IPG Politécnico da Guarda
Polytechnic of Guarda

RELATÓRIO DE PROJETO

Licenciatura em Engenharia Informática

Joel Gonçalves Caetano

dezembro | 2016

PROJETO DE INFORMÁTICA

Aplicação Móvel Android - MediRemind

JOEL GONÇALVES CAETANO

RELATÓRIO DE PROJETO DE INFORMÁTICA PARA A OBTENÇÃO DO GRAU DE LICENCIADA EM ENGENHARIA INFORMÁTICA

DEZEMBRO 2016

Ficha de identificação

Nome: Joel Gonçalves Caetano

Número do aluno: 1010795

Morada: Avenida Joaquim Almeida Coelho nº41

Localidade: Dornelas, Aguiar da Beira

Telemóvel: 963630019

Correio eletrónico: jogcaetano89@hotmail.com

Instituição

Instituto Politécnico da Guarda

Escola Superior de Tecnologia e Gestão

Morada: Avenida Dr. Francisco Sá Carneiro 50, 6300-559 Guarda

Telefone: 271 220 100

Fax: 271 220 150

Correio eletrónico: estg-geral@ipg.pt

Dados do projeto:

- Início: 1 de julho de 2016

- Final: 17 de novembro de 2016

Agradecimentos

Em primeiro lugar, agradeço à minha família e namorada pelo apoio incondicional prestado durante todo o meu percurso pessoal e académico.

Em segundo lugar, também gostaria de agradecer ao professor Noel Lopes, por me ter ajudado a ultrapassar as dificuldades durante o desenvolvimento deste projeto.

Em terceiro lugar gostaria de agradecer à professora Maria Clara Silveira, por me ter ajudado com a planificação, toda a análise de requisitos de todo o projeto.

Resumo

O presente relatório descreve o projeto desenvolvido no âmbito da Unidade Curricular de Projeto de Informática do curso de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

O projeto desenvolvido é uma plataforma, designada de MediRemind, para ajudar na gestão de medicamentos de um agregado familiar. Os membros do agregado familiar iniciam sessão com o mesmo login, na aplicação Mobile, e assim registam todos os medicamentos que têm em casa, podendo, cada membro, definir planos de toma individualmente.

A plataforma está dividida em duas partes, a aplicação *Android*, onde há a interação com o utilizador e onde são guardados os planos de toma na Base de Dados SQLite, e o *Webservice*, onde são guardados os medicamentos de cada utilizador na Base de Dados MySQL.

A plataforma visa registar os medicamentos do agregado familiar para que estejam sempre disponíveis no *Smartphone*, evitando desperdícios dos mesmos. A aplicação envia, também, notificações de alerta a cada membro, quando estiver na hora da toma do medicamento.

Palavras-chave: Android, Webservice, agregado familiar, medicamentos, planos de toma.

Abstract

The document describes the project developed under the degree, in Computer Engineering, from the school of Technology and Management, Polytechnic Institute of Guarda.

The project developed is a platform, designated MediRemind to help in medicine management of a hosehold. The family members log with the same login, in the mobile application, and thus regist all medicines they have at home and can and program take plans for each member.

The project is divided in two parts, the Android application where there is interaction with the user and where are stored the plans in SQLite database, and Webservice, wich are saved all user medications in the MySQL database.

The platform aims to register the household medications that are always avaliable on the Smartphone, to avoid wasting them. The application sends, too, to the user alert notifications, at the time of taking the medicine or when i tis ending.

Keywords: Android, Webservice, hosehold, medications, taking plans, family.

Glossário de Siglas

Sigla	Descrição
JSON	JavaScript Object Notation
BD	Base de Dados
PHP	PHP Hypertext Preprocessor
XML	XML - Extensible Markup Language
UML	Unified Modeling Language
HTTP	Hypertext Transfer Protocol
REST	Representation State Transfer
SQL	Structure Query Language
IDE	Integrated Development Environment
PDO	PHP Data Object

Índice

Índice de figuras

Índice de tabelas

1. Introdução

O presente relatório descreve o projeto desenvolvido pelo aluno Joel Gonçalves Caetano, no âmbito da Unidade Curricular de Projeto de Informática do curso de Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

A evolução dos dispositivos móveis tornou possível a existência de inúmeras aplicações que possibilitam melhorar a qualidade de vida das pessoas, resolvendo problemas simples do quotidiano, através da utilização desses mesmos dispositivos móveis, tornando assim possível a realização deste projeto.

O projeto desenvolvido é uma plataforma, aplicação Mobile e *Webservice*, designada MediRemind, que faz a gestão de medicamentos de um agregado familiar.

A aplicação Mobile, em *Android*, é onde se faz toda a interação com o utilizador. Os membros do agregado familiar iniciam sessão com o mesmo login, tendo assim acesso a todos os medicamentos registados por cada membro. Os medicamentos são registados numa base de dados externa MySQL, através de *Webservice*, possibilitando assim que todos os membros tenham acesso aos mesmo medicamentos.

A plataforma possibilita, também, a criação de planos de toma individuais dos medicamentos registados, alertando o utilizador na hora da toma do medicamento, possibilitando assim que cada membro tenha os seus próprios planos de toma e alertas. Isto é possível devido ao facto de que todos os planos de toma serem guardados na base de dados interna do *Android*, o SQLite, o que quer dizer que um plano de toma só fica disponível no dispositivo onde foi criado.

1.1. Motivação

A principal motivação para o desenvolvimento deste projeto foi criar uma solução viável para ajudar as pessoas a não deixar medicamentos esquecidos, evitando assim o seu desperdício.

Muitas vezes, os medicamentos ficam esquecidos em casa e, sem as pessoas darem conta, acabam por adquirir o mesmo medicamento, escusadamente.

Outro motivo para a realização deste projeto foi ajudar as pessoas na hora de tomar um medicamento, com planos de toma e alertas de medicamentos.

A aplicação destina-se a todas as pessoas, visto que toda a gente tem medicamentos em casa e assim, facilita no controlo dos medicamentos, evitando assim o seu desperdício.

1.2. Objetivos previstos

O principal objetivo é criar uma aplicação para gestão de medicamentos, que permita:

- Inserir, alterar e eliminar medicamentos
- Inserir, alterar e eliminar planos de toma
- Receber alertas da hora da toma
- Gerir os medicamentos do agregado familiar
- Gerir planos de toma para cada membro

1.3. Estrutura do documento

O presente relatório é constituído por 6 capítulos. No primeiro capítulo é apresentada uma breve introdução ao problema do projeto. No segundo capítulo é apresentado o estado da arte, fazendo uma breve descrição de algumas aplicações existentes no mercado que se enquadram no tema do projeto. No terceiro capítulo é apresentada a metodologia e a descrição de todas as tarefas necessárias para o desenvolvimento do projeto. No quarto capítulo é descrita toda a análise de requisitos, tais como diagrama de casos de uso, descrição de cada caso de uso, diagrama de classes, etc. No quinto capítulo são apresentadas todas as tecnologias e ferramentas utilizadas no desenvolvimento do projeto e finalmente no sexto capítulo é apresentado e descrito todo o processo de desenvolvimento da aplicação.

2. Estado da Arte

Neste capítulo serão apresentadas algumas as aplicações existentes no mercado. Depois de pesquisas feitas a aplicações similares, foram encontradas duas, a "Farmácias Portuguesas" e "A sua Farmácia Online".

2.1. Introdução

As tecnologias de Informação estão a evoluir exponencialmente e cada vez mais são essenciais no nosso quotidiano (Altivo de Almeida, 2016). Uma das vertentes das Tecnologias de Informação são as Tecnologias de Apoio.

Hoje em dia existem inúmeras aplicações de apoio ao cliente, tais como aplicações de farmácias online, em que para comprar um simples medicamento já não é necessário a deslocação à farmácia, basta aceder à própria aplicação. As aplicações existentes nesta área focam-se essencialmente na venda de medicamentos, plano de tomas e entrega ao domicílio.

2.2. Exemplos de Aplicações Existentes

As aplicações de farmácia online existentes descritas neste documento são: "Farmácias Portuguesas"[1] e "A Sua Farmácia Online"[2], que serão descritos nos subcapítulos 2.2.1 e 2.2.2, respetivamente.

2.2.1. Farmácias Portuguesas

Uma das aplicações mais conhecidas é a "Farmácias Portuguesas". A presente aplicação dispõe de uma versão Web e uma versão mobile. Uma das características desta aplicação é um cartão virtual designado "Cartão Saúda". Com este cartão, o utilizador tem acesso aos pontos disponíveis, à data de validade dos pontos e quanto já poupou em compras.

[1] Farmácias Portuguesas, Farminveste – Investimentos, Participações e Gestão, S.A (https://www.farmaciasportuguesas.pt)
[2] A Sua Farmácia Online, Farmácia Sousa Torres, S.A (https://www.asuafarmaciaonline.pt)

Para fazer compras dentro da aplicação é necessária prévia adesão, visto que o utilizador compra os produtos com os pontos disponíveis no cartão (Farmácias Portuguesas, 2016).

Na versão Web, o utilizador tem acesso ao catálogo de produtos, informação sobre os produtos e doenças e localização das farmácias locais.

O principal foco da aplicação é na versão Mobile, como se pode ver na Figura 1, onde o utilizador tem acesso ao mesmo conteúdo da versão Web e, para além disso, pode registar os seus parâmetros de saúde, guardar as farmácias nos favoritos e inserir plano de tomas. Com a aplicação Mobile, o utilizador tem a possibilidade de receber alertas na hora da toma do medicamento e do fim da embalagem (Farminveste, 2016).

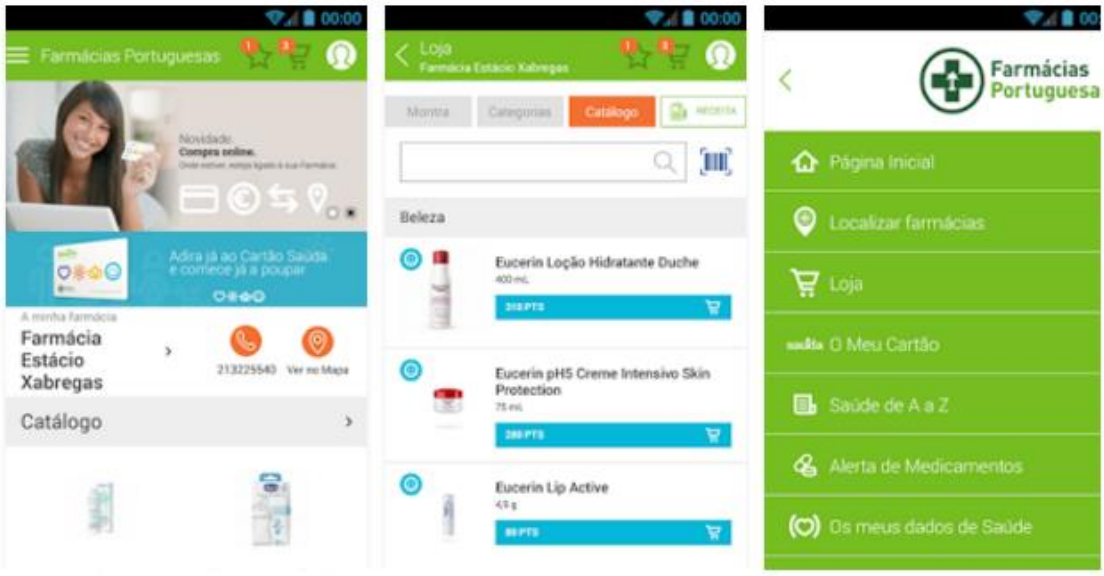


Figura 1 - Farmácias Online - Versão Mobile

Excetuando compras e informação do cartão saúda, o utilizador tem acesso a todas as funcionalidades, mesmo sem estar registado, ou seja, qualquer utilizador da aplicação pode registar plano de tomas, procurar farmácias e receber alertas.

2.2.2. A Sua Farmácia Online

A aplicação "A Sua Farmácia Online", da empresa Farmácia Sousa Torres, que só existe em versão Web, concentra-se essencialmente na funcionalidade de venda de medicamentos e entregas ao domicílio, mas também dispõe de informação sobre os produtos e doenças.

Na Figura 2 é apresentada a página principal da aplicação Web "A Sua Farmácia Online".

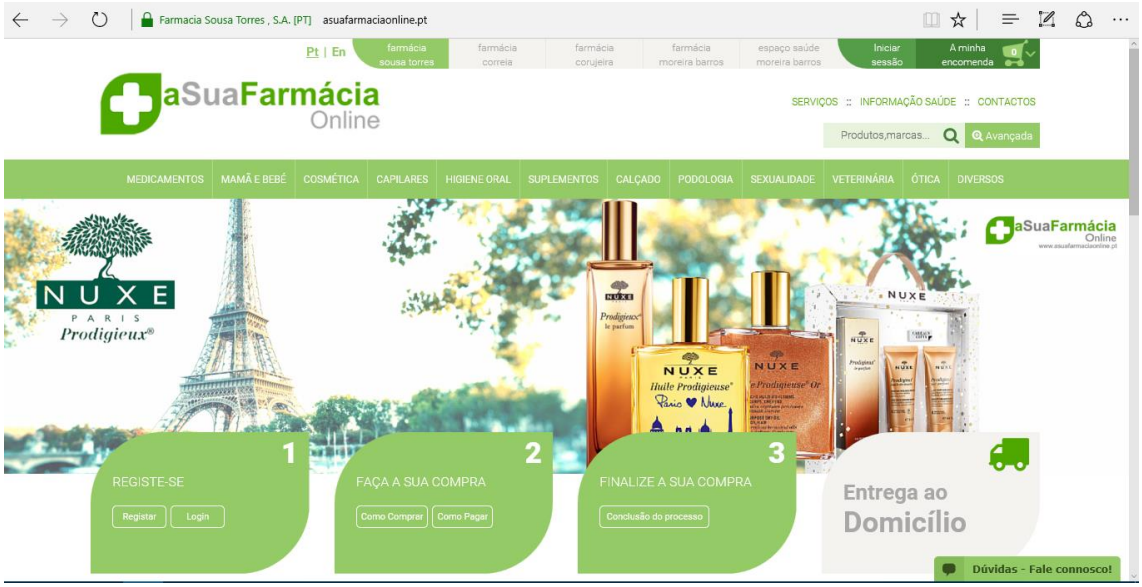


Figura 2 - A Sua Farmácia Online

Nesta aplicação, o utilizador tem a acesso a um vasto número de produtos e informação sobre eles.

Além destas, existem mais algumas aplicações de farmácia online, como "Farmácias MFO[3]", para encontrar produtos das farmácias mais próximas, "Farmácia Mobile[4]", uma aplicação para encontrar farmácias de serviço e "Farmácia Pipa[5]", que tem como função a venda de produtos ao consumidor. Também existem outras para gerir a toma dos medicamentos do utilizador, como a "MediSafe[6]" e "Recordatorio Medicina[7]".

[3] Farmácias MFO - https://play.google.com/store/apps/details?id=com.adoramedia.appmifarmaciaonline&hl=pt_PT
[4] Farmácias Mobile - https://play.google.com/store/apps/details?id=com.goingsolutions.farmaciamobile&hl=pt_PT
[5] Farmácia Pipa - https://play.google.com/store/apps/details?id=farmPipa.cartoes&hl=pt_PT
[6] MediSafe - https://play.google.com/store/apps/details?id=com.medisafe.android.client&hl=pt_PT
[7] Recordatorio Medicina - https://play.google.com/store/apps/details?id=com.naticode.recordatoriomedicina&hl=pt_PT

2.3. Análise comparativa

Estas duas aplicações ("Farmácias Portuguesas" e "A Sua Farmácia Online") foram destacadas devido às diferenças entre elas. Por um lado, a aplicação "A sua Farmácia Online" é uma aplicação Web que a sua principal função é a venda e entrega de medicamentos ao domicilio, enquanto que a aplicação "Farmácia Portuguesa", dispõe igualmente de uma aplicação Web para a venda de medicamentos, mas o seu principal foco é na aplicação Mobile que permite controlar plano de tomas e receber alertas. Estas aplicação, sendo muito completas, não permitem a gestão de medicamentos dos utilizadores. A seguinte tabela (Tabela 1), mostra as funcionalidades de cada uma das aplicações apresentadas anteriormente.

Tabela 1 - Funcionalidades das aplicações similares existentes

Funcionalidades	**Farmácias Portuguesas**	**A sua Farmácia Online**
Plano de Tomas	Sim	Não
Alertas	Sim	Não
Gestão de medicamentos do agregado familiar	Não	Não
Pesquisa de medicamentos	Sim	Sim
Informação de saúde	Sim	Não
Venda de medicamentos	Sim	Sim
Pesquisa de farmácias	Sim	Não

Como se pode ver na tabela acima, a aplicação mais completa é "Farmácias Portuguesas", mas nenhuma faz gestão de medicamentos do agregado familiar, como é o caso da aplicação MediRemind. Além disso, as aplicações acima descritas são para um utilizador, não havendo partilha de medicamentos entre membros, ou seja, no caso da "Farmácias Portuguesas", se vários utilizadores iniciarem sessão com o mesmo login, terão todos acessos aos mesmos planos de toma, não existe um plano de toma para cada dispositivo individualmente.

A principal vantagem da aplicação MediRemind é que é para ser usada pelo agregado familiar, podendo os seus membros iniciar sessão com o mesmo login e assim terem todos acesso aos mesmos medicamentos, podendo definir planos de toma individualmente.

3. Metodologia e Resultados esperados

Neste capítulo será apresentada a metodologia usada na aplicação, assim como a descrição de cada tarefa para a realização do presente projeto.

3.1. Metodologia

Para a realização do projeto, foi utilizada a Metodologia XP (Extreme Programming). A Metodologia XP é uma Metodologia Ágil para empresas médias e pequenas que desenvolvem software que sofrem alterações rapidamente. As principais diferenças entre a Metodologia XP e as restantes Metodologias são: o feedback constante, a abordagem incremental e maior comunicação entre os programadores e os clientes (Fonseca, 2016).

A escolha desta metodologia foi para facilitar a comunicação com o professor orientador Noel Lopes, facultando-lhe, periodicamente, uma nova versão do documento.

Vantagens da escolha da Metodologia XP no enquadramento do projeto:

- **Pessoas e interações, ao contrário de processos e ferramentas –** houve sempre comunicação com os professores envolvidos em vez de se manter a análise inicial de requisitos.
- **Software a funcionar, ao contrário de documentação extensa –** sempre que possível, foi mostrado ao orientador o estado da aplicação, com as funcionalidades já implementadas para termos um feedback do estado do desenvolvimento e se está a ser seguido o melhor caminho.
- **Respostas rápidas às mudanças, ao contrário de seguir planos previamente definidos –** houve várias alterações aos requisitos do projeto.
- **Planeamento de iterações –** cada release pode ter várias iterações, ou seja, o software pode sofrer várias mudanças ao longo do seu desenvolvimento.

O principal foco da Metodologia XP é dar agilidade ao desenvolvimento e procurar garantir a satisfação do cliente. As práticas, regras e valores desta metodologia garantem um agradável ambiente de desenvolvimento de software.

A plataforma foi desenvolvida segundo eta metodologia e pode verificar-se na Figura 3.

Figura 3 - Modelo em espiral da aplicação MediRemind

3.2. Descrição das Tarefas

Neste capítulo serão apresentados os passos para o desenvolvimento do projeto e o que foi feito em cada um deles.

Tarefa 1: Análise de Requisitos e definição das funcionalidades da aplicação.

Tarefa 2: Estudo das API's necessárias para a realização da aplicação.

Tarefa 3: Implementação da solução.

Tarefa 4: Testes da aplicação.

Tarefa 5: Elaboração do relatório.

O Agendamento das tarefas é apresentado na Tabela 2.

Tabela 2- Descrição das tarefas do presente projeto

	Data de inicio	**Data de fim**	**Duração (em dias)**	**Média de horas por semana (h)**
Tarefa 1	03/jul	25/jul	22	6
Tarefa 2	27/jul	07/ago	10	5
Tarefa 3	08/ago	10/set	32	5
Tarefa 4	13/set	10/out	27	7
Tarefa 5	12/out	20/nov	39	5

4. Análise de Requisitos

Neste capítulo será abordada toda a análise de requisitos. Para isso foi utilizada a linguagem UML (Unified Modeling Language) através do qual foram implementados os seguintes diagramas:

- Diagrama de Contexto
- Diagrama de Casos de Uso
- Diagrama de Sequência
- Diagrama de Classes
- Diagrama de Atividades
- Diagrama de Componentes
- Diagrama de Instalação

4.1. Requisitos do Sistema

A aplicação será usada pelo utilizador, que precisa de se registar para poder utilizar a mesma.

A aplicação deverá permitir ao utilizador:

- Inserir medicamentos
- Inserir fotos dos respetivos medicamentos
- Inserir planos de toma
- Inserir intervalo de plano de toma para receber notificações
- Consultar medicamentos
- Tomar medicamento
- Editar medicamento
- Eliminar medicamento
- Editar plano de toma
- Eliminar plano de toma
- Gerir medicamentos do agregado familiar
- Gerir planos de toma para cada membro

4.2. Diagrama de Contexto

O diagrama de contexto mostra a relação que o sistema tem com as entidades externas. O diagrama de contexto não fornece informação sobre o tempo ou sincronização de processos, como por exemplo, que processos ocorrem em sequência ou em paralelo, o seu principal objetivo é simplificar a interação dos atores ou sistemas externos com a aplicação.

Neste caso, a única interação externa que existe é o utilizador.

A Figura 4 demonstra o diagrama de contexto da aplicação MediRemind.

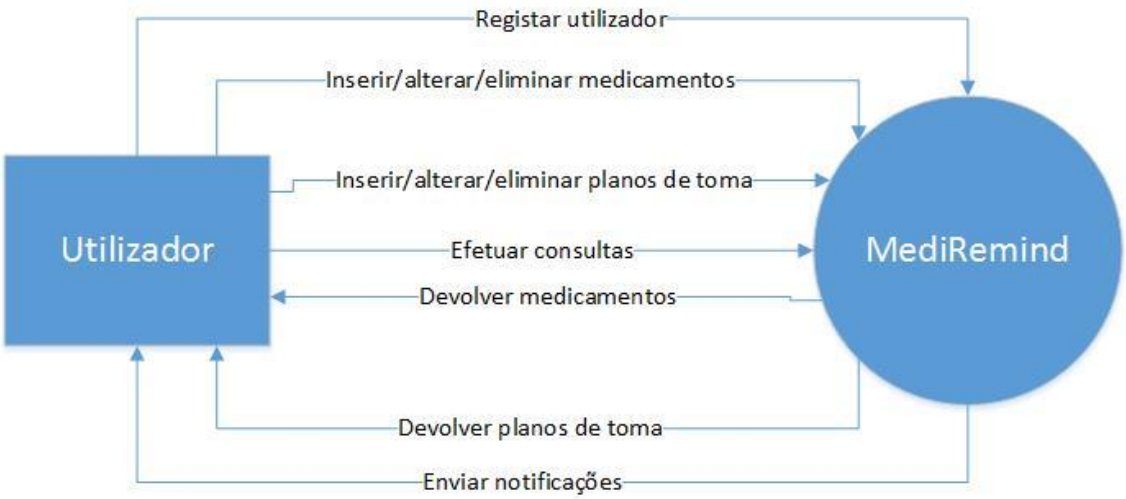


Figura 4 - Diagrama de Contexto da Aplicação MediRemind

4.3. Diagrama de Casos de Uso

Um diagrama de casos de uso é uma representação da interação do ator com o sistema, que mostra a relação entre o ator e os diferentes casos de uso em que o ator está envolvido (Wikipedia, 2016).

Na Figura 5 está representado o diagrama de casos de uso da aplicação MediRemind. Para utilizar a aplicação, o utilizador tem que ter um login válido.

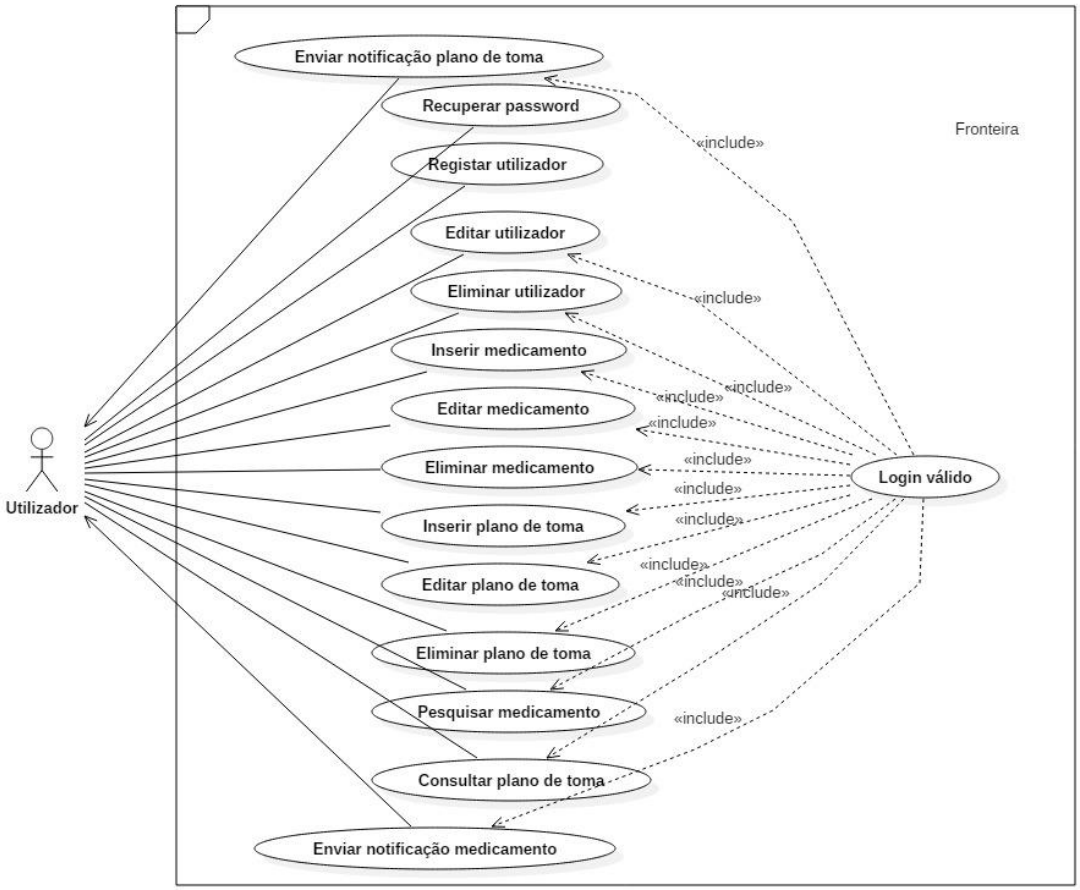


Figura 5 - Diagrama de Casos de Uso

4.4. Descrição dos Casos de Uso

Neste capítulo será especificado cada caso de uso, descrevendo a sequência de ações com o objetivo de demonstrar o comportamento do sistema através de interações com os atores. Apresentam também cenários alternativos ao caso de uso e testes necessários para garantir o bom funcionamento do caso de uso.

4.4.1. Descrição do caso de uso "Registar utilizador"

A Tabela 3 descreve com detalhe o caso de uso "Registar utilizador"

Tabela 3 - Descrição do caso de uso "Registar Utilizador"

Descrição de caso de uso "Registar utilizador"	
Nome:	Registar utilizador
Descrição:	Este caso de uso tem como objetivo registar um utilizador
Pré-condição:	Nenhuma
Caminho Principal:	1) O Ator seleciona a opção "Novo registo" 2) O sistema disponibiliza o formulário para o registo (nome, email, username, password e confirmar password) 3) O Ator preenche os campos e clica no botão "Registar" 4) O sistema envia uma mensagem ao ator "Utilizador registado com sucesso"
Caminhos Alternativos	A qualquer momento o ator pode sair do sistema, clicando no botão "Cancelar" 3. a) Sistema envia mensagem ao ator "Utilizador já existe" 3. b) Campos vazios ou inválidos 3. c) O campo "password" e "confirmar password" são diferentes 4. a) Sistema envia mensagem ao ator "Erro ao registar utilizador, verifique a ligação à Internet"
Pós-Condições	Nenhuma
Suplementos ou adornos	1) Testar criar utilizador sem campos 2) Testar validação dos campos: nome, e-mail, username, password e confirmar password.

	3) Testar encriptação SHA1 (encriptação de 160 bits que cria 40 caracteres hexadecimais)

4.4.2. Descrição do caso de uso "Eliminar utilizador"

A Tabela 4 descreve com detalhe o caso de uso "Eliminar utilizador"

Tabela 4 - Descrição do caso de uso "Eliminar utilizador"

Descrição de caso de uso "Eliminar utilizador"	
Nome:	Eliminar utilizador
Descrição:	Este caso de uso tem como objetivo eliminar o utilizador que tem sessão iniciada
Pré-condição:	Login válido
Caminho Principal:	1) O ator seleciona a opção "eliminar" 2) O sistema envia um aviso ao ator "Tem a certeza que pretende eliminar o utilizador?" 3) O ator clica no botão "Sim" 4) O sistema envia uma mensagem ao ator "Utilizador eliminado com sucesso"
Caminhos Alternativos	A qualquer momento o ator pode sair do sistema, clicando no botão "Não" 4. a) O utilizador tem planos de toma ativos, sistema envia mensagem ao ator "Não foi possível eliminar o utilizador" 4. b) Tem medicamentos associados, sistema envia mensagem ao ator "Não foi possível eliminar o utilizador" 4. c) Não tem acesso à Internet
Pós-Condições	Nenhuma
Suplementos ou adornos	Testar eliminar utilizador com planos de toma ativos Testar eliminar utilizador com medicamentos associados

4.4.3. Descrição do caso de uso “Recuperar password”

A Tabela 5 descreve com detalhe o caso de uso “Recuperar password”.

Tabela 5 - Descrição do caso de uso "Recuperar password"

Descrição de caso de uso “Recuperar password”	
Nome:	Recuperar password
Descrição:	Este caso de uso tem como objetivo recuperar a password de um utilizador registado
Pré-condição:	Nenhuma
Caminho Principal:	1) O ator seleciona a opção “Esqueceu-se da password?” 2) O sistema disponibiliza o Dialog com o formulário para introduzir o e-mail de registo 3) O ator introduz o e-mail e clica no botão “Enviar” 4) O sistema verifica se o e-mail introduzido existe na base de dados, envia uma password aleatória para o e-mail e envia uma mensagem ao ator “E-Mail enviado com sucesso”
Caminhos Alternativos	A qualquer momento o ator pode sair do sistema, clicando no botão “Cancelar” 3. a) E-Mail inválido 4. a) Sistema envia mensagem ao ator “O E-Mail não existe” 4. b) O sistema envia uma mensagem ao ator “Erro ao enviar E-Mail, verifique a ligação à Internet”
Pós-Condições	Nenhuma
Suplementos ou adornos	Testar enviar e-mail vazio Testar enviar e-mail inválido

4.4.4. Descrição do caso de uso “Login válido”

A Tabela 6 descreve com detalhe o caso de uso “Login válido”

Tabela 6 - Descrição do caso de uso "Login válido"

Descrição de caso de uso “Login válido”	
Nome:	Login válido
Descrição:	Este caso de uso tem como objetivo o utilizador iniciar sessão na aplicação
Pré-condição:	Nenhuma
Caminho Principal:	1) O ator preenche os campos username e password e clica no botão “Login” 2) O sistema verifica se o username e password inseridos coincidem com username e password da tabela “Utilizadores” da base de dados e envia uma mensagem ao ator “Login efetuado com sucesso!”.
Caminhos Alternativos	2. a) Campos vazios ou inválidos 2. b) O username e/ou password não coincidem com o username e/ou password da tabela “Utilizadores” da base de dados 2. c) Sistema envia mensagem ao ator “Username não existe” 2. d) Sistema envia mensagem ao ator “Não foi possível iniciar sessão, verifique a ligação à Internet”
Pós-Condições	Inserir utilizador que iniciou sessão na tabela “Utilizador_Interno” da base de dados SQLite
Suplementos ou adornos	1) Testar iniciar sessão com campos vazios ou inválidos 2) Testar iniciar sessão com username e/ou password errados

4.4.5. Descrição do caso de uso "Inserir medicamento"

A Tabela 7 descreve com detalhe o caso de uso "Inserir medicamento"

Tabela 7 - Descrição de caso de uso "Inserir medicamento"

Descrição de caso de uso "Inserir medicamento"	
Nome:	Inserir medicamento
Descrição:	Este caso de uso tem como objetivo inserir um novo medicamento que tem em casa
Pré-condição:	Login válido
Caminho Principal:	1) O ator seleciona a opção inserir (botão "+") 2) O sistema disponibiliza o formulário para inserir um medicamento e a lista de categorias do respetivo idioma do dispositivo móvel 3) O ator seleciona a categoria 4) O sistema disponibiliza para introduzir a dosagem e a quantidade 5) O ator introduz a dosagem e a quantidade (por exemplo nº de comprimidos); e seleciona a data de validade do medicamento a partir do calendário e clica no botão inserir. 6) O sistema verifica se existe algum medicamento com o mesmo nome na base de dados e envia mensagem ao ator "Medicamento inserido com sucesso".
Caminhos Alternativos	A qualquer momento o ator pode sair do sistema, clicando no botão "Cancelar" 3. a) Não existe a categoria pretendida 5. a) Na quantidade e dosagem apensa deixar inserir caracteres numéricos 5. b) Data de validade menor que data do sistema 5. c) Campos (nome, dosagem, quantidade) vazios 6. b) Sistema envia mensagem "Não foi possível inserir o medicamento, verifique ligação à Internet"
Suplementos ou adornos	1) Testar inserir medicamento sem campos (nome, dosagem, quantidade) 2) Testar campos obrigatórios: Nome, dosagem e quantidade 3) Testar inserir data de validade menor que data do sistema

4.4.6. Descrição do caso de uso "Eliminar medicamento"

A Tabela 8 descreve com detalhe o caso de uso "Eliminar medicamento"

Tabela 8 - Descrição de caso de uso "Eliminar medicamento"

Descrição de caso de uso "Eliminar medicamento"	
Nome:	Eliminar medicamento
Descrição:	Este caso de uso tem como objetivo eliminar um medicamento
Pré-condição:	Login válido
Caminho Principal:	1) O ator clica no ícone de eliminar 2) O sistema envia um aviso ao ator "Tem a certeza que pretende eliminar o medicamento?" 3) O ator clica no botão "Sim" 4) O sistema envia uma mensagem ao ator "Medicamento eliminado com sucesso"
Caminhos Alternativos	A qualquer momento o ator pode sair do sistema, clicando no botão "Não" 4. a) Sistema envia mensagem "Não foi possível eliminar o medicamento porque está num plano de toma ativo" 4. b) Sistema envia mensagem "Não foi possível eliminar o medicamento, verifique a ligação à Internet"
Pós-Condições	Nenhuma
Suplementos ou adornos	Nenhum

4.4.7. Descrição do caso de uso "Inserir plano de toma"

A Tabela 9 descreve com detalhe o caso de uso "Inserir plano de toma"

Tabela 9 - Descrição de caso de uso "Inserir plano de toma"

Descrição de caso de uso "Inserir plano de toma"	
Nome:	Inserir plano de toma
Descrição:	Este caso de uso tem como objetivo inserir um novo plano de toma
Pré-condição:	Login válido
Caminho Principal:	1) O ator seleciona a opção inserir (botão "+") do menu "Alerta de medicamentos" 2) O sistema disponibiliza o formulário para inserir um plano de tomas e a lista de medicamentos 3) O ator seleciona o medicamento 4) O sistema disponibiliza o medicamento e a respetiva fotografia 5) O ator clica no campo "Data de início" 6) O sistema disponibiliza o calendário para a introdução da data de início do plano 7) O ator preenche os campos duração, intervalo e quantidade e clica no botão "Inserir" 8) O sistema envia uma mensagem ao ator "Plano de toma inserido com sucesso"
Caminhos Alternativos	A qualquer momento o ator pode sair do sistema, clicando no botão "Cancelar" 3. a) Não existe o medicamento pretendido porque não tem acesso à internet 7. a) Se campos vazios (duração, intervalo, quantidade), pede para introduzir novamente 8. a) Sistema envia mensagem "Não foi possível inserir o plano de toma"
Pós-Condições	1) Gerar notificações em função do intervalo de tomas 2) Enviar notificação ao utilizador na hora da toma 3) Atualizar quantidade do medicamento tomado

Suplementos ou adornos	1) Testar inserir plano de toma com campos vazios 2) Testar campos obrigatórios (duração, intervalo, quantidade)

4.4.8. Descrição do caso de uso "Eliminar plano de toma"

A Tabela 10 descreve com detalhe o caso de uso "Eliminar plano de toma".

Tabela 10 - Descrição de caso de uso "Eliminar plano de toma"

Descrição de caso de uso "Eliminar plano de toma"	
Nome:	Eliminar plano de toma
Descrição:	Este caso de uso tem como objetivo eliminar um plano de toma
Pré-condição:	Login válido
Caminho Principal:	1) O ator seleciona a opção eliminar 2) O sistema envia um aviso ao ator "Tem a certeza que pretende eliminar o plano de toma?" 3) O ator clica no botão "Sim" 4) O sistema envia uma mensagem ao ator "Plano de toma eliminado com sucesso"
Caminhos Alternativos	A qualquer momento o ator pode sair do sistema, clicando no botão "Não" 6. a) Sistema envia mensagem "Não foi possível eliminar o plano de toma porque está ativo"
Pós-Condições	Nenhuma
Suplementos ou adornos	Testar eliminar um plano de toma ativo

4.4.9. Descrição do caso de uso "Pesquisar medicamento"

A Tabela 11 descreve com detalhe o caso de uso "Pesquisar medicamento".

Tabela 11 - Descrição do caso de uso "Pesquisar medicamento"

Descrição de caso de uso "Pesquisar medicamento"	
Nome:	Pesquisar medicamento
Descrição:	Este caso de uso tem como objetivo pesquisar, por nome, um medicamento que tem em casa
Pré-condição:	Login válido
Caminho Principal:	1) O ator seleciona a opção pesquisar (ícone lupa) 2) O sistema disponibiliza a escrita de caracteres 3) O ator introduz caracteres alfanuméricos 4) O sistema apresenta o(s) medicamento(s) que contém a sequência de caracteres introduzidos
Caminhos Alternativos	A qualquer momento o ator pode sair do sistema, clicando na seta "voltar" 4. a) O sistema apresenta uma lista vazia quando não há nenhum nome do medicamento que corresponde aos caracteres introduzidos 4. b) Caracteres inválidos
Pós-Condições	Nenhuma
Suplementos ou adornos	Testar pesquisar medicamentos por caracteres inválidos (números, caracteres especiais)

4.5. Diagramas de Sequência

O diagrama de sequência representa a sequência de processos no programa. Por outras palavras, descreve a maneira como os grupos de objetos se comportam ao longo do tempo.

Nesta secção encontram-se representados os diagramas de sequência que dizem respeito aos casos de uso apresentados anteriormente. Em cada um deles são usados objetos e mensagens que demostram a interação do ator para a realização de um caso de uso.

4.5.1. Diagrama de Sequência "Registar utilizador"

Para utilizar a aplicação, o utilizador precisa de se registar. Na Figura 6 está o diagrama de sequência para efetuar o registo.

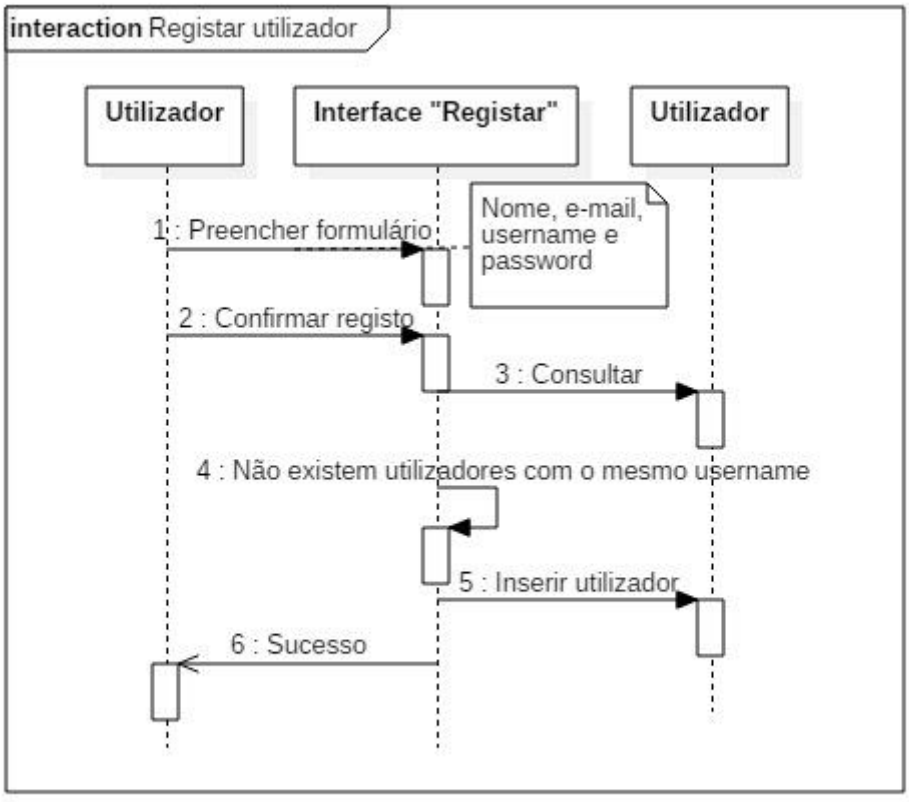


Figura 6 - Diagrama de Sequência "Registar utilizador" na aplicação MediRemind

4.5.2. Diagrama de Sequência “Eliminar utilizador”

A Figura 7 demonstra como é feita a eliminação do utilizador que tem sessão iniciada. Só é permitida a eliminação do utilizador se não tiver medicamentos associados ou planos de toma ativos.

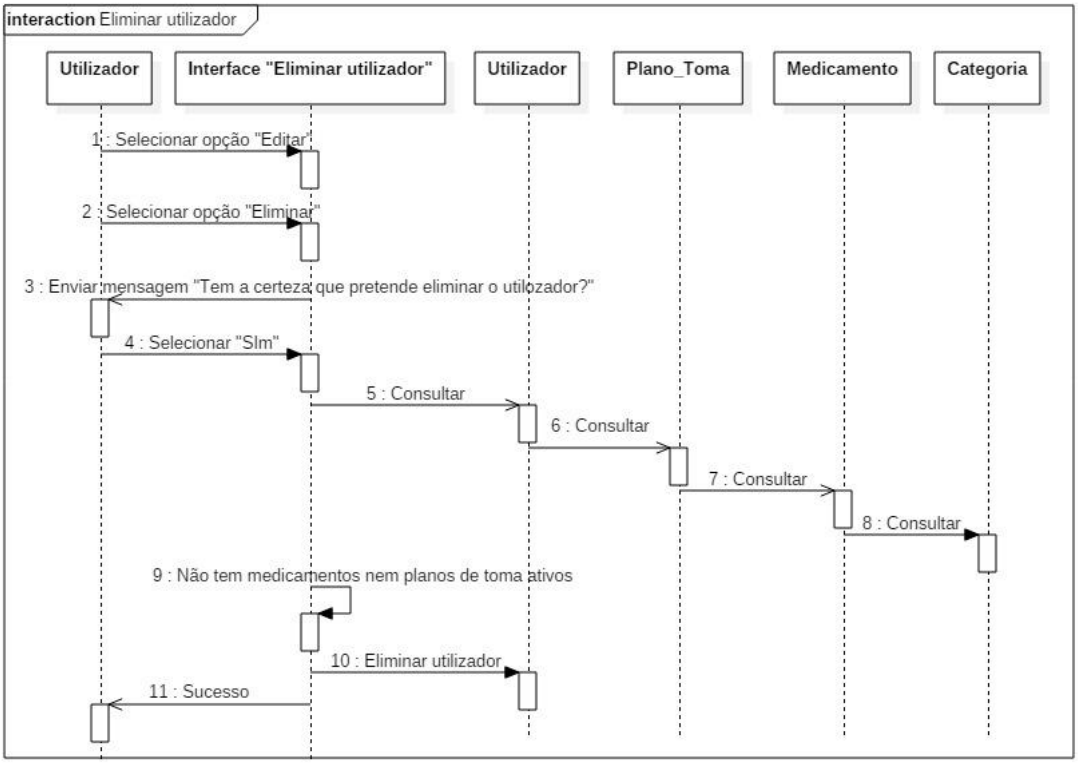


Figura 7 - Diagrama de Sequência "Eliminar utilizador"

4.5.3. Diagrama de Sequência "Login válido"

Ao fazer login, o sistema insere os dados do utilizador que iniciou sessão na tabela Utilizado_Interno da base de dados SQLite para que, da próxima vez que o utilizador iniciar a aplicação, não precise de voltar a inserir o username e password.

A Figura 8 demonstra como é feito o login do utilizador.

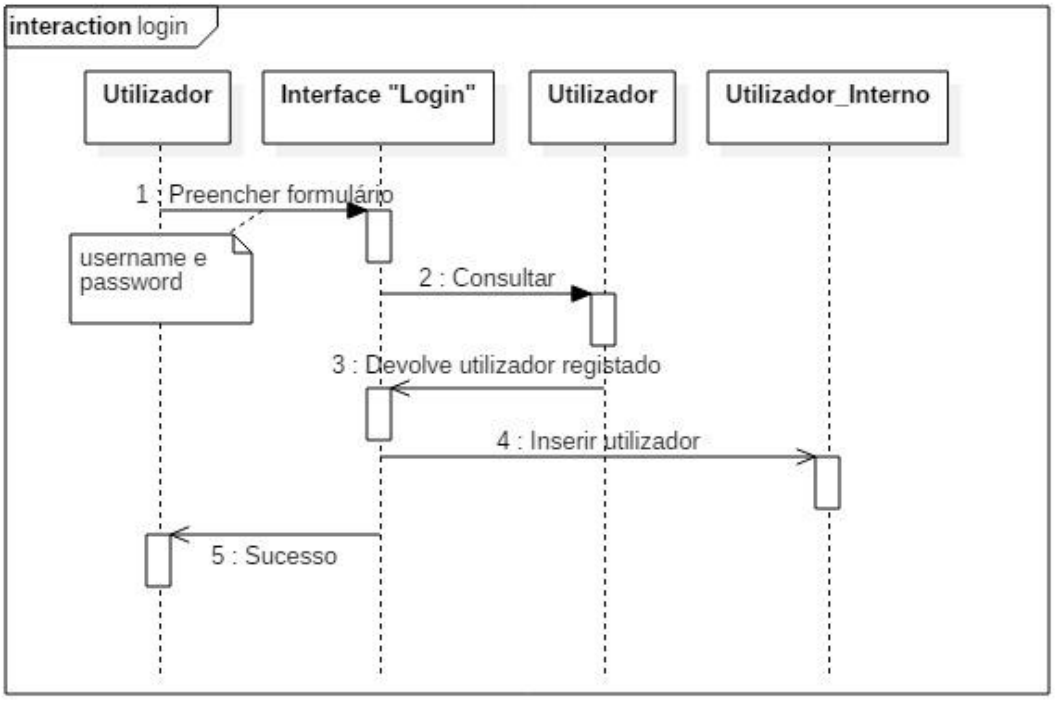


Figura 8 – Diagrama de Sequência "Login" na aplicação MediRemind

4.5.4. Diagrama de Sequência “Inserir medicamento”

A Figura 9 demonstra como é feito o registo de um novo medicamento que tem em casa. Ao inserir um novo medicamento com o mesmo nome, o utilizador é notificado que já existe um medicamento com o mesmo nome e decide se quer inserir ou não.

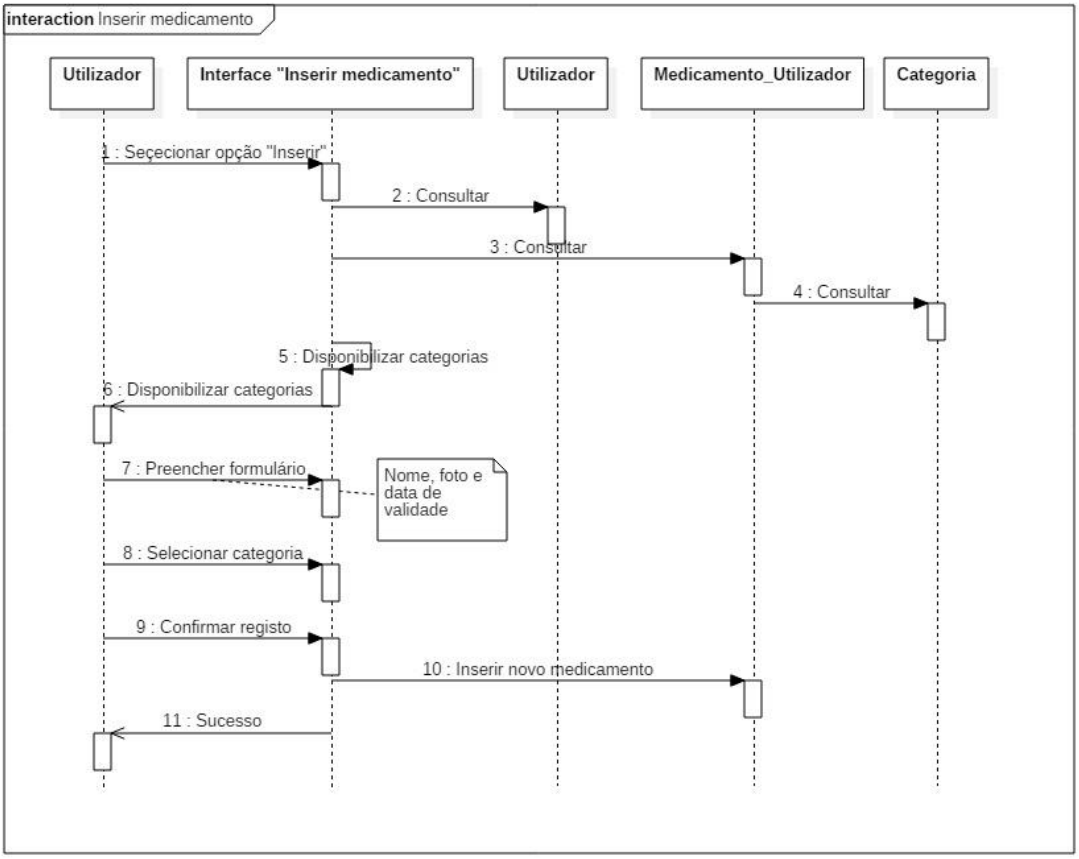


Figura 9 - Diagrama de Sequência "Inserir medicamento" na aplicação MediRemind

4.5.5. Diagrama de Sequência “Eliminar medicamento”

A Figura 10 demonstra como é feita a eliminação de um medicamento. Só é permitida a eliminação do medicamento se este não estiver associado a um plano de toma ativo.

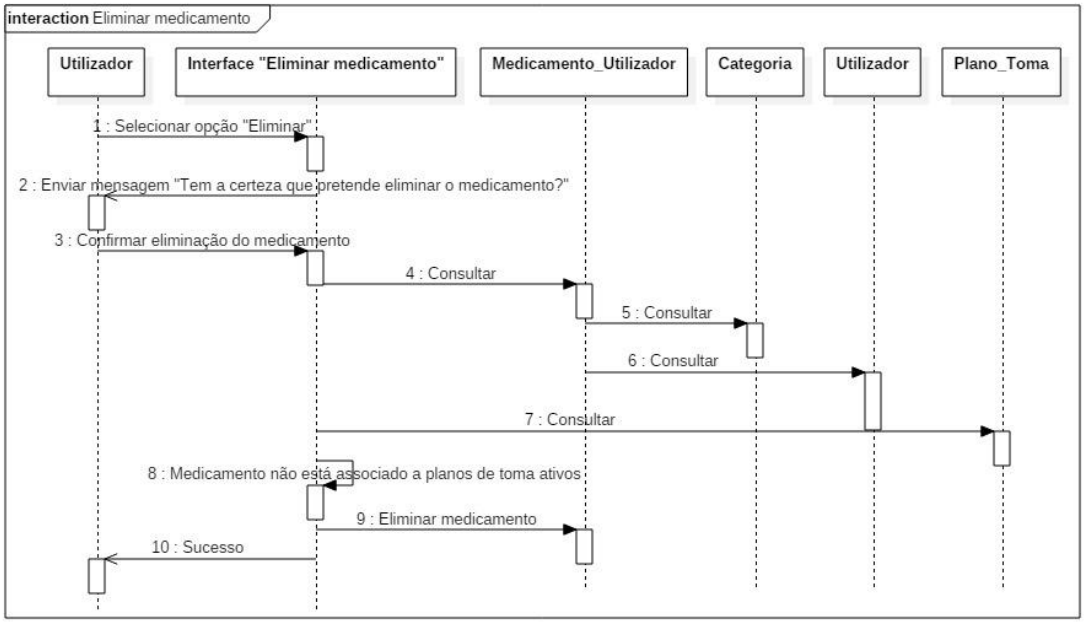


Figura 10 - Diagrama de Sequência "Eliminar medicamento" na aplicação MediRemind

4.5.6. Diagrama de Sequência “Inserir plano de toma”

A Figura 11 demonstra como é feito o registo de um novo plano de toma.

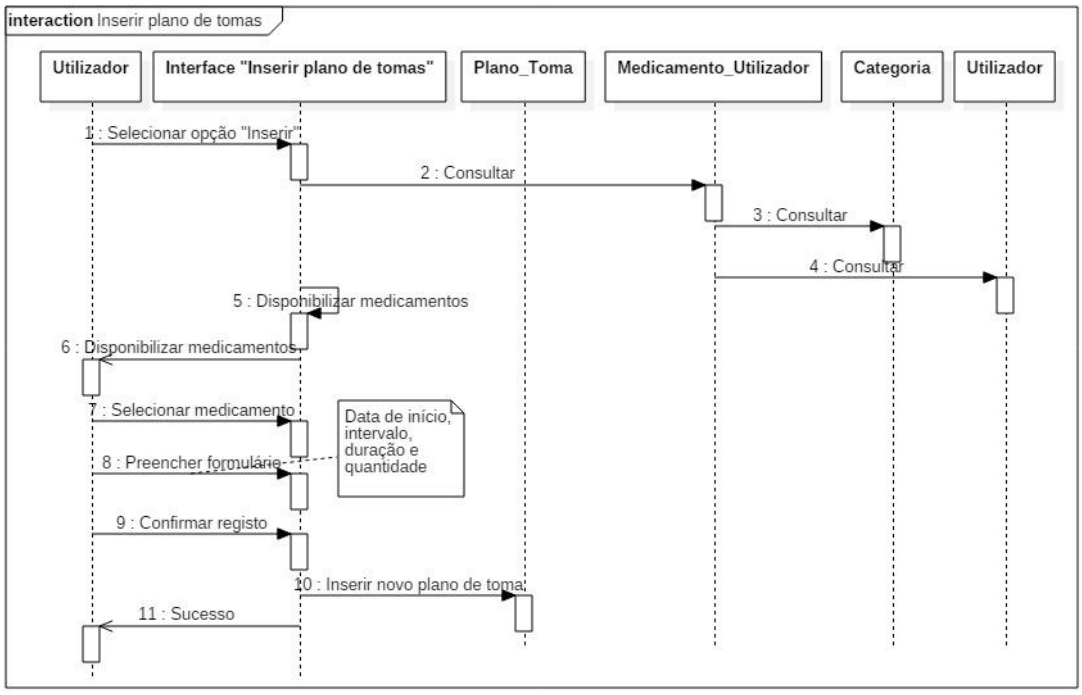


Figura 11 - Diagrama de Sequência "Inserir plano de toma"

4.5.7. Diagrama de Sequência “Eliminar plano de toma”

A Figura 12 demonstra como é feita a eliminação de um plano de toma. Só é permitida a eliminação do plano de toma este não estiver ativo.

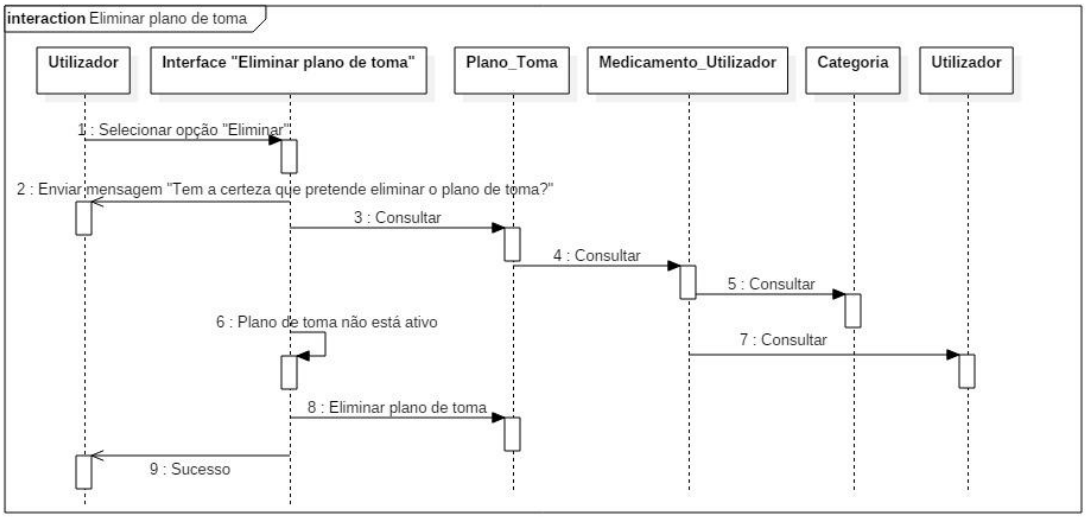


Figura 12 - Diagrama de Sequência "Eliminar plano de toma

4.6. Diagrama de Classes

Um Diagrama de Classes é um diagrama estático que representa a estrutura completa da aplicação.

Cada classe é composta pelo nome, atributos e respetivas operações, que representa o papel dos atores no sistema.

Neste projeto foram utilizadas duas bases de dados, como é demonstrado na Figura 13, uma base de dados externa (MySQL) e uma base de dados interna do Android (SQLite) que comunicam entre si através de Webserevice.

A comunicação é feita através das classes Medicamento_UItilizador e Plano_Tomas. O utilizador só pode inserir um novo plano de toma se tiver o medicamento associado à sua conta.

A classe Utilizador_Interno serve apenas para guardar a informação do utilizador que inicia sessão, se este assim o permitir, para que da próxima vez que iniciar a aplicação, não precisar de voltar a inserir os dados de login.

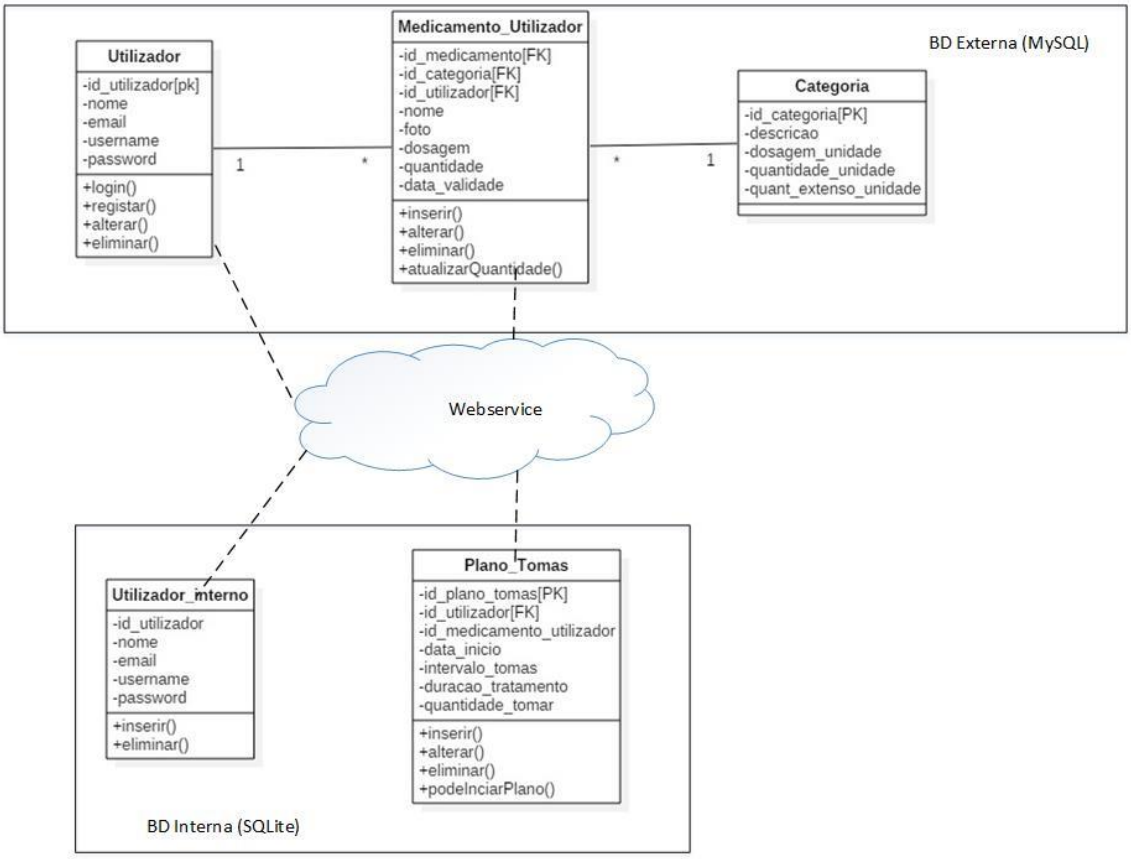


Figura 13 - Diagrama de Classes da aplicação MediRemind

4.7. Semântica das Classes

A semântica das classes é a descrição das classes utilizadas na aplicação. É composta pelos atributos, tipo de dados, descrição, valores válidos, formato e restrições de cada classe do diagrama.

- **Atributo –** campos da classe.
- **Tipo de dados –** valores que compõem o atributo.
- **Descrição –** descrição do que representa o atributo da classe.
- **Valores válidos –** esta coluna tem como objetivo referenciar os valores os valores válidos no contexto onde são inseridos.
- **Formato –** representa o atributo, por exemplo, no caso da data, DD-MM-AAA.
- **Restrição –** a maneira como é tratado o atributo, se é tratado pelo sistema ou pelo utilizador.

Antes de apresentar a descrição das semânticas das classes, será apresentado o algoritmo das mesmas.

4.7.1. Algoritmos

Algoritmo 1 - Registar

Algoritmo +Registar()

```
procedure +Registar()
        O sistema gera o ID automaticamente
        Introduzir todos os campos obrigatórios
        SE campos obrigatórios preenchidos e username inexistente
                Registar
        FIM SE
end procedure
```

Algoritmo 2 - Login

Algoritmo +Login()

procedure +Login()
Introduzir username e password
SE campos preenchidos e username e password introduzido = username e password da base de dados
Login
SE check box "Lembrar password?" estiver ativa
Inserir dados do utilizador na tabela Utilizador_Interno
FIM SE
FIM SE
end procedure

Algoritmo 3 – Inserir

Algoritmo +Inserir()

procedure +Inserir()
O sistema gera o ID automaticamente
Introduzir todos os campos obrigatórios
SE campos obrigatórios preenchidos
Inserir
FIM SE
end procedure

Algoritmo 4 - Alterar

Algoritmo +Alterar()

procedure +Alterar()
O sistema disponibiliza o registo onde o ID do registo = ID do registo selecionado
Editar os campos
SE campos obrigatórios preenchidos
Alterar
FIM SE
end procedure

Algoritmo 5 - Eliminar

Algoritmo +Eliminar()

procedure +Eliminar()
Sistema disponibiliza registo para eliminar onde o ID do registo = ID do registo selecionado
Confirmar
end procedure

Algoritmo 6 - AlterarQuantidade

Algoritmo +AtualizarQuantidade()

procedure AtualizarQuantidade()
Sistema disponibiliza o registo onde o ID do registo = ID do registo selecionado
SE categoria = Xarope
Quantidade_tomar = 15ml
SENÃO
quantidade_tomar = 1
FIM SE
end procedure

Algoritmo 7 - PodeIniciarPlano

Algoritmo + PodeIniciarPlano()

procedure PodeIniciarPlano()
medicamentos necessarios = (duracao / intervalo de tomas) * quantidade de medicamentos a tomar
SE quantidade de medicamentos a tomar < quantidade disponivel de medicamentos
false
SENÃO
true
FIM SE
end procedure

4.7.2. Base de dados Externa (MySQL)

Classe Utilizador

Tabela 12 - Semântica da classe Utilizador

Classe: Utilizador					
Atributo	**Tipo de Dados**	**Descrição**	**Valores válidos**	**Formato**	**Restrição**
Id_utilizador (PK)	Inteiro	Número sequencial que identifica a viatura	Maior que 0	Até 10 dígitos	Gerado pelo sistema, não alterável
nome	String	Nome do registo do utilizador	Caracteres de A a Z	Até 50 caracteres	Obrigatório
email	String	E-Mail de registo do utilizador	Caracteres de A a Z, números de 0 a 9 e caracteres especiais	Até 50 caracteres com, incluindo (.) e (@)	Obrigatório
username	String	Nome de utilizador registado	Caracteres de A a Z	Até 50 caracteres	Obrigatório
password	String	Palavra passe do utilizador	Caracteres de A a Z, números de 0 a 9 e caracteres especiais	Até 70 caracteres	Obrigatório

Operações da classe Utilizador

A tabela de operações descrever todas as operações (métodos) da classe Utilizador

Tabela 13 - Operações da classe "Utilizador"

Operações	
Nome	**Descrição**
Login()	Operação que permite iniciar sessão na aplicação
Registar()	Operação que permite registar-se na aplicação
Alterar()	Operação que permite alterar os seus dados
Eliminar()	Operação que permite eliminar o utilizador

Classe Medicamento_Utilizador

Tabela 14 - Semântica da classe "Medicamento_Utilizador"

Classe: Utilizador					
Atributo	**Tipo de Dados**	**Descrição**	**Valores válidos**	**Formato**	**Restrição**
Id_medicamento (PK)	Inteiro	Número sequencial que identifica a viatura	Maior que 0	Até 10 dígitos	Gerado pelo sistema, não alterável
Id_categoria (FK)	Inteiro	Número que identifica a categoria	Maior que 0	Até 10 dígitos	Obrigatório
Id_utilizador (FK)	Inteiro	Número que identifica o utilizador	Maior que 0	Até 10 dígitos	Obrigatório
nome	String	Nome do medicamento	Caracteres de A a Z e números de 0 a 9	Até 50 caracteres	Obrigatório
dosagem	Inteiro	Dosagem do medicamento	Maior que 0	Opção	Obrigatório
quantidade	Inteiro	Quantidade de medicamentos	Maior que 0	Opção	Obrigatório
data_validade	Data	Data de validade do medicamento	Dígitos separados por (-)	AAAA-MM-DD	Obrigatório

Operações da classe "Medicamento_Utilizador"

A tabela de operações descreve todas as operações (métodos) da classe "Medicamento_Utilizador".

Tabela 15 - Operações da classe "Medicamento_Utilizador"

Operações	
Nome	**Descrição**
Inserir()	Operação que permite inserir um novo medicamento 1) O sistema gera o ID automaticamente 2) Selecionar categoria 3) Sistema insere ID do utilizador que iniciou sessão 4) Inserir nome 5) Inserir dosagem 6) Selecionar data de validade
Alterar()	Operação que permite alterar um medicamento 1) Alterar categoria 2) Alterar nome 3) Alterar dosagem 4) Alterar data de validade
Eliminar()	Operação que permite eliminar um medicamento
AtualizarQuantidade()	Operação que permite atualizar a quantidade do medicamento cada vez que o utilizador o toma 1) Inserir quantidade a tomar 2) Atualizar medicamento

Classe Categoria

Tabela 16 - Semântica da classe "Categoria"

Classe: Utilizador					
Atributo	**Tipo de Dados**	**Descrição**	**Valores válidos**	**Formato**	**Restrição**
Id_categoria (PK)	Inteiro	Número sequencial que identifica a viatura	Maior que 0	Até 10 dígitos	Gerado pelo sistema, não alterável
descricao	String	Nome da categoria	Caracteres de A a Z	Até 50 caracteres	Obrigatório
Dosagem_unidade	String	Unidades da dosagem abreviadas (ex: mg/ml)	Caracteres de A a Z	Até 20 caracteres	Obrigatório
quantidade_unidade	String	Unidades da quantidade abreviadas (ex: ml)	Caracteres de A a Z	Até 20 caracteres	Obrigatório
quant_extenso_unidade	String	Unidades da quantidade por extenso (ex: mililitros)	Maior que 0	Opção	Obrigatório

4.7.3. Base de Dados Interna (SQLite)

Classe Utilizador_Interno

Tabela 17 - Diagrama de Sequência "Utilizador_Interno"

Classe: Utilizador					
Atributo	**Tipo de Dados**	**Descrição**	**Valores válidos**	**Formato**	**Restrição**
Id_utilizador_json	Inteiro	Número que identifica o utilizador da BD externa	Maior que 0	Até 10 dígitos	Obrigatório
nome	String	Nome do registo do utilizador	Caracteres de A a Z	Até 50 caracteres	Obrigatório
username	String	Nome de utilizador registado	Caracteres de A a Z	Até 50 caracteres	Obrigatório
password	String	Palavra passe do utilizador	Caracteres de A a Z, números de 0 a 9 e caracteres especiais	Até 70 caracteres	Obrigatório

Operações da Classe "Utilizador_Interno"

Tabela 18 - Operações da Classe "Utilizador_Interno"

Operações	
Nome	**Descrição**
inserir()	Operação que permite inserir um utilizador
eliminar()	Operação que permite eliminar um utilizador

Classe Plano_Tomas

Tabela 19 - Semântica da classe "Plano_Tomas"

Classe: Utilizador					
Atributo	**Tipo de Dados**	**Descrição**	**Valores válidos**	**Formato**	**Restrição**
Id_plano (PK)	Inteiro	Número sequencial que identifica a viatura	Maior que 0	Até 10 dígitos	Gerado pelo sistema, não alterável
Id_utilizador (FK)	Inteiro	Número que identifica o utilizador	Maior que 0	Até 10 dígitos	Obrigatório
Id_medicamento (FK)	Inteiro	Número que identifica o medicamento	Maior que 0	Até 10 dígitos	Obrigatório
data_inicio	Data	Data de início da toma	Dígitos separados por (-)	AAAA-MM-DD	Obrigatório
Intervalo_tomas	Inteiro	Intervalo entre cada toma	Maior que 0	Opção	Obrigatório
duração_tratamento	Inteiro	Duração total do plano	Maior que 0	Opção	Obrigatório
quantidade_tomar	Inteiro	Quantidade de unidades tomadas por toma	Maior que 0	Opção	Obrigatório

Operações da classe "Plano_Tomas"

A tabela de operações descreve todas as operações (métodos) da classe "Plano_Tomas".

Tabela 20 - Operações da classe "Plano de Tomas"

Operações	
Nome	**Descrição**
Inserir()	Operação que permite inserir um novo plano de tomas 1) O Sistema gera o ID automaticamente 2) Selecionar data de início 3) Sistema insere ID do utilizador que iniciou sessão 4) Selecionar medicamento 5) Inserir intervalo da toma 6) Inserir duração de tratamento 7) Inserir quantidade de unidades
Alterar()	Operação que permite alterar um plano de tomas 1) Alterar data de início 2) Alterar medicamento 3) Alterar intervalo da toma 4) Alterar duração do tratamento 5) Alterar quantidade de unidades
Eliminar()	Operação que permite eliminar um medicamento

4.8. Diagrama de Atividades

Um diagrama de Atividades descreve os aspetos dinâmicos do sistema. É um fluxograma para representar o fluxo de uma atividade para outra que pode ser uma operação do sistema.

Na Figura 14 está representado o diagrama de atividades da aplicação MediRemind.

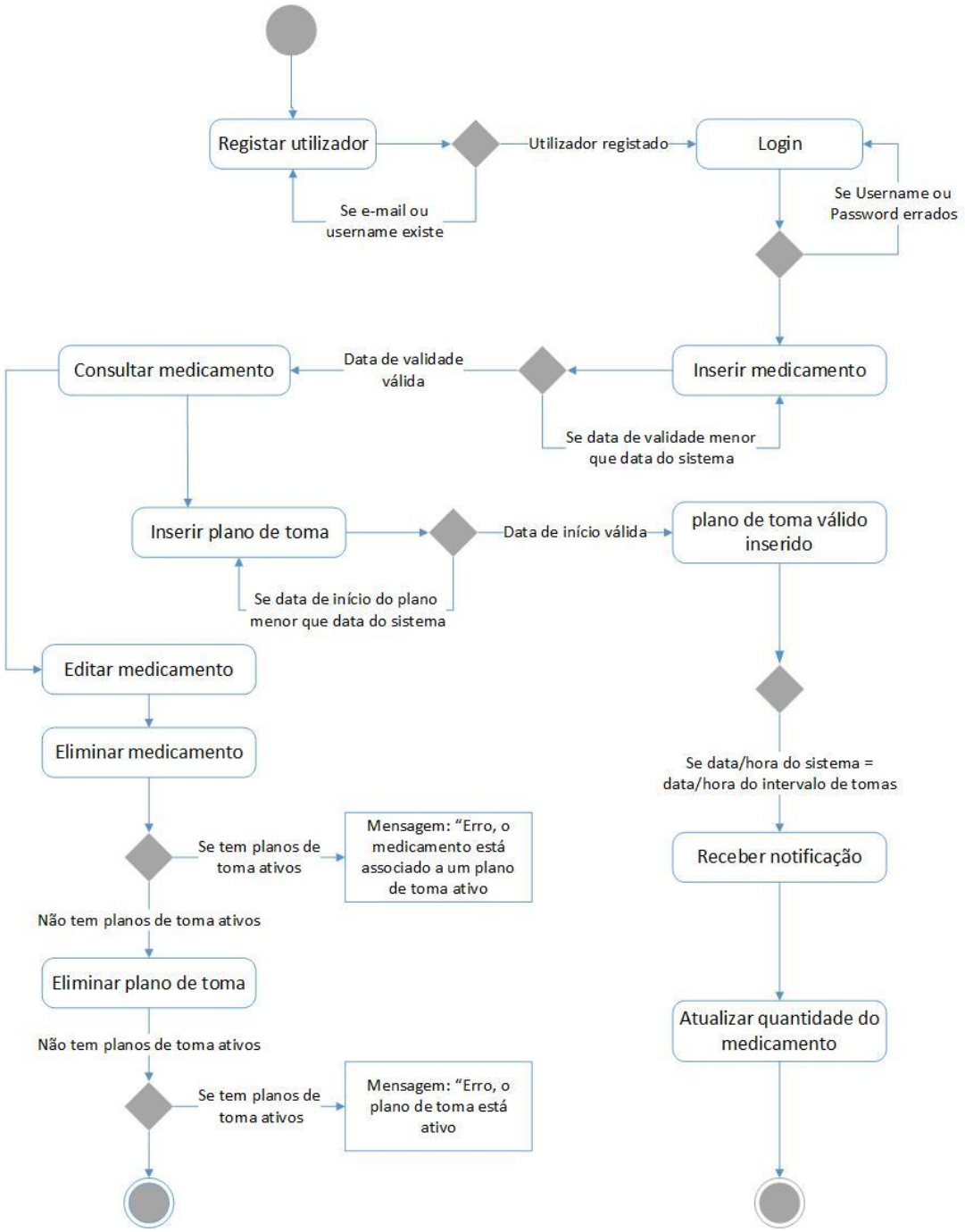


Figura 14 - Diagrama de Atividades da aplicação MediRemind

4.9. Diagrama de Componentes

Um diagrama de componentes mostra as partes do projeto de um sistema de *software*. Um diagrama de componentes ajuda a visualizar a estrutura do sistema por meio de interfaces.

O diagrama está dividido por packages, cada um com as suas funcionalidades. Quando o utilizador inicia a aplicação é mostrado o SplashScreenActivity.java, que vai verificar se existe algum utilizado na Base de Dados SQLite do *Android*. Se existir, inicia a HomeActivity.java, se não existir, inicia o LoginActivity.java para iniciar sessão ou registar-se.

Ao iniciar a HomeActivity.java, são apresentados os medicamentos do utilizador, através do *Fragment* MeusMedicamentosFragment.java, onde estão listados os medicamentos e onde recebe notificações quando estes estiverem a terminar. O mesmo acontece com o PlanosTomaFragment.java. Na base de dados SQLite é onde são guardados os planos de toma e, como citado anteriormente, onde fica guardado o utilizador que iniciou sessão caso o permita. Na base de Dados MySQL é onde são guardados os medicamentos.

Na Figura 15 está demonstrado o diagrama de componentes da aplicação MediRemind.

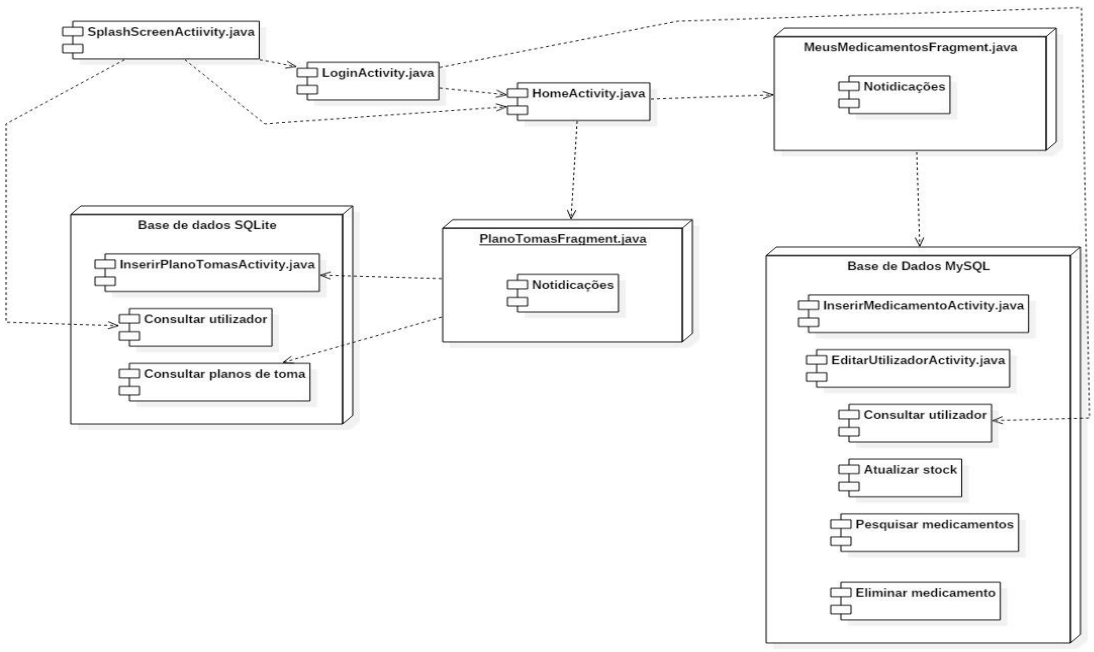


Figura 15 - Diagrama de Componentes

4.10. Diagrama de Instalação

O diagrama de instalação da Figura 16, mostra as ligações entre a aplicação mobile e o servidor através de Webservice, e serve para visualizar como tudo se interliga de modo a obter a aplicação final.

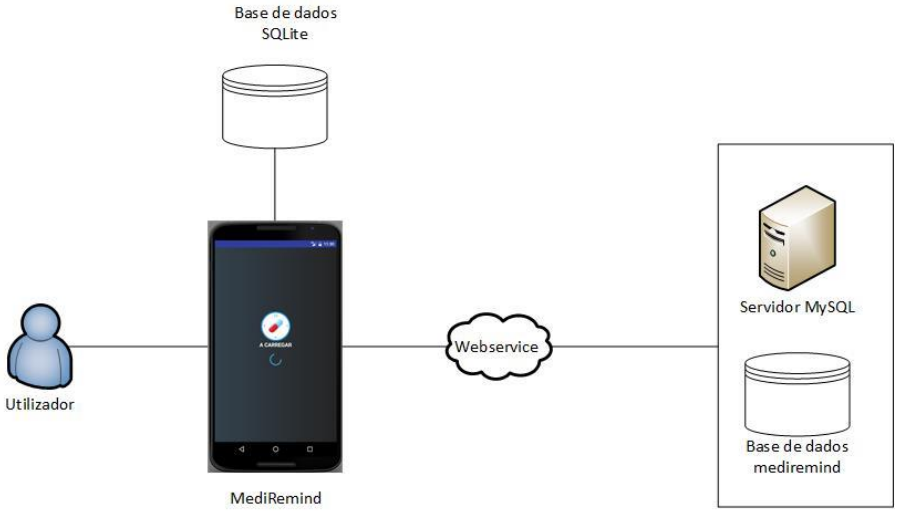


Figura 16 - Diagrama de Instalação da aplicação MediRemind

4.11. Diagrama de Instalação com Componentes

A Figura 17 apresenta o diagrama de instalação com componentes.

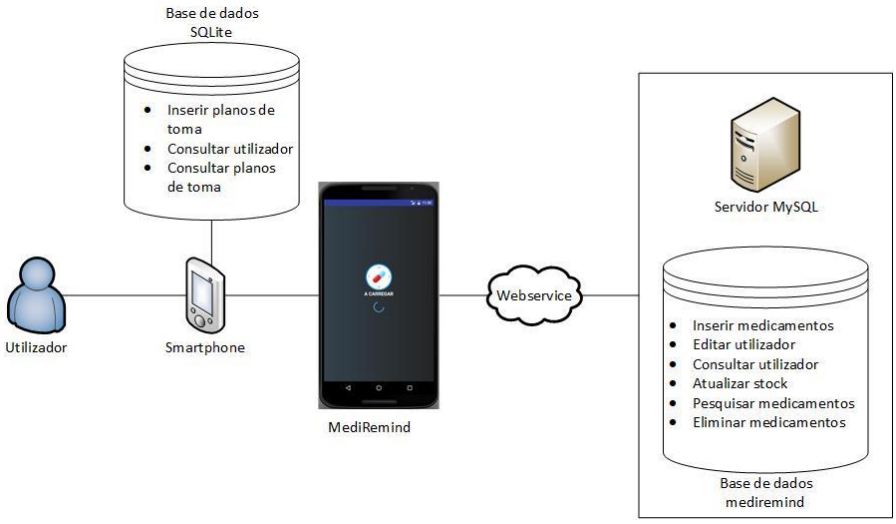


Figura 17 - Diagrama de Instalação com Componentes da aplicação MediRemind

5. Tecnologias e ferramentas utilizadas

Neste capítulo serão descritas todas as tecnologias de modelação e desenvolvimento utilizadas no decorrer do projeto.

Para a realização do relatório foram utilizadas três linguagens de programação:

- **Java –** para o desenvolvimento da aplicação mobile em Android.
- **PHP –** para o desenvolvimento do *Webservice*.
- **UML –** para o desenho dos diagramas.

5.1. Sistema de Versões e Configurações

Um controlo de versões é um sistema que regista as mudanças num ficheiro ou conjunto de ficheiros ao longo do tempo, para que se possam recuperar versões específicas.

Estes sistemas são utilizados, maioritariamente, em desenvolvimento de *software* para controlar as diferentes versões (histórico e desenvolvimento) do código-fonte (Wikipedia, 2016).

O sistema de versões e configurações utilizado no presente relatório foi o GIT com o GitHub. O GitHub é um repositório Web GIT que oferece um repositório com histórico completo e habilidade total de acompanhamento de versões. As funções da plataforma GitHub, as aplicações Desktop e o GitHub Enterprise, tornam a tarefa de escrever código muito mais fácil.

5.2. Sistema Operativo Android

O *Android* é constituído por um sistema operativo baseado no *kernel Linux*. Foi inicialmente desenvolvido por uma empresa designada Android Inc., por Rubin, Rico Miner e Nick Sears em 2003 e foi mais tarde adquirida pela Google, em 2005 (Hammerschmidt, 2016).

A Figura 18 mostra a evolução do *Android* ao longo dos anos até à versão 6.0, também conhecido como *Android* M (MarshMallow).

Figura 18 - Evolução do sistema operativo Android

Fonte: http://www.cubettech.com/blog/wp-content/uploads/2015/07/imgo.jpg

Atualmente, na versão 7.0 (*Android Nougat*), é utilizada maioritariamente em dispositivos móveis, continuando a crescer exponencialmente. Mas apesar da maioria dos dispositivos *Android* serem *Smartphones*, o sistema operativo está também presente em relógios (Android Wear), consolas, câmaras e televisões (Smart TV) (Wikipédia, 2016).

5.3. Linguagem Java

A linguagem de programação Java baseia-se numa Programação Orientada a Objetos (POO), criada pela empresa *Sun Mucrosystems* no início dos anos 90 e, anos mais tarde, foi vendida à empresa *Oracle*.

A empresa quis criar uma linguagem de programação que resolvesse problemas que havia com outras linguagens, como ponteiros, gestão de memória ou falta de bibliotecas. Alguns desses problemas foram explorados, porque uma das grandes motivações para

criar a linguagem Java era para que fosse usada em pequenos dispositivos (Caelum, 2016).

Na Figura 19 está representado um exemplo de código em Java que escreve no ecrã o texto “Olá mundo”.

```
1  //pacote "estudo";
2  package estudo;
3
4  //a classe "OlaMundo" pertence está no pacote "estudo"
5  public class OlaMundo {
6     //método principal - inicia a execução do aplicativo java
7     public static void main(String[] args){
8
9        //Exibe a frase "Olá Mundo!" no console.
10       System.out.println("Olá Mundo!");
11
12    }//fim do método principal
13
14 }// fim da classe "OlaMundo"
15
```

Figura 19 - Exemplo de código em Java

5.4. SQLite

O SQLite é um Sistema de Gestão de Base de Dados (SGBD) relacional, contido numa biblioteca em linguagem C que implementa uma base de dados SQL (*Structure Query Language)* embutida no programa (PHP Group, 2016).

Ao contrário da Base de Dados (BD) comum, o SQLite não é uma Base de Dados cliente-servidor e, como funciona diretamente no programa, não necessita de da execução de uma DMBS (*Database Management System*) separadamente.

5.5. Linguagem PHP

PHP (*PHP: Hypertext Preprocessor)* é uma linguagem script embutida no HTML (*HyperText Markup Language*) interpretada do lado do servidor de código aberto, capaz de gerar código dinâmico e muito utilizada no desenvolvimento *Web* (PHP Group, 2016).

A linguagem PHP foi utilizada para programar a aplicação MediRemind do lado do servidor, que faz consultas e inserção dos dados da aplicação *Mobile* na Base de Dados através de JSON.

Existem várias linguagens de programação para a *Web*, no entanto foi escolhida a linguagem PHP pelas seguintes razões:

- **Open Source:** o PHP é completamente gratuito e, sendo gratuito, é mais fácil encontrar informação e ajuda na resolução de problemas com o código.
- **Facilidade na aprendizagem:** o PHP tem vários elementos de linguagens Java e C.

5.6. Webservice

Um *Webservice* é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Esta tecnologia usa um sistema de mensagens XML ou JSON padronizado, que são usadas para codificar todas as informações a um serviço Web. Por exemplo, um cliente invoca um *Webservice*, enviando uma mensagem em XML, em seguida, aguarda uma resposta também em XML (Tutorialspoint, 2016). Como toda a comunicação está na mesma linguagem, diferentes plataformas podem comunicar entre si.

Características:

- Interação entre novas e antigas aplicações
- Compatibilidade de plataformas
- Agilidade de processos
- Segurança

5.7. HTTP

O HTTP (Hypertext Transfer Protocol) é um protocolo de comunicação entre sistemas de informação que permite a transferência de dados entre redes de computadores.

O HTTP funciona como um protocolo pedido-resposta. Um cliente (Browser por exemplo) submete um pedido de mensagens HTTP para o servidor e o servidor devolve uma mensagem de resposta para o cliente.

5.8. URI

O URI (Uniform Resource Identifier) é uma sequência de caracteres que identifica um recurso físico ou lógico. Existem várias especificações de esquemas que ditam como a identificação é feita (TechTarget, 2016). O URI normalmente descreve:

- O mecanismo usado para aceder ao recurso.
- O computador específico em que o recurso está alojado.
- O nome específico do recurso.

O URI é muito utilizado para especificar um recurso num URL (Uniform Resource Locator), que não é mais que um endereço *Web*.

Por exemplo, o URI "http://www.medicamentos.pt/comprimidos?id=13", identifica que este medicamento pode ser acedido usando o protocolo HTTP, porque começa por "http://", que é usado num servidor chamado "medicamentos.pt", está localizado em "comprimidos" e é especificado que é o comprimido com o id 13.

5.9. REST

O REST (Representation State Transfer) foi criado em 2000, através de uma tese de doutoramento, por Roy Fielding, que é um dos criadores do protocolo HTTP. É um estilo arquitetural que consiste num conjunto de restrições arquiteturais aplicadas a componentes, conetores e elementos de dados distribuído (Wikipedia, 2016).

O REST é frequentemente aplicado a *Webservices*, fornecendo API's para acesso a um servidor *Web*. Ele usa integralmente as mensagens HTTP para se comunicar através do protocolo já definido.

Principais características:

- Usa o protocolo HTTP (Accept headers, Content-Types,…) de forma explícita e representativa para se comunicar e utiliza uma notação comum para transferência de dados, como XML ou JSON.
- É *Stateless*, ou seja, não guarda o estado entre comunicações, cada comunicação é independente e uniforme, precisando por isso passar toda a informação necessária em cada pedido.

RESTfull

Ao ser criado um *Webservice* na arquitetura REST, a sua designação passa a ser RESTfull, que não é mais que usar o protocolo HTTP com os métodos GET e POST.

5.10. JSON

Um ficheiro JSON (JavaScript Object Notation) é um formato de ficheiro leve para troca entre dados computacionais. O JSON é um conjunto da notação do objeto de JavaScript, mas o seu uso não requer exclusivamente JavaScript (Wikipedia, JSON, 2016).

Devido à sua simplicidade, o JSON está a evoluir e o seu uso tem sido cada vez mais frequente, especialmente por alternativa ao XML (eXtensible Markup Language), e é mais usado em ambientes onde o tamanho de fluxo de dados é importante, como por exemplo em dispositivos móveis.

As imagens seguintes (Figura 20 e Figura 21, respetivamente) demonstram como seria representada uma pessoa com nome e idade, nos formatos JSON e XML.

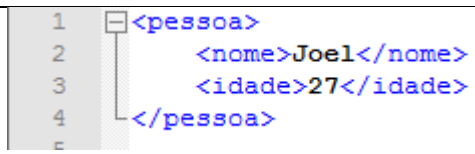


Figura 20 - Exemplo de um ficheiro XML

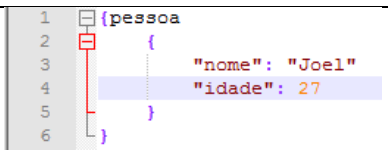


Figura 21 - Exemplo de um ficheiro JSON

5.11. PHPMailer

O PHPMailer é uma classe pronta para envio de E-Mails através de PHP por via SMTP (Simple Mail Transfer Protocol) e POP3 (Post Office Protocol) muito utilizada em todo o mundo (Enviar e-mails pelo PHP usando o PHPMailer, 2016). O seu método de envio é muito melhor que o mail(), a função padrão do PHP para enviar E-Mails.

Em seguida está um exemplo de como se envia um e-mail pelo PHPMailer.

```
$mail = new PHPMailer();
//$mail->SMTPDebug = 3;                               // Usar debug output
$mail->isSMTP();                                      // Usar SMTP
$mail->Username = 'user@example.com';                 // Username do servidor SMTP (e-mail)
$mail->Password = 'secret';                           // Password do servidor SMTP
$mail->SMTPSecure = 'tls';                            // Enable TLS encryption, `ssl` also accepted
$mail->Port = 587;                                    // Porta TCP

$mail->setFrom('from@example.com', 'Mailer');         // E-mail do remetente
$mail->addAddress('joe@example.net', 'Joe User');     // Add a recipient
$mail->addAddress('ellen@example.com');               // Name is optional
$mail->addReplyTo('info@example.com', 'Information'); // Responder para

$mail->isHTML(true);                                  // Enviar e-mail em formato HTML

$mail->Subject = 'Here is the subject';
$mail->Body    = 'This is the HTML message body <b>in bold!</b>';
$mail->AltBody = 'This is the body in plain text for non-HTML mail clients';

if(!$mail->send()) {
    echo 'Message could not be sent.';
    echo 'Mailer Error: ' . $mail->ErrorInfo;
} else {
    echo 'Message has been sent';
}
```

5.12. PDO

PDO (PHP Data Object) é uma classe que fornece uma interface padronizada para trabalhar com base de dados, cuja finalidade é abstrair a ligação e interações com a mesma. Ou seja, independentemente da base de dados que estiver a ser utilizada, os métodos serão os mesmos.

Vantagens:

- Abstração da ligação e interação com a base de dados.
- Segurança
- Suporte a diversos drivers.

Exemplo de uma ligação com a base de dados MySQL:

```
$conn = new PDO('mysql:host=localhost;dbname=example-pdo', 'joel', '123456');
```

5.13. MySQL

O MySQL é uma Sistema de Gestão de Base de Dados (SGBD) que utiliza a linguagem SQL como interface de acesso aos dados. É gratuito e tem grande desempenho e estabilidade. Atualmente, é uma das Bases de Dados mais utilizadas no mundo (Wikipedia, 2016).

O MySQL possui também grande compatibilidade com o PHP, tendo este, funções próprias para se ligar facilmente à base de dados. Para além de garantir um nível de segurança mais alto que muitos outros Sistemas de Gestão de Base de Dados, possui também uma maior eficiência a programar.

A Figura 22 demonstra um exemplo de ligação à Base de Dados MySQL através do PHP.

```php
<?php
$conecta = mysqli_connect("HOST", "LOGIN", "SENHA") or print (mysql_error());
print "Conexão OK!";
mysqli_close($conecta);
?>
```

Figura 22 - Ligação à Base de Dados MySQL através do PHP

5.14. Software utilizado

Neste capítulo será apresentado todo o *software* utilizado para o desenvolvimento da aplicação MediRemind.

Android Studio

O Android Studio é um IDE (Integrated Development Enviromnent), baseado na plataforma JetBrains Intellij IDEA[8], dedicado à programação em *Android*, tornando-se no único *software* presente no mercado para esse efeito, visto que o Eclipse, que era uma alternativa ao Android Studio, se tornou obsoleto.

Na Figura 23, está demonstrado o ícone do Android Studio.

Figura 23 - Ícone Android Studio

Fonte: https://commons.wikimedia.org/wiki/File:Android_Studio_icon.svg

PHP Storm

O PHP Storm é um IDE baseado também na plataforma JetBrains Intellij IDEA, dedicado à programação em PHP, podendo também utilizador HTML ou JavaScript, com análise de código e prevenção de erros.

Na Figura 24, está demonstrado o ícone do PHPStorm.

Figura 24 - Ícone PHPStorm

Fonte: https://pbs.twimg.com/profile_images/674918306000711680/3rPeiTKt.png

[8] https://en.wikipedia.org/wiki/IntelliJ_IDEA

StarUML

StarUML é um software de modelação de sistemas que suporta a tecnologia UML (Unified Modeling Language), é baseado em notações UML e permite a conceção de 11 tipos de diagramas.

É software que permite a personalização do ambiente de trabalho e tem uma alta capacidade de extensão das suas funcionalidades oferecendo um grande número de variáveis de personalização que permite ao utilizador um desenvolvimento metódico.

Proporciona também uma extensibilidade e flexibilidade fornecendo estruturas que estendem para lá das próprias ferramentas.
Foi projetado para permitir o acesso de todas as funções de modelagem e fornece extensões de menus personalizados. Alem disso os utilizadores podem criar as suas próprias abordagens de acordo com as suas metodologias.

Na Figura 25, está demonstrado o ícone do StarUML.

Figura 25 - Ícone StarUML

Fonte: https://upload.wikimedia.org/wikipedia/en/7/71/StarUML_logo.png

6. Implementação da solução

Neste capítulo serão apresentados e explicados todos os passos necessários para o desenvolvimento da plataforma MediRemind. A plataforma está dividida em duas partes, a parte do servidor, onde se se encontra o *Webservice* que faz a consulta e inserção dos dados na Base de Dados MySQL, e a parte da aplicação móvel em *Android*, onde há a interação do utilizador com a aplicação.

Para o desenvolvimento da aplicação, foram utilizados vários componentes do *Android*. Em seguida serão indicados e explicados cada um deles.

Activity

Uma *Activity* (ou Atividade) é uma coisa única focada no que o utilizador pode fazer. Quase todas as Atividades interagem com o utilizador, normalmente uma classe Atividade tem um ficheiro XML associado, para criação da Interface do Utilizador (UI). Embora as Atividades sejam apresentadas ao utilizador em modo de ecrã inteiro, elas também podem ser usadas na forma de janela flutuante (Android Developer, 2016).

Numa classe que estende a classe Activity, é necessário implementar dois métodos:

- **onCreate(Bundle):** este método é chamado quando a Activity é criada. Neste método é onde se indica o ficheiro XML da Activity, que tem o *layout* com o método setContentView(int), e se usa o findViewById(int) para recuperar os ID's dos *Widgets* utilizados no *layout*.

Ciclo de vida de uma *Activity*

A responsabilidade do ciclo de vida de uma aplicação é do Sistema Operativo. Quando uma *Activity* está a ser executada, vai para o topo da pilha (*stack*) e torna-se na *Activity* principal. A *Activity* anterior permanece sempre em baixo na pilha e não é mostrada ao utilizador até a *Activity* principal estiver em execução. (Nakahara, 2016)

Uma *Activity* pode assumir quatro estados:

- Em execução
- Interrompida

- Em segundo plano
- Destruída

No momento em que uma *Activity* é interrompida ou passa para segundo plano, o sistema, consoante as necessidades, pode destruí-la para libertar memória do sistema. Nestes estados, pode ser importante guardar algumas informações da aplicação para que seja para que, quando a aplicação passar para o estado "Em execução", voltar no mesmo ponto em que foi deixada.

Na Figura 26 é mostrado todo o movimento através do ciclo de vida das Atividades.

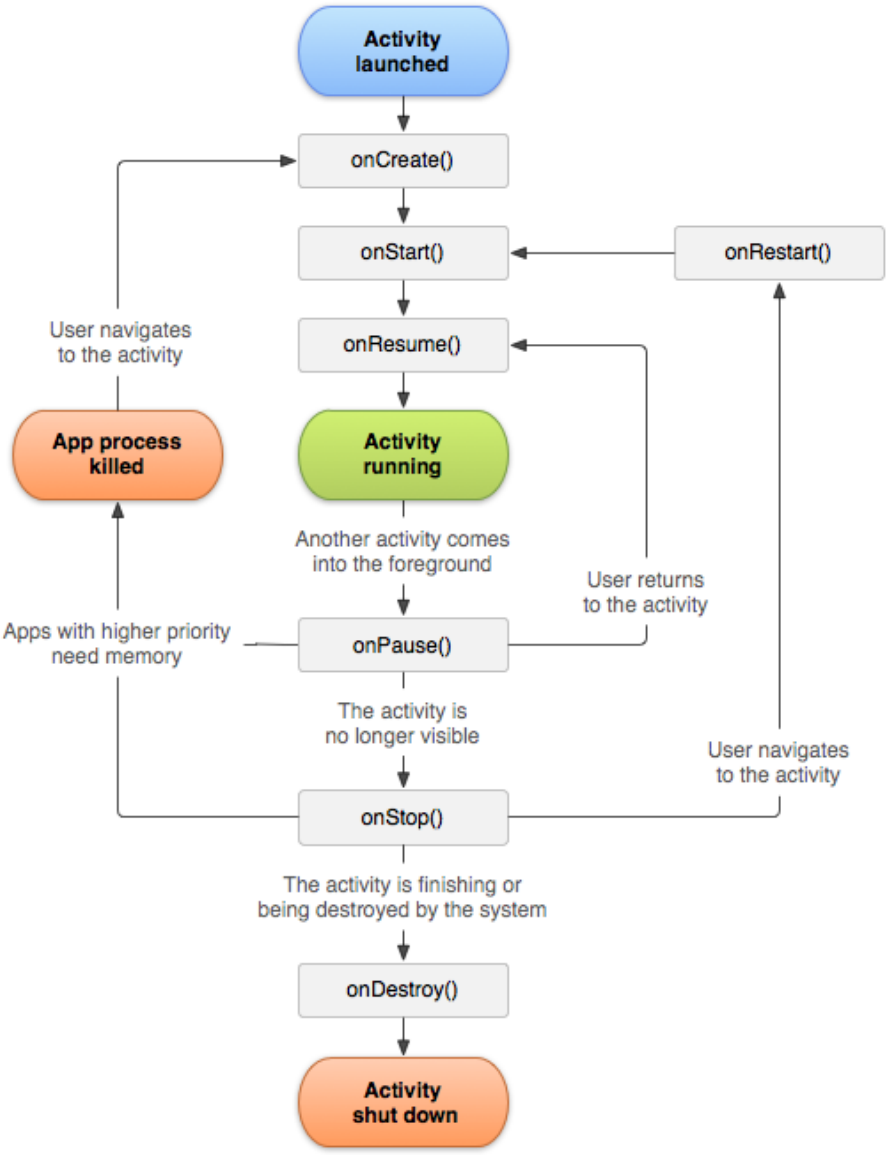


Figura 26 - Ciclo de vida de uma Activity

Fonte: http://www.javatpoint.com/images/androidimages/Android-Activity-Lifecycle.png

Na Tabela 21 é explicada a transição entre o ciclo de vida das Atividades.

Tabela 21 - Ciclo de vida de uma Activity

Método	Descrição	Próximo método
onCreate()	É chamado apenas uma vez quando a aplicação é criada. Após a sua execução é chamado o método onStart().	**onStart()**
onStrart()	Pode ser executado após o onCreate() ou onRestart(). Após a sua execução, pode ser chamado o método onResume() ou onStop().	**onResume() / onStop()**
onRestart()	Este método é chamado quando a *Activity* sai do estado "interrompida" ou "em segundo plano". É sempre seguida do onStart().	**onStart()**
onResume()	Chamado quando a *Activity* inicia a interação com o utilizador.	**onPause()**
onPause()	É executado quando a *Activity* é interrompida. Aqui deve ser guardado o estado da aplicação para que seja possível uma recuperação pelo método onResume(). Se seguir para o onStop(), fica invisível ao utilizador.	**onResume() / onStop()**
onStop()	É chamado quando a *Activity* está a ser terminada. Isto acontece quando outra *Activity* está a ser executada. Seguida do onRestart() se voltar a interagir com o utilizador, ou seguida do onDestroy() se estiver a ser destruída.	**onRestart() / onDestroy**
onDestroy()	O último método chamado antes da *Activity* ser encerrada. Isto pode acontecer se estiver a ser encerrada pelo utilizador, chamando o método finish(), ou porque o sistema está com falta de memória e precisa de ser encerrada.	**Nada**

Fragment

Um *Fragment* representa um comportamento ou uma parte da interface do utilizador numa *Activity*.

Numa única *Activity* podem ser combinados vários *Fragments* para construir uma interface de utilizador com múltiplos painéis e reutilizar um *Fragment* em múltiplas *Activities*.

Pode-se pensar num *Fragment* como uma secção modular de uma *Activity* que tem o seu próprio ciclo de vida, recebe os seus próprios eventos de entrada e que se pode adicionar ou remover enquanto a *Activity* está em execução, como uma espécie de sub-Activity que se pode reutilizar em diferentes *Activities*.

Um *Fragment* deve ser sempre implementado dentro de uma *Activity* e o ciclo de vida do *Fragment* é diretamente afetado pelo ciclo de vida da *Activity* "mãe". Por exemplo, quando uma *Activity* estiver no estado "interrompida", os *Fragments* que estiverem associados, também ficarão no mesmo estado, e se a *Activity* fot destruída, todos os *Fragments* associados também serão destruídos.

No entanto, quando uma *Activity* está em execução, os *Fragments* podem ser manipulados de forma independente, tais como adicionar ou remover (Android Developer, 2016).

Ciclo de vida de um *Fragment*

Para criar um *Fragment* é necessário criar uma classe que estende a classe *Fragment*. O código da classe criada é muito parecido com uma *Activity*, o *Fragment* contém métodos como onCreate(), onStrart(), onPause() e onStop().

Normalmente, num *Fragment*, é necessário implementar pelo menos os seguintes métodos:

- **onCreate():** o sistema chama este método quando o *Fragment* é criado. Dentro da aplicação, devem-se iniciar os componentes importantes quando o *Fragment* é interrompido ou parado, depois de ser resumido.

- **onCreateView():** este método é chamado quando o *Fragment* chama a interface de utilizador pela primeira vez. O retorno deste método deve ser a raiz do layout. Aqui também são inicializados os *Widgets* do layout.
- **onPause():** este método é chamado quando há a indicação que o utilizador está a deixar o *Fragment*, ou seja, quando o *Fragment* é interrompido. Aqui deve ser guardado o estado da aplicação para que possível uma recuperação no método onResume().

  Na Figura 27, está demonstrado o ciclo de vida de um *Fragment*.

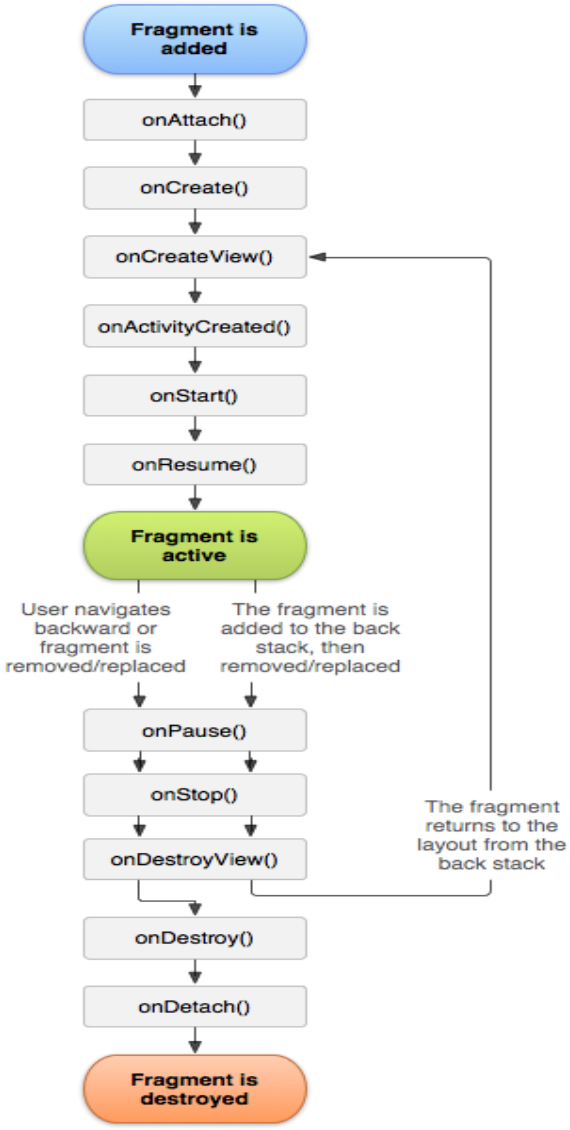


Figura 27 - Ciclo de vida de um Fragment

Fonte: https://developer.android.com/guide/components/fragments.html

View e ViewGroup

As classes *View* e *ViewGroup* são classes centrais em aplicações *Android*, porque são classes que estão por detrás de uma *Activity*, ou seja, são os componentes de Interface do utilizador (Jenkov, 2016).

View

A classe *View* é uma superclasse para todos os componentes de UI (Interface de Utilizador) em *Android* que responde à interação do utilizador.

Exemplos de *Views*:

- *TextView*
- *EditText*
- *ImageView*
- *ProgressBar*
- *Button*
- …

ViewGroup

ViewGroups são contentores invisíveis que contêm *Views* e *ViewGroups*. É a classe principal num *layout*, já que todos os componentes têm que ter, pelo menos, um *ViewGroup*.

Exemplos de *ViewGroups*:

- *LinearLayout*
- *RelativeLayout*
- *GridView*
- *ListView*
- *RecyclerView*
- …

Na Figura 28, está demonstrada a estrutura de *Views* e *ViewGroups*.

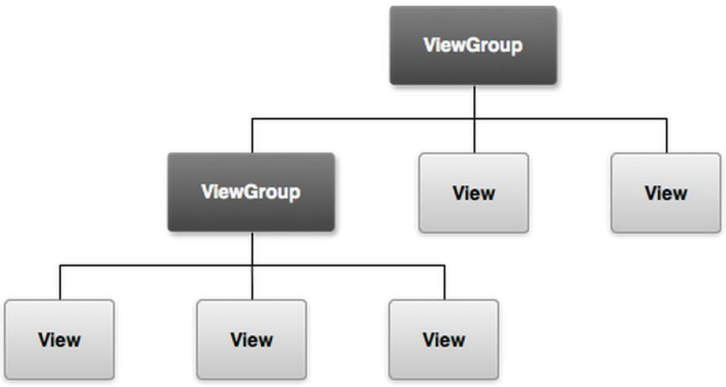


Figura 28 - Estrutura de Views e Viewgroups

Fonte: https://i.stack.imgur.com/KSCNf.png

Como se pode ver na figura acima, um *ViewGroup* pode conter *Views*, mas também pode ter outros *ViewGroups* que por sua vez têm mais *Views* e mais *ViewGroups*.

RecyclerView

O *Widget* RecyclerView é uma versão mais avançada e flexível do ListView. Este *Widget* é um recipiente que serve para exibir um grande conjunto de dados de forma mais eficiente, mantendo um número limite de *views*. Usa-se o RecyclerView quando há uma lista de dados, cujos elementos mudam em tempo real, baseado em eventos do utilizador ou de rede (ex. Base de Dados).

A classe RecyclerView Simplifica a exibição e manipulação de dados, fornecendo *Layout managers* para posicionar os itens e animações por defeito para as operações mais comuns, tais como remover ou adicionar itens.

A Figura 29, mostra funcionamento do RecyclerView.

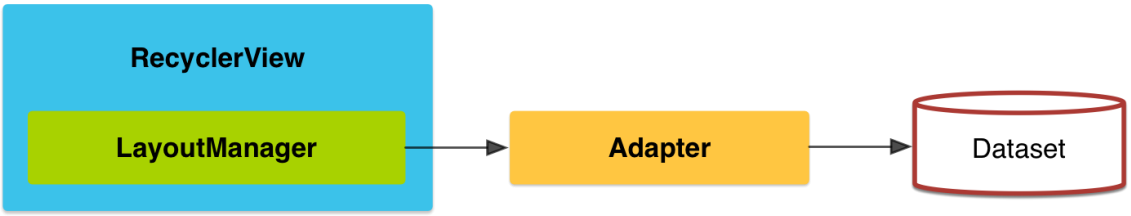


Figura 29 – RecyclerView

Fonte: https://developer.android.com/training/material/images/RecyclerView.png

Como mostra a figura acima, o RecyclerView possui um LayoutManager que determina quando reutilizar um item *View* que já não está visível para o utilizador. Para reutilizar um *View*, o *LayoutManager* pode pedir ao *Adapter* para substituir o conteído da *View* por um elemento diferente do *Dataset* (Android Developer, 2016).

O RecyclerView possui dois tipos de *LayoutManager* por defeito:

- **LinearLayoutManager:** mostra os itens na vertical ou na horizontal
- **GridLayoutManager:** mostra os itens numa Grid (grella)

Dialog

Em *Android*, um *Dialog* é uma pequena janela que pede ao utilizador para tomar uma decisão ou para mostrar alguma informação adicional. Um *Dialog* não preenche totalmente o ecrã e é utilizador, normalmente, por eventos pontuais, que exige ao utilizador tomar alguma decisão antes de prosseguir (Android Developer, 2016).

Na Figura 30 está um exemplo de dois *Dialog*.

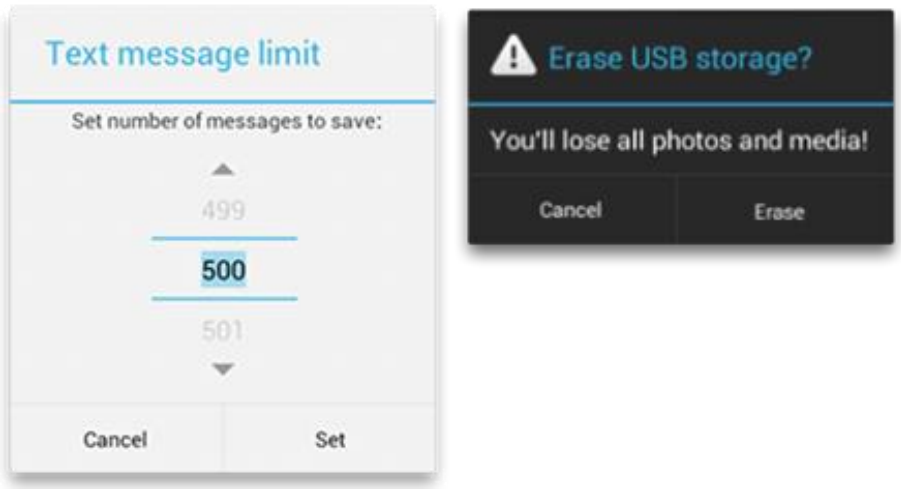


Figura 30 – Dialog

Fonte: https://developer.android.com/guide/topics/ui/dialogs.html

Notification

O sistema de notificações do *Android* é uma maneira de informar os eventos que acontecem. A interface permite visualizar e interagir com a aplicação que gerou a notificação.

Há duas classes para gerar notificações em *Android*:

- **NotificationCompact:** para versões do *Android* a partir do API 4 (versão 1.6).
- **Notification:** para versões do *Android* a partir da API 11 (versão 3.0).

Para criar uma notificação é importante definir um ícone, um título e uma descrição da notificação e a ação é definida através da classe PendingIntent que devolve um objeto Intent que pode iniciar uma *Activity*.

Em seguida está um exemplo de criação de uma notificação.

```
// Cria a notificação com as informações de
   // ícone, título, prioridade e texto
   Notification.Builder mBuilder =
         new Notification.Builder(ctx)
         .setSmallIcon(R.drawable.ic_notification_romar)
         .setContentTitle("Modo avião")
         .setPriority(Notification.PRIORITY_DEFAULT)
         .setContentText("Foi alterado o modo de ligação do " +
            "telemóvel.");

   // Cria o intent que vai ser chamada quando a notificação
   // for selecionada. Neste caso exibe a Activity PrincipalActivity
   Intent resultIntent = new Intent(ctx, PrincipalActivity.class);

   // Cria uma pilha de tarefas para garantir que a navegação a
   // partir da Activity chamada pela notificação, retorne para
   // a tela principal do Android, quando selecionado o botão voltar
   TaskStackBuilder stackBuilder = TaskStackBuilder.create(ctx);
   stackBuilder.addParentStack(PrincipalActivity.class);
   stackBuilder.addNextIntent(resultIntent);
   PendingIntent resultPendingIntent =
         stackBuilder.getPendingIntent(
            0,
            PendingIntent.FLAG_UPDATE_CURRENT
         );
   mBuilder.setContentIntent(resultPendingIntent);
   NotificationManager mNotificationManager =
      (NotificationManager) ctx.getSystemService
      (Context.NOTIFICATION_SERVICE);

   // A constante ALTEROU_MODO_CONEXAO é uma identificação
   // que pode ser utilizada para atualizar a notificação posteriormente
   mNotificationManager.notify(ALTEROU_MODO_CONEXAO, mBuilder.build());
```

Broadcast Reciever

Um BroadCast Reciever é um componente do *Android* que permite registar eventos do sistema ou aplicações. Todos os recievers registados para um evento são notidicados pelo

Android em run time, uma vez que este evento acontece. Por exemplo, a aplicação pode registar-se para eventos do sistema ACTION_BOOT_COMPLETED, que é disparado quando o *Android* é reiniciado.

Thread

Nos dispositivos móveis, as tarefas podem ser executadas simultaneamente de duas formas: pelo Paralelismo ou pelo Pseudo-paralelismo.

No Paralelismo, a execução de tarefas depende do número de núcleos que o processador possui. Poe exemplo, se o processador tiver dois núcleos, só pode realizar duas tarefas simultaneamente e se só tiver um núcleo, só pode executar uma tarefa em simultâneo, o que é inviável.

Para resolver o problema do Paralelismo, nasceu o Pseudo-Paralelismo, que é um sistema interno que permite realizar várias tarefas em curtos espaços de tempo, ou seja, a tarefa é executada por tempo determinado. Acabando esse tempo, ocorre o "time-slice", que retira o processo de execução e coloca-o no final da fila de processos que estão prontos para serem executados (Romanato, 2016).

O tempo em que o time-slice acontece é impercetível ao utilizador, o que dá a sensação de paralelismo, mesmo quando o processador possui apensa um núcleo.

Quando uma aplicação é iniciada, ela funciona na Main Thread (Thread principal), também conhecida como UI Thread (User Interface Thread). Nalguns casos, é necessário executar processamentos demorados e que sejam independentes da interface do utilizador, pela simples razão de que a aplicação não pode ficar "parada" quando é necessário um determinado tempo para executar uma tarefa.

Para resolver isso, existe no *Android* a classe *Thread*, que não é mais do que uma tarefa que é executada em segundo plano, paralelamente à Main Thread. Depois de executada, devolve os dados à Thread principal, que neste caso é a interface do utilizador (Lagvankar, 2016).

Handler

A classe Handler permite enviar e processar mensagens e objetos associados à fila de uma Thread. Cada instância do Handler está associada a uma única linha e fila de mensagens da Thread (Android Developer, 2016).

AsynkTask

Em *Android* podem ser executadas várias tarefas em simultâneo, utilizando *Threads*, que permitem a execução de várias instruções em paralelo. Uma maneira de fazer isso é utilizando a classe Handler, mas para atualizar a interface gráfica com as informações processadas é um pouco complicado e demorado.

Para facilitar o processamento de dados em segundo plano e atualizar a interface do utilizador, foi criada a classe AsynkTask. Esta classe, encapsula todo o processo de Threads e Handlers (Android Developer, 2016).

Para se utilizar, é necessário criar uma classe que estende a classe AsynkTask. Ao criar a classe, é necessário implementar um método obrigatório e três opcionais:

- **onPreExecute():** este método é executado sempre antes da Thread ser iniciada e é onde são mostradas as mensagens no ecrã do utilizador. Normalmente utiliza-se uma barra de progresso para demonstrar que está a acontecer alguma coisa. O método não é obrigatório e é executado na mesma Thread que a interface gráfica.
- **doInBackground():** Este é o único método obrigatório e é responsável pelo processamento pesado, visto que é executado numa Thread separada. Enquanto está a ser processado, o ecrã não fica "bloqueado", mostrando ao utilizador as mensagens colocadas no onPreExecute(). O retorno do método é passado por parâmetro para o método onPostExecute().
- **onPostExecute():** Este método é o que recebe por parâmetro o retorno do doInBackground() e é chamado utilizando um Handler. O método é executado na mesma Thread que a interface gráfica e a sua utilização não é obrigatória.
- **onProgressUpdate():** este método só costuma ser implementado para fazer download de algum ficheiro da Internet e é responsável por receber as informações para mostrar a percentagem de download no ecrã do utilizador. Também não é obrigatório e é executado na mesma Thread que a interface gráfica.

Na Figura 31 está demonstrado um exemplo de uma classe que estende a classe AsynkTask.

```
21  public class MainActivity extends AppCompatActivity {
22
23      @Override
24      protected void onCreate(Bundle savedInstanceState) {
25          super.onCreate(savedInstanceState);
26          setContentView(R.layout.activity_main);
27
28          // Ao criar a Activity, é executada a classe TaskCalculadora para somar dois números numa nova thread,
29          // que recebe como parâmetros dois números inteiros
30          new TaskCalculadora().execute(2, 4);
31      }
32
33      // Exemplo de uma calculadora com AsynkTask
34      private class TaskCalculadora extends AsyncTask<Integer, Integer, Integer>{ // <Parâmetro, Progresso, Resultado>
35
36          @Override
37          protected void onPreExecute() {
38              // Neste método normalmente é mostrado um progressBar
39          }
40
41          @Override
42          protected Integer doInBackground(Integer... params) {
43              // Este método tem como parâmetro Integers que são os números que o
44              // utilizador escolhe para fazer a conta neste exemplo
45              int num1 = params[0];
46              int num2 = params[1];
47
48              // o método publishProgress serve para atualizar o valor de i que é
49              // o parâmetro do método onProgressUpdate
50              int i = 0;
51              publishProgress(i);
52
53              // Este método devolve um Integer para o método onPostExecute para
54              // mostrar o resultado ao utilizador
55              return num1 + num2;
56          }
57
58          @Override
59          protected void onProgressUpdate(Integer... values) {
60              super.onProgressUpdate(values);
61          }
62
63          // Este método recebe como parâmetro o retorno do doInBackground, que
64          // neste caso é um Integer, e apresenta-o ao utilizador
65          @Override
66          protected void onPostExecute(Integer integer) {
67              super.onPostExecute(integer);
68          }
69      }
70  }
```

Figura 31 - Exemplo de AsynkTask

Como se pode ver na figura acima, a *Activity* MainActivity tem uma classe "TaskCaluladora" que estende a classe AsynkTask. Este pequeno exemplo, recebe dois números como parâmetro e calcula a soma entre eles numa nova thread.

A classe AsynkTask tem três tipos, como demonstrados na linha 34 da figura acima. Neste exemplo, todos os tipos são Integer, porque o AsynkTask recebe dois números inteiros como parâmetro, o progresso é de um em um e o resultado da conta também é um inteiro.

- **Parâmetro:** são variáveis de qualquer valor que serão passadas no método doInBackground(). No caso do exemplo acima, o parâmetro é um conjunto de Integers.
- **Progresso:** é o tipo de dados que mostra o progresso no ecrã. Normalmente é utilizado um Integer para mostrar a percentagem do progresso (Por exemplo, pode ser usado para mostrar a percentagem de download de um ficheiro).
- **Resultado:** é o retorno da Thread de processamento para a Thread principal que será usado para ser mostrado na interface do utilizador. Neste exemplo, é um Integer com o resultado da soma dos dois números do parâmetro.

6.1. Aplicação *Android*

Na aplicação *Android* é onde é feita toda a interação com o utilizador. Neste capítulo será explicada cada Activity individualmente, usada para o desenvolvimento da aplicação.

Estrutura da aplicação

Uma aplicação *Android* é estruturada por três pastas principais: manifest, java e res.

A pasta manifest contém o ficheiro AndroidManifest.xml, que contém todas as definições e configurações da aplicação.

A pasta java contém todos os ficheiros java, sejam *Activities*, *Fragments* ou simples classes. Na presente aplicação, o conteúdo da pasta java está estruturada por *packages* para ser mais fácil a sua organização.

A pasta res contém todos os ficheiros de design, sejam layouts das *Activities*, imagens ou animações, ou seja, são todos os ficheiros XML.

6.1.1. Ficheiro *Manifest*

O *Manifest* é um ficheiro no formato XML que fornece todas as informações essenciais que o sistema deve ter sobre a aplicação, antes de executar qualquer código (Android Developer, 2016).

Além da declaração das Activities, no ficheiro *Manifest* da aplicação MediRemind foi necessário declarar permissões necessárias ao correto funcionamento da mesma.

A presente aplicação necessita de cinco permissões, como demonstrado na Figura 32.

```
6     <uses-permission android:name="android.permission.INTERNET" />
7     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
8     <uses-permission android:name="android.permission.CAMERA" />
9     <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
10    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

Figura 32 - Permissões AndroidManifest.xml da aplicação MediRemind

Como se pode ver na figura acima, as permissões são:

- **Acesso à Internet (linha 6):** é necessário o acesso à internet para o utilizador poder guardar os medicamentos que tem em casa, na Base de Dados externa MySQL.
- **Escrita no armazenamento externo (linha 7):** para permitir guardar as fotos, tiradas da câmara, no armazenamento externo do *Smartphone*.
- **Acesso à câmara (linha 8):** para permitir o aceso à câmara do dispositivo para tirar fotos aos medicamentos.
- **Leitura do armazenamento externo (linha 9):** para permitir escolher uma foto, quando o utilizador escolher se quer tirar uma nova foto ou inserir uma existente.
- **Iniciar Reciever quando o sistema for reiniciado (linha 10):** esta permissão serve para, quando o sistema for reiniciado, ter acesso às notificações, iniciando o BoradCast Reciever.

6.1.2. Design

O design da aplicação foi mudando ao longo do tempo, para que ficasse mais intuitiva e dinâmica para o utilizador, para isso foram usadas várias combinações de cores e animações.

Logotipo

O logotipo da aplicação é muito importante numa aplicação *mobile*, porque é a primeira imagem que é mostrada ao utilizador, antes de iniciar a aplicação.

A Figura 33 mostra o logotipo da aplicação MediRemind.

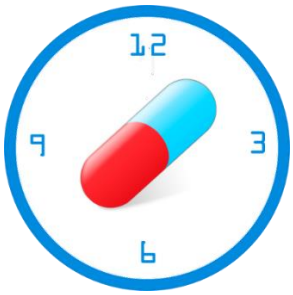


Figura 33 - Logotipo da aplicação MediRemind

Cores

As cores estão definidas num ficheiro XML, designado de colors.xml. Na tabela 22, está demonstrado o conjunto de cores e o respetivo código hexadecimal.

Tabela 22 - Cores dos componentes da aplicação

Cor	Código Hexadecimal	Utilização
	#ff4391c1	Títulos
	#33414b	CarView background
	#2e5169	TextView background detalhes
	#ffff0101	Botão eliminar
	#ffff5151	Botão eliminar selecionado
	#ff8c8b8b	Botão cancelar
	#ffb1adad	Botão cancelar selecionado

Na tabela acima, estão representadas apenas as cores dos componentes da aplicação. Para o *background*, foi utilizado um gradiente, para isso, foi criado um ficheiro background.xml. Dentro do ficheiro foi desenhado um retângulo com um gradiente onde foram definidas a cor inicial, a cor final, o raio do gradiente e o tipo.

A Figura 34 mostra o código do ficheiro background.xml.

```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">

    <gradient
        android:startColor="#33414b"
        android:endColor="#242424"
        android:gradientRadius="326"
        android:type="linear"/>
</shape>
```

Figura 34- Ficheiro background.xml

Botões

Para a aplicação ficar mais intuitiva e agradável, foram criados botões personalizados. Para isso foram criados três ficheiros XML para três tipos de botões diferentes:

- button_normal.xml
- button_eliminar.xml
- button_cancelar.xml

O ficheiro é um seletor que tem dois estados (normal e pressionado) e, conforme o estado, é criado um *shape* diferente.

A Figura 35 mostra o código do ficheiro button_normal.xml. O código é muito parecido em todos os botões, mudando apenas a cor e o tamanho, mas este botão tem uma

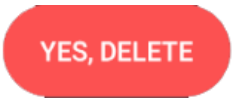
particularidade, tem um contorno (*stroke*) a branco em ambos os estados.

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true" android:state_focused="false">
        <shape>
            <solid android:color="@color/azulClaro"/>
            <stroke
                android:width="3dp"
                android:color="#fff" />
            <corners
                android:radius="50dp" />
        </shape>
    </item>

    <item android:state_pressed="false" android:state_focused="false">
        <shape>
            <stroke
                android:width="3dp"
                android:color="#fff" />
            <corners
                android:radius="50dp" />
        </shape>
    </item>

</selector>
```

Figura 35- Ficheiro button_normal.xml

A Tabela 23, mostra os botões nos diferentes estados.

Tabela 23 - Botões e estados

Ficheiro	Estado normal	Estado pressionado
button_normal.xml		
button_eliminar.xml		
button_cancelar.xml		

EditTexts

Similar aos botões, os EditTexts também têm diferentes estados, normal e pressionado. Em ambos os estados, tem fundo brando, mas no estado pressionado fica com o contorno azul e no estado normal fica com o contorno cinzento.

A Figura 36 e a Figura 37, demonstram um EditText no estado normal e no estado pressionado, respetivamente.

Figura 36 - EditText estado normal

Figura 37 - EditText estado pressionado

Animações

As animações adicionam sinais visuais e subtis que notificam o utilizador sobre o que está a acontecer na aplicação. As animações são especialmente úteis quando o ecrã muda de estado. (Android Developer, 2016)

As animações são criadas em ficheiros XML dentro da pasta anim.

A aplicação possui sete ficheiros com diferentes animações. Na Figura 38 está a lista de animações utilizadas na aplicação.

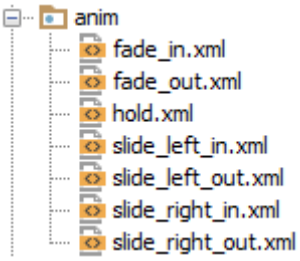


Figura 38 - Lista de animações da aplicação MediRemind

Todas as animações são percetíveis pelo nome. A que pode suscitar dúvidas é o ficheiro hold.xml. Este ficheiro não é propriamente uma animação, a sua função é deixar o *Fragment* visível, enquanto é sobreposto por uma *Activity* com a animação slide_right_in.xml.

Internacionalização

Para chegar a um público mais abrangente, é necessário internacionalizar a aplicação, ou seja, traduzir os textos para diferentes idiomas. Para isso, o *Android* dispõe de um ficheiro XML designado "strings.xml" para cada idioma diferente. No caso da aplicação MediRemind, está traduzida em português e inglês, o que equivale a ter dois ficheiros "string.xml", como demonstra a Figura 39.

Figura 39 - Ficheiros string.xml

Como demonstrado na figura acima, um dos ficheiros tem (en), o que quer dizer que dentro dele está a tradução em inglês. O dispositivo deteta automaticamente qual ficheiro deve escolher, consoante o idioma do dispositivo e, para isso acontecer, os atributos do nome são iguais nos dois ficheiros, mas cada um tem um texto diferente. A seguir estão dois exemplos, um string.xml em português e um em inglês.

```
<string name="nome_string">texto da string</string> <!-- String em português -->
<string name="nome_string">string text</string> <!-- String em inglês -->
```

Dimensões

Existem dispositivos *Android* com várias dimensões diferentes, por isso existe outro ficheiro XML designado "dimens.xml". Este ficheiro serve para guardar as dimensões com nomes, facilitando assim o seu uso, visto que se introduz o nome em vez da dimensão. Em seguida é mostrado um exemplo de uma dimensão.

```
<dimen name="titulo_dialog">20sp</dimen>
```

Como se pode ver no exemplo acima, foi definida uma dimensão com o nome "titulo_dialog" com uma dimensão de 20sp.

6.1.3. Classes

A aplicação contém várias classes auxiliares mais importantes da aplicação. Em seguida, vão ser enumeradas e explicadas as classes mais importantes.

- **Utilizador, Medicamento, PlanToma, Categoria:** são classes das tabelas da base de dados. É nestas classes que são guardados os dados que a aplicação recebe da base de dados.
- **BDHelper:** classe serve par gerir a criação e as versões da Base de Dados SQLite. Esta classe reescreve (*Override*) os métodos oncreate(), onde é criada a base de dados e as tabelas, e onUpgrade(), onde faz a gestão da base de dados quando houver uma nova versão da mesma.

  Na Figura 40, está o código do método mais importante, o oncreate(). Nas linhas 23 e 24, são criadas as tabelas Utilizador e PlanoTomas, respetivamente.

```
21      @Override
22      public void onCreate(SQLiteDatabase sqLiteDatabase) {
23          new TabelaUtilizadores(sqLiteDatabase).cria();
24          new TabelaPlanoTomas(sqLiteDatabase).cria();
25      }
```

Figura 40- onCreate da classe BDHelper

- **TabelaUtilizadores, TabelaPlanoTomas:** estas classes servem para gerir a as tabelas Utilizador e PlanoTomas, respetivamente, tais como criar, editar, eliminar e consultar os dados das respetivas tabelas.
- **JsonRequest:** esta classe possui dois métodos, getJsonConnection(), que é responsável por receber os dados do *Webservice* pelo método GET, e JsonSendPOST(), responsáveis por enviar e receber dados para o *Webservice* pelo método POST.
- **AdapterMedicamentos e AdapterPlanosToma:** estas classes, que estendem a classe RecyclerView.Adapter, é responsável por cada elemento do RecyclerView. A classe reescreve os métodos onBindViewHolder(), que é chamado pelo RecyclerView para mostrar os dados numa posição específica, e onCreateViewHolder(), que é chamado pelo RecyclerView quando precisa de um novo ViewHolder de um dado tipo que representa um item.

6.1.4. Activities

Em seguida será demonstrada e explicada cada *Activity* individualmente, com todas as suas funcionalidades.

Começando pela LoginActivity, que é a *Activity* principal, visto que para o utilizador poder usar a aplicação, necessita de ter um login válido.

6.1.4.1. LoginActivity

Esta *Activity* está dividida em duas partes, uma parte com o formulário para os utilizadores iniciarem sessão e outra parte para fazerem o registo.

A Figura 41 mostra o ecrã de Login.

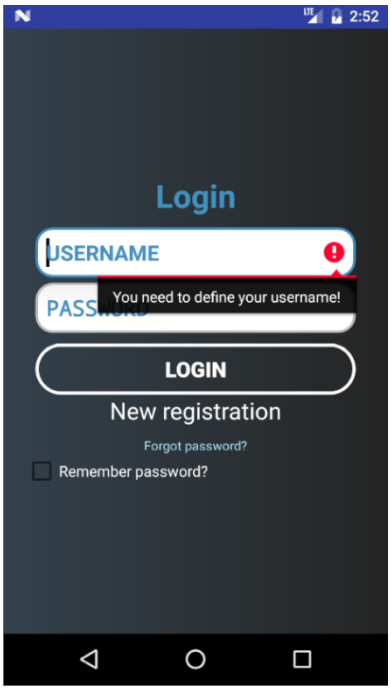


Figura 41 - Interface de login da aplicação MediRemind

Como demonstrado na figura acima, os campos são validados com uma mensagem (consoante o idioma do dispositivo) com o método setError(String). Ao clicar no botão "Login", a aplicação envia o URL com o username e password, pelo método GET, para o *Webservice* e, se o utilizador existir, é iniciada a aplicação HomeActivity.java, que é a *Activity* principal, se não existir, o servidor envia uma mensagem de erro.

Ao clicar em "Login", se a caixa "Remember password?" estiver ativa e se o utilizador existir, o mesmo é também guardado na Base de Dados SQLite, para que, da próxima vez que o utilizador iniciar a aplicação, não ser necessária a introdução dos dados de início de sessão novamente. Essa verificação é feita na *Activity* SplashScreenActivity,java.

A classe responsável por enviar os dados do login é a "Login", que não é mais que uma *AsnkTask*. Na Figura 42 está o método mais importante da classe, o doInBackground(), que é onde se passa todo o processo.

```
293         @Override
294         protected Utilizador doInBackground(String... strings) {
295             final String USERNAME = "username";
296             final String PASSWORD = "password";
297             Utilizador utilizador = null;
298 
299             try {
300                 // Construir o endereço com os parâmetros
301                 Uri.Builder builder = Uri.parse(enderecoLogin + "?").buildUpon();
302                 builder.appendQueryParameter(IDIOMA, idioma);
303                 builder.appendQueryParameter(USERNAME, strings[0]);
304                 builder.appendQueryParameter(PASSWORD, strings[1]);
305                 builder.build();
306 
307                 HttpURLConnection connection = BDConnection.getConnection(builder.toString());
308                 JsonRequest jsonRequest = new JsonRequest(connection);
309 
310                 //Receber resposta
311                 JSONObject jsonObjectCompleto = jsonRequest.getJsonConnection();
312                 JSONArray jsonArrayClientes = jsonObjectCompleto.getJSONArray("utilizadores");
313 
314                 //Recebe um novo utilizador para o guardar no sqlite
315                 JSONObject jsonObjectFinal = jsonArrayClientes.getJSONObject(0);
316 
317                 String erro;
318                 //Só se o erro for vazio é que guarda o utilizador
319                 if (jsonObjectFinal.toString().contains("erro")) {
320                     erro = jsonObjectFinal.getString("erro");
321                     resultado = erro;
322                 }
323                 else {
324                     utilizador = new Utilizador();
325                     utilizador.setId(jsonObjectFinal.getInt("id"));
326                     utilizador.setNome(jsonObjectFinal.getString("nome"));
327                     utilizador.setEmail(jsonObjectFinal.getString("email"));
328                     utilizador.setUsername(jsonObjectFinal.getString("username"));
329                     utilizador.setPassword(jsonObjectFinal.getString("password"));
330                 }
331 
332             } catch (IOException | JSONException e) {
333                 e.printStackTrace();
334                 resultado = e.getMessage();
335             }
336 
337             return utilizador;
338         }
```

Figura 42 - Código da classe Login

Como se pode ver na figura acima, o que o método doInBackground() faz é criar um URI (linha 301 a 305) com base no URL, onde é passado o idioma do dispositivo, o username

e a password introduzidos. Em seguida é guardado o URI na variável "connection", com o método estático "getConnection" (linha 307) e é criado um novo JSONObject com o utilizador recebido. Se o utilizador não existir, o servidor envia uma mensagem de erro que fica guardada na variável "erro" e sai do método, se existir, é criado um novo utilizador com os valores devolvidos pelo servidor.

Em seguida, estão os resultados do *Webservice*. A Figura 43 mostra a mensagem de erro e a Figura 44 mostra os dados de um utilizador, ou seja, com o login efetuado com sucesso.

{"utilizadores": [{"erro":"Wrong username or password!"}]}

Figura 43 - Mensagem de erro login

{"utilizadores": [{"id":"16","nome":"manel","email":"man@dad.com","username":"manel","password":"ed4ca94

Figura 44 - Utilizador no formato JSON

Ao clicar em "Forgor password?", é apresentado um Dialog para a introdução do E-Mail de registo para recuperar a password, como se pode ver na Figura 45.

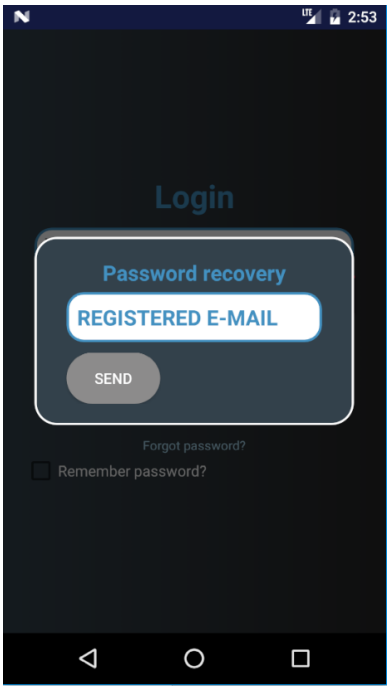


Figura 45 - Recuperar password

Ao clicar em "Send", os dados são enviados por POST para o *Webservice*, que verifica se o E-Mail existe na base de dados de utilizadores. Se existir, o servidor envia um E-

Mail ao utilizador com uma password aleatória e atualiza o campo da password do utilizador. Se o E-Mail não existir, o utilizador receber uma mensagem de erro.

Ao clicar em "New Registration", são apresentados os campos para registar um novo utilizador, como é demonstrado na Figura 46.

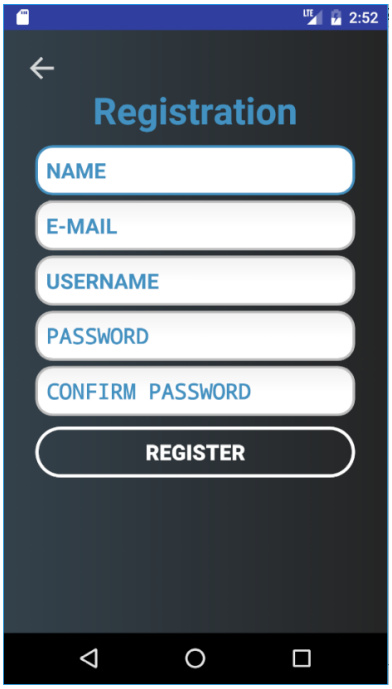


Figura 46 - Interface de registo de utilizadores na aplicação MediRemind

Ao clicar em "Register", os dados são enviados por POST para o *Webservice* que verifica se o Username ou o E-Mail já existem, se existirem, o servidor envia uma mensagem de erro. Os campos Password e Confirm Password também são validados, estes têm que ser iguais.

Se o Username e o E-Mail não existirem, o utilizador é criado e a aplicação volta à interface de Login para o utilizador iniciar sessão.

6.1.4.2. SplashScreenActivity

Esta é a primeira *Activity* a ser executada ao iniciar a aplicação. A sua função é mostrar ao utilizador que a aplicação está a iniciar, enquanto que, numa nova *Thread*, verifica se existe algum utilizador na Base de Dados SQLite. Se existir, é iniciada a *Activity* HmeActivity, se não existir, é iniciada a *Activity* LoginActivity.

A Figura 47 mostra a interface da SplashScreenActivity.

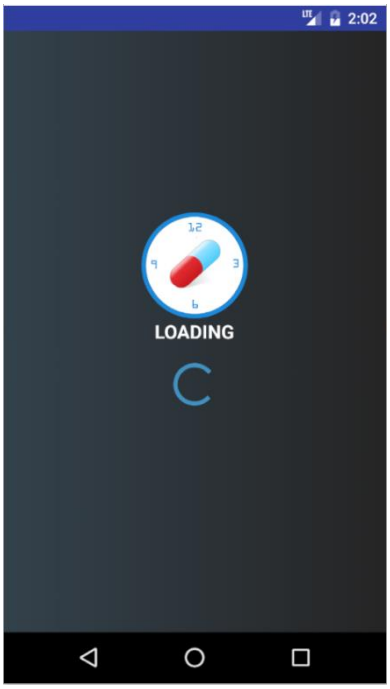


Figura 47 - SplashScreenActivity

Enquanto é mostrada a interface da imagem da figura acima, em segundo plano (*background*), é executada uma classe, designada *Background*, que não é mais que uma *AsynkTask*, como demostrado na Figura 48. Como foi referido anteriormente, esta classe serve apenas para fazer uma consulta à Base de Dados SQLite do *Android* e verificar se existe, ou não, algum utilizador.

```
private class Background extends AsyncTask<String, Void, Utilizador>{

    @Override
    protected Utilizador doInBackground(String... strings) {
        BDHelper bdHelper = new BDHelper(SplashActivity.this);
        SQLiteDatabase database = bdHelper.getReadableDatabase();
        TabelaUtilizadores tabelaUtilizadores = new TabelaUtilizadores(database);

        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        return tabelaUtilizadores.getUtilizador();
    }

    @Override
    protected void onPostExecute(Utilizador utilizador) {
        super.onPostExecute(utilizador);

        if (utilizador != null){
            SplashActivity.utilizadorLogin = utilizador;
            startActivity(new Intent(SplashActivity.this, HomeActivity.class));
        } else {
            startActivity(new Intent(SplashActivity.this, LoginActivity.class));
        }

        finish();
    }
}
```

Figura 48 - Código da classe Background

Como se pode observar na figura acima, a classe *Background*, que estende a classe *AsynkTask*, tem três tipos: String, Void e Utilizador.

Na aplicação MediRemind, o tipo mais importante é o terceiro, que é o resultado que o método doInBackground() devolve para o método onPostExecute().

Ao consultar a Base de Dados, o método doInBackground() devolve o utilizador para o método onPostExecute(), onde verifica se o utilizador existe, ou seja, se a variável utilizador não é nula. Se o utilizador existir, o utilizador é guardado numa variável estática do mesmo tipo, isto ajuda a aplicação a saber qual é o utilizador que está a utilizar a aplicação, e em seguida inicia a *Activity* HomeActivity. Se o utilizador não existir (a variável é nula), inicia a *Activity* LoginActivity.

6.1.4.3. HomeActivity

A HomeActivity é a *Activity* principal da aplicação, que contém um NavigationDrawer, que é um painel que possui as principais opções de navegação da aplicação.

Na aplicação MediRemind, o NavigationDrawer possui as seguintes opções de navegação:

- **Os meus medicamentos:** esta opção exibe o *Fragment* MeusMedicamentosFragment com a lista de medicamentos do utilizador
- **Alerta de medicamentos:** esta opção exibe o *Fragment* PlanoTomasFragment com a lista dos planos de toma ativos do utilizador.
- **Sair:** esta opção serve para fazer logout da aplicação. Ao clicar nesta opção, e se o utilizador estiver na Base de Dados SQLite, é eliminado e a aplicação volta para a Activity LoginActivity.

O NavigationDrawer possui também um cabeçalho onde são mostrados o nome e o e-mail do utilizador que iniciou sessão. Através do cabeçalho, também é possível editar o utilizador. A Figura 49 mostra a interface do NavigationDrawer e a Figura 50 mostra a *Activity* EditarUtilizadorActivity.

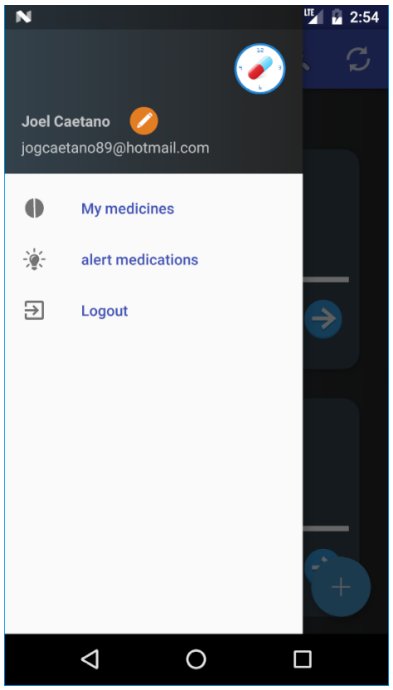


Figura 49 - NavigationDrawer

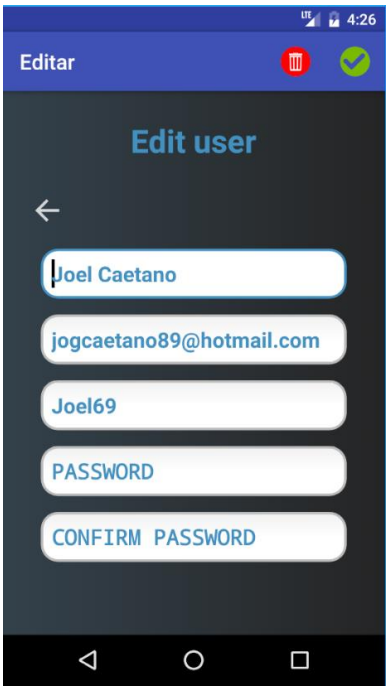


Figura 50 - EditarUtilizadorActivity

Ao iniciar a *Activity* EditarUtilizadorActivity, os campos do formulário são preenchidos automaticamente com os dados do utilizador que iniciou sessão, menos a password, que

tem que a definir. Para confirmar a edição, o utilizador carrega no ícone verde da *Toolbar* e os dados são enviados por JSON para o *Webservice*. A partir desta *Activity*, também é possível eliminar o utilizador, mas só é possível a eliminação, se o utilizador não possuir nenhum medicamento ou nenhum plano de toma ativo.

Com o login válido, é iniciada a HomeActivity com o *Fragment* MeusMedicamentosFragemt, que contém todos os medicamentos do utilizador registados.

O HomeActivity possui também um Floating Action Button (FAB), que é um botão de ação flutuante e que, no caso da aplicação MediRemind, serve para inserir medicamentos e planos de toma. Se o *Fragment* ativo for o MeusMedicamentosFragment, será iniciada a *Activity* InserirMedicamentosActivity, se for o PlanoTomasFragment, será iniciada a *Activity* InserirPlanoTomasActivity.

Nas Figura 51 está a interface da *Acitivity* InserirMedicamentoAcitivity e na Figura 52 está a *Activity* InserirPlanoTomasActivity.

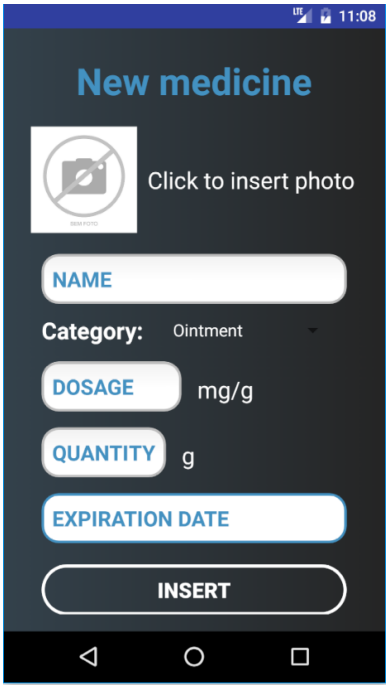


Figura 51 - InserirMedicamentosActivity

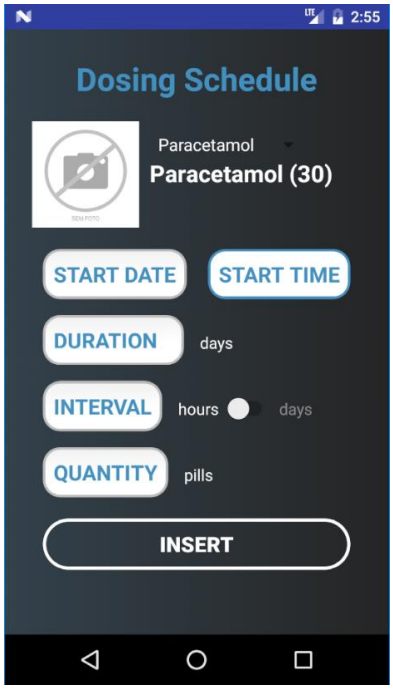


Figura 52 - InserirPlanoTomasActivity

Como demonstrado na Figura 51, para inserir um novo medicamento, é necessário introduzir o nome, a dosagem, a quantidade e a data de validade, assim como escolher a

categoria (por exemplo, comprimido, spray…). Também é possível inserir a fotografia do medicamento.

Ao clicar em "Click to insert photo", é apresentado um Dialog, onde o utilizador escolhe se quer inserir uma foto a partir da câmara ou a partir da galeria de fotos do *Android*.

Para obter a lista de categorias, é utilizada uma classe *AsynkTask*, designada "ReturnCategorias", que recebe os dados do *Webservice*, semelhante ao Login, e apresenta-os num Splinner.

Ao clicar em "Insert", é enviado o nome do medicamento para o *Webservice*, semelhante à classe "Login", e é verificado se esse nome já existe na base de dados. Se existir, é apresentado um Dialog ao utilizador a informar que já existe um medicamento com o mesmo nome, perguntando em seguida se pretende inserir uma nova unidade. Ao clicar em "Yes, insert", os dados são enviados para o *Webservice*, pelo método POST, onde são guardados na tabela Medicamento_Utilizador da Base de Dados MySQL. A fotografia do medicamento também é guardada na base de dados, mas, como a foto do dispositivo está no formato *Bitmap*, precisa de ser convertida previamente para o formato de texto. Em seguida está o método responsável por essa conversão.

```
public static String bitmapToString(Bitmap bitmap){
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.JPEG, 100, outputStream);
    byte[] bytes = outputStream.toByteArray();

    return Base64.encodeToString(bytes, Base64.DEFAULT);
}
```

Para inserir um novo plano de toma, como demonstrado acima, na Figura 52, é necessário escolher a lista de medicamentos, introduzidos previamente, assim como introduzir a data e hora de início, o intervalo entre tomas (pode ser em horas ou em dias) e a quantidade de medicamentos a tomar. Só é permitida a inserção de um novo plano de toma se tiver medicamentos suficientes para a duração (ver Algoritmo 7).

Ao contrário dos medicamentos, que ficam disponíveis para todo o agregado familiar por entrarem com o mesmo utilizador, o plano de tomas é exclusivo para cada membro, ou seja, ao clicar em "Insert", os dados são guardados na Base de Dados interna SQLite do *Android*, o que faz com que estejas disponível apenas no dispositivo onde o plano foi criado.

No momento da criação de um novo plano de toma, é criada uma notificação que irá notificar o utilizador quando estiver na hora de tomar o medicamento, consoante o intervalo de tomas definido. Para isso foi criado um método, na *Activity* InserirPlanoTomaActivity, para criar essa mesma notificação, como demonstrado no exemplo a seguir.

```
private void setNotification(String mensagem, String dataInicio, int intervaloTomas){
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(ConversorDatas.convertStringToDate(dataInicio));

    long intervalo = ConversorDatas.convHorasToMillis(intervaloTomas);

    Intent intent = new Intent(this, NotificationRecieve.class);
    intent.putExtra(NotificationRecieve.MENSAGEM_NOTIFICACAO, mensagem);

    PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);

    // Definir o alarme para notificar em cada intervalo de toma
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), intervalo, pendingIntent);

}
```

Como se pode ver no exemplo acima, foi definido um calendário com a data de início do plano de toma e, com a classe Intent, é passada a mensagem e iniciada a classe NotificationReciever, que estende a classe BoradcastReciever. Em seguida é criado um alarme com repetição em todos os intervalos de toma definidos.

NotificationReciever

Esta classe, tem como método obrigatório o "onRecieve", é responsável por criar a notificação, com a mensagem recebida pelo Intent.

Em seguida está o código do método "onRecieve".

```
@Override
public void onReceive(Context context, Intent intent) {
    // Receber a mensagem pelo intent
    String mensagem = intent.getStringExtra(MENSAGEM_NOTIFICACAO);
    createNotification(context, "Tomar medicamento", mensagem);
}
```

O método "createNotification" é semelhante ao exemplo da página 69, ou seja, cria uma notificação com um título e uma mensagem.

6.1.4.4. MeusMedicamentosFragment

O *Fragment* MeusMedicamentosFragment contém um *RecyclerView* com a lista de todos os medicamentos de um utilizador. Na Figura 53, está demonstrado o *Fragment* MeusMedicamentosFragment.

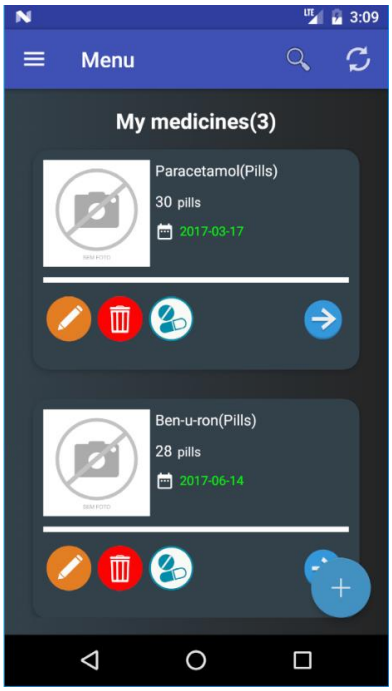


Figura 53 - MeusMedicamentosFragment

Como se pode ver na figura acima, no título aparece um número entre parêntesis, que é o número de medicamentos que o utilizador possui. Por cada medicamento é mostrada a fotografia, o nome, a quantidade, com a descrição da categoria, e a data de validade. Se o medicamento estiver na validade, esta aparece a verde, caso contrário aparece a vermelho.

Tal como para guardar a fotografia na Base de Dados MySQL é preciso convertê-la para formato de texto, para apresenta-la ao utilizador, também é necessário convertê-la, ou seja, ao receber a fotografia da Base de Dados, que está em formato de texto, a mesma é convertida para Bitmap. Em seguida está o método responsável por essa conversão.

```
public static Bitmap stringToBitmap(String string){
   byte[] bytes = Base64.decode(string, Base64.DEFAULT);
   return BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
}
```

A classe responsável por receber os medicamentos do servidor, pelo método GET, é a "AllMedicamentos", que é um *AsynkTask*. Esta classe é semelhante à classe "Login", mas, ao contrário da classe "Login", esta recebe vários objetos dentro de um JSONArray.

Em baixo, está parte do código onde a classe percorre todas as posições do JSONArray, com um ciclo "for" e, por cada posição é guardado um novo medicamento na lista de medicamentos.

```
for (int i = 0; i < jsonArray.length(); i++){
    Medicamento medicamento = new Medicamento();
    JSONObject jsonObjectFinal = jsonArray.getJSONObject(i);

    medicamento.setIdMedicamentoJson(jsonObjectFinal.getInt("id_medicamento"));
    medicamento.setNome(jsonObjectFinal.getString("nome"));
    medicamento.setFoto(jsonObjectFinal.getString("foto"));
    medicamento.setIdUtilizador(jsonObjectFinal.getInt("user"));
    medicamento.setDosagem(jsonObjectFinal.getString("dosagem"));
    medicamento.setQuantidade(jsonObjectFinal.getInt("quantidade"));
    medicamento.setDataValidade(jsonObjectFinal.getString("data"));

    JSONObject jsonObjectCategoria = jsonObjectFinal.getJSONObject("categoria");
    Categoria categoria = new Categoria();
    categoria.setIdJson(jsonObjectCategoria.getInt("id_categoria"));
    categoria.setDescricao(jsonObjectCategoria.getString("descricao"));
    categoria.setDosagem(jsonObjectCategoria.getString("cat_dosagem"));
    categoria.setQuantidade(jsonObjectCategoria.getString("cat_quantidade"));
    categoria.setQuantExtenso(jsonObjectCategoria.getString("quant_ext"));
    categoria.setIdiomaCategoria(idioma, categoria);
    medicamento.setCategoria(categoria);
    medicamentos.add(medicamento);
}
```

A classe AllMedicamentos é executada no método onCreate(), ou seja, cada vez que o *Fragment* é iniciado, é feita uma consulta à Base de Dados para verificar se há novos medicamentos.

Na Figura 54, está demonstrado o resultado esquematizado da Figura 53, numa página *Web*. A figura mostra todos os medicamentos do utilizador com o ID 1.

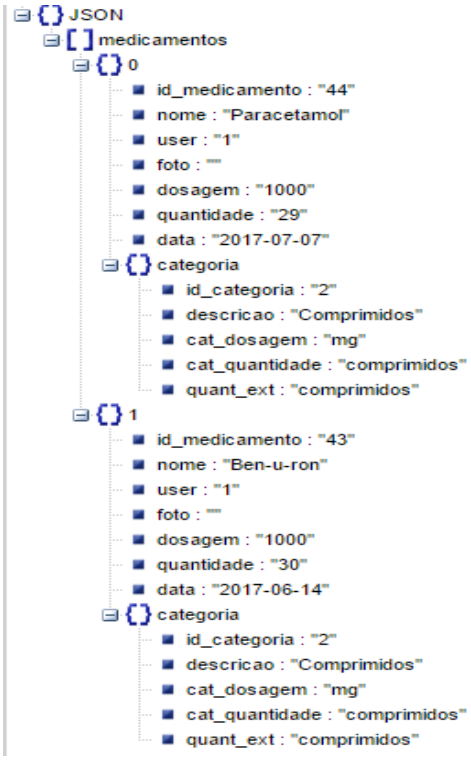


Figura 54 - Esquema JSON classe AllMedicamentos

Por cada medicamento também é possível editar, eliminar e tomar, assim como ver todos os detalhes do mesmo.

Na Figura 55, Figura 56 e Figura 57, estão demonstradas o as interfaces para editar, eliminar e tomar um medicamento, respetivamente.

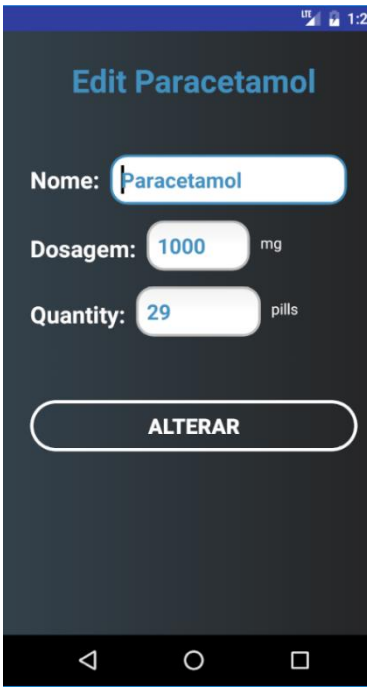


Figura 55 - EditarMedicamentoActivity

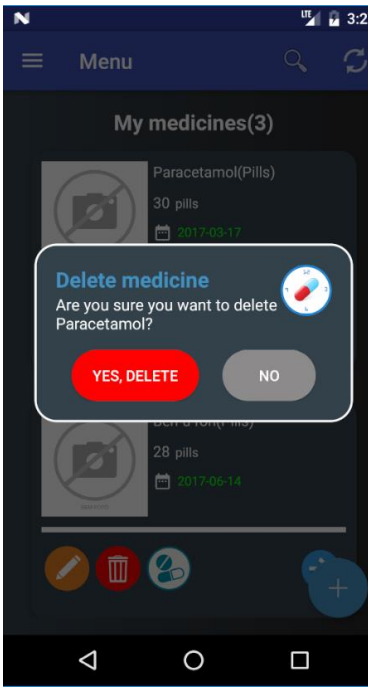


Figura 56 - Dialog eliminar medicamento

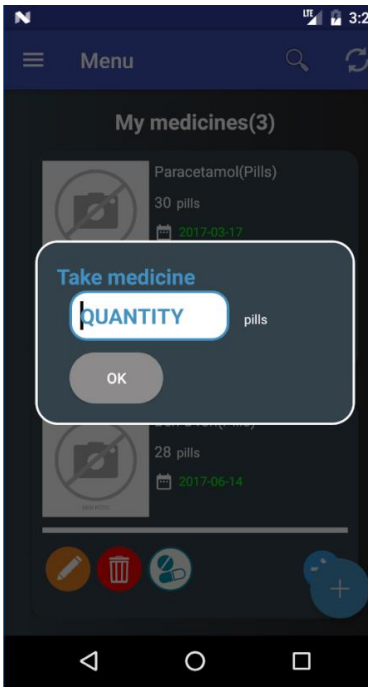


Figura 57 - Dialog tomar medicamento

No MeusMedicamentosFragement, ao clicar no ícone laranja (editar), é iniciada uma nova *Activity*, designada EditarMedicamentoActivity (Figura 55) que, como o próprio nome indica, serve para o utilizador editar o medicamento selecionado. Só é permitido editar três atributos do medicamento: o nome, a dosagem e a quantidade. Os atributos do medicamento são preenchidos automaticamente com os valores existentes, podendo o utilizador alterar qualquer um dos campos. Ao clicar em "Change", os dados são enviados por JSON, através da classe "EditarMedicamento", para o servidor PHP, onde são alterados na tabela Medicamento_Utilizador da Base de Dados.

A classe EditarMedicamento também estende a classe *AsynkTask* e o seu modo de funcionamento é semelhante à da classe Login (ver capítulo 6.1.4.1), mudando apenas os valores enviados e recebidos.

Ao clicar no ícone vermelho (eliminar), é apresentado um Dialog a perguntar se tem a certeza que pretende eliminar o medicamento (Figura 56). Ao clicar "Yes, delete", é enviada a informação para o servidor PHP e, se não estiver num plano de toma ativo, o medicamento é eliminado e desaparece da lista de medicamentos do MeusMedicamentosFragment.

Outro dos ícones apresentados é o de tomar medicamento (Figura 57). Ao clicar neste, é apresentado um *Dialog* para o utilizador inserir a quantidade de medicamentos a tomar. O campo tem validação, ou seja, não deixa inserir números negativos.

Ao clicar em "Ok", a quantidade introduzida é enviada para o servidor PHP e, essa quantidade é subtraída à quantidade total do medicamento que está na Base de Dados.

Por fim, ao clicar na seta (detalhes), é apresentada uma nova *Activity* onde são mostrados todos os detalhes do respetivo medicamento.

6.1.4.5. DetalhesMedicamentosActivity

A *Activity* DetalhesMedicamentosActivity é onde são mostrados todos os atributos de um único medicamento, tais como a fotografia, o nome, a dosagem, a quantidade, e a respetiva categoria, e a data de validade, como mostra a Figura 58.

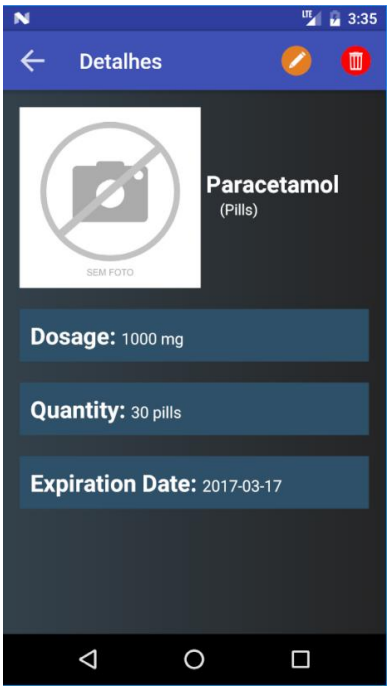


Figura 58 - DetalhesMedicamentoActivity

Esta *Activity*, dispões ainda de dois ícones na barra de tarefas (*Toolbar*) para editar e eliminar o medicamento, respetivamente, que têm as mesmas funcionalidades mencionadas anteriormente para os mesmos ícones.

Para além destas funcionalidades, o *Fragment* MeusMedicamentosFragment, possui ainda dois ícones na *Toolbar*, um para pesquisar um medicamento específico e outro para atualizar a lista de medicamentos.

Ao clicar no ícone de pesquisar (lupa), é apresentado um *Input* para o utilizador poder escrever. Ao introduzir caracteres, a aplicação procura dentro da lista de medicamentos se há algum que corresponde aos caracteres introduzidos. Se sim é/são apresentado(s) o(s) medicamento(s) correspondente(s).

A Figura 59 e a Figura 60, mostram duas formas de pesquisar, pela primeira letra ou por uma letra qualquer do nome, respetivamente.

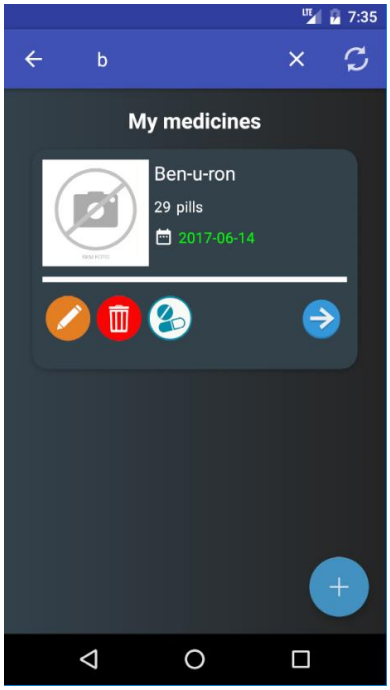


Figura 59 - Pesquisa pela primeira letra

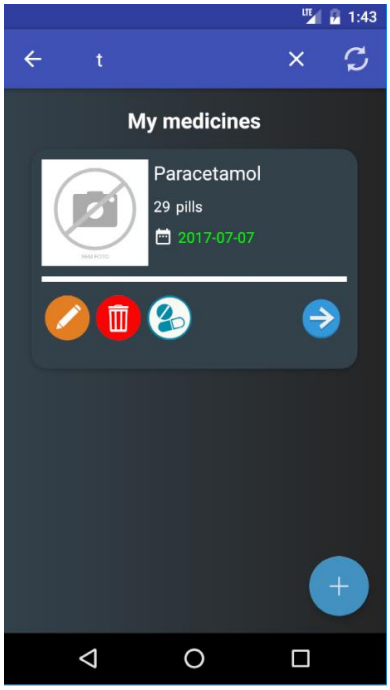


Figura 60 - Pesquisa por uma letra intermédia

Como se pode ver acima, na Figura 59, foi introduzido o caracter “b” e, como na lista de medicamentos só existe um Ben-u-ron e um Paracetamol, só é apresentado o primeiro. O mesmo se passa na Figura 60, mas neste caso foi introduzido um caracter contido no nome.

O ícone de atualizar, executa a mesma classe que o *Fragment* MeusMedicamentosFragment (AllMedicamentos) quando é iniciado, ou seja, envia os dados pelo método POST para o *Webservice* e devolve os medicamentos que estão na base de dados para o utilizador qua iniciou sessão.

6.1.4.6. PlanoTomasFragment

Como o *Fragement* MeusMedicamentosFragment, este também contém um *RecyclerView*, mas neste caso é para mostrar todos os planos de toma de um utilizador, como mostra a Figura 61.

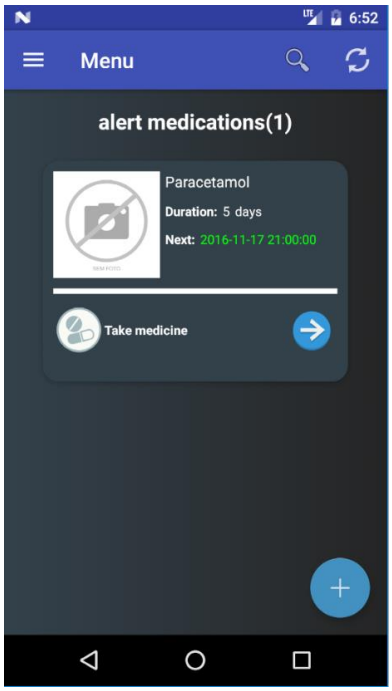


Figura 61 - PlanoTomasFragment

Por cada plano de toma é mostrado o medicamento (fotografia e nome), a duração do tratamento e a data da próxima toma, calculada pelo intervalo entre tomas.

Para mostrar a lista dos planos de toma, é utilizada a classe Cursor que faz uma consulta por cada linha do plano de toma existente na Base de Dados SQLite e insere-o no AdapterPlanoTomas. O AdapterPlanoTomas é o responsável por mostrar cada linha do plano de tomas.

A partir da lista dos planos de toma, é possível tomar o medicamento, sem abrir os detalhes do mesmo, bastando para isso clicar no ícone que diz “Take medicine” e, consoante o número de medicamentos que o utilizador definiu quando inseriu o plano de tomas, esse valor é descontado ao medicamento.

Ao clicar no ícone da seta (detalhes), é possível ver todos os detalhes do plano de toma selecionado, como demonstra a Figura 62.

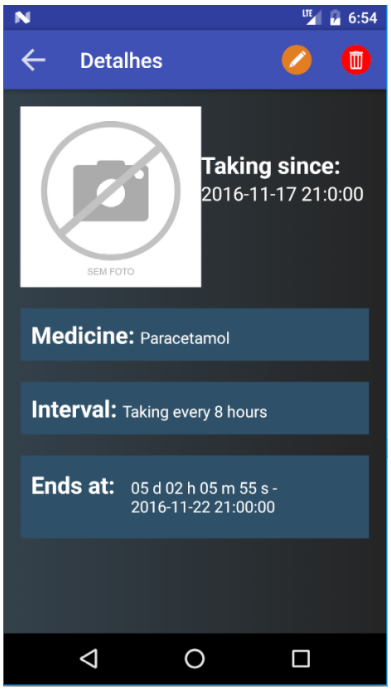


Figura 62 - DetalhesPlanoTomaActivity

Tal como a *Activity* DetalhesMedicamentosActivity, esta também mostra todas as informações acerca de cada plano de toma, tais como a foto e nome do medicamento, a data de início e o intervalo de horas entre tomas.

Além de todas as informações acima citadas, a *Activity* mostra também a data de fim, que é calculada através dos dias da duração, introduzida no momento da definição do plano de toma. Além da data, é também mostrada uma contagem decrescente entre a data do sistema e a data de fim.

Semelhante a *Activity* DetalhesMedicamentosActivity, esta também dispõe de dois ícones da *Toolbar*, para editar e eliminar o plano de toma, respetivamente. Tanto para editar como para eliminar o plano de toma, é apresentado um *Dialog*, como demonstrado na Figura 63 e na Figura 64.

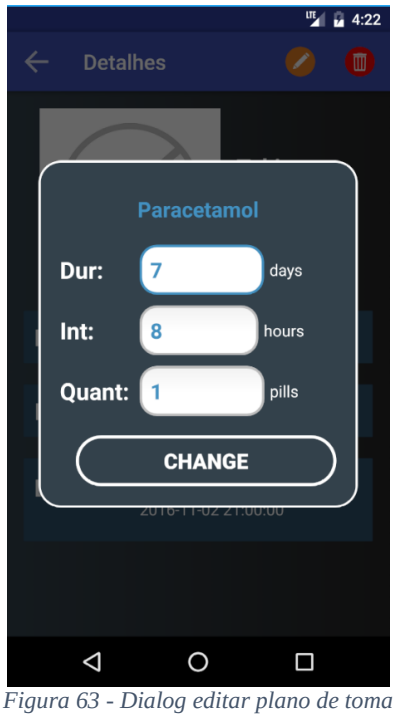


Figura 63 - Dialog editar plano de toma

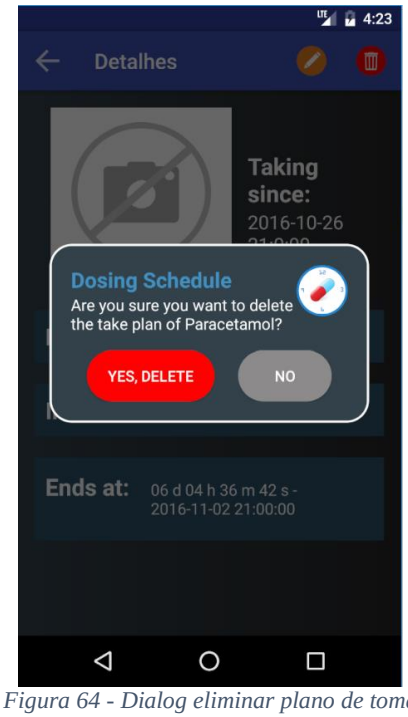


Figura 64 - Dialog eliminar plano de toma

Como se pode ver acima, na Figura 63, só é possível alterar os campos duração, intervalo de horas entre tomas e a quantidade a tomar em cada intervalo. Ao clicar em "Change", os dados são alterados na tabela Plano_Tomas da Base de Dados SQLite do *Android*.

No *Dialog* da Figura 64, é perguntado ao utilizador se tem a certeza que pretende eliminar o plano de toma. Se clicar "Yes, delete", o sistema vai verificar se o plano está ativo, ou seja, se a data do sistema é menor que a data do fim da toma. Se sim, o utilizador recebe uma mensagem de erro a dizer que não foi possível eliminar o plano de toma. Por outro lado, o plano de toma é eliminado da base de dadas e, consequentemente, da lista.

Para a comunicação com o *webservice*, a aplicação, além de enviar as informações necessárias, pelos métodos GET ou POST, envia também o Username e Password do utilizador que iniciou sessão e, antes de receber as informações, é verificado no servidor se os dados são válidos, ou seja, é sempre verificado o username e password com os campos da base de dados, para garantir que é o utilizador que está a fazer os pedidos.

6.2. Webservice

Para o desenvolvimento do *Webservice*, foi utilizada a linguagem PHP, devido à sua facilidade e simplicidade em trabalhar com valores de objeto JSON, por dispor de funções integradas na biblioteca do próprio PHP.

6.2.1. JSON no PHP

Em PHP, a estrutura de dados JSON para definir uma pessoa pode ser identificada de duas formas:

1) Como uma coleção de pares nome/valor, ficando semelhante a um array associativo em PHP: $pessoa = {"id": "1", "nome": "Joel", "idade": "27"}
2) Numa lista ordenada de valores, sem associações nome/valor: $pessoa = ["1", "Joel", "27"]

Como pode ser visto acima, foi criado um array em PHP para definir uma pessoa, mas, como está no formato de um array, é preciso converter os dados para um objeto JSON. Para isso, o PHP dispõe de duas funções:

- **json_decode(string variavel_json, boolean = false):** função que serve para descodificar um objeto JSON, ou seja, converte um objeto JSON (variável do primeiro parâmetro) num objeto PHP, se o segundo parâmetro for false ou nulo, ou num array associativo se o segundo parâmetro for true.

  Exemplo: converter um objeto em formato JSON para um objeto e array PHP.

```php
// Variável com um objeto JSON
$string = '{"nome": "Joel", "idade": "27"}';

// Utilizando a função json_decode para converter o objeto JSON num objeto PHP
$json = json_decode($string);

// Utilizando a função json_decode para converter o objeto JSON num array associativo PHP
$json = json_decode($string, true);


// Escrever o valor do parâmetro nome
echo $json->{'nome'};

// Escrever o valor da posição do nome
echo $json['nome'];
```

Como se pode ver no exemplo acima, depois de se converter um objeto JSON num objeto e num array em PHP, os valores podem ser acedidos individualmente como se variáveis se tratassem.

- **json_encode(variável_json):** função que serve para codificar um array em PHP para um objeto JSON, ou seja, converte um array associativo em PHP para uma string em formato JSON.

  Exemplo: converter um array em PHP para um objeto JSON.

```php
// Array associativo em PHP
$array = array("nome" => "Joel", "idade" => "27");

// Utilizando a função json_encode para converter o array num objeto JSON
$json = json_encode($array);

//Escrever a string completa do objeto JSON
echo $json;
```

  No exemplo acima, o processo foi invertido, ou seja, a partir de um array associativo em PHP, foi criado um objeto no formato JSON.

Consumindo os dados

A conversão dos dados de JSON para PHP e vice-versa, normalmente é feita com valores obtidos da Base de Dados e precisam de ser consumidos (consultados) por outros dispositivos, como no caso do presente projeto, por um dispositivo Mobile.

Já foi explicado no capítulo 9.1 como consumir os dados do *Webservice* através de um dispositivo *Android*, mas, o PHP também precisa de tratar os dados que recebe do *Android* e para isso é utilizada mais uma função nativa do PHP, file_get_contents(string filename). Esta função devolve um ficheiro string a que se quer aceder. O parâmetro da função pode ser um endereço *Web*, um ficheiro de texto, um input…

Para receber os dados do *Android*, o PHP não acede a nenhum endereço, mas como os dados em *Android* são enviados pelo método POST ou GET, é utilizado o filename "php://input", que serve para receber os dados por esses métodos.

6.2.2. Classes

A classe mais importante do servidor é a BDConnection, porque é a partir dela que é feita a ligação à base de dados através da classe PDO. Esta classe serve para inicializar o nome, username, password e hostname da base de dados, e o porto de comunicação.

No exemplo abaixo, está representado o construtor da classe BDConnection.

```
public function __construct() {
    $this->bdNome = LOCAL_DB_NAME;
    $this->bdPassword = LOCAL_DB_PASS;
    $this->bdUsername = LOCAL_DB_USER;
    $this->bdHostname = LOCAL_DB_HOST;
    try{
        $options = array(1002 => 'SET NAMES UTF8');
        $this->bdPorto = '3306';
        $this->connection = new PDO("mysql:host={$this->bdHostname}; dbname={$this->bdNome};
        port={$this->bdPorto}", $this->bdUsername, $this->bdPassword, $options);
        $this->connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    } catch(PDOException $pe){
        echo ($pe->getMessage());
    }
}
```

Como se pode ver no exemplo acima, além de inicializar as variáveis bdNome, bdPassword, bdUsername e bdHostname com os valores para a ligação à base de dados, também é criado um novo objeto PDO com esses mesmos valores (ver linha a sublinhado).

As variáveis LOCAL_DB_NOME, LOCAL_DB_PASS, LOCAL_DB_USER e LOCAL_DB_HOST, são constantes que estão definidas num ficheiro designado "config.php", que é onde estão definidas as configurações para a ligação à base de dados.

Em seguida está representado o ficheiro "config.php".

```
define('LOCAL_DB_NAME', 'mediremind');
define('LOCAL_DB_USER', 'root');;
define('LOCAL_DB_PASS', '');
define('LOCAL_DB_HOST', 'localhost');
```

Para a internacionalização da aplicação, também foi necessário traduzir as mensagens em português e inglês, para isso foi criada uma classe "IdiomaAndroid" que, consoante o idioma que o servidor receber do dispositivo, é enviada a mensagem nesse mesmo idioma.

Em seguida está parte do código da classe "IdiomaAndroid".

```
switch ($this->idioma){
   case 'English':
      //inserirUtilizador
      $lang['USERNAME_EXISTE'] = 'Username already exists!';
      $lang['EMAIL_EXISTE'] = 'E-Mail already exists!';
      $lang['REGISTO_SUCESSO'] = 'Successfully registered user!';
      $lang['ERRO_REGISTAR'] = 'Error registered user!';
      break;
   default:
      //Inserir utilizador
      $lang['USERNAME_EXISTE'] = 'O username já existe!';
      $lang['EMAIL_EXISTE'] = 'O E-Mail já existe!';
      $lang['REGISTO_SUCESSO'] = 'Utilizador registado com sucesso!';
      $lang['ERRO_REGISTAR'] = 'Erro ao registar utilizador!';
      break;
}
```

Como se pode ver no código acima, consoante o idioma, as posições do array "lang" tem diferentes mensagens. Para se utilizar, é só substituir a mensagem na função pela variável.

Tal como na aplicação *Android*, o servidor também possui as classes para a manipulação dos dados de cada tabela, são elas: Utilizador, MedicamentoUtilizador e Categoria.

Existem também duas classes para fazer a gestão dos valores das classes na base de dados, como inserção, consulta ou eliminação de dados, são elas a classe TabelaUtilizador e a classe TabelaMedicamentos.

6.2.3. Ficheiros PHP

No servidor, foi utilizado um ficheiro PHP por cada comunicação com o *Android*. Em seguida serão explicados, detalhadamente, cada ficheiro e respetivas funcionalidades, assim como exemplificado com um pedaço de código.

Em todos os ficheiros, há partes de código que são idênticos, se os dados são recebidos pelo método GET, começam por "if(isset($_GET['…'])", se são recebidos pelo método POST, começam por "if(isset($_POT['…'])". Em seguida é demonstrado um exemplo de cada um.

```
if (isset($_GET)) // quando os dados são recebidos pelo método GET
if (isset($_POST)) // quando os dados são recebidos pelo método POST
```

decodeJson.php

Este ficheiro serve apenas para receber o objeto JSON, pela função file_get_contents('php://input'), e descodifica-lo com a função json_decode, como demonstrado no exemplo abaixo.

```
$json = file_get_contents('php://input');
$ficheiros = json_decode($json);
```

Este ficheiro é incluído em todos os outros ficheiros que utilizam o método POST, visto que é por aqui que o PHP recebe e trata os objetos JSON.

Login.php

Este ficheiro serve para o utilizador iniciar sessão na aplicação *Android* (ver capítulo 9.1.4.1). O servidor recebe o username e password do utilizador que tentou iniciar sessão, assim como o idioma do dispositivo. Em seguida é chamada a função "getUtilizador" da classe TabelaUtilizadores, que faz a consulta à tabela "Utilizador" pelo username e password e, se existir, devolve o utilizador completo em formato JSON, com a função json_encode, se não existir, devolve uma mensagem de erro.

Código da função "getUtilizador":

```
public function getUtilizador($username, $password){
    $sql = "SELECT * FROM utilizador WHERE username = :username AND password = :password";
    $resultado = $this->connection->prepare($sql);
    $resultado->bindParam('username', $username);
    $resultado->bindParam('password', $password);
    $resultado->execute();
    $utilizadores = null;
    if ($obj = $resultado->fetch(PDO::FETCH_OBJ)) {
        $utilizadores[] = array("id" => $obj->id_utilizador, "nome" => $obj->nome, "email" => $obj->email,
            "username" => $obj->username, "password" => $obj->password);
    } else
        $utilizadores[] = array("erro" => $this->lang['USER_PASS_ERRADOS']);

    header('content-type: application/json');
    return '{"utilizadores": ' . json_encode($utilizadores) . '}';
}
```

Como se pode ver na função acima, o resultado da consulta fica guardada no array "utilizadores" que, se houver algum utilizador, são guardados os valores do utilizador, senão, é guardada uma mensagem de erro.

Registo.php

Este ficheiro serve para registar um novo utilizador na aplicação *Android* (ver capítulo 6.1.4.1). Ao receber os dados do formulário de registo, é criado um novo utilizador e é inserido na base de dados pela função "inserirUtilizador" da classe TabelaUtilizadores. A função insere os dados na Base de Dados e devolve uma mensagem com o resultado, em formato JSON.

Código da função "inserirUtilizador":

```
public function inserirMedicamento(MedicamentoUtilizador $medicamentoUtilizador){
   $resultado = null;
   try{
      $sql = "INSERT INTO medicamento_utilizador (id_categoria, id_utilizador, nome, foto, dosagem, quantidade,
data_validade)
      VALUES (:id_categoria, :id_utilizador, :nome, :foto, :dosagem, :quantidade, :data_validade)";
      $resultado = $this->connection->prepare($sql);

      $idCategoria = $medicamentoUtilizador->getCategoria();
      $idUtilizador = $medicamentoUtilizador->getIdUtilizador();
      $nome = $medicamentoUtilizador->getNome();
      $foto = $medicamentoUtilizador->getFoto();
      $dosagem = $medicamentoUtilizador->getDosagem();
      $quantidade = $medicamentoUtilizador->getQuantidade();
      $dataValidade = $medicamentoUtilizador->getDataValidade();

      $resultado->bindParam(':id_categoria', $idCategoria);
      $resultado->bindParam(':id_utilizador', $idUtilizador);
      $resultado->bindParam(':nome', $nome);
      $resultado->bindParam(':foto', $foto);
      $resultado->bindParam(':dosagem', $dosagem);
      $resultado->bindParam(':quantidade', $quantidade);
      $resultado->bindParam(':data_validade', $dataValidade);

      $resultado->execute();
      $result = $nome . $this->lang['INSERIDO_SUCESSO'];
   } catch (PDOException $pe){
      $result = $this->lang['ERRO_INSERIR'] . $pe->getMessage();
   }
   $inserido[] = array("resultado" => $result);

   return '{"resultados": ' . json_encode($inserido) . '}';
}
```

Ao inserir um novo utilizador, é guardado o resultado na variável "result" que, recebe a mensagem de sucesso ou de erro.

ForgotPass.php

Este ficheiro serve para o utilizador poder recuperar a password pelo E-Mail, inserido no formulário da aplicação *Android* (ver capítulo 9.1.4.1). Em seguida é chamada a função "getMail" da classe TabelaUtilizadores, onde é gerada uma password aleatória com letras minúsculas e maiúsculas, números e símbolos, com dez caracteres. Depois é feita uma consulta à base de dados para verificar se o E-Mail existe e, se existir é utilizada a classe

PHPMailer (ver capítulo 5.11) para enviar um E-Mail ao utilizador com uma password aleatória, em seguida, a password do utilizador é atualizada na base de dados.

Código da função "getMail":

```
public function getMail($email, $subject, $lang){
   $geraPass = $this->geraPassword();
   $mensagem = "
        <!DOCTYPE html>
         <html>
            <head>
            </head>
            <body>
               <p><strong>" . $lang['SOLICIT_PASS'] . "</strong></p>
               <p>" . $lang['NOVA_PASS'] . $geraPass . "</p>
            </body>
         </html>";

   $mail = new PHPMailer();
   $mail->isSMTP();
   $mail->SMTPDebug = 0;
   $mail->Debugoutput = 'html';
   $mail->Host = 'smtp.gmail.com';
   $mail->Port = 587;
   $mail->SMTPSecure = 'tls';
   $mail->SMTPAuth = true;
   $mail->Username = 'mediremind.esy.es@gmail.com';
   $mail->Password = MAIL_GMAIL;
   $mail->setFrom('mediremind.esy.es@gmail.com', $lang['EQUIPA_MEDI']);
   $mail->addAddress($email, 'MediRemind');
   $mail->Subject = $subject;
   $mail->msgHTML($mensagem);
   $mail->send();

   if (!$this->existeMailEnvio($email))
      $resultado = $lang['MAIL_INEXISTENTE'];

   else if (!$mail->isError()) {
      $resultado = $lang['MAIL_ENVIADO'];
      $this->updatePassword($email, sha1($geraPass));
   }
   else
      $resultado = $lang['ERRO_ENVIO'];

   $result[] = array("resultado" => $resultado);

   header('content-type: application/json');
   echo '{"enviado": ' . json_encode($result) . '}';
}
```

A função "getMail" começa por gerar uma nova password que fica guardada na variável "geraPass". Em seguida são definidas várias variáveis do PHPMailer para poder enviar o E-Mail, definido na variável "mensagem", e envia uma mensagem de sucesso ou erro.

EditarUtilizador.php

Para alterar, o utilizador altera os dados no formulário do *Andorid* (ver capítulo 9.1.4.1) e estes são enviados para o servidor. Em seguida os dados são alterados na tabela Utilizadores através da função "editarUtilizador" da classe TabelaUtilizadores e é devolvida uma mensagem de sucesso ou erro, em formato JSON, que é enviada para o *Android*.

O código da função editarUtilizador é semelhaste à do registo, mas em vez de inserir, altera, mudando apenas a variável "sql", como se pode ver no excerto de código abaixo.

```
$sql = "UPDATE utilizador SET nome = :nome, email = :email, username = :username, password = :password WHERE
id_utilizador = :id_utilizador";
```

EliminarUtilizador.php

Este ficheiro é executado quando o utilizador clica no botão eliminar, da *Activity* EditarUtilizadorActivity, no *Android*. O servidor recebe o id do utilizador que iniciou sessão e elimina o utilizador da base de dados, com a função "eliminarUtilizador". Por fim, é enviada a mensagem devolvida pela função, para o *Android*.

Código da função "eliminarUtilizador":

```
public function eliminarUtilizador($idUtilizador){
    $resultado = null;
    try{
        $sql = "DELETE FROM utilizador WHERE id_utilizador = :id_utilizador";
        $resultado = $this->connection->prepare($sql);
        $resultado->bindParam('id_utilizador', $idUtilizador);

        $resultado->execute();
        if (empty($idUtilizador))
            $result = $this->lang['ERRO_ID'];
        else
            $result = $this->lang['ELIIMINADO_SUCESSO'];
    } catch (PDOException $pe){
        $result = $this->lang['ERRO_ELIMINAR'];
    }
    $eliminado[] = array("resultado" => $result);

    return '{"resultados": ' . json_encode($eliminado);
}
```

ListaMedicamentos.php

Ao iniciar o *Fragment* MeusMedicamentosFragment (ver capítulo 6.1.4.4) no *Android*, é enviado o id do utilizador que iniciou sessão para o servidor. Em seguida é consultada a base de dados com a função "getAllMedicamentos" da classe TabelaMedicamentos, pelo id do utilizador e a função devolve todos os medicamentos que o utilizador possui, em formato JSON.

Código da função "getAllMedicametos":

```
public function getAllMedicamentos($idUtilizador){
    $sql = "SELECT mu.id_medicamento_utilizador, mu.nome, mu.id_utilizador, mu.foto, mu.dosagem, mu.quantidade,
mu.data_validade, c.id_categoria AS cat, c.descricao,
        c.dosagem AS cdos, c.quantidade AS cquant, c.quant_extenso
        FROM medicamento_utilizador mu, categoria c WHERE mu.id_categoria = c.id_categoria AND
        mu.id_utilizador = :id_utilizador";
    $resultado = $this->connection->prepare($sql);
    $resultado->bindParam('id_utilizador', $idUtilizador);
    $resultado->execute();

    $medicamentos = "";
    while ($obj = $resultado->fetch(PDO::FETCH_OBJ)){
        $medicamentos[] = array(
            "id_medicamento" => $obj->id_medicamento_utilizador,
            "nome" => $obj->nome, "user" =>$obj->id_utilizador,
            "foto" =>$obj->foto,
            "dosagem" => $obj->dosagem,
            "quantidade" => $obj->quantidade,
            "data" => $obj->data_validade,
            "categoria" => array('id_categoria' => $obj->cat, 'descricao' => $obj->descricao,
                "cat_dosagem" => $obj->cdos, "cat_quantidade" => $obj->cquant, 'quant_ext' => $obj->quant_extenso)
        );
    }

    header('content-type: application/json');

    if ($medicamentos == null)
        $medicamentos[] = array("erro" => $this->lang['SEM_MEDICAMENTOS']);

    return '{"medicamentos": ' . json_encode($medicamentos) . '}';
}
```

Tal como a função "getUtilizador", esta também guarda os medicamentos num array, neste caso designado "medicamentos".

VerificarMedicamento.php

Antes de inserir um medicamento, é consultada a base de dados para verificar se já existe algum medicamento com o mesmo nome. Para isso, foi criada a função "getNomeMedicamento" que devolve "Existe" ou "Novo", como se pode ver no código abaixo.

```
public function getNomeMedicamento($nome){
    $sql = "SELECT nome FROM medicamento_utilizador WHERE nome LIKE :nome";
    $resultado = $this->connection->prepare($sql);
    $resultado->bindParam('nome', $nome);
    $resultado->execute();

    if ($resultado->rowCount() > 0)
        $result = "Existe";
    else
        $result = "Novo";

    $mesmo[] = array("resultado" => $result);

    return '{"resultados": ' . json_encode($mesmo) . '}';
}
```

InserirMedicamento.php

A forma para inserir um medicamento é semelhante à forma de inserir um utilizador (registo). O servidor recebe os dados do medicamento que o utilizador preenche no formulário do *Android* e cria um novo medicamento. Os dados são inseridos na base de dados pela função "inserirMedicamento" da classe TabelaMedicamentos. A função devolve uma mensagem de sucesso ou erro em formato JSON.

EliminarMedicamento.php

Para eliminar um medicamento, depois do utilizador clicar em "Yes, delete" (ver página 82), o id do medicamento é recebido pelo o servidor e é eliminado através da função "eliminaMedicamento" da classe TabelaMedicamentos. Em seguida, a função devolve uma mensagem de sucesso ou erro, em formato JSON, e é enviada para o *Android*. Esta função é muito semelhante à função "eliminarUtilizador".

EditarMedicamento.php

A forma para editar um medicamento é semelhante à forma de editar um utilizador. O servidor recebe os dados preenchidos no formulário da *Activity* EditarMedicamentoActivity no *Android*, assim como o id do utilizador que iniciou sessão. Em seguida os dados são alterados através da função "updateMedicamento" da classe TabelaMedicamentos e esta devolve uma mensagem, em formato JSON, que é enviada para o *Android*. Esta função é muito semelhante à função "editarUtilizador".

UpdateQuantidadeMedicamento.php

Quando o utilizador clica na opção de tomar medicamento, no *Android* (ver página 82), a quantidade inserida, assim como o id do medicamento e a sua quantidade é enviada para o servidor, através da função "updateQuantidade" da classe TabelaMedicamentos, onde é subtraída a quantidade ao total de medicamentos (ver Algoritmo 6). Esse resultado é atualizado ao total de medicamentos da base de dados. Por fim, a função devolve o total de medicamentos, depois da atualização, e esse total é enviado para o *Android*, em formato JSON.

Código da função "updateQuantidade":

```
public function updateQuantidade($idMedicamento, $quantidade, $quantidadeTomar){
    $quantFinal = $quantidade - $quantidadeTomar;
    $sql = "UPDATE medicamento_utilizador SET quantidade = :quantidade WHERE
          id_medicamento_utilizador = :id_medicamento";
    $resultado = $this->connection->prepare($sql);
    $resultado->bindParam(':quantidade', $quantFinal);
    $resultado->bindParam(':id_medicamento', $idMedicamento);
    $resultado->execute();

    $select = "SELECT quantidade FROM medicamento_utilizador WHERE id_medicamento_utilizador = :id";
    $rest = $this->connection->prepare($select);
    $rest->bindParam(':id', $idMedicamento);
    $rest->execute();

    $quant = "";
    if ($obj = $rest->fetch(PDO::FETCH_OBJ))
        $quant[] = array("quantidade" => $obj->quantidade);

    header('content-type: application/json');
    echo '{"total": ' . json_encode($quant) . '}';
}
```

Conclusão

A realização deste projeto foi crucial para o aprofundamento dos conhecimentos de programação, obtidos ao longo do curso de Engenharia Informática, especialmente com um projeto que vai ao encontro de necessidades reais.

O objetivo do projeto era o desenvolvimento de uma aplicação *Android* para a gestão de medicamentos de um agregado familiar, com ligação a um *Webservice*, e pode-se dizer que o objetivo foi cumprido. A aplicação está pronta a ser usada pelo utilizador final.

As maiores dificuldades encontradas foram implementar o *Webservice* e fazer a ligação entre o *Android* e o servidor, e a implementar as notificações no *Android*, para o fim da embalagem dos medicamentos e planos de toma.

Para o trabalho futuro, prevê-se a implementação de novas funcionalidades que não foram possíveis no decorrer do projeto (exemplo: mapas com a localização das farmácias nas proximidades), e a implementação de uma plataforma *Web*.

Bibliografia

Altivo de Almeida. (Julho de 2016). *A Evolução das Tecnologias de Informação.* Obtido de 2A Tecnologia: http://www.2atecnologia.com.br/new/index.php/a-evolucao-da-tecnologia-da-informacao-e-os-novos-desafios-do-gestor-de-ti/

Android Developer. (Julho de 2016). *Activity*. Obtido de Android developer: https://developer.android.com/reference/android/app/Activity.html

Android Developer. (Dezembro de 2016). *Animations*. Obtido de Android developer: https://developer.android.com/guide/topics/graphics/view-animation.html

Android Developer. (Agosto de 2016). *App Manifest*. Obtido de Android Developer: https://developer.android.com/guide/topics/manifest/manifest-intro.html

Android Developer. (Agosto de 2016). *AsyncTask*. Obtido de Android developer: https://developer.android.com/reference/android/os/AsyncTask.html

Android Developer. (Agosto de 2016). *Creating Lists and Cards*. Obtido de Android developer: https://developer.android.com/training/material/lists-cards.html

Android Developer. (Setembro de 2016). *Dialogs*. Obtido de Android Developer: https://developer.android.com/guide/topics/ui/dialogs.html

Android Developer. (Agosto de 2016). *Fragments*. Obtido de Android developer: https://developer.android.com/guide/components/fragments.html

Android Developer. (Julho de 2016). *Handler*. Obtido de Android developer: https://developer.android.com/reference/android/os/Handler.html

Caelum. (Setembro de 2016). *O que é Java*. Obtido de Caelum: https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/

Enviar e-mails pelo PHP usando o PHPMailer. (Agosto de 2016). Obtido de LocaWeb: https://wiki.locaweb.com.br/pt-br/Enviar_e-mails_pelo_PHP_usando_o_PHPMailer

Farmácias Portuguesas. (Junho de 2016). *Nova App*. Obtido de PFP Mobile: http://pfpmobile.anf.pt/

Farminveste. (Junho de 2016). *Farmácias Portuguesas*. Obtido de Farmácias Portuguesas: https://www.farmaciasportuguesas.pt/app

Fonseca, D. (Julho de 2016). *Metodologias Clássicas x Extreme Programming)*. Obtido de DEVMEDIA: http://www.devmedia.com.br/conceitos-basicos-sobre-metodologias-ageis-para-desenvolvimento-de-software-metodologias-classicas-x-extreme-programming/10596

Hammerschmidt, R. (Junho de 2016). *Linha do Tempo: por dentro da evolução do Android*. Obtido de TecMundo: http://www.tecmundo.com.br/android/82344-linha-tempo-dentro-evolucao-do-sistema-android.htm

Jenkov. (Agosto de 2016). *Android View and ViewGroup*. Obtido de Jenkov: http://tutorials.jenkov.com/android/view-viewgroup.html

Lagvankar, T. (Julho de 2016). *Android : Passing data between main thread and worker threads*. Obtido de AdvanTej: http://techtej.blogspot.pt/2011/02/android-passing-data-between-main.html

Nakahara, L. (Junho de 2016). *Ciclo de vida de uma aplicação Android*. Obtido de Celeiro Android: http://celeiroandroid.blogspot.pt/2011/02/ciclo-de-vida-de-uma-aplicacao-android.html

PHP Group. (Julho de 2016). *Informações gerais*. Obtido de PHP: https://secure.php.net/manual/pt_BR/faq.general.php

PHP Group. (Julho de 2016). *PHP*. Obtido de Introdução SQLite: https://secure.php.net/manual/pt_BR/intro.sqlite.php

Romanato, A. (Julho de 2016). *Trabalhando com AsyncTask no Android*. Obtido de DevMedia: http://www.devmedia.com.br/trabalhando-com-asynctask-no-android/33481

TechTarget. (Julho de 2016). *URI (Uniform Resource Identifier)*. Obtido de Techtarget: http://searchsoa.techtarget.com/definition/URI

Tutorialspoint. (Junho de 2016). *What are Web Services?* Obtido de Tutorialspoint: https://www.tutorialspoint.com/webservices/what_are_web_services.htm

Wikipédia. (Agosto de 2016). *Android*. Obtido de Wikipedia: https://pt.wikipedia.org/wiki/Android

Wikipedia. (Outubro de 2016). *Android Studio*. Obtido de Android Studio: https://pt.wikipedia.org/wiki/Android_Studio

Wikipedia. (Setembro de 2016). *JSON*. Obtido de Wikipedia: https://pt.wikipedia.org/wiki/JSON

Wikipedia. (Julho de 2016). *MySQL*. Obtido de Wikipedia: https://pt.wikipedia.org/wiki/MySQL

Wikipedia. (Agosto de 2016). *Representational state transfer*. Obtido de Wikipedia: https://en.wikipedia.org/wiki/Representational_state_transfer

Wikipedia. (Junho de 2016). *Sistema de controle de versões*. Obtido de Wikipedia: https://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%B5es

Wikipedia. (Julho de 2016). *Use case diagram*. Obtido de Wikipedia: https://en.wikipedia.org/wiki/Use_case_diagram