UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia Elétrica e de Computação

Implementação de um servidor utilizando Linux embarcado com acesso e gerenciamento através de *smartphone*

André Ricardo Gouveia Barros

São Carlos - SP

Implementação de um servidor utilizando Linux embarcado com acesso e gerenciamento através de *smartphone*

André Ricardo Gouveia Barros

Orientador: *Evandro Luís Linhari Rodrigues*

Monografia final de conclusão do curso Engenharia de Computação apresentada ao Departamento de Engenharia Elétrica e de Computação – EESC-USP.

USP - São Carlos
Outubro de 2013

Barros, André Ricardo Gouveia
B268i Implementação de um servidor utilizando Linux embarcado com acesso e gerenciamento através de smartphone / André Ricardo Gouveia Barros; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2013.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2013.

1. Raspberry Pi. 2. Android. 3. Cliente/Servidor. 4. GPIO. I. Título.

FOLHA DE APROVAÇÃO

Nome: André Ricardo Gouveia Barros

Título: "Implementação de um servidor utilizando Linux embarcado com acesso e gerenciamento através de smartphone"

Trabalho de Conclusão de Curso defendido em 11 / 11 / 2013.

Comissão Julgadora:	**Resultado:**
Prof. Associado Evandro Luís Linhari Rodrigues Orientador - SEL/EESC/USP	APROVADO
Prof. Dr. Marcelo Andrade da Costa Vieira SEL/EESC/USP	APROVADO
Prof. Dr. Valdir Grassi Júnior SEL/EESC/USP	Aprovado

Coordenador pela EESC/USP do Curso de Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

”Pepe! Já tirei a vela!”

Ruben Aguirre.

Chapolin: A Troca de Cérebros.

Dedicatória

À minha mãe Rosangela e meu pai Aclézio, pela compreensão, apoio e contribuição para minha formação pessoal e acadêmica.

Agradecimentos

Agradeço a minha família pelo apoio e disposição em me ajudar a conquistar meus sonhos e explorar meus potenciais, ao corpo docente da USP pelo conhecimento compartilhado e aos meus amigos pelo auxílio e incentivo neste trabalho, além da companhia e alegrias durante a graduação.

Resumo

Com o surgimento de sistemas embarcados com propósitos educacionais a criação de projetos embarcados tornou-se economicamente viável, aumentando o número de usuários que se aventuram nessa área. Neste trabalho, é projetado e desenvolvido um servidor implementado na Raspberry Pi que fornece funções relacionadas a sua câmera e a seus pinos de GPIO. O acesso a esse servidor é feito remotamente por clientes desenvolvidos em Java e executados em *smartphone* Android. Com esta combinação é possível a utilização em diversas áreas, como por exemplo um sistema de segurança e vigilância, com acesso a camêra e possibilidade de acionamento de alarmes, sensores e lâmpadas e tantas outras aplicações que se possa imaginar no campo de automação via Web. O desenvolvimento deste projeto possibilitou uma maneira interessante de introdução nesse universo de conhecimento de forma prática e funcional, agregando valor técnico, teórico e científico.

Palavras-chaves: Sistemas Embarcados, Linux Embarcado, Android, Raspberry Pi, RaspiVid, RaspiStill.

Abstract

With the growing of embedded systems for educational purposes, creating embedded project became economically viable, increasing the number of users who venture into this area. In this work is the design and developed a server implemented in a Raspberry Pi that provides functions related to its camera and its GPIO pins. Access to this server is done remotely by clients on smartphone Android. With this combination is possible to use in various areas, such as a security system, having access to the camera and the possibility of triggering alarms, sensors and lamps, and many other applications imaginable in automation via web. The development of this project allowed an interesting way of introducing this universe of knowledge in a practical and functional, adding value technical, theoretical and scientific.

Keywords: Embedded Systems, Embedded Linux, Android, Raspberry Pi, RaspiVid, RaspiStill.

Sumário

Lista de Figuras

1 Introdução

1.1 Contextualização e Motivação

O telefone móvel, ou celular, foi lançado em 1973, em Nova Iorque. Na época, o aparelho era gigantesco, tanto que o primeiro a ser viável comercialmente pesava em torno de 800 gramas e foi lançado uma década após sua criação. No seu início não existia o sinal digital, somente o analógico e seu uso era restrito para a parcela da população com maior poder aquisitivo. Outro aspecto que se alterou completamente foi a questão da percepção do usuário em relação ao seu uso uma vez que no seu início as pessoas não viam tanta utilidade a não ser poder falar em diferentes localidades com o mesmo telefone. Atualmente, o celular é um dos equipamentos tecnológicos mais comuns no país. É extremamente difícil encontrar alguém com mais de 12 anos que não possua ou já tenha utilizado um aparelho. [17].

O *smartphone* é basicamente um celular com funcionalidades avançadas que podem ser estendidas através de programas e aplicações, tendo sua utilidade em diversas áreas, como ferramenta de trabalho, lazer e comunicação. O uso de *smartphones* não para de aumentar, o Brasil é o quarto país do mundo em número de *smartphones* no mundo são 70 milhões. [19]

Outra parte importante a considerar é a crescente adoção e uso da internet em dispositivos móveis. É esperado que nos próximos anos, os usuários de Internet troquem os computadores pessoais pelos seus *smartphones*. Segundo o IDC (*International Data Corporation*), espera-se que em 2015 o número de usuários conectados à internet via *smartphones* seja consideravelmente maior dos que o fazem através do computador.

Outra área abrangida por este trabalho é o de sistemas embarcados, mais especificamente aqueles capazes de rodar Linux, no caso, a Raspberry Pi.

Sistemas embarcados consistem em sistemas microprocessados no qual o poder de processamento é voltado à realização de uma atividade específica. Se comparado com um computador normal, o sistema embarcado é construído para realizar sua atividade mais rapidamente, ocupando menor espaço, consumindo menos energia e com custo reduzido. Existem dispositivos

com sistemas embarcados com diversos propósitos, alguns mais dedicados e outros mais abrangentes.

Sistemas embarcados com propósito dedicado tem seu uso mais restrito. Como por exemplo: microondas, geladeiras, freios ABS, impressoras, entre outros. Esses sistemas tem seus componentes computacionais dimensionados de acordo com suas tarefas, podendo possuir um processador de menor frequência, mas que consegue responder em tempo esperado para desempenho da aplicação [22].

Sistemas embarcados com propósito geral possuem geralmente processadores e memórias mais potentes e de maior desempenho se comparados aos dedicados. Seu uso pode variar de acordo com sua finalidade [22]. Um exemplo comum desses sistemas é a Raspberry Pi, um computador do tamanho de um cartão de crédito criado para estimular o ensino de programação e tecnologia. Foi anunciado em 2011 e lançado em 29 de fevereiro de 2012, idealizado pelo inglês Pete Lomas para ser o computador mais barato do mercado, com o preço de 25 dólares (modelo A) ou 35 dólares (modelo B).

1.2 Objetivos

O objetivo deste trabalho é unir as duas áreas citadas, *smartphone* e sistema embarcado, mais especificamente, Android e Raspberry Pi, criando uma plataforma que ofereça acesso a câmera e aos pinos GPIO da Raspberry Pi, controlados por um celular Android. Essa combinação possibilita o uso em diversas áreas, desde segurança residencial com acesso a camêra e acionamento de alarmes, sensores e lâmpadas, até ao controle de uma estufa, com acompanhamento do crescimento das plantas através de imagens capturadas de tempo em tempo.

O trabalho pretende focar no controle de algumas funções presentes na Raspberry Pi através de qualquer *smartphone* que utilize Android versão 2.2 ou superior, englobando 98 por cento dos usuários de Android.

O celular Android conectado à internet com o aplicativo desenvolvido neste projeto atuará como cliente, que permitirá o acesso, monitoramento e controle de funções fornecidas pela Raspberry Pi.

A Raspberry Pi conectada a internet, atuará como servidor, fornecendo acesso a sua câmera e a seus pinos de propósito geral (*General-purpose input/output* - GPIO).

Com essa combinação espera-se obter o acesso remoto a Raspberry Pi, possibilitando:

- Movimentação da câmera com uso de servos controlados pelo acelerômetro do Android.
- Captura de imagem em tempo real.
- Agendamento para captura de imagem.
- Visualização de vídeo em tempo real.
- Agendamento para gravação de vídeo.
- Acesso, *download* e visualização dos arquivos de imagem e vídeo gerados.
- Acionamento dos pinos GPIO.
- Agendamento do funcionamento dos pinos GPIO.
- Uso do acelerômetro do *smartphone* para controle do posicionamento da câmera.

1.3 Organização do Trabalho

Nos próximos capítulos são apresentados a fundamentação teórica pesquisada e utilizada durante o desenvolvimento do projeto, a implementação do servidor, do cliente e da comunicação. Por fim, é apresentado os resultados obtidos, dificuldades encontradas e possíveis trabalhos futuros.

2 *Materiais e Métodos*

Neste capítulo são apresentados os materiais, métodos e as terminologias básicas da área em que o projeto se insere e o levantamento bibliográfico necessário para realização deste trabalho. Em particular, a descrição dos principais trabalhos de pesquisa relacionados com este, bem como dos trabalhos que serviram de base para a solução proposta por este projeto.

2.1 Raspberry

A Raspberry Pi é um computador de pequeno porte, do tamanho de um cartão de crédito, com dimensões de 85.60mm x 56mm x 21mm que pesa 45g [8].

Raspberry Pi foi anunciado em 2011, idealizado pelo inglês Pete Lomas para ser o computador mais barato do mercado, com o preço de 25 dólares (Modelo A) ou 35 dólares (Modelo B). O aparelho foi lançado em 29 de fevereiro de 2012 com finalidades educativas. O computador de código aberto foi criado para estimular o ensino de programação e tecnologia [12].

É baseado em um *system on a chip Broadcom BCM2835* [7], que inclui um processador *ARM1176JZF-S* de 700 MHz com operações em ponto flutuante, 256 ou 512 MB de memória RAM e uma GPU Videocore 4, capaz de reproduzir vídeos em qualidade de BluRay, utilizando H264 a taxas de 40MBits/s, com acesso a OpenGL ES2.0 e bibliotecas OpenVG.

Existem dois modelos: Modelo A que conta com 256 MB RAM, uma porta USB e nenhuma porta Ethernet e o Modelo B, figura 2.1, com 512 MB RAM, 2 portas USB e uma porta Ethernet [12].

Os requisitos de energia do dispositivo são bem comuns, ela é alimentada por 5V via porta micro USB. Para o Modelo B é necessário uma fonte que forneça até 700mA, enquanto para o Modelo A, são apenas 300mA. Muitos carregadores de celular atendem a essa exigência [10].

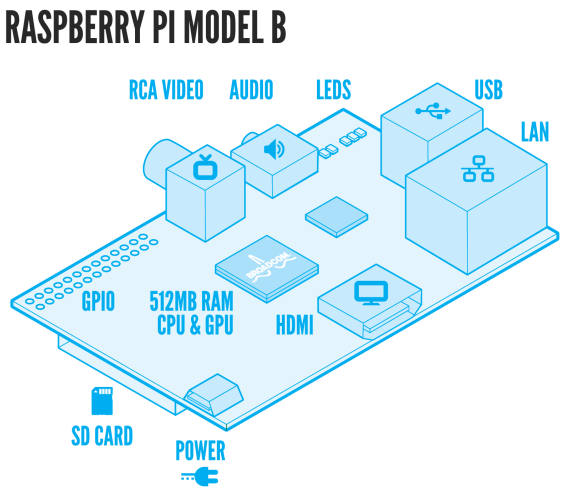


Figura 2.1: Raspberry Pi modelo B [8]

2.1.1 Sistema Operacional

Existem diversas possibilidades de sistema operacional para a Raspberry Pi, no site oficial [9] são disponibilizadas algumas versões. A seguir são descritos brevemente os principais sistemas operacionais fornecidos.

Raspbian Wheezy

Raspbian é um sistema operacional livre baseado em *Debian* otimizado para o hardware da Raspberry Pi, que conta com mais de 35 mil pacotes.

A construção inicial destes pacotes otimizados do *Raspbian* foi concluída em junho de 2012. No entanto, o *Raspbian* encontra-se em desenvolvimento ativo, com ênfase em melhorar a estabilidade e o desempenho de tantos pacotes quanto possível.

Raspbian é a distribuição recomendada para a maioria dos usuários da Raspberry Pi. A exceção é para os usuários que são dependentes de software que ainda não está presente ou funcional em raspbian.

Soft Float Debian Wheezy

O *Soft Float Debian Wheezy* é idêntico ao *Raspbian wheezy*, porém utiliza *soft-float*. É recomendado para se utilizar com softwares como Oracle JVM, que ainda não tem suporte ao *hard-float* utilizado pelo *Raspbian*.

Arch Linux ARM

O *Arch Linux ARM* é baseado no *Arch* Linux. Foca na simplicidade e controle total para o usuário final, dando-lhe total controle e responsabilidade sobre o sistema. Essa versão pode ser complicada para iniciantes.

Fornece suporte a *soft-float* para ARMv5 e suporte *hard-float* para ARMv6 e ARMv7 [2].

RISC OS

É um sistema operacional projetado em Cambrigde pela Acorn. Lançado em 1987, suas origens levam ao time original de desenvolvedores do ARM.

De 1988 a 1998 o RISC OS era vastamente utilizado em todo computador baseado em ARM produzido pela Acorn. Após o desmembramento da Acorn, em 1998, o desenvolvimento do sistema operacional foi bifurcada e continuou separadamente por várias empresas. Em 2011 foi anunciado uma versão de desenvolvimento para a Raspberry Pi [21].

2.1.2 Raspberry Pi Camera

A câmera da Raspberry Pi é um módulo adicional. Ela é ligada a um dos dois *sockets* superiores presentes na Raspberry Pi. Esse *socket* trata-se de uma interface CSI (*Camera Serial Interface*), que é desenhada especialmente para câmeras, sendo capaz de suportar altas taxas de transferência de dados [14].

A placa da câmera em si é pequena e pesa menos de 4 gramas, se tornando excelente para aplicações móveis ou em aplicações que o tamanho e peso são importantes.

O sensor da câmera tem resolução nativa de 5 megapixel com lentes fixas. Em termos de imagem estática, a câmera é capaz de gerar fotos de até 2592 x 1944 pixel, além de suportar vídeos em 1080p a 30fps, 720p a 60fps e 480p a 90fps.

A câmera foi lançado para venda em 14 de Maio de 2013 com produção inicial de dez mil unidades que teve seu estoque esgotado alguns dias depois do início das vendas.

A imagem 2.2 mostra uma foto do módulo da câmera acoplado a Raspberry Pi.

Tabela 2.1: Dados técnicos da câmera Raspberry Pi

Tipo de sensor	OmniVision OV5647 Color CMOS QSXGA (5 megapixel)
Tamanho do sensor	3.67 x 2.74 (mm)
Quantidade de pixel	2592 x 1944
Tamanho do pixel	1.4 x 1.4 (um)
Anglo de visão	54 x 41 (graus)
Campo de visão	2.0 x 1.33 m há 2 m
Lente equivalente	35 mm
Foco fixo	1 m
Video	1080p a 30 fps com codec H264
Tamanho da placa	25 x 24 (mm)

Figura 2.2: Raspberry Pi modelo B e módulo da câmera [8]

2.1.3 General-purpose input/output - GPIO

Um *General Purpose Input/Output*, ou GPIO, é um pino genérico que pode ter seu comportamento (de entrada ou de saída) controlado/programado por software. Fornece saídas lógicas 0 e 1, podendo ser utilizado para o acionamento de diversos atuadores.

A Raspberry Pi possui 26 pinos, alinhados em duas colunas de 13, que fornecem: 8 GPIO, I2C, SPI, UART, +3.3V, +5.0V e GND. Todos os pinos de UART, SPI e I2C podem ser reconfigurados como GPIO, sendo possível alcançar 17 pinos GPIO [24].

O nível de tensão do GPIO é de 3.3V e não de 5V. Não existe qualquer tipo de proteção de sobrecarga, a intenção é que pessoas interessadas no uso intensivo destes pinos utilizem uma placa externa ao invés da ligação direta na Raspberry Pi, porém nada impede o uso direto, desde que seja tomada as devidas precauções. A figura 2.3 exibe o posicionamento dos pinos da Raspberry Pi.

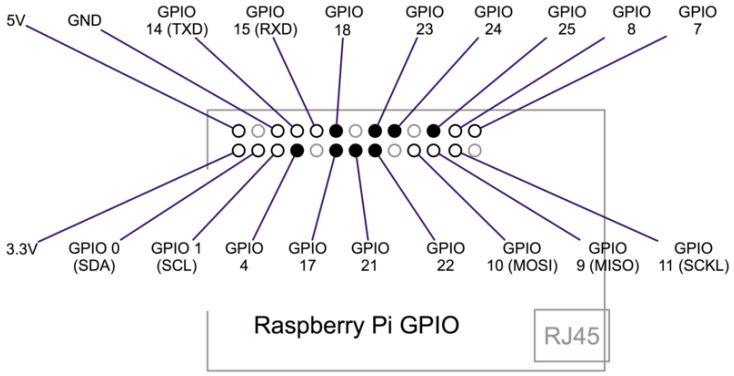


Figura 2.3: Pinos GPIO da Raspberry Pi. No modelo B a nomenclatura muda, o pino 21 passa a ser 27 [24].

2.2 Android

O Android é um sistema operacional baseado em Linux projetado para dispositivos móveis *touchscreen*, como *smartphones* e *tablet*.

Foi inicialmente desenvolvido pela *Android Inc*, com apoio financeiro do Google, que posteriormente comprou a empresa. Foi lançado em 2007 junto com a fundação *Open Handset Alliance*, um consórcio de hardware, software, telecomunicações e empresas dedicadas ao avanço dos dispositivos móveis [16]. O primeiro telefone com Android foi vendido em outubro de 2008.

O Android é *open source* e o Google libera o código sob a licença Apache, permitindo que o software seja livremente modificado e distribuído por fabricantes de aparelhos, operadoras sem fio e desenvolvedores entusiastas [15].

Além disso, o Android tem uma grande comunidade de desenvolvedores que criam aplicativos que estendem a funcionalidade de dispositivos, os quais são escritos principalmente em uma versão personalizada da linguagem de programação Java.

Em outubro de 2012, havia cerca de 700 mil aplicativos disponíveis para Android [27]. O número estimado de aplicativos instalados a partir do *Google Play*, principal loja de aplicativos do Android, é de 25 bilhões, como mostra a figura 2.4.

Esses fatores têm contribuído para tornar o Android plataforma de *smartphone* mais usado do mundo.

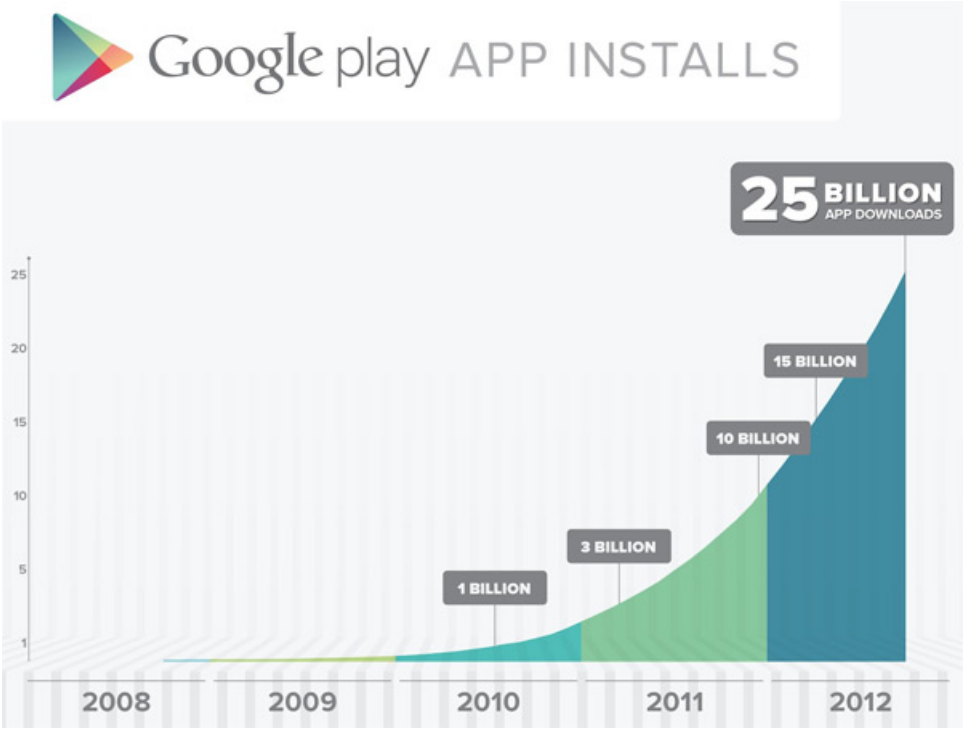


Figura 2.4: Número aplicativos Android instalados entre 2009 e 2012 [18].

2.2.1 Eclipse e Plugin ADT

O Eclipse é um ambiente de desenvolvimento multi-linguagem Integrado (IDE), escrito principalmente em Java. Pode ser usado para desenvolver aplicações em Java e, por meio de vários *plugins*, outras linguagens de programação, incluindo Ada, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy, Scheme e Erlang.

O *Android Development Tools* (ADT) é um *plugin* projetado para o Eclipse IDE para fornecer um ambiente poderoso e integrado para a criação de aplicativos Android [1].

O ADT amplia os recursos do Eclipse para que se possa criar novos projetos, interfaces do usuário, adicionar pacotes com base no quadro API Android, depurar seus aplicativos usando as ferramentas do SDK do Android e até mesmo exportar o aplicativo a fim de distribuir a sua aplicação .

2.2.2 Android Studio

O Android Studio é um novo ambiente de desenvolvimento Android com base no *IntelliJ IDEA*. Semelhante ao Eclipse com o *plugin* ADT, Android Studio fornece ferramentas de desenvolvimento integradas Android para desenvolvimento e depuração [13].

Foi anunciado em 16 de Maio de 2013 na conferência Google I/O. Em junho de 2013, ele foi disponível em forma de beta para os usuários.

2.3 Video Streaming

Streaming é utilizado para distribuir conteúdo multimídia através da Internet. Em *streaming*, as informações multimídia não são, usualmente, arquivadas pelo usuário que está recebendo o *streaming* (a não ser o armazenamento temporário no *cache* do sistema ou que o usuário ativamente faça a gravação dos dados) a mídia é reproduzida a medida que chega ao usuário, desde que a sua largura de banda seja suficiente para reproduzir os conteúdos em tempo real.

Assim, para visualização da câmera em tempo real se faz necessário a utilização do serviço de *streaming*. Existem *softwares* que provem tal serviço. Abaixo são listados alguns, bem como uma breve descrição de cada um.

2.3.1 Motion

O motion é um programa que controla o sinal de vídeo a partir de uma ou mais câmeras. É capaz de detectar se uma parte significativa da imagem mudou, ou seja, ele pode detectar o movimento [4]. O programa é escrito em C e é feito para o sistema operacional Linux, usando a interface video4linux.

2.3.2 MJPG-streamer

MJPG-streamer é uma aplicação baseada em linha de comando para transmitir arquivos JPG através de uma rede IP da webcam para um navegador.

É possível fazer uso do sistema de compressão que certas câmeras dispõem, a fim de reduzir o custo de processamento no servidor. Isso o torna uma solução leve para dispositivos embarcados e servidores comuns, que não devem usar a maior parte de seu processamento para comprimir quadros [23].

2.3.3 FFmpeg

FFmpeg é um dos principais *framework* multimídia. Ele é capaz de codificar, decodificar , mux, demux, realizar *streaming*, filtrar e rodar praticamente qualquer formato.

O FFmpeg é um projeto que tenta oferecer boas soluções técnicas para os desenvolvedores de aplicações e usuários finais. Para isso ele combina as melhores opções de software disponíveis gratuitamente [3].

2.3.4 RTSP

O *Real Time Streaming Protocol* é um protocolo de controle de rede projetado para uso em sistemas de entretenimento e de comunicação para controlar servidores de *streaming* de mídia.

A transmissão de *streaming* de dados em si não é uma tarefa do protocolo RTSP. A maioria dos servidores RTSP usam o *Real-time Transport Protocol* (RTP) em conjunto com o *Real-time Control Protocol* (RTCP) para entrega em fluxo de mídia.

O RTSP foi desenvolvido pelo *Multiparty Multimedia Session Control Working Grouo* da *Internet Engineering Task Force* (IETF) e publicado como RFC 2326 em 1998 [6].

2.4 Modulação por largura de pulso - PWM

Uma possibilidade para o controle da câmera é utilização de servos montados como mostra a figura 2.5. O posicionamento dos servos é feito por PWM, como ângulo definido pela largura do pulso.

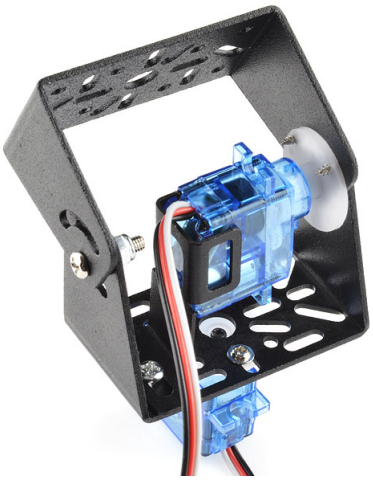


Figura 2.5: Sistema para controle do posicionamento da câmera [25].

A modulação por largura de pulso (PWM, *Pulse Width Modulation*) é uma técnica largamente utilizada para o controle de dispositivos e sinais, desde iluminação e acionamento de motores até áudio. Consiste em uma onda que alterna seu estado em nível lógico alto e um nível lógico baixo e varia seu *duty cicle*. Com isso é possível o controle de motores e servos.

Apesar de dispor de diferentes comunicações (GPIO, I2C, SPI, UART, +3.3V, +5.0V e GND), a Raspberry Pi não conta com PWM. A seguir são listadas algumas maneiras pesquisadas e utilizadas para se conseguir gerar PWM para a Raspberry Pi.

2.4.1 ServoBlaster

ServoBlaster é um software para a Raspberry Pi, que fornece uma interface para geração de PWM através dos pinos GPIO. É possível se comunicar com o *driver* informando qual a largura de pulso desejada. O *driver* cria um arquivo de dispositivo */dev/servoblaster*, no qual você pode enviar comandos.

O *driver* funciona através da criação de uma lista encadeada de blocos de controle de DMA, com o último ligado ao primeiro, por isso, uma vez iniciado o controlador de DMA roda continuamente e o *driver* não precisa se envolver, a não ser quando uma largura de pulso precisa ser mudado. Com isso, a influência do uso do processador na geração do PWM é reduzida.

Existem duas implementações de *ServoBlaster*, uma sendo baseado em um módulo do *kernel* e outra baseado no *user space daemon*. O módulo baseado no *kernel* é o original, porém difícil de se alcançar, pois é preciso de um *kernel* que combine com o versão utilizada pelo desenvolvedor. A implementação no *user space daemon* é muito mais conveniente para se utilizar e conta com os mesmos recursos [20].

O desempenho do *ServoBlaster* para controle dos servos é satisfatório, porém, o uso de PWM desta forma interfere com algumas outras saídas da Raspberry Pi, como por exemplo, a saída de áudio de 3.5mm e com a própria câmera.

2.4.2 Pi-blaster

Pi-blaster é um software que foi feito a partir do *ServoBlaster*, tornando possível a criação de oito sinais PWM que alcançam de zero a cem por cento da largura do pulso, diferente do *ServoBlaster* que alcança apenas doze por cento quando todas as saídas são utilizadas [26].

O desempenho é o mesmo alcançado com o *ServoBlaster*, inclusive no que se diz respeito as interferências nos outros pinos.

3 *Desenvolvimento do Trabalho*

Para o desenvolvimento deste trabalho foi adotada uma arquitetura semelhante ao paradigma cliente/servidor. Com a Raspberry Pi atuando como servidor e o *smartphone* Android atuando como cliente. A conexão entre cliente e servidor é feita em qualquer rede através de *sockets* TCP/IP.

O servidor aceita conexão do cliente durante toda sua execução. O cliente por sua vez deve efetuar a conexão utilizando o aplicativo desenvolvido neste trabalho, no qual informa o endereço IP e a porta do servidor.

Quando estabelecido a conexão entre ambas as partes, inicia-se a troca de mensagens, sendo o cliente o responsável por fazer requisições ao servidor e, com isso, obter acesso as funções disponíveis.

As funções que o servidor oferece são:

- Captura de imagem em tempo real com controle de parâmetros da imagem.
- Agendamento para captura de imagem, com controle de parâmetros e do tempo entre a captura das imagens.
- Acionamento e desligamento do *streaming* de vídeo.
- Agendamento para captura de vídeo com controle de parâmetros da imagem.
- Acionamento e desligamento dos oito pinos GPIO.
- Agendamento dos oito pinos GPIO.
- *Download* e visualização das imagens capturadas.
- *Download* e visualização dos vídeos capturados.

As seções a seguir descrevem as etapas do desenvolvimento, seguido pela descrição do funcionamento e implementação do servidor, do cliente e da comunicação.

3.1 Descrição das Etapas de Desenvolvimento

O primeiro passo no desenvolvimento do trabalho foi a criação da comunicação, para isso foram criados dois *softwares* simples que se conectavam por *socket* e trocavam mensagens. Após a verificação do funcionamento e estabelecimento dessa conexão inicial, o *software* servidor foi incrementado para permitir a conexão de mais dispositivos ao mesmo tempo.

O passo seguinte foi a pesquisa e teste das maneiras disponíveis para geração do PWM. Com o método PiBlaster escolhido, foi feita a implementação do acionamento do PWM através da linguagem Java. O acelerômetro do *smartphone* foi mapeado para controlar o valor do PWM, com isso, o posicionamento da câmera era controlado pelo movimento do *smartphone*.

Posteriormente, foi realizado a instalação, configuração e testes inicias da câmera. Após o aprendizado básico da câmera, foi feita a implementação de seu acionamento e controle através da linguagem Java.

Com isso o servidor tinha suas principais funções implementadas, então foi iniciado a criação do aplicativo Android. A primeira versão criada possibilitava a captura de imagem sem alteração dos parâmetros e a geração de PWM para dois servos utilizados para controle da câmera.

Com versões básicas do servidor e do cliente, foi iniciado o incremento da comunicação com a criação das requisições.

Durante o incremento de ambos os softwares, cliente e servidor, eram realizados testes de uso. Em um desses testes foi detectado um grave problema: o método escolhido para geração do PWM alterava todas as saídas da Raspberry Pi, inclusive na saída utilizada pela câmera, causando seu mal funcionamento. Os outros métodos de PWM foram testados, porém nenhum gerou o resultado esperado. A opção escolhida foi a não utilização do PWM, perdendo com isso o controle de movimento da câmera.

Com a incompatibilidade eliminada, foram incrementadas as funções de GPIO, com a expansão para oito pinos e criação do agendamento dos mesmos. O mesmo agendamento foi feito para captura de imagem e vídeo, seguido da criação do *streaming* e da captura de imagem em tempo real.

Finalizadas as funções de câmera e GPIO, o próximo passo foi a organização, comunicação e transferência dos arquivos gerados para o Android.

Com servidor e cliente funcionando, o passo final foi a organização e melhoria visual do aplicativo.

3.2 Servidor - Raspberry Pi

A programação do servidor poderia ter sido realizada em diversas linguagens, necessitando apenas que oferecesse acesso aos *sockets*, mas a linguagem escolhida foi Java, devido ao fato do cliente Android ser necessariamente programado em Java, assim tanto cliente como servidor compartilham a mesma linguagem. Existe também o uso da linguagem XML para armazenar algumas informações relacionadas ao agendamento das funções do servidor.

Para descrever o funcionamento com mais detalhes os tópicos a seguir abordam as funcionalidades e características do servidor, mostrando quais as estratégias e ferramentas foram utilizadas para sua realização.

Controle dos Agendamentos

Para realizar o controle dos agendamentos o servidor conta com a utilização de arquivos XML para armazenar as informações e com *threads* que acessam, obtém e modificam essas informações.

Quando o servidor é inicializado, são criadas dez *threads*. Essas *threads* são executadas indefinidamente, sendo oito delas responsáveis por verificar o agendamento dos pinos GPIO e duas por verificar o agendamento da câmera, uma para imagem e outra para vídeo.

A cada dez segundos cada uma dessas *threads* realiza a leitura de um arquivo XML específico de sua função, obtendo informações a respeito do agendamento das funções. Através dessas informações a *thread* toma as ações necessárias, ou seja, liga ou desliga o GPIO ou realiza a gravação de vídeo ou captura de imagem de acordo com a informação guardada no XML.

A cada nova programação realizada pelo cliente o arquivo XML deve ser atualizado. Na figura 3.1 é apresentado um exemplo do XML responsável por salvar informações referente ao agendamento da captura de imagem. Nele é possível notar a hora de início, hora de fim, bem como os parâmetros utilizados na captura, como por exemplo qualidade, resolução e brilho.

Nas figuras 3.2 e 3.3 são apresentados exemplos do agendamento da captura de vídeo e do acionamento do pino GPIO número 0.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schedule>
    <picture>
        <startdate>25/12/2010</startdate>
        <starthour>11:00</starthour>
        <enddate>01/01/2011</enddate>
        <endhour>11:00</endhour>
        <interval>60</interval>
        <sizew>720</sizew>
        <sizeh>480</sizeh>
        <quality>100</quality>
        <brightness>50</brightness>
        <contrast>0</contrast>
        <saturation>0</saturation>
        <sharpness>0</sharpness>
        <effect>none</effect>
        <exposure>auto</exposure>
    </picture>
</schedule>
```

Figura 3.1: Estrutura do XML correspondente a captura de imagem

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schedule>
    <video>
        <startdate>04/08/2013</startdate>
        <starthour>12:00</starthour>
        <enddate>04/08/2013</enddate>
        <endhour>12:30</endhour>
        <sizew>720</sizew>
        <sizeh>480</sizeh>
        <brightness>50</brightness>
        <contrast>0</contrast>
        <saturation>0</saturation>
        <sharpness>0</sharpness>
        <effect>none</effect>
        <exposure>auto</exposure>
    </video>
</schedule>
```

Figura 3.2: Estrutura do XML correspondente a gravação de vídeo

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schedule>
    <gpio0>
        <state>0.0</state>
        <startdate>08/10/2013</startdate>
        <starthour>18:18</starthour>
        <enddate>08/10/2013</enddate>
        <endhour>18:20</endhour>
    </gpio0>
</schedule>
```

Figura 3.3: Estrutura do XML correspondente ao GPIO número 0

Gerenciamento de Clientes

Depois de iniciadas as *threads*, o servidor cria um *socket* que fica a espera de clientes.

Quando um cliente realiza a conexão com o servidor, o gerenciamento da conexão é transferido para uma nova *thread*, permitindo que o servidor aceite novas conexões. A figura 3.4 ilustra o funcionamento das *threads*.

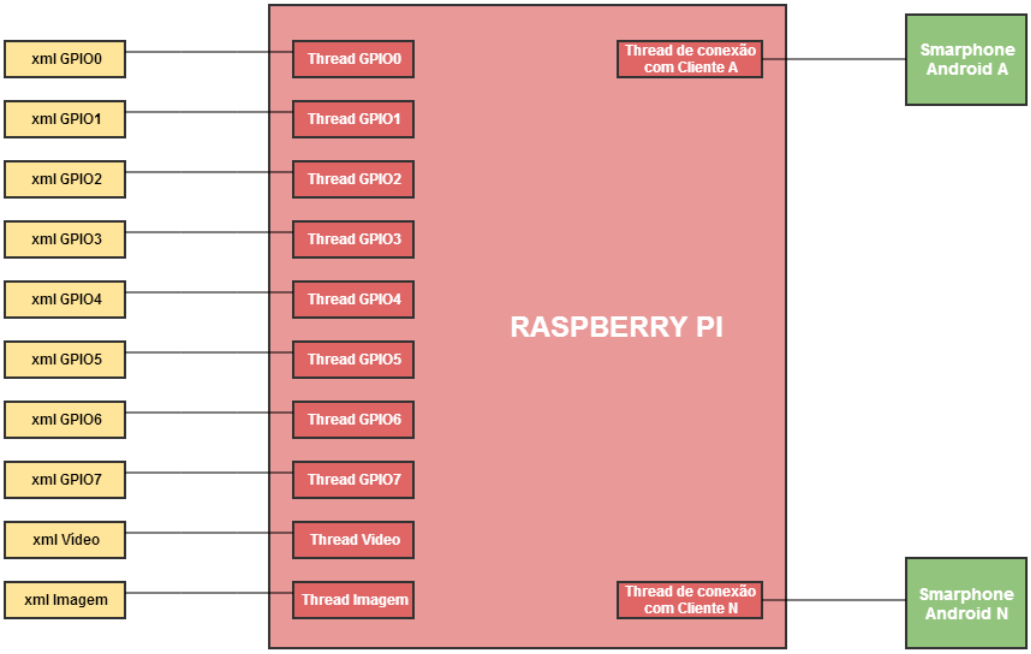


Figura 3.4: Estrutura das *threads* no servidor

Após conectado, o servidor fica na espera de requisições do cliente. Ao recebe-las, o servidor faz a analise, realiza as tarefas requisitadas e retorna uma mensagem ao cliente. As requisições, bem como exemplos da comunicação serão mostradas mais a frente nesta seção.

Utilização da Câmera

O uso da câmera se baseia no uso de programas oferecidos pela própria Raspberry Pi. Esses programas são executados diretamente no *bash*. O *bash* é um interpretador de comandos, uma espécie de tradutor entre o sistema operacional e o usuário.

Para a execução desses programas diretamente no *bash* foi necessário o uso de classe *Runtime* em Java que permite a interação da aplicação com o ambiente no qual está sendo executada, assim o servidor consegue executar ambos os programas como se fossem executados pelo próprio cliente diretamente no *bash*.

O programa *raspistill* é responsável pela captura de imagens e o *raspivid* pela captura de

vídeo. Os parâmetros utilizados na execução são obtidos das requisições realizadas pelo cliente. Esses parâmetros são listados nas tabelas 3.1 e 3.2.

Tabela 3.1: Parâmetros oferecidas pelo *raspistill* [11]

Comando	Descrição
-?	Ajuda
-w	Define largura da imagem
-h	Define altura da imagem
-q	Define a qualidade do jpeg (0 a 100)
-r	Adiciona dados ao metadata do jpeg
-o	Nome do arquivo
-v	Informações detalhadas de saída durante a execução
-t	Tempo antes de tirar a foto (ms)
-th	Definie parametros do *thumbnail*
-d	Roda um modo demo, sem captura
-e	Define o tipo de *encoding* (jpg, bmp, gif, png)
-x	*EXIF tag*
-tl	Modo *timelapse*
-sh	Define o *sharpness* (-100 a 100)
-co	Define o contraste (-100 a 100)
-br	Define o brilho (0 a 100)
-sa	Define a saturação (-100 a 100)
-ISO	Define o ISO
-vs	Liga a estabilização de vídeo
-ev	Define a compensação EV (-10 a 10)
-ex	Define o modo de exposição
-awb	Habilita o modo AWB
-ifx	Define o efeito da imagem
-cfx	Define os efeitos de cores
-mm	Define o modo *metering*
-rot	Define a rotação da imagem(0 a 359)
-hf	Define o *flip* horizontal
-vf	Define o *flip* vertical

Tabela 3.2: Parâmetros oferecidas pelo *raspivid* [11]

Comando	Descrição
-?	Ajuda
-w	Define largura da imagem
-h	Define altura da imagem
-b	Define o *bitrate*
-o	Nome do arquivo
-v	Informações detalhadas de saída durante a execução
-t	Tempo antes de tirar a foto (ms)
-d	Roda um modo demo, sem captura
-fps	Definie o numero de imagens por segundo
-e	Define o tipo de *encoding* (jpg, bmp, gif, png)
-sh	Define o *sharpness* (-100 a 100)
-co	Define o contraste (-100 a 100)
-br	Define o brilho (0 a 100)
-sa	Define a saturação (-100 a 100)
-ISO	Define o ISO
-vs	Liga a estabilização de vídeo
-ev	Define a compensação EV (-10 a 10)
-ex	Define o modo de exposição
-awb	Habilita o modo AWB
-ifx	Define o efeito da imagem
-cfx	Define os efeitos de cores
-mm	Define o modo *metering*
-rot	Define a rotação da imagem(0 a 359)
-hf	Define o *flip* horizontal
-vf	Define o *flip* vertical

Soluções para o Streaming

Para o *streaming* de vídeo foram considerados vários métodos, entre eles, os já citados: *Motion*, *MJPEG* e *FFmpeg*. Todos eles se baseiam no dispositivo mapeado no arquivo */dev/video0* do Linux.

O módulo da câmera da Raspberry Pi não fornece o *driver* necessário para o mapeamento no arquivo */dev/video0*, assim foi necessário a busca de novas soluções. *Observação: atualmente usuários da Raspberry Pi estão em um projeto para produção um driver para a câmera, porém, a última versão testada ainda conta com instabilidade para vídeos H264.*

A solução encontrada foi o uso do *streaming* através do uso do pacote VLC. Assim o servidor consegue que seu vídeo passe a ser transmitido por uma porta definida pelo programador.

O cliente pode visualizar o *streaming* através programa VLC em qualquer computador pessoal, através do endereço fornecido pelo aplicativo, como será mostrado nas seções seguintes.

Assim como os programas *raspistill* e *raspivid*, o VLC também é executado no *bash* e como a função de *streaming* pode rodar durante um tempo indefinido, o processo responsável pelo *streaming* é executado em um *thread* em background, para que o acesso ao *bash* não seja bloqueado.

Controle e acesso dos pinos de gpio

Para controle e acesso dos pinos GPIO existem diversos *wrappers* em diversas linguagens que oferecem as mesmas funcionalidades. Por exemplo, a biblioteca *WiringPi* para C, a biblioteca *RaspberryPi.Net* para C *sharp*, o módulo *RPi.GPIO* para Python, entre outros.

Para este projeto foi utilizado o *Pi4j*, um projeto destinado a fornecer uma ponte entre as bibliotecas nativas e o Java para acesso total a Raspberry Pi. Além do controle do GPIO, *Pi4j* o ainda oferece suporte para comunicação serial, I2C e SPI.

Sua utilização é bem simples, como mostra o trecho de código a seguir.

```
final GpioController gpio = GpioFactory.getInstance();
GpioPinDigitalOutput myGpio = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_04, "My GPIO", PinState.LOW);
myGpio.setState(PinState.HIGH);
myGpio.setState(PinState.LOW);
```

Gerenciamento dos arquivos de imagem e vídeos

Todos os arquivos gerados são armazenados na Raspberry Pi. Eles podem ser de imagem no formato *jpg* ou de vídeo, nos formatos *H264* ou *MP4*.

A quantidade de imagens e vídeos armazenados variam de acordo com o cartão SD utilizado na Raspberry PI. Existe a possibilidade de se utilizar um HD externo para aumentar a capacidade de armazenamento.

Caso haja necessidade de conversão do formato *H264* para outros formatos, pode-se utilizar o programa MP4Box que é uma ferramenta baseada em linha de comando que converte arquivos dos tipos MPEG-4, DivX, XviD, 3ivx e *H264* em arquivo *MP4* [5]. Sua utilização é simples, rápida e os resultados são de boa qualidade.

Além da conversão, o servidor possibilita a opção do cliente realizar o *download* do arquivo para depois visualizá-lo no próprio *smartphone*.

3.3 Cliente - Aplicativo Android

A programação do aplicativo é mais restrita que a do servidor. Por se tratar de um aplicativo Android a programação deve ser feita em Java e a parte visual deve ser feita em XML.

Os tópicos a seguir abordam o funcionamento do aplicativo de acordo com cada tela apresentada.

Conexão e Tela Principal

A primeira tela do aplicativo que o usuário tem contato é a tela de *login*. O usuário informa o IP, a porta do servidor e realiza a conexão, como mostra a figura 3.5.

Após estabelecido a conexão, é apresentado ao usuário a tela principal. Nela existem os três principais blocos de funções disponíveis: Câmera, GPIO e Arquivos, como mostra a figura 3.6. A seguir são listados detalhadamente o que cada um desses blocos contém e o seu funcionamento.

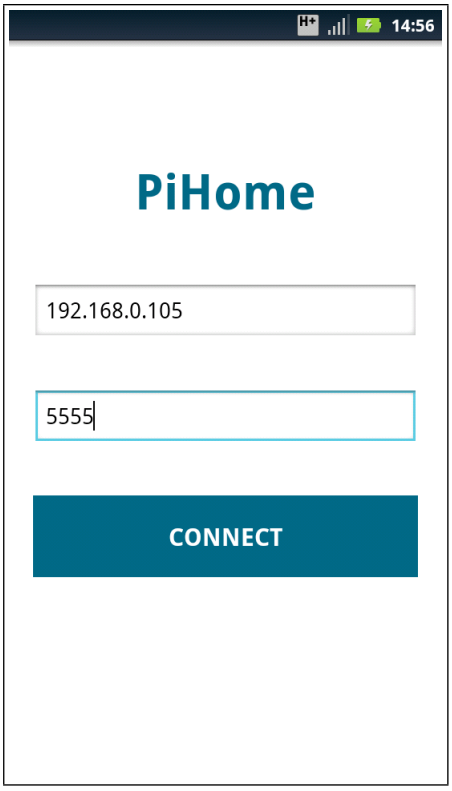


Figura 3.5: Tela de *login* do aplicativo

Figura 3.6: Tela principal do aplicativo

Funções da Câmera

Como exibido na figura 3.7, existem quatro funções relacionado à câmera:

- Captura de imagem
- Agendamento da captura de imagem
- Agendamento da captura de vídeo
- Gerenciamento do *streaming*

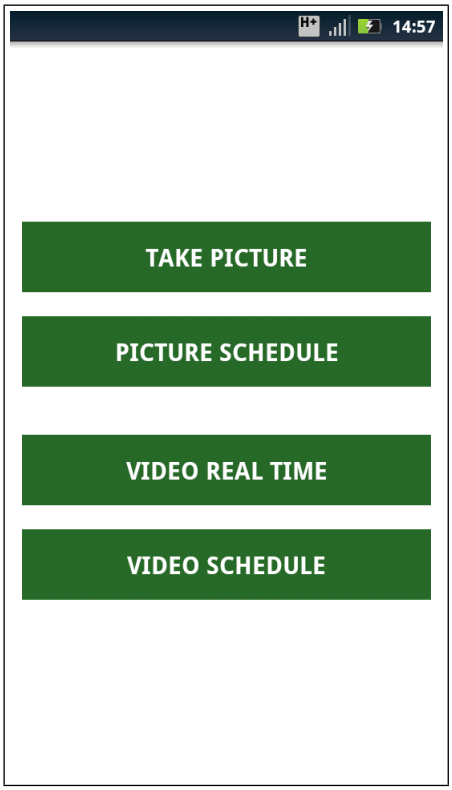


Figura 3.7: Tela com as funções relacionadas a câmera

Na opção **captura de imagem** o usuário pode escolher como deseja capturar a imagem, podendo variar os parâmetros: resolução, qualidade, brilho, contraste, saturação, *sharpness*, efeito e exposição, como mostra na figura 3.8.

A opção de **agendamento para captura de imagem** oferece ao usuário captura de imagens de tempo em tempo durante um intervalo desejado.

É possível escolher as mesmas características presentes na captura de imagem, além da data/hora de início/fim e o intervalo, em minutos, que se deseja entre as fotos. Ao acessar a

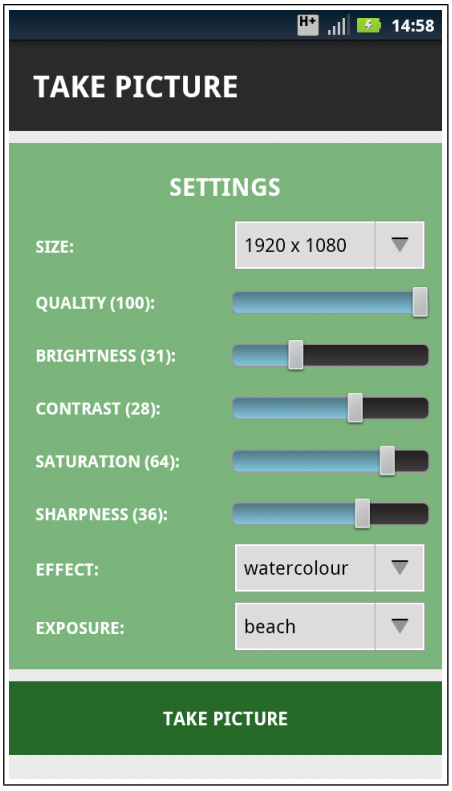


Figura 3.8: Tela utilizada para captura de imagem

opção de agendamento, é mostrado ao usuário o último agendamento e os parâmetros utilizados pelo servidor. A figura 3.9 ilustra um exemplo dessa função.

Na opção de **streaming** é possível ligá-lo e desligá-lo. Quando ligado é fornecido um link que o usuário deve usar em qualquer computador pessoal para visualizar o *streaming* através do programa VLC. Existe a opção de receber o link por email para facilitar o acesso. A imagem 3.10 demonstra as duas opções, ligado e desligado.

A idéia inicial para a opção de *streaming* era a visualização direta do vídeo captado pela Raspberry Pi no *smartphone*, porém nenhuma das opções testadas teve desempenho aceitável, demonstraram excessiva lentidão, com grandes atrasos e congelamentos devido a baixa maleabilidade do formato *H264* em *smartphones*.

A opção de **agendamento para captura de vídeo** oferece ao usuário a captura de vídeo durante um tempo definido. O usuário escolhe a data/hora de início/fim para a captura e os parâmetros: resolução, brilho, contraste, saturação, *sharpness*, efeito e exposição. A figura 3.11 mostra um exemplo desta função. Ao acessar a opção de agendamento é mostrado ao usuário o último agendamento e os parâmetros utilizados pelo servidor, semelhante ao agendamento da captura de imagem.

Figura 3.9: Tela utilizada para agendamento da captura de imagem

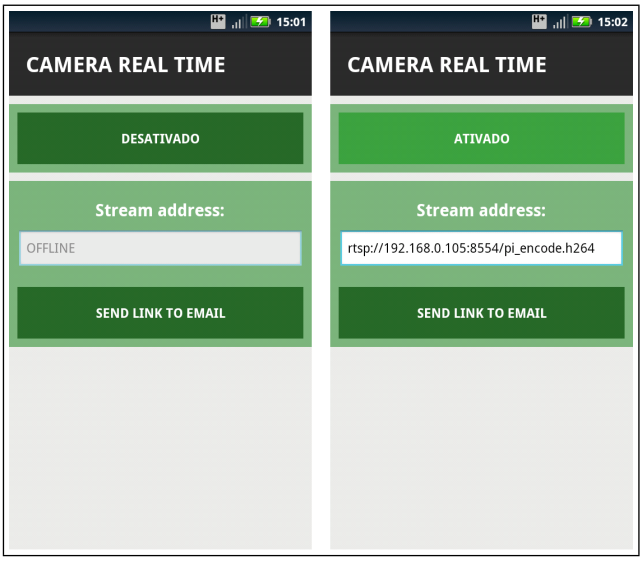


Figura 3.10: Tela de controle do *streaming*

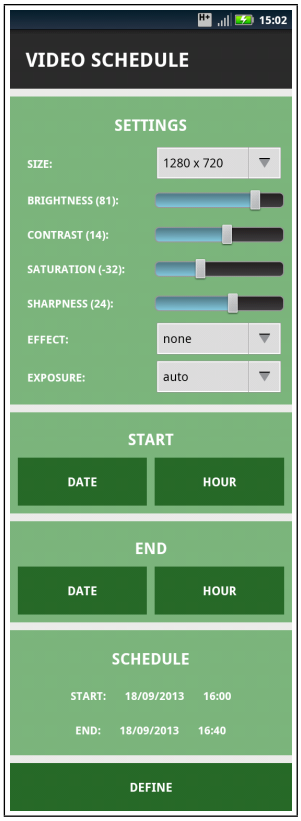


Figura 3.11: Tela utilizada para agendamento da captura de vídeo

Funções relacionadas ao GPIO

A segunda opção, GPIO, é uma função com uso ilimitado. Pode ser utilizado para ativar diversos sensores e atuadores, por exemplo, acionamento de alarmes e lâmpadas para segurança residencial, ou então, acionamento de umificadores, aquecedores e ventiladores para uso em uma estufa.

É apresentado ao usuário todos os oito pinos GPIO disponíveis na Raspberry Pi, como mostra a figura 3.12, cada um tem seu uso e gerenciamento individual, sem influência dos demais.

Para os todos os pinos GPIO o usuário pode optar por ligar e desligar diretamente acionando o botão no canto superior direito do aplicativo, ou optar pelo agendamento. Para o agendamento o usuário escolhe a data/hora que deseja ligar e a data/hora que deseja desligar. A figura 3.13 mostra um exemplo para a pino zero.

Figura 3.12: Tela inicial de acesso para aos pinos GPIO

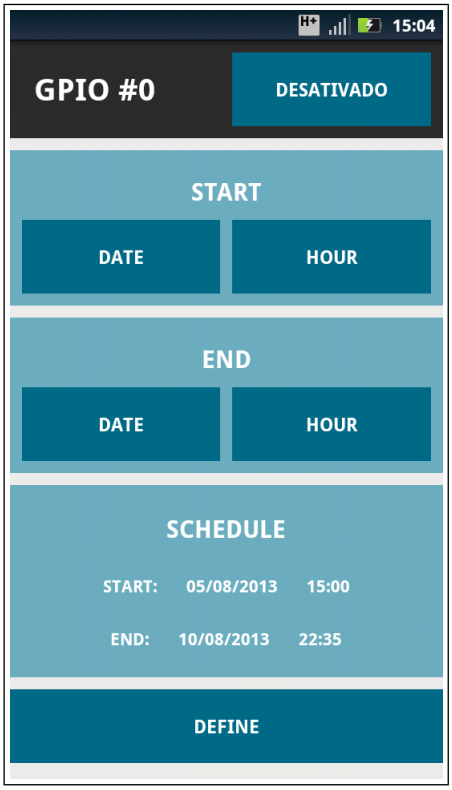


Figura 3.13: Tela de controle e agendamento do pino GPIO 0

Funções relacionadas aos arquivos

Na terceira opção o usuário tem contato com os arquivos de vídeo e imagem gerados. Inicialmente o usuário deve optar se deseja visualizar a lista de arquivos de imagem ou de vídeo, como apresentado na figura 3.14.

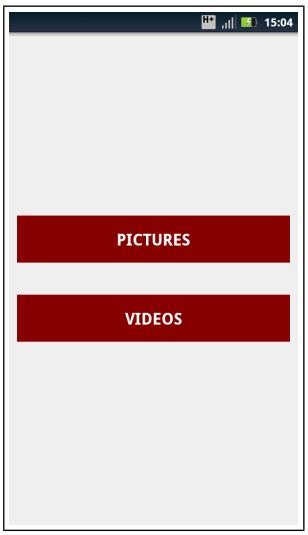
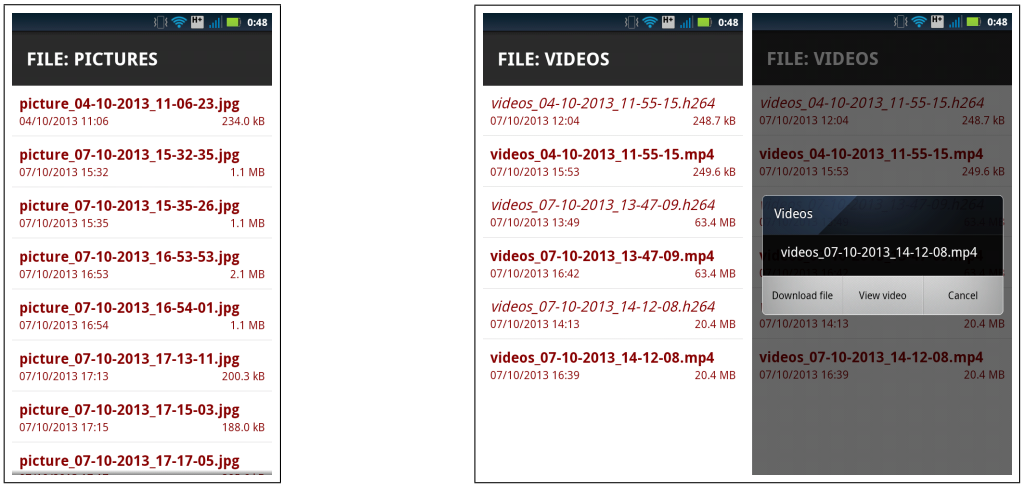


Figura 3.14: Tela inicial para escolha dos arquivos

Após escolher entre imagem ou vídeo, o usuário se depara com uma lista de arquivos disponíveis no servidor. Baseado nessa lista é possível realizar o *download* de qualquer arquivo para o *smartphone* e a visualização do mesmo através do próprio aplicativo. A figura 3.15 exibe um exemplo para imagens e vídeos.

Como a reprodução de vídeos no formato *H264* não é tão simples em *smartphones*, existe a opção de conversão dos vídeos para o formato *MP4*. Essa conversão é realizada no servidor que mantém ambos os arquivos.



(a) Arquivos de imagem (b) Arquivos de vídeo e suas opções

Figura 3.15: Telas com arquivos de imagem e vídeo

3.3.1 Comunicação

A comunicação entre cliente e servidor pode utilizar até 3 portas, dependendo do uso que o cliente deseja.

Como opção de projeto foram escolhidas as portas de número 5555, 5556 e 8854, por se tratarem de portas genéricas e disponíveis para uso.

A principal porta é a 5555. É nela que ocorre todo o fluxo das requisições, sendo responsável por todas as mensagens trocadas entre servidor e cliente.

A porta 5556 é utilizada apenas para a realização do *download* de arquivos do servidor para o cliente. A requisição para o *download* é feita através da porta 5555, o cliente responde na mesma porta 5555 e então libera a porta 5556 para o *download*.

A porta 8854 é utilizada para a o *streaming*. Pelo motivos citados no capitulo anterior, essa porta é utilizada apenas entre a Raspberry Pi e um computador pessoal para transmissão do vídeo em tempo real. Assim como no *download*, a troca de requisições também é feita na porta 5555 e ao final o *streaming* é realizado pela porta 8854.

A figura 3.16 mostra as relação entre as portas do servidor e dos clientes.

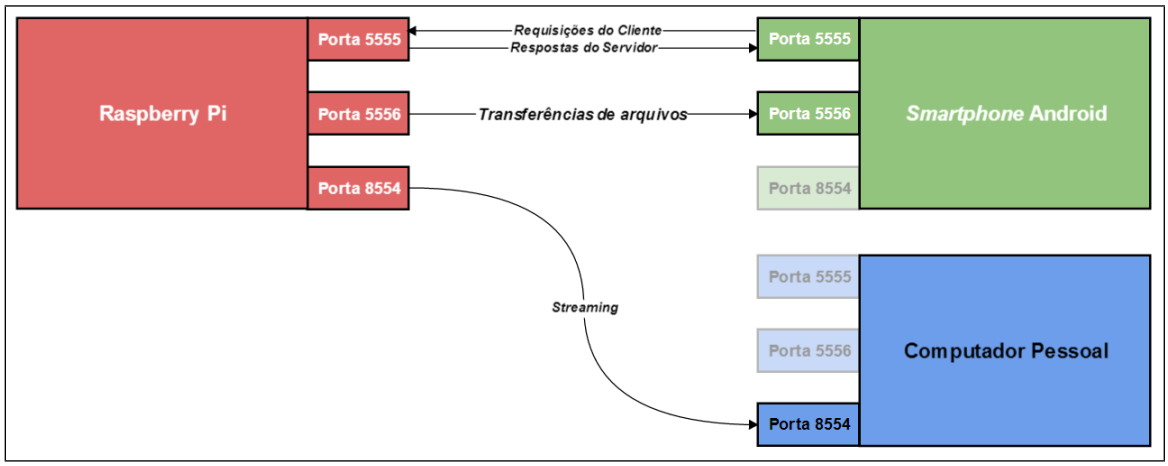


Figura 3.16: Esquema de comunicação do projeto entre Cliente e Servidor

Toda comunicação criada entre cliente e servidor é baseada na troca de mensagens. Todas as mensagens são constituídas por campos, separados por espaços, e seguem o mesmo padrão. O primeiro campo corresponde ao tipo da requisição. Os demais campos variam de acordo com a função:

```
Campo1 Campo2 Campo3 ... CampoN
Requisição Dado1 Dado2 ... DadoN
```

As possíveis requisições e os valores que as identificam são listadas a seguir com uma breve explicação de cada uma, seguidas por exemplos de comunicação.

Requisições do GPIO

Existem três requisições ligadas a função dos pinos GPIO. O trecho em Java a seguir foi extraido do código do servidor e lista as possíveis requisições:

```
public static final int MSG_WHAT_GPIO = 1;
public static final int MSG_WHAT_GPIO_PROGRAMMATION = 2;
public static final int MSG_WHAT_GPIO_PROGRAMMATIONREQUEST = 3;
```

A primeira requisição, "***MSG_WHAT_GPIO***" , é utilizada quando o usuário deseja ligar ou desligar qualquer pino GPIO. A mensagem enviada pelo cliente tem o seguinte formato:

```
1 pino_gpio estado_desejado
```

O servidor retorna um *echo* da mensagem recebida. No exemplo de comunicação a seguir, o cliente requisita que o pino 0 seja ligado (1):

- Cliente envia: *1 0 1*
- Servidor responde: *1 0 1*

A requisição "***MSG_WHAT_GPIO_PROGRAMMATION***" é utilizada quando o usuário realiza o agendamento de um pino GPIO. A mensagem enviada pelo cliente tem o seguinte formato:

```
2 pino_gpio estado_pino data_inicial hora_inicial data_final hora_final
```

O servidor retorna um *echo* da mensagem recebida. Neste exemplo, o pino GPIO 0 encontra-se desligado (0), será ligado no dia 20/09/2013 às 10:00 horas e desligado 20/09/2013 às 11:30 horas.

- Cliente envia: 2 0 0 20/9/2013 10:00 20/9/2013 11:30
- Servidor responde: 2 0 0 20/10/2013 10:00 20/10/2013 11:30

A requisição "***MSG_WHAT_GPIO_PROGRAMMATIONREQUEST***" é utilizada toda vez que o usuário acessa a tela relacionada com GPIO. Essa requisição é responsável por obter a última hora agendada e o estado atual do pino. A mensagem enviada pelo cliente tem o seguinte formato:

```
3 pino_gpio
```

O servidor retorna o estado atual do pino GPIO e a data/hora de início/fim do agendamento. O formato dessa mensagem é o seguinte:

```
3 estado_pino data_inicial hora_inicial data_final hora_final
```

O exemplo a seguir mostra uma requisição do agendamento relacionado ao pino GPIO 3. O servidor informa que o estado atual é desligado (0), programado para ligar no dia 01/10/2013 às 00:52 e desligar no dia 05/10/2013 às 10:55.

- Cliente envia: 3 3
- Servidor responde: 3 0 1/10/2013 00:52 05/10/2013 10:55

Requisições da Câmera

Para a câmera, existem sete requisições, sendo 4 de vídeo e 3 de imagem. Todas elas são listadas a seguir:

```
public static final int MSG_WHAT_CAMERA_VIDEO_REALTIME = 4;
public static final int MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS = 5;
public static final int MSG_WHAT_CAMERA_VIDEO_PROGRAMMATION = 6;
public static final int MSG_WHAT_CAMERA_VIDEO_PROGRAMMATIONREQUEST = 7;

public static final int MSG_WHAT_CAMERA_PICTURE_TAKEPICTURE = 8;
public static final int MSG_WHAT_CAMERA_PICTURE_PROGRAMMATION = 9;
public static final int MSG_WHAT_CAMERA_PICTURE_PROGRAMMATIONREQUEST = 10;
```

As duas primeiras requisições, "***MSG_WHAT_CAMERA_VIDEO_REALTIME***" e "***MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS***" são bem simples. A primeira habilita e desabilita o serviço de *streaming*. A segunda é utilizada para verificar a situação do *streaming*, se está ligado ou desligado.

A mensagem enviada pelo cliente é formada apenas pelo valor 4 ou 5. O servidor retorna um *echo* da mensagem recebida.

A requisição seguinte, "***MSG_WHAT_CAMERA_VIDEO_PROGRAMMATION***" é semelhante a "***MSG_WHAT_CAMERA_PICTURE_PROGRAMMATION***". Ambas são utilizadas quando o usuário realiza o agendamento da captura de imagem ou vídeo.

A mensagem enviada pelo cliente tem o seguinte formato:

```
6 data_inicial hora_inicial data_final hora_final parametros[...]
9 data_inicial hora_inicial data_final hora_final parametros[...]
```

O servidor retorna um *echo* da mensagem recebida.

Os dois exemplos a seguir mostram o uso destas variáveis. No primeiro, a gravação de vídeo deve iniciar no dia 15/08/2013 às 05:00 e finalizar no mesmo dia às 05:15. Os valores seguintes se referem, respectivamente, ao tamanho horizontal (1280), tamanho vertical (720), brilho (66), contraste (26), saturação (-24), *sharpness* (52), efeito (*pastel*) e exposição (*auto*).

- Cliente envia: 6 15/08/2013 05:00 15/08/2013 05:15 1280 720 66 26 -24 52 pastel auto
- Servidor responde: 6 15/08/2013 05:00 15/08/2013 05:15 1280 720 66 26 -24 52 pastel auto

No caso de captura de imagem o comando enviado pelo cliente é bem parecido. No exemplo a seguir o cliente deseja que a captura das imagem se inicie no dia 20/10/2013 às 03:00 e finalize no mesmo dia às 06:00, com um intervalo de 5 minutos entre cada imagem. Os valores seguintes se referem, respectivamente, ao tamanho horizontal (1920), tamanho vertical (1080), qualidade (100), brilho (66), contraste (40), saturação (18), *sharpness* (36), efeito (*none*) e exposição (*auto*).

- Cliente envia: 9 20/10/2013 03:00 20/10/2013 06:00 5 1920 1080 100 66 40 18 36 none auto
- Servidor responde: 9 20/10/2013 03:00 20/10/2013 06:00 5 1920 1080 100 66 40 18 36 none auto

A requisição "***MSG_WHAT_CAMERA_VIDEO_PROGRAMMATIONREQUEST***" e "***MSG_WHAT_CAMERA_PICTURE_PROGRAMMATIONREQUEST***" também são semelhantes. Ambas são utilizadas quando o usuário acessa a tela para captura de imagem ou vídeo. São responsáveis

por obter do servidor a última data e hora agendada para as respectivas funções, bem como os parâmetros utilizados.

A mensagem enviada pelo cliente é formada apenas pelo valor 7 para vídeo ou 10 para imagem. O servidor retorna os dados solicitados no seguinte formato:

```
7 data_inicial hora_inicial data_final hora_final parametros[...]
10 data_inicial hora_inicial data_final hora_final parametros[...]
```

Os exemplos a seguir demonstram a requisição do agendamento de vídeo e imagem respectivamente. No primeiro exemplo o servidor informa que a gravação de vídeo deve iniciar no dia 15/08/2013 às 05:00 e finalizar no mesmo dia às 05:15. Os valores seguintes se referem, respectivamente, ao tamanho horizontal (1280), tamanho vertical (720), brilho (66), contraste (26), saturação (-24), *sharpness* (52), efeito (*pastel*) e exposição (*auto*).

- Cliente envia: 7
- Servidor responde: 7 15/08/2013 05:00 15/08/2013 05:15 1280 720 66 26 -24 52 pastel auto

Para o exemplo de imagem o servidor informa que a captura das imagens deve iniciar no dia 20/10/2013 às 03:00 e finalizar no mesmo dia às 06:00 com um intervalo de 5 minutos entre cada imagem. Os valores seguintes se referem, respectivamente, ao tamanho horizontal (1920), tamanho vertical (1080), qualidade (100), brilho (66), contraste (40), saturação (18), *sharpness* (36), efeito (*none*) e exposição (*auto*).

- Cliente envia: 10
- Servidor responde: 10 20/10/2013 03:00 20/10/2013 06:00 5 1920 1080 100 66 40 18 36 none auto

A requisição "***MSG_WHAT_CAMERA_PICTURE_TAKEPICTURE***"é utilizada para captura de imagem. A mensagem enviada pelo cliente tem o seguinte formato:

```
8 parametros[...]
```

O servidor retorna um *echo* da mensagem recebida. No exemplo a seguir o cliente deseja a captura de uma imagem com os seguintes parâmetros: ao tamanho horizontal (720), tamanho vertical (480), qualidade (100), brilho (43), contraste (36), saturação (-38), *sharpness* (60), efeito (*pastel*) e exposição (*auto*).

- Cliente envia: 8 720 480 100 43 36 -38 60 pastel auto
- Servidor responde: 8 720 480 100 43 36 -38 60 pastel auto

Requisições dos Arquivos

Existem cinco requisições ligadas aos arquivos:

```
public static final int MSG_WHAT_FILES_PICTURES_FILELIST = 11;
public static final int MSG_WHAT_FILES_PICTURES_DOWNLOAD = 12;

public static final int MSG_WHAT_FILES_VIDEOS_FILELIST = 13;
public static final int MSG_WHAT_FILES_VIDEOS_DOWNLOAD = 14;
public static final int MSG_WHAT_FILES_VIDEOS_CONVERT = 15;
```

As requisições "***MSG_WHAT_FILES_PICTURES_FILELIST***" e "***MSG_WHAT_FILES_VIDEOS_FILELIST***" são utilizadas para se obter os arquivos disponíveis no servidor. A mensagem enviada pelo cliente consiste apenas o valor 11 para imagem ou 13 para vídeo.

O servidor responde a quantidade de arquivos, o nome do arquivo, a última data de modificação e o tamanho em bytes. O padrão que o servidor utiliza é o seguinte:

```
11  quantidade_de_arquivos  nome @ ultima_data_modificacao @ tamanho
13  quantidade_de_arquivos  nome @ ultima_data_modificacao @ tamanho
```

Observação: no Java, a data da última modificação é um valor em milisegundos desde a data 1 de janeiro de 1970, 00:00:00 GMT, esse valor é convertido no cliente para o formato dd/mm/aaaa.

No exemplo a seguir o servidor informa que existe apenas 1 arquivo de imagem seguido pelo campo com suas informações. Desse campo são obtidos o nome do arquivo: *picture_04_10_2013_11_06_23.jpg*, a última data de modificação: 1380895585000 milisegundos (04/10/2013 14:06) e tamanho: 233.955 bytes.

- Cliente envia: 11
- Servidor responde: 11 1 picture_04_10_2013_11_06_23.jpg@1380895585000@233955

O exemplo a seguir representa a requisição para os arquivos de vídeo. O servidor informa que existem 2 arquivos seguidos de suas informações.

- Cliente envia: 13
- Servidor responde: 13 2 videos_04_10_2013_11_55_15.h264@1381158272000@248683 videos_07_10_2013_14_12_08.mp4@1381174783000@20377408

As requisições "***MSG_WHAT_FILES_PICTURES_DOWNLOAD***" e "***MSG_WHAT_FILES_VIDEOS_DOWNLOAD***" são bem semelhantes, ambas são utilizadas quando o cliente requisita o *download* de um arquivo. A mensagem enviada pelo cliente tem o seguinte formato:

```
12  nome_do_arquivo
14  nome_do_arquivo
```

O servidor retorna um *echo* da mensagem recebida. Os exemplos a seguir ilustram a comunicação para *download* de uma imagem e de um vídeo.

- Cliente envia: 12 picture_04_10_2013_11_06_23.jpg
- Servidor responde: 12 picture_04_10_2013_11_06_23.jpg
- Cliente envia: 14 videos_04_10_2013_11_55_15.mp4
- Servidor responde: 14 videos_04_10_2013_11_55_15.mp4

Por fim, a requisição "***MSG_WHAT_FILES_VIDEOS_CONVERT***" é para converter arquivos de vídeo do formato *H264* para o formato *MP4*. A mensagem enviada pelo cliente tem o seguinte formato:

```
15  nome_do_arquivo
```

O servidor retorna o nome do vídeo, a data da última modificação e o tamanho em bytes no seguinte formato:

```
15  nome_do_arquivo @ ultima_data_modificacao @ tamanho
```

O exemplo a seguir ilustra a conversão de um arquivo.

- Cliente envia: 15 videos_04_10_2013_11_55_15.h264
- Servidor responde: 15 videos_04_10_2013_11_55_15.mp4@1382317532000@498970

4 Resultados e Discussões

4.1 Resultados Obtidos

Este projeto resultou em uma Raspberry Pi atuando como servidor que, conectado a internet, permite ao usuário o controle de todos os seus pinos GPIO, o uso de algumas funções relacionadas a câmera e o acesso, *download* e visualização dos arquivos geradas por estas funções. Tudo isso feito remotamente através de um *smartphone*.

O *Apêndice A* apresenta algumas imagens capturadas na versão final do projeto com diferentes parâmetros.

Apesar de contar com um grande número de *threads* e de constantes verificações a arquivos, o servidor não apresentou atrasos para a execução e resposta das requisições. A utilização da *cpu* atingiu picos de vinte por cento durante uso intenso, exceto durante a conversão do vídeo, que o uso da cpu é o máximo possível.

Resultou também em um aplicativo para *smartphone* Android, que possibilita a comunicação com o servidor e o acesso as funções disponibilizadas através da internet.

Nos *smartphones* testados, o aplicativo apresentou uma utilização fluida, respondendo rápido aos comandos do usuário, sem travamentos ou *crashs* inesperados. O fator negativo foi em relação a execução de vídeos *H264*, que é prejudicada em alguns *smartphones* de baixo custo. A solução foi a adoção de vídeos no formato *MP4*.

Para a comunicação, os principais testes foram realizados em rede local, assim a influência de possíveis atrasos pela variação da internet foram descartados.

Os testes mostraram que ambos os *softwares* trocam mensagens praticamente de forma instantânea. A comunicação se mostrou estável, sem perdas de mensagem ou de conexão.

O desempenho das funções de arquivo pode ser visto na tabela 4.1 que exibe os tempos necessários para conversão e *download* de dois arquivos de vídeo com tamanhos diferentes.

Tabela 4.1: Tempo para conversão e *download* de arquivos de vídeo

Tamanho do Arquivo	Tempo para conversão	Tempo para *download*
20.4 MB	25 segundos	15 segundos
63.4 MB	70 segundos	58 segundos

Em relação a porta GPIO, não houve o uso de nenhum atuador específico. Seu uso pode ser vasto, cabendo ao usuário a escolha. No projeto existem *leds* nos pinos GPIO para ilustrar o possível agendamento e controle dos teóricos atuadores.

Os demais pinos de comunicação oferecidos pela Raspberry Pi permanecem intactos, podendo ser utilizados sem interferência alguma do projeto.

4.2 Dificuldades e Limitações

A dificuldade inicial foi em relação ao PWM, desde a pesquisa das diferentes formas de geração da onda até a utilização.

A maioria das informações e projetos disponibilizados para PWM são criados por usuários, sendo alguns incompletos ou mal documentados.

A utilização esbarrou na limitação que os módulos geradores de PWM criam: a influência/interferência nas demais portas. Com isso é reduzido o número de projetos que podem ser desenvolvido com a utilização desses módulos.

Outra dificuldade foi no ajuste das *threads* que realizam a verificação do agendamento dos pinos GPIO junto aos arquivos XML.

A solução para que essa verificação não ficasse lenta foi a criação de uma *thread* e um arquivo XML para cada GPIO, desse modo, os oito pinos são verificados em paralelo e funcionam individualmente. A utilização de um único arquivo XML não é interessante pois limita o acesso a diferentes pinos no mesmo intervalo de tempo, pois a utilização de qualquer pino bloquearia o acesso aos demais durante o intervalo de tempo que este é utilizado.

Em um trabalho futuro que se deseja utilizar PWM na Raspberry Pi é altamente recomendado o uso de alguma outra comunicação para sua geração, como por exemplo o uso de I2C ou a utilização da saída de áudio com uma pequena alteração no hardware da placa, colocando em curto o capacitor de acoplamento.

5 *Conclusões*

Neste trabalho foram desenvolvidas duas aplicações para duas plataformas distintas, a Raspberry Pi e *smartphones* Android, que permitiram um bom envolvimento com a área de comunicação utilizando *sockets*, com liberdade total de criação, partindo do projeto do protocolo de comunicação até a sua implementação e uso.

Permitiu também o aprofundamento na programação para sistemas Android, dado a construção total do aplicativo, desde a escolha das ilustrações utilizadas até a programação dos *sockets* e das *threads*.

Comparando os resultados com os objetivos iniciais propostos pode-se notar algumas alterações:

- O controle de posicionamento da câmera foi excluído para não alterar o funcionamento dos pinos de comunicação oferecidos pela Raspberry Pi e viabilizar o uso da câmera.
- O *streaming* não pode ser exibido diretamente no aplicativo, porém, seu uso se tornou viável em computadores pessoais com o uso do programa VLC.
- A melhoria no uso da câmera, com a possibilidade de alteração de diversos parâmetros de imagem.
- Maior elaboração no acesso aos arquivos, permitindo conversão de vídeos e *download* de todos arquivos gerados.

A utilização da Raspberry Pi confirmou-se uma escolha correta, pois, considerando todas as características que ela oferece, constitui uma plataforma economicamente viável para entusiastas e amantes de sistemas embarcados, que conta com um grande grupo ativo de usuários, com diferentes projetos surgindo todo os dias pelos fóruns de discussão.

Apesar de Java não ser a linguagem mais recomendada para sistemas embarcados, por não se preocupar com o uso de memória, o servidor não apresentou problemas de performance. Seu

desempenho foi suficiente para a implementação do sistema proposto, havendo possibilidade de futuras expansões.

O Android por sua vez demonstrou ser uma excelente plataforma para criação de aplicativos móveis.

Existem diversos pacotes para as mais variadas funções desejadas e conta com uma infinidade de desenvolvedores em diversos fóruns que participam ativamente na criação de tutoriais e na solução de problemas, além da boa documentação oferecida pelo Google.

Em geral, tanto servidor como cliente apresentaram o funcionamento de todas as funções implementadas sem qualquer tipo de travamento ou lentidão, oferecendo o resultado projetado de cada função.

5.1 Relacionamento entre o Curso e o Projeto

A realização desse projeto possibilitou a utilização de conceitos, ferramentas e técnicas aprendidas durante o curso de Engenharia de Computação para a formação do sistema final.

Os conhecimentos adquiridos em disciplinas de programação, redes de computadores, sistemas embarcados, multimídia e hipermídia, circuitos elétricos e circuitos eletrônicos foram fundamentais para a realização deste projeto, permitindo realizar tarefas de forma mais rápida e prática.

Realizar a integração de tecnologias diversas e ajudar o desenvolvimento de novas e melhores tecnologias é parte do papel de engenheiro de computação e reflete o conhecimento obtido durante o curso de graduação. O desenvolvimento deste projeto possibilitou a verificação de uma ótima maneira de introduzir esse universo de conhecimento de maneira prática e funcional, agregando grande valor técnico, teórico e científico obtido com o desenvolvimento.

5.2 Trabalhos Futuros

Projetos futuros podem ser realizados expandindo o acesso ao servidor e a inserção de novas funções.

A expansão do acesso pode ser feita através da criação de uma pagina web, permitindo que qualquer computador tenha acesso as funções disponíveis.

Para novas funções, pode-se incluir o movimento da câmera, com uso do protocolo I2C.

Existem pequenos dispositivos que permitem o controle de várias saídas PWM através de I2C, um exemplo é o *Adafruit 16 Channel 12 bit PWM/Servo Driver PCA9685* que pemite até 16 sinais PWM a partir de um I2C.

Outra opção para expansão das funções é a inclusão de métodos de monitoramento, sendo possível ao cliente verificar algumas informações do servidor como temperatura, *uptime*, latência da comunicação e quantidade de clientes conectados.

É possível também a transformação do servidor em uma estação multimídia, já que a Raspberry Pi é capaz de reproduzir vídeos em alta resolução, permitindo seu controle através da internet.

Referências Bibliográficas

[1] Adt plugin. Disponível em: http://developer.android.com/tools/sdk/eclipse-adt.html. Acesso em 24 Out. 2013. Citado na página 15.

[2] Arch linux arm. Disponível em: http://archlinuxarm.org/. Acesso em 24 Out. 2013. Citado na página 12.

[3] Ffmpeg. Disponível em: http://www.ffmpeg.org. Acesso em 24 Out. 2013. Citado na página 17.

[4] Motion, a software motion detector. Disponível em: http://www.lavrsen.dk/foswiki/bin/view/Motion/WebHome. Acesso em 24 Out. 2013. Citado na página 16.

[5] Mp4box overview. Disponível em: http://gpac.wp.mines-telecom.fr/mp4box/. Acesso em 24 Out. 2013. Citado na página 26.

[6] Real time streaming protocol (rtsp). Disponível em: http://tools.ietf.org/html/rfc2326:Acesso em 24 Out. 2013. Citado na página 17.

[7] High definition 1080p embedded multimedia applications processor, 2012. Disponível em: http://www.broadcom.com/products/BCM2835. Acesso em 24 Out. 2013. Citado na página 10.

[8] Oficial faq raspberry pi, 2012. Disponível em: http://www.raspberrypi.org/faqs. Acesso em 24 Out. 2013. Citado nas páginas V, 10, 11, e 13.

[9] Raspberry pi os, 2012. Disponível em: http://www.raspberrypi.org/downloads. Acesso em 24 Out. 2013. Citado na página 11.

[10] Raspberry pi quick start guide v1.2, 2012. Disponível em: . Acesso em 24 Out. 2013. Citado na página 10.

[11] Raspicam documentation, 2012. Disponível em: http://www.raspberrypi.org/wp-content/uploads/2013/07/RaspiCam-Documentation.pdf. Acesso em 24 Out. 2013. Citado na página 24.

[12] Site oficial raspberry pi, 2012. Disponível em: http://www.raspberrypi.org/. Acesso em 24 Out. 2013. Citado na página 10.

[13] Getting started with android studio, 2013. Disponível em: http://developer.android.com/sdk/installing/studio.html. Acesso em 24 Out. 2013. Citado na página 15.

[14] Mipi Alliance. Camera interface specifications, 2013. Disponível em: http://www.mipi.org/specifications/camera-interface. Acesso em 24 Out. 2013. Citado na página 12.

[15] Open Handset Alliance. Android overview. Disponível em: http://www.openhandsetalliance.com/android_overview.html. Acesso em 24 Out. 2013. Citado na página 14.

[16] Open Handset Alliance. Industry leaders announce open platform for mobile devices, 2007. Disponível em: http://www.openhandsetalliance.com/press_110507.html. Acesso em 24 Out. 2013. Citado na página 14.

[17] Fernando Campanholli, Ana Alice Vilas Boas, AndrÃ©ia de Souza Pereira, and Gerard Fillion. Aplicabilidade e importancia do celular pro uso pessoal e profissional, 2013. Disponível em: http://www.aedb.br/seget/artigos12/34816478.pdf. Acesso em 24 Out. 2013. Citado na página 7.

[18] Eurodroid, 2013. Disponível em: http://eurodroid.com/2012/09/26/google-celebrates-25-billion-app-downloads-with-25-themed-sales/. Acesso em 24 Out. 2013. Citado nas páginas V e 15.

[19] Saulo Pereira GuimarÃ£es. Brasil Ã© o quarto país do mundo em nÃºmero de smartphones, 2013. Disponível em: http://exame.abril.com.br/tecnologia/noticias/brasil-e-o-quarto-pais-do-mundo-em-numero-de-smartphones. Acesso em 24 Out. 2013. Citado na página 7.

[20] Richard Hirst. Servoblaster, 2013. Disponível em: https://github.com/richardghirst/PiBits/tree/master/ServoBlaster. Acesso em 24 Out. 2013. Citado na página 18.

[21] Thom Holwerda. Raspberry pi to embrace risc os. Disponível em: http://www.osnews.com/story/25276/Raspberry_Pi_To_Embrace_RISC_OS. Acesso em 24 Out. 2013. Citado na página 12.

[22] Daniel Elias Machado Junho. Controle de robo movel embarcado, 2013. Citado na página 8.

[23] Jackson Liam. Mjpg streamer. Disponível em: https://github.com/jacksonliam/mjpg-streamer. Acesso em 24 Out. 2013. Citado na página 16.

[24] Simon Monk. The gpio connector. Disponível em: http://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/the-gpio-connector. Acesso em 24 Out. 2013. Citado nas páginas V, 13, e 14.

[25] SparkFun, 2013. Disponível em: https://www.sparkfun.com/tutorials/304. Acesso em 24 Out. 2013. Citado nas páginas V e 17.

[26] Thomas. Piblaster, 2013. Disponível em: https://github.com/sarfata/pi-blaster/. Acesso em 24 Out. 2013. Citado na página 18.

[27] Brian Womack. Google says 700,000 applications available for android, 2013. Disponível em: http://www.businessweek.com/news/2012-10-29/google-says-700-000-applications-available-for-android-devices. Acesso em 24 Out. 2013. Citado na página 14.

6 *Apêndice A - Imagens capturadas com diferentes parâmetros*

Neste apêndice são apresentadas algumas figuras captadas a partir do projeto finalizado.

A tabela 6.1 exibe os parâmetros utilizado em cada uma das imagens. A figura 6.1 é a imagem de referência, seus parâmetros são os padrões. As imagens seguintes a 6.1 tem apenas um parâmetro alterado por vez.

Tabela 6.1: Parâmetros das imagens capturadas

Figura	6.1	6.2	6.3	6.4	6.5	6.6	6.7	6.8	6.9	6.10
Resolução	1280x720	1280x720	1280x720	1280x720	1280x720	1280x720	1280x720	1280x720	1280x720	1280x720
Qualidade	100	100	100	100	100	100	100	100	100	100
Brilho	50	**70**	50	50	50	50	50	50	50	50
Contraste	0	0	**100**	0	0	0	0	0	0	0
Saturação	0	0	0	**30**	**- 40**	0	0	0	0	0
Sharpness	0	0	0	0	0	**60**	0	0	0	0
Efeito	*none*	*none*	*none*	*none*	*none*	*none*	***colorswap***	***emboss***	***negative***	*none*
Exposição	auto	auto	auto	auto	auto	auto	auto	auto	auto	***off***

Figura 6.1: Imagem capturada com os parâmetros padrões

Figura 6.2: Imagem capturada com 70 de brilho

Figura 6.3: Imagem capturada com 100 de contraste

Figura 6.4: Imagem capturada com 30 de saturacao

Figura 6.5: Imagem capturada com -40 de saturação

Figura 6.6: Imagem capturada com 60 de *sharpness*

Figura 6.7: Imagem capturada com efeito *colorswap*

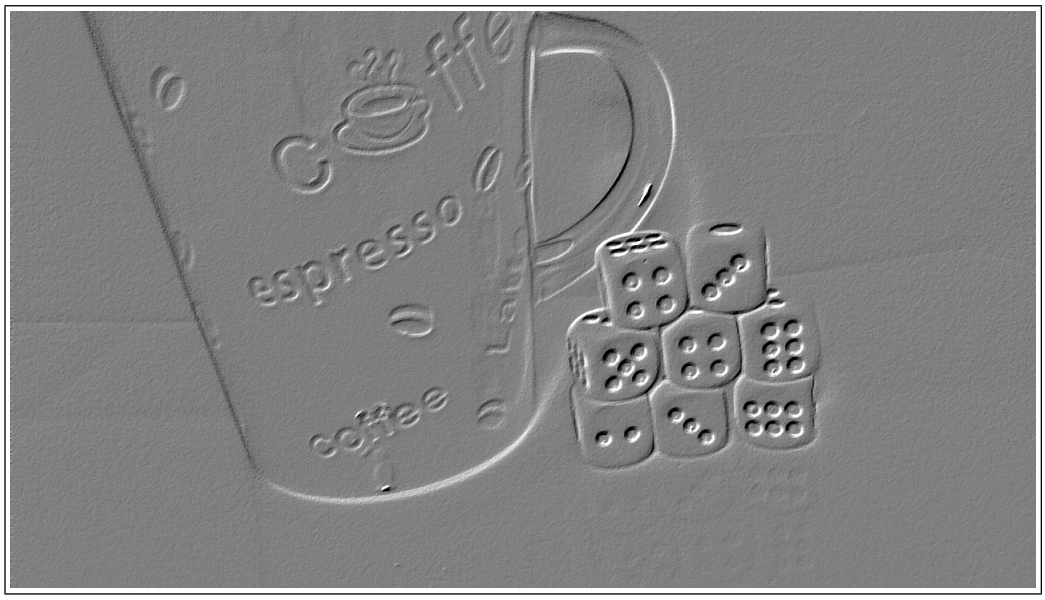


Figura 6.8: Imagem capturada com efeito *emboss*

Figura 6.9: Imagem capturada com efeito *negative*

Figura 6.10: Imagem capturada com exposição *off*

7 Apêndice B - Código do Servidor

7.1 Server

7.1.1 Server.Java

```
public class Server {

public static void main(String argv[]) {

Pi4J.InitGpio();

Gpio0ScheduleThread gpioScheduleThread0 = new Gpio0ScheduleThread();
Gpio1ScheduleThread gpioScheduleThread1 = new Gpio1ScheduleThread();
Gpio2ScheduleThread gpioScheduleThread2 = new Gpio2ScheduleThread();
Gpio3ScheduleThread gpioScheduleThread3 = new Gpio3ScheduleThread();
Gpio4ScheduleThread gpioScheduleThread4 = new Gpio4ScheduleThread();
Gpio5ScheduleThread gpioScheduleThread5 = new Gpio5ScheduleThread();
Gpio6ScheduleThread gpioScheduleThread6 = new Gpio6ScheduleThread();
Gpio7ScheduleThread gpioScheduleThread7 = new Gpio7ScheduleThread();
gpioScheduleThread0.start("0");
gpioScheduleThread1.start("1");
gpioScheduleThread2.start("2");
gpioScheduleThread3.start("3");
gpioScheduleThread4.start("4");
gpioScheduleThread5.start("5");
gpioScheduleThread6.start("6");
gpioScheduleThread7.start("7");

CameraPictureScheduleThread.start();

CameraVideoScheduleThread.start();

waitForConnections(5555);
}

public static void waitForConnections(int serverPort) {
while (true) {
try {
ServerSocket serverSocket = new ServerSocket(serverPort);
Socket socket = null;

System.err.println("SimpleServer: Waiting connection.");
socket = serverSocket.accept();
System.err.println("SimpleServer: Accepted new socket.");

System.err.println("SimpleServer: Creating new handler.");
ServerHandler handler = new ServerHandler(socket);
handler.start();

serverSocket.close();
System.err.println("SimpleServer: Finished with socket.");
} catch (IOException e) {
e.printStackTrace(System.err);
}
```

```
}
}
}
```

7.1.2 ServerHandler.java

```
public class ServerHandler implements Runnable {
private Socket socket = null;
private InputStream inputSocket = null;
private OutputStream outputSocket = null;
private Thread acceptThread = null;

public static final int MSG_WHAT_PWM = 0;

public static final int MSG_WHAT_GPIO = 1;
public static final int MSG_WHAT_GPIO_PROGRAMMATION = 2;
public static final int MSG_WHAT_GPIO_PROGRAMMATIONREQUEST = 3;

public static final int MSG_WHAT_CAMERA_VIDEO_REALTIME = 4;
public static final int MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS = 5;
public static final int MSG_WHAT_CAMERA_VIDEO_PROGRAMMATION = 6;
public static final int MSG_WHAT_CAMERA_VIDEO_PROGRAMMATIONREQUEST = 7;

public static final int MSG_WHAT_CAMERA_PICTURE_TAKEPICTURE = 8;
public static final int MSG_WHAT_CAMERA_PICTURE_PROGRAMMATION = 9;
public static final int MSG_WHAT_CAMERA_PICTURE_PROGRAMMATIONREQUEST = 10;

public static final int MSG_WHAT_FILES_PICTURES_FILELIST = 11;
public static final int MSG_WHAT_FILES_PICTURES_DOWNLOAD = 12;

public static final int MSG_WHAT_FILES_VIDEOS_FILELIST = 13;
public static final int MSG_WHAT_FILES_VIDEOS_DOWNLOAD = 14;
public static final int MSG_WHAT_FILES_VIDEOS_CONVERT = 15;

public ServerHandler(Socket socket) throws IOException {
this.socket = socket;
this.inputSocket = socket.getInputStream();
this.outputSocket = socket.getOutputStream();

this.acceptThread = new Thread(this);

System.out.println("SimpleHandler: New handler created.");
}

public void start() {
acceptThread.start();
}

// All this method does is wait for some bytes from the
// connection, read them, then write them back again, until the
// socket is closed from the other side.
public void run() {
System.out.println("SimpleHandler: Handler run() starting.");
while (true) {
byte[] buf = new byte[1024];
int bytes_read = 0;
try {
// This call to read() will wait forever, until the
// program on the other side either sends some data,
// or closes the socket.
bytes_read = inputSocket.read(buf, 0, buf.length);
if (bytes_read < 0) {
System.out.println("SimpleHandler: Tried to read from socket, read() returned < 0,  Closing socket.");
break;
}

System.out.println("SimpleHandler: Received: " + (new String(buf, 0, bytes_read)));
String input = (new String(buf, 0, bytes_read));

handleMessage(input);
```

```
System.out.println("SimpleHandler: Sending to client: " + (new String(buf, 0, bytes_read)));
outputSocket.write(buf, 0, bytes_read);
outputSocket.flush();

} catch (Exception e) {
e.printStackTrace(System.err);
break;
}
}

try {
System.out.println("SimpleHandler: Closing socket.");
socket.close();
} catch (Exception e) {
System.out.println("SimpleHandler: Exception while closing socket, e = " + e);
e.printStackTrace(System.err);
}

}

private void handleMessage(String input) {

String[] inputMessage = input.replace("\n", "").replace("\r", "").split(" ");

int what = Integer.valueOf(inputMessage[0].replace(" ", ""));
int channelNumber;
double value;

switch (what) {

case MSG_WHAT_PWM:
// channelNumber = Integer.valueOf(inputMessage[1].replace(" ", ""));
// value = Double.valueOf(inputMessage[2].replace(" ", ""));
break;

case MSG_WHAT_GPIO:
channelNumber = Integer.valueOf(inputMessage[1].replace(" ", ""));

value = Double.valueOf(inputMessage[2].replace(" ", ""));
if (value == 0) {
Pi4J.getGpioPinDigitalOutput(Integer.valueOf(channelNumber)).setState(PinState.LOW);
GpioScheduleHandler.writeXmlDataGpioState(String.valueOf(channelNumber), "0.0");
} else if (value == 1) {
Pi4J.getGpioPinDigitalOutput(Integer.valueOf(channelNumber)).setState(PinState.HIGH);
GpioScheduleHandler.writeXmlDataGpioState(String.valueOf(channelNumber), "1.0");
}
break;

case MSG_WHAT_GPIO_PROGRAMMATION:
channelNumber = Integer.valueOf(inputMessage[1].replace(" ", ""));
value = Double.valueOf(inputMessage[2].replace(" ", ""));

String startDate = inputMessage[3].replace(" ", "");
String startHour = inputMessage[4].replace(" ", "");
String endDate = inputMessage[5].replace(" ", "");
String endHour = inputMessage[6].replace(" ", "");
GpioScheduleHandler.writeXmlData(String.valueOf(channelNumber), String.valueOf(value), startDate, startHour, endDate, endHour);
break;

case MSG_WHAT_GPIO_PROGRAMMATIONREQUEST:
channelNumber = Integer.valueOf(inputMessage[1].replace(" ", ""));
try {
if (outputSocket != null) {
ArrayList<String> xmlData = GpioScheduleHandler.getXmlData(String.valueOf(channelNumber));

String gpioProgrammationMessage = MSG_WHAT_GPIO_PROGRAMMATIONREQUEST + " " + xmlData.get(1) + " " + xmlData.get(2) + " "
+ xmlData.get(3) + " " + xmlData.get(4) + " " + xmlData.get(5) + " ";

System.out.println("SimpleHandler: Sending to client (PROGRAMATIONREQUEST): " + gpioProgrammationMessage);
outputSocket.write(gpioProgrammationMessage.getBytes(Charset.forName("UTF-8")));
outputSocket.flush();
}
} catch (NumberFormatException e) {
```

```
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
break;

case MSG_WHAT_CAMERA_VIDEO_REALTIME:
if (inputMessage[1].equalsIgnoreCase("1"))
CameraVideoRealTimeThread.start();
else
CameraVideoRealTimeThread.end();

break;

case MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS:
try {
if (outputSocket != null) {
String realtimeResponse;

if (CameraVideoRealTimeThread.isStreaming)
realtimeResponse = MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS + " " + 1 + " ";
else
realtimeResponse = MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS + " " + 0 + " ";

System.out.println("SimpleHandler: Sending to client (PROGRAMATIONREQUEST): " + realtimeResponse);
outputSocket.write(realtimeResponse.getBytes(Charset.forName("UTF-8")));
outputSocket.flush();
}
} catch (NumberFormatException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
break;

case MSG_WHAT_CAMERA_VIDEO_PROGRAMMATION:
CameraVideoScheduleHandler.writeXmlData(inputMessage[1].replace(" ", ""), inputMessage[2].replace(" ", ""),
inputMessage[3].replace(" ", ""), inputMessage[4].replace(" ", ""), inputMessage[5].replace(" ", ""),
inputMessage[6].replace(" ", ""), inputMessage[7].replace(" ", ""), inputMessage[8].replace(" ", ""),
inputMessage[9].replace(" ", ""), inputMessage[10].replace(" ", ""), inputMessage[11].replace(" ", ""),
inputMessage[12].replace(" ", ""));
break;

case MSG_WHAT_CAMERA_VIDEO_PROGRAMMATIONREQUEST:
try {
if (outputSocket != null) {
ArrayList<String> xmlData = CameraVideoScheduleHandler.getXmlData();
String videoProgrammationMessage = MSG_WHAT_CAMERA_VIDEO_PROGRAMMATIONREQUEST + " " + xmlData.get(0) + " " + xmlData.get(1)
+ " " + xmlData.get(2) + " " + xmlData.get(3) + " " + xmlData.get(4) + " " + xmlData.get(5) + " " + xmlData.get(6)
+ " " + xmlData.get(7) + " " + xmlData.get(8) + " " + xmlData.get(9) + " " + xmlData.get(10) + " " + xmlData.get(11)
+ " ";

System.out.println("SimpleHandler: Sending to client (PROGRAMATIONREQUEST): " + videoProgrammationMessage);
outputSocket.write(videoProgrammationMessage.getBytes(Charset.forName("UTF-8")));
outputSocket.flush();
}
} catch (NumberFormatException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
break;

case MSG_WHAT_CAMERA_PICTURE_TAKEPICTURE:
CameraPictureTakeThread.start(input);
break;

case MSG_WHAT_CAMERA_PICTURE_PROGRAMMATION:
CameraPictureScheduleHandler.writeXmlData(inputMessage[1].replace(" ", ""), inputMessage[2].replace(" ", ""),
inputMessage[3].replace(" ", ""), inputMessage[4].replace(" ", ""), inputMessage[5].replace(" ", ""),
inputMessage[6].replace(" ", ""), inputMessage[7].replace(" ", ""), inputMessage[8].replace(" ", ""),
inputMessage[9].replace(" ", ""), inputMessage[10].replace(" ", ""), inputMessage[11].replace(" ", ""),
inputMessage[12].replace(" ", ""), inputMessage[13].replace(" ", ""), inputMessage[14].replace(" ", ""));
```

```
break;

case MSG_WHAT_CAMERA_PICTURE_PROGRAMMATIONREQUEST:
try {
if (outputSocket != null) {
ArrayList<String> xmlData = CameraPictureScheduleHandler.getXmlData();
//STARTDATE STARTHOUR ENDDATE ENDHOUR INTERVAL SIZEW SIZEH QUALITY BRIGHTNESS CONTRAST SATURATION SHARPNESS EFFECT EXPOSURE
String pictureProgrammationMessage = MSG_WHAT_CAMERA_PICTURE_PROGRAMMATIONREQUEST + " " + xmlData.get(0) + " "
+ xmlData.get(1) + " " + xmlData.get(2) + " " + xmlData.get(3) + " " + xmlData.get(4) + " " + xmlData.get(5) + " "
+ xmlData.get(6) + " " + xmlData.get(7) + " " + xmlData.get(8) + " " + xmlData.get(9) + " " + xmlData.get(10) + " "
+ xmlData.get(11) + " " + xmlData.get(12) + " " + xmlData.get(13) + " ";

System.out.println("SimpleHandler: Sending to client (PROGRAMATIONREQUEST): " + pictureProgrammationMessage);
outputSocket.write(pictureProgrammationMessage.getBytes(Charset.forName("UTF-8")));
outputSocket.flush();
}
} catch (NumberFormatException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
break;

case MSG_WHAT_FILES_PICTURES_FILELIST:
File picturesFolder = new File("/home/pi/tcc/server/pictures");
File[] listOfPictures = picturesFolder.listFiles();
String picturesListMessage = "";
int numPictures = 0;

ArrayList<String> pictureList = new ArrayList<>();

for (int i = 0; i < listOfPictures.length; i++) {
if (listOfPictures[i].isFile()) {
pictureList.add(listOfPictures[i].getName() + "@" + listOfPictures[i].lastModified() + "@" + listOfPictures[i].length());
numPictures++;
}
}

Collections.sort(pictureList);

for (int i = 0; i < pictureList.size(); i++)
picturesListMessage = picturesListMessage + pictureList.get(i) + " ";

picturesListMessage = MSG_WHAT_FILES_PICTURES_FILELIST + " " + numPictures + " " + picturesListMessage;

try {
if (outputSocket != null) {
System.out.println("SimpleHandler: Sending to client (FILELIST): " + picturesListMessage);
outputSocket.write(picturesListMessage.getBytes(Charset.forName("UTF-8")));
outputSocket.flush();
}
} catch (IOException e) {
e.printStackTrace();
}

break;

case MSG_WHAT_FILES_PICTURES_DOWNLOAD:
FilePicturesTransferThread.start(inputMessage[1]);
break;

case MSG_WHAT_FILES_VIDEOS_FILELIST:
File videoFolder = new File("/home/pi/tcc/server/videos");
File[] listOfVideos = videoFolder.listFiles();
String videoListMessage = "";
int numVideos = 0;

ArrayList<String> videoList = new ArrayList<>();

for (int i = 0; i < listOfVideos.length; i++) {
if (listOfVideos[i].isFile()) {
videoList.add(listOfVideos[i].getName() + "@" + listOfVideos[i].lastModified() + "@" + listOfVideos[i].length());
numVideos++;
```

```
}
}

Collections.sort(videoList);

for (int i = 0; i < videoList.size(); i++)
videoListMessage = videoListMessage + videoList.get(i) + " ";

videoListMessage = MSG_WHAT_FILES_VIDEOS_FILELIST + " " + numVideos + " " + videoListMessage;

try {
if (outputSocket != null) {
System.out.println("SimpleHandler: Sending to client (FILELIST): " + videoListMessage);
outputSocket.write(videoListMessage.getBytes(Charset.forName("UTF-8")));
outputSocket.flush();
}
} catch (IOException e) {
e.printStackTrace();
}
break;

case MSG_WHAT_FILES_VIDEOS_DOWNLOAD:
FileVideosTransferThread.start(inputMessage[1]);
break;

case MSG_WHAT_FILES_VIDEOS_CONVERT:
String message;
Process p = null;
try {
p = Runtime.getRuntime().exec(
"MP4Box -add " + "/home/pi/tcc/server/videos/" + inputMessage[1] + " " + "/home/pi/tcc/server/videos/"
+ inputMessage[1].replace("h264", "mp4"));
p.waitFor();

File convertedFile = new File("/home/pi/tcc/server/videos/" + inputMessage[1].replace("h264", "mp4"));

message = MSG_WHAT_FILES_VIDEOS_CONVERT + " " + convertedFile.getName() + "@" + convertedFile.lastModified() + "@"
+ convertedFile.length() + " ";
if (outputSocket != null) {
System.out.println("SimpleHandler: Sending to client (FILELIST): " + message);
outputSocket.write(message.getBytes(Charset.forName("UTF-8")));
outputSocket.flush();
}
} catch (IOException | InterruptedException e) {
e.printStackTrace();
} finally {
p.destroy();
}
break;

default:
break;
}

}
}
```

7.2 GPIO

7.2.1 Gpio0ScheduleThread.java

```
public class Gpio0ScheduleThread implements Runnable {

private int TIME_BETWEEN_CHECKS = 10000;
private static String channelNumber;

public void start(String channel) {
channelNumber = channel;
(new Thread(new Gpio0ScheduleThread())).start();
}
```

```
public void run() {

boolean flagStart = false;
boolean flagEnd = false;

ArrayList<String> xmlData;

Date current;
Date start;
Date end;

try {
while (true) {
while (!flagStart) {

xmlData = GpioScheduleHandler.getXmlData(channelNumber);
current = getCurrentDate();
start = getStartDate(xmlData);
end = getEndDate(xmlData);

if (current.compareTo(start) >= 0 && current.compareTo(end) < 0) {
System.out.println(channelNumber + ": START");
flagStart = true;
Pi4J.getGpioPinDigitalOutput(Integer.valueOf(channelNumber)).setState(PinState.HIGH);
GpioScheduleHandler.writeXmlDataGpioState(channelNumber, "1.0");
} else {
Thread.sleep(TIME_BETWEEN_CHECKS);
}
}
flagStart = false;

while (!flagEnd) {

xmlData = GpioScheduleHandler.getXmlData(channelNumber);
current = getCurrentDate();
start = getStartDate(xmlData);
end = getEndDate(xmlData);

if (current.compareTo(end) >= 0) {
System.out.println(channelNumber + ": END");
flagEnd = true;
Pi4J.getGpioPinDigitalOutput(Integer.valueOf(channelNumber)).setState(PinState.LOW);
GpioScheduleHandler.writeXmlDataGpioState(channelNumber, "0.0");
} else {
Thread.sleep(TIME_BETWEEN_CHECKS);
}
}
flagEnd = false;
}

} catch (InterruptedException e) {
e.printStackTrace();
}
}

private Date getCurrentDate() {
try {
Date date = new Date();
Calendar calendar = GregorianCalendar.getInstance();
calendar.setTime(date);
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(calendar.get(Calendar.DAY_OF_MONTH) + "/"
+ (calendar.get(Calendar.MONTH) + 1) + "/" + calendar.get(Calendar.YEAR) + " " + calendar.get(Calendar.HOUR_OF_DAY) + ":"
+ calendar.get(Calendar.MINUTE));
} catch (ParseException e) {
e.printStackTrace();
}
return null;
}

private Date getStartDate(ArrayList<String> xmlData) {
try {
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(xmlData.get(2) + " " + xmlData.get(3));
```

```
} catch (ParseException e) {
e.printStackTrace();
}

return null;
}

private Date getEndDate(ArrayList<String> xmlData) {
try {
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(xmlData.get(4) + " " + xmlData.get(5));
} catch (ParseException e) {
e.printStackTrace();
}
return null;
}
}
```

7.2.2 GpioScheduleHandler.java

```
public class GpioScheduleHandler {

private static final String GPIO_SCHEDULE_DIR = "/home/pi/tcc/server/gpioSchedule/";

private static final String GPIO = "gpio";
private static final String TAGNAME_STATE = "state";
private static final String TAGNAME_STARTDATE = "startdate";
private static final String TAGNAME_STARTHOUR = "starthour";
private static final String TAGNAME_ENDDATE = "enddate";
private static final String TAGNAME_ENDHOUR = "endhour";

public static boolean writeXmlData(String channelNumber, String state, String startdate, String starthour, String enddate, String endhour) {
try {
DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
Document document = documentBuilder.parse(GPIO_SCHEDULE_DIR + "gpio" + channelNumber + "Schedule.xml");

Node nodeGpio = document.getElementsByTagName(GPIO + channelNumber).item(0);
NodeList nodeList = nodeGpio.getChildNodes();

for (int i = 0; i < nodeList.getLength(); i++) {
Node node = nodeList.item(i);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_STATE))
node.setTextContent(state);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_STARTDATE))
node.setTextContent(startdate);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_STARTHOUR))
node.setTextContent(starthour);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_ENDDATE))
node.setTextContent(enddate);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_ENDHOUR))
node.setTextContent(endhour);
}

TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource domSource = new DOMSource(document);
StreamResult streamResult = new StreamResult(new File(GPIO_SCHEDULE_DIR + "gpio" + channelNumber + "Schedule.xml"));
transformer.transform(domSource, streamResult);

return true;

} catch (ParserConfigurationException pce) {
pce.printStackTrace();
} catch (TransformerException tfe) {
tfe.printStackTrace();
} catch (IOException ioe) {
```

```
ioe.printStackTrace();
} catch (SAXException sae) {
sae.printStackTrace();
}

return false;
}

public static boolean writeXmlDataGpioState(String channelNumber, String state) {
try {
DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
Document document = documentBuilder.parse(GPIO_SCHEDULE_DIR + "gpio" + channelNumber + "Schedule.xml");

Node nodeGpio = document.getElementsByTagName(GPIO + channelNumber).item(0);
NodeList nodeList = nodeGpio.getChildNodes();

for (int i = 0; i < nodeList.getLength(); i++) {
Node node = nodeList.item(i);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_STATE))
node.setTextContent(state);
}

TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource domSource = new DOMSource(document);
StreamResult streamResult = new StreamResult(new File(GPIO_SCHEDULE_DIR + "gpio" + channelNumber + "Schedule.xml"));
transformer.transform(domSource, streamResult);
return true;

} catch (ParserConfigurationException pce) {
pce.printStackTrace();
} catch (TransformerException tfe) {
tfe.printStackTrace();
} catch (IOException ioe) {
ioe.printStackTrace();
} catch (SAXException sae) {
sae.printStackTrace();
}

return false;
}

/* Return: CHANNELNUMBER, STATE, STARTDATE, STARTHOUR, ENDDATE, ENDHOUR */
public static ArrayList<String> getXmlData(String channelNumber) {

ArrayList<String> xmlData = new ArrayList<String>();

try {
File xmlFile = new File(GPIO_SCHEDULE_DIR + "gpio" + channelNumber + "Schedule.xml");

DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
Document document = documentBuilder.parse(xmlFile);
document.getDocumentElement().normalize();

NodeList nodeList = document.getElementsByTagName(GPIO + channelNumber);

for (int i = 0; i < nodeList.getLength(); i++) {
Node node = nodeList.item(i);

if (node.getNodeType() == Node.ELEMENT_NODE) {
Element element = (Element) node;

xmlData.add(channelNumber);
xmlData.add(element.getElementsByTagName(TAGNAME_STATE).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_STARTDATE).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_STARTHOUR).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_ENDDATE).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_ENDHOUR).item(0).getTextContent());
}
}
```

```
} catch (Exception e) {
e.printStackTrace();
}
return xmlData;
}
}
```

7.2.3 Pi4J

```
public class Pi4J {

static GpioPinDigitalOutput myGpio0;
static GpioPinDigitalOutput myGpio1;
static GpioPinDigitalOutput myGpio2;
static GpioPinDigitalOutput myGpio3;
static GpioPinDigitalOutput myGpio4;
static GpioPinDigitalOutput myGpio5;
static GpioPinDigitalOutput myGpio6;
static GpioPinDigitalOutput myGpio7;

public static boolean InitGpio() {
try {
GpioController gpio = GpioFactory.getInstance();
myGpio0 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_00, "MyLED0", PinState.LOW);
myGpio1 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01, "MyLED1", PinState.LOW);
myGpio2 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_02, "MyLED2", PinState.LOW);
myGpio3 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_03, "MyLED3", PinState.LOW);
myGpio4 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_04, "MyLED4", PinState.LOW);
myGpio5 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_05, "MyLED5", PinState.LOW);
myGpio6 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_06, "MyLED6", PinState.LOW);
myGpio7 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_07, "MyLED7", PinState.LOW);
return true;
} catch (Exception e) {
return false;
}
}

public static GpioPinDigitalOutput getGpioPinDigitalOutput(int pinNumber) {
switch (pinNumber) {
case 0:
return myGpio0;
case 1:
return myGpio1;
case 2:
return myGpio2;
case 3:
return myGpio3;
case 4:
return myGpio4;
case 5:
return myGpio5;
case 6:
return myGpio6;
case 7:
return myGpio7;
default:
return null;
}
}
}
```

7.3 Câmera - Imagem

7.3.1 CameraPictureScheduleThread.java

```
public class CameraPictureScheduleThread implements Runnable {
```

```
private int TIME_BETWEEN_CHECKS = 10000;

public static void start() {
(new Thread(new CameraPictureScheduleThread())).start();
}

public void run() {

ArrayList<String> xmlData = CameraPictureScheduleHandler.getXmlData();

Calendar c = Calendar.getInstance();
Date current = getCurrentDate();
Date start = getStartDate(xmlData);
Date end = getEndDate(xmlData);
Date pictureDate = start;
int interval = Integer.parseInt(xmlData.get(4));
int numPictures = 0;

while (true) {
xmlData = CameraPictureScheduleHandler.getXmlData();
current = getCurrentDate();
start = getStartDate(xmlData);
end = getEndDate(xmlData);
pictureDate = start;
interval = Integer.parseInt(xmlData.get(4));

c.setTime(pictureDate);
c.add(Calendar.MINUTE, interval * numPictures);
pictureDate = c.getTime();

if (current.compareTo(pictureDate) == 0 && end.compareTo(pictureDate) >= 0) {
System.out.println("SMILE !!!!");
Process p = null;
try {

DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy_HH-mm-ss");
Date currentDate = new Date();

p = Runtime.getRuntime().exec(
"raspistill" + " -o pictures/picture_" + dateFormat.format(currentDate).toString() + ".jpg" + " -t 1000" + " -w "
+ xmlData.get(5) + " -h " + xmlData.get(6) + " -q " + xmlData.get(7) + " -br " + xmlData.get(8) + " -sh "
+ xmlData.get(9) + " -co " + xmlData.get(10) + " -sa " + xmlData.get(11) + " -ifx " + xmlData.get(12) + " -ex "
+ xmlData.get(13));
p.waitFor();
numPictures++;
} catch (IOException | InterruptedException e) {
e.printStackTrace();
} finally {
p.destroy();
}
} else {
try {
Thread.sleep(TIME_BETWEEN_CHECKS);
} catch (InterruptedException e) {
e.printStackTrace();
}
}

if (end.compareTo(current) <= 0)
numPictures = 0;
}
}

private Date getCurrentDate() {
try {
Date date = new Date();
Calendar calendar = GregorianCalendar.getInstance();
calendar.setTime(date);
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(calendar.get(Calendar.DAY_OF_MONTH) + "/"
+ (calendar.get(Calendar.MONTH) + 1) + "/" + calendar.get(Calendar.YEAR) + " " + calendar.get(Calendar.HOUR_OF_DAY) + ":"
+ calendar.get(Calendar.MINUTE));
} catch (ParseException e) {
e.printStackTrace();
```

```
}
return null;
}

private Date getStartDate(ArrayList<String> xmlData) {
try {
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(xmlData.get(0) + " " + xmlData.get(1));
} catch (ParseException e) {
e.printStackTrace();
}
return null;
}

private Date getEndDate(ArrayList<String> xmlData) {
try {
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(xmlData.get(2) + " " + xmlData.get(3));
} catch (ParseException e) {
e.printStackTrace();
}
return null;
}
}
```

7.3.2 CameraPictureScheduleHandler.java

```
public class CameraPictureScheduleHandler {

private static final String CAMERAPICTURE_SCHEDULE_DIR = "/home/pi/tcc/server/cameraSchedule/pictureSchedule.xml";

private static final String PICTURE = "picture";
private static final String TAGNAME_STARTDATE = "startdate";
private static final String TAGNAME_STARTHOUR = "starthour";
private static final String TAGNAME_ENDDATE = "enddate";
private static final String TAGNAME_ENDHOUR = "endhour";
private static final String TAGNAME_INTERVAL = "interval";
private static final String TAGNAME_SIZEW = "sizew";
private static final String TAGNAME_SIZEH = "sizeh";
private static final String TAGNAME_QUALITY = "quality";
private static final String TAGNAME_BRIGHTNESS = "brightness";
private static final String TAGNAME_CONTRAST = "contrast";
private static final String TAGNAME_SATURATION = "saturation";
private static final String TAGNAME_SHARPNESS = "sharpness";
private static final String TAGNAME_EFFECT = "effect";
private static final String TAGNAME_EXPOSURE = "exposure";

public static boolean writeXmlData(String startdate, String starthour, String enddate, String endhour, String interval, String sizeW,
String sizeH, String quality, String brightness, String contrast, String saturation, String sharpness, String effect, String exposure) {
try {
DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
Document document = documentBuilder.parse(CAMERAPICTURE_SCHEDULE_DIR);

Node nodeGpio = document.getElementsByTagName(PICTURE).item(0);
NodeList nodeList = nodeGpio.getChildNodes();

for (int i = 0; i < nodeList.getLength(); i++) {
Node node = nodeList.item(i);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_STARTDATE))
node.setTextContent(startdate);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_STARTHOUR))
node.setTextContent(starthour);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_ENDDATE))
node.setTextContent(enddate);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_ENDHOUR))
node.setTextContent(endhour);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_INTERVAL))
```

```
node.setTextContent(interval);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_SIZEW))
node.setTextContent(sizeW);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_SIZEH))
node.setTextContent(sizeH);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_QUALITY))
node.setTextContent(quality);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_BRIGHTNESS))
node.setTextContent(brightness);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_CONTRAST))
node.setTextContent(contrast);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_SATURATION))
node.setTextContent(saturation);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_SHARPNESS))
node.setTextContent(sharpness);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_EFFECT))
node.setTextContent(effect);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_EXPOSURE))
node.setTextContent(exposure);

}

TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource domSource = new DOMSource(document);
StreamResult streamResult = new StreamResult(new File(CAMERAPICTURE_SCHEDULE_DIR));
transformer.transform(domSource, streamResult);

return true;

} catch (ParserConfigurationException pce) {
pce.printStackTrace();
} catch (TransformerException tfe) {
tfe.printStackTrace();
} catch (IOException ioe) {
ioe.printStackTrace();
} catch (SAXException sae) {
sae.printStackTrace();
}

return false;
}

/* Return: STARTDATE STARTHOUR ENDDATE ENDHOUR INTERVAL SIZEW SIZEH QUALITY BRIGHTNESS CONTRAST SATURATION SHARPNESS EFFECT EXPOSURE */
public static ArrayList<String> getXmlData() {

ArrayList<String> xmlData = new ArrayList<String>();

try {
File xmlFile = new File(CAMERAPICTURE_SCHEDULE_DIR);

DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
Document document = documentBuilder.parse(xmlFile);
document.getDocumentElement().normalize();

NodeList nodeList = document.getElementsByTagName(PICTURE);

for (int i = 0; i < nodeList.getLength(); i++) {
Node node = nodeList.item(i);

if (node.getNodeType() == Node.ELEMENT_NODE) {
Element element = (Element) node;
xmlData.add(element.getElementsByTagName(TAGNAME_STARTDATE).item(0).getTextContent());
```

```
xmlData.add(element.getElementsByTagName(TAGNAME_STARTHOUR).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_ENDDATE).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_ENDHOUR).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_INTERVAL).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_SIZEW).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_SIZEH).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_QUALITY).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_BRIGHTNESS).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_CONTRAST).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_SATURATION).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_SHARPNESS).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_EFFECT).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_EXPOSURE).item(0).getTextContent());
}
}
} catch (Exception e) {
e.printStackTrace();
}

return xmlData;
}
}
```

7.3.3 CameraPictureTakeThread.java

```
public class CameraPictureTakeThread implements Runnable {

private static String inputString;

public static void start(String message) {
inputString = message;
(new Thread(new CameraPictureTakeThread())).start();
}

public void run() {

System.out.println("SMILE !!!!");

DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy_HH-mm-ss");
Date currentDate = new Date();

Process p = null;
try {
p = Runtime.getRuntime().exec(
"raspistill" + " -o pictures/picture_" + dateFormat.format(currentDate).toString() + ".jpg" + " -t 1000" + " -w "
+ inputString.split(" ")[1] + " -h " + inputString.split(" ")[2] + " -q " + inputString.split(" ")[3] + " -br "
+ inputString.split(" ")[4] + " -sh " + inputString.split(" ")[5] + " -co " + inputString.split(" ")[6] + " -sa "
+ inputString.split(" ")[7] + " -ifx " + inputString.split(" ")[8] + " -ex " + inputString.split(" ")[9]);
p.waitFor();
} catch (IOException | InterruptedException e) {
e.printStackTrace();
} finally {
p.destroy();
}
}
}
```

7.4 Câmera - Vídeo

7.4.1 CameraVideoThread.java

```
public class CameraVideoRealTimeThread implements Runnable {

private static Process p;

public static boolean isStreaming;
```

```
public static void start() {
isStreaming = false;
(new Thread(new CameraVideoRealTimeThread())).start();
}

public static void end() {
if (p != null)
p.destroy();
}

public void run() {
System.out.println("STREAMING !!!!");

p = null;
try {
isStreaming = true;
p = Runtime.getRuntime().exec("sudo ./stream > /dev/null &");
// p.waitFor();
BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));

@SuppressWarnings("unused")
String line = null;
while ((line = in.readLine()) != null) {
// System.out.println(line);
}
isStreaming = false;
System.out.println("STREAM DONE !!!!");

} catch (IOException e) {
isStreaming = false;
System.out.println("STREAM CLOSED !!!!");
} finally {
if (p != null) {
isStreaming = false;
p.destroy();
}
}
}
}
```

7.4.2 CameraVideoScheduleThread.java

```
public class CameraVideoScheduleThread implements Runnable {

private int TIME_BETWEEN_CHECKS = 10000;

public static void start() {
(new Thread(new CameraVideoScheduleThread())).start();
}

public void run() {

ArrayList<String> xmlData = CameraVideoScheduleHandler.getXmlData();

boolean filming = false;
Date current = getCurrentDate();
Date start = getStartDate(xmlData);
Date end = getEndDate(xmlData);
long duration = end.getTime() - start.getTime();

while (true) {
xmlData = CameraVideoScheduleHandler.getXmlData();
current = getCurrentDate();
start = getStartDate(xmlData);
end = getEndDate(xmlData);
duration = end.getTime() - start.getTime();

if (current.compareTo(start) == 0 && end.compareTo(current) >= 0 && !filming && duration != 0) {
filming = true;
System.out.println("FILMING !!!!");
```

```
Process p = null;
try {
DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy_HH-mm-ss");
Date currentDate = new Date();

p = Runtime.getRuntime().exec(
"raspivid" + " -o videos/videos_" + dateFormat.format(currentDate).toString() + ".h264" + " -t " + duration + " -w "
+ xmlData.get(4) + " -h " + xmlData.get(5) + " -br " + xmlData.get(6) + " -sh " + xmlData.get(7) + " -co "
+ xmlData.get(8) + " -sa " + xmlData.get(9) + " -ifx " + xmlData.get(10) + " -ex " + xmlData.get(11));
p.waitFor();
} catch (IOException | InterruptedException e) {
e.printStackTrace();
} finally {
p.destroy();
}
} else {
try {
Thread.sleep(TIME_BETWEEN_CHECKS);
} catch (InterruptedException e) {
e.printStackTrace();
}
}

if (end.compareTo(current) <= 0) {
filming = false;
}
}
}

private Date getCurrentDate() {
try {
Date date = new Date();
Calendar calendar = GregorianCalendar.getInstance();
calendar.setTime(date);
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(calendar.get(Calendar.DAY_OF_MONTH) + "/"
+ (calendar.get(Calendar.MONTH) + 1) + "/" + calendar.get(Calendar.YEAR) + " " + calendar.get(Calendar.HOUR_OF_DAY) + ":"
+ calendar.get(Calendar.MINUTE));
} catch (ParseException e) {
e.printStackTrace();
}
return null;
}

private Date getStartDate(ArrayList<String> xmlData) {
try {
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(xmlData.get(0) + " " + xmlData.get(1));
} catch (ParseException e) {
e.printStackTrace();
}
return null;
}

private Date getEndDate(ArrayList<String> xmlData) {
try {
return new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ENGLISH).parse(xmlData.get(2) + " " + xmlData.get(3));
} catch (ParseException e) {
e.printStackTrace();
}
return null;
}
}
```

7.4.3 CameraVideoScheduleHandler.java

```
public class CameraVideoScheduleHandler {

private static final String CAMERAVIDEO_SCHEDULE_DIR = "/home/pi/tcc/server/cameraSchedule/videoSchedule.xml";

private static final String VIDEO = "video";
private static final String TAGNAME_STARTDATE = "startdate";
```

```
private static final String TAGNAME_STARTHOUR = "starthour";
private static final String TAGNAME_ENDDATE = "enddate";
private static final String TAGNAME_ENDHOUR = "endhour";
private static final String TAGNAME_SIZEW = "sizew";
private static final String TAGNAME_SIZEH = "sizeh";
private static final String TAGNAME_BRIGHTNESS = "brightness";
private static final String TAGNAME_CONTRAST = "contrast";
private static final String TAGNAME_SATURATION = "saturation";
private static final String TAGNAME_SHARPNESS = "sharpness";
private static final String TAGNAME_EFFECT = "effect";
private static final String TAGNAME_EXPOSURE = "exposure";

public static boolean writeXmlData(String startdate, String starthour, String enddate, String endhour, String sizeW, String sizeH,
String brightness, String contrast, String saturation, String sharpness, String effect, String exposure) {
try {
DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
Document document = documentBuilder.parse(CAMERAVIDEO_SCHEDULE_DIR);

Node nodeGpio = document.getElementsByTagName(VIDEO).item(0);
NodeList nodeList = nodeGpio.getChildNodes();

for (int i = 0; i < nodeList.getLength(); i++) {
Node node = nodeList.item(i);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_STARTDATE))
node.setTextContent(startdate);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_STARTHOUR))
node.setTextContent(starthour);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_ENDDATE))
node.setTextContent(enddate);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_ENDHOUR))
node.setTextContent(endhour);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_SIZEW))
node.setTextContent(sizeW);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_SIZEH))
node.setTextContent(sizeH);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_BRIGHTNESS))
node.setTextContent(brightness);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_CONTRAST))
node.setTextContent(contrast);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_SATURATION))
node.setTextContent(saturation);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_SHARPNESS))
node.setTextContent(sharpness);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_EFFECT))
node.setTextContent(effect);

if (node.getNodeName().equalsIgnoreCase(TAGNAME_EXPOSURE))
node.setTextContent(exposure);

}

TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource domSource = new DOMSource(document);
StreamResult streamResult = new StreamResult(new File(CAMERAVIDEO_SCHEDULE_DIR));
transformer.transform(domSource, streamResult);

return true;

} catch (ParserConfigurationException pce) {
pce.printStackTrace();
```

```
} catch (TransformerException tfe) {
tfe.printStackTrace();
} catch (IOException ioe) {
ioe.printStackTrace();
} catch (SAXException sae) {
sae.printStackTrace();
}

return false;
}

/* Return: STARTDATE STARTHOUR ENDDATE ENDHOUR SIZEW SIZEH QUALITY BRIGHTNESS CONTRAST SATURATION SHARPNESS EFFECT EXPOSURE */
public static ArrayList<String> getXmlData() {

ArrayList<String> xmlData = new ArrayList<String>();

try {
File xmlFile = new File(CAMERAVIDEO_SCHEDULE_DIR);

DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
Document document = documentBuilder.parse(xmlFile);
document.getDocumentElement().normalize();

NodeList nodeList = document.getElementsByTagName(VIDEO);

for (int i = 0; i < nodeList.getLength(); i++) {
Node node = nodeList.item(i);

if (node.getNodeType() == Node.ELEMENT_NODE) {
Element element = (Element) node;
xmlData.add(element.getElementsByTagName(TAGNAME_STARTDATE).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_STARTHOUR).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_ENDDATE).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_ENDHOUR).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_SIZEW).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_SIZEH).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_BRIGHTNESS).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_CONTRAST).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_SATURATION).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_SHARPNESS).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_EFFECT).item(0).getTextContent());
xmlData.add(element.getElementsByTagName(TAGNAME_EXPOSURE).item(0).getTextContent());
}
}
} catch (Exception e) {
e.printStackTrace();
}

return xmlData;
}
}
```

7.5 Arquivos

7.5.1 FilePicturesTransferThread.java

```
public class FilePicturesTransferThread implements Runnable {

private static String filename;

public static void start(String name) {
filename = name;
(new Thread(new FilePicturesTransferThread())).start();
}

@SuppressWarnings("resource")
public void run() {
try {
```

```
ServerSocket serverSocket;
serverSocket = new ServerSocket(5556);
Socket socket = serverSocket.accept();
File transferFile = new File("/home/pi/tcc/server/pictures/" + filename);
byte[] bytearray = new byte[8096];
FileInputStream fin = new FileInputStream(transferFile);
BufferedInputStream bin = new BufferedInputStream(fin);
OutputStream os = socket.getOutputStream();
int count;
while ((count = bin.read(bytearray, 0, 8096)) >= 0) {
os.write(bytearray, 0, count);
os.flush();
}
os.flush();
socket.close();
System.out.println("File transfer complete");
serverSocket.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

7.5.2 FileVideosTransferThread.java

```
public class FileVideosTransferThread implements Runnable {

private static String filename;

public static void start(String name) {
filename = name;
(new Thread(new FileVideosTransferThread())).start();
}

@SuppressWarnings("resource")
public void run() {
try {
ServerSocket serverSocket;
serverSocket = new ServerSocket(5556);
Socket socket = serverSocket.accept();
File transferFile = new File("/home/pi/tcc/server/videos/" + filename);
byte[] bytearray = new byte[8096];
FileInputStream fin = new FileInputStream(transferFile);
BufferedInputStream bin = new BufferedInputStream(fin);
OutputStream os = socket.getOutputStream();
int count;
while ((count = bin.read(bytearray, 0, 8096)) >= 0) {
os.write(bytearray, 0, count);
os.flush();
}
os.flush();
socket.close();
System.out.println("File transfer complete");
serverSocket.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

8 Apêndice C - Código do Cliente

8.1 Cliente

8.1.1 LoginActivity.Java

```
public class LoginActivity extends Activity {

static String ip;
static String port;

SharedPreferences prefs;

EditText editTextIp;
EditText editTextPort;

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_login);

prefs = this.getSharedPreferences("com.tcc.clienttest", Context.MODE_PRIVATE);
ip = prefs.getString("IP", "");
port = prefs.getString("PORT", "");

editTextIp = (EditText) findViewById(R.id.EditTextIp);
editTextPort = (EditText) findViewById(R.id.editTextPort);
editTextIp.setText(ip);
editTextPort.setText(port);

}

public void OnButtonClickConnect(View view) {
ip = editTextIp.getText().toString();
port = editTextPort.getText().toString();

if (isValidIp(ip)) {

prefs.edit().putString("IP", ip).commit();
prefs.edit().putString("PORT", port).commit();

// connect to the server
new ConnectionAsyncTask().execute(ip, port);

Intent intent = new Intent(LoginActivity.this, MainActivity.class);
startActivity(intent);
} else {
editTextIp.requestFocus();
editTextIp.setError("Not well formed");
}
}

public static boolean isValidIp(final String ip) {
Pattern pattern = Pattern.compile("^([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\." + "([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\."
+ "([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\." + "([01]?\\d\\d?|2[0-4]\\d|25[0-5])$");
```

```
Matcher matcher = pattern.matcher(ip);
return matcher.matches();
}
}
```

8.1.2 MainActivity.Java

```
public class MainActivity extends Activity {

private ConnectionUtilities client;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

client = ConnectionAsyncTask.getClient();

Button buttonGpio = (Button) findViewById(R.id.buttonGpio);
buttonGpio.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
Intent intent = new Intent(MainActivity.this, GpioListActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}
});

Button buttonCamera = (Button) findViewById(R.id.buttonCamera);
buttonCamera.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
Intent intent = new Intent(MainActivity.this, CameraListActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}
});

Button buttonFiles = (Button) findViewById(R.id.buttonFiles);
buttonFiles.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
Intent intent = new Intent(MainActivity.this, FileListActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}
});

// ProgressDialog 'Connecting'
new ConnectionDialog().execute("");
}

// Watch the state of ConnectionTask.
// Display the progress dialog while the status is STATE_CONNECTING
// Dismiss the dialog when the status is STATE_CONNECTED or STATE_CONNECTIONFAIL
protected class ConnectionDialog extends AsyncTask<String, String, ConnectionUtilities> {

ProgressDialog progressDialog;

@Override
protected void onPreExecute() {
super.onPreExecute();
progressDialog = new ProgressDialog(MainActivity.this);
progressDialog.setMessage("Connecting...");
progressDialog.show();
}

@Override
protected void onPostExecute(ConnectionUtilities result) {
super.onPostExecute(result);
progressDialog.dismiss();
```

```
}

@Override
protected ConnectionUtilities doInBackground(String... message) {
while (client.getConnectionState().equalsIgnoreCase(ConnectionUtilities.STATE_CONNECTING)) {
}
return null;
}
}

}
```

8.1.3 ConnectionUtilities.Java

```
public class ConnectionUtilities {

public static final String STATE_CONNECTING = "connecting";
public static final String STATE_CONNECTED = "connected";
public static final String STATE_CONNECTIONFAIL = "connection_fail";
public static final String STATE_FINISHED = "finished";

public static final int MSG_WHAT_PWM = 0;

public static final int MSG_WHAT_GPIO = 1;
public static final int MSG_WHAT_GPIO_PROGRAMMATION = 2;
public static final int MSG_WHAT_GPIO_PROGRAMMATIONREQUEST = 3;

public static final int MSG_WHAT_CAMERA_VIDEO_REALTIME = 4;
public static final int MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS = 5;
public static final int MSG_WHAT_CAMERA_VIDEO_PROGRAMMATION = 6;
public static final int MSG_WHAT_CAMERA_VIDEO_PROGRAMMATIONREQUEST = 7;

public static final int MSG_WHAT_CAMERA_PICTURE_TAKEPICTURE = 8;
public static final int MSG_WHAT_CAMERA_PICTURE_PROGRAMMATION = 9;
public static final int MSG_WHAT_CAMERA_PICTURE_PROGRAMMATIONREQUEST = 10;

public static final int MSG_WHAT_FILES_PICTURES_FILELIST = 11;
public static final int MSG_WHAT_FILES_PICTURES_DOWNLOAD = 12;

public static final int MSG_WHAT_FILES_VIDEOS_FILELIST = 13;
public static final int MSG_WHAT_FILES_VIDEOS_DOWNLOAD = 14;
public static final int MSG_WHAT_FILES_VIDEOS_CONVERT = 15;

private PrintWriter out;
private BufferedReader in;
private String state;
private String serverMessage;
private String serverIp;
private String serverPort;

public void startConnection() {

try {
setConnectionState(STATE_CONNECTING);
Log.d("ConnectionUtilities", "Connecting...");

//create a socket to make the connection with the server
Socket socket = new Socket(InetAddress.getByName(serverIp), Integer.parseInt(serverPort));

try {
//send the message to the server
out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())), true);

//receive the message which the server sends back
in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

setConnectionState(STATE_CONNECTED);
Log.d("ConnectionUtilities", "Connected.");

//in this while the client listens for the messages sent by the server
while (getConnectionState().equals(STATE_CONNECTED)) {
```

```
serverMessage = in.readLine();

if (serverMessage != null) {
Log.d("ConnectionUtilities", "Mensagem recebida: " + serverMessage);
handleReceivedMessage(serverMessage);
}
serverMessage = null;
}
} finally {
//the socket must be closed. It is not possible to reconnect to this socket
// after it is closed, which means a new socket instance has to be created.
socket.close();
}

} catch (Exception exception) {
setConnectionState(STATE_CONNECTIONFAIL);
Log.e("ConnectionUtilities", exception.getMessage());
}

}

public void endConnection() {
setConnectionState(STATE_FINISHED);
}

public void setConnectionState(String state) {
this.state = state;
}

public String getConnectionState() {
return state;
}

public void setIp(String ip) {
this.serverIp = ip;
}

public String getIp() {
return this.serverIp;
}

public void setPort(String port) {
this.serverPort = port;
}

public String getPort() {
return this.serverPort;
}

public void sendMessage(String message) {
if (out != null && !out.checkError()) {
out.println(message);
out.flush();

Log.d("ConnectionUtilities", "Mensagem enviada: " + message);
}
}

public String getMessage() {
return serverMessage;
}

private void handleReceivedMessage(String serverMessage) {
String[] inputMessage = serverMessage.replace("\n", "").replace("\r", "").split(" ");

int what = Integer.valueOf(inputMessage[0].replace(" ", ""));

switch (what) {

case MSG_WHAT_GPIO_PROGRAMMATIONREQUEST:
GpioActivity.scheduleResponse(serverMessage);
break;
```

```
case MSG_WHAT_CAMERA_VIDEO_PROGRAMMATIONREQUEST:
CameraVideoScheduleActivity.scheduleResponse(serverMessage);
break;

case MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS:
CameraRealTimeActivity.streamStatusResponse(serverMessage);
break;

case MSG_WHAT_CAMERA_PICTURE_PROGRAMMATIONREQUEST:
CameraPictureScheduleActivity.scheduleResponse(serverMessage);
break;

case MSG_WHAT_FILES_PICTURES_FILELIST:
FilePicturesActivity.fileList(serverMessage);
break;

case MSG_WHAT_FILES_PICTURES_DOWNLOAD:
FilePicturesTransferThread.start(serverIp, serverPort, inputMessage[1]);
break;

case MSG_WHAT_FILES_VIDEOS_FILELIST:
FileVideosActivity.fileList(serverMessage);
break;

case MSG_WHAT_FILES_VIDEOS_DOWNLOAD:
FileVideosTransferThread.start(serverIp, serverPort, inputMessage[1]);
break;

case MSG_WHAT_FILES_VIDEOS_CONVERT:
FileVideosActivity.fileConversionSucess(serverMessage);
break;

default:
break;
}

}
}
```

8.1.4 ConnectionAsyncTask.Java

```
public class ConnectionAsyncTask extends AsyncTask<String, String, ConnectionUtilities> {

private static ConnectionUtilities client;

@Override
protected ConnectionUtilities doInBackground(String... message) {

client = new ConnectionUtilities();
client.setIp(message[0].toString());
client.setPort(message[1].toString());
client.startConnection();
return null;
}

public static ConnectionUtilities getClient() {
return client;
}
}
```

8.2 Câmera

8.2.1 CameraListActivity.Java

```
public class CameraListActivity extends Activity {

Button buttonCameraRealTime;
```

```
Button buttonCameraVideoSchedule;
Button buttonCameraPictureSchedule;
Button buttonCameraPictureTake;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_camera_list);

buttonCameraRealTime = (Button) findViewById(R.id.buttonCameraRealTime);
buttonCameraVideoSchedule = (Button) findViewById(R.id.buttonCameraVideoSchedule);
buttonCameraPictureSchedule = (Button) findViewById(R.id.buttonCameraPictureSchedule);
buttonCameraPictureTake = (Button) findViewById(R.id.buttonCameraPictureTake);
}

public void onClickButtonCameraRealTime(View view) {
Intent intent = new Intent(CameraListActivity.this, CameraRealTimeActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonCameraVideoSchedule(View view) {
Intent intent = new Intent(CameraListActivity.this, CameraVideoScheduleActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonCameraPictureSchedule(View view) {
Intent intent = new Intent(CameraListActivity.this, CameraPictureScheduleActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonCameraPictureTake(View view) {
Intent intent = new Intent(CameraListActivity.this, CameraPictureTakeActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}
}
```

8.2.2 CameraPictureScheduleActivity.Java

```
public class CameraPictureScheduleActivity extends FragmentActivity {

private ConnectionUtilities client;

static TextView textViewStartDate;
static TextView textViewStartHour;
static TextView textViewEndDate;
static TextView textViewEndHour;
static TextView textViewInterval;

static SeekBar seekBarQuality;
static SeekBar seekBarBrightness;
static SeekBar seekBarContrast;
static SeekBar seekBarSaturation;
static SeekBar seekBarSharpness;

static TextView textViewQuality;
static TextView textViewBrightness;
static TextView textViewContrast;
static TextView textViewSaturation;
static TextView textViewSharpness;

static Spinner spinnerSize;
```

```
static Spinner spinnerEffect;
static Spinner spinnerExposure;

private int quality = 100;
private int brightness = 50;
private int contrast = 0;
private int saturation = 0;
private int sharpness = 0;

private List<String> sizes;
private List<String> effects;
private List<String> exposures;

private static boolean flagWaitingSchedule;
private static String serverMessage;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_camera_picture_schedule);

initTextView();
initSeekBar();
initSpinner();

flagWaitingSchedule = true;
client = ConnectionAsyncTask.getClient();
new AsyncTaskSchedule().execute();
}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}

public void onClickDefine(View v) {
if (textViewInterval.getText().toString().equalsIgnoreCase("")) {
showIntervalPickerDialog(v);
}

if (textViewEndHour.getText().toString().equalsIgnoreCase("")) {
showEndTimePickerDialog(v);
}

if (textViewEndDate.getText().toString().equalsIgnoreCase("")) {
showEndDatePickerDialog(v);
}

if (textViewStartHour.getText().toString().equalsIgnoreCase("")) {
showStartTimePickerDialog(v);
}

if (textViewStartDate.getText().toString().equalsIgnoreCase("")) {
showStartDatePickerDialog(v);
}

// WHAT STARTDATE STARTHOUR ENDDATE ENDHOUR INTERVAL SIZEW SIZEH QUALITY BRIGHTNESS CONTRAST SATURATION SHARPNESS EFFECT EXPOSURE
if (!textViewInterval.getText().toString().equalsIgnoreCase("") && !textViewEndHour.getText().toString().equalsIgnoreCase("")
&& !textViewEndDate.getText().toString().equalsIgnoreCase("") && !textViewStartHour.getText().toString().equalsIgnoreCase("")
&& !textViewStartDate.getText().toString().equalsIgnoreCase("")) {

String sizeW = sizes.get(spinnerSize.getSelectedItemPosition()).split(" x ")[0];
String sizeH = sizes.get(spinnerSize.getSelectedItemPosition()).split(" x ")[1];
String effect = effects.get(spinnerEffect.getSelectedItemPosition());
String exposure = exposures.get(spinnerExposure.getSelectedItemPosition());

client.sendMessage(ConnectionUtilities.MSG_WHAT_CAMERA_PICTURE_PROGRAMMATION + " " + textViewStartDate.getText().toString() + " "
+ textViewStartHour.getText().toString() + " " + textViewEndDate.getText().toString() + " "
+ textViewEndHour.getText().toString() + " " + textViewInterval.getText().toString().replace(" min", "") + " " + sizeW + " "
+ sizeH + " " + quality + " " + brightness + " " + contrast + " " + saturation + " " + sharpness + " " + effect + " " + exposure);

AlertDialog.Builder builder = new AlertDialog.Builder(CameraPictureScheduleActivity.this);
```

```
builder.setMessage("Schedule defined");
builder.setNegativeButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
}
});
builder.create().show();
}
}

// AsyncTask
private class AsyncTaskSchedule extends AsyncTask<Void, Void, Void> {

@Override
protected void onPreExecute() {
super.onPreExecute();
if (client != null)
client.sendMessage(ConnectionUtilities.MSG_WHAT_CAMERA_PICTURE_PROGRAMMATIONREQUEST + "");
}

@Override
protected Void doInBackground(Void... arg0) {
if (client != null) {
while (flagWaitingSchedule) {
}
}
return null;
}

@Override
protected void onPostExecute(Void result) {
if (serverMessage.split(" ").length >= 5) {
textViewStartDate.setText(serverMessage.split(" ")[1]);
textViewStartHour.setText(serverMessage.split(" ")[2]);
textViewEndDate.setText(serverMessage.split(" ")[3]);
textViewEndHour.setText(serverMessage.split(" ")[4]);
textViewInterval.setText(serverMessage.split(" ")[5] + " min");
}
}

}

public static void scheduleResponse(String serverMessage) {
CameraPictureScheduleActivity.serverMessage = serverMessage;
flagWaitingSchedule = false;
}

private void initTextView() {
textViewStartDate = (TextView) findViewById(R.id.textViewStartDate);
textViewStartHour = (TextView) findViewById(R.id.textViewStartHour);
textViewEndDate = (TextView) findViewById(R.id.textViewEndDate);
textViewEndHour = (TextView) findViewById(R.id.textViewEndHour);
textViewInterval = (TextView) findViewById(R.id.textViewInterval);

textViewQuality = (TextView) findViewById(R.id.textViewQuality);
textViewBrightness = (TextView) findViewById(R.id.textViewBrightness);
textViewContrast = (TextView) findViewById(R.id.textViewContrast);
textViewSaturation = (TextView) findViewById(R.id.textViewSaturation);
textViewSharpness = (TextView) findViewById(R.id.textViewSharpness);

textViewQuality.setText("QUALITY (" + quality + "):");
textViewBrightness.setText("BRIGHTNESS (" + brightness + "):");
textViewContrast.setText("CONTRAST (" + contrast + "):");
textViewSaturation.setText("SATURATION (" + saturation + "):");
textViewSharpness.setText("SHARPNESS (" + sharpness + "):");
}

private void initSeekBar() {
seekBarQuality = (SeekBar) findViewById(R.id.seekBarQuality);
seekBarBrightness = (SeekBar) findViewById(R.id.seekBarBrightness);
seekBarContrast = (SeekBar) findViewById(R.id.seekBarContrast);
seekBarSaturation = (SeekBar) findViewById(R.id.seekBarSaturation);
seekBarSharpness = (SeekBar) findViewById(R.id.seekBarSharpness);
```

```
seekBarQuality.setProgress(100);
seekBarBrightness.setProgress(50);
seekBarContrast.setProgress(50);
seekBarSaturation.setProgress(50);
seekBarSharpness.setProgress(50);

seekBarQuality.setOnSeekBarChangeListener(OnSeekBarChangeListenerQuality);
seekBarBrightness.setOnSeekBarChangeListener(OnSeekBarChangeListenerBrightness);
seekBarContrast.setOnSeekBarChangeListener(OnSeekBarChangeListenerContrast);
seekBarSaturation.setOnSeekBarChangeListener(OnSeekBarChangeListenerSaturation);
seekBarSharpness.setOnSeekBarChangeListener(onSeekBarChangeListenerSharpness);
}

private void initSpinner() {
spinnerSize = (Spinner) findViewById(R.id.spinnerSize);
sizes = new ArrayList<String>();
sizes.add("720 x 480");
sizes.add("800 x 600");
sizes.add("1366 x 768");
sizes.add("1280 x 720");
sizes.add("1920 x 1080");
ArrayAdapter<String> dataAdapterSize = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, sizes);
dataAdapterSize.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerSize.setAdapter(dataAdapterSize);

spinnerEffect = (Spinner) findViewById(R.id.spinnerEffect);
effects = new ArrayList<String>();
effects.add("none");
effects.add("negative");
effects.add("sketch");
effects.add("emboss");
effects.add("oilpaint");
effects.add("pastel");
effects.add("watercolour");
effects.add("film");
effects.add("colourswap");
effects.add("cartoon");
ArrayAdapter<String> dataAdapterEffect = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, effects);
dataAdapterEffect.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerEffect.setAdapter(dataAdapterEffect);

spinnerExposure = (Spinner) findViewById(R.id.spinnerExposure);
exposures = new ArrayList<String>();
exposures.add("auto");
exposures.add("off");
exposures.add("night");
exposures.add("sports");
exposures.add("snow");
exposures.add("beach");
exposures.add("fireworks");
exposures.add("backlight");
ArrayAdapter<String> dataAdapterExposure = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, exposures);
dataAdapterExposure.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerExposure.setAdapter(dataAdapterExposure);

}

// SEEK BAR
OnSeekBarChangeListener OnSeekBarChangeListenerQuality = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
quality = progress;
textViewQuality.setText("QUALITY (" + quality + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener OnSeekBarChangeListenerBrightness = new OnSeekBarChangeListener() {
```

```
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
brightness = progress;
textViewBrightness.setText("BRIGHTNESS (" + brightness + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener OnSeekBarChangeListenerContrast = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
contrast = (progress * 2 - 100);
textViewContrast.setText("CONTRAST (" + contrast + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener OnSeekBarChangeListenerSaturation = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
saturation = (progress * 2 - 100);
textViewSaturation.setText("SATURATION (" + saturation + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener onSeekBarChangeListenerSharpness = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
sharpness = (progress * 2 - 100);
textViewSharpness.setText("SHARPNESS (" + sharpness + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

// PICKERS
public void showIntervalPickerDialog(View v) {
DialogFragment newFragment = new IntevalPickerFragment();
newFragment.show(getSupportFragmentManager(), "timePicker");
}

public void showStartDatePickerDialog(View v) {
DialogFragment newFragment = new StartDatePickerFragment();
newFragment.show(getSupportFragmentManager(), "datePicker");
}

public void showStartTimePickerDialog(View v) {
DialogFragment newFragment = new StartTimePickerFragment();
newFragment.show(getSupportFragmentManager(), "timePicker");
}

public void showEndDatePickerDialog(View v) {
DialogFragment newFragment = new EndDatePickerFragment();
```

```
newFragment.show(getSupportFragmentManager(), "datePicker");
}

public void showEndTimePickerDialog(View v) {
DialogFragment newFragment = new EndTimePickerFragment();
newFragment.show(getSupportFragmentManager(), "timePicker");
}

// DATE PICKER START
public static class StartDatePickerFragment extends DialogFragment implements DatePickerDialog.OnDateSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
int day = c.get(Calendar.DAY_OF_MONTH);

DatePickerDialog datePickerDialog = new DatePickerDialog(getActivity(), this, year, month, day);
datePickerDialog.setTitle("Start Date");
return datePickerDialog;
}

public void onDateSet(DatePicker view, int year, int month, int day) {
String monthString = String.valueOf(month + 1);
String dayString = String.valueOf(day);
if (monthString.length() == 1)
monthString = 0 + monthString;
if (dayString.length() == 1)
dayString = 0 + dayString;
textViewStartDate.setText(dayString + "/" + monthString + "/" + year);
}
}

// HOUR PICKER START
public static class StartTimePickerFragment extends DialogFragment implements TimePickerDialog.OnTimeSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int hour = c.get(Calendar.HOUR_OF_DAY);
int minute = c.get(Calendar.MINUTE);

TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(), this, hour, minute + 1, DateFormat.is24HourFormat(getActivity()));
timePickerDialog.setTitle("Start Hour");
return timePickerDialog;
}

public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
String hourString = String.valueOf(hourOfDay);
String minuteString = String.valueOf(minute);
if (hourString.length() == 1)
hourString = 0 + hourString;
if (minuteString.length() == 1)
minuteString = 0 + minuteString;
textViewStartHour.setText(hourString + ":" + minuteString);
}
}

// INTERVAL PICKER
public static class IntevalPickerFragment extends DialogFragment implements TimePickerDialog.OnTimeSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(), this, 0, 1, DateFormat.is24HourFormat(getActivity()));
timePickerDialog.setTitle("Interval between pictures");
return timePickerDialog;
}

public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
int totalInterval = hourOfDay * 60 + minute;
textViewInterval.setText(totalInterval + " min");
}
```

```
}

// DATE PICKER END
public static class EndDatePickerFragment extends DialogFragment implements DatePickerDialog.OnDateSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
int day = c.get(Calendar.DAY_OF_MONTH);

DatePickerDialog datePickerDialog = new DatePickerDialog(getActivity(), this, year, month, day);
datePickerDialog.setTitle("End Date");
return datePickerDialog;
}

public void onDateSet(DatePicker view, int year, int month, int day) {
String monthString = String.valueOf(month + 1);
String dayString = String.valueOf(day);
if (monthString.length() == 1)
monthString = 0 + monthString;
if (dayString.length() == 1)
dayString = 0 + dayString;
textViewEndDate.setText(dayString + "/" + monthString + "/" + year);
}
}

// HOUR PICKER END
public static class EndTimePickerFragment extends DialogFragment implements TimePickerDialog.OnTimeSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int hour = c.get(Calendar.HOUR_OF_DAY);
int minute = c.get(Calendar.MINUTE);

TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(), this, hour, minute + 2, DateFormat.is24HourFormat(getActivity()));
timePickerDialog.setTitle("End Hour");
return timePickerDialog;
}

public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
String hourString = String.valueOf(hourOfDay);
String minuteString = String.valueOf(minute);
if (hourString.length() == 1)
hourString = 0 + hourString;
if (minuteString.length() == 1)
minuteString = 0 + minuteString;
textViewEndHour.setText(hourString + ":" + minuteString);
}
}
}
```

8.2.3 CameraPictureTakeActivity.Java

```
public class CameraPictureTakeActivity extends Activity {

private ConnectionUtilities client;

static SeekBar seekBarQuality;
static SeekBar seekBarBrightness;
static SeekBar seekBarContrast;
static SeekBar seekBarSaturation;
static SeekBar seekBarSharpness;

static TextView textViewQuality;
static TextView textViewBrightness;
static TextView textViewContrast;
static TextView textViewSaturation;
static TextView textViewSharpness;
```

```
static Spinner spinnerSize;
static Spinner spinnerEffect;
static Spinner spinnerExposure;

private int quality = 100;
private int brightness = 50;
private int contrast = 0;
private int saturation = 0;
private int sharpness = 0;

private List<String> sizes;
private List<String> effects;
private List<String> exposures;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_camera_picture_take);

initTextView();
initSeekBar();
initSpinner();

client = ConnectionAsyncTask.getClient();
}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}

public void onClickTakePicture(View v) {
String sizeW = sizes.get(spinnerSize.getSelectedItemPosition()).split(" x ")[0];
String sizeH = sizes.get(spinnerSize.getSelectedItemPosition()).split(" x ")[1];
String effect = effects.get(spinnerEffect.getSelectedItemPosition());
String exposure = exposures.get(spinnerExposure.getSelectedItemPosition());

client.sendMessage(ConnectionUtilities.MSG_WHAT_CAMERA_PICTURE_TAKEPICTURE + " " + sizeW + " " + sizeH + " " + quality + " " + brightness
+ " " + contrast + " " + saturation + " " + sharpness + " " + effect + " " + exposure);

new ConnectionDialog().execute("");
}

protected class ConnectionDialog extends AsyncTask<String, String, ConnectionUtilities> {

ProgressDialog progressDialog;

@Override
protected void onPreExecute() {
super.onPreExecute();
progressDialog = new ProgressDialog(CameraPictureTakeActivity.this);
progressDialog.setMessage("Taking picture...");
progressDialog.setCancelable(false);
progressDialog.show();
}

@Override
protected ConnectionUtilities doInBackground(String... message) {
try {
Thread.sleep(2000);
} catch (InterruptedException e) {
e.printStackTrace();
}
return null;
}

@Override
protected void onPostExecute(ConnectionUtilities result) {
super.onPostExecute(result);
progressDialog.dismiss();
```

```
AlertDialog.Builder builder = new AlertDialog.Builder(CameraPictureTakeActivity.this);
builder.setMessage("Picture taken successfully");
builder.setNegativeButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
}
});
builder.create().show();
}
}

private void initTextView() {
textViewQuality = (TextView) findViewById(R.id.textViewQuality);
textViewBrightness = (TextView) findViewById(R.id.textViewBrightness);
textViewContrast = (TextView) findViewById(R.id.textViewContrast);
textViewSaturation = (TextView) findViewById(R.id.textViewSaturation);
textViewSharpness = (TextView) findViewById(R.id.textViewSharpness);

textViewQuality.setText("QUALITY (" + quality + "):");
textViewBrightness.setText("BRIGHTNESS (" + brightness + "):");
textViewContrast.setText("CONTRAST (" + contrast + "):");
textViewSaturation.setText("SATURATION (" + saturation + "):");
textViewSharpness.setText("SHARPNESS (" + sharpness + "):");
}

private void initSeekBar() {
seekBarQuality = (SeekBar) findViewById(R.id.seekBarQuality);
seekBarBrightness = (SeekBar) findViewById(R.id.seekBarBrightness);
seekBarContrast = (SeekBar) findViewById(R.id.seekBarContrast);
seekBarSaturation = (SeekBar) findViewById(R.id.seekBarSaturation);
seekBarSharpness = (SeekBar) findViewById(R.id.seekBarSharpness);

seekBarQuality.setProgress(100);
seekBarBrightness.setProgress(50);
seekBarContrast.setProgress(50);
seekBarSaturation.setProgress(50);
seekBarSharpness.setProgress(50);

seekBarQuality.setOnSeekBarChangeListener(OnSeekBarChangeListenerQuality);
seekBarBrightness.setOnSeekBarChangeListener(OnSeekBarChangeListenerBrightness);
seekBarContrast.setOnSeekBarChangeListener(OnSeekBarChangeListenerContrast);
seekBarSaturation.setOnSeekBarChangeListener(OnSeekBarChangeListenerSaturation);
seekBarSharpness.setOnSeekBarChangeListener(onSeekBarChangeListenerSharpness);
}

private void initSpinner() {
spinnerSize = (Spinner) findViewById(R.id.spinnerSize);
sizes = new ArrayList<String>();
sizes.add("720 x 480");
sizes.add("800 x 600");
sizes.add("1366 x 768");
sizes.add("1280 x 720");
sizes.add("1920 x 1080");
ArrayAdapter<String> dataAdapterSize = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, sizes);
dataAdapterSize.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerSize.setAdapter(dataAdapterSize);

spinnerEffect = (Spinner) findViewById(R.id.spinnerEffect);
effects = new ArrayList<String>();
effects.add("none");
effects.add("negative");
effects.add("sketch");
effects.add("emboss");
effects.add("oilpaint");
effects.add("pastel");
effects.add("watercolour");
effects.add("film");
effects.add("colourswap");
effects.add("cartoon");
ArrayAdapter<String> dataAdapterEffect = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, effects);
dataAdapterEffect.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerEffect.setAdapter(dataAdapterEffect);

spinnerExposure = (Spinner) findViewById(R.id.spinnerExposure);
```

```
exposures = new ArrayList<String>();
exposures.add("auto");
exposures.add("off");
exposures.add("night");
exposures.add("sports");
exposures.add("snow");
exposures.add("beach");
exposures.add("fireworks");
exposures.add("backlight");
ArrayAdapter<String> dataAdapterExposure = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, exposures);
dataAdapterExposure.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerExposure.setAdapter(dataAdapterExposure);

}

// SEEK BAR
OnSeekBarChangeListener OnSeekBarChangeListenerQuality = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
quality = progress;
textViewQuality.setText("QUALITY (" + quality + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener OnSeekBarChangeListenerBrightness = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
brightness = progress;
textViewBrightness.setText("BRIGHTNESS (" + brightness + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener OnSeekBarChangeListenerContrast = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
contrast = (progress * 2 - 100);
textViewContrast.setText("CONTRAST (" + contrast + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener OnSeekBarChangeListenerSaturation = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
saturation = (progress * 2 - 100);
textViewSaturation.setText("SATURATION (" + saturation + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener onSeekBarChangeListenerSharpness = new OnSeekBarChangeListener() {
```

```
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
sharpness = (progress * 2 - 100);
textViewSharpness.setText("SHARPNESS (" + sharpness + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};
}
```

8.2.4 CameraVideoScheduleActivity.Java

```
public class CameraVideoScheduleActivity extends FragmentActivity {

private ConnectionUtilities client;

static TextView textViewStartDate;
static TextView textViewStartHour;
static TextView textViewEndDate;
static TextView textViewEndHour;

static SeekBar seekBarBrightness;
static SeekBar seekBarContrast;
static SeekBar seekBarSaturation;
static SeekBar seekBarSharpness;

static TextView textViewBrightness;
static TextView textViewContrast;
static TextView textViewSaturation;
static TextView textViewSharpness;

static Spinner spinnerSize;
static Spinner spinnerEffect;
static Spinner spinnerExposure;

private int brightness = 50;
private int contrast = 0;
private int saturation = 0;
private int sharpness = 0;

private List<String> sizes;
private List<String> effects;
private List<String> exposures;

private static boolean flagWaitingSchedule;
private static String serverMessage;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_camera_video_schedule);

initTextView();
initSeekBar();
initSpinner();

flagWaitingSchedule = true;
client = ConnectionAsyncTask.getClient();
new AsyncTaskSchedule().execute();
}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}
```

```
public void onClickDefine(View v) {
if (textViewEndHour.getText().toString().equalsIgnoreCase("")) {
showEndTimePickerDialog(v);
}

if (textViewEndDate.getText().toString().equalsIgnoreCase("")) {
showEndDatePickerDialog(v);
}

if (textViewStartHour.getText().toString().equalsIgnoreCase("")) {
showStartTimePickerDialog(v);
}

if (textViewStartDate.getText().toString().equalsIgnoreCase("")) {
showStartDatePickerDialog(v);
}

// WHAT STARTDATE STARTHOUR ENDDATE ENDHOUR SIZEW SIZEH BRIGHTNESS CONTRAST SATURATION SHARPNESS EFFECT EXPOSURE
if (!textViewEndHour.getText().toString().equalsIgnoreCase("") && !textViewEndDate.getText().toString().equalsIgnoreCase("")
&& !textViewStartHour.getText().toString().equalsIgnoreCase("") && !textViewStartDate.getText().toString().equalsIgnoreCase("")) {

String sizeW = sizes.get(spinnerSize.getSelectedItemPosition()).split(" x ")[0];
String sizeH = sizes.get(spinnerSize.getSelectedItemPosition()).split(" x ")[1];
String effect = effects.get(spinnerEffect.getSelectedItemPosition());
String exposure = exposures.get(spinnerExposure.getSelectedItemPosition());

client.sendMessage(ConnectionUtilities.MSG_WHAT_CAMERA_VIDEO_PROGRAMMATION + " " + textViewStartDate.getText().toString() + " "
+ textViewStartHour.getText().toString() + " " + textViewEndDate.getText().toString() + " "
+ textViewEndHour.getText().toString() + " " + sizeW + " " + sizeH + " " + brightness + " " + contrast + " " + saturation + " "
+ sharpness + " " + effect + " " + exposure);

AlertDialog.Builder builder = new AlertDialog.Builder(CameraVideoScheduleActivity.this);
builder.setMessage("Schedule defined");
builder.setNegativeButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
}
});
builder.create().show();
}
}

// AsyncTask
private class AsyncTaskSchedule extends AsyncTask<Void, Void, Void> {

@Override
protected void onPreExecute() {
super.onPreExecute();
if (client != null)
client.sendMessage(ConnectionUtilities.MSG_WHAT_CAMERA_VIDEO_PROGRAMMATIONREQUEST + "");
}

@Override
protected Void doInBackground(Void... arg0) {
if (client != null) {
while (flagWaitingSchedule) {
}
}
return null;
}

@Override
protected void onPostExecute(Void result) {
if (serverMessage.split(" ").length >= 4) {
textViewStartDate.setText(serverMessage.split(" ")[1]);
textViewStartHour.setText(serverMessage.split(" ")[2]);
textViewEndDate.setText(serverMessage.split(" ")[3]);
textViewEndHour.setText(serverMessage.split(" ")[4]);
}
}
}

public static void scheduleResponse(String serverMessage) {
CameraVideoScheduleActivity.serverMessage = serverMessage;
```

```
flagWaitingSchedule = false;
}

private void initTextView() {
textViewStartDate = (TextView) findViewById(R.id.textViewStartDate);
textViewStartHour = (TextView) findViewById(R.id.textViewStartHour);
textViewEndDate = (TextView) findViewById(R.id.textViewEndDate);
textViewEndHour = (TextView) findViewById(R.id.textViewEndHour);

textViewBrightness = (TextView) findViewById(R.id.textViewBrightness);
textViewContrast = (TextView) findViewById(R.id.textViewContrast);
textViewSaturation = (TextView) findViewById(R.id.textViewSaturation);
textViewSharpness = (TextView) findViewById(R.id.textViewSharpness);

textViewBrightness.setText("BRIGHTNESS (" + brightness + "):");
textViewContrast.setText("CONTRAST (" + contrast + "):");
textViewSaturation.setText("SATURATION (" + saturation + "):");
textViewSharpness.setText("SHARPNESS (" + sharpness + "):");
}

private void initSeekBar() {
seekBarBrightness = (SeekBar) findViewById(R.id.seekBarBrightness);
seekBarContrast = (SeekBar) findViewById(R.id.seekBarContrast);
seekBarSaturation = (SeekBar) findViewById(R.id.seekBarSaturation);
seekBarSharpness = (SeekBar) findViewById(R.id.seekBarSharpness);

seekBarBrightness.setProgress(50);
seekBarContrast.setProgress(50);
seekBarSaturation.setProgress(50);
seekBarSharpness.setProgress(50);

seekBarBrightness.setOnSeekBarChangeListener(OnSeekBarChangeListenerBrightness);
seekBarContrast.setOnSeekBarChangeListener(OnSeekBarChangeListenerContrast);
seekBarSaturation.setOnSeekBarChangeListener(OnSeekBarChangeListenerSaturation);
seekBarSharpness.setOnSeekBarChangeListener(onSeekBarChangeListenerSharpness);
}

private void initSpinner() {
spinnerSize = (Spinner) findViewById(R.id.spinnerSize);
sizes = new ArrayList<String>();
sizes.add("100 x 100");
sizes.add("720 x 480");
sizes.add("800 x 600");
sizes.add("1366 x 768");
sizes.add("1280 x 720");
sizes.add("1920 x 1080");
ArrayAdapter<String> dataAdapterSize = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, sizes);
dataAdapterSize.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerSize.setAdapter(dataAdapterSize);

spinnerEffect = (Spinner) findViewById(R.id.spinnerEffect);
effects = new ArrayList<String>();
effects.add("none");
effects.add("negative");
effects.add("sketch");
effects.add("emboss");
effects.add("oilpaint");
effects.add("pastel");
effects.add("watercolour");
effects.add("film");
effects.add("colourswap");
effects.add("cartoon");
ArrayAdapter<String> dataAdapterEffect = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, effects);
dataAdapterEffect.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerEffect.setAdapter(dataAdapterEffect);

spinnerExposure = (Spinner) findViewById(R.id.spinnerExposure);
exposures = new ArrayList<String>();
exposures.add("auto");
exposures.add("off");
exposures.add("night");
exposures.add("sports");
exposures.add("snow");
```

```
exposures.add("beach");
exposures.add("fireworks");
exposures.add("backlight");
ArrayAdapter<String> dataAdapterExposure = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, exposures);
dataAdapterExposure.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinnerExposure.setAdapter(dataAdapterExposure);

}

// SEEK BAR
OnSeekBarChangeListener OnSeekBarChangeListenerBrightness = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
brightness = progress;
textViewBrightness.setText("BRIGHTNESS (" + brightness + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener OnSeekBarChangeListenerContrast = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
contrast = (progress * 2 - 100);
textViewContrast.setText("CONTRAST (" + contrast + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener OnSeekBarChangeListenerSaturation = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
saturation = (progress * 2 - 100);
textViewSaturation.setText("SATURATION (" + saturation + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

OnSeekBarChangeListener onSeekBarChangeListenerSharpness = new OnSeekBarChangeListener() {

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
sharpness = (progress * 2 - 100);
textViewSharpness.setText("SHARPNESS (" + sharpness + "):");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

public void onStopTrackingTouch(SeekBar seekBar) {
}
};

// PICKERS
public void showStartDatePickerDialog(View v) {
DialogFragment newFragment = new StartDatePickerFragment();
newFragment.show(getSupportFragmentManager(), "datePicker");
}

public void showStartTimePickerDialog(View v) {
```

```
DialogFragment newFragment = new StartTimePickerFragment();
newFragment.show(getSupportFragmentManager(), "timePicker");
}

public void showEndDatePickerDialog(View v) {
DialogFragment newFragment = new EndDatePickerFragment();
newFragment.show(getSupportFragmentManager(), "datePicker");
}

public void showEndTimePickerDialog(View v) {
DialogFragment newFragment = new EndTimePickerFragment();
newFragment.show(getSupportFragmentManager(), "timePicker");
}

// DATE PICKER START
public static class StartDatePickerFragment extends DialogFragment implements DatePickerDialog.OnDateSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
int day = c.get(Calendar.DAY_OF_MONTH);

DatePickerDialog datePickerDialog = new DatePickerDialog(getActivity(), this, year, month, day);
datePickerDialog.setTitle("Start Date");
return datePickerDialog;
}

public void onDateSet(DatePicker view, int year, int month, int day) {
String monthString = String.valueOf(month + 1);
String dayString = String.valueOf(day);
if (monthString.length() == 1)
monthString = 0 + monthString;
if (dayString.length() == 1)
dayString = 0 + dayString;
textViewStartDate.setText(dayString + "/" + monthString + "/" + year);
}
}

// HOUR PICKER START
public static class StartTimePickerFragment extends DialogFragment implements TimePickerDialog.OnTimeSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int hour = c.get(Calendar.HOUR_OF_DAY);
int minute = c.get(Calendar.MINUTE);

TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(), this, hour, minute + 1, DateFormat.is24HourFormat(getActivity()));
timePickerDialog.setTitle("Start Hour");
return timePickerDialog;
}

public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
String hourString = String.valueOf(hourOfDay);
String minuteString = String.valueOf(minute);
if (hourString.length() == 1)
hourString = 0 + hourString;
if (minuteString.length() == 1)
minuteString = 0 + minuteString;
textViewStartHour.setText(hourString + ":" + minuteString);
}
}

// DATE PICKER END
public static class EndDatePickerFragment extends DialogFragment implements DatePickerDialog.OnDateSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
```

```
int day = c.get(Calendar.DAY_OF_MONTH);

DatePickerDialog datePickerDialog = new DatePickerDialog(getActivity(), this, year, month, day);
datePickerDialog.setTitle("End Date");
return datePickerDialog;
}

public void onDateSet(DatePicker view, int year, int month, int day) {
String monthString = String.valueOf(month + 1);
String dayString = String.valueOf(day);
if (monthString.length() == 1)
monthString = 0 + monthString;
if (dayString.length() == 1)
dayString = 0 + dayString;
textViewEndDate.setText(dayString + "/" + monthString + "/" + year);
}
}

// HOUR PICKER END
public static class EndTimePickerFragment extends DialogFragment implements TimePickerDialog.OnTimeSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int hour = c.get(Calendar.HOUR_OF_DAY);
int minute = c.get(Calendar.MINUTE);

TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(), this, hour, minute + 2, DateFormat.is24HourFormat(getActivity()));
timePickerDialog.setTitle("End Hour");
return timePickerDialog;
}

public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
String hourString = String.valueOf(hourOfDay);
String minuteString = String.valueOf(minute);
if (hourString.length() == 1)
hourString = 0 + hourString;
if (minuteString.length() == 1)
minuteString = 0 + minuteString;
textViewEndHour.setText(hourString + ":" + minuteString);
}
}
}
```

8.2.5 CameraRealTimeActivity.Java

```
public class CameraRealTimeActivity extends Activity {

private ConnectionUtilities client;

boolean toggleButtonOn = false;

private static String STREAM_LINK;
private static String VLC_LINK = "http://www.videolan.org/vlc/";

EditText edittextLink;
ToggleButton toggleButtonRealTime;

private static boolean flagWaitingStatus;
private static String serverMessage;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_camera_video_realtime);

client = ConnectionAsyncTask.getClient();

STREAM_LINK = "rtsp://" + LoginActivity.ip + ":8554/pi_encode.h264";

edittextLink = (EditText) findViewById(R.id.edittextLink);
```

```
edittextLink.setInputType(InputType.TYPE_NULL);
edittextLink.setEnabled(false);
edittextLink.setText("OFFLINE");

toggleButtonRealTime = (ToggleButton) findViewById(R.id.toggleButtonRealTime);

flagWaitingStatus = true;
client = ConnectionAsyncTask.getClient();
new AsyncTaskStreamStatus().execute();
}

// AsyncTask
private class AsyncTaskStreamStatus extends AsyncTask<Void, Void, Void> {

@Override
protected void onPreExecute() {
super.onPreExecute();
if (client != null)
client.sendMessage(ConnectionUtilities.MSG_WHAT_CAMERA_VIDEO_REALTIMESTATUS + "");
}

@Override
protected Void doInBackground(Void... arg0) {
if (client != null) {
while (flagWaitingStatus) {
}
}
return null;
}

@Override
protected void onPostExecute(Void result) {
if (serverMessage.split(" ")[1].equalsIgnoreCase("1")) {
toggleButtonRealTime.setChecked(true);
edittextLink.setEnabled(true);
edittextLink.setText(STREAM_LINK);
} else {
toggleButtonRealTime.setChecked(false);
edittextLink.setEnabled(false);
edittextLink.setText("OFFLINE");
}
}
}

public static void streamStatusResponse(String serverMessage) {
CameraRealTimeActivity.serverMessage = serverMessage;
flagWaitingStatus = false;
}

public void toggleButtonRealTime(View view) {
toggleButtonOn = ((ToggleButton) view).isChecked();
if (toggleButtonOn) {
edittextLink.setEnabled(true);
edittextLink.setText(STREAM_LINK);
client.sendMessage(ConnectionUtilities.MSG_WHAT_CAMERA_VIDEO_REALTIME + " " + "1");
} else {
edittextLink.setEnabled(false);
edittextLink.setText("OFFLINE");
client.sendMessage(ConnectionUtilities.MSG_WHAT_CAMERA_VIDEO_REALTIME + " " + "0");
}
}

public void onClickSendEmail(View view) {
if (toggleButtonOn) {
Intent i = new Intent(Intent.ACTION_SEND);
i.setType("message/rfc822");
i.putExtra(Intent.EXTRA_EMAIL, new String[] { "" });
i.putExtra(Intent.EXTRA_SUBJECT, "Raspberry Pi Stream");
i.putExtra(Intent.EXTRA_TEXT,
"Link to stream: " + STREAM_LINK + System.getProperty("line.separator") + System.getProperty("line.separator")
+ "Link to VLC Media Player: " + VLC_LINK);
try {
startActivity(Intent.createChooser(i, "Send mail..."));
```

```
} catch (android.content.ActivityNotFoundException ex) {
Toast.makeText(CameraRealTimeActivity.this, "There are no email clients installed.", Toast.LENGTH_SHORT).show();
}
} else {
Toast.makeText(CameraRealTimeActivity.this, "Stream offline.", Toast.LENGTH_SHORT).show();
}

}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}
}
```

8.3 GPIO

8.3.1 GpioListActivity.Java

```
public class GpioListActivity extends Activity {

Button buttonGpio0;
Button buttonGpio1;
Button buttonGpio2;
Button buttonGpio3;
Button buttonGpio4;
Button buttonGpio5;
Button buttonGpio6;
Button buttonGpio7;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_gpio_list);

buttonGpio0 = (Button) findViewById(R.id.buttonGpio0);
buttonGpio1 = (Button) findViewById(R.id.buttonGpio1);
buttonGpio2 = (Button) findViewById(R.id.buttonGpio2);
buttonGpio3 = (Button) findViewById(R.id.buttonGpio3);
buttonGpio4 = (Button) findViewById(R.id.buttonGpio4);
buttonGpio5 = (Button) findViewById(R.id.buttonGpio5);
buttonGpio6 = (Button) findViewById(R.id.buttonGpio6);
buttonGpio7 = (Button) findViewById(R.id.buttonGpio7);

}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}

public void onClickButtonGpio0(View view) {
Intent intent = new Intent(GpioListActivity.this, GpioActivity.class);
intent.putExtra("CHANNELNUMBER", "0");
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonGpio1(View view) {
Intent intent = new Intent(GpioListActivity.this, GpioActivity.class);
intent.putExtra("CHANNELNUMBER", "1");
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonGpio2(View view) {
Intent intent = new Intent(GpioListActivity.this, GpioActivity.class);
intent.putExtra("CHANNELNUMBER", "2");
```

```
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonGpio3(View view) {
Intent intent = new Intent(GpioListActivity.this, GpioActivity.class);
intent.putExtra("CHANNELNUMBER", "3");
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonGpio4(View view) {
Intent intent = new Intent(GpioListActivity.this, GpioActivity.class);
intent.putExtra("CHANNELNUMBER", "4");
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonGpio5(View view) {
Intent intent = new Intent(GpioListActivity.this, GpioActivity.class);
intent.putExtra("CHANNELNUMBER", "5");
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonGpio6(View view) {
Intent intent = new Intent(GpioListActivity.this, GpioActivity.class);
intent.putExtra("CHANNELNUMBER", "6");
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonGpio7(View view) {
Intent intent = new Intent(GpioListActivity.this, GpioActivity.class);
intent.putExtra("CHANNELNUMBER", "7");
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

}
```

8.3.2 GpioActivity.Java

```
public class GpioActivity extends FragmentActivity {

static ToggleButton toggleButtonGpio;
static TextView textViewGpioNumber;
static TextView textViewStartDate;
static TextView textViewStartHour;
static TextView textViewEndDate;
static TextView textViewEndHour;
private ConnectionUtilities client;

private static String gpioNumber;

private static boolean flagWaitingSchedule;
private static String serverMessage;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_gpio_schedule);

Intent intent = getIntent();
gpioNumber = intent.getStringExtra("CHANNELNUMBER");

toggleButtonGpio = (ToggleButton) findViewById(R.id.toggleButtonGpio);
textViewGpioNumber = (TextView) findViewById(R.id.textViewGpioNumber);
textViewGpioNumber.setText("GPIO #" + gpioNumber);
textViewStartDate = (TextView) findViewById(R.id.textViewStartDate);
textViewStartHour = (TextView) findViewById(R.id.textViewStartHour);
```

```
textViewEndDate = (TextView) findViewById(R.id.textViewEndDate);
textViewEndHour = (TextView) findViewById(R.id.textViewEndHour);

flagWaitingSchedule = true;
client = ConnectionAsyncTask.getClient();
new AsyncTaskSchedule().execute();

}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}

public void onClickDefine(View v) {
if (textViewEndHour.getText().toString().equalsIgnoreCase("")) {
showEndTimePickerDialog(v);
}

if (textViewEndDate.getText().toString().equalsIgnoreCase("")) {
showEndDatePickerDialog(v);
}

if (textViewStartHour.getText().toString().equalsIgnoreCase("")) {
showStartTimePickerDialog(v);
}

if (textViewStartDate.getText().toString().equalsIgnoreCase("")) {
showStartDatePickerDialog(v);
}

if (!textViewEndHour.getText().toString().equalsIgnoreCase("") && !textViewEndDate.getText().toString().equalsIgnoreCase("")
&& !textViewStartHour.getText().toString().equalsIgnoreCase("") && !textViewStartDate.getText().toString().equalsIgnoreCase("")) {

client.sendMessage(ConnectionUtilities.MSG_WHAT_GPIO_PROGRAMMATION + " " + gpioNumber + " " + "0" + " "
+ textViewStartDate.getText().toString() + " " + textViewStartHour.getText().toString() + " "
+ textViewEndDate.getText().toString() + " " + textViewEndHour.getText().toString());
}
}

public void onToggleClicked(View view) {
boolean on = ((ToggleButton) view).isChecked();
if (on) {
client.sendMessage(ConnectionUtilities.MSG_WHAT_GPIO + " " + gpioNumber + " " + "1");
} else {
client.sendMessage(ConnectionUtilities.MSG_WHAT_GPIO + " " + gpioNumber + " " + "0");
}
}

public void showStartDatePickerDialog(View v) {
DialogFragment newFragment = new StartDatePickerFragment();
newFragment.show(getSupportFragmentManager(), "datePicker");
}

public void showStartTimePickerDialog(View v) {
DialogFragment newFragment = new StartTimePickerFragment();
newFragment.show(getSupportFragmentManager(), "timePicker");
}

public void showEndDatePickerDialog(View v) {
DialogFragment newFragment = new EndDatePickerFragment();
newFragment.show(getSupportFragmentManager(), "datePicker");
}

public void showEndTimePickerDialog(View v) {
DialogFragment newFragment = new EndTimePickerFragment();
newFragment.show(getSupportFragmentManager(), "timePicker");
}

// AsyncTask
private class AsyncTaskSchedule extends AsyncTask<Void, Void, Void> {
```

```
@Override
protected void onPreExecute() {
super.onPreExecute();
if (client != null)
client.sendMessage(ConnectionUtilities.MSG_WHAT_GPIO_PROGRAMMATIONREQUEST + " " + gpioNumber);
}

@Override
protected Void doInBackground(Void... arg0) {
if (client != null) {
while (flagWaitingSchedule) {
}
}
return null;
}

@Override
protected void onPostExecute(Void result) {
if (serverMessage.split(" ")[1].equalsIgnoreCase("1.0"))
toggleButtonGpio.setChecked(true);
else if (serverMessage.split(" ")[1].equalsIgnoreCase("0.0"))
toggleButtonGpio.setChecked(false);

textViewStartDate.setText(serverMessage.split(" ")[2]);
textViewStartHour.setText(serverMessage.split(" ")[3]);
textViewEndDate.setText(serverMessage.split(" ")[4]);
textViewEndHour.setText(serverMessage.split(" ")[5]);
}

}

public static void scheduleResponse(String serverMessage) {
GpioActivity.serverMessage = serverMessage;
flagWaitingSchedule = false;
}

// PICKERS
// DATE PICKER START
public static class StartDatePickerFragment extends DialogFragment implements DatePickerDialog.OnDateSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
int day = c.get(Calendar.DAY_OF_MONTH);

DatePickerDialog datePickerDialog = new DatePickerDialog(getActivity(), this, year, month, day);
datePickerDialog.setTitle("Start Date");
return datePickerDialog;
}

public void onDateSet(DatePicker view, int year, int month, int day) {
String monthString = String.valueOf(month + 1);
String dayString = String.valueOf(day);
if (monthString.length() == 1)
monthString = 0 + monthString;
if (dayString.length() == 1)
dayString = 0 + dayString;
textViewStartDate.setText(dayString + "/" + monthString + "/" + year);
}
}

// HOUR PICKER START
public static class StartTimePickerFragment extends DialogFragment implements TimePickerDialog.OnTimeSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int hour = c.get(Calendar.HOUR_OF_DAY);
int minute = c.get(Calendar.MINUTE);

TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(), this, hour, minute, DateFormat.is24HourFormat(getActivity()));
```

```
timePickerDialog.setTitle("Start Hour");
return timePickerDialog;
}

public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
String hourString = String.valueOf(hourOfDay);
String minuteString = String.valueOf(minute);
if (hourString.length() == 1)
hourString = 0 + hourString;
if (minuteString.length() == 1)
minuteString = 0 + minuteString;
textViewStartHour.setText(hourString + ":" + minuteString);
}
}

// DATE PICKER END
public static class EndDatePickerFragment extends DialogFragment implements DatePickerDialog.OnDateSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
int day = c.get(Calendar.DAY_OF_MONTH);

DatePickerDialog datePickerDialog = new DatePickerDialog(getActivity(), this, year, month, day);
datePickerDialog.setTitle("End Date");
return datePickerDialog;
}

public void onDateSet(DatePicker view, int year, int month, int day) {
String monthString = String.valueOf(month + 1);
String dayString = String.valueOf(day);
if (monthString.length() == 1)
monthString = 0 + monthString;
if (dayString.length() == 1)
dayString = 0 + dayString;
textViewEndDate.setText(dayString + "/" + monthString + "/" + year);
}
}

// HOUR PICKER END
public static class EndTimePickerFragment extends DialogFragment implements TimePickerDialog.OnTimeSetListener {

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
final Calendar c = Calendar.getInstance();
int hour = c.get(Calendar.HOUR_OF_DAY);
int minute = c.get(Calendar.MINUTE);

TimePickerDialog timePickerDialog = new TimePickerDialog(getActivity(), this, hour, minute, DateFormat.is24HourFormat(getActivity()));
timePickerDialog.setTitle("End Hour");
return timePickerDialog;
}

public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
String hourString = String.valueOf(hourOfDay);
String minuteString = String.valueOf(minute);
if (hourString.length() == 1)
hourString = 0 + hourString;
if (minuteString.length() == 1)
minuteString = 0 + minuteString;
textViewEndHour.setText(hourString + ":" + minuteString);
}
}

}
```

8.4 Arquivos

8.4.1 FileListActivity.Java

```
public class FileListActivity extends Activity {

Button buttonPictures;
Button buttonVideos;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_file_list);

buttonPictures = (Button) findViewById(R.id.buttonPictures);
buttonVideos = (Button) findViewById(R.id.buttonVideos);
}

public void onClickButtonPictures(View v) {
Intent intent = new Intent(FileListActivity.this, FilePicturesActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

public void onClickButtonVideos(View v) {
Intent intent = new Intent(FileListActivity.this, FileVideosActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}
}
```

8.4.2 FilePicturesActivity.Java

```
@SuppressLint({ "SimpleDateFormat" })
public class FilePicturesActivity extends Activity {

private ConnectionUtilities client;

ListView listViewPictureFiles;
static ArrayList<String> arrayAdapter;

ProgressDialog progressDialog;
static ListViewFilePicturesAdapter listViewFilePicturesAdapter;
private static String serverMessage = "";

private static boolean flagWaitingFileList;
private static boolean flagWaitingFileTransfer = true;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_file_pictures);

arrayAdapter = new ArrayList<String>();
listViewFilePicturesAdapter = new ListViewFilePicturesAdapter(this, arrayAdapter);
creatListView();

flagWaitingFileList = true;
client = ConnectionAsyncTask.getClient();
new AsyncTaskFillFileList().execute();

}

@Override
```

```
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}

private void creatListView() {
listViewPictureFiles = (ListView) findViewById(R.id.listViewPictureFiles);
listViewPictureFiles.setAdapter(listViewFilePicturesAdapter);
listViewPictureFiles.setOnItemClickListener(new OnItemClickListener() {

@Override
public void onItemClick(final AdapterView<?> arg0, final View arg1, final int arg2, final long arg3) {

final String fileName = arrayAdapter.get(arg2).split("@")[0];

AlertDialog.Builder builder = new AlertDialog.Builder(FilePicturesActivity.this);
builder.setTitle("Pictures");
builder.setMessage(arrayAdapter.get(arg2).split("@")[0]);
builder.setPositiveButton("Download file", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
downloadFile(fileName);
}

});
builder.setNeutralButton("View picture", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
viewImage(fileName);
}

});
builder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
}
});
builder.create().show();

}
});
}

private void downloadFile(String fileName) {
new AsyncTaskFileTransferProgress().execute();
client.sendMessage(ConnectionUtilities.MSG_WHAT_FILES_PICTURES_DOWNLOAD + " " + fileName);
}

private void viewImage(String fileName) {
if (!(new File(Environment.getExternalStorageDirectory() + "/TCC/pictures/" + fileName).isFile())) {
Toast.makeText(FilePicturesActivity.this, "Download the file first", Toast.LENGTH_SHORT).show();
} else {
String imagePath = Environment.getExternalStorageDirectory() + "/TCC/pictures/" + fileName;

try {
Intent myIntent = new Intent(android.content.Intent.ACTION_VIEW);
File file = new File(imagePath);
String extension = android.webkit.MimeTypeMap.getFileExtensionFromUrl(Uri.fromFile(file).toString());
String mimetype = android.webkit.MimeTypeMap.getSingleton().getMimeTypeFromExtension(extension);
myIntent.setDataAndType(Uri.fromFile(file), mimetype);
startActivity(myIntent);
} catch (Exception e) {
}

}
}

// ASYNCTASK FILE TRANSFER PROGRESS
protected class AsyncTaskFileTransferProgress extends AsyncTask<Void, Void, ConnectionUtilities> {
ProgressDialog progressDialog;

@Override
protected void onPreExecute() {
super.onPreExecute();
progressDialog = new ProgressDialog(FilePicturesActivity.this);
progressDialog.setMessage("Downloading...");
```

```
progressDialog.show();
}

@Override
protected ConnectionUtilities doInBackground(Void... message) {
while (flagWaitingFileTransfer) {
}
flagWaitingFileTransfer = true;
return null;
}

@Override
protected void onPostExecute(ConnectionUtilities result) {
super.onPostExecute(result);

progressDialog.dismiss();

AlertDialog.Builder builder = new AlertDialog.Builder(FilePicturesActivity.this);
builder.setMessage("File transfer successfully");
builder.setNegativeButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
}
});
builder.create().show();
}
}

public static void FileTransferSucess() {
flagWaitingFileTransfer = false;
}

// ASYNCTASK FILL FILE LIST
protected class AsyncTaskFillFileList extends AsyncTask<Void, Void, ConnectionUtilities> {

@Override
protected void onPreExecute() {
super.onPreExecute();
if (client != null) {
client.sendMessage(ConnectionUtilities.MSG_WHAT_FILES_PICTURES_FILELIST + "");

progressDialog = new ProgressDialog(FilePicturesActivity.this);
progressDialog.setMessage("Loading file list...");
progressDialog.show();
}
}

@Override
protected ConnectionUtilities doInBackground(Void... message) {
if (client != null) {
while (flagWaitingFileList) {
}
int numPictures = Integer.parseInt(serverMessage.split(" ")[1]);
arrayAdapter.clear();
for (int i = 2; i < numPictures + 2; i++) {
arrayAdapter.add(serverMessage.split(" ")[i]);
}
}
return null;
}

@Override
protected void onPostExecute(ConnectionUtilities result) {
super.onPostExecute(result);
listViewFilePicturesAdapter.notifyDataSetChanged();
progressDialog.dismiss();
}
}

public static void fileList(String serverMessage) {
FilePicturesActivity.serverMessage = serverMessage;
flagWaitingFileList = false;
}
```

```
// LISTVIEW ADAPTER
public class ListViewFilePicturesAdapter extends BaseAdapter {
Context context;
ArrayList<String> item;
LayoutInflater inflater = null;

public ListViewFilePicturesAdapter(Context context, ArrayList<String> data) {
this.context = context;
this.item = data;
inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
}

@Override
public int getCount() {
return item.size();
}

@Override
public Object getItem(int position) {
return item.get(position);
}

@Override
public long getItemId(int position) {
return position;
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
View vi = convertView;
if (vi == null)
vi = inflater.inflate(R.layout.row_files, null);

TextView fileName = (TextView) vi.findViewById(R.id.textViewFileName);
TextView fileDate = (TextView) vi.findViewById(R.id.textViewFileDate);
TextView fileSize = (TextView) vi.findViewById(R.id.textViewFileSize);

fileName.setText(item.get(position).split("@")[0]);
fileDate.setText(new SimpleDateFormat("dd/MM/yyy HH:mm").format(new Date(Long.parseLong(item.get(position).split("@")[1]))));
fileSize.setText(humanReadableByteCount(Long.parseLong(item.get(position).split("@")[2]), true));

fileName.setTextColor(getResources().getColor(R.color.red_button));
fileDate.setTextColor(getResources().getColor(R.color.red_button));
fileSize.setTextColor(getResources().getColor(R.color.red_button));

return vi;
}

public String humanReadableByteCount(long bytes, boolean si) {
int unit = si ? 1000 : 1024;
if (bytes < unit)
return bytes + " B";
int exp = (int) (Math.log(bytes) / Math.log(unit));
String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp - 1) + (si ? "" : "i");
return String.format(Locale.ENGLISH, "%.1f %sB", bytes / Math.pow(unit, exp), pre);
}
}
}
```

8.4.3 FileVideosActivity.Java

```
@SuppressLint("SimpleDateFormat")
public class FileVideosActivity extends Activity {

private ConnectionUtilities client;

ListView listViewvideoFiles;
ArrayList<String> arrayAdapter;

ProgressDialog progressDialog;
ListViewFileVideosAdapter listViewFileVideosAdapter;
```

```
private static String serverMessage = "";

private static boolean flagWaitingFileList = true;
private static boolean flagWaitingFileTransfer = true;
private static boolean flagWaitingFileConversion = true;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_file_videos);

arrayAdapter = new ArrayList<String>();
listViewFileVideosAdapter = new ListViewFileVideosAdapter(this, arrayAdapter);
creatListView();

new FillFileList().execute();

client = ConnectionAsyncTask.getClient();
client.sendMessage(ConnectionUtilities.MSG_WHAT_FILES_VIDEOS_FILELIST + "");
}

@Override
public void onBackPressed() {
super.onBackPressed();
overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
}

private void creatListView() {
listViewvideoFiles = (ListView) findViewById(R.id.listViewVideoFiles);
listViewvideoFiles.setAdapter(listViewFileVideosAdapter);
listViewvideoFiles.setOnItemClickListener(new OnItemClickListener() {

@Override
public void onItemClick(final AdapterView<?> arg0, final View arg1, final int arg2, final long arg3) {

final String fileName = arrayAdapter.get(arg2).split("@")[0];

AlertDialog.Builder builder = new AlertDialog.Builder(FileVideosActivity.this);
builder.setTitle("Videos");
builder.setMessage(arrayAdapter.get(arg2).split("@")[0]);
builder.setPositiveButton("Download file", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
downloadFile(fileName);
}

});

if (arrayAdapter.get(arg2).split("@")[0].split("\\.")[1].equalsIgnoreCase("h264"))
builder.setNeutralButton("Convert video", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
convertVideo(fileName);
}

});
else
builder.setNeutralButton("View video", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
viewVideo(fileName);
}

});

builder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
}
});
builder.create().show();

}
});
}

private void downloadFile(String fileName) {
```

```
new FileTransferProgress().execute();
client.sendMessage(ConnectionUtilities.MSG_WHAT_FILES_VIDEOS_DOWNLOAD + " " + fileName);
}

private void convertVideo(String fileName) {
new FileConvertProgress().execute();
client.sendMessage(ConnectionUtilities.MSG_WHAT_FILES_VIDEOS_CONVERT + " " + fileName);
}

private void viewVideo(String fileName) {
if (!(new File(Environment.getExternalStorageDirectory() + "/TCC/videos/" + fileName).isFile())) {
Toast.makeText(FileVideosActivity.this, "Download the file first", Toast.LENGTH_SHORT).show();
} else {
String filePath = Environment.getExternalStorageDirectory() + "/TCC/videos/" + fileName;

try {
Intent myIntent = new Intent(android.content.Intent.ACTION_VIEW);
File file = new File(filePath);
String extension = android.webkit.MimeTypeMap.getFileExtensionFromUrl(Uri.fromFile(file).toString());
String mimetype = android.webkit.MimeTypeMap.getSingleton().getMimeTypeFromExtension(extension);
myIntent.setDataAndType(Uri.fromFile(file), mimetype);
startActivity(myIntent);
} catch (Exception e) {
}

}
}

// ASYNCTASK FILE CONVERT PROGRESS
protected class FileConvertProgress extends AsyncTask<Void, Void, ConnectionUtilities> {

ProgressDialog progressDialog;

@Override
protected void onPreExecute() {
super.onPreExecute();
progressDialog = new ProgressDialog(FileVideosActivity.this);
progressDialog.setMessage("Converting...");
progressDialog.show();
}

@Override
protected ConnectionUtilities doInBackground(Void... message) {
while (flagWaitingFileConversion) {
}
flagWaitingFileConversion = true;
arrayAdapter.add(serverMessage.split(" ")[1]);
return null;
}

@Override
protected void onPostExecute(ConnectionUtilities result) {
super.onPostExecute(result);

progressDialog.dismiss();

listViewFileVideosAdapter.notifyDataSetChanged();
AlertDialog.Builder builder = new AlertDialog.Builder(FileVideosActivity.this);
builder.setMessage("File conversion successfully");
builder.setNegativeButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
}
});
builder.create().show();
}
}

public static void fileConversionSucess(String serverMessage) {
FileVideosActivity.serverMessage = serverMessage;
flagWaitingFileConversion = false;
}

// ASYNCTASK FILE TRANSFER PROGRESS
```

```
protected class FileTransferProgress extends AsyncTask<Void, Void, ConnectionUtilities> {
ProgressDialog progressDialog;

@Override
protected void onPreExecute() {
super.onPreExecute();
progressDialog = new ProgressDialog(FileVideosActivity.this);
progressDialog.setMessage("Downloading...");
progressDialog.show();
}

@Override
protected ConnectionUtilities doInBackground(Void... message) {
while (flagWaitingFileTransfer) {
}
flagWaitingFileTransfer = true;
return null;
}

@Override
protected void onPostExecute(ConnectionUtilities result) {
super.onPostExecute(result);

progressDialog.dismiss();

listViewFileVideosAdapter.notifyDataSetChanged();
AlertDialog.Builder builder = new AlertDialog.Builder(FileVideosActivity.this);
builder.setMessage("File transfer successfully");
builder.setNegativeButton("OK", new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int id) {
}
});
builder.create().show();
}
}

public static void fileTransferSucess() {
flagWaitingFileTransfer = false;
}

// ASYNCTASK FILL FILE LIST
protected class FillFileList extends AsyncTask<Void, Void, ConnectionUtilities> {

@Override
protected void onPreExecute() {
super.onPreExecute();
progressDialog = new ProgressDialog(FileVideosActivity.this);
progressDialog.setMessage("Loading file list...");
progressDialog.show();
}

@Override
protected ConnectionUtilities doInBackground(Void... message) {
while (flagWaitingFileList) {
}
flagWaitingFileList = true;
int numVideos = Integer.parseInt(serverMessage.split(" ")[1]);
arrayAdapter.clear();
for (int i = 2; i < numVideos + 2; i++) {
arrayAdapter.add(serverMessage.split(" ")[i]);
}
return null;
}

@Override
protected void onPostExecute(ConnectionUtilities result) {
super.onPostExecute(result);
listViewFileVideosAdapter.notifyDataSetChanged();
progressDialog.dismiss();
}
}

public static void fileList(String serverMessage) {
```

```
FileVideosActivity.serverMessage = serverMessage;
flagWaitingFileList = false;
}

// LISTVIEW ADAPTER
public class ListViewFileVideosAdapter extends BaseAdapter {
Context context;
ArrayList<String> item;
LayoutInflater inflater = null;

public ListViewFileVideosAdapter(Context context, ArrayList<String> data) {
this.context = context;
this.item = data;
inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
}

@Override
public int getCount() {
return item.size();
}

@Override
public Object getItem(int position) {
return item.get(position);
}

@Override
public long getItemId(int position) {
return position;
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
View vi = convertView;
if (vi == null)
vi = inflater.inflate(R.layout.row_files, null);

TextView fileName = (TextView) vi.findViewById(R.id.textViewFileName);
TextView fileDate = (TextView) vi.findViewById(R.id.textViewFileDate);
TextView fileSize = (TextView) vi.findViewById(R.id.textViewFileSize);

fileName.setText(item.get(position).split("@")[0]);
fileDate.setText(new SimpleDateFormat("dd/MM/yyy HH:mm").format(new Date(Long.parseLong(item.get(position).split("@")[1]))));
fileSize.setText(humanReadableByteCount(Long.parseLong(item.get(position).split("@")[2]), true));

fileName.setTextColor(getResources().getColor(R.color.red_button));
fileDate.setTextColor(getResources().getColor(R.color.red_button));
fileSize.setTextColor(getResources().getColor(R.color.red_button));

if (item.get(position).split("@")[0].contains("h264"))
fileName.setTypeface(null, Typeface.ITALIC);
else
fileName.setTypeface(null, Typeface.BOLD);

return vi;
}

public String humanReadableByteCount(long bytes, boolean si) {
int unit = si ? 1000 : 1024;
if (bytes < unit)
return bytes + " B";
int exp = (int) (Math.log(bytes) / Math.log(unit));
String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp - 1) + (si ? "" : "i");
return String.format(Locale.ENGLISH, "%.1f %sB", bytes / Math.pow(unit, exp), pre);
}
}

}
```

8.4.4 FileVideosTransferThread.Java

```
public class FileVideosTransferThread implements Runnable {
```

```
private static String serverIp;
private static String serverPort;
private static String fileName;

public static void start(String ip, String port, String name) {
serverIp = ip;
serverPort = port;
fileName = name;
(new Thread(new FileVideosTransferThread())).start();
}

public void run() {

if (!(new File(Environment.getExternalStorageDirectory() + "/TCC/videos/").isDirectory()))
new File(Environment.getExternalStorageDirectory() + "/TCC/videos/").mkdirs();

try {
Socket socket;
socket = new Socket(InetAddress.getByName(serverIp), Integer.parseInt(serverPort) + 1);
byte[] bytearray = new byte[8096];
InputStream is = socket.getInputStream();
FileOutputStream fos = new FileOutputStream(Environment.getExternalStorageDirectory() + "/TCC/videos/" + fileName);
BufferedOutputStream bos = new BufferedOutputStream(fos);

int count;
while ((count = is.read(bytearray)) >= 0) {
bos.write(bytearray, 0, count);
}
bos.flush();
bos.close();
socket.close();

FileVideosActivity.fileTransferSucess();
} catch (UnknownHostException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
try {
Thread.sleep(2000);
(new Thread(new FileVideosTransferThread())).start();
} catch (InterruptedException e1) {
e1.printStackTrace();
}
}
}
}
```

8.4.5 FilePicturesTransferThread.Java

```
public class FilePicturesTransferThread implements Runnable {

private static String serverIp;
private static String serverPort;
private static String fileName;

public static void start(String ip, String port, String name) {
serverIp = ip;
serverPort = port;
fileName = name;
(new Thread(new FilePicturesTransferThread())).start();
}

public void run() {

if (!(new File(Environment.getExternalStorageDirectory() + "/TCC/pictures/").isDirectory()))
new File(Environment.getExternalStorageDirectory() + "/TCC/pictures/").mkdirs();

try {
Socket socket;
socket = new Socket(InetAddress.getByName(serverIp), Integer.parseInt(serverPort) + 1);
```

```
byte[] bytearray = new byte[8096];
InputStream is = socket.getInputStream();
FileOutputStream fos = new FileOutputStream(Environment.getExternalStorageDirectory() + "/TCC/pictures/" + fileName);
BufferedOutputStream bos = new BufferedOutputStream(fos);

int count;
while ((count = is.read(bytearray)) >= 0) {
bos.write(bytearray, 0, count);
}
bos.flush();
bos.close();
socket.close();

FilePicturesActivity.FileTransferSucess();
} catch (UnknownHostException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
try {
Thread.sleep(2000);
(new Thread(new FilePicturesTransferThread())).start();
} catch (InterruptedException e1) {
e1.printStackTrace();
}
}
}
}
```