UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
ENGENHARIA DE COMPUTAÇÃO

DESENVOLVIMENTO DE APLICATIVO ANDROID PARA MONITORAMENTO DE USINAS FOTOVOLTAICAS

TRABALHO DE GRADUAÇÃO

Pedro Grassi Xavier

Santa Maria, RS, Brasil

2015

DESENVOLVIMENTO DE APLICATIVO ANDROID PARA MONITORAMENTO DE USINAS FOTOVOLTAICAS

Pedro Grassi Xavier

Trabalho de Graduação apresentado ao curso de Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Engenharia de Computação

Orientador: Prof. Dr. Leandro Michels

Santa Maria, RS, Brasil

2015

Universidade Federal de Santa Maria
Centro de Tecnologia
Engenharia de Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

DESENVOLVIMENTO DE APLICATIVO ANDROID PARA MONITORAMENTO DE USINAS FOTOVOLTAICAS

elaborado por
Pedro Grassi Xavier

como requisito parcial para obtenção do grau de
Bacharel em Engenharia de Computação

COMISSÃO EXAMINADORA:

Leandro Michels, Dr.
(Presidente/Orientador)

Carlos Henrique Barriquello, Dr. (UFSM)

Mateus Beck Rutzig, Dr. (UFSM)

Santa Maria, 03 de Julho de 2015.

RESUMO

Trabalho de Graduação
Engenharia de Computação
Universidade Federal de Santa Maria

DESENVOLVIMENTO DE APLICATIVO ANDROID PARA MONITORAMENTO DE USINAS FOTOVOLTAICAS

AUTOR: PEDRO GRASSI XAVIER
ORIENTADOR: LEANDRO MICHELS
Local da Defesa e Data: Santa Maria, 03 de Julho de 2015.

Este trabalho apresenta o desenvolvimento de um aplicativo para o sistema operacional Android com a função de monitorar usinas fotovoltaicas. O monitoramento é um recurso importante para acompanhar a operação de sistemas fotovoltaicos. Sistemas de monitoramento são utilizados para se obter dados de produção, efetuar comparações, identificar tendências, detectar problemas e definir estratégias que permitam um melhor desempenho técnico e econômico. No aplicativo será possível visualizar informações de produção em tempo real, gráficos relativos a produção diária, visualizar a localização geográfica dos inversores, entre outras funções. O aplicativo será escalável, ou seja, permitirá que inversores possam ser adicionados ou removidos do sistema sem que o aplicativo precise ser alterado.

Palavras-chave: Android. Fotovoltaica. Monitoramento.

ABSTRACT

Undergraduate Final Work
Bachelor of Computation Engineering
Federal University of Santa Maria

DEVELOPMENT OF ANDROID APPLICATION FOR MONITORING OF FOTOVOLTAIC PLANTS

AUTHOR: PEDRO GRASSI XAVIER
ADVISOR: LEANDRO MICHELS
Defense Place and Date: Santa Maria, July 03th, 2015.

This paper presents the development of an application for the Android operating system with the function of monitoring photovoltaic plants. Monitoring is an important resource for monitoring the operation of photovoltaic systems. Monitoring systems are used to obtain production data, make comparisons, identify trends, detect problems and strategies that lead to improved technical and economic performance. In the application you can view production information in real time, graphs of the daily production , display the geographical location of the inverters, among other functions. The application is scalable, allow inverters to be added or removed from the system without the application needs to be changed.

Keywords: Android, Fotovoltaic, Monitoring.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Apk	*Android application package*
GPS	*Global Positioning System*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
kWh	Quilowatt-hora
MySQL	*My Structured Query Language*
PHP	*Hypertext Preprocessor*
SDK	*Software Development Kit*
SGDB	Sistema de Gerenciamento de Banco de Dados
SOAP	*Simple Object Access Protocol*
REST	*Representational State Transfer*

SUMÁRIO

1 INTRODUÇÃO

A busca por alternativas renováveis de energia vem se tornando cada vez mais importante. Tanto a nível industrial quanto residencial, sistemas de geração de energia elétrica que utilizam fontes deste tipo são tendências mundiais. Entre estas fontes, destacam-se o vento, através de sistemas de geração eólicos e o Sol, através de sistemas de geração fotovoltaicos.

Sistemas fotovoltaicos se utilizam da energia do Sol para geração de energia elétrica. Estima-se que o Sol ainda vai gerar energia por mais seis bilhões de anos, com fornecimento de energia aproximadamente quatro mil vezes maior que aquela consumida pelo planeta Terra por ano (LOBO, 2004).

Além disso, sistemas fotovoltaicos apresentam algumas vantagens pontuais em relação a outras fontes renováveis de energia: sistemas fotovoltaicos podem atuar junto ou distante do ponto de consumo, placas fotovoltaicas podem ser instaladas no solo ou em telhados, possuem elevada confiabilidade e operam de forma limpa e silenciosa.

Unidades de monitoramento em um sistema fotovoltaico são importantes para monitorar e avaliar o sistema. Elas permitem que administradores ou usuários finais do sistema possam ter acesso a informações relevantes acerca do aproveitamento do mesmo, permitindo que novas estratégias possam ser definidas na busca dos melhores resultados possíveis.

1.1 Motivação

Dentro do contexto de monitoremento de sistemas fotovoltaicos, é interessante que as informações possam ser acessadas de forma rápida e fácil a partir de qualquer local a qualquer momento. Nesse cenário, aplicativos desenvolvidos para dispositivos móveis surgem como uma opção interessante para este tipo de tarefa.

O Android é um sistema operacional de código aberto baseado em Linux lançado pela Google em Setembro de 2008. Atualmente, é o sistema operacional móvel mais popular do mundo com mais de 75% da fatia de mercado no mundo (IDC, 2014), estando presente em *smartphones, tablets, smart TVs* e até *smartwatches*.

Considerando estes pontos apresentados, o desenvolvimento de aplicativos para a plataforma Android que permitam o monitoramento de usinas fotovoltaicas mostra-se como uma importante ferramenta para estimular um cresimento ainda maior deste tipo de sistema, tanto em nível residencial quanto industrial.

1.2 Objetivos

Este trabalho tem o objetivo de desenvolver um aplicativo para o sistema Android com a função de monitorar sistemas fotovoltaicos. Serão abordados diversos assuntos estudados ao longo do curso de Engenharia de Computação, como programação orientada a objetos, conceitos de eletrônica de potência e protocolos de comunicação.

O monitoramento consiste em apresentar informações relativas a geração de energia, potência atual, localização geográfica, entre outros recursos, tudo de forma clara e objetiva. O aplicativo tem como foco inicial o monitoramento da usina fotovoltaica do Grupo de Eletrônica de Potência e Controle (GEPOC) da Universidade Federal de Santa Maria (UFSM). Porém, ele será desenvolvido de forma que seja possível sua expansão a sistemas maiores, integrando novos dispositvos e informações.

A parte do sistema de monitoramento reponsável pela comunicação entre os painéis fotovoltaicos, inversores, *data loggers* e banco de dados já foi desenvolvida pelo GEPOC. Esse sistema será brevemente apresentado ao longo deste trabalho, porém, o principal objetivo é a apresentação do aplicativo de monitoramento integrado a este sistema.

1.3 Organização do Trabalho

O presente trabalho possui a seguinte organização: no capítulo 2 são apresentados sistemas de monitoramento semelhantes ao desenvolvido, o funcionamento do protocolo SunSpec e da plataforma Android.

No capítulo 3 é apresentado o sistema de monitoramento atualmente em funcionamento, envolvendo a comunicação entre placas fotovoltaicas, inversores, *data loggers* e o banco de dados. Também é apresentada a idéia de funcionamento do aplicativo.

O capítulo 4 aborda mais profudamente o desenvolvimento do aplicativo, apresentado as ferramentas utilizadas, estrutura do aplicativo, conceitos considerados, bibliotecas adicionais utilzadas, entre outras informações relevantes.

No capítulo 5 são apresentados os resultados obtidos, mostrando o sistema implementado e os resultados obtidos a partir dele.

O capítulo 6 apresenta a conclusão do trabalho, onde são feitas a análise dos resultados obtidos e sugestões para trabalhos futuros.

2 MONITORAMENTO DE SISTEMAS FOTOVOLTAICOS

2.1 Geração de energia solar fotovoltaica

O Sol é a origem de toda as formas de energia que o homem vem utilizando durante sua história. As três principais formas de aproveitamento direto da energia solar são a solar térmica, solar térmico-elétrica e solar fotovoltaica (BLUESOL, 2010). A geração solar térmica transforma a radiação proveniente do Sol em calor. As placas, chamadas de coletores, captam o calor proveniente do Sol e aquecem, normalmente, água. Esta água aquecida é então utilizada para banhos, piscinas ou processos industriais.

As usinas térmico-elétricas utilizam o calor do Sol para gerar eletricidade convencional. Consiste em um grande arranjo de refletores direcionados para o aquecimento da água e geração de vapor, que por fim movimenta turbinas capazes de gerar eletricidade. É um processo de conversão de energia mecânica para energia elétrica, semelhante as hidréletricas convencionais.

Já os sistemas fotovoltaicos, fazem a conversão direta da radiação solar em energia elétrica. A energia fotovoltaica se baseia no princípio do efeito fotoelétrico, que consiste na utilização dos fótons, partícula contida na luz, para gerar energia elétrica. A capacidade de geração depende da incidência de luz na região, e não do calor. Esta energia está pronta para ser utilizada por qualquer equipamento eletroeletrônico. Sistemas deste tipo vem apresentando um elevado crescimento no últimos anos, conforme mostra a figura 2.1.

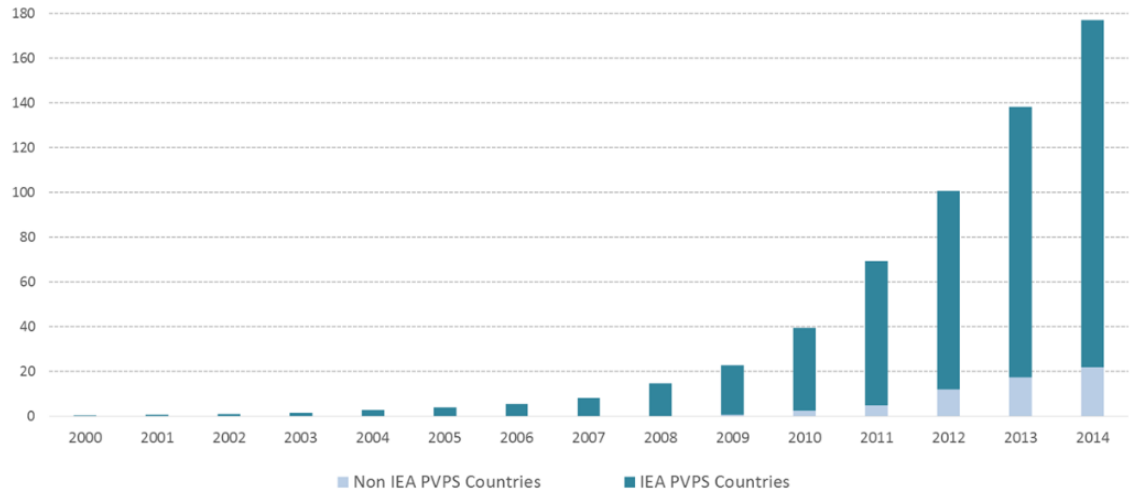

Figura 2.1 – Estimativa da potência acumulada de instalações fotovoltaicas no mundo, em gigawatts (IEA, 2014).

Os sistemas fotovoltaicos são classificados em três tipos com relação à sua conexão aos

sistemas públicos de energia: autônomos, conectados à rede e híbridos (PEREIRA; OLIVEIRA, 2011). Sistemas autônomos são aqueles que não possuem nenhuma ligação com a rede elétrica convencional, normalmente apresentam armazenamento de energia em baterias para regular a intermitência da fonte. Sistemas conectados à rede não possuem armazenamento de energia. A regulagem da intermitência da fonte é realizada pela rede elétrica. Quando a potência produzida é maior do que a consumida, o excedente é transferido à rede elétrica, mas, quando a potência consumida é superior à produzida, a rede elétrica fornece a potência necessária para a alimentação da carga. Já os sistemas híbridos são sistemas conectados à rede, mas que possuem armazenamento de energia para garantir a alimentação da carga no caso de faltas da rede elétrica. Assim, funcionam como sistemas ininterruptos de energia, mas com parte da potência gerada pelo sistema fotovoltaico. O sistema fotovoltoico no qual este trabalho foi desenvolvido é do tipo conectado à rede.

2.2 Protocolo SunSpec

A crescente popularização de sistemas fotovoltaicos despertou novas necessidades de mercado, entre elas a necessidade de uma padronização dos sistemas fotovoltaicos em todo o mundo. A *SunSpec Alliance* é uma aliança comercial internacional que tem por objetivo padronizar aspectos operacionais envolvendo sistemas fotovoltaicos, acelerar o crescimento da indústria de energia distribuída e expandir o mercado de energia renovável, principalmente a fotovoltaica (ALLIANCE, 2012). Ela é reponsável por desenvolver um padrão de modelos de informações, protocolos de comunicação e interfaces de sistema, conforme demonstado na figura 2.2.

O protocolo SunSpec é baseado no protocolo Modbus e permite não somente a leitura dos dispositivos em uso, mas também o controle destes através da rede. O padrão SunSpec também busca eliminar alguns problemas recorrentes em sistema fotovoltaicos, tais como particularidades de cada fabricante e dificuldade em substituir elementos do sistema sem alterar outros.

2.3 Monitoramento de geração fotovoltaica

O monitoramento é um recurso importante para avaliar o aproveitamento de sistemas fotovoltaicos. Diversos estudos foram realizados sobre formas de se obter e apresentar dados

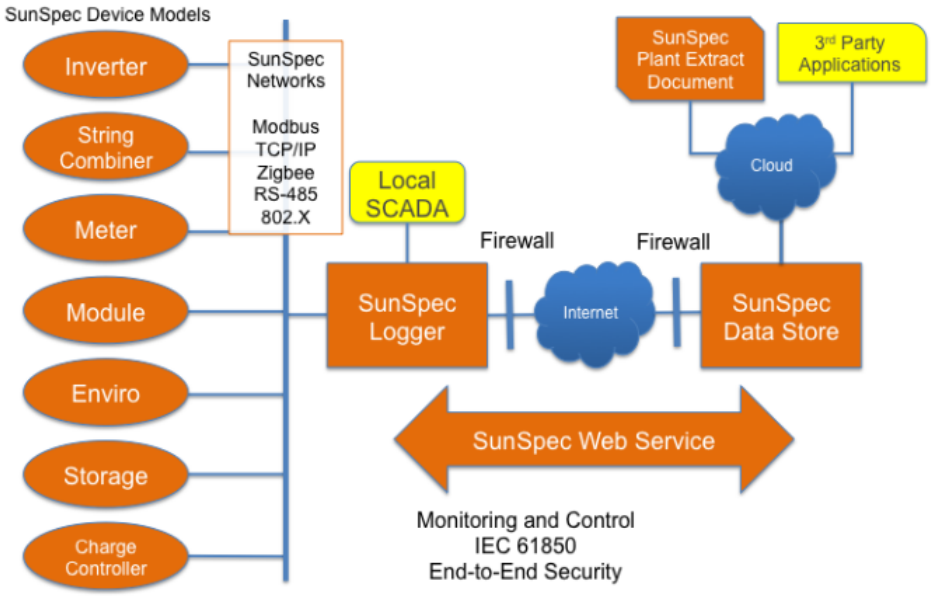

Figura 2.2 – Arquitetura Sunspec completa (ALLIANCE, 2012)

relativos a sistemas de monitoramento. No trabalho de (ANDREONI, 2012), foi implementado um sistema de monitoramento remoto de um sistema fotovoltaico através de uma rede *wireless*, utilizando o protocolo de comunicação *Zigbee*. O resultado obtido foi bastante interessante, apresentando as informações de forma precisa e detalhada. O sistema porém esbarra na limitação do acesso ao alcance da rede *wireless* local. Em (KUN et al., 2013) foi desenvolvido um aplicativo Android que através do protocolo *bluetooth* se conecta diretamente ao *data logger* local. Este tipo de sistema dispensa o uso de servidores, porém, devido ao pequeno alcance da rede *bluetooth*, seu funcionamento fica limitado ao local de instalação dos inversores, não permitindo que seus dados possam ser acessados diretamente a partir de qualquer local. Além disso, esse tipo de aplicação só pode ser implementada em inversores que possuam o recurso de conexão *bluetooth*, que representa uma parcela pouco significativa do mercado.

Os autores em (LIN et al., 2014), também propuseram uma alternativa para o monitoramento e controle de sistemas fotovoltaicos baseada em dispositivos móveis. Tal implementação utiliza o protocolo IPv6 e permite comunicação rápida com as informações fornecidas pelos inversores, porém o sistema apresenta poucas informações e recursos. Já (PEREIRA et al., 2014) apresentou uma arquitetura de um aplicativo Android capaz de exibir informações relativas a um sistema fotovoltaico, independente de seu tamanho. O trabalho apresenta diversos conceitos interessantes, como a exibição dos *status* atual de cada equipamento e sua produção diária. Ele se utiliza de serviços intermediários que facilitam o acesso do aplicativo as informações vindas dos inversores.

Todos os trabalhos acima possuem os mesmos objetivos que são o monitoramento de sistemas e a apresentação dos dados aos usuários. O trabalho aqui apresentado também compartilha do mesmo objetivo. Este também permite que tais informações possam ser consultadas de forma rápida através de um *smartphone* conectado a internet, utilizando uma arquitetura semelhante a apresentada em (PEREIRA et al., 2014), porém com diferentes recursos e funcionalidades. No aplicativo proposto, o usuário tem acesso a informações sobre a geração, consumo de energia elétrica, localização dos inversores fotovoltaicos, entre outras. A escalabilidade também é um fator importante a ser destacado, pois permite que novos inversores possam ser adicionados ou removidos do sistema automaticamente, sem a necessidade de reconfiguração do aplicativo.

2.4 Plataforma Android

Um dos grandes responsáveis pela popularização dos *smartphones*, o sistema operacional Android é um sistema operacional de código aberto lançado pelo Google em Setembro de 2008. Baseado em Linux, o sistema apresenta uma enorme compatibilidade de hardware, sendo possível executá-lo em dispostivos das mais variadas marcas e modelos.

Conforme demonstrado na figura 2.3, o Android é o sistema operacional móvel mais utilizado do mundo (IDC, 2014). Dispositivos com o sistema Android vendem mais que eletrônicos com Windows, iOS e Mac OS X combinados (YAROW, 2014). Em julho de 2013, a loja de aplicativos Google Play possuía mais de 1 milhão de aplicativos disponíveis, baixados mais de 50 bilhões de vezes (PHONEARENA, 2013). Na conferência anual Google I/O de 2014, a companhia revelou que existem mais de 1 bilhão de usuários Android ativos. Em junho de 2013, este número era de 538 milhões.

Além disso, o Android é muito popular entre empresas de tecnologia que buscam um software pronto, de baixo custo e personalizável para dispositivos de alta tecnologia. A natureza do software de código aberto do sistema operacional tem encorajado uma grande comunidade de programadores e entusiastas no desenvolvimento de projetos que adicionam recursos para usuários mais avançados, ou trazem o Android para dispositivos que inicialmente não foram lançados com a plataforma.

Os aplicativos para o sistema Android são desenvolvidos principalmente na linguagem Java, através do sistema de desenvolvimento do software Android (SDK). O SDK inclui um conjunto de ferramentas de desenvolvimento, que incluem um depurador, bibliotecas, um emu-

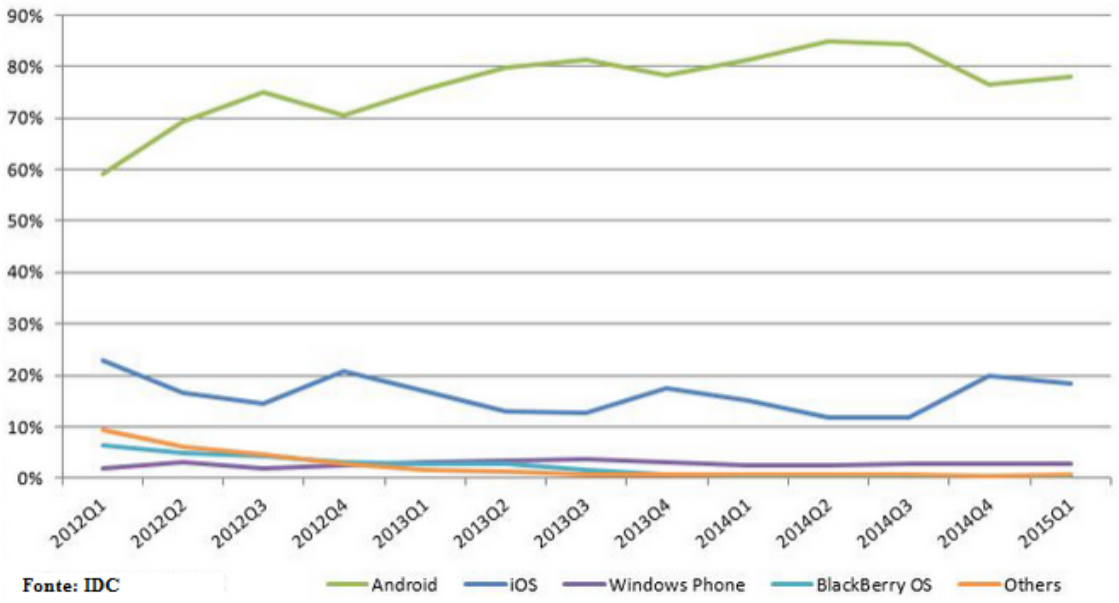

Figura 2.3 – *Market Share* sistemas operacionais *mobile* no mundo

lador, documentação, códigos de exemplos e tutoriais. O ambiente de desenvolvimento oficialmente suportado atualmente é o Android Studio, com versões para Windows, Linux e MacOS. O desenvolvimento de aplicativos é livre e gratuito. Tais aplicativos podem ser adquiridos por usuários através do download e instalação do arquivo Apk ou através do download direto da loja de aplicativos virtual. A Google Play é a loja oficial dos usuários do Android e permite ao usuário visualizar, comprar, fazer downloads, reportar problemas, avaliar e atualizar aplicativos desenvolvidos pelo Google ou por terceiros.

2.4.1 Arquitetura do sistema operacional

O sistema Android possui uma arquitetura dividida em camadas, conforme mostra a figura 2.4. Cada camanda desempenha tarefas em diferentes níveis de acesso aos recursos do dispositivo.

A camada de Aplicações é onde executam todos os aplicativos Android, incluindo cliente de e-mail, navegador, contatos, entre outros. A maioria dos aplicativos são escritos usando a linguagem Java. O Quadro de Aplicações (ou *frameworks*) serve como suporte para o funcionamento da camada de Aplicações. Todos os aplicativos Android são baseados nestes *frameworks*. A camada de Bibliotecas inclui uma série de bibliotecas C/C++ usadas por vários componentes do sistema operacional e fornece suporte para o funcionamento dos *frameworks*.

A camada Android Tempo de Execução (ou *Runtime Android*) inclui uma série de bibli-

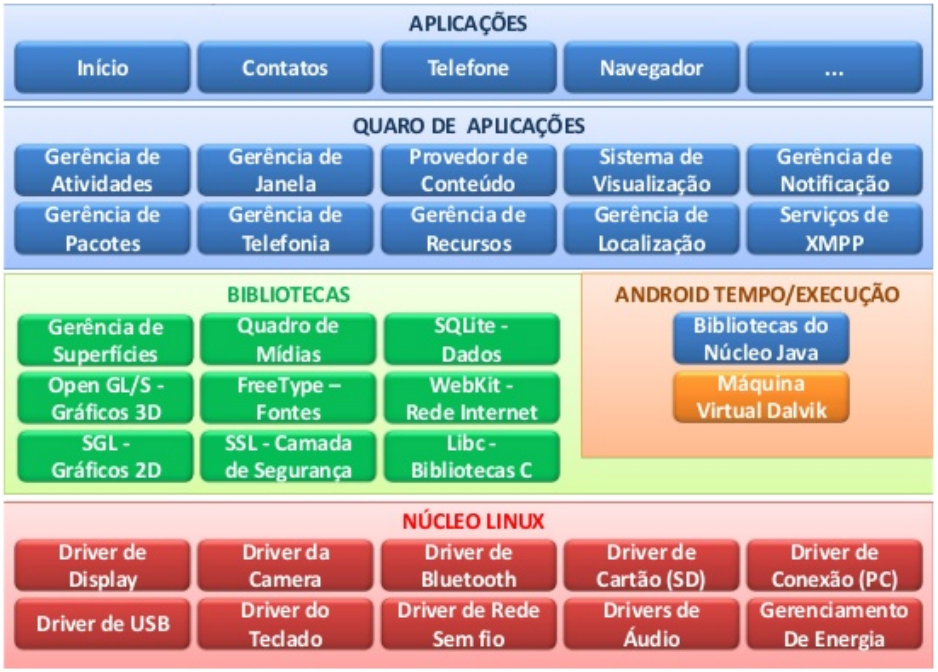

Figura 2.4 – Arquitetura sistema operacional Android (RIBEIRO, 2013)

otecas fundamentais e uma máquina virtual Java (Dalvik) que foi redefinida e otimizada pela Google para se adequar a plataforma Android. O núcleo Linux é localizado na camada mais baixa do sistema Android e atua como uma camada de abstração entre o hardware e o software. Ele fornece serviços fundamentais do sistema como segurança, gerenciamento de memória, processos e drivers.

2.4.2 Conceitos de aplicações

As aplicações para Android são escritas em linguagem Java. O código, depois de compilado para *bytecodes Dalvik*, é empacotado juntamente com outros recursos utilizados pela aplicação, como ícones e imagens, em um arquivo com a extensão Apk. Este arquivo é o meio de distribuição para os usuários instalarem a aplicação em seus dispositivos.

Por padrão, cada aplicação é executada em um ou mais processos próprios. Para cada aplicação é designado um ID de usuário Linux único e as permissões são dadas de forma que os arquivos fiquem visíveis apenas para a aplicação dona. As aplicações são construídas utilizando componentes que são instanciados no momento em que se tornam necessários. Existem quatro tipos de componentes básicos: atividades (ou *activities*), serviços (ou *services*), provedores de conteúdo (ou *content providers*) e receptores de broadcast (ou *broadcast receivers*).

- Uma atividade corresponde a uma tela para o usuário, normalmente permitindo interação

e resposta.

- Serviços não possuem uma interface gráfica e são utilizados para executar tarefas em segundo plano, como eventos periódicos de verificação em um banco de dados por exemplo.
- Os provedores de conteúdo servem para acessar ou disponibilizar dados específicos de uma aplicação para outras aplicações, por exemplo, um aplicativo de reprodução de músicas pode acessá-las em um cartão de memória.
- Os receptores de broadcast são componentes que ficam inativos e respondem a eventos em nível de sistema, como por exemplo, alerta de bateria fraca ou tela desligada.

Com exceção dos *content providers*, os outros três componentes são ativados através de mensagens assíncronas. *Intent* é o nome do objeto que contem a mensagem com a ação que se deseja executar. Por exemplo, para iniciar uma atividade é necessário enviar um *Intent* cujo conteúdo especifique esta intenção.

Todo arquivo Apk possui também um arquivo de manifesto onde estão declarados todos os componentes da aplicação. O arquivo é estruturado em linguagem XML e também é utilizado para declarar as bibliotecas utilizadas, permissões, versão e requisitos.

Quando o primeiro componente de uma aplicação precisa ser executado, um processo com uma única *thread* é iniciado. Por padrão, todos os componentes da aplicação são executados nesta *thread*. O Android dispõe de um mecanismo que permite a chamada de procedimentos remotos, executados em outros processos. Em determinado momento, quando a memória está escassa, por exemplo, o Android pode destruir um processo. Consequentemente, todos os componentes da aplicação que estão sendo executados naquele processo são destruídos. Para decidir qual processo deve ser eliminado, é levada em conta a importância dos processos para o usuário. Por exemplo, o sistema irá destruir primeiro aqueles que contêm atividades que não estão mais visíveis na tela. Este sistema de gerenciamento de memória é chamado *Garbage Collector*.

Cada componente de uma aplicação tem um ciclo de vida bem definido que inicia no momento em que é instanciado e acaba quando é destruído. Enquanto o componente está sendo executado, seu estado pode ser alterado diversas vezes.

Atividades no sistema são gerenciadas como uma pilha. Quando uma nova atividade é iniciada, ela é colocado no topo da pilha e torna-se a atividade de funcionamento. A figura 2.5 demonstra todos os estados possíveis de uma atividade, seus métodos e suas respectivas transições:

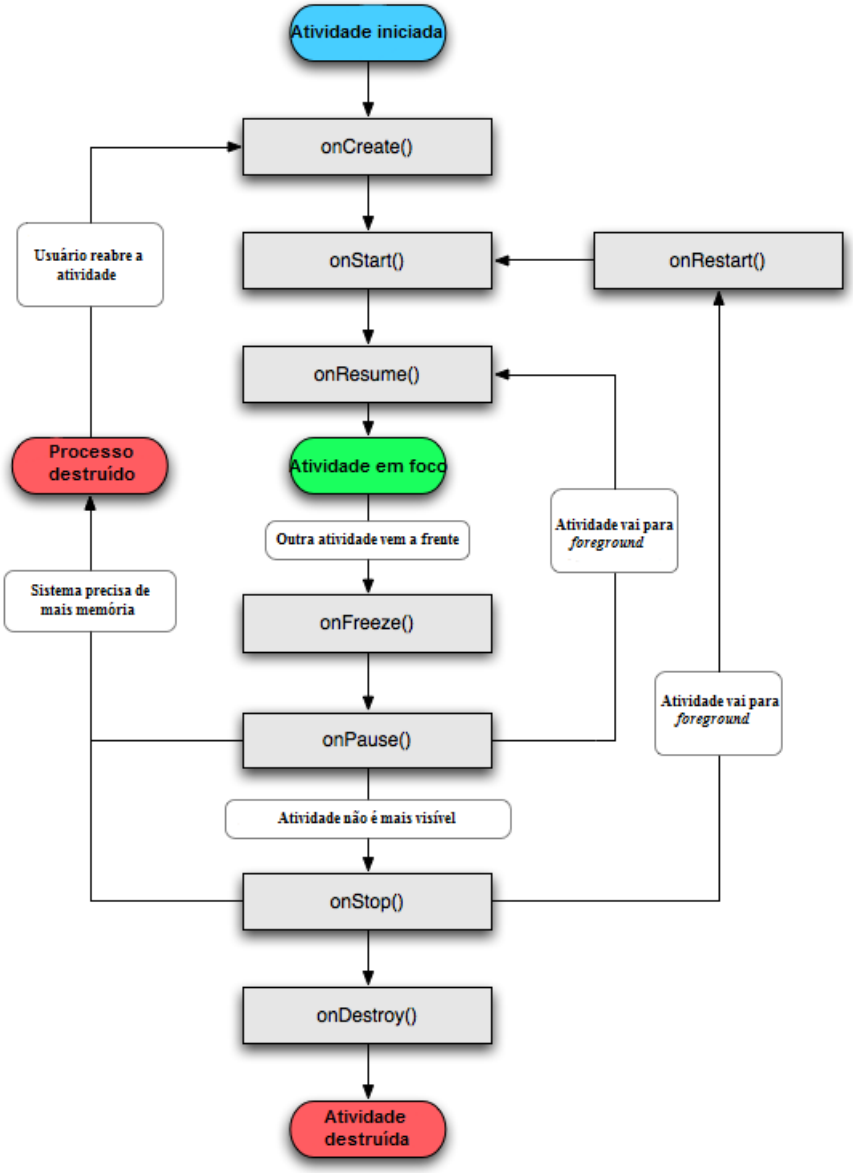

Figura 2.5 – Ciclo de uma atividade Android (DEVELOPER, 2015)

- Se uma atividade está no primeiro plano da tela, ou seja no topo da pilha de execução, ela é considerada ativa ou em execução.

- Se uma atividade tem o foco perdido, mas ainda é visível (isto é, uma nova atividade de menor tamanho ou transparente tem o foco em cima desta atividade), ela está em pausa . A atividade em pausa mantém todas as variáveis de estado, informações e é visível ao usuário (pelo menos parcialmente). Porém, atividades neste estado podem ser fechados pelo sistema operacional em casos de memória extremamente baixa.

- Se uma atividade está completamente sobreposta por outra atividade, ela é considerada parada. Ela ainda mantém todas as variáveis de estado, no entanto, não é mais visível

para o usuário e pode ser fechada caso o sistema operacional necessite de mais memória em outros processos.

- Se uma atividade está em pausa ou parada, o sistema pode eliminar a atividade da memória, seja solicitando seu término ou simplesmente matando seu processo. Quando a atividade é exibida novamente ao usuário, ela deve ser completamente reiniciada.

Como as atividades, os serviços também possuem estados e métodos de ciclo de vida que podem ser implementados para responder a mudanças de estado. Os receptores de broadcast possuem apenas um método de ciclo de vida que é chamado no momento em que uma mensagem chega para ele e o componente é considerado ativo apenas durante a execução deste método, ou seja, enquanto reage à mensagem.

2.4.3 Interface com o usuário

Aplicativos Android possuem a interface gráfica dividida em hierarquias. Todos os elementos da interface em um aplicativo Android são construídos usando objetos do tipo *View* e *ViewGroup*. A *View* é um objeto que desenha algo na tela com que o usuário pode ou não interagir, como botões, caixas de texto ou figuras. A *ViewGroup* é um objeto que contém outros objetos do tipo *View* (ou *ViewGroup*), com a função de definir o layout da interface, conforme demonstra a figura 2.6.

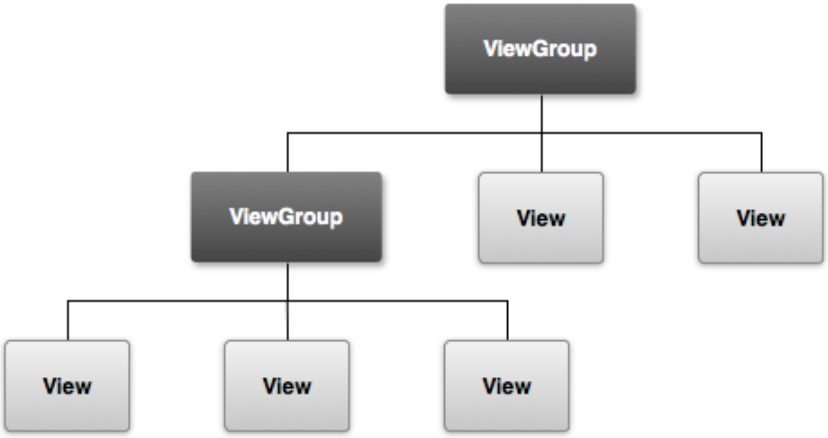

Figura 2.6 – Hierarquia de layout em aplicativo Android

Os objetos podem ser organizados automaticamente, dentro de uma região específica da tela de forma linear, relativa, entre outras,. Para cada uma delas, deve ser utilizado um layout específico. Por exemplo, os objetos filhos do *ViewGroup LinearLayout* são dispostos na tela

linearmente um seguido do outro, de forma horizontal ou vertical. Já no *RelativeLayout*, os elementos são posicionados com referência à outros elementos, pode-se especificar por exemplo, que um botão estará abaixo de um campo de texto, ao lado ou acima dele.

O Android Studio disponibiliza nativamente diversos *widgets* e *layouts* prontos que, em conjunto, permitem criar uma interface com o usuário bastante rica. Porém, se necessário, também é possível construir e adicionar componentes e temas personalizados.

2.4.4 Publicação de aplicativos

Hoje o Google Play hospeda aproximadamente 1,4 milhão de aplicativos, sendo o maior centro de aplicativos móveis do mundo (APPFIGURES, 2014). Para manter a ordem e valorizar aplicativos de boa qualidade, existe um sistema de reputação, onde usuários podem avaliar a qualidade dos aplicativos disponíveis. A publicação de novos aplicativos é feita através do Console de Desenvolvedor do Google Play. Através dele, com o pagamento de uma taxa de registro, desenvolvedores podem facilmente disponibilizar seus aplicativos a usuários de todo o mundo ou países específicos.

Conforme citado anteriormente, uma aplicação é distribuída através de um arquivo Apk que empacota seu código compilado juntamente com dados e recursos utilizados. Para publicar uma aplicação é necessário que ela esteja assinada digitalmente. Portanto, deve ser utilizada uma chave do desenvolvedor para identificá-lo como responsável pela aplicação. Essa chave é chamada de *KeyStore* e é de fundamental importância para a publicação do aplicativo. Depois de assinada, ela está pronta para ser distribuída para usuários com dispositivos equipados com a plataforma Android.

3 SISTEMA DE MONITORAMENTO SOLAR CLOUD

O sistema fotovoltaico atualmente existente no GEPOC e que foi utilizado para a integração do aplicativo emprega a estrutura mostrada na Figura 3.1. Os inversores recebem energia elétrica diretamente a partir dos painéis fotovoltaicos. Através do protocolo SunSpec desenvolvido nos inversores, é possível realizar a comunicação com outros dispositivos, como *data loggers*. Um *data logger* implementado em uma placa embarcada *BeagleBone Black* é conectado a um ou mais inversores fotovoltaicos, através de um barramento RS485, para obter seus dados, sendo também responsável pela aquisição e distribuição das um informações para um banco de dados MySQL *online*. O aplicativo desenvolvido neste trabalho, é definido a partir deste banco de dados.

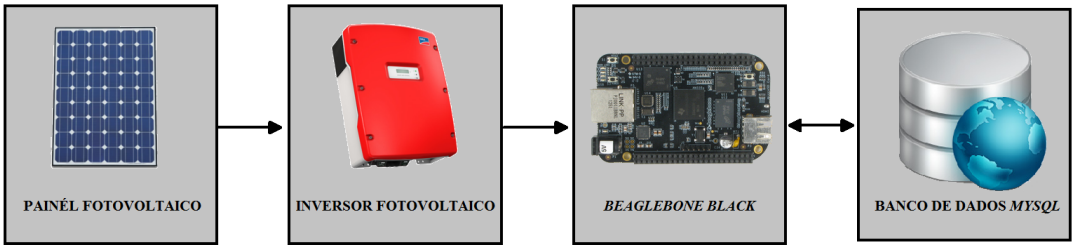

Figura 3.1 – Sistema de monitoramento

3.1 Banco de dados MySQL

O armazenamento e a recuperação de informações desempenham um papel fundamental na informática. (OLIVEIRA, 2002). Um sistema de banco de dados é uma coleção de dados inter-relacionados e um conjunto de programas que permitem aos usuários acessar e modificar esses dados (SILBERSCHATZ, 2006). Uma importante finalidade desse tipo de sistema é fornecer aos usuários uma visão abstrata dos dados, ou seja, o sistema oculta detalhes de como os dados são armazenados e consultados.

SQL do acrônimo *Structured Query Language*, ou seja Linguagem de Consulta Estruturada é a linguagem padrão de gerenciamento de dados, utilizada para manipular informações em bancos de dados relacionais. Os bancos de dados relacionais são o tipo de banco de dados mais comumente utilizados e são compostos de relações, mais comumente chamadas de tabelas (LUKE WELLING, 2005). O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL como interface. É atualmente um dos bancos de dados

mais populares, com mais de 10 milhões de instalações pelo mundo (DB-ENGINES, 2015).

O banco de dados do sistema fotovoltaico do GEPOC possuí tabelas independentes para cada inversor fotovoltaico. Ou seja, novos dispositivos podem ser adicionados ao sistema com a adição de uma nova tabela. Estas tabelas individuais armazenam as informações relativas a produção de energia, potência, tempo de uso e condição atual de cada inversor e são preenchidas diretamente a partir dos *data loggers* através do protocolo TCP/IP.

Para o correto funcionamento da escalabilidade e integralização do aplicativo foi necessário o desenvolvimento de novas tabelas no banco de dados. Entre elas, uma tabela de índice. Tal tabela é reponsável por armazenar informações relativas a marca, modelo, longitude, latitude entre outras. É fundamental atualizar a tabela de índice na adição ou remoção de inversores do sistema, pois ela é a principal referência do aplicativo quando este realiza consultas ao banco de dados.

Outra tabela desenvolvida exclusivamente para o aplicativo foi a tabela de mensagens. Esta tabela tem a função de receber mensagens informativas dos administradores do sistema. Tais mensagens são posteriormente mostradas dentro do aplicativo.

Também foi criada uma tabela para receber as informações meteorológicas. Essas informações ainda não foram totalmente obtidas e foram adicionadas manualmente. Porém a base está pronta para quando os dados meteorológicos estiverem disponíveis.

Outras duas tabelas foram desenvolvidas especificamente para o funcionamento do aplicativo. Uma delas é responsável por fornecer informações financeiras, como o preço do kWh. Outra é responsável por fornecer a produção esperada de energia do sistema em cada hora do dia.

3.2 *Web Services* PHP

O PHP, do acrônimo *Hypertext Preprocessor*, é uma linguagem de criação de scripts do lado do servidor projetada especificamente para a *Web*. Dentro de uma página HTML, pode-se embutir código de PHP que será executado toda vez que a página for visitada. O código de PHP é interpretado no servidor *Web* e gera HTML ou outra saída (LUKE WELLING, 2005).

Web services são componentes que permitem a comunicação através da internet entre diferentes tipos de aplicações. Eles fornecem uma base por meio da qual um programa cliente em uma organização pode interagir com um servidor em outra organização, sem supervisão humana (COULOURIS, 2007). Os dois principais tipos de protocolos de *Web Services* são o

SOAP (*Simple Object Access Protocol*) e o REST (*Representational State Transfer*).

O SOAP é um robusto protocolo de transferência de mensagens em formato XML para uso em ambientes distribuídos. O padrão SOAP funciona como um tipo de *framework* que permite a comunicação entre diversas plataformas com mensagens personalizadas. Em geral, SOAP é uma boa opção para sistemas com padrões rígidos e ambientes complexos (várias plataformas e sistemas). Este protocolo inclui uma série de descritores de segurança, que adicionam um *overhead* considerável na sua manipulação.

O REST é um protocolo de comunicação baseado mais simples. Ele não impõe restrições ao formato da mensagem, apenas no comportamento dos componentes envolvidos. A maior vantagem do protocolo REST é sua flexibilidade e velocidade. O desenvolvedor pode optar pelo formato mais adequado para as mensagens do sistema de acordo com sua necessidade específica. Os formatos mais comuns são *JavaScript Object Notation* (JSON) e texto puro, mas em teoria qualquer formato pode ser usado.

Por questões de segurança, a consulta ao banco de dados não foi realizada diretamente a partir do aplicativo. Para isso, foram criados *web services* na linguagem PHP junto ao servidor, que quando solicitados pelo aplicativo, acessam o banco de dados e retornam com a informação desejada. Tais *Web Services* foram desenvolvidos utilizando o protocolo REST devido a sua maior flexibilidade e velocidade. A segurança se explica pelo fato que através dos Web Services, as credenciais para acesso ao banco de dados não são transmitidas através internet, visto que o acesso ao banco de dados é feito por meio do servidor local. Além disso, o banco de dados da UFSM não permite acessos remotos, sendo obrigatório realizar as consultas dentro da rede da instituição.

No aplicativo, a solicitação aos arquivos PHP do servidor é feita através de uma conexão *Hypertext Transfer Protocol* (*HTTP*). O protocolo *HTTP* faz a comunicação entre o cliente e o servidor por meio de mensagens. O cliente envia uma mensagem de requisição de um recurso e o servidor envia uma mensagem de resposta ao cliente com a solicitação. Para cada conexão, o arquivo PHP realiza a consulta ao banco de dados e obtém a resposta. Uma vez obtida, a resposta é codificada para o formato *JSON array* e retornada ao aplicativo. A figura 3.2 exemplifica como funciona a organização do formato *JSON array* do arquivo PHP responsável pelas informações dos inversores.

A estrutura do formato *JSON array* dividide a resposta em seções e permite ao aplicativo extrair e processar a informação desejada conforme sua necessidade. Cada arquivo PHP

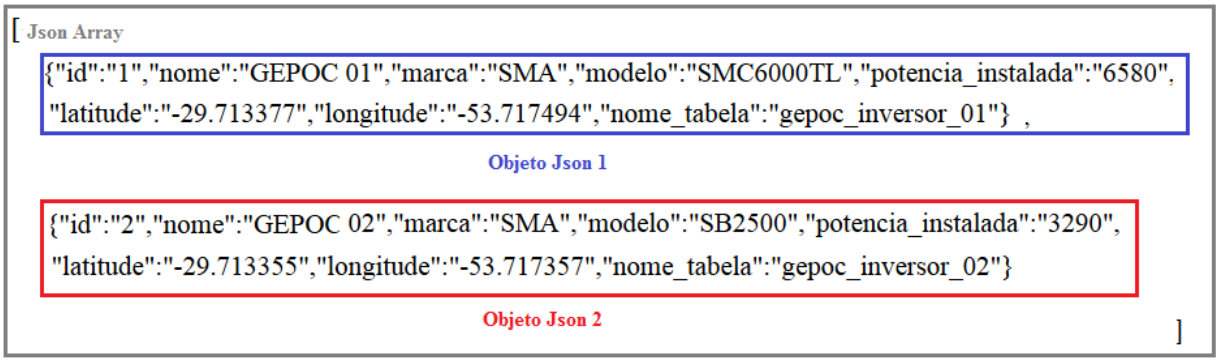

Figura 3.2 – Exemplo da estrutura *JSON array*

possui uma função específica dentro do sistema e suas solicitações são feitas conforme sua tela corresponde é acionada.

Por uma questão de organização e desempenho, foram criados *Web Services* independentes para cada parte do sistema. Dessa forma, o serviço não precisa carregar todas as informações do banco de dados a cada consulta, e sim somente as que lhe interessam no momento. Para o funcionamento correto de todos os recursos do aplicativo foram criados o total de 9 *Web Services*, são eles:

- CARREGA_GRAFICO: responsável por carregar os valores de produção de energia em cada hora do dia de um inversor especificado por parâmetro.
- CLIMA_ATUAL: responsável por carregar as informações meteorológicas.
- ENERGIA_DIA_MES_ANO: reponsável por carregar a energia produzida no dia, mês e ano de um inversor especificado por parâmetro.
- MENSAGEM_ATUAL: responsável por carregar mensagens inseridas no banco de dados.
- PREÇO_ATUAL: responsável por carregar o preço do kWh inserido no banco de dados.
- PRODUÇÂO_ESPERADA: responsável por carregar os valores esperados de produção em cada hora.
- SELECT_ALL: responsável por carregar todas informações da tabela de índice.
- SELECT_INFOS: responsável por carregar informações técnicas da tabela de índice de um inversor especificado por parâmetro.

- SELECT_LAST_NOME: responsável por carregar a potência atual de um inversor especificado por parâmetro.

  Os apêndices A, B, C, D, E, F, G, H e I apresentam o código relativo a cada *Web Service* desenvolvido no trabalho. Por questões de segurança, foi omitida a senha de acesso ao banco de dados.

3.3 Aplicativo Android

A base do funcionamento do aplicativo proposto neste trabalho necessita de 3 módulos para seu funcionamento: o banco de dados *MySQL*, *web services* PHP e o próprio aplicativo Android. A Figura 3.3 ilustra a organização desse sistema.

Figura 3.3 – Diagrama sistema Android PHP MySQL

O banco de dados é reponsável por armazenar as informações provindas dos inversores fotovoltaicos, da estação meteorológica, entre outras. Os *web services* são reponsáveis por intermediar a comunicação entre o aplicativo e o banco de dados. O aplicativo tem como função consultar os *Web Services*, interpretar a resposta e apresentar os resultados.

4 DESENVOLVIMENTO DO APLICATIVO

Este capítulo vai abordar mais profundamente o desenvolvimento do aplicativo Android, desde sua parte gráfica quanto da parte funcional.

4.1 Ferramentas utilizadas

O aplicativo foi desenvolvido através do Android Studio, ferramenta oficial de desenvolvimento do Google, lançado oficialmente em Dezembro de 2014. O Android Studio permite o desenvolvimento integrado da parte gráfica e da parte funcional do aplicativo, além de fornecer um ambiente completo para testes em máquinas virtuais ou dispositivos reais.

O banco de dados utilizado foi desenvolvido através do software PHP MyAdmin. Através dele foram criadas e configuradas todas as tabelas necessárias para o correto funcionamento do aplicativo. Os *Web Services* PHP foram construídos utilizando o editor de texto Notepad++. Os ícones e figuras usados dentro do aplicativo foram criados atráves do software InkScape.

4.2 Estrutura do aplicativo

O aplicativo foi desenvolvido utilizando o conceito de abas. Cada aba apresenta informações relativas a determinados contextos, são elas:

- Aba 1 - Potência atual do sistema, irradiação, temperatura, velocidade do vento e precipitação.
- Aba 2 - Geração diária, mensal, anual e gráfico da produção diária do sistema.
- Aba 3 - Marca, modelo, potência instalada e energia gerada no dia de cada inversor do sistema.
- Aba 4 - Localização geográfica dos inversores com imagens de satélite.
- Aba 5 - Dados relativos a econômia mensal e anual.
- Aba 6 - Créditos do sistema.

Para o funcionamento adequado da interface de abas, foi criada uma atividade especifica chamada *TabActivity*. Esse tipo especial de atividade utiliza o objeto *TabHost*, que é responsável

por gerenciar e instanciar as demais telas dentro dela mesma. Dessa forma, no instante em que uma aba é selecionada, a atividade correspondente a aquela aba é instanciada dentro da *TabActivity*, tornando a resposta do aplicativo rápida e eficiente.

Cada aba foi construída em arquivos independentes para facilitar o acesso a cada uma delas. Os arquivos referentes a cada aba e as demais classes desenvolvidas no trabalho e suas respectivas funções estão listadas abaixo:

- MainActivity.class: Gerenciar todas as abas e fornecer a estrutura do aplicativo através da *Tab Activity*.
- Intro.class: Apresentar a tela incial do aplicativo.
- Tela1.class: Atividade da aba 1.
- Tela2.class: Atividade da aba 2.
- Tela3.class: Atividade da aba 3.
- Tela4.class: Atividade da aba 4.
- Tela5.class: Atividade da aba 5.
- Tela6.class: Atividade da aba 6.
- MontaGrafico.class: Construir o gráfico que é usado na aba 2.
- IsrToString.class: Converter a resposta do *Web Service* para o formato String, utilizado nas abas 1, 2, 3, 4, 5 e na *Tab Activity*.

4.3 Conexão ao banco de dados

A conexão do aplicativo ao banco de dados foi através através de *Web Services* em PHP. O aplicativo tem a função de solicitar o acesso aos arquvos PHP através de uma conexão do tipo *HTTP*. Cada arquivo PHP é responsável por acessar o banco de dados efetivamente e fazer as consultas necessárias. O banco de dados por sua vez retorna a resposta para o arquivo PHP, que codifica essa resposta em um formato JSON, que será lido e interpretado pelo aplicativo.

A conexão *HTTP* é realizada através do método POST. Este método é seguro pois os parâmetros não são armazenados no histórico ou nos logs dos Web Services. Além disso, ele permite que qualquer tipo de dados sejam transmitidos.

Os dados vindos da conexão *HTTP* chegam no formato *InputStream*. O *InputStream* apresenta os dados como uma sequência contínua de bytes. Para interpretar esta sequência de bytes é necessário fazer sua conversão para o formato String. Tal conversão é feita através do arquivo IsrToString.class. A função responsável por tal conversão está apresentada no apêndice J.

Depois da resposta ser convertida para String, foi utilizado um objeto do tipo *JSON array*. Esse objeto é responsável por interpretar a String que está estruturada como um vetor de objetos JSON. Um laço de repetição percorre este *JSON array*, armazenando os valores desejados em suas respectivas variáveis, seja do tipo inteiro, String ou outras. A figura 4.1 ilustra como funciona o sistema de conexão e mostra algumas das funções utilizadas.

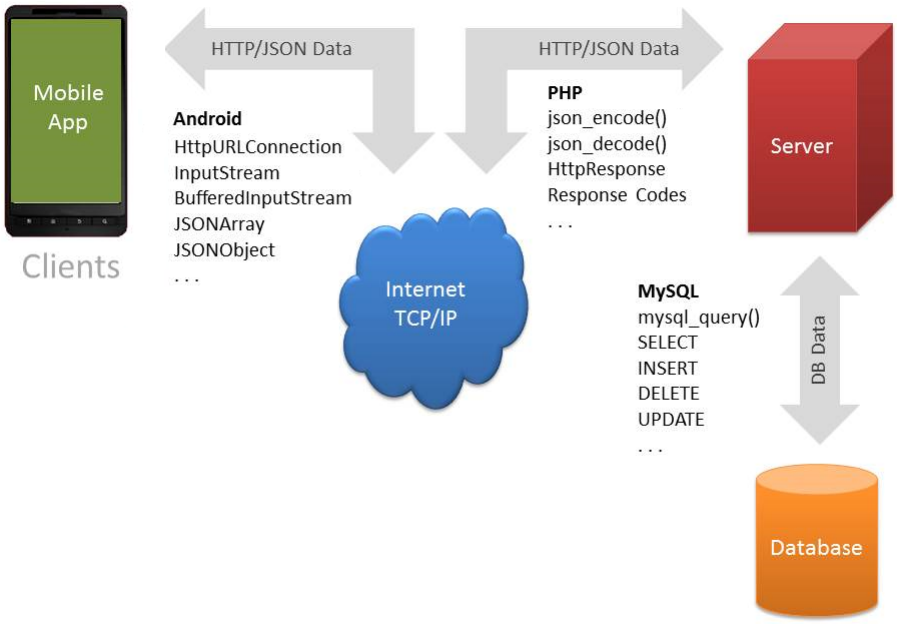

Figura 4.1 – Conexão ao banco de dados via *HTTP* e *Web Services*

Para que o sistema continuasse respondendo enquanto realiza a comunicação através da internet, foi necessário que toda a operação envolvendo uso de internet fosse feita através de *threads*. Para isso, foi utilizada a classe AsyncTask, que gerencia automaticamente as *threads* no sistema operacional. O apêndice K mostra um exemplo de conexão a um dos *Web Services* do sistema.

4.4 Bibliotecas adicionais

Além das bibliotecas inerentes ao SDK do Android Studio, outras duas bibliotecas foram utilizadas para a implementação dos recursos do aplicativo, a *AChartEngine* e a Google Maps API V2.

4.4.1 *AChartEngine*

A biblioteca *AChartEngine* é utilizada para a criação de gráficos em aplicativos Android. Através dela, é possível criar gráficos de linhas, barras, dispersão, sobrepostos, entre outros. Neste trabalho foi criado um gráfico de barras sobrepostas que representam a energia gerada pelo sistema ao longo do dia e seus valores esperados conforme a potência instalada.

Os gráficos desenvolvidos através desta biblioteca são instanciados dentro de um *layout* próprio, podendo ocupar toda a tela do aplicativo ou parte dela. Neste projeto o gráfico foi construído dentro da atividade referente a segunda aba, sendo instanciado dentro de um *layout* linear.

O gráfico é construído através da classe *GraphicalView*. Nesta classe, os dados são manipulados como objetos do tipo *CategorySeries*, que são simplesmente séries de qualquer tipo de dado. Neste trabalho foram utilizadas duas séries de dados do tipo *Double*.

Antes de serem enviados para classe construtora do gráfico, todos os valores são normalizados para porcentagem, com base na energia máxima obtível. Dessa forma, a visualização do gráfico pode ser feita de forma mais clara e limpa.

A primeira série recebe os valores referentes a energia produzida em cada hora do dia atual. Esses dados são obtidos a partir do banco de dados e corresepondem a energia somada de todos os inversores do sistema.

A segunda série recebe os valores refentes a quanto de energia era esperada em cada hora do dia, conforme a potência total instalada no sistema. Estes valores esperados são colocados manualmente no banco de dados e são calculados com base em algorítmos ainda em desenvolvimento pela equipe do GEPOC.

Depois de adicionadas as séries que servem como dados para a criação do gráfico, é necessário especificar o tipo de gráfico desejado. Neste caso foi projetado um gráfico de barras sobrepostas. Também é possível configurar diversas opções relativas ao visual do gráfico, como cor de fundo, tamanho da fonte, textos exibidos, legendas, entre outros.

4.4.2 Google Maps API V2

O aplicativo desenvolvido também fornece a localização geográfica dos inversores do sistema através de imagens de satélite. Tais imagens são obtidas a partir da biblioteca *Google Maps*. Para inserir um Google Map em um aplicativo Android é necessário o uso da API *Google Maps* V2. Para executar um recurso Google dentro de qualquer aplicativo é necessária a licença Google. Tal licença é dividida em dois tipos de chaves, ambas gratuitas:

- Chave para desenvolvimento - Essa chave é utilizada para o desenvolvimento e testes dentro do ambiente de desenvolvimento.
- Chave para distribuição - Essa chave é necessária para o funcionamento do mapa na versão final do aplicativo (arquivo Apk).

Para o correto funcionamento do aplicativo em dispositivos reais foi necessária a obtenção da chave para distribuição. Essa chave é obtida a partir do arquivo *KeyStore* no qual o aplicativo foi assinado digitalmente e deve ser atualizada caso seja utilizado outro *KeyStore*.

O componente *GoogleMap* exibe um mapa cujo conteúdo é requisitado em tempo real utilizando uma conexão com a internet. Portanto, é necessário que a aplicação esteja conectada à internet. Caso não haja conexão com a internet será simplesmente exibido um mapa do mundo.

O *GoogleMap* disponibiliza métodos para controlar o mapa que está sendo exibido. Neste trabalho, quando a função de criação do mapa é executada, o aplicativo faz uma consulta ao banco de dados e carrega os dados referentes a latitude e longitude de todos os inversores do sistema. Uma vez carregados estes dados, o aplicativo desenha marcadores especificando a localização de cada inversor, bem como seu nome.

A classe dispõe ainda de um sistema integrado de localização geográfica com base no GPS. Ela permite que seja traçada uma rota até o inversor a partir da localização atual do dispositivo. Obviamente, para tal funcionamente é necessário que o GPS do sistema esteja ativado. Tal função é inerente a esta biblioteca e não foi desenvolvida neste trabalho, mas pode ser usada normalmente.

4.5 Escalabilidade

Um dos principais desafios encontrados durante o desenvolvimento do aplicativo foi fazê-lo de forma escalável, ou seja, ele é capaz de se apdatar a sistemas fotovoltaicos de dife-

rentes tamanhos sem a necessidade de reconfiguração. Para tal feito, foi necessário criar uma tabela de índices no banco de dados, que precisa ser devidamente atualizada quando na adição ou remoção de inversores do sistema.

Através desta tabela de índice, o aplicativo é capaz de consultar o número de inversores atualmente instalados. Para isso foi definida uma variável do tipo inteiro que armazena o número de inversores do sistema. Através desta variável, o aplicativo é então capaz de executar as demais consultas dentro de um laço de repetição e armazená-los em variáveis ou listas correspondentes.

A tabela de índice também possui o nome de cada tabela referente a cada inversor, esse nome é importante pois é o parâmetro a ser enviado junto ao *Web Service*. Dessa forma, o *Web Service* sabe qual tabela deve ser consultada no banco de dados.

4.6 Sistema de mensagens

O aplicativo também dispõe de um sistema de mensagens, que permite que administradores do sistema divulguem informações ou avisos relevantes. Os administradores podem facilmente colocar novas mensagens no servidor acessando o servidor do banco de dados. As mensagens são carregadas a partir do banco de dados e acessadas através de um *Web Service* específico.

Foi desenvolvido um *timer* onde, a cada 10 minutos, é realizada a consulta ao servidor. Caso a mensagem atual possua um identificador diferente da mensagem anterior, ele é exibido através do elemento Toast.

A classe Toast fornece um feedback simples e rápido ao usuário em uma pequena janela. Essa janela é exibida na quantidade de espaço necessário para a mensagem que lhe foi determinada, permanecendo visível por um tempo limitado.

4.7 Permissões necessárias

O gerenciamento de permissões é uma função muito importante em dispositvos com sistema Android. As permissões gerenciam quais recursos do aparelho estarão disponíveis para o acesso do aplicativo. Todas as funções, atividades e autorizações necessárias para a execução deste aplicativo estão especificadas no arquivo AndroidManifest.xml, que pode ser visualizado no apêndice L. São elas:

- *INTERNET*: permissão fundamental para todo o funcionamento do sistema de conexão com a internet, indispensável para todas as telas do aplicativo.
- *ACCESS_NETWORK_STATE*: responsável por permitir a verificação do estado atual da conexão com a rede.
- *ACCESS_WIFI_STATE*: responsável por permitir a verificação do estado atual da conexão através da wi-fi.
- *WRITE_EXTERNAL_STORAGE*: esta é um requisito para a integração com o Google Maps, ela permite que dados sejam escritos em dispostivos externos de armazenamento, como cartões de memória.
- *READ_GSERVICES*: também é um requisito para a integração com o Google Maps, ela permite a leitura do estado atual dos serviços Google.
- *ACCESS_COARSE_LOCATION*: outro requisito para o Google Maps, esta permissão autoriza o aplicativo a consultar a localização atual do dispositivo através do sistema GPS.
- *ACCESS_FINE_LOCATION*: semelhante a permissão anterior, porém esta permite o acesso a localização de forma mais precisa.

4.8 Telas geradas

Esta seção vai abordar como cada tela do aplicativo foi desenvolvida, tanto a nível gráfico (arquivos XML), quanto a nível funcional (arquivos Java). Todas as imagens aqui apresentas foram obtidas a partir do ambiente de desenvolvimento e não representam o resultado final obtido em dispositivos reais. Tais resultados serão apresentados no capítulo seguinte.

4.8.1 Tela de introdução

A tela de introdução, ilustrada na figura 4.2, é a primeira tela executada na inicialização do aplicativo. Ela consiste de um *layout* relativo com uma imagem centralizada e caixas de textos no rodapé da tela.

Sua parte funcional também é simples e consiste apenas de um *timer* que, após 2 segundos, carrega a tela seguinte por meio de um *Intent*.

Figura 4.2 – Tela de introdução

4.8.2 *MainActivity*

Esta tela, ilustrada na figura 4.3 é responsável por instanciar as demais abas dentro dela, através do elemento TabHost. Sua interface foi feita sobre um *layout* relativo, que é divido em três partes principais: outro *layout* relativo onde são exibidas informações de data, hora e localização, um *FrameLayout*, onde as demais abas serão instanciadas, e um *TabWidget*, onde os ícones referentes a cada aba são exibidos.

Sua parte funcional consiste na declaração das demais abas e seus respectivos ícones, bem como as ações a serem tomadas quando cada ícone é pressionado. O apêndice M apresenta como foi o desenvolvido o objeto *TabHost* neste trabalho. Esta atividade também possui uma função onde a data e hora atual do sistema é sincronizada com o dispositivo e exibida na parte superior. Também é responsabilidade desta tela carregar e exibir as mensagens vindas do banco de dados através de um *Web Service* específico e do elemento *Toast*.

4.8.3 Primeira aba

Esta tela, ilustrada na figura 4.4 representa a primera das seis abas do sistema. Para o desenvolvimente de sua interface foi utilizado um *layout* linear. A tela consiste basicamente de imagens seguidas de caixas de textos informativos. Também existe um elemento do tipo *ProgressBar*, que exibe uma barra de progresso personalizável relativa a porcentagem da potência

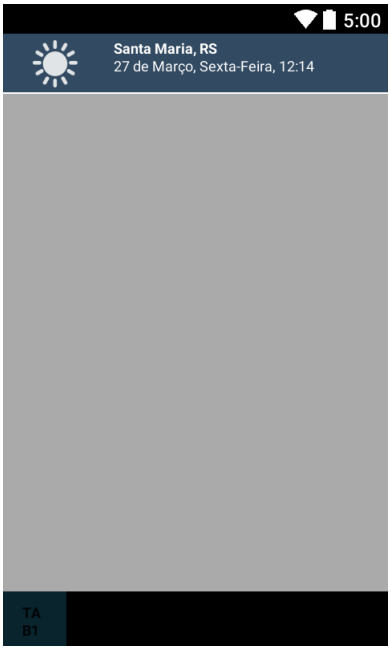

Figura 4.3 – *MainActivity*

atual com base na potência instalada.

Figura 4.4 – Aba 1

Sua parte funcional é composta por funções que acessam o banco de dados por meio dos *Web Services*, tratam e exibem os valores obtidos em seus respectivos campos. Também foi criada uma função para tratar especificamente da barra de progresso e seu texto. A cada 15 segundos, um *timer* executa as novamente todas estas funções e atualiza todos os valores da tela. Todas as consultas a *Web Services* foram realizadas através de *threads* independentes

através da função AsyncTask para manter a fluência do sistema.

4.8.4 Segunda aba

Esta aba, ilustrada na figura 4.5 apresenta informações relativas a produção de energia diária, mensal e anual do sistema fotovoltaico. Além disso, essa tela também mostra o gráfico da produção de energia em cada hora do dia. Sua interface foi feita através de um *layout* relativo com figuras, caixas de texto e um *frame* para o gráfico .

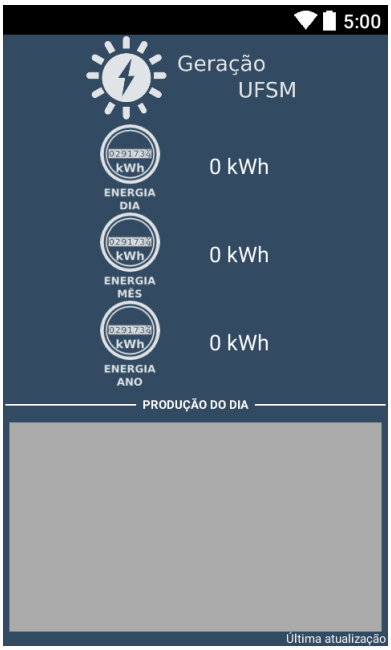

Figura 4.5 – Aba 2

O funcionamento desta tela é semelhante a aba 1, o aplicativo é responsável por consultar o banco de dados através de Web Services e adquirir as informações relativas a produção de energia diária, mensal e anual. Para o funcionamento do gráfico, o aplicativo executa a consulta ao banco de dados para os valores obtidos em cada hora do dia e também para os valores esperados de produção.

Depois de obtidos os valores de produção de cada hora e da produção esperada, uma função normaliza os valores para porcentagem com base no valor máximo esperado e os envia para a classe responsável pela criação do gráfico (MontaGrafico). Finalmente, a classe MontaGrafico retorna o gráfico construído e o posiciona no local especificado. O código responsável por construir o gráfico utilizado neste aplicativo pode ser encontrado no apêndice N. Esta tela também possui um *timer* que atualiza todos os valores a cada 10 segundos.

4.8.5 Terceira aba

Esta aba, ilustrada na figura 4.6 apresentas informações técnicas relativas a cada inversor do sistema. Sua interface possui um botão que quando clicado exibe uma caixa de diálogo listando todos os inversores do sistema. Quando um dos inversores é selecionado, suas informações são buscadas no banco de dados e colocadas em suas caixas de texto correspondentes.

Figura 4.6 – Aba 3

Para a implementação da caixa de diálogo foi criado um objeto do tipo *AlertDialog* que é instanciado no momento em que o botão é clicado. Para preencher este objeto com os nomes de cada inversor foi necessário fazer uma consulta a tabela de índice do banco de dados e armazenar estes nomes na variável listaInversores. O apêndice O mostra como foi implementado o *AlertDialog*.

Também foi necessário armazenar o nome de cada tabela de inversor do sistema. Esse dado é armazenado na variável listaTabelasInversores e serve como parâmetro no momento da consulta ao *Web Service*.

4.8.6 Quarta aba

A interface da quarta aba do aplicativo, ilustrada na figura 4.7, é composta somente por um fragmento onde o mapa é instanciado. Tal fragmento não exibe nada no ambiente de desenvolvimento pois a construção do mapa é feita em tempo de execução e não pré-definida.

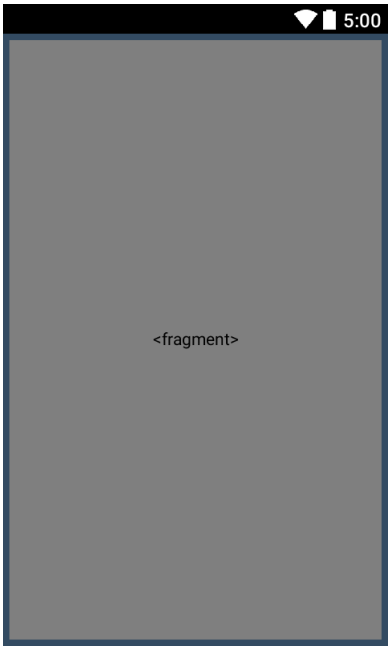

Figura 4.7 – Aba 4

A parte funcional dessa aba possui a função LerTodosInversores. Esta função é responsável por consultar o banco de dados através de um *Web Service* específico e armazenar latitude, longitude e nome de cada inversor do sistema.

Uma vez adquiridos estes valores, a função *setUpMap* é responsável por construir o mapa. Nesta função é especificado que o mapa deverá usar imagens de satélite e definida a taxa de aproximação incial da câmera. Ainda nesta função *setUpMap* são adicionados os marcadores com o nome de cada inversor em suas respectivas localizações. O apêndice P mostra um trecho do código responsável pela construção do mapa.

4.8.7 Quinta aba

A quinta aba do sistema, ilustrada na figura 4.8 apresenta informações relativas a economia gerada devido ao sistema fotovoltaico. Sua interface é simples, composta por figuras e caixas de texto em um *layout* relativo.

A parte funcional desta aba é relativamente simples. Ela possui as funções lerPreco-Atual e energia_dia_mes_ano que são responsáveis por consultar *Web Services* específicos e armazenar o valor do preço atual e da energia mensal e anual de todo o sistema. Com base nesses dados, é calculada a economia mensal e anual gerada pelo sistema e os dados são colocados em seus respectivos campos.

Figura 4.8 – Aba 5

4.8.8 Sexta aba

A sexta aba, ilustrada na figura 4.9 exibe os créditos do sistema completo, desde a parte de hardware e protocolos até o desenvolvimento do aplicativo. Sua interface foi construída com figuras e caixas de texto em múltiplos *layoutes* relativos.

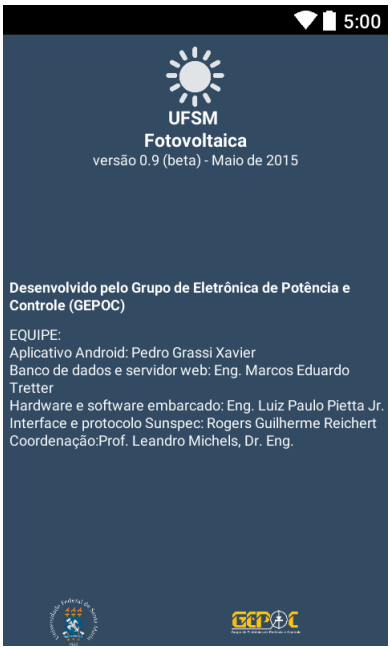

Figura 4.9 – Aba 6

5 RESULTADOS

Esta capítulo mostra os resultado obtidos em um dispositivo real. Todas as imagens apresentadas neste capítulo foram obtidas com o aplicativo rodando em um *smartphone* Motorola Razr D1 com sistema operacional Android 4.4.

5.1 Sistema implementado

A figura 5.1 mostra a primeira aba com o sistema de monitoramento sendo executado. Pode-se observar a geração de potência do sistema fotovoltaico e informações meteorológicas no momento em que a tela foi capturada.

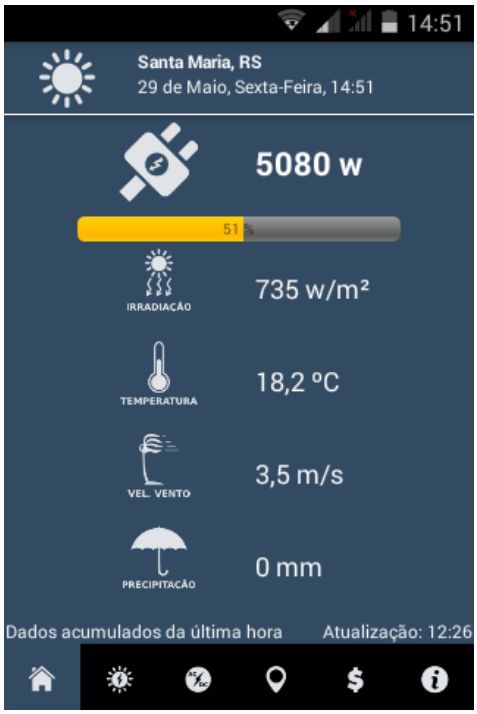

Figura 5.1 – Primeira aba em um dispositvo real.

A tabela 5.1 mostra os dados de potência atual de cada inversor do sistema no momento em que a imagem foi capturada. A partir dela, podemos comprovar que a informação apresentada pelo aplicativo esta correta.

A figura 5.2 mostra a segunda aba com o sistema de monitoramento sendo executado. Pode-se observar a geração diária, mensal e anual do sistema fotovoltaica bem como a produção

Tabela 5.1 – Potência gerada no sistema

INVERSOR	TIMESTAMP	POTÊNCIA ATUAL
GEPOC 01	2015-05-29 14:50:36	3475 W
GEPOC 2	2015-05-29 14:50:36	1605 W
POTÊNCIA TOTAL		5080 W

de energia em cada hora do dia até o momento em que a imegem foi capturada.

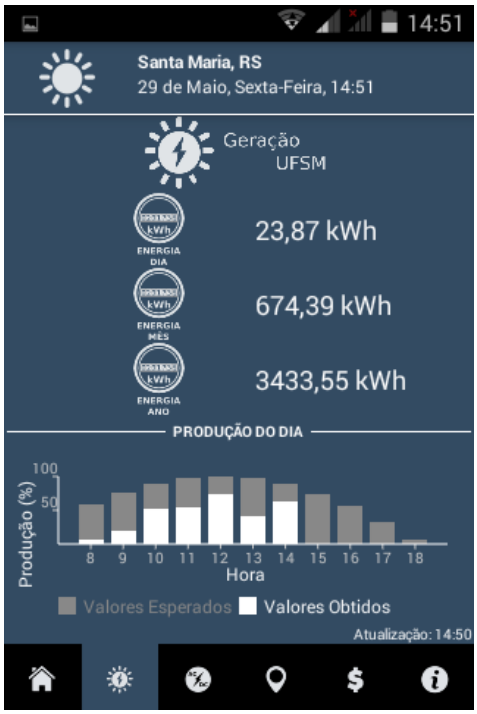

Figura 5.2 – Segunda aba em um dispositvo real.

A tabela 5.2 mostra a energia acumulada ao longo do dia até o momento em que a figura foi capturada. A partir dela, observa-se que a energia total produzida pelos dois inversores ao longo do dia é igual a apresentada pelo aplicativo.

Tabela 5.2 – Energia gerada pelos inversores no dia

INVERSOR	ENERGIA TOTAL	ENERGIA INÍCIO DO DIA	ACUMULADA
GEPOC 01	7323,61 kWh	7316,12 kWh	7,49 kWh
GEPOC 02	16046,14 kWh	16029,76 kWh	16,38 kWh
TOTAL			23,87 kWh

A tabela 5.3 mostra a energia acumulada ao longo do mês até o momento em que a figura foi capturada. A partir dela, observa-se que a energia total produzida pelos dois inversores ao longo do mês é igual a apresentada pelo aplicativo.

Tabela 5.3 – Energia gerada pelos inversores no mês

INVERSOR	ENERGIA TOTAL	ENERGIA INÍCIO DO MÊS	ACUMULADA
GEPOC 01	7323,61 kWh	7114,3 kWh	209,31 kWh
GEPOC 02	16046,14 kWh	15581,06 kWh	465,08 kWh
TOTAL			674,39 kWh

A tabela 5.4 mostra a energia acumulada ao longo do ano até o momento em que a figura foi capturada. A partir dela, observa-se que a energia total produzida pelos dois inversores ao longo do ano é igual a apresentada pelo aplicativo.

Tabela 5.4 – Energia gerada pelos inversores no ano

INVERSOR	ENERGIA TOTAL	ENERGIA INÍCIO DO ANO	ACUMULADA
GEPOC 01	7323,61 kWh	6252,80 kWh	1070,81 kWh
GEPOC 02	16046,14 kWh	13683,40 kWh	2362,74 kWh
TOTAL			3433,55 kWh

Os dados do gráfico também foram conferidos e correspondem aos valores apresentados no banco de dados.

A figura 5.3 mostra a terceira aba com o sistema de monitoramento sendo executado. Foi selecionado o inversor GEPOC 01 para visualização de suas informações técnicas e geração diária de energia.

A figura 5.4 mostra a quarta aba com o sistema de monitoramento sendo executado. Nela, pode-se observar o sistema de localização geográfica em funcionamento com a indicação correta da localização dos inversores do sistema.

A figura 5.5 mostra a quinta aba com o sistema de monitoramento sendo executado. Pode-se observar a economia resultante no mês e no ano com a implementação do sistema

Figura 5.3 – Terceira aba em um dispositvo real.

Figura 5.4 – Quarta aba em um dispositvo real.

fotovoltaico tendo como base o preço atual do kWh.

A figura 5.6 mostra a sexta e última aba do sistema. Nela, pode-se observar a identificação dos desenvolvedores de cada parte do sistema.

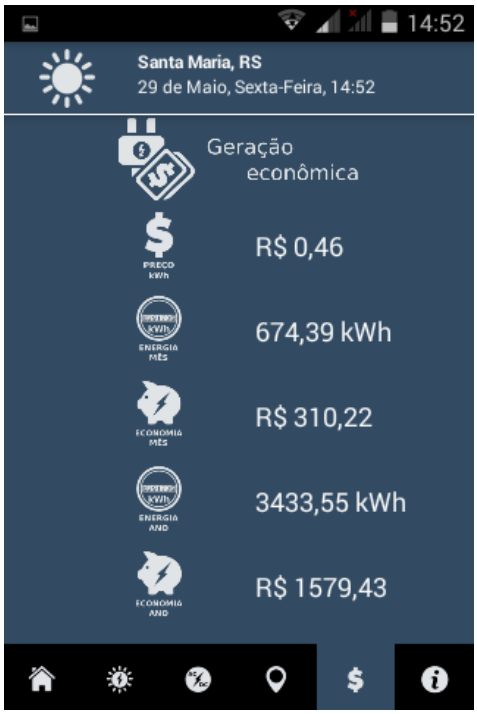

Figura 5.5 – Quinta aba em um dispositvo real.

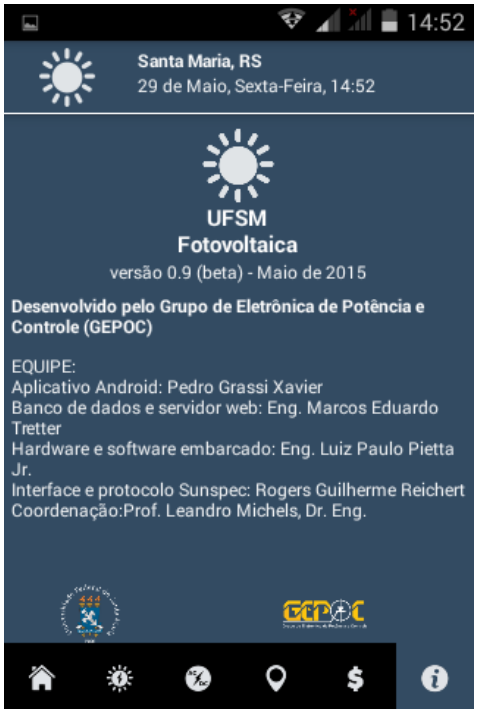

Figura 5.6 – Sexta aba em um dispositvo real.

6 CONCLUSÃO

Este capítulo apresenta considerações a respeito do trabalho e do sistema de monitoramento desenvolvido, bem como sugestões para outros trabalhos relacionados.

6.1 Sistema implementado

O objetivo deste trabalho era desenvolver um aplicativo para a plataforma Android capaz de monitorar usinas fotovoltaicas. O aplicativo foi desenvolvido de forma integrada ao sistema de geração de energia fotovoltaica existente no GEPOC/UFSM. No geral, o trabalho transcorreu de forma positiva, com o correto cumprimento das metas estipuladas.

Porém, algumas dificuldades foram encontradas ao longo do desenvolvimento do aplicativo. Entre elas, o funcionamento da biblioteca do Google Maps em dispostivos reais, pois foi necessária a obtenção da chave de distribuição, que não é a mesma utilizada durante o ambiente de desenvolvimento. Outro imprevisto ocorreu quando da leitura de mensagens oriundas do banco de dados, pois a codificação empregada no banco de dados era diferente daquela lida pelo aplicativo. Foi necessário fazer a conversão para o formato UTF-8 através do *Web Service*. No período de testes em dispositivos reais observou-se uma incompatibilida com o antivírus *Avast Mobile*, que acusava um alerta de ameaça pelo fato do aplicativo ainda não ter sido devidamente publicado. Foi contatado o suporte técnico do *Avast* e resolvido o problema.

No geral, o aplicativo Android mostrou-se como uma útil ferramenta para o monitoramento de sistemas fotovoltaicos. Tanto a nível residencial, comercial ou industrial, o aplicativo desenvolvido apresenta as informações de forma rápida e precisa, além de se adequar automaticamente a sistemas de qualquer tamanho devido a sua escalabilidade.

6.2 Sugestões para trabalhos futuros

Trabalhos futuros podem complementar ainda mais o sistema aqui desenvolvido. Novos recursos podem ser implementados ao aplicativo utilizando diversos dos conceitos apresentados neste trabalho. Um exemplo seria um sistema de notificações que emite um alerta ao dispositivo quando determinado evento importante acontece no sistema fotovoltaico. Ou ainda, explorar a possibilidade de controle remoto dos inversores do sistema através do protocolo SunSpec. Além disso, o sistema poderia ser implementado para outros sistemas operacionais móveis.

REFERÊNCIAS

ALLIANCE, S. **Communicating the Customer Benefits of Information Standards**. Acessado em Junho/2015, http://sunspec.org/wp-content/uploads/2012/06/SunSpec-White-Paper-Benefits-of-Standards.pdf.

ANDREONI, M. Implementation of wireless remote monitoring and control of solar photovoltaic (PV) system. In: Montevideo, Uruguai. **Anais...** IEEE Computer Society, 2012. p.1–6.

APPFIGURES. **App Stores Growth Accelerates in 2014**. Acessado em Maio/2015, http://blog.appfigures.com/app-stores-growth-accelerates-in-2014/.

BLUESOL. **As três principais formas de aproveitamento da energia solar**. Acessado em Maio/2015, http://www.blue-sol.com/energia-solar/as-tres-principais-formas-de-aproveitamento-da-energia-solar/.

COULOURIS. **Sistemas distribuídos Conceitos e Projeto**. [S.l.]: Editora Bookman, 2007.

DB-ENGINES. **DB-Engines Ranking**. Acessado em Maio/2015, http://db-engines.com/en/ranking.

DEVELOPER, A. **Activities**. Acessado em Junho/2015, http://developer.android.com/guide/components/activities.html.

IDC. **Smartphone OS Market Share, Q4 2014**. Acessado em Maio/2015, http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

IEA. **Snapshot of Global PV Markets**. Acessado em Junho/2015, http://helapco.gr/pdf/PVPS-report-A-Snapshot-of-Global-PV-1992-2014.pdf.

KUN, X. et al. Design and application of the photovoltaic inverter equipment monitoring system on Android platform. In: IEEE COMPUTER SOCIETY, Busan, Coréia do Sul. **Anais...** IEEE Computer Society, 2013. p.1494 – 1497.

LIN, W. Y. et al. Design and Implementation of Health Monitoring System for Solar Panel in IPv6 Network. In: IEEE COMPUTER SOCIETY, Barcelona, Espanhal. **Anais...** IEEE Computer Society, 2014. p.57–60.

LOBO, A. F. **Simulação de sistemas de aquecimento solar com materiais em mudança de fase (MMF) adaptados de resíduos pesados do refino do petróleo sob condições transitórias de insolação e demanda.** 2004. Dissertação de Mestrado — Universidade Federal do Paraná, Curitiba - Paraná.

LUKE WELLING, L. T. **PHP e MySQL Desenvolvimento Web**. [S.l.]: Editora Campus, 2005.

OLIVEIRA, C. H. P. de. **SQL Curso Prático**. [S.l.]: Editora Novatec, 2002.

PEREIRA, A. N. et al. Architecture of information system for monitoring of photovoltaic plants. In: IEEE COMPUTER SOCIETY, Kitakyushu, Japãol. **Anais...** IEEE Computer Society, 2014. p.1–6.

PEREIRA, F. D. S.; OLIVEIRA, M. . S. **Curso técnico instalador de energia solar fotovoltaica.** 2011.

PHONEARENA. **Android's Google Play beats App Store with over 1 million apps, now officially largest**. Acessado em Maio/2015, http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest.

RIBEIRO. **Android**: um novo paradigma de desenvolvimento móvel. Acessado em Junho/2015, http://www.cesar.org.br/site/files/file/WM18-Android.pdf.

SILBERSCHATZ. **Sistemas de Banco de Dados**. [S.l.]: Editora Campus, 2006.

YAROW, J. **This Chart Shows Google's Incredible Domination Of The World's Computing Platforms**. Acessado em Maio/2015, http://www.businessinsider.com/androids-share-of-the-computing-market-2014-3.

APÊNDICES

APÊNDICE A – CARREGA_GRAFICO.php

```
<?php
  $inversor = $_POST['inversor'];
  $tabela = (string)$inversor;
  date_default_timezone_set("Brazil/East");
  $hora_8 = date('Y-m-d').' 08:00:00';
  $hora_9 = date('Y-m-d').' 09:00:00';
  $hora_10 = date('Y-m-d').' 10:00:00';
  $hora_11 = date('Y-m-d').' 11:00:00';
  $hora_12 = date('Y-m-d').' 12:00:00';
  $hora_13 = date('Y-m-d').' 13:00:00';
  $hora_14 = date('Y-m-d').' 14:00:00';
  $hora_15 = date('Y-m-d').' 15:00:00';
  $hora_16 = date('Y-m-d').' 16:00:00';
  $hora_17 = date('Y-m-d').' 17:00:00';
  $hora_18 = date('Y-m-d').' 18:00:00';
  $hora_19 = date('Y-m-d').' 19:00:00';
  // Conecta com o banco de dados
  $con =
     mysqli_connect('localhost','fotovoltaica','xxx','FOTOVOLTAICA');

  // Obtem a primeira linha da hora 8
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela WHERE
    timestamp > '$hora_8' ORDER BY id LIMIT 1"));
  $primeiro_valor = $inversor01['e_total'];
  // Obtem a primeira linha da hora 9
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela WHERE
    timestamp > '$hora_8' AND timestamp < '$hora_9' ORDER BY id
       DESC LIMIT 1"));
  $ultimo_valor = $inversor01['e_total'];
  $e_hora = floatval ($ultimo_valor - $primeiro_valor);
  $e_hora8 = round($e_hora, 2); // Arredondar


  // Obtem a primeira linha da hora 9
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela WHERE
    timestamp > '$hora_9' ORDER BY id LIMIT 1"));
  $primeiro_valor = $inversor01['e_total'];
  // Obtem a primeira linha da hora 10
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela WHERE
    timestamp > '$hora_9' AND timestamp < '$hora_10' ORDER BY id
       DESC LIMIT 1"));
  $ultimo_valor = $inversor01['e_total'];
  $e_hora = floatval ($ultimo_valor - $primeiro_valor);
  $e_hora9 = round($e_hora, 2); // Arredondar
```

```
// Obtem a primeira linha da hora 10
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_10' ORDER BY id LIMIT 1"));
$primeiro_valor = $inversor01['e_total'];
// Obtem a primeira linha da hora 11
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_10' AND timestamp < '$hora_11' ORDER BY id
     DESC LIMIT 1"));
$ultimo_valor = $inversor01['e_total'];
$e_hora = floatval ($ultimo_valor - $primeiro_valor);
$e_hora10 = round($e_hora, 2); // Arredondar


// Obtem a primeira linha da hora 11
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_11' ORDER BY id LIMIT 1"));
$primeiro_valor = $inversor01['e_total'];
// Obtem a primeira linha da hora 12
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_11' AND timestamp < '$hora_12' ORDER BY id
     DESC LIMIT 1"));
$ultimo_valor = $inversor01['e_total'];
$e_hora = floatval ($ultimo_valor - $primeiro_valor);
$e_hora11 = round($e_hora, 2); // Arredondar


// Obtem a primeira linha da hora 12
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_12' ORDER BY id LIMIT 1"));
$primeiro_valor = $inversor01['e_total'];
// Obtem a primeira linha da hora 13
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_12' AND timestamp < '$hora_13' ORDER BY id
     DESC LIMIT 1"));
$ultimo_valor = $inversor01['e_total'];
$e_hora = floatval ($ultimo_valor - $primeiro_valor);
$e_hora12 = round($e_hora, 2); // Arredondar


// Obtem a primeira linha da hora 13
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_13' ORDER BY id LIMIT 1"));
```

```
$primeiro_valor = $inversor01['e_total'];
// Obtem a primeira linha da hora 14
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_13' AND timestamp < '$hora_14' ORDER BY id
     DESC LIMIT 1"));
$ultimo_valor = $inversor01['e_total'];
$e_hora = floatval ($ultimo_valor - $primeiro_valor);
$e_hora13 = round($e_hora, 2); // Arredondar


// Obtem a primeira linha da hora 14
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_14' ORDER BY id LIMIT 1"));
$primeiro_valor = $inversor01['e_total'];
// Obtem a primeira linha da hora 15
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_14' AND timestamp < '$hora_15' ORDER BY id
     DESC LIMIT 1"));
$ultimo_valor = $inversor01['e_total'];
$e_hora = floatval ($ultimo_valor - $primeiro_valor);
$e_hora14 = round($e_hora, 2); // Arredondar


// Obtem a primeira linha da hora 15
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_15' ORDER BY id LIMIT 1"));
$primeiro_valor = $inversor01['e_total'];
// Obtem a primeira linha da hora 16
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_15' AND timestamp < '$hora_16' ORDER BY id
     DESC LIMIT 1"));
$ultimo_valor = $inversor01['e_total'];
$e_hora = floatval ($ultimo_valor - $primeiro_valor);
$e_hora15 = round($e_hora, 2); // Arredondar


// Obtem a primeira linha da hora 16
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_16' ORDER BY id LIMIT 1"));
$primeiro_valor = $inversor01['e_total'];
// Obtem a primeira linha da hora 17
$inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
   $tabela WHERE
  timestamp > '$hora_16' AND timestamp < '$hora_17' ORDER BY id
     DESC LIMIT 1"));
```

```
  $ultimo_valor = $inversor01['e_total'];
  $e_hora = floatval ($ultimo_valor - $primeiro_valor);
  $e_hora16 = round($e_hora, 2); // Arredondar


  // Obtem a primeira linha da hora 17
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela WHERE
    timestamp > '$hora_17' ORDER BY id LIMIT 1"));
  $primeiro_valor = $inversor01['e_total'];
  // Obtem a primeira linha da hora 18
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela WHERE
    timestamp > '$hora_17' AND timestamp < '$hora_18' ORDER BY id
       DESC LIMIT 1"));
  $ultimo_valor = $inversor01['e_total'];
  $e_hora = floatval ($ultimo_valor - $primeiro_valor);
  $e_hora17 = round($e_hora, 2); // Arredondar


  // Obtem a primeira linha da hora 18
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela WHERE
    timestamp > '$hora_18' ORDER BY id LIMIT 1"));
  $primeiro_valor = $inversor01['e_total'];
  // Obtem a primeira linha da hora 19
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela WHERE
    timestamp > '$hora_18' AND timestamp < '$hora_19' ORDER BY id
       DESC LIMIT 1"));
  $ultimo_valor = $inversor01['e_total'];
  $e_hora = floatval ($ultimo_valor - $primeiro_valor);
  $e_hora18 = round($e_hora, 2); // Arredondar

  //Obtem timestamp
  $query = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela ORDER BY id DESC LIMIT 1"));
  $timestamp = $query['timestamp'];

  $output[]=array('hora8' => $e_hora8, 'hora9' => $e_hora9, 'hora10'
     => $e_hora10,
  'hora11' => $e_hora11, 'hora12' => $e_hora12, 'hora13' =>
     $e_hora13, 'hora14' => $e_hora14,
  'hora15' => $e_hora15, 'hora16' => $e_hora16, 'hora17' =>
     $e_hora17, 'hora18' => $e_hora18,
  'timestamp'=>$timestamp);

  print(json_encode($output));
  mysqli_close($con);
?>
```

APÊNDICE B – CLIMA_ATUAL.php

```
<?php
  $con = mysqli_connect("coralx.ufsm.br","fotovoltaica",
    "xxx","FOTOVOLTAICA") or die ("Erro");

  // Consulta
  $query = ("SELECT * FROM clima ORDER BY id DESC LIMIT 1");

  $resultado = mysqli_query($con, $query);

  while($row = mysqli_fetch_assoc($resultado)){
    $output[]=array_map('utf8_encode',$row);
  }

  print(json_encode($output));

  mysqli_close($con);

?>
```

APÊNDICE C – ENERGIA_DIA_MES_ANO.php

```
<?php

    $inversor = $_POST['inversor'];
   //$inversor = 'gepoc_inversor_01';
    $tabela = (string)$inversor;


   // PEGA O VALOR DO DIA

   $data = date("Y-m-d");

   $data_inicial = $data.' 00:00:00';
   $data_final = $data.' 23:59:59';

    // Conecta com o banco de dados
    $con =
       mysqli_connect('localhost','fotovoltaica','xxx','FOTOVOLTAICA');

    // Obtem a primeira linha do banco de dados de acordo com a
       pesquisa
    $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
       $tabela
     WHERE timestamp BETWEEN '$data_inicial' AND '$data_final' ORDER
        BY id LIMIT 1"));
    $primeiro_valor = $inversor01['e_total'];

    // Obtem a ultima linha do banco de dados de acordo com a pesquisa
    $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
       $tabela
     WHERE timestamp BETWEEN '$data_inicial' AND '$data_final' ORDER
        BY id DESC LIMIT 1"));
    $ultimo_valor = $inversor01['e_total'];

   $e_dia = $ultimo_valor-$primeiro_valor;


   // PEGA O VALOR DO MES

    $data = date("Y-m");

   $data_inicial = $data.'-01 00:00:00';
   $data_final = $data.'-31 23:59:59';

    // Obtem a primeira linha do banco de dados de acordo com a
       pesquisa
    $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
       $tabela
```

```
    WHERE timestamp BETWEEN '$data_inicial' AND '$data_final' ORDER
       BY id LIMIT 1"));
  $primeiro_valor = $inversor01['e_total'];

  // Obtem a ultima linha do banco de dados de acordo com a pesquisa
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela
    WHERE timestamp BETWEEN '$data_inicial' AND '$data_final' ORDER
       BY id DESC LIMIT 1"));
  $ultimo_valor = $inversor01['e_total'];

 $e_mes = $ultimo_valor-$primeiro_valor;


 // PEGA O VALOR DO ANO

  $data = date("Y");

 $data_inicial = $data.'-01-01 00:00:00';
 $data_final = $data.'-12-31 23:59:59';

  // Obtem a primeira linha do banco de dados de acordo com a
     pesquisa
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela
    WHERE timestamp BETWEEN '$data_inicial' AND '$data_final' ORDER
       BY id LIMIT 1"));
  $primeiro_valor = $inversor01['e_total'];

  // Obtem a ultima linha do banco de dados de acordo com a pesquisa
  $inversor01 = mysqli_fetch_assoc(mysqli_query($con,"SELECT * FROM
     $tabela
    WHERE timestamp BETWEEN '$data_inicial' AND '$data_final' ORDER
       BY id DESC LIMIT 1"));
  $ultimo_valor = $inversor01['e_total'];

 $e_ano = $ultimo_valor-$primeiro_valor;

  $output[]=array('edia' => $e_dia, 'emes' => $e_mes, 'eano' =>
     $e_ano);

 print(json_encode($output));

  mysqli_close($con);

?>
```

APÊNDICE D – MENSAGEM_ATUAL.php

```
<?php

  $con = mysqli_connect("coralx.ufsm.br","fotovoltaica",
    "xxx","FOTOVOLTAICA") or die ("Erro");

  // Consulta
  $query = ("SELECT * FROM mensagens ORDER BY id DESC LIMIT 1");

  $resultado = mysqli_query($con, $query);

  while($row = mysqli_fetch_assoc($resultado)){
    $output[]=array_map('utf8_encode',$row);
  }

  print(json_encode($output));

  mysqli_close($con);

?>
```

APÊNDICE E – PRECO_ATUAL.php

```
<?php

  $con = mysqli_connect("coralx.ufsm.br","fotovoltaica",
    "xxx","FOTOVOLTAICA") or die ("Erro");

  // Consulta
  $query = ("SELECT * FROM custos ORDER BY id DESC LIMIT 1");

  $resultado = mysqli_query($con, $query);

  while($row = mysqli_fetch_assoc($resultado)){
    $output[]=$row;
  }

  print(json_encode($output));

  mysqli_close($con);

?>
```

APÊNDICE F – PRODUCAO_ESPERADA.php

```
<?php

  $con = mysqli_connect("coralx.ufsm.br","fotovoltaica",
    "xxx","FOTOVOLTAICA") or die ("Erro");

  // Consulta
  $query = ("SELECT * FROM producao_esperada ORDER BY id DESC LIMIT
     1");

  $resultado = mysqli_query($con, $query);

  while($row = mysqli_fetch_assoc($resultado)){
    $output[]=$row;
  }

  print(json_encode($output));

  mysqli_close($con);

?>
```

APÊNDICE G – SELECT_ALL.php

```
<?php

  $con = mysqli_connect("coralx.ufsm.br","fotovoltaica",
    "xxx","FOTOVOLTAICA") or die ("Erro");

  // Consulta
  $query = ("SELECT * FROM dispositivos_inversor ORDER BY id");

  $resultado = mysqli_query($con, $query);

  while($row = mysqli_fetch_assoc($resultado)){
    $output[]=array_map('utf8_encode',$row);
  }

  print(json_encode($output));

  mysqli_close($con);

?>
```

APÊNDICE H – SELECT_INFOS.php

```
<?php

  $inversor = $_POST['inversor'];
      $inversor = (string)$inversor;

  $con = mysqli_connect("coralx.ufsm.br","fotovoltaica",
    "xxx","FOTOVOLTAICA") or die ("Erro");

  // Consulta
  $query = ("SELECT * FROM dispositivos_inversor WHERE nome =
     '$inversor'");

  $resultado = mysqli_query($con, $query);

  while($row = mysqli_fetch_assoc($resultado)){
    $output[]=array_map('utf8_encode',$row);
  }

  print(json_encode($output));

  mysqli_close($con);

?>
```

APÊNDICE I – SELECT_LAST_NOME.php

```
<?php

  $inversor = $_POST['inversor'];

      $tabela = (string)$inversor;

  $con = mysqli_connect("coralx.ufsm.br","fotovoltaica",
    "xxx","FOTOVOLTAICA") or die ("Erro");

  // Consulta
  $query = ("SELECT * FROM $tabela ORDER BY id DESC LIMIT 1");

  $resultado = mysqli_query($con, $query);

  while($row = mysqli_fetch_assoc($resultado)){
    $output[]=$row;
  }

  print(json_encode($output));

  mysqli_close($con);

?>
```

APÊNDICE J – Função responsável pela conversão de *InputStream* para String

```
public String converte(InputStream isr){

    // Converte Resposta para String
    try {
       assert isr != null;
       BufferedReader reader = \\new BufferedReader(new
          InputStreamReader(isr, "iso-8859-1"), 8);
       StringBuilder sb = new StringBuilder();
       String line;
       while ((line = reader.readLine()) != null) {
          sb.append(line).append("\n");
       }
       isr.close();
       result = sb.toString();
    } catch (Exception e) {
       e.printStackTrace();
    }

    return result;

}
```

APÊNDICE K – Exemplo de código para acesso e leitura de um Web Service

```
// Carrega a potencia total, o numero de inversores e o nome da
   tabela de cada inversor.
private class lerPotenciaTotal extends AsyncTask<String, Void, Void>
   {

      protected Void doInBackground(String... params) {

         potencia_total = 0;
         numero_de_inversores = 0;
         nomes_tabelas.clear();

         String result;
         InputStream isr = null;

         // Conecta ao PHP
         try {
            HttpClient httpclient = new DefaultHttpClient();
            HttpPost httpPost = new
            HttpPost("http://w3.ufsm.br/fotovoltaica/app/SELECT_ALL.php");
            HttpResponse response = httpclient.execute(httpPost);
            HttpEntity entity = response.getEntity();
            isr = entity.getContent();
         } catch (Exception e) {
            e.printStackTrace();
         }

         // Converte Resposta para String
         result = conversor.converte(isr);

         // Parse Json Data
         try {
            JSONArray jArray = new JSONArray(result);
            for (int i = 0; i < jArray.length(); i++) {
               JSONObject json = jArray.getJSONObject(i);
               potencia_total = potencia_total +
                  json.getInt("potencia_instalada");
               numero_de_inversores++;
               nomes_tabelas.add(i, json.getString("nome_tabela"));
            }
         } catch (Exception e) {
            e.printStackTrace();
         }
         return null;
      }
}
```

APÊNDICE L – AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="fotovoltaica.gepoc.ufsm.br.ufsmfotovoltaica">

   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission
      android:name="android.permission.ACCESS_NETWORK_STATE" />
   <uses-permission
      android:name="android.permission.ACCESS_WIFI_STATE" />
   <uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <uses-permission android:name=
   "com.google.android.providers.gsf.permission.READ_GSERVICES" />
   <uses-permission
      android:name="android.permission.ACCESS_COARSE_LOCATION" />
   <uses-permission
      android:name="android.permission.ACCESS_FINE_LOCATION" />

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="UFSM Fotovoltaica"
      android:theme="@style/AppTheme">
      <activity
         android:name=".MainActivity"
         android:label="MainActivity"
         android:screenOrientation="portrait" />
      <activity
         android:name=".Intro"
         android:label="UFSM Fotovoltaica"
         android:screenOrientation="portrait">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category
               android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity
         android:name=".Tela6"
         android:label="Infos"
         android:screenOrientation="portrait" />
      <activity
         android:name=".Tela4"
         android:label="Mapa"
         android:screenOrientation="portrait" />

      <meta-data
```

```
        android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="AIzaSyC4zK9Sn7x5DXey50T2RAb7IpVn9do0YSs" />

    <activity
        android:name=".Tela1"
        android:label="Home"
        android:screenOrientation="portrait" />
    <activity
        android:name=".Tela2"
        android:label="Foto"
        android:screenOrientation="portrait" />
    <activity android:name="org.achartengine.GraphicalActivity" />
    <activity
        android:name=".Tela3"
        android:label="Inversores"
        android:screenOrientation="portrait" />
    <activity
        android:name=".Tela5"
        android:label="Econ"
        android:screenOrientation="portrait" />
</application>
```

APÊNDICE M – Exemplo de uso do objeto *Tab Host*

```
final TabHost tabHost = getTabHost();

    TabSpec spec1 = tabHost.newTabSpec("Tab 1");
    spec1.setIndicator("",
       getResources().getDrawable(R.drawable.icon_home));
    Intent homeIntent = new Intent(this, Tela1.class);
    spec1.setContent(homeIntent);

    TabSpec spec2 = tabHost.newTabSpec("Tab 2");
    spec2.setIndicator("",
       getResources().getDrawable(R.drawable.icon_foto));
    Intent fotoIntent = new Intent(this, Tela2.class);
    spec2.setContent(fotoIntent);

    TabSpec spec3 = tabHost.newTabSpec("Tab 3");
    spec3.setIndicator("",
       getResources().getDrawable(R.drawable.icon_acdc));
    Intent inversoresIntent = new Intent(this, Tela3.class);
    spec3.setContent(inversoresIntent);

    TabSpec spec4 = tabHost.newTabSpec("Tab 4");
    spec4.setIndicator("",
       getResources().getDrawable(R.drawable.icon_maps));
    Intent mapaIntent = new Intent(this, Tela4.class);
    spec4.setContent(mapaIntent);

    TabSpec spec5 = tabHost.newTabSpec("Tab 5");
    spec5.setIndicator("",
       getResources().getDrawable(R.drawable.icon_econ));
    Intent econIntent = new Intent(this, Tela5.class);
    spec5.setContent(econIntent);

    TabSpec spec6 = tabHost.newTabSpec("Tab 6");
    spec6.setIndicator("",
       getResources().getDrawable(R.drawable.icon_info));
    Intent infosIntent = new Intent(this, Tela6.class);
    spec6.setContent(infosIntent);

    tabHost.addTab(spec1);
    tabHost.addTab(spec2);
    tabHost.addTab(spec3);
    tabHost.addTab(spec4);
    tabHost.addTab(spec5);
    tabHost.addTab(spec6);
```

APÊNDICE N – Função responsável por construir o gráfico

```
public GraphicalView getView(Context context, ArrayList<Double> y,
   ArrayList<Double> producao_esperada) {

      CategorySeries series = new CategorySeries("Valores Obtidos");

      for (int i = 0; i < y.size(); i++) {
         series.add(y.get(i));
      }

      CategorySeries series2 = new CategorySeries("Valores
         Esperados");

      for (int i = 0; i < producao_esperada.size(); i++) {
         series2.add(producao_esperada.get(i));
      }

      XYMultipleSeriesDataset dataset = new
         XYMultipleSeriesDataset();
      dataset.addSeries(series2.toXYSeries());
      dataset.addSeries(series.toXYSeries());


      XYMultipleSeriesRenderer mRenderer = new
         XYMultipleSeriesRenderer();
      mRenderer.setXTitle("Hora");
      mRenderer.setYTitle("Producao (%)");
      mRenderer.setAxesColor(Color.WHITE);
      mRenderer.setLabelsColor(Color.WHITE);
      mRenderer.setBarSpacing(0.3);
      mRenderer.setXAxisMax(series.getItemCount() + 1);
      //mRenderer.setXAxisMax(12);
      mRenderer.setXAxisMin(0);
      mRenderer.setYAxisMax(100);
      mRenderer.setYAxisMin(0);
      mRenderer.setYLabelsAlign(Paint.Align.RIGHT);
      mRenderer.setMarginsColor(Color.argb(0x00, 0x01, 0x01, 0x01));
      mRenderer.setPanEnabled(false, false);
      mRenderer.setXLabels(0);
      mRenderer.setYLabels(0);
      mRenderer.addYTextLabel(50, "50");
      mRenderer.addYTextLabel(100, "100");
      mRenderer.setLegendTextSize(12);
      mRenderer.setFitLegend(true);

      for (int i = 1; i < 12; i++) {
         mRenderer.addXTextLabel(i, String.valueOf(i + 7));
      }
```

```
    // Customizar barra 1
     XYSeriesRenderer renderer = new XYSeriesRenderer();
     renderer.setDisplayChartValues(false);
     renderer.setColor(Color.GRAY);
     mRenderer.addSeriesRenderer(renderer);


    // Customizar barra 2
     XYSeriesRenderer renderer2 = new XYSeriesRenderer();
     renderer2.setDisplayChartValues(false);
     renderer2.setColor(Color.WHITE);
     mRenderer.addSeriesRenderer(renderer2);

     return ChartFactory.getBarChartView(context, dataset,
        mRenderer, BarChart.Type.STACKED);

}
```

APÊNDICE O – Ação ao selecionar um inversor e construção da caixa de diálogo.

```
// Acao ao clicar no botao do inversor.
   public void clickInv(View view) {

     // Carrega a lista de inversores a partir da lista que contem o
        nome dos inversores
      final CharSequence[] inversores =
      listaInversores.toArray(new
         CharSequence[listaInversores.size()]);


      // Cria a caixa de dialogo com os nomes dos inversores
      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setTitle("Selecione o inversor: ");
      builder.setSingleChoiceItems(inversores, -1, new
         DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int item) {

         // salva o nome do inversor selecionado a partir da lista
            de inversores
            nome_inversor = listaInversores.get(item);

            // salva o nome da tabela do inversor selecionado a
               partir da lista de tabelas
            nome_tabela_inversor = listaTabelasInversores.get(item);

            bInv.setText(nome_inversor); // Exibe na tela o nome do
               inversor
            new lerDados().execute(); // Chama a funcao para ler as
               informacoes do inversor selecionado
            new energia_dia().execute(); // Chama a funcao para ler
               a energia produzida no dia
            invDialog.dismiss();

         }
      });
      invDialog = builder.create();
      invDialog.show();

   }
```

APÊNDICE P – Trecho responsável por carregar os dados e construir o mapa

```
// Carrega os nomes, latitudes e longitudes.
   private class LerTodosInversores extends AsyncTask<String, Void,
      Void> {

      protected Void doInBackground(String... params) {
         String result;
         InputStream isr = null;

         // Conecta ao PHP
         try {
            HttpClient httpclient = new DefaultHttpClient();
            HttpPost httpPost = new
            HttpPost("http://w3.ufsm.br/fotovoltaica/app/SELECT_ALL.php");
            HttpResponse response = httpclient.execute(httpPost);
            HttpEntity entity = response.getEntity();
            isr = entity.getContent();
         } catch (Exception e) {
            e.printStackTrace();
         }

         // Converte Resposta para String
         result = conversor.converte(isr);

         // Parse Json Data
         try {
            JSONArray jArray = new JSONArray(result);
            for (int i = 0; i < jArray.length(); i++) {

               numero_de_inversores++;
               JSONObject json = jArray.getJSONObject(i);
               nomes.add(i, json.getString("nome"));
               latitudes.add(i, json.getString("latitude"));
               longitudes.add(i, json.getString("longitude"));

            }
         } catch (Exception e) {
            e.printStackTrace();
         }
         return null;
      }


      // Chama a funcao para construir o mapa
      protected void onPostExecute(Void result) {
         if (numero_de_inversores != 0)
```

```
            setUpMapIfNeeded();
        }

// Chama a funcao que monta o mapa
    private void setUpMapIfNeeded() {
        if (mMap == null) {
            mMap = ((SupportMapFragment)
            getSupportFragmentManager().findFragmentById(R.id.map)).getMap();
            if (mMap != null) {
                setUpMap();
            }
        }
    }

// Monta o mapa.
    private void setUpMap() {
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new
            LatLng(Double.parseDouble(latitudes.get(0)),
            Double.parseDouble(longitudes.get(0))), 15));
        mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);

        // Para cada inversor coloca um marcador no mapa
        for (int i = 0; i < numero_de_inversores; i++) {
            mMap.addMarker(new MarkerOptions().position(new
                LatLng(Double.parseDouble(latitudes.get(i)),
                Double.parseDouble(longitudes.get(i)))).title(nomes.get(i)));
        }
    }
```