognaedis é uma empresa com foco em questões de segurança de informação e cibersegurança, fornecendo aos seus clientes, entre outros serviços, assentes sobre outras áreas de negócio, a monitorização continua e resposta a incidentes que possam incidir sobre as suas redes de comunicação e serviços de informação.
Para tal, a Dognaedis faz-se valer de um conjunto de plataformas de monitorização e de consolas agregadoras de registos escrito por terminais e equipamentos de rede, cabendo actualmente ao analista de segurança, correlacionar informação patente nestes registos e plataformas, para detectar possíveis incidentes de segurança, que como tal devam ser tratados e mitigados.
Como a empresa presta serviços de 24x7 em vários países por vezes é necessário que mesmo que um operador não esteja presente na empresa, este esteja disponível (on-Call).
Com a implementação da plataforma mobile aqui proposta, o trabalho de análise enquanto on-Call seria tornado mais eficaz, levando a uma melhoria não só na capacidade de análise do operador de segurança a eventos de intrusão, mas também no seu tempo de reacção e na optimização de recursos e gastos neste modelo pela empresa.

OBJECTIVOS

O estágio proposto, pretende optimizar a forma de análise dos operadores da organização. Ao invés de receber um SMS com uma pequena descrição do problema, de seguida ter de se ligar à infraestrutura da empresa e só depois às máquinas e serviços, passará a poder observar directamente alarmes dos vários sistemas da organização no telemóvel da equipa de operações.
O presente projecto pretende atingir os seguintes objectivos genéricos:

- Implementar um backoffice de agregação de alarmes provenientes de várias fontes de informação;

- Construir uma aplicação mobile que permita ao operador analisar os alarmes recebidos de forma intuitiva;
- Representar os tipos de alarmes com informação visual pertinente;
- Filtrar alarmes recebidos por severidade, permitindo apenas a análise dos alarmes mais importantes.
- A partir do alarme apresentado despoletar a criação de tickets, simplificando o processo de monitorização e contacto com o cliente;

A linguagem de programação e tecnologias a utilizar para a elaboração da plataforma são do critério do estagiário. Não obstante, dadas as suas funcionalidades da plataforma, será necessário a integração com as plataformas da empresa desenvolvidas em Python, mesmo que por intermédio de uma API REST. A utilização de tecnologias como JAVA e XML serão sempre necessárias para apresentação de resultados ao utilizador em mobile.

PROGRAMA DE TRABALHOS

O estágio consistirá nas seguintes actividades e respectivas tarefas:

- T1 – *Estudo de Plataformas* – Nesta fase o estagiário vai perceber a forma de operação das plataformas de monitorização de rede e a informação que estas fornecem ao operador de monitorização para a realização do seu trabalho.
- T2 – *Levantamento de Requisitos* – Dadas as plataformas e registos existentes, o estagiário deverá em T2, perceber quais os requisitos necessários ao registo de informação, visualização e comunicação do sistema.
- T3 – *Desenvolvimento* – Desenvolvimento de requisitos identificados em T2.
- T4 – *Testes* – Testes funcionais e atributos de qualidade ao trabalho desenvolvido ao longo das tarefas anteriores.
- T5 – *Relatório* – Relatório de Estágio.

CALENDARIZAÇÃO DE TAREFAS

As Tarefas acima descritas, incluindo os testes de validação de cada módulo, serão executadas de acordo com a seguinte calendarização:

O plano de escalonamento dos trabalhos é apresentado em seguida, sendo que está planeada a execução de 1200 horas de trabalho, o que equivale a 30 semanas.

INI		Início dos trabalhos
M1	(INI + 2 Semanas)	Tarefa T1 terminada
M2	(INI + 2 Semanas)	Tarefa T2 terminada
M3	(INI + 16 Semanas)	Tarefa T3 terminada
M4	(INI + 4 Semanas)	Tarefa T4 terminada
M5	(INI + 6 Semanas)	Tarefa T5 terminada

RESULTADOS

Os resultados a apresentar pelo estagiário serão consubstanciados nas várias metas estabelecidas para o projecto, sendo que serão espectáveis 2 períodos de avaliação fundamentais:

- Setembro de 2017

Avaliação preliminar de arquitectura: Serão analisados os resultados do trabalho efectuado para as tarefas T1. T2 e no inicio de T3 verificando a exequibilidade da arquitectura e o estado da implementação já iniciada;

- Julho de 2017

Avaliação da plataforma e de resultados: Será avaliada nesta fase a adequabilidade e resposta da proposta desenvolvida aos requisitos propostos inicialmente pela Dognaedis;
Será também avaliada a qualidade da documentação elaborada pelo estagiário, materializada no seu relatório de estágio.

LOCAL DE TRABALHO

O trabalho será realizado a partir dos escritórios da Dognaedis em Coimbra.
Horário de Trabalho: 9h00 às 18h00, com uma hora para almoço.