IPG Politécnico da Guarda
Polytechnic of Guarda

RELATÓRIO DE PROJETO

Licenciatura em Engenharia Informática

Tiago Daniel Moura Fernandes

dezembro | 2015

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

INSTITUTO POLITÉCNICO DA GUARDA

EXPLORINOVA - ANIMA MOBILE

APLICAÇÃO MÓVEL PARA REGISTO E EXPORTAÇÃO DE DADOS EM EXPLORAÇÕES ANIMAIS

Tiago Daniel Moura Fernandes

Nº 1010400

Projeto de Informática em Contexto de Estágio do Curso de Licenciatura em Engenharia Informática

dezembro 2015

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

INSTITUTO POLITÉCNICO DA GUARDA

EXPLORINOVA - ANIMA MOBILE

APLICAÇÃO MÓVEL PARA REGISTO E EXPORTAÇÃO DE DADOS EM EXPLORAÇÕES ANIMAIS

Tiago Daniel Moura Fernandes

Nº 1010400

Projeto de Informática em Contexto de Estágio do Curso de Licenciatura em Engenharia Informática

Orientador: Professor Doutor Carlos Carreto

Coordenador: António Fernandes, Gerente da empresa Explorinova

dezembro 2015

Elementos Identificativos

Aluno

Nome: Tiago Fernandes

Número: 1010400

Curso: Engenharia Informática

Estabelecimento de Ensino

Escola Superior de Tecnologia e Gestão – Instituto Politécnico da Guarda

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

Telefone: 271220120 | Fax: 271220150

Instituição Acolhedora do Estágio

Nome: Explorinova

Morada: Avenida Dr. Francisco Sá Carneiro, Nº 50, Instituto Politécnico da Guarda, Policasulo Nº 5, 6300-559 Guarda

Duração do Estágio:

Inicio: 15 de Junho de 2015

Fim: 15 de Setembro de 2015

Orientador de Estágio:

Nome: Carlos Carreto

Grau Académico: Doutor

Coordenador:

Nome: António Fernandes

Cargo: Gerente da empresa Explorinova

Agradecimentos

Agradeço em primeiro lugar ao orientador Professor Doutor Carlos Carreto pela sua disponibilidade e tempo dispensado para ajuda na realização do relatório do projeto em causa.

Agradeço também a todos os elementos da empresa Explorinova pela ajuda e orientação que me deram. Esta ajuda demonstrou-se fundamental para o sucesso da realização deste projeto.

Por fim agradeço também à Professora Doutora Maria Clara, e Professor Doutor José Carlos Fonseca pela disponibilidade que dispuseram para esclarecimento de dúvidas.

Resumo

Este documento descreve o trabalho realizado na unidade curricular de Projeto de Informática da licenciatura em Engenharia Informática da escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda. O projeto foi desenvolvido em contexto de estágio na empresa Explorinova situada nos policasulos da escola Superior de Tecnologia e Gestão, e tem como tema uma aplicação móvel a desenvolver AnimaMobile.

A empresa Explorinova desenvolveu a aplicação Anima que tem a finalidade de ajudar explorações agropecuárias a gerir informação dos seus animais. Embora a aplicação ajude o agricultor neste sentido, sentiu-se a necessidade de facilitar ainda mais o processo de introdução de informação na aplicação.

Foi desenvolvida uma aplicação móvel que complemente a aplicação Anima, facilitando o registo de informação sobre os animais de uma exploração agropecuária. Esta aplicação permite o registo num ficheiro de texto de alguns dados dos animais como peso, produção de leite, e datas de quarentena, para mais tarde exportar os mesmos para a aplicação Anima. A aplicação também permite a consulta de registos efetuados. Foram implementadas duas formas de comunicação entre as duas aplicações, via Bluetooth, e via cabo USB. A comunicação via cabo USB é importante pois é necessário ter em conta a possibilidade de algumas explorações possuírem computadores mais antigos que não possuam tecnologia Bluetooth.

A aplicação foi desenvolvida para dispositivos *Android* utilizando o IDE Android Studio.

Os testes realizados à aplicação desenvolvida mostram que os objetivos foram alcançados e que a aplicação está muito perto de uma versão final.

Palavras-chave: Aplicação Android, Bluetooth, USB, Agropecuária.

Abstract

This document describes the work done in the course of Computer Project, in Computer Engineering degree from the Superior School of Technology and Management of the Polytechnic Institute of Guarda. This project was developed in the context of internship for the company Explorivona, located in the Technology and Management School, and has the theme AnimaMobile, witch is the mobile application that is going to be developed.

Explorinova has developed the application Anima with the intent of helping agricultural exploitations to manage the information of the animals that it owns. Although the application helps the farmer in this regard, he felt the need to further facilitate the process of entering information in the application.

A mobile application that complements the Anima application, facilitating the recording of information on the animals of an agricultural exploitation was developed. This application allows logging to a text file of some animal data such as weight, milk production, and dates of quarantine for later export them to Anima application. The application also allows the consultation of this records.

They were implemented two forms of communication between the two applications, Bluetooth, and USB cable. The communication via USB cable is important because it is necessary to take into account that some farms have older computers that do not have Bluetooth technology.

The application was developed for Android devices using the Android Studio IDE.

The tests performed to the application have shown that the objectives have been achieved and that the application is very close to a final version.

Keywords: Android Application, Bluetooth, USB, Agricultural.

Glossário

TCP – Transmission Control Protocol é um protocolo de nível da camada de transporte do modelo OSI

IP - Internet Protocol, é um protocolo de comunicação usado entre todas as máquinas em uma rede para encaminhamento dos dados.

MD5 – Message-Digest algorithm 5, é uma função criptográfica.

UUID – Universally Unique Identifier, é um identificador unico de 128 bits standart para aplicações informáticas.

NIF – Número de Identificação Fiscal, é um número que identifica uma entidade fiscal.

USB – Universal Serial Bus, é um tipo de conexão que permite a fácil conexão de periféricos.

ADB – Android Debug Bridge, é uma ferramenta de linha de comandos que permite a comunicação com dispositivos Android físicos, ou emulados.

SDK – Software Development Kit, é um *kit* de ferramentas para desenvolvimento de aplicações informáticas.

IDE – Integrated Development Environment, é um ambiente de desenvolvimento integrado para programação em diversas linguagens.

GPRS - General Packet Radio Service**,** é uma tecnologia que aumenta as taxas de transferência de dados nas redes GSM existentes.

Índice

1. Capítulo 1 Introdução 1
    1.1. Enquadramento e Motivação 1
    1.2. Definição de Problemas e Objetivos 3
    1.3. Solução Proposta para Resolução do Problema 4
    1.4. Organização do Relatório 5
2. Capitulo 2 Trabalhos Relacionados 6
    2.1. R.campo 6
    2.2. Tambeiro.com 7
3. Capitulo 3 Processo de Desenvolvimento de *Software* 8
4. Capitulo 4 Sistema Desenvolvido 10
    4.1. Arquitetura do Sistema Desenvolvido 10
    4.2. Aplicação *Android* 11
        4.2.1. Registo na Aplicação 12
        4.2.2. Efetuar Registos 14
        4.2.3. Registos Efetuados 20
        4.2.4. Comunicação 22
    4.3. Aplicação Desktop 38
        4.3.1. Editar Registos 39
        4.3.2. Base de Dados 42
5. Capitulo 5 Testes e Resultados 46
6. Capitulo 6 Conclusão e Trabalho Futuro 48

Índice De Figuras

Figura 1 - Aplicação Anima 1
Figura 2 - Arquitetura da aplicação Anima 2
Figura 3 - Aplicação R.campo 6
Figura 4 - Aplicação tambero.com 7
Figura 5 - Modelo Genérioco De Desenvolvimento De Software 8
Figura 6 - Arquitetura Do Sistema Desenvolvido 10
Figura 7 - Janela Principal Da Aplicação AnimaMobile 11
Figura 8 - Módulo Registo Na Aplicação 12
Figura 9 - Mensagem NIF Inválido 12
Figura 10 - Exemplo de Mensagens Com Layout Personalizado 13
Figura 11 - Módulo Efetuar Registos 14
Figura 12 - Trama De Dados De Um Registo 14
Figura 13 - Menus De Escolha De Registos 15
Figura 14 - Mensagem De Erro De Campos Nulos 16
Figura 15 - Formulário De Registos De Quarentena 16
Figura 16 - Trama De Dados De Um Registo De Quarentena 16
Figura 17 - Calendário Para Escolha De Data De Quarentenas 17
Figura 18 - Mensagem De Erro De Data De Quarentena Inválida 18
Figura 19 - Mensagem De Registo Inserido Com Sucesso 19
Figura 20 - Módulo De Consulta De Registos Efetuados 20
Figura 21 - Janela De Confirmação Para Apagar Registo 21
Figura 22 - Menú De Escolha De Tipo De Comunicação 22
Figura 23 - Diagrama Temporal De Comunicação Entre Aplicação Android E Aplicação Desktop 23
Figura 24 - Exemplo De Trama De Dados Com Checksum 24
Figura 25 - Janela De Permissão De Ativação De Bluetooth 26
Figura 26 - Diagrama de Verificação de Mensagens Recebidas (*Android*) 28
Figura 27 - Mensagem De Erro Bluetooth Não Suportado 30
Figura 28 - Janela De Escolha De Dispositivo Com Que Se Pertende Comunicar 31
Figura 29 - Diagrama de Verificação de Mensagens Recebidas (*Desktop*) 32

Figura 30 - Comando forward da Ferramenta ADB 34
Figura 31 - Modulo de Receção de Registos da Aplicação Anima 38
Figura 32 - Tabelas de Registos Recebidos 39
Figura 33 - Mensagem de Coonfirmação Para Apagar Registo da Tabela 40
Figura 34 - Mensagem de Erro: Animal Não Exisente na Base de Dados 41
Figura 35 - Mensagem de Erro: Grupo de Animais Não Existente na Base de Dados 41
Figura 36 - Mensagem de Sucesso de Registos Inseridos na Base de Dados 41
Figura 37 - Modelo Entidade Relacionamento da Base de Dados Criada 42

Índice De Tabelas

Tabela 1 - Entidade Animal 43
Tabela 2 - Entidade Exploração 44
Tabela 3 - Entidade Produção 45
Tabela 4 - Entidade Peso 45
Tabela 5 - Entidade Qurentena 46
Tabela 6 - Testes de Erros nos Registos Recebidos 47
Tabela 7 - Testes de Falhas na Comunicação 48

Capítulo 1

Introdução

1.1 - Enquadramento e Motivação

A Explorinova é uma empresa situada nos poli-casulos da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda, tendo em desenvolvimento vários projetos, como o projeto Anima.

O projeto Anima tem como finalidade a criação de um *software* gratuito para gestão de explorações agropecuárias, tanto de grandes como de pequenas dimensões. Este *software* permite gerir dados relevantes para uma exploração desse tipo, tais como a produção de leite, o peso dos animais, estipular diferentes dietas alimentares para diferentes animais e organizar visitas dos veterinários, entre outros. Na Figura 1 está representada uma imagem ilustrativa da aplicação Anima.

Figura 1 - Aplicação Anima

Entre as diversas funcionalidades da aplicação Anima podemos destacar:

- Gestão de informação dos animais (inserir, editar, consultar, apagar);
- Registo de dados da produção de leite de um animal, ou de um grupo de animais;
- Atribuir dietas alimentares aos animais com base na informação do seu peso;
- Controlo veterinário: acompanhamento sanitário (consultas veterinários, medicação, estados de quarentena, ou datas de monta).

A aplicação Anima funciona em qualquer tipo de computador com sistema operativo Windows. Para guardar os dados da exploração e dos seus animais é utilizada uma base de dados em MySQL. É necessário a ligação à Internet para ter ligação ao servidor da Explorinova.

Esta aplicação foi desenvolvida na linguagem de programação C#.

A Figura 2, ilustra a arquitetura da aplicação Anima.

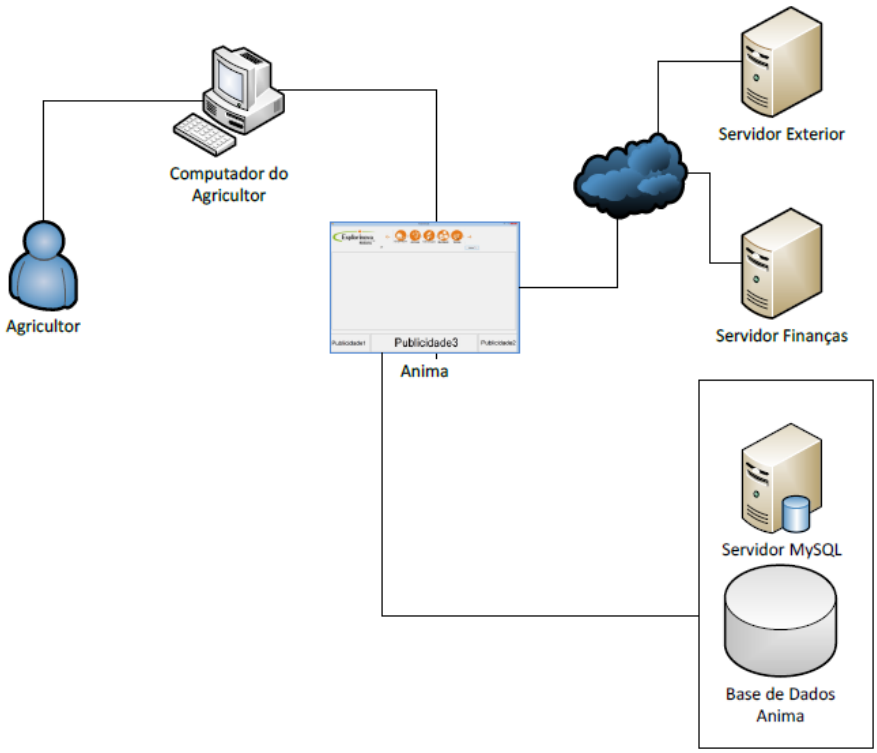

Figura 2 - Arquitetura da aplicação Anima

Embora a aplicação Anima facilite muito a gestão de uma exploração deste tipo, continua a existir um atraso na recolha de dados e na introdução dos mesmos na aplicação. O funcionário da exploração tem de registar os dados em papel e posteriormente introduzi-los na aplicação, instalada num computador situado num local próprio. Ainda que, hoje em dia, a existência de computadores portáteis permita o transporte dos mesmos, estes não são práticos para este tipo de situações. A solução para este problema passa pela utilização de dispositivos móveis como *tablets* ou *smartphones*. Para tal, foi desenvolvida uma aplicação móvel para registar todos os

dados necessários sobre produção de leite, peso e quarentena dos animais, que mais tarde são exportados para a aplicação *desktop* via ligações *Bluetooth* ou USB.

1.2 - Definição de Problema e Objetivos

Como foi referenciado na secção 1.1, é necessária a criação de uma aplicação móvel que complemente a aplicação Anima. Esta deve diminuir o tempo e a dificuldade de inserção de registos na aplicação Anima. Para tal, é necessário que seja fácil e rápida de utilizar.

De todos os sistemas operativos móveis existentes, destacam-se três para a criação desta aplicação, o sistema *Android*, sistema *IOS*, e sistema *Windows Phone*. Foi feito um breve estudo de cada um destes sistemas para averiguar qual deles seria melhor para o desenvolvimento de uma aplicação deste tipo para dispositivos *Tablet*. Tendo em conta que uma exploração animal é um local de trabalho propício a acidentes e sujidades, em que o dispositivo móvel pode ser facilmente danificado, temos de ter em conta o custo destes equipamentos. Quanto ao sistema operativo *IOS* existem equipamentos com valores entre 300€, e 1200€, e com sistema operativo *Windows Phone* os equipamentos têm custos dos 400€ aos 1800€. Já os equipamentos com sistema operativo *Android* têm custos dos 80€ aos 600€.

Podemos concluir que os equipamentos *Android* são significativamente mais baratos e, desta forma, os mais apropriados para este local de trabalho.

O sistema *Android* tem também outras vantagens como: ser um sistema *open source*, ser um dos sistemas com maior adesão por parte dos consumidores, ter um ambiente de desenvolvimento próprio (Android Studio[2]), o que fornece várias ajudas como a possibilidade de desenvolvimento para vários dispositivos diferentes ao mesmo tempo, um editor de código inteligente, editor de layout sofisticado e prático, entre outros.

Depois de escolhido o sistema operativo apropriado para a aplicação em causa, foram decididas os objetivos a alcançar no fim do projeto, sendo estes:

- Registo de peso, produção de leite e quarentena dos animais, ou grupos de animais.
- Consulta dos registos efetuados.
- Exportação dos registos para a aplicação Anima via Bluetooth.
- Exportação dos registos para a aplicação Anima via cabo USB.
- Validação dos registos recebidos na aplicação Anima.
- Inserção dos registos recebidos na base de dados da aplicação Anima.

1.3 - Soluções Propostas Para Resolução Do Problema

A solução encontrada para este problema, de acordo com os objetivos propostos na secção 1.2, foi a criação de uma aplicação móvel que ajudasse na recolha de registos de peso, produção de leite e quarentena dos animais de uma exploração.

A aplicação deve ser intuitiva e fácil de utilizar para diminuir o tempo de aprendizagem e manuseamento da mesma.

Esta aplicação deve ser desenvolvida para dispositivos com sistema operativo *Android*, uma vez que este oferece mais vantagens em relação a outros dispositivos com sistemas operativos diferentes.

1.4 - Organização do Documento

O presente documento está dividido em 6 capítulos distintos.

No primeiro capítulo é feita uma introdução ao projeto onde é exposta a motivação, a definição do problema, os objetivos previstos, e a solução encontrada para o problema em causa.

No segundo capítulo são descritas algumas aplicações relacionadas com a aplicação móvel desenvolvida neste projeto.

No terceiro capítulo é descrito o processo de desenvolvimento de *software*, todas as suas etapas, e no que consiste cada uma delas.

O quarto capítulo contém a implementação da solução encontrada para o problema em causa. É explicada a arquitetura do sistema criado bem como cada um dos seus módulos individuais.

No quinto capítulo são explicados os testes feitos á aplicação já desenvolvida, no que consistem e resultados obtidos.

O sexto capítulo é composto pelas conclusões finais e trabalho futuro.

Capítulo 2

Trabalhos Relacionados

Foi feito um estudo de duas soluções móveis para recolha de dados de animais no setor agropecuário.

Este estudo revelou a falta de aplicações móveis para o setor agropecuário, o que aumentou a motivação para o desenvolvimento deste projeto.

2.1 - R.campo

R.campo[3] é uma aplicação móvel paga, desenvolvida pela empresa Ruralbit que permite a recolha de dados no campo para posteriormente fazer a sua sincronização com outras plataformas. Algumas das plataformas compatíveis com esta aplicação são Genpro, e-Exploração e Ia-AçoresA. É também possível adaptar a aplicação a outras plataformas através de *webservice*, sendo que para tal é necessário contactar a empresa e expor esta situação. Na Figura 3, está representada uma imagem ilustrativa da aplicação R.campo.

Figura 3 - Aplicação R.campo

A recolha de dados é feita completamento *offline*, não necessitando de qualquer tipo de rede. Posteriormente a sincronização com outras plataformas é feita via GPRS, ou Wi-Fi.

A aplicação encontra-se disponível para a plataforma *Android*, embora no futuro possa ser desenvolvida para outras plataformas.

2.2 - Tambero.com

Tambero.com[4] é uma solução gratuita no formato WebPage para gestão de gado de engorda e criação de gado leiteiro. Permite adicionar e editar animais ou rebanhos, gerir épocas de reprodução e ordenhas, além de que fornece algumas informações úteis como períodos de chuva. Na Figura 4 estão representadas algumas das funcionalidades desta aplicação.

Figura 4 - Aplicação tambero.com

Sendo uma solução em formato WebPage está disponível através de um *browser* em qualquer dispositivo com acesso à internet. Caso não exista acesso à internet, não é possível aceder à aplicação.

Os dados são guardados diretamente numa base de dados presente nos servidores remotos.

Capítulo 3

Processo De Desenvolvimento De Software

Este projeto foi concebido seguindo um modelo genérico de desenvolvimento de projetos de *software*. Este modelo é iterativo e composto por quatro etapas base, nomeadamente: especificação, arquitetura e implementação, validação, e evolução. O modelo foi seguido e revelou-se muito prático para a boa realização deste projeto. Na Figura 5, está representada a sequência destas etapas.

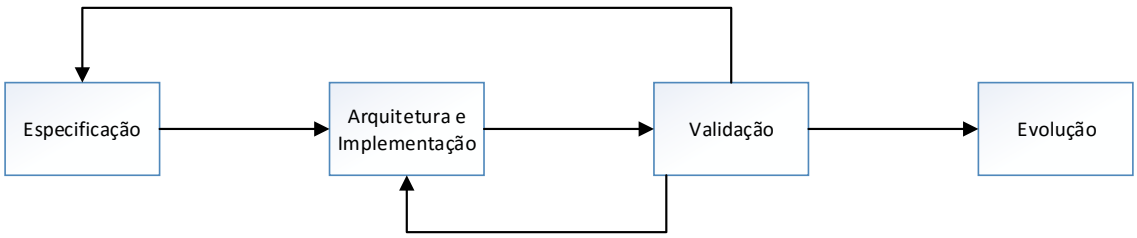

Figura 5 - Modelo Genérioco De Desenvolvimento De Software

Quando o projeto se encontrava em fase de Especificação, foram feitas várias reuniões com a empresa responsável para definir os objetivos referenciados na secção 1.2, e estudar soluções para alguns problemas que foram surgindo.

Depois de definidos os objetivos, passamos para a fase de Arquitetura e Implementação onde foi desenhada a arquitetura do sistema e implementada a solução da mesma, presente no capítulo 4.

Quando o desenvolvimento do projeto chega ao fim, este passou a um estado de Validação onde foi feita uma reunião com a empresa. Nesta fase, foi necessário alterar alguns aspetos no projeto, o que levou a uma revisão das especificações do mesmo.

Uma vez validado o projeto, passamos para a fase de Evolução, onde foram melhorados alguns funcionamentos do sistema.

Para facilitar o desenvolvimento do projeto, foram determinadas algumas tarefas a realizar por ordem cronológica. A elaboração destas tarefas levou a um desenvolvimento mais fluido e natural do projeto.

Ordenação das tarefas realizadas:

- Estudo do problema.
- Desenho da aplicação para *Android*.
- Implementação da recolha de registos.
- Implementação da comunicação via *Bluetooth* e via cabo USB.
- Implementação do tratamento de dados dos registos recebidos.
- Criação de base de dados.
- Implementação da inserção de registos na base de dados.
- Testes

Capítulo 4

Sistema Desenvolvido

Neste capítulo, vai ser descrita a implementação de cada módulo do sistema desenvolvido para a solução do problema referido na secção 1.2.

4.1 - Arquitetura Do Sistema Desenvolvido

Na Figura 6 está representada a arquitetura do sistema desenvolvido.

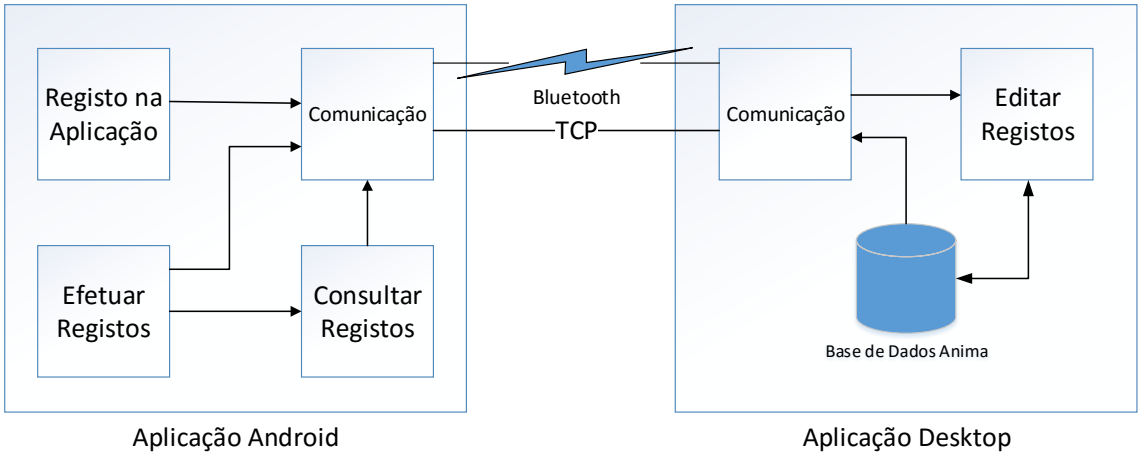

Figura 6 - Arquitetura Do Sistema Desenvolvido

Como se pode verificar existem dois módulos principais, a aplicação *Android* e a aplicação *desktop*. A aplicação *Android* é composta por quatro módulos distintos. No módulo "Registo na Aplicação", o utilizador regista o NIF da exploração animal na aplicação, informação essencial para a comunicação entre a aplicação *Android* e aplicação *desktop*. No módulo "Efetuar Registos" são feitos os registos de peso, produção de leite e quarentena dos animais, que são guardados num ficheiro de texto. No módulo "Consultar Registos" podem ser consultados os registos efetuados, existentes no ficheiro de texto. Por último, o módulo "Comunicação", onde é feita a comunicação com a aplicação *desktop* para exportação dos registos (via tecnologia *Bluetooth*, ou via cabo USB utilizando *sockets* TCP).

A aplicação *desktop* é composta por três módulos principais, existindo também um módulo "Comunicação" para importar os registos da aplicação *Android*. No módulo

"Editar Registos" é possível visualizar, editar ou apagar os registos recebidos. Por último, o módulo "Base de Dados", onde são inseridos todos os novos registos na base de dados da aplicação.

4.2 - Aplicação Android

Como verificamos na arquitetura do sistema (Figura 6), a aplicação AnimaMobile contém quatro funcionalidades principais, sendo estas: Registo na Aplicação, Efetuar Registos, Consultar Registos, e Exportar Registos (comunicação com aplicação *desktop*). Nas secções seguintes será feita a descrição detalhada de todas estas funcionalidades.

Na Figura 7, está representada a página principal da aplicação, que funciona como menu para as diferentes funcionalidades.

Figura 7 - Janela Principal Da Aplicação AnimaMobile

4.2.1 - Registo na Aplicação

Na primeira utilização da aplicação, é pedido ao utilizador que seja introduzido o NIF da exploração, como representado na Figura 8. Esta informação vai ser fundamental para a comunicação com a aplicação *desktop*.

Para validar o NIF foi criada a função validaNIF(). Esta função verifica se a caixa de texto contém informação, e verifica também a integridade do NIF, isto é, se o número inserido na caixa de texto corresponde a um NIF real. Para ser validado, o NIF tem um tamanho de 9 algarismos, sendo o 9º algarismo o dígito de controlo, que também é confirmado dentro da função. Caso seja detetado um erro no NIF, é mostrada uma mensagem de erro "NIF inválido", como representado na Figura 9.

Figura 8 - Módulo Registo Na Aplicação

Figura 9 - Mensagem NIF Inválido

O código da função validaNIF(String nif) encontra-se no Anexo A - 1.

Se a função validaNIF() retornar “verdadeiro”, o NIF é guardado num ficheiro de texto situado na memória *flash* do dispositivo *Android*, e é aberta a página principal da aplicação. É também mostrada uma mensagem de sucesso.

Caso a exploração animal já se encontre registada na aplicação, o utilizador é redirecionado automaticamente para página principal da aplicação. O código que verifica se uma exploração já se encontra registada está presente no Anexo B-1.

Para tornar mais visíveis as mensagens mostradas ao utilizador, foram criadas mensagens de texto personalizadas. Para tal foram criados dois novos *layouts*, um para as mensagens de sucesso e outro para mensagens de erro, como é visível na Figura 10.

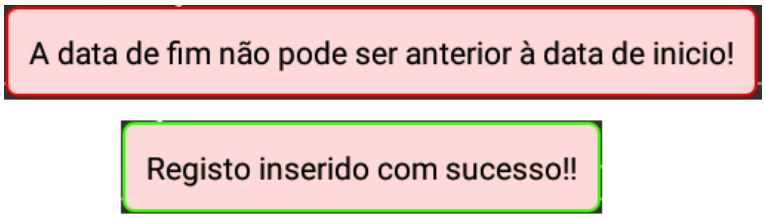

Figura 10 - Exemplo de Mensagens Com Layout Personalizado

Para mostrar as mensagens ao utilizador foi criada uma função toasts(int tipo, String texto), com dois parâmetros de entrada, sendo o primeiro parâmetro o tipo de mensagem (mensagem de erro ou de sucesso) e o segundo parâmetro o texto a mostrar na mensagem. Em anexo (Anexo A- 2,) encontra-se o código respetivo a esta função.

4.2.2 - Efetuar Registos

Neste módulo é realizado o registo de produção de leite, peso e quarentena dos animais. A Figura 11 representa o *layout* deste módulo.

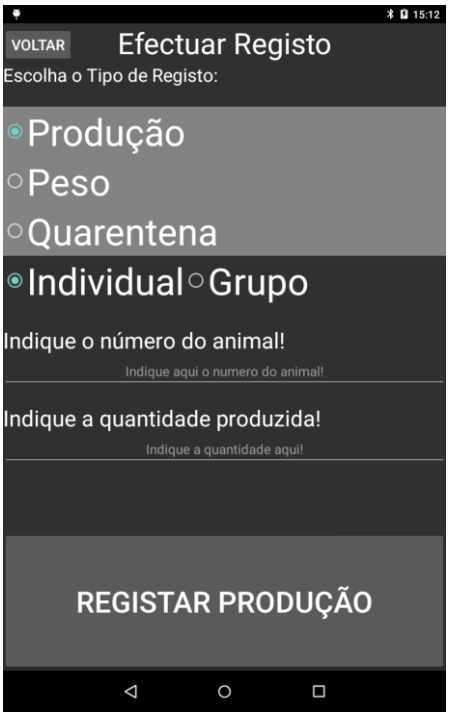

Figura 11 - Módulo Efetuar Registos

Os registos produção de leite e peso podem ser individuais, onde é registada a identificação do animal; ou de um grupo de animais, em que é registada a identificação do grupo. Quanto à quarentena dos animais, é apenas permitido realizar registos individuais.

Estes registos são guardados num ficheiro de texto situado na memória do dispositivo (ficheiro criado quando feito o registo da exploração na aplicação). Cada linha deste ficheiro vai conter apenas um registo, com toda a sua informação dividida pelo caracter '#'. A Figura 12, mostra a estrutura de um registo guardado no ficheiro de texto.

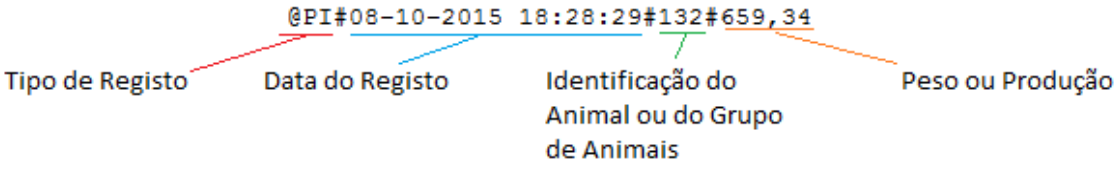

Figura 12 - Trama De Dados De Um Registo

Para facilitar a recolha de registos, foram implementados duas ferramentas *RadioGroup* que funcionam como menus de escolha, em que o utilizador tem várias possibilidades e escolhe a pretendida.

O primeiro menu contém escolhas relacionadas com o tipo de registo a efetuar, isto é, peso, produção de leite ou quarentena. Após a escolha de uma destas opções, é mostrado um segundo menu que contém apenas duas escolhas possíveis, "Individual" ou "Grupo", onde é definido se o registo em causa tem como alvo um animal específico ou um grupo de animais. Quando é escolhida a opção do segundo menu, é mostrado o formulário pretendido.

Como foi referido anteriormente, os registos de quarentena são apenas individuais, desta forma quando o utilizador escolhe a opção "Quarentena", é imediatamente aberto o formulário de quarentena, e não o segundo menu.

Na Figura 13, estão representados os menus referidos anteriormente.

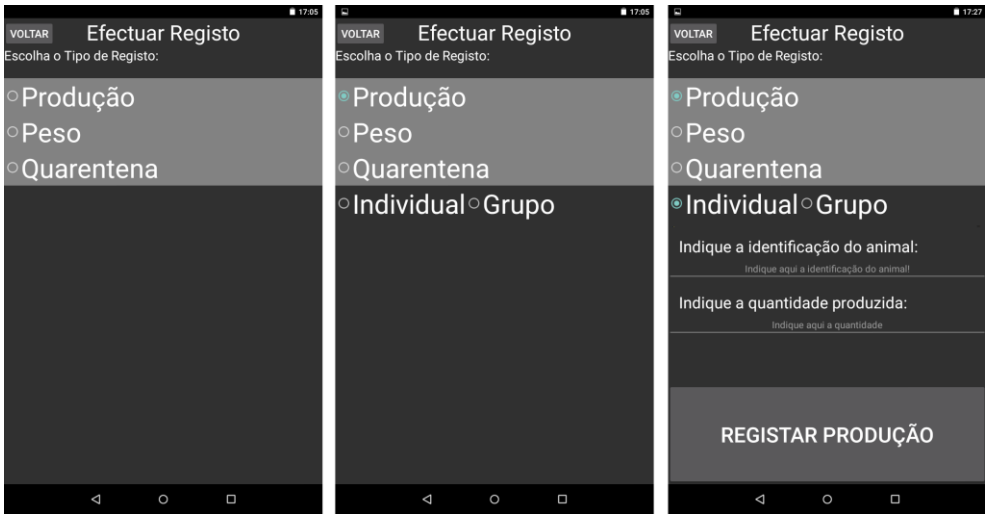

Figura 13 - Menus De Escolha De Registos

Quando é efetuado um registo, para todos os campos, sejam eles de identificação de animais ou quantidade produzida e peso, é feita uma validação que garante que nenhum parâmetro é registado com um valor nulo. Caso esta validação não se verifique, é mostrada uma mensagem de erro como na Figura 14 e o registo não é inserido no ficheiro de texto.

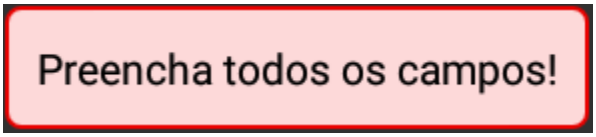

Figura 14 - Mensagem De Erro De Campos Nulos

Quanto aos registos de quarentena, é pedido ao utilizador a identificação do animal em questão, a data de início de quarentena, a data de fim de quarentena e o motivo pelo qual o animal se encontra em quarentena. A Figura 15 ilustra o formulário a preencher para efetuar o registo de uma quarentena.

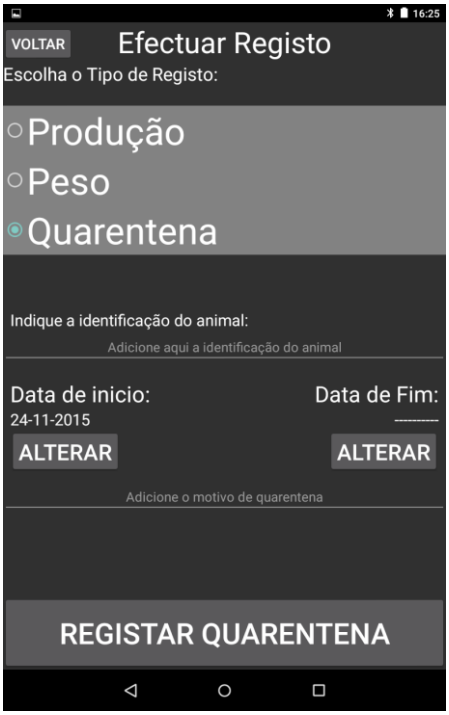

Figura 15 - Formulário De Registos De Quarentena

A Figura 16 mostra a estrutura de um registo de quarentena, guardado no ficheiro de texto.

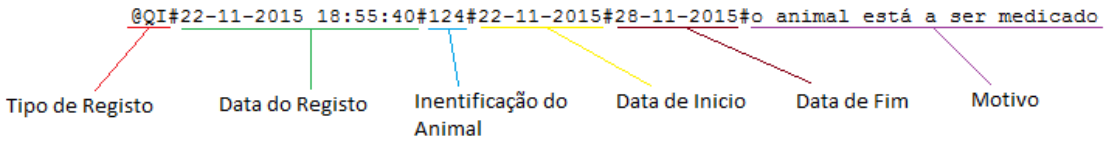

Figura 16 - Trama De Dados De Um Registo De Quarentena

Para ajudar o utilizador a inserir as datas de início e de fim de quarentena, é utilizado um *DatePickerDialog*, como é representado na Figura 17. Esta ferramenta mostra um calendário onde o utilizador seleciona uma data pretendida. Por omissão, a data de início de quarentena é preenchida com a data atual, podendo esta ser alterada.

Figura 17 - Calendário Para Escolha De Data De Quarentenas

Como em qualquer outro tipo de registo, é feita uma validação que garante que não são inseridas informações nulas.

Foi também criada uma função validaDataQuarentena() que impede o utilizador de registar uma quarentena em que a data de fim, seja inferior a data de início. Esta função vai devolver o valor "*true*" se a data for validada, e é devolvido o valor "*false*" caso a data não seja validada.

O código apresentado a seguir representa a função validaDataQuarentena().

```
public boolean validaDataQuarentena(){

  if (anoFim < ano){
      return false;
  }
  if (ano == anoFim) {
      if (mesFim < mes) {
          return false;
      }
  }
  if (mesFim == mes){
      if (diaFim<dia){
          return  false;
      }
  }
  return true;
}
```

Caso a função retorne um valor “false”, ou seja, a data de fim de quarentena seja anterior à data de início, é mostrada uma mensagem de erro, como é exemplificado na Figura 18, e o registo não é inserido no ficheiro de texto.

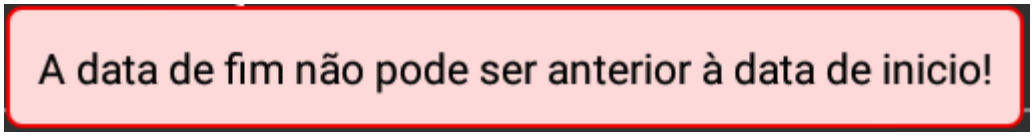

Figura 18 - Mensagem De Erro De Data De Quarentena Inválida

Se não existirem campos nulos, o novo registo vai ser guardado no ficheiro de texto. Para guardar um novo registo no ficheiro de texto foi criada a função insereRegistoTXT(String registo) que tem a linha de registo como parâmetro de entrada.

A seguir encontra-se exposto o código da função que insere um registo no ficheiro de texto.

```
public void insereRegistosTXT(String registo){

        StringBuilder buf=new StringBuilder();
        //PRIMEIRO ABRIMOS O FICHEIRO E GUARDAMOS TUDO O
QUE TEMOS LA
        try {

            InputStream in = openFileInput("ficheiro.txt");
            if (in != null) {

                InputStreamReader tmp=new
InputStreamReader(in);
                BufferedReader reader=new
BufferedReader(tmp);
                String str;

                while ((str = reader.readLine()) != null) {
                    buf.append(str + "\n");
                }

                in.close();
            }
        }
        catch (java.io.FileNotFoundException e) {
        }
        catch (Throwable t) {
        }
        //AGORA GUARDOMOS TUDO MAIS O NOVO REGISTO
        try {
                OutputStreamWriter out = new
OutputStreamWriter(openFileOutput("ficheiro.txt", 0));
                out.write(buf.toString());
                out.append(registo);
                out.close();

                toasts(1, "Registo efectuado com
sucesso.");
        }
        catch (Throwable t) {
            toasts(0, "Erro ao inserir registo.");
        }
}
```

Se o registo for inserido com sucesso, vai ser mostrada uma mensagem de sucesso “Registo inserido com sucesso!”, como representa a Figura 19.

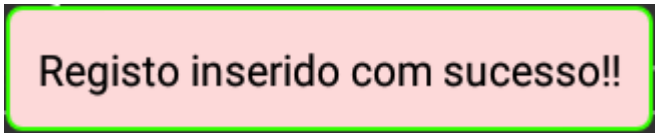

Figura 19 - Mensagem De Registo Inserido Com Sucesso

4.2.3 - Registos Efetuados

Na Figura 20, está representado o módulo Registos Efetuados onde é permitido visualizar todos os registos presentes no ficheiro de texto.

Figura 20 - Módulo De Consulta De Registos Efetuados

Para facilitar a consulta, os registos são separados em duas tabelas, uma para os registos de produção de leite e peso e outra para os registos de quarentena dos animais.

Para mostrar os registos nas tabelas foram criadas duas funções, uma para ler os registos do ficheiro de texto (lerFicheiro(), Anexo A - 3) e outra para adicionar cada um dos registos à tabela pretendida (novaLinha(String registo), Anexo A - 4). Dentro da função lerFicheiro(), para cada registo lido, é chamada a função novaLinha(String registo).

Além de permitir visualizar os registos, é também permitido apagar registos do ficheiro de texto. Para isso foi criado um botão “Apagar” para cada registo nas tabelas, que vai mostrar uma janela *popup* que pede a confirmação para eliminar o registo em causa, como é representado na Figura 21.

Figura 21 - Janela De Confirmação Para Apagar Registo

Para apagar um registo do ficheiro foi criada a função apagaRegisto(int linha), que tem como parâmetro de entrada a linha em que se encontra o registo em causa.

Esta função vai manter todos os registos contidos no ficheiro, menos o registo que pretendemos apagar.

O código da função criada para apagar um registo do ficheiro de texto encontrasse no Anexo A – 5.

No canto superior direito da janela de consulta de registos efetuados é também mostrada uma informação da quantidade de registos existentes por exportar. Esta informação está também disponível da mesma forma na página inicial da aplicação. Para fazer a contagem dos registos por exportar foi criada a aplicação contaRegistos() (Anexo A - 6).

4.2.4 - Comunicação

Para a exportação de registos efetuados para a aplicação *desktop* existem duas possibilidades, via cabo USB, ou via tecnologia Bluetooth como representa a Figura 22.

Sendo a tecnologia Bluetooth um meio de comunicação sem fios (wireless), esta é muito prática pois dispensa a utilização de um cabo como elo de ligação. Devido à possibilidade de uma exploração possuir computadores sem esta tecnologia, é necessário que a aplicação possibilite outro meio de comunicação que funcione em qualquer computador, daí a implementação de comunicação via cabo USB.

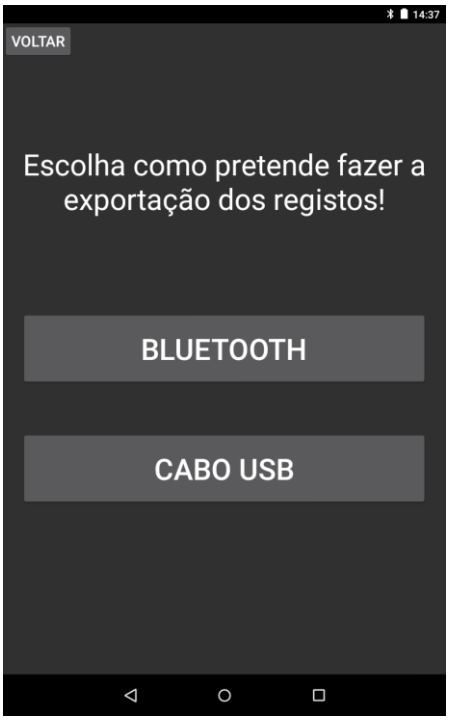

Figura 22 - Menu De Escolha De Tipo De Comunicação

Diagrama de Comunicação

Na Figura 23 está representada a forma como a aplicação *desktop* e a aplicação *Android* comunicam após se conectarem. Esta forma é aplicada tanto na comunicação via Bluetooth, como na comunicação via cabo USB.

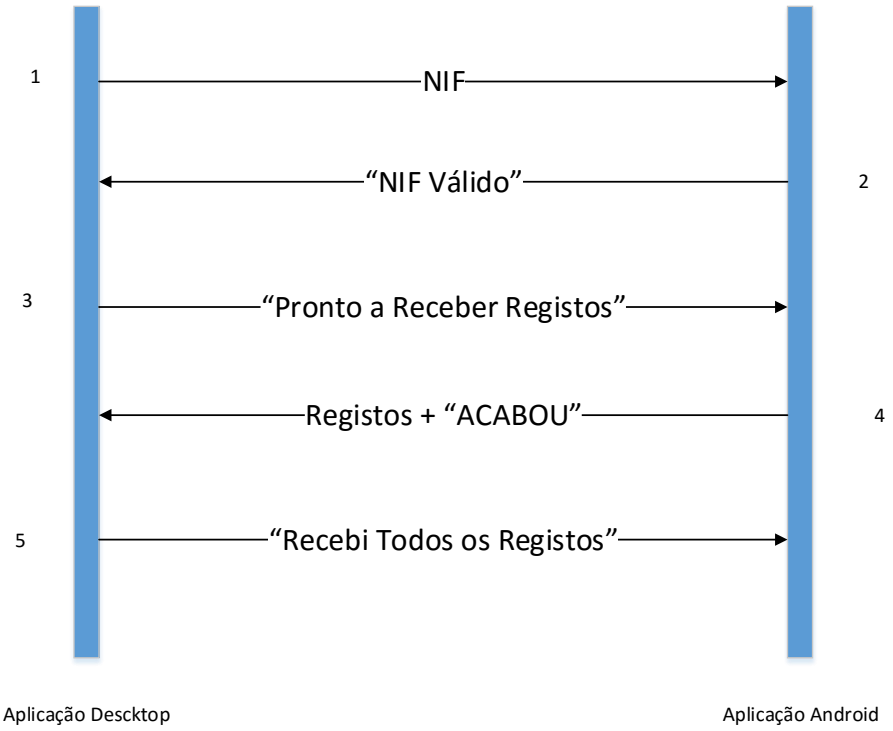

Figura 23 - Diagrama Temporal De Comunicação Entre Aplicação Android E Aplicação Desktop

A comunicação entre a aplicação *desktop* e *android* é uma comunicação com base de troca de mensagens, isto é, uma das entidades envia uma mensagem, e a entidade que recebe a mensagem, vai tratar a mensagem, e responder de acordo com a mesma.

Como podemos verificar no diagrama, a comunicação começa com o envio do NIF da exploração para a aplicação *android* que vai receber a mensagem e vai verificar se o NIF recebido corresponde ao NIF registado. Se o NIF não for igual, a aplicação *android* responde "NIF INVALIDO", e em seguida a comunicação vai ser fechada. Por

outro lado, se o NIF for igual, a aplicação responde "NIF valido", o que indica que se encontra pronta para exportar os registos.

Quando a aplicação *desktop* recebe a mensagem "NIF valido", imediatamente responde com um pedido para enviar os registos ("pronto a receber registos"). A aplicação *android* recebe esta mensagem e envia todos os registos. Envia também uma mensagem a informar que todos os registos foram enviados ("ACABOU").

Desta forma a aplicação desktop sabe que todos os registos foram recebidos, e vai fazer uma verificação dos mesmos. Se forem detetados erros nos registos, é enviada uma mensagem de erro ("erro") para a aplicação *android*. Por outro lado se não existirem erros é enviada uma mensagem de sucesso ("recebi todos os registos") para a aplicação *android*, e é concluída a comunicação.

Validação De Registos Recebidos

Devido à possibilidade de existência de falhas na comunicação, que levam à receção de registos incompletos ou errados, foi implementado um *checksum* em cada um dos registos a enviar. Este *checksum* vai garantir que todos os registos recebidos estão intactos e sem nenhuma alteração. No caso de existência de registos em que o valor de *checksum* não corresponda ao original, é enviada uma mensagem "erro" para a aplicação *android*, que posteriormente volta a enviar os registos.

O *checksum* implementado tem como base o método de encriptação MD5. Cada registo vai ser encriptado utilizando este método, e o valor correspondente a esta encriptação vai ser adicionado à linha desse mesmo registo, antes de ser enviado para a aplicação *desktop*. A Figura 24, representa uma trama de dados enviada para a aplicação *desktop*.

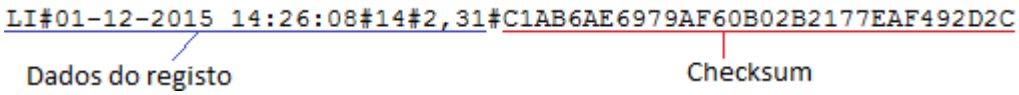

Figura 24 - Exemplo De Trama De Dados Com Checksum

Para encriptar cada registo foi criada a função getMD5(String registo), com o registo a encriptar como parâmetro de entrada. Esta função retorna o valor da encriptação do registo em causa.

O código que se segue corresponde à função getMD5(String registo) que devolve a encriptação de um registo.

```
public static String getMD5(String registo){
    MessageDigest mdEnc = null;
    try {
        mdEnc = MessageDigest.getInstance("MD5");
    } catch (NoSuchAlgorithmException e) {
        Log.v("Erro: ", "Erro ao encriptar registo");
    }
    mdEnc.update(registo.getBytes(), 0, registo.length());
    String md5 = new BigInteger(1,
mdEnc.digest()).toString(16);
    while ( md5.length() < 32 ) {
        md5 = "0"+md5;
    }
    return md5.toUpperCase();
}
```

Quando a aplicação *desktop* recebe todos os registos, passa a fazer uma validação dos mesmos. Para cada registo recebido vai ser retirado o valor de *checksum* da linha de registo, volta a encriptar-se o registo recebido e confirma-se se o valor da encriptação recebida é o mesmo que o valor da encriptação feita do lado da aplicação *desktop*. Para tal, foi criada a função validaRegistos(StringBuilder dados) (Anexo A - 7), com parâmetro de entrada uma variável *StringBuilder* que contém todos os registos recebidos, e como parâmetro de saída um valor booleano "*true*" se não existirem erros nos registos, ou "false" se existirem erros nos registos.

Comunicação Bluetooth (*Android*)

Para utilizar Bluetooth numa aplicação *Android*, é necessário adicionar essa permissão ao manifesto da aplicação. A linha a adicionar ao manifesto é a que se segue.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
```

Quando o utilizador opta por exportar os registos via Bluetooth, a aplicação faz imediatamente uma verificação se o dispositivo suporta esta tecnologia. Se este facto não se confirmar, é mostrada uma mensagem de erro "Este dispositivo não suporta Bluetooth" como é demostrado nas linhas de código a seguir.

```
btAdapter = BluetoothAdapter.getDefaultAdapter();
//SE O DISPOSITIVO NÃO SUPORTA BLUETOOTH
if (btAdapter == null) {
    toast("Este dispositivo não suporta Bluetooth");
    finish();
    return;
}
```

Caso o dispositivo suporte esta tecnologia, é feita uma verificação se o Bluetooth se encontra ligado, caso não se verifique, é pedido ao utilizador que permita a aplicação ligar o Bluetooth como representado na Figura 25. O código para esta finalidade é demostrado a seguir.

```
if (!btAdapter.isEnabled()) {

        Intent enableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);

        startActivityForResult(enableIntent,
REQUEST_ENABLE_BT);
}
```

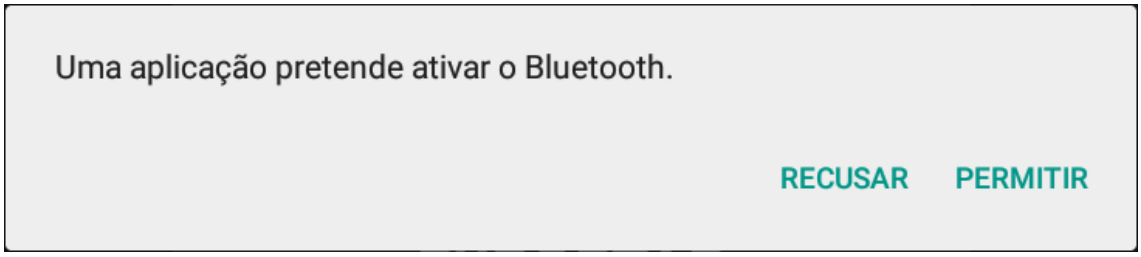

Figura 25 - Janela De Permissão De Ativação De Bluetooth

Uma vez ligado o Bluetooth do dispositivo, a aplicação inicia uma *Thread* (ServerSocketThread()) que age como servidor da ligação e fica à escuta de clientes que se queiram conectar. Neste caso o cliente é a aplicação *desktop*.

Para que duas aplicações possam comunicar, estas necessitam de um UUID, que funciona como uma "chave" de ligação entre as duas aplicações. Para este projeto foi utilizado o site UUIDGenerator[1] para gerar um UUID único. A seguir é demostrado como é utilizado o UUID na aplicação.

```
private final UUID uuid = UUID.fromString("c9f5d322-9558-
48b5-ab95-31f31eba4c44");
private BluetoothAdapter btAdapter
 = BluetoothAdapter.getDefaultAdapter();

private BluetoothServerSocket serverSocket
  = btAdapter.listenUsingRfcommWithServiceRecord(uuid);
```

Quando um cliente, neste caso a aplicação *desktop* se tenta conectar, é criado um *socket* de comunicação, e é iniciada uma nova *Thread (*AcceptThread()), que vai gerir as mensagens recebidas e enviadas através deste *socket*, como é mostrado nas linhas de código seguintes.

```
while (true) {

   try {
      socket = serverSocket.accept();
      Log.v("SOCKET", socket + "");
   }catch (IOException e) {
   break;
   }
   if (socket != null) {

      myAcceptThread = new AcceptThread(socket);
      myAcceptThread.start();
   }
}
```

Uma vez dentro da *Thred* "AcceptThread", vai ser feita uma verificação constante da existência de mensagens recebidas com o auxílio de um ciclo *while*. Caso exista uma mensagem recebida, esta vai ser tratada e será enviada uma resposta.

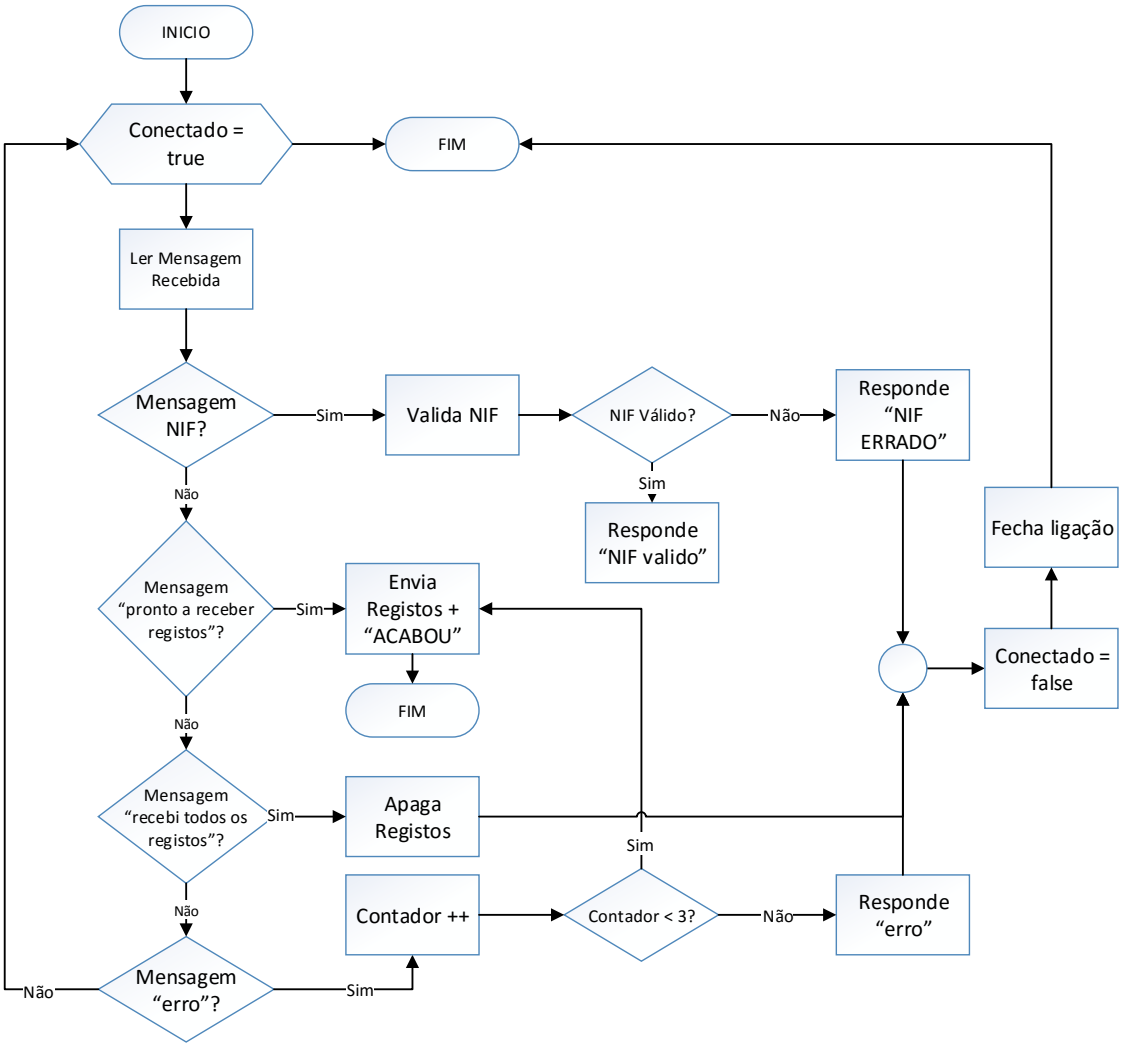

Figura 26 - Diagrama de Verificação de Mensagens Recebidas (*Android*)

Como podemos verificar no diagrama da Figura 26, quando é recebida uma mensagem, são verificadas todas possibilidades dessa mensagem. Se a mensagem for o NIF da exploração, esse valor é validado com o NIF guardado no ficheiro de texto do dispositivo *Android*, e se o NIF for igual, é enviada a resposta "NIF valido", caso contrário é enviada a resposta "NIF ERRADO", e é fechada a ligação.

Caso a mensagem recebida seja "pronto a receber registos", serão enviados os registos contidos no ficheiro de texto, juntamente com uma mensagem "ACABOU".

Se a mensagem recebida for “recebi todos os registos”, vai ser chamada a função apagaRegistos() (Anexo A - 8), que vai apagar todos os registos do ficheiro de texto, sendo de seguida fechada a ligação.

Por último, se a mensagem recebida for “erro”, é incrementado um contador, e se esse contador for menor que 3, a aplicação volta a enviar todos os registos. Caso o contador seja igual a 3, é enviada a resposta “erro”, e é fechada a ligação.

É também feita uma verificação constante do estado da ligação à aplicação *desktop*. Esta verificação é feita através da leitura de uma exceção originada dentro de um *try/catch*, como é evidenciado a seguir.

```
try{
    in = socket.getInputStream();
    bytes = in.read(buffer);
    final String strReceived = new String(buffer, 0,
bytes);
    final String msgReceived = String.valueOf(bytes);
    .
    .
    .

} catch (IOException e)
{
    e.printStackTrace();
    if (e.toString().contains("socket closed") && !acabou){
        try {
            connectedBluetoothSocket.close();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        erro = true;
    }
}
```

Quando dentro da *Thread* se tenta fazer uma leitura (`in = socket.getInputStream();`) e por alguma razão o *socket* de ligação com a aplicação *desktop* não se encontra ativo, vai ser originada uma exceção que pode ser detetada dentro do *catch*. Quando é originada esta exceção, sem que a comunicação tenha chegado ao fim, é mostrada uma mensagem de erro “Erro de comunicação!”, a ligação é fechada e a aplicação é redirecionada para a página principal, como é representado a seguir.

```
if (erro && !acabou){

    toasts("Erro de ligação!");
    ligado = false;
    cancel();

    Intent myIntent = new Intent(ExportarBluetooth.this,
Default.class);
    startActivity(myIntent);
}
```

Comunicação Bluetooth (*Desktop*)

Para utilizar a tecnologia Bluetooth na aplicação *desktop*, foi utilizada a biblioteca 32Feet[5]. Esta biblioteca pode ser descarregada do *website* oficial, fornecendo métodos e ferramentas que ajudam na utilização de Bluetooth em aplicações *desktop*.

Quando o utilizador escolhe a opção de comunicação via Bluetooth, é feita uma verificação se o computador suporta ou não esta tecnologia. Se o computador em questão não suporta Bluetooth, é mostrada uma mensagem de erro "Este computador não suporta Bluetooth", como representado na Figura 27. A seguir é apresentado o código que verifica se o dispositivo suporta *Bluetooth*.

```
if (BluetoothRadio.IsSupported)
{
     myself = new BluetoothClient();
     btnConectar.Visible = false;
     btnImportarUSB.Visible = false;
     btnImportarBluetooth.Visible = false;
     btnSelecionarDispositivo.Visible = true;
     lblDispositivo.Visible = false;
}
else {
     MessageBox.Show("Este computador não suporta
Bluetooth");
}
```

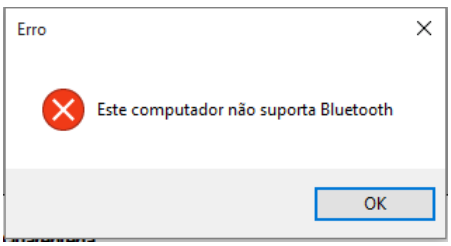

Figura 27 - Mensagem De Erro Bluetooth Não Suportado

Caso o computador suporte Bluetooth, um botão “Selecionar Dispositivo” passa a encontrar-se visível. Este botão abre uma janela onde é permitido selecionar o dispositivo *Android* do qual se pretende importar registos, como representa a Figura 28.

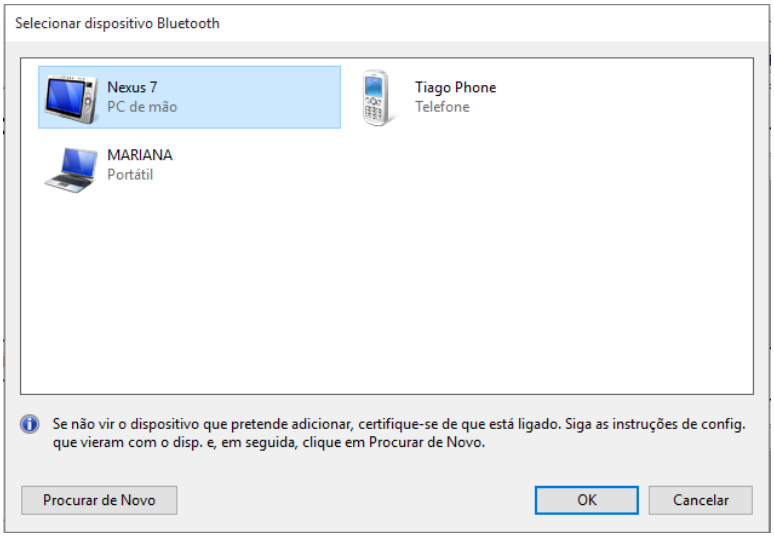

Figura 28 - Janela De Escolha De Dispositivo Com Que Se Pertende Comunicar

Uma vez selecionado o dispositivo pretendido, é feita uma tentativa de conexão ao mesmo. Se a conexão for bem-sucedida, é enviado o NIF da exploração para iniciar a comunicação. Para obter o NIF foi criada uma função nif(), que faz uma leitura à base de dados, e devolve o NIF da exploração. As linhas de código que se seguem representam a função utilizada para conexão à aplicação *Android*.

```
private void conectar()
{
    try
    {
        if (bTServerDevice != null)
        {
                myself.BeginConnect(bTServerDevice.Device
                Address,        uuid, new
                AsyncCallback(callback), "NIF"+nif());
        }
    }
    catch (Exception e)
    {
    Console.WriteLine(e);
    }
}
```

Quando iniciamos a comunicação, é também iniciada uma *Thread* (clientThread()), que faz a leitura de mensagens recebidas e responde adequadamente a cada mensagem.

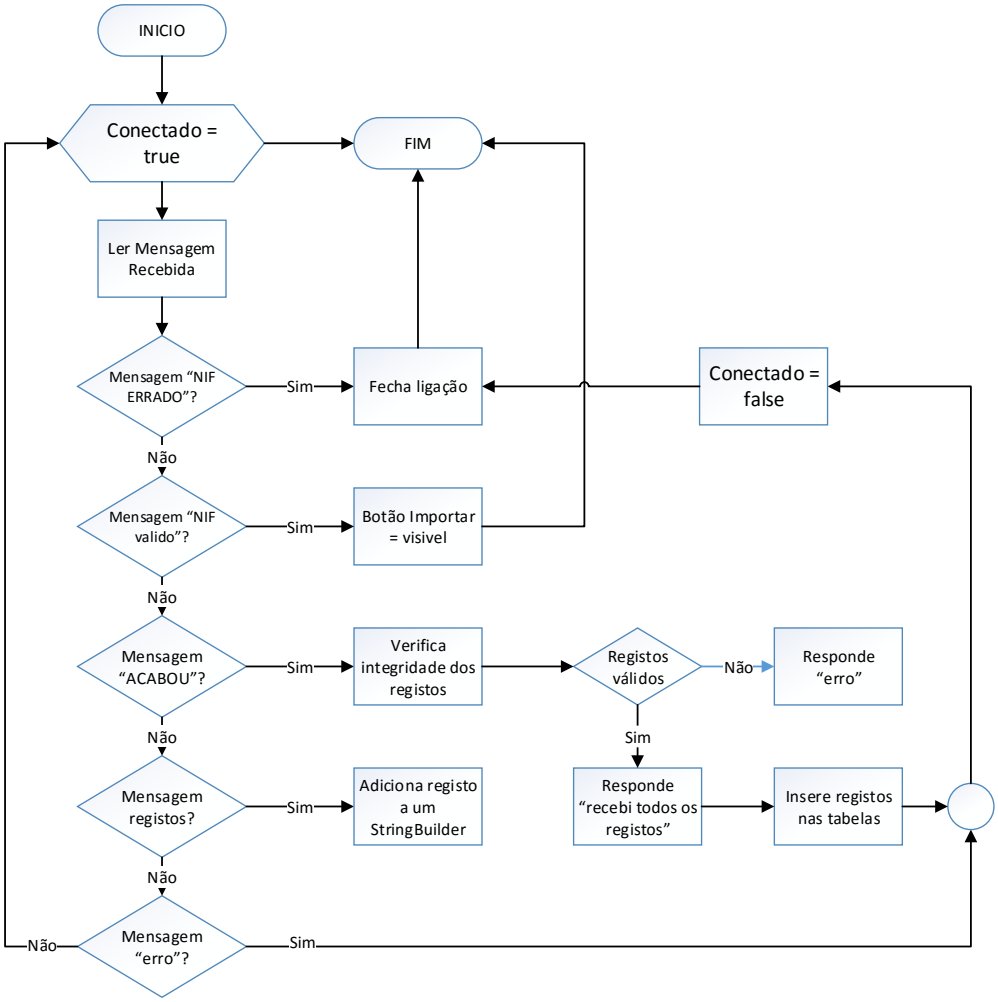

Figura 29 - Diagrama de Verificação de Mensagens Recebidas (*Desktop*)

Como podemos verificar no diagrama da Figura 29, à semelhança do que é feito na aplicação *Android*, também na aplicação *desktop* é feita uma leitura constante de mensagens recebidas, que são verificadas e tratadas adequadamente.

Se a mensagem recebida for "NIF ERRADO", a ligação é fechada.

No caso de a mensagem recebida seja "NIF valido", o botão Importar passa a um estado visível.

Caso a mensagem recebida seja "ACABOU", vai ser feita uma verificação da integridade dos registos recebidos, se estes forem validados, vai ser enviada a resposta "recebi todos os registos", caso contrário é enviada a resposta "erro".

Se a mensagem recebida conter registos importados da aplicação *Android*, esses registos vão ser adicionados a uma variável *StringBuilder*.

Por último, se a mensagem recebida for "erro", vai ser fechada a ligação.

Também na aplicação desktop é feita uma verificação constante do estado da ligação utilizando um *try/catch*. Quando dentro da *Thread* clientThread() se tenta ler uma mensagem quando esta não existe é gerada uma exceção que vai ser tratada dentro do *catch*. Para impedir que a *Thread* fique bloqueada, à espera de uma mensagem, foi implementado um tempo limite de espera, como é exemplificado a seguir:

```
stream = myself.GetStream();
stream.ReadTimeout = 4000;
```

Para impedir que esta exceção seja originada, em situações que não afetam o funcionamento da comunicação, é feita uma validação em que só é gerado um erro quando a aplicação *Android* e a aplicação *desktop* estão a comunicar. De seguida, encontra-se exemplificado o código que origina esta exceção.

```
try
{
    //LER OS DADOS A RECEBER
    byte[] myReadBuffer = new byte[1024];
    StringBuilder myCompleteMessage = new StringBuilder();
    int numberOfBytesRead = 0;

    numberOfBytesRead = stream.Read(myReadBuffer, 0,
myReadBuffer.Length);
    myCompleteMessage.AppendFormat("{0}",
Encoding.ASCII.GetString(myReadBuffer, 0,
numberOfBytesRead));
    .
    .
    .

catch (Exception e)
{
    if (ligado)
    {
        MessageBox.Show("Erro de ligação", "Erro",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        lblStatus.Invoke((MethodInvoker)(() =>
lblStatus.Text = "Status: Não Conectado"));
        myself.Close();
    }
}
```

Comunicação USB

Para a comunicação via cabo USB foram estudadas duas possibilidades. A primeira possibilidade consiste na criação de um novo *driver* para a porta USB do dispositivo *Android***,** com o objetivo de emular uma porta serie. Desta forma, a informação entre as duas aplicações é feita por portas serie. A segunda possibilidade passa pela utilização do ADB, que funciona como uma ponte de comunicação com o dispositivo *Android***.** O ADB é uma ferramenta de linha de comandos para comunicação com dispositivos *Android* físicos ou emulados.

Para este projeto foi determinado que a melhor forma de comunicação via cabo USB passa pela utilização da ferramenta ADB, uma vez que esta solução não necessita da criação de uma nova *driver* para a porta USB do dispositivo *Android*.

Esta ferramenta está contida no SDK do *Android*, que por sua vez é instalado em conjunto com o IDE Android Studio. Caso a ferramenta não esteja instalada, pode ser descarregada do site oficial *Android Developers*[2].

Para comunicar utilizando o cabo USB via ADB vamos utilizar o comando *forward* do ADB. Este comando vai impor que um certo porto do computador comunique com outro porto do dispositivo *Android.* Para tal, é necessário abrir a linha de comandos e introduzir o comando mostrado a seguir.

```
Adb forward tcp:6000 tcp:6000
```

Depois de executado este comando, a aplicação *Android* pode comunicar através do porto 6000, com o porto 6000 do computador.

Para verificar se o comando "*forward"* foi bem executado, podemos utilizar o comando "adb forward –list", que devolve os dispositivos ligados ao computador, passando estes a ter esta configuração. Na imagem seguinte podemos ver esta sequência de comandos. Na Figura 30, está representada a execução dos comandos referidos anteriormente.

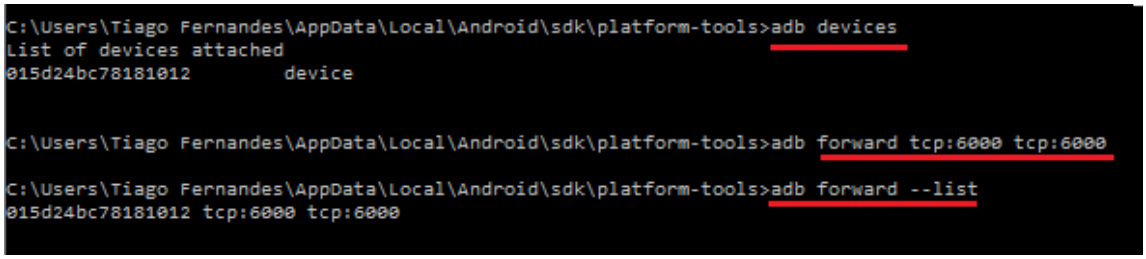

Figura 30 - Comando forward da Ferramenta ADB

Comunicação USB (Android)

Para utilizar a porta USB com comunicação via *sockets*, é necessário adicionar uma permissão ao manifesto da aplicação, pois embora pequena, é criada uma rede entre o dispositivo *Android* e o computador. Esta permissão está descrita na linha a seguir.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Como na comunicação via *Bluetooth*, quando o utilizador opta por comunicar via USB, é iniciada uma *Thread* que vai agir como servidor da comunicação e fica à espera no porto 6000 (porto definido quando é utilizado o comando *forward* do ADB) que um cliente se ligue para comunicar. As linhas de código seguintes exemplificam a criação do servidor no porto especificado.

```
public class SocketServerThread extends Thread {

        server = new ServerSocket(6000);
        socket = server.accept();

        .
        .
        .
}
```

Uma vez ligado um cliente, que neste contexto é a aplicação *desktop*, vai ser iniciada uma troca de mensagens que, à semelhança da comunicação via *Bluetooth*, vão ser tratadas dentro de um ciclo *while*.

Como referido anteriormente, a troca de mensagens entre a aplicação *Android* e a aplicação *desktop* é a mesma para a comunicação *Bluetooth*, e para a comunicação via cabo USB. Desta forma o ciclo *while*, que trata da troca de mensagens para a comunicação via USB, é idêntico ao ciclo representado na Figura 26.

Também à semelhança do que é feito na comunicação via Bluetooth, é utilizado um *try/catch* para detetar erros na comunicação, como por exemplo se o cabo é desligado a meio da mesma. Quando se tenta ler uma mensagem recebida e a ligação via *sokets* não se encontra estabelecida, é gerada uma exceção que é originada dentro do *catch.* As linhas de código que se seguem mostram como é tratada esta exceção.

```
try {
    inputText = in.readLine();
}catch (Exception e){
    if (!fim) {
        ExportarUSB.this.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            toasts("Erro de comunicação!");
            Intent myIntent = new Intent(ExportarUSB.this,
Default.class);
            startActivity(myIntent);

            try {
                server.close();
            } catch (IOException e) {
            e.printStackTrace();
            }
        }
        });
    }
}
```

Quando é detetada a exceção referida anteriormente, é mostrada uma mensagem de erro “Erro de comunicação.”, sendo fechada a ligação e a aplicação redirecionada para a página principal.

Comunicação USB (*Desktop*)

Como foi referido anteriormente, para comunicar via cabo USB é necessário o comando "f*orward"* do ADB para que um porto do dispositivo móvel comunique com um outro porto do computador. Assim, quando o utilizador opta por comunicar via USB, é executada uma série de linhas de código (Anexo B - 2) em que a linha de comandos é aberta e é executado este comando, tudo de forma automática. Para verificar se o comando é executado corretamente, é também executado o comando "*adb forward –list*", que devolve os dispositivos com a nova configuração. Se a resposta deste último comando não retornar nenhum dispositivo com a configuração pretendida, é detetado que o dispositivo *Android* não se encontra ligado via cabo USB e é mostrada uma mensagem de erro como é demostrado a seguir.

No caso de existir um dispositivo ligado com a configuração pretendida, é assumido que o comando foi executado corretamente e é iniciada a ligação com o dispositivo *Android*. Para este efeito, foi criada a função iniciaComunicacaoUSB() que vai criar um novo *socket* de ligação no porto 6000 (pois é o porto utilizado no comando *forward*), e é enviada a primeira mensagem – sendo esta o NIF da exploração.

Para criar este novo *socket*, é utilizado o IP 127.0.0.1, sendo este o endereço referente ao *localhost* do computador.

Dentro da função iniciaComunicacaoUSB(), é também iniciada uma nova *Thread*, ThreadUSB(), que vai ler as mensagens recebidas via USB e responder adequadamente.

Para fazer a leitura das mensagens recebidas, é utilizado um ciclo *while* à semelhança do que é feito na comunicação Bluetooth. Este ciclo é idêntico ao referido na comunicação via *Bluetooth*, que está representado na Figura 29.

4.3 - Aplicação Desktop

Foi criado um novo módulo na já existente aplicação Anima, para que esta possa comunicar e receber os registos da aplicação AnimaMobile.

Figura 31 - Modulo de Receção de Registos da Aplicação Anima

Na figura 31 está ilustrado o *layout* do módulo criado. Como é representado na figura, existem dois botões para a escolha do tipo de comunicação com a aplicação *Android*, "USB", "Bluetooth". Existe também uma *label* que indica o estado da ligação com a aplicação móvel, "Conectado" ou "Não conectado". Foram também introduzidas duas tabelas em que são inseridos os registos recebidos.

4.3.1 - Editar Registos

Uma vez recebidos e validados os registos da aplicação *Android*, estes são inseridos em duas tabelas: uma para os registos de peso e produção de leite e outra para os registos de quarentena, como representado na Figura 32. Estas tabelas permitem editar ou apagar registos pretendidos.

Figura 32 - Tabelas de Registos Recebidos

Para inserir os registos nas tabelas, foi criada a função insereRegistosGrid(StringBuilder registos) (Anexo A - 9), que tem como parâmetro de entrada uma variável *StringBuilder* que contém todos os registos. Esta função vai tratar a informação de cada registo e inserir essa informação na tabela adequada.

Depois de inseridos os registos nas tabelas, o utilizador pode alterar a informação dos mesmos ou até apagar um registo pretendido. Para apagar um registo foi adicionado um botão "Apagar" a cada linha da tabela que contém um registo. Quando o utilizador pretende apagar um registo e clica no botão "Apagar", é mostrada uma janela que pede a confirmação do utilizador, como representa a Figura 33. Caso este confirme a operação, a linha da tabela que contém o registo pretendido é apagada da tabela. Nas linhas que se seguem, é apresentado o código necessário para apagar um registo de uma tabela.

```
private void dgvRegistos_CellClick(object sender,
DataGridViewCellEventArgs e)
{
        if (e.ColumnIndex == 4)
        {
            if (MessageBox.Show("Pretende mesmo apagar este
registo?", "Warning", MessageBoxButtons.YesNo) ==
System.Windows.Forms.DialogResult.Yes)
            {
                int x = dgvRegistos.CurrentCell.RowIndex;
                dgvRegistos.Rows.RemoveAt(x);
            }
        }
}
```

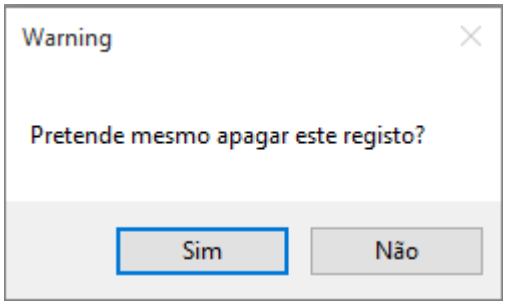

Figura 33 - Mensagem de Coonfirmação Para Apagar Registo da Tabela

Quando o utilizador acaba de alterar ou apagar os registos nas tabelas, estes podem ser inseridos na base de dados utilizando o botão "Validar Registos". Este botão vai executar a função insereBD()(Anexo A - 10). Esta função vai correr cada linha das tabelas, e, de acordo com a informação dessa linha vai inserir um novo registo na tabela adequada da base de dados.

Caso a função tente inserir um registo em que a identificação de um animal não exista na base de dados, é gerado um erro de aviso a informar o utilizador como representado na Figura 34, e esse registo não é inserido. Este erro é detetado quando ocorre uma exceção dentro de um *try/catch*.

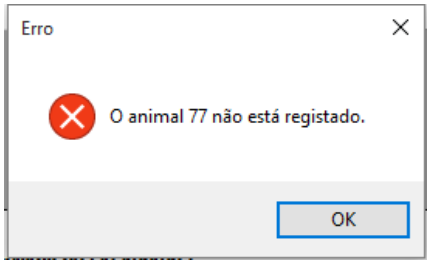

Figura 34 - Mensagem de Erro: Animal Não Exisente na Base de Dados

A função insereBD() deteta também quando se tenta inserir um registo em que a identificação de um grupo de animais esteja incorreta. Para tal é utilizada a função gruposAnimais() que devolve um vetor com a identificação de todos os grupos de animais presentes na base de dados. Assim, quando se pretende inserir um registo em que a identificação do grupo não se encontra no vetor retornado pela função, é mostrada uma mensagem de erro (Figura 35), e esse registo não é inserido na base de dados.

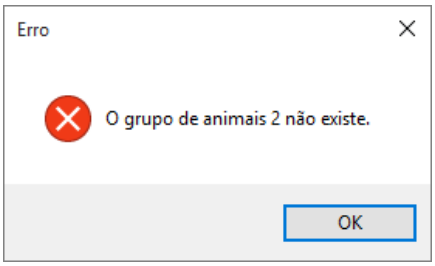

Figura 35 - Mensagem de Erro: Grupo de Animais Não Existente na Base de Dados

Caso não existam erros, é mostrada uma mensagem de sucesso, que informa a quantidade de registos que foram inseridos na base de dados como é ilustrado na Figura 36.

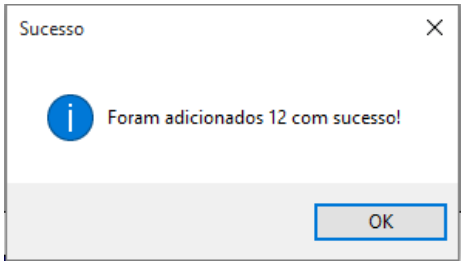

Figura 36 - Mensagem de Sucesso de Registos Inseridos na Base de Dados

4.3.2 - Base de dados

Para este projeto foi criada uma base de dados nova, com todas as tabelas necessárias para o bom funcionamento da aplicação.

Esta base de dados vai guardar todos os registos provenientes da aplicação móvel, bem como o NIF da exploração, e os animais existentes nessa mesma exploração.

A base de dados foi criada em *MySQL* com o auxilio da aplicação SQL Manager[6], que contem ferramentas muito praticas para criação e manutenção da mesma.

Para fazer a ligação da base de dados à aplicação desktop é utilizado o servidor MiniServX.

A Figura 37, represente o diagrama entidade relacionamento da base de dados criada para este projeto.

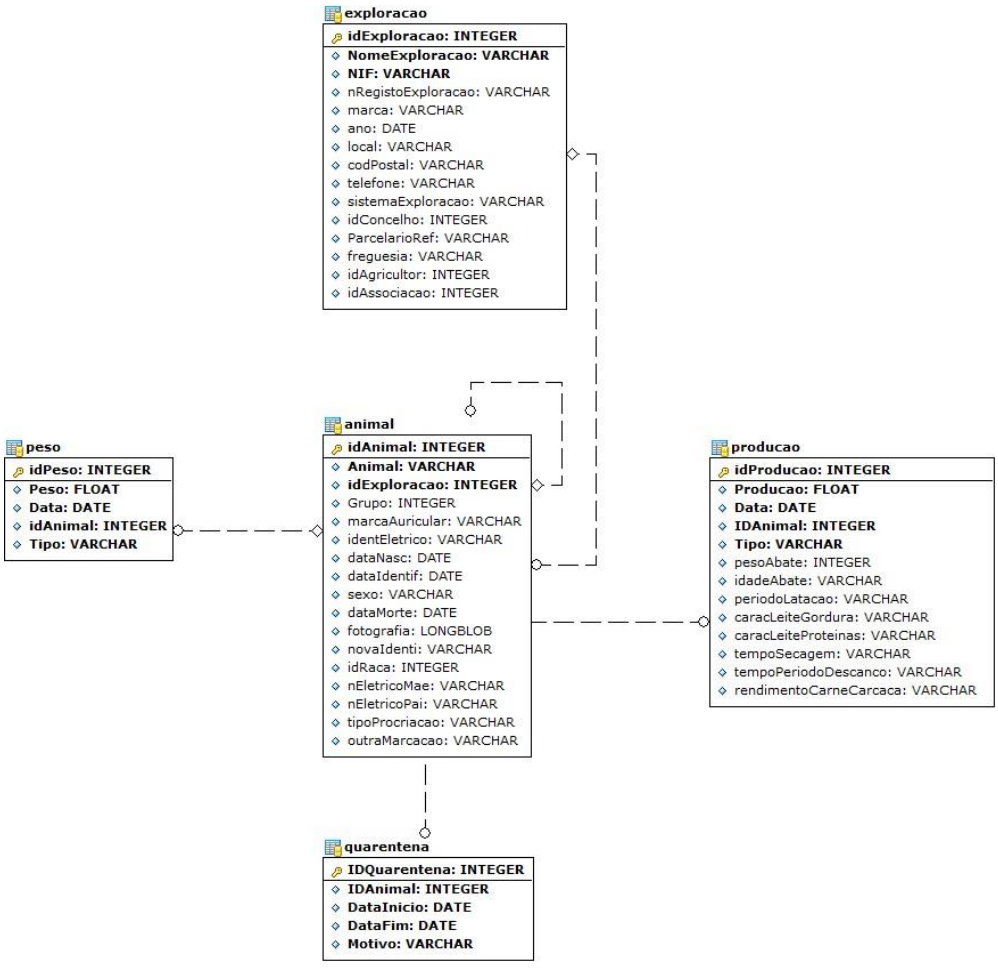

Figura 37 - Modelo Entidade Relacionamento da Base de Dados Criada

Dicionário de Dados

As tabelas que se seguem fazem referência a todos os campos das tabelas da base de dados criada.

Na Tabela 1 estão descritos todos os campos presentes na classe animal.

Tabela 1 - Entidade Animal

Table: animal

Fields

Name	Type	Not Null	Unique	P/K	A/I	Binary
idAnimal	int(11)	Not Null		Yes	Yes	
Animal	varchar(20)	Not Null				
idExploracao	int(11)	Not Null				
Grupo	int(11)					
marcaAuricular	varchar(20)					
identEletrico	varchar(20)					
dataNasc	date					
dataIdentif	date					
sexo	varchar(20)					
dataMorte	date					
fotografia	longblob					
novaIdenti	varchar(20)					
idRaca	int(11)					
nEletricoMae	varchar(20)					
nEletricoPai	varchar(20)					
tipoProcriacao	varchar(20)					
outraMarcacao	varchar(20)					

Indices

Index Name	On Field	Unique	Full Text	Sorting
PRIMARY	`idAnimal`	Yes		Ascending
animal_fk	`idExploracao`			Ascending
animal_fk1	`Grupo`			Ascending

Na Tabela 2 estão descritos todos os campos presentes na classe exploração.

Tabela 2 - Entidade Exploração

Table: exploracao

Fields

Name	Type	Not Null	Unique	P/K	A/I	Binary
idExploracao	int(11)	Not Null		Yes	Yes	
NomeExploracao	varchar(20)	Not Null				
NIF	varchar(20)	Not Null	Yes			
nRegistoExploracao	varchar(20)					
marca	varchar(20)					
ano	date					
local	varchar(20)					
codPostal	varchar(20)					
telefone	varchar(20)					
sistemaExploracao	varchar(20)					
idConcelho	int(11)					
ParcelarioRef	varchar(20)					
freguesia	varchar(20)					
idAgricultor	int(11)					
idAssociacao	int(11)					

Indices

Index Name	On Field	Unique	Full Text	Sorting
PRIMARY	`idExploracao`	Yes		Ascending
NIF	`NIF`	Yes		Ascending

Na Tabela 3 estão descritos todos os campos presentes na classe produção.

Tabela 3 - Entidade Produção

Table: producao

Fields

Name	Type	Not Null	Unique	P/K	A/I	Binary
idProducao	int(11)	Not Null		Yes	Yes	
Producao	float(9,3)	Not Null				
Data	date	Not Null				
IDAnimal	int(11)	Not Null				
Tipo	varchar(20)	Not Null				
pesoAbate	int(11)					
idadeAbate	varchar(20)					
periodoLatacao	varchar(20)					
caracLeiteGordura	varchar(20)					
caracLeiteProteinas	varchar(20)					
tempoSecagem	varchar(20)					
tempoPeriodoDescanco	varchar(20)					
rendimentoCarneCarcaca	varchar(20)					

Indices

Index Name	On Field	Unique	Full Text	Sorting
PRIMARY	`idProducao`	Yes		Ascending
producao_fk	`IDAnimal`			Ascending

Na Tabela 4 estão descritos todos os campos presentes na classe peso.

Tabela 4 - Entidade Peso

Table: peso

Fields

Name	Type	Not Null	Unique	P/K	A/I	Binary
idPeso	int(11)	Not Null		Yes	Yes	
Peso	float(9,3)	Not Null				
Data	date	Not Null				
idAnimal	int(11)	Not Null				
Tipo	varchar(20)	Not Null				

Indices

Index Name	On Field	Unique	Full Text	Sorting
PRIMARY	`idPeso`	Yes		Ascending
peso_fk	`idAnimal`			Ascending

Na Tabela 5 estão descritos todos os campos presentes na classe quarentena.

Tabela 5 - Entidade Qurentena

Table: quarentena

Fields

Name	Type	Not Null	Unique	P/K	A/I	Binary
IDQuarentena	int(11)	Not Null		Yes	Yes	
IDAnimal	int(11)	Not Null				
DataInicio	date	Not Null				
DataFim	date	Not Null				
Motivo	varchar(50)	Not Null				

Indices

Index Name	On Field	Unique	Full Text	Sorting
PRIMARY	`IDQuarentena`	Yes		Ascending
quarentena_fk	`IDAnimal`			Ascending

Description

InnoDB free: 8192 kB; (`IDAnimal`) REFER `anima_db/animal`(`IDAnimal`)

Capítulo 5

Testes e Resultados

Foram realizados alguns testes à aplicação criada com o intuito de estudar a fiabilidade da aplicação.

O primeiro teste consiste na exportação de diferentes quantidades de registos e verificar se são detetados erros quando esses registos são recebidos pela aplicação desktop. A Tabela 6 faz referência aos resultados obtidos neste teste.

Tabela 6 - Testes de Erros nos Registos Recebidos

Número de Registos Exportados	Tipo de Comunicação	Erro	Resultado esperado?
10	Bluetooth	NÃO	SIM
20	Bluetooth	NÃO	SIM
50	Bluetooth	NÃO	SIM
100	Bluetooth	NÃO	SIM
10	Cabo USB	NÃO	SIM
20	Cabo USB	NÃO	SIM
50	Cabo USB	NÃO	SIM
100	Cabo USB	NÃO	SIM

Como se pode visualizar na Tabela 6, não foram detetados erros em todos os testes que foram realizados.

No segundo teste são provocados erros na comunicação entre as duas aplicações com o intuito de saber se as aplicações detetam estes erros, e respondem corretamente.

Quanto à comunicação via Bluetooth os testes foram feitos a uma distância média de 10m com uma parede como obstáculo.

Todos os testes foram elaborados *indoor*.

Tabela 7 - Testes de Falhas na Comunicação

Tipo de teste	Erro detetado pela aplicação Android	Aplicação Android Age Corretamente	Erro detetado pela aplicação Desktop	Aplicação Desktop Age Corretamente	Resultado Esperado
Retirar cabo USB a meio de comunicação	SIM	SIM	SIM	SIM	SIM
Desligar Blutooth a meio de uma comunicação	SIM	SIM	SIM	SIM	SIM
Adicionar um registo com checksum errado aos registos exportados	SIM	SIM	SIM	SIM	SIM

Como é visível na Tabela anterior, Tabela 7, tanto a aplicação *Android*, como a aplicação desktop detetam os erros provocados e agem corretamente para resolver os mesmos.

Capítulo 6

Conclusão e Trabalho Futuro

No início deste projeto foi feito um estudo para averiguar qual o sistema operativo móvel que mais se adequa a este tipo aplicação. Devido ao custo significativamente mais baixo dos equipamentos *Android* foi decidido que este era o sistema adequado a utilizar.

Após algumas reuniões com a empresa Explorinova, foram estipulados alguns objetivos essenciais para o bom funcionamento da aplicação desenvolvida. Foi também feita uma estruturação de etapas para a realização do desenvolvimento do projeto, que se provaram fundamentais para o sucesso do mesmo.

Foram aparecendo alguns problemas relacionados com a base de dados criada para a aplicação desktop que levaram a um atraso na realização do projeto. Todos estes problemas foram resolvidos.

Ao longo do desenvolvimento do projeto foram alterados e melhorados alguns métodos implementados no mesmo, que levaram a uma aplicação final bem estorturada, e muito perto da sua versão final.

Na conclusão do projeto foram efetuados alguns testes à aplicação desenvolvida para verificar a fiabilidade da mesma. Estes testes comprovam o sucesso do desenvolvimento da aplicação, não tendo sido detetada qualquer anomalia.

Podemos concluir que todos os objetivos estipulados foram alcançados.

O sistema desenvolvido neste projeto é uma ótima solução para o problema proposto. Embora facilite muito o trabalhos de recolha dos dados pretendidos, o utilizador continua a ser obrigado a introduzir estes mesmos dados na aplicação móvel para posteriormente exportar para a aplicação *desktop*.

Uma melhoria possível para este sistema passa pela implementação de sistemas embebidos que façam a recolha automática dos dados pretendidos como peso ou produção de leite. Estes sistemas iriam fazer a recolha desses dados que imediatamente enviam via Bluetooth para o dispositivo móvel. Desta forma o utilizador apenas necessita de acionar a recolha de dados no dispositivo móvel, e posteriormente fazer a exportação desses registos para a aplicação móvel.

Um sistema embebido que faria a recolha de dados e envia estes mesmos dados diretamente para a aplicação *desktop* está fora de questão pois as explorações vão ter espaços físicos diferentes para a recolha de dados onde se encontram os animais, e o local onde se encontra o computador *desktop*. Esta distância origina uma serie de complicações na comunicação entre os sistemas embebidos e o computador desktop pois vão existir obstáculos no meio como paredes ou árvores, a distancia pode ser demasiado longa para se conseguir efetuar uma ligação, entre outros. Desta forma a aplicação móvel torna-se indispensável, uma vez que atua como elo de ligação entre os sistemas embebidos que recolhem os dados dos animais, e o computador desktop.

Bibliografia

[1] **"Online UUID Generator". *UUID Generator.* [Online] https://www.uuidgenerator.net/.**

[2] Android Studio. *Android Developers.* [Online] http://developer.android.com/tools/studio/index.html.

[3] Aplicação R.campo. *R.campo.* [Online] https://rcampo.ruralbit.com/index.php.

[4] Aplicação tambero.com. *tambero.com.* [Online] https://www.tambero.com/pt.

[5. Biblioteca 32Feet. *32Feet.* [Online] https://32feet.codeplex.com/.

[6] SQL Manager. *SQLManager.net.* [Online] http://www.sqlmanager.net/products/mysql/manager/.

Anexos

Anexo A - Funções

Anexo A – 1 – Função para Validar NIF

```
public boolean validaNIF(String nif){

    if (nif.length()!=9){
        return false;
    }

    int control;
    int []digit = {
            Character.getNumericValue(nif.charAt(0)),
            Character.getNumericValue(nif.charAt(1)),
            Character.getNumericValue(nif.charAt(2)),
            Character.getNumericValue(nif.charAt(3)),
            Character.getNumericValue(nif.charAt(4)),
            Character.getNumericValue(nif.charAt(5)),
            Character.getNumericValue(nif.charAt(6)),
            Character.getNumericValue(nif.charAt(7)),
            Character.getNumericValue(nif.charAt(8)),
    };

    int x = 2;
    int soma=0;
    for (int i = 7; i > -1; i--){
        soma = soma + digit[i] * x;
        x++;
    }
    if (soma%11 == 0 || soma%11 == 1){
        control = 0;
    }else{
        control = 11- soma%11;
    }

    if (control == digit[8]){
        return true;
    }else {
        return false;
    }
}
```

Anexo A – 2 – Função para Mostrar Mensagens Personalizadas

```
public void toasts(int tipo, String text){
    //0 - VERMELHO
    //1 - VERDE
    switch (tipo){
        case 0:

            LayoutInflater inflaterErro=getLayoutInflater();
            View viewToastErro =
inflaterErro.inflate(R.layout.toast_erro, null);

            TextView texto = (TextView)viewToastErro .
findViewById(R.id.tvMSG);
            texto.setText(text.toString());

            Toast toastErro = new Toast(getApplicationContext());
            toastErro.setView(viewToastErro);
            toastErro.setGravity(Gravity.CENTER_HORIZONTAL |
Gravity.CENTER_VERTICAL,0, 0);
            toastErro.setDuration(Toast.LENGTH_SHORT);
            toastErro.show();

            break;

        case 1:

            LayoutInflater inflaterSucesso=getLayoutInflater();
            View viewToastSucesso =
inflaterSucesso.inflate(R.layout.toast_sucesso, null);

            TextView textoSucesso = (TextView)viewToastSucesso .
findViewById(R.id.tvMSG);
            textoSucesso.setText(text.toString());

            Toast toastSucesso = new Toast(getApplicationContext());
            toastSucesso.setView(viewToastSucesso);
            toastSucesso.setGravity(Gravity.CENTER_HORIZONTAL |
Gravity.CENTER_VERTICAL,0, 0);
            toastSucesso.setDuration(Toast.LENGTH_SHORT);
            toastSucesso.show();

            break;
    }
}
```

Anexo A – 3 – Função Para Ler Ficheiro

```
public void lerFicheiro() {

    try {

        InputStream in = openFileInput("ficheiro.txt");
        StringBuilder buf = new StringBuilder();
        if (in != null) {
            InputStreamReader tmp = new InputStreamReader(in);
            BufferedReader reader = new BufferedReader(tmp);
            String str;

            int count = 0;


            while ((str = reader.readLine()) != null) {

                if (count > 0) {
                    buf.append(str);
                    novaLinha(str, count);

                }
                count++;
            }
            in.close();
            txtnumeroRegistos.setText(count - 1 + " Registos por
exportar.");
            numRegistos = count - 1;


        }
    } catch (java.io.FileNotFoundException e) {
    } catch (Throwable t) {
        Toast
                .makeText(this, "Exception: " + t.toString(),
Toast.LENGTH_LONG)

                .show();
    }
}
```

Anexo A – 4 – Função novaLinha

```
    public void novaLinha(String dados, int id) {

String[] aux = dados.split("#");
if (!aux[0].equals("@QI")) {

    if (aux[0].equals("@LI")){
        aux[0]=" Produção individual";
    }else
    if (aux[0].equals("@LG")){
        aux[0]=" Produção grupo";
    }else
    if (aux[0].equals("@PI")){
        aux[0]=" Peso individual";
    }else
    if (aux[0].equals("@PG")){
        aux[0]=" Peso grupo";
    }



    TableRow linhaRegisto = new TableRow(this);

    TextView cellData = new TextView(this);
    cellData.setWidth(260);
    cellData.setHeight(50);
    cellData.setTextSize(20);
    cellData.setText(aux[1]);
    cellData.setVisibility(View.VISIBLE);

    TextView cellTipo = new TextView(this);
    cellTipo.setWidth(170);
    cellTipo.setHeight(95);
    cellTipo.setText(aux[0].substring(1, aux[0].length()));
    cellTipo.setTextSize(20);
    cellTipo.setVisibility(View.VISIBLE);

    TextView cellIDAnimal = new TextView(this);
    cellIDAnimal.setWidth(100);
    cellIDAnimal.setHeight(95);
    cellIDAnimal.setTextSize(20);
    cellIDAnimal.setText(aux[2]);
    cellIDAnimal.setGravity(Gravity.CENTER);
    cellIDAnimal.setVisibility(View.VISIBLE);

    TextView cellQuantidade = new TextView(this);
    cellQuantidade.setWidth(150);
    cellQuantidade.setHeight(50);
    cellQuantidade.setTextSize(20);

    if (aux[0].equals("@PG") || aux[0].equals("@PI")) {
        cellQuantidade.setText(aux[3] + " Kg");
    } else {
        cellQuantidade.setText(aux[3] + " L");
    }

    cellQuantidade.setVisibility(View.VISIBLE);
```

```
            linhaRegisto.addView(cellData);
            linhaRegisto.addView(cellTipo);
            linhaRegisto.addView(cellIDAnimal);
            linhaRegisto.addView(cellQuantidade);

            final Button btn = new Button(this);
            btn.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {

                    int linha = btn.getId();
                    initiatePopupWindow(linha);
                }
            });

            btn.setId(Integer.valueOf(id));

            btn.setText("Apagar");
            linhaRegisto.addView(btn);

            table.addView(linhaRegisto);
        }else{

            //DADOS DA TABELA DE QUARENTENA-------------------------------
-----
            TableRow linhaRegistoQuarentena = new TableRow(this);

            TextView cellData = new TextView(this);
            cellData.setWidth(270);
            cellData.setHeight(50);
            cellData.setTextSize(20);
            cellData.setText(aux[1]);
            cellData.setVisibility(View.VISIBLE);

            TextView cellID = new TextView(this);
            cellID.setWidth(110);
            cellID.setHeight(50);
            cellID.setText(aux[2]);
            cellID.setGravity(Gravity.CENTER);
            cellID.setTextSize(20);

            cellID.setVisibility(View.VISIBLE);

            TextView cellDataInicio = new TextView(this);
            cellDataInicio.setWidth(160);
            cellDataInicio.setHeight(50);
            cellDataInicio.setTextSize(20);
            cellDataInicio.setText(aux[3]);
            cellDataInicio.setVisibility(View.VISIBLE);

            TextView cellDataFim = new TextView(this);
            cellDataFim.setWidth(160);
            cellDataFim.setHeight(50);
            cellDataFim.setTextSize(20);
            cellDataFim.setText(aux[4]);
            cellDataFim.setVisibility(View.VISIBLE);

            TextView cellDescricao = new TextView(this);
            cellDescricao.setWidth(260);
            cellDescricao.setHeight(80);
            cellDescricao.setTextSize(20);
```

```
        cellDescricao.setText(aux[5]);
        cellDescricao.setMovementMethod(new
ScrollingMovementMethod());
        cellDescricao.setVisibility(View.VISIBLE);

        //linhaRegistoQuarentena.addView(cellData);
        linhaRegistoQuarentena.addView(cellID);
        linhaRegistoQuarentena.addView(cellDataInicio);
        linhaRegistoQuarentena.addView(cellDataFim);
        linhaRegistoQuarentena.addView(cellDescricao);

        final Button btn = new Button(this);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                int linha = btn.getId();
                initiatePopupWindow(linha);
            }
        });

        btn.setId(Integer.valueOf(id));

        btn.setText("Apagar");
        linhaRegistoQuarentena.addView(btn);

        tableQuarentena.addView(linhaRegistoQuarentena);
    }
}
```

Anexo A - 5 – Função Para Apagar um Registo do Ficheiro de Texto

```
public void apagRegisto(int linha) {
    StringBuilder buf = new StringBuilder();
    //PRIMEIRO ABRIMOS O FICHEIRO E GUARDAMOS TUDO MENOS O REGISTO EM
CAUSA
    try {
        InputStream in = openFileInput("ficheiro.txt");
        if (in != null) {
            InputStreamReader tmp = new InputStreamReader(in);
            BufferedReader reader = new BufferedReader(tmp);
            String str;

            int count = 0;
            while ((str = reader.readLine()) != null) {

                if (count != linha) {
                    buf.append(str + "\n");
                }
                count++;
            }
             in.close();
        }
   } catch (java.io.FileNotFoundException e) {
   } catch (Throwable t) {
   }
   //AGORA GUARDOMOS TUDO MENOS O REGISTO EM CAUSA
   try {
        try {
           OutputStreamWriter out = new
OutputStreamWriter(openFileOutput("ficheiro.txt", 0));
           out.write(buf.toString());
           out.close();

        } catch (Throwable t) {
        }
    } catch (Throwable t) {
    }
}
```

Anexo A – 6 – Função Para Contar Registos no Ficheiro de Texto

```
public void contaRegistos() {

    try {

        InputStream in = openFileInput("ficheiro.txt");
        StringBuilder buf = new StringBuilder();
        if (in != null) {
            InputStreamReader tmp = new InputStreamReader(in);
            BufferedReader reader = new BufferedReader(tmp);
            String str;

            int count = 0;


            while ((str = reader.readLine()) != null) {

                count++;
            }
            in.close();
            tvNumRegistos.setText(count - 1 + " Registos por
exportar!");
            numRegistos = count - 1;

        }
    } catch (java.io.FileNotFoundException e) {
    } catch (Throwable t) {
        Toast
                .makeText(this, "Exception: " + t.toString(),
Toast.LENGTH_LONG)

                .show();
    }
}
```

Anexo A – 7 – Função Para Validar Registos Recebidos

```
//FUNÇÃO PARA VALIDAR REGISTOS
public bool validaRegistos(StringBuilder dados)
{
    //PARA CADA REGISTO NO VECTOR VERIFICAR SE O REGISTO CHEGOU
INTACTO
    for (int i = 1; i < registos.Length; i++)
    {

        String registo = registos[i];

        String[] aux = registo.Split('#');

        String dadosRegisto = "";
        String md5S = "";
        Console.WriteLine(aux[0]);

        if (aux[0].Equals("QI"))
        {

            dadosRegisto ="@" +  aux[0] + "#" + aux[1] + "#" + aux[2]
+ "#" + aux[3] + "#" + aux[4] + "#" + aux[5];
            md5S = aux[6];
            if (md5S.Length > 32)
            {
                md5S = md5S.Substring(0, 32);
            }
            Console.WriteLine(dadosRegisto);
            Console.WriteLine(md5S);

        }
        else if (aux[0].Equals("LI") || aux[0].Equals("LG") ||
aux[0].Equals("PI") || aux[0].Equals("PG"))
        {
            dadosRegisto ="@" + aux[0] + "#" + aux[1] + "#" + aux[2] +
"#" + aux[3];
            md5S = aux[4];
            if (md5S.Length > 32)
            {
                md5S = md5S.Substring(0, 32);
            }
            Console.WriteLine(dadosRegisto);
            Console.WriteLine(md5S);

        }

        //FAZER A CONVERSÃO DESTE LADO
        if (dadosRegisto.Contains("@"))
        {
            string hash;
            using (System.Security.Cryptography.MD5 md5 =
System.Security.Cryptography.MD5.Create())
            {
                hash = BitConverter.ToString(

md5.ComputeHash(Encoding.UTF8.GetBytes(dadosRegisto))
                ).Replace("-", String.Empty);
            }
```

```
            Console.WriteLine("REGISTO: " + dadosRegisto);
            Console.WriteLine("MD5S:" + md5S + "/" + md5S.Length);
            Console.WriteLine("hash: " + hash + "/" + hash.Length);


            if (hash.Equals(md5S))
            {
                Console.WriteLine("IGUAL");
            }
            else
            {
                Console.WriteLine("DIFERENTE");
                return false;
            }

        }


    }


        return true;
}
```

Anexo A – 8 – Função Para Apagar Registos do Ficheiro de Texto

```
public void apagRegistos() {

    StringBuilder buf = new StringBuilder();

    try {
        InputStream in = openFileInput("ficheiro.txt");
        if (in != null) {
            InputStreamReader tmp = new InputStreamReader(in);
            BufferedReader reader = new BufferedReader(tmp);
            String str;

            str = reader.readLine();
            buf.append(str + "\n");


            in.close();
        }
    } catch (java.io.FileNotFoundException e) {
    } catch (Throwable t) {
    }

    try {
        try {
            OutputStreamWriter out = new
OutputStreamWriter(openFileOutput("ficheiro.txt", 0));
            out.write(buf.toString());
            out.close();

        } catch (Throwable t) {
        }
    } catch (Throwable t) {
    }

}
```

Anexo A – 9 – Função Para Inserir Registos Nas Tabelas

```
public void insereRegistosGrid(StringBuilder registos)
{
    Console.WriteLine(registos);

    //SEPARAR OS REGISTOS POR UM VECTOR
    String[] dados = registos.ToString().Split('@');

    int n = 0;

    try
    {
        //PARA CADA REGISTO NO VECTOR, TRATA A SUA INFORMAÇÃO E
ADICIONAR A GRID
        for (int i = 1; i < dados.Length; i++)
        {
            String[] aux = dados[i].Split('#');

            //SE FOR UM REGISTO DE QUARENTENA
            if (aux[0].Equals("QI"))
            {
                dgvRegistosQuarentena.Invoke((MethodInvoker)(() => n =
(int)dgvRegistosQuarentena.Rows.Add()));
                dgvRegistosQuarentena.Invoke((MethodInvoker)(() =>
dgvRegistosQuarentena.Rows[n].Cells[0].Value = aux[2]));
                dgvRegistosQuarentena.Invoke((MethodInvoker)(() =>
dgvRegistosQuarentena.Rows[n].Cells[1].Value = aux[3]));
                dgvRegistosQuarentena.Invoke((MethodInvoker)(() =>
dgvRegistosQuarentena.Rows[n].Cells[2].Value = aux[4]));
                dgvRegistosQuarentena.Invoke((MethodInvoker)(() =>
dgvRegistosQuarentena.Rows[n].Cells[3].Value = aux[5]));
            }
                //SE FOR UM REGISTO DE PESO OU PRODUÇÃO
            else
            {
                if (aux[0].Equals("PI"))
                {
                    aux[0] = "Peso Individual";
                    aux[3] = aux[3] + " Kg";
                }
                else if (aux[0].Equals("PG"))
                {
                    aux[0] = "Peso Grupo";
                    aux[3] = aux[3] + " Kg";
                }
                else if (aux[0].Equals("LI"))
                {
                    aux[0] = "Produção Individual";
                    aux[3] = aux[3] + " L";
                }
                else
                {
                    aux[0] = "Produção Grupo";
                    aux[3] = aux[3] + " L";
                }
                dgvRegistos.Invoke((MethodInvoker)(() => n =
(int)dgvRegistos.Rows.Add()));
```

```
                dgvRegistos.Invoke((MethodInvoker)(() =>
dgvRegistos.Rows[n].Cells[0].Value = aux[2]));
                dgvRegistos.Invoke((MethodInvoker)(() =>
dgvRegistos.Rows[n].Cells[1].Value = aux[3]));
                dgvRegistos.Invoke((MethodInvoker)(() =>
dgvRegistos.Rows[n].Cells[2].Value = aux[1]));
                dgvRegistos.Invoke((MethodInvoker)(() =>
dgvRegistos.Rows[n].Cells[3].Value = aux[0]));
            }

        }
    }
    catch (Exception e)
    {
        Console.WriteLine("ERRO" + e);
    }
}
```

Anexo A – 10 – Função Para Inserir Registos na Base de Dados

```
public void insereBD()
{
    int registosInseridos = 0;
    String IDANIMAL = "";
    String DATAINICIO;
    String DATAFIM;

    String QUANTIDADE;
    String GRUPO;
    String TIPO;

    try
    {
        conn = new MySqlConnection(cs);
        conn.Open();


        //PARA OS DADOS DA QUARENTENA
        while (dgvRegistosQuarentena.Rows.Count > 1)
        {
            IDANIMAL = "";
            MySqlCommand cmd = new MySqlCommand();
            cmd.Connection = conn;

            IDANIMAL =
dgvRegistosQuarentena.Rows[0].Cells[0].Value.ToString();
            DATAINICIO =
dgvRegistosQuarentena.Rows[0].Cells[1].Value.ToString();
            DATAFIM =
dgvRegistosQuarentena.Rows[0].Cells[2].Value.ToString();

            cmd.CommandText = "INSERT INTO quarentena(IDAnimal,
DataInicio, DataFim) VALUES(@IDAnimal, STR_TO_DATE(@DataInicio,'%d-%m-
%Y'), STR_TO_DATE(@DataFim,'%d-%m-%Y'))";
            cmd.Prepare();
            cmd.Parameters.AddWithValue("@IDAnimal", IDANIMAL);
            cmd.Parameters.AddWithValue("@DataInicio", DATAINICIO);
            cmd.Parameters.AddWithValue("@DataFim", DATAFIM);
            cmd.ExecuteNonQuery();
            dgvRegistosQuarentena.Rows.RemoveAt(0);
            registosInseridos++;
        }


        //PARA OS DADOS DOS REGISTOS
        while (dgvRegistos.Rows.Count > 1)
        {
            IDANIMAL = "";
            MySqlCommand cmd = new MySqlCommand();
            cmd.Connection = conn;

            //PARA PESO INDIVIDUAL
            if
(dgvRegistos.Rows[0].Cells[3].Value.ToString().Equals("Peso
Individual")){
                TIPO = "PI";
```

```
                IDANIMAL =
dgvRegistos.Rows[0].Cells[0].Value.ToString();
                QUANTIDADE =
dgvRegistos.Rows[0].Cells[1].Value.ToString().Replace(",", ".");
                DATAINICIO =
dgvRegistos.Rows[0].Cells[2].Value.ToString();

                cmd.CommandText = "INSERT INTO peso(IDAnimal, Peso,
Data, Tipo) VALUES(@IDAnimal, @Peso,  STR_TO_DATE(@Data,'%d-%m-%Y'),
@Tipo)";
                cmd.Prepare();
                cmd.Parameters.AddWithValue("@IDAnimal", IDANIMAL);
                cmd.Parameters.AddWithValue("@Peso", QUANTIDADE);
                cmd.Parameters.AddWithValue("@Data", DATAINICIO);
                cmd.Parameters.AddWithValue("@Tipo", TIPO);

                cmd.ExecuteNonQuery();
                dgvRegistos.Rows.RemoveAt(0);
                registosInseridos++;


            }else

            //PARA PRODUÇÃO INDIVIDUAL
            if
(dgvRegistos.Rows[0].Cells[3].Value.ToString().Equals("Produção
Individual"))
            {
                TIPO = "LI";
                IDANIMAL =
dgvRegistos.Rows[0].Cells[0].Value.ToString();
                QUANTIDADE =
dgvRegistos.Rows[0].Cells[1].Value.ToString().Replace(",", ".");
                DATAINICIO =
dgvRegistos.Rows[0].Cells[2].Value.ToString();

                cmd.CommandText = "INSERT INTO producao(IDAnimal,
Producao, Data, Tipo) VALUES(@IDAnimal, @Producao,
STR_TO_DATE(@Data,'%d-%m-%Y'), @Tipo)";
                cmd.Prepare();
                cmd.Parameters.AddWithValue("@IDAnimal", IDANIMAL);
                cmd.Parameters.AddWithValue("@Producao", QUANTIDADE);
                cmd.Parameters.AddWithValue("@Data", DATAINICIO);
                cmd.Parameters.AddWithValue("@Tipo", TIPO);

                cmd.ExecuteNonQuery();
                dgvRegistos.Rows.RemoveAt(0);
                registosInseridos++;

            }else

            //PARA PESO GRUPO
            if
(dgvRegistos.Rows[0].Cells[3].Value.ToString().Equals("Peso Grupo"))
            {


                TIPO = "PG";
                GRUPO = dgvRegistos.Rows[0].Cells[0].Value.ToString();
                QUANTIDADE =
dgvRegistos.Rows[0].Cells[1].Value.ToString().Replace(",", ".");
```

```
                DATAINICIO =
dgvRegistos.Rows[0].Cells[2].Value.ToString();


                if (gruposAnimais().Contains(GRUPO))
                {

                    cmd.CommandText = "INSERT INTO peso(IDAnimal,
Peso, Data, Tipo) VALUES(@IDAnimal, @Peso,  STR_TO_DATE(@Data,'%d-%m-
%Y'), @Tipo)";
                    cmd.Prepare();
                    cmd.Parameters.AddWithValue("@IDAnimal", GRUPO);
                    cmd.Parameters.AddWithValue("@Peso", QUANTIDADE);
                    cmd.Parameters.AddWithValue("@Data", DATAINICIO);
                    cmd.Parameters.AddWithValue("@Tipo", TIPO);

                    cmd.ExecuteNonQuery();
                    dgvRegistos.Rows.RemoveAt(0);
                    registosInseridos++;

                }
                else
                {
                    MessageBox.Show("O grupo de animais " + GRUPO + "
não existe.", "Erro", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }

            }else

            //PARA PRODUÇÃO GRUPO
            if
(dgvRegistos.Rows[0].Cells[3].Value.ToString().Equals("Produção
Grupo"))
            {
                TIPO = "LG";
                GRUPO = dgvRegistos.Rows[0].Cells[0].Value.ToString();
                QUANTIDADE =
dgvRegistos.Rows[0].Cells[1].Value.ToString().Replace(",", ".");
                DATAINICIO =
dgvRegistos.Rows[0].Cells[2].Value.ToString();


                if (gruposAnimais().Contains(GRUPO))
                {
                cmd.CommandText = "INSERT INTO producao(IDAnimal,
Producao, Data, Tipo) VALUES(@IDAnimal, @Producao,
STR_TO_DATE(@Data,'%d-%m-%Y'), @Tipo)";
                cmd.Prepare();
                cmd.Parameters.AddWithValue("@IDAnimal", GRUPO);
                cmd.Parameters.AddWithValue("@Producao", QUANTIDADE);
                cmd.Parameters.AddWithValue("@Data", DATAINICIO);
                cmd.Parameters.AddWithValue("@Tipo", TIPO);

                cmd.ExecuteNonQuery();
                dgvRegistos.Rows.RemoveAt(0);
                registosInseridos++;
                }
                else
                {
```

```
                    MessageBox.Show("O grupo de animais " + GRUPO + "
não existe.", "Erro", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }
            }
        }

    }
    catch (MySqlException ex)
    {
        Console.WriteLine("Error: {0}", ex.ToString());
        if (ex.ToString().Contains("foreign key constraint fails"))
        {
            MessageBox.Show("O animal " + IDANIMAL + " não está
registado.", "Erro", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else
        {
            MessageBox.Show("Erro ao inserir registos.", "Erro",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

    }
    finally
    {
        if (conn != null)
        {
            conn.Close();
            MessageBox.Show("Foram adicionados " + registosInseridos +
" com sucesso!", "Sucesso", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        if (dgvRegistos.Rows.Count == 1 &&
dgvRegistosQuarentena.Rows.Count == 1)
        {
            btnValidarRegistos.Visible = false;
        }
    }
}
```

Anexo B – Extratos de Código

Anexo B – 1 – Código que Verifica se uma Exploração se Encontra Registada

```
try {

    InputStream in = openFileInput("ficheiro.txt");

    if (in != null) {

        InputStreamReader tmp=new InputStreamReader(in);
        BufferedReader reader=new BufferedReader(tmp);
        String str;
        StringBuilder buf=new StringBuilder();

        while ((str = reader.readLine()) != null) {
            buf.append(str+"\n");

controlString =
buf.toString().split(Pattern.quote("@"));
        }

        in.close();
    }

    //SE A EXPLORACAO JA EXISTIR PASSA PARA A PAGINA DEFAULT
    if (controlString[0].length() == 9){
Intent myIntent = new Intent(NovaExploracao.this,Default.class);

        startActivity(myIntent);
    }
}

catch (java.io.FileNotFoundException e) {

}
catch (Throwable t) {

}
```

FAnexo B – 2 – Iniciar Ligação Via USB (Desktop)

```
Process cmd = new Process();
cmd.StartInfo.FileName = "cmd.exe";
cmd.StartInfo.RedirectStandardInput = true;
cmd.StartInfo.RedirectStandardOutput = true;
cmd.StartInfo.CreateNoWindow = true;
cmd.StartInfo.UseShellExecute = false;
cmd.StartInfo.WorkingDirectory = @"C:\Users\Tiago
Fernandes\AppData\Local\Android\sdk\platform-tools";
cmd.Start();

cmd.StandardInput.WriteLine("adb forward tcp:6000 tcp:6000");
cmd.StandardInput.Flush();
cmd.StandardInput.Close();
cmd.WaitForExit();

//-------------------------------------------------------
//DEPOIS VERIFICAMOS SE FICOU BEM FEITO
cmd.StartInfo.FileName = "cmd.exe";
cmd.StartInfo.WorkingDirectory = @"C:\Users\Tiago
Fernandes\AppData\Local\Android\sdk\platform-tools";
cmd.Start();

cmd.StandardInput.WriteLine("adb forward --list");
cmd.StandardInput.Flush();
cmd.StandardInput.Close();
cmd.WaitForExit();
String aux = cmd.StandardOutput.ReadToEnd();

if (aux.Contains("tcp:6000 tcp:6000"))
{
    //CASO SUCESSO INICIAMOS A NOSSA COMUNICAÇÃO
    iniciarComunicacaoUSB();
}
else
{
    //CASO ERRO
    lblStatus.Text = "Status: Não Conectado";
    MessageBox.Show("Dispositivo não conectado.", "Erro",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```