FUNDAÇÃO UNIVERSIDADE DO TOCANTINS
CURSO DE SISTEMAS DE INFORMAÇÃO

JAIRO GERVASIO DE CARVALHO

UTILIZAÇÃO DE GEOREFERENCIAMENTO EM DIPOSITIVOS MÓVEIS APLICADOS A GESTÃO DE ÁREAS COMPROMETIDAS POR ÁGUAS PLUVIAIS

Palmas – TO,
2015.

JAIRO GERVASIO DE CARVALHO

UTILIZAÇÃO DE GEOREFERENCIAMENTO EM DIPOSITIVOS MÓVEIS APLICADOS A GESTÃO DE ÁREAS COMPROMETIDAS POR ÁGUAS PLUVIAIS

Trabalho de Conclusão do Curso de Sistemas de Informação da Fundação Universidade do Tocantins, apresentado como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.
Orientador: Prof. Msc. Alex Coelho

Palmas – TO,

2015.

Dados Internacionais da catalogação na publicação (CIP)
Biblioteca da Universidade Estadual do Tocantins
Campus Graciosa – Palmas – TO

Carvalho, Jairo Gervasio de

C331u Utilização de georeferenciamento em dispositivos móveis aplicados a gestão de áreas comprometidas por águas pluviais / Jairo Gervasio de Carvalho – Palmas - TO, 2018.
87 fls.; il.; col.

Inclui CD-ROM

Orientação: Profº. Msc. Alex Coelho

TCC (Trabalho de Conclusão de Curso). Sistemas de Informação. Universidade Estadual do Tocantins. 2018

1. Arduino. 2. Energia Renovável. 3. Smart Cities. 4. Web Services - Dispositivos computacionais móveis. I. Coelho, Alex II. Título. III. Sistemas de informação.

CDD: 004.06

Ficha catalográfica elaborada pela Bibliotecária – Maria Madalena Camargo – CRB 2/1527

UNITINS

Coordenação do Curso de Sistemas de Informação

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO DO CURSO DE SISTEMAS DE INFORMAÇÃO DA FUNDAÇÃO UNIVERSIDADE DO TOCANTINS - UNITINS

Aos **26** dias do mês de **Junho** de **2015**, reuniu-se na Fundação Universidade do Tocantins, às **09:40 horas**, sob a Coordenação do Professor **Silvano Malfatti**, a banca examinadora de Trabalho de Conclusão de Curso em Sistemas de Informação, composta pelos examinadores Professor **Alex Coelho** (Orientador) e Professor **Igor Ypes**, para avaliação da defesa do trabalho intitulado **"Utilização de Georreferenciamento em Dispositivos Móveis Aplicados a Gestão de Áreas Comprometidas por Águas Pluviais"** da acadêmico **Jairo Gervasio de Carvalho** como requisito para aprovação na disciplina Trabalho de Conclusão de Curso (TCC). Após exposição do trabalho realizado pelo acadêmica e arguição pelos Examinadores da banca, em conformidade com o disposto no Regulamento de Trabalho de Conclusão de Curso em Sistemas de Informação, a banca atribuiu a pontuação: 9,0.

Sendo, portanto, o Acadêmico: (X) Aprovado ( ) Reprovado

Assinam esta Ata:

Professor Orientador: _____

Examinador: _____

Examinador: Silvano Maneck Malfatti

Acadêmico: Jairo Gervasio de Carvalho

Silvano Maneck Malfatti
Coordenação de Sist. de Informação
Matrícula 001558
Port/Unitins/GRE

Professor Silvano Malfatti
Coordenador do Curso de Sistemas de Informação

DEDICATÓRIA

Aos meus pais, pelo apoio incondicional.

AGRADECIMENTOS

Agradeço a Deus pelo dom da força, perseverança e pela oportunidade de concluir o curso, toda honra e gloria seja direcionada a ti meu Deus! Aos meus pais Juraci e Maria e meu irmão que me amaram e me acompanharam durante toda minha vida. A minha noiva que suportou e compreendeu o fato de ficarmos varias semanas sem nos vermos. A todos os meus tios e tias em especial: Gilberto, Mª Aparecida, Mª Luiza, Eurípedes, Jamira e Mirce, que sempre me apoiaram. Agradeço também a vários primos e amigos com carinho Cristiane, Wherbert, Suzane, Ismael, Jeane, Wesley[2], Jonh e Joilson pela amizade e pela vontade de me verem feliz. Agradeço aos amigos e colegas que me acompanharam nesta jornada acadêmica e hoje seguem outros caminhos, em especial Agnélio, Adria, Celia, Diogenes, Cristhiann, Andreia, Luídine, Franscisco, Átila e Brunno. Agradecer aos professores que me foram exemplo e espelho, me ensinando com muita humildade, sabedoria e puxões de orelha, dos quais lembrarei e guardarei com muito carinho pela importância que tiveram em minha vida. Obrigado pelos vários livros emprestados tio Silvano. E por fim meu orientador exemplo de profissional, um mestre na acepção da palavra, Alex Coelho, por acreditar em meu potencial.

EPÍGRAFE

“No que diz respeito ao desempenho, ao compromisso, ao esforço, à dedicação, não existe meio termo. Ou você faz uma coisa bem feita ou não faz”.

Ayrton Senna.

RESUMO

As tecnologias da informação e comunicação aplicadas às cidades proporciona aos habitantes uma experiência ágil e agradável em todos os aspectos, a facilidade de informações privilegiadas ou serviços como mobilidade urbana, segurança publica, meteorologia, saúde e distribuição de energia; na palma das mãos através dos dispositivos computacionais móveis antecipa possíveis problemas e norteia para soluções viáveis e eficazes. As Cidades Inteligentes (*Smart Cities*) agregam valor quanto a fontes de energias renováveis, fluidez no transito, estratégias de coibição da criminalidade e indicação de áreas de risco. Este trabalho propõe o desenvolvimento de um sistema de gestão para áreas de risco por alagamento, inferindo a posição de uma região alagada no mapa através de sistemas de georeferenciamento. Para o desenvolvimento deste sistema foram utilizadas diversas tecnologias, sendo as principais a placa *Arduino*, Painel Fotovoltaico, *Web Service RESTful* e a plataforma *Android*.

Palavras-Chave: Arduino, Energia Renovável, Smart Cities, Web Services, Dispositivos Computacionais Móveis, Georeferenciamento.

ABSTRACT

Information and communication technologies applied to cities give people an quick and enjoyable experience in all their aspects, the facility to have access to privileged information or services such as urban mobility, public safety, meteorology, health and energy distribution; on our hands via mobile computing devices anticipates potential issues and guides for viable and effective solutions. The Smart Cities (Smart Cities) add value as sources of renewable energy, fluidity in traffic, deterrence strategies of crime and indicating risk areas. This paper proposes the development of a management system for hazardous areas by flooding, inferring the position of a flooded region on the map through geo-referencing systems. To develop this system it was used many technologies, the main ones were Arduino board, Photovoltaic Panel, RESTful Web Service and the Android platform.

Keywords: Arduino, Renewable Energy, Smart Cities, Web Services, Computer Mobile Devices, Georeferencing.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS

1- API (Application Programming Interface)
2- App / Apps (Application)
3- CM (Centímetro)
4- BD (Banco de Dados)
5- IDE (Integrated Development Environment)
6- JSON (JavaScrip Object Notation)
7- MVC (Model View Control)
8- MM (Milímetro)
9- M/S (Metros por Segundo)
10- SO (Sistema Operacional)
11- OTA (Over The Air)
12- SOAP (Simple Object Access Protocol)
13- TIC´s (Tecnologia da Informação e Comunicação)
14- UDDI (Universal Description, Discovery and Integration)
15- UI (User Interface)
16- URL (Uniform Resource Location)
17- URI (Uniform Resource Identifier)
18- USB (Universal Serial Bus)
19- XML (Extensible Markup Language)
20- WS (Web Service)
21- WSDL (Web Service Description Language)

SUMÁRIO

1. INTRODUÇÃO

As cidades modernas em torno do mundo possuem a informação como sua principal aliada no âmbito da gestão eficiente e eficaz. O objetivo das Smart Cities (Cidades Inteligentes) não é apenas a construção de uma cidade, com processos bem definidos, posicionada em localização estratégica e com logística privilegiada, mas aliar esses benefícios ao crescimento econômico de forma sustentável.

Com o desenvolvimento de Centrais de Informação que ao receber dados e processa-los, possa tomar decisões e apresentar soluções. A partir dos relatórios emitidos por suas avaliações é possível identificar na concepção de um novo bairro diversos indicadores e fatores como: quais os recursos a serem investidos? Que tipo de impactos ambientais essa construção causará? Possíveis áreas de risco devido à localização geográfica? Analise do custo/beneficio, entre outros.

Ao disponibilizar aplicações que integram as centrais de informação aos dispositivos computacionais móveis, as cidades podem oferecer diversos tipos de serviços para a população, tais como: gestão inteligente de estacionamentos, rotas alternativas para mobilidade urbana, previsões climáticas, gestão das redes de água e energia e indicar áreas de riscos devido as inconstâncias climáticas como no caso de enchentes.

Os dispositivos computacionais móveis surgem como grande ferramenta de disseminação da informação gerada pelas Smart Cities. Devido às características que apresentam, permitem o fácil deslocamento do usuário no espaço que se encontra. A capacidade de alguns aparelhos após ser carregado ficarem dias sem depender de uma fonte de energia e conectividade via wifi, são fatores que permitem ao usuário realizar suas atividades e se locomover enquanto o dispositivo realiza seus serviços sem perda de conexão.

Segundo (OEHLMAN & BLANC, 2015) os serviços de geolocalização estão cada vez mais populares, devido a grande disseminação de smartphones e tablet´s. Os aplicativos acessíveis a estes dispositivos possuem diversas finalidades, busca ao caixa eletrônico e restaurante mais próximo, rota e frequência dos ônibus, encontrar e pedir um taxi com facilidade, entre outros.

Vale ressaltar que este projeto alia as tecnologias de sensores alimentados por painéis fotovoltaicos, armazena os dados coletadas pelo sensor em um web service que disponibiliza recursos onde informações estão disponíveis para serem consumidas em tempo real por uma aplicação mobile.

1.1. OBJETIVO

Este trabalho tem como objetivo principal criar um aplicativo que forneça informações em tempo real, das condições de trafego de vias públicas em dias chuvosos.

1.1.1. OBJETIVOS ESPECIFICOS

- Realizar estudo teórico sobre, desenvolvimento para a plataforma Arduino e plataforma Android.
- Estudar e compreender *Web Services* implementados com o paradigma REST.
- Desenvolver aplicação para *SmartPhones* com sistema operacional *Android*.
- Desenvolver tecnologia que identifique a altura da lamina d'água em determinadas áreas.
- Desenvolver *Web Service* que disponibiliza recursos como: rota informando a coordenada e altura da lamina d'água em determinadas vias públicas de trafego.
- Efetuar os testes com os *softwares* implementados.

1.2. ORGANIZAÇÃO DO DOCUMENTO

O trabalho esta estruturado da seguinte forma:

- Capitulo 1 – Introdução;
- Capitulo 2 – Revisão de Literatura;
- Capitulo 3 – Metodologia;
- Capitulo 4 – Desenvolvimento;
- Capitulo 5 – Conclusões;
- Capitulo 6 – Referências;
- Capitulo 7 – Anexos; e

- Capitulo 8 – Apêndice.

A Revisão Literária está dividida em quatro partes: Cidades Inteligentes, Energia Renovável, Estrutura de Hardware, Estrutura de Software e Dispositivos Computacionais Móveis. Neste capitulo serão abordados os principais conceitos relacionados ao trabalho.

O capitulo 3 aborda a metodologia e as ferramentas utilizadas para obter os resultados, desde a concepção da ideia da aplicação proposta, até a fase de testes.

O Desenvolvimento apresenta como foi à criação do *Hardware* e a implementação dos *softwares* necessários para que se consiga alcançar o objetivo, é a visão geral do trabalho.

Capitulo 5, mostra a conclusão do trabalho e perspectivas futuras de melhorias e aprimoramento da aplicação.

O capítulo referências apresenta as obras que foram utilizadas como fomentação de informação sem as quais não teria embasamento teórico para desenvolver tal trabalho.

No capitulo 7 são apresentadas contribuições de outros desenvolvedores e por fim o oitavo capitulo Apêndice esboça trechos de códigos desenvolvidos pelo autor desta obra com o intuito de alcançar os objetivos expostos na seção 1.1.1.

2. REVISÃO DE LITERATURA

Este capítulo aborda conceitos relevantes sobre os conteúdos base para o desenvolvimento do projeto: Cidades Inteligentes, Energia Renovável, Estrutura de Hardware, Estrutura de Software e Dispositivos Computacionais Móveis e Georeferenciamento.

2.1. CIDADES INTELIGENTES - SMART CITIES

As cidades a partir do século XVIII sofreram um inchaço com a migração da população do campo para a área urbana. O êxodo rural pegou as cidades de surpresa, logo elas cresceram de forma rápida e desordenada. Devido à fraca infraestrutura os governantes e a sociedade buscaram novos meios para resolver esses problemas (TAURION, 2014).

A população está acostumada, a apelar para o lado físico estrutural, novas obras tais como: ruas, avenidas, escolas entre outros artifícios, causando um crescimento desordenado para o futuro como pode ser observado na Figura 1: favelas, prédios abandonados, vias inutilizadas, áreas alagadas, entre outros. Logo essas manobras já não serão mais eficientes.

Fonte: Scott Wallace / World Bank, 2014.

Figura 1. Crescimento Desordenado

A construção e reconstrução de cidades modernas podem minimizar de forma inteligente muitos desses problemas sem a necessidade da expansão territorial. Segundo Santos et. al. (2013, p.9):

> "Cidades inteligentes são uma evolução natural de todas as facilidades que as cidades têm provido a seus cidadãos, como transporte, infraestrutura para água, luz, esgoto, telefonia etc. No entanto, por serem fortemente baseadas em tecnologias da informação e comunicação, as cidades inteligentes dependem bastante de serviços e aplicações que sejam úteis às pessoas."

As *Smart Cities* têm como foco auxiliar e proporcionar o bem estar ao ser humano. Viabilizar atividades que outrora demandavam muito esforço e tempo, garantir melhor qualidade de vida a sociedade e propiciar um crescimento econômico sustentável.

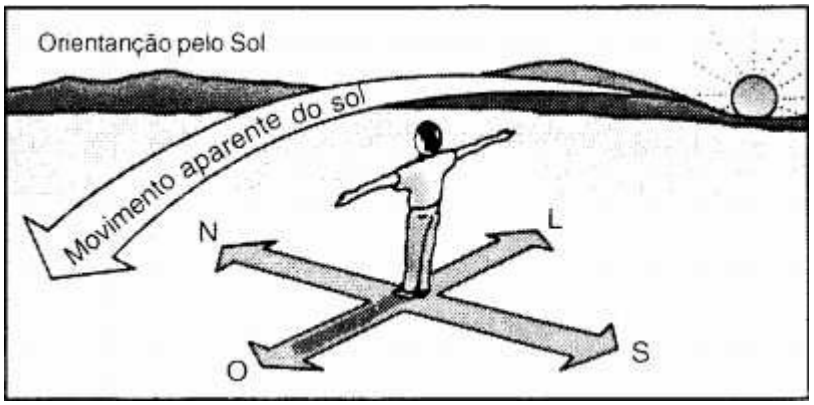
Para Selada et. al. (2012) existem vários pontos de vista explicativa para o termo cidades inteligente embora não exista um conceito único e universal aceito. Os projetos de *Smart Cities* demostram diversos graus de maturação em alguns casos elas são construídas a partir do zero (Masdar – Emirados Árabes Unidos e Songdo – Coreia do Sul) o que facilita (menos restrições) na concepção de centros modernos, nestas ocasiões encontra-se um cenário ideal, pois o projeto pode abordar vários pontos de vista que irão tornar-se soluções para o futuro. Já os projetos de reestruturação presentes na Europa e América do Norte visão sanar um problema devido à falta de planejamento ou melhorar algum serviço já existente, encontram como empecilhos as limitações espaciais e infraestrutura.

De acordo com Taurion, (2014) a infraestrutura é uma extensa e complexa rede de componentes, em que a falha ou falta de algum destes acarretará problemas gravíssimos, chegando a paralisar uma cidade, mas ao aliar infraestrutura as TICs, possíveis eventos indesejáveis podem ser mapeados e gerenciados de maneira pontual e ágil.

2.1.1. GEORREFERENCIAMENTO

Conforme pode ser observado na Figura 2, o primeiro artifício utilizado pelo homem para obter a localização dos objetos no plano terrestre foi o sol. Sabe-se que o sol nasce ao leste e se põe ao oeste do globo, a partir desta informação é possível adquirir mais duas orientações, logo se uma pessoa estende sua mão direita em

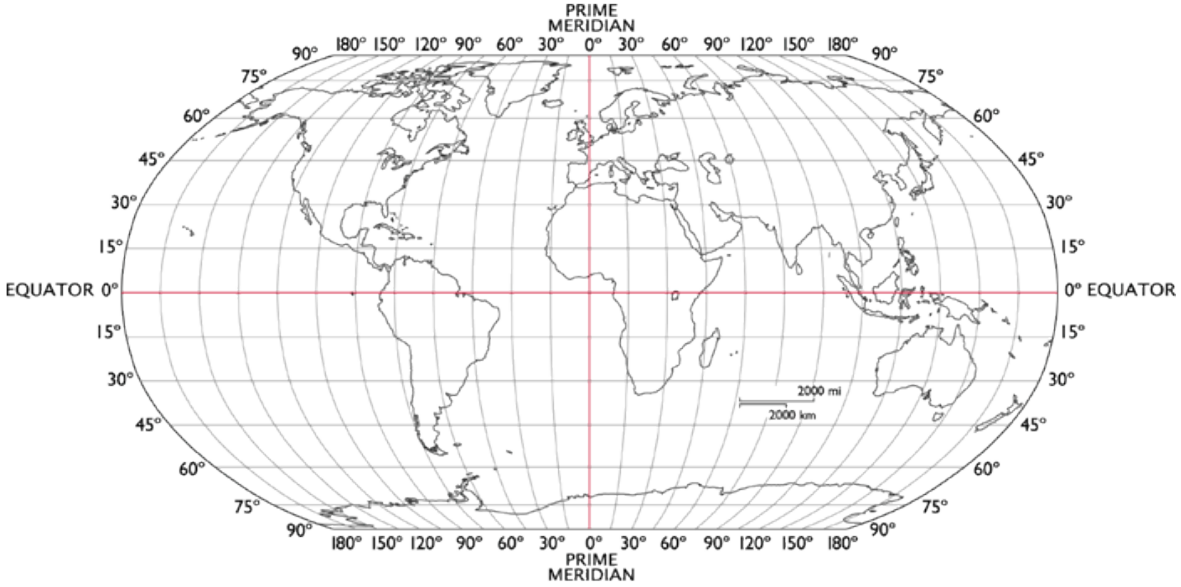
direção ao nascente e a esquerda para o poente, o norte ficará a frente e o sul na retaguarda. Por meados do século XIII surgiu à bússola, objeto em forma de circunferência com a representação da rosa dos ventos como plano de fundo e uma agulha imantada fixada em um eixo vertical em seu centro, por muitos anos e ainda hoje essa tecnologia tem sido utilizada (CARDOSO, 2006).



Fonte: Borba; Veigas, 2001.

Figura 2. Orientação pelo Sol.

Atualmente os sistemas de georreferenciamento utilizam a latitude e a longitude, a partir destas coordenadas pode-se determinar a posição exata de um ponto no mapa ou plano terrestre. O marco zero (0º) para a latitude é delimitado pela linha do EQUADOR, com variação de 0º a 90º para o Norte e de 0º a 90º para o Sul. Cada grau é dividido em 60 minutos e cada minuto em 60 segundos. Já a longitude possui como ponto inicial o meridiano de *GREENWICH*, pode ser ocidental ou oriental variando de 0º a 180º para cada. Um grau é dividido em 60 minutos e cada minuto em 60 segundos (CARDOSO, 2006). Logo mais abaixo a Figura 3 esboça um mapa com foco nas coordenadas de latitude e longitude.



Fonte: Worldatlas, 2015.

Figura 3. Globo Coordenadas Latitude e Longitude.

A necessidade de identificar determinado ponto geográfico com precisão fez com que surgisse o GPS (Sistema de Posicionamento Global). Desenvolvido pelo Departamento de Defesa dos Estados Unidos, tem como núcleo do sistema uma constelação de 24 satélites, para que dispositivos receptores possam calcular posição e altitude, utiliza-se no mínimo 4 satélites, porém com apenas 3 já é possível inferir a localização de dado objeto no mapa. Ao emitir um sinal de radiofrequência é calculado o tempo que o sinal leva para chegar ao receptor, cruza o resultado obtido dos quatro satélites e obtém-se a posição do dispositivo receptor, esse processo recebe o nome de trilateração (HUERTA et. al., 2005).

Por volta do ano de 2005 o departamento de defesa norte americano liberou o uso do GPS para fins publico civil (HUERTA et. al., 2005). Esta tecnologia exige de seu usuário conhecimentos de navegação, princípios básicos, saber aonde esta e para onde vai. Como a terra é um espaço tridimensional é necessário que se tenha três coordenas: latitude, longitude e altitude para identificar determinada posição real no globo terrestre (FASCIONI, 2013).

Com o avanço das tecnologias e o intuito das empresas deste ramo em prover serviços que auxiliam usuários com geolocalização, surgiram diversas aplicações otimizadas que aliam Mapas e GPS. Como exemplos bem sucedidos tem-se o *Google Maps* e o *OpenStreetMap*.

Os serviços oferecidos por Georreferencia abrangem um leque enorme de soluções para o cotidiano dos usuários, com o *Google Maps* é possível encontrar um restaurante, calcular a distância e o tempo gasto até ele; evitar engarrafamentos, entre outros serviços (GOOGLE, 2015A).

2.1.2. AUTOMAÇÃO COMPUTACIONAL

O século XXI trouxe a tona o jargão *Smart*, que traduzido do inglês para o português significa “inteligente”. A necessidade de diminuir os custos e aumentar a qualidade de atendimento ao cliente faz empresários e pessoas comuns, acreditar que podem tornar uma atividade arcaica em algo inovador através da tecnologia ao investir na inteligência computacional.

Para Andrade (2014) “inovação – é a invenção utilizada no mercado. É a invenção colocada em prática e aderida em seu mercado. Se isso não acontece, é apenas e tão somente uma invenção”.

O empreendedor possui como pilares para o desenvolvimento de um produto de sucesso e inovador as boas praticas de desenvolvimento, plano de negócio e conhecimento do mercado (DORNELAS, 2005).

O mundo globalizado exige que as empresas ofereçam qualidade de atendimento ao cliente e minimizem os gastos. Para prestar um serviço satisfatório e de baixo custo, empresas começaram investir nas TICs para oferecer um diferencial perante os concorrentes e maximizar seus lucros. É bem verdade que no século XXI ter um empreendimento fortemente baseado em tecnologias avançadas não é uma questão de diferencial, mas sim de sobrevivência Mendes (apud EAN Brasil, 2005).

Idealizada para auxiliar os seres humanos, nas residências, no trabalho, nas ruas, entre outras circunstancias, a automação está baseada em três grandes áreas da engenharia, são elas (MARTINS, 2012):

- Informática responsável por centralizar as bases de informação dos empreendimentos, integrar os serviços de forma transparente as filiais e realizar a gestão de processos.
- Mecânica desenvolvimento e manutenção de maquinas que transformam matéria prima em bens de consumo.

- Engenharia Elétrica com motores, braços e seus acionamentos garante a manutenção de uma produção padronizada, sempre com as mesmas características: menor custo, maior quantidade, menor tempo e maior qualidade.

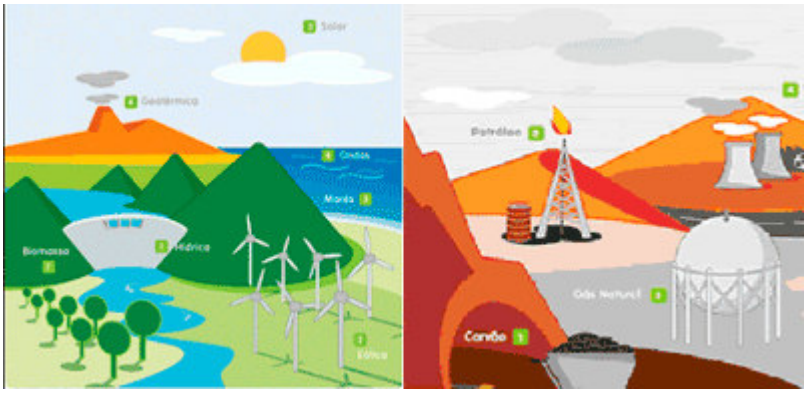
Para implantar a automação computacional de forma bem sucedida é necessário que o *hardware* ocupe espaço reduzido e apresente baixo consumo de energia (MARTINS, 2012).

Seguindo os conceitos adotados para gestão eficiente de energia limpa pelas cidades inteligentes, a próxima seção aborda o assunto fontes de energias renováveis.

2.1.3. ENERGIA RENOVÁVEL

Sobre os métodos tradicionais de geração de energia incide uma gama de problemas, a saber, devastação ambiental, poluição, dependência de combustíveis, entre outros (RÜTHER, 2004). Para a construção de uma hidrelétrica exige-se o comprometimento de grandes áreas territoriais, o que afeta diretamente a vida dos ribeirinhos e em alguns casos até a extinção de determinadas espécies da fauna e flora.

Segundo Brasil (2014) as fontes de energia derivadas do carbono (c) são as mais utilizadas na atualidade, como petróleo, carvão e gás natural, possuem como agravantes ambientais o fato de serem obtidos de origem fóssil (não renovável). A queima destas formas de combustíveis está associada a graves prejuízos ao meio ambiente como efeito estufa, aquecimento global, instabilidade climática e chuva ácida.



Fonte: Geo AMB e Rui Soares, 2014.

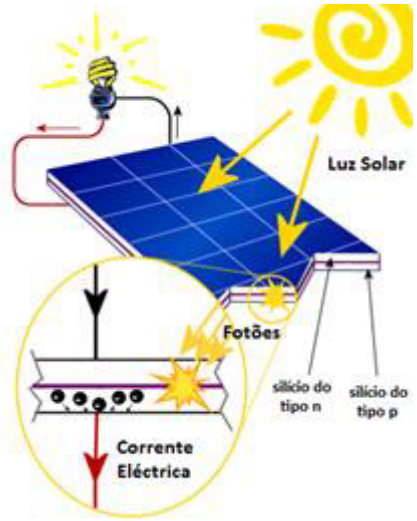
Figura 4. Fontes Renováveis X Fontes Não Renováveis.

A Figura 4 logo acima, apresenta as principais fontes de energia da atualidade. Segundo ANELL (2014) em suma maioria as fontes de energia: biomassa, eólica, combustíveis fosseis, hidráulica e energia dos oceanos, são provenientes da energia solar. Dentre estes métodos as tecnologias que se apresentam com grande destaque são aquecimento de água e a produção de energia fotovoltaica. Estas fontes de energia estão diretamente ligadas ao efeito da radiação solar devido à inclinação do eixo imaginário em que a Terra realiza os movimentos de rotação e translação, a incidência da radiação é dependente das condições atmosféricas, latitude local e posição no tempo, fatores que não expressam fortes perdas de radiação solar no território brasileiro.

Observa-se que a capital brasileira mais ao Sul tem duração solar/dia variada entre 10 e 13 horas. Algumas capitais relativamente próximas à linha do Equador apresentam números em torno de 16 a 18 horas solar/dia (ANELL, 2014). A intensidade de calor que chega a superfície da terra ao meio dia é de 1000 W/m^2, fator que torna o Brasil país de cenário ideal para produção de energia oriunda de fontes renováveis (Sol) (RÜTHER, 2004).

2.1.3.1. PAINEL FOTOVOLTAICO

Painel Fotovoltaico é um gerador de energia limpa e renovável, mediante comparações com outros métodos utilizados para produção de energia elétrica. Este modelo destaca-se pela eficácia e baixo ou nenhum dano empregado ao meio ambiente no processo de produção. Para (GREEN et al., 2000) citado por ANEEL(2014), a exposição de alguns materiais a luz solar provoca excitação em seus elétrons originando o efeito fotovoltaico. Os materiais mais adequados para a conversão da radiação solar em energia elétrica são chamados de células solares ou fotovoltaicas, entre as quais se destaca o silício cristalino (c-Si). Esta forma de produção de energia é apresentada na Figura 5.



Fonte: Karloto ,2014.

Figura 5. Painel Fotovoltaico.

O desenvolvimento de sistemas híbridos pode ser uma solução para os dias de pouca incidência solar ou para períodos noturnos. Os sistemas que integram painéis solares e banco de baterias têm grande utilidade em lugares isolados dos grandes centros que o sistema de energia pública ainda não contempla. A integração com a rede pública de energia torna-se mais viável nas áreas urbanas, pois não exigem os gastos e manutenção necessários para se criar um sistema com bancos de baterias.

Nos últimos anos os painéis fotovoltaicos veem sendo integrados a construção civil, tendo dupla função geradora de energia e componente arquitetônico (RÜTHER, 2004).

O custo de produção dos painéis c-SI é considerado por muitos críticos uma tecnologia cara e inconveniente em sistemas de larga escala. Leva-se em consideração que o tempo médio esperado para que o módulo gere energia equivalente à utilizada em sua produção em certas ocasiões, seja superior a dois (2) anos (RÜTHER, 2004).

Mas em contrapartida outros analistas levam em consideração os painéis fotovoltaicos que são projetados para serem utilizados em ambientes externos, sob agentes climáticos tais como chuva e sol, tendo uma vida útil nestas condições em torno de 30 anos, o que torna viável sua utilização ao relacionar custo, impacto ambiental e benefícios.

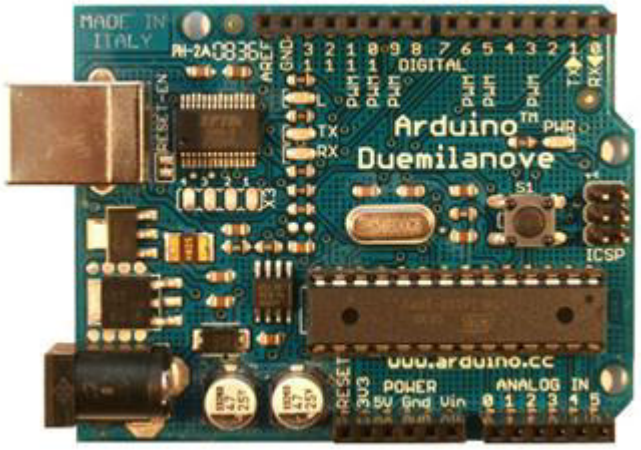
2.2. ESTRUTURA DE HARDWARE

Segundo o dicionário Michaelis (2014A) é considerado como *hardware* o conjunto de componentes de computador ou seus periféricos é a parte física. As próximas sucessões abordarão as temáticas: Arduino que é o *hardware* central e Sensores que são os periféricos da estrutura computacional.

2.2.1. ARDUINO

O microcontrolador (placa) Arduino permite a partir de um computador desenvolver programas que irão transferir informações para a placa através de USBs, *Bluotooth, Wireless* e/ou Infravermelho. Estas instruções irão controlar os componentes conectados a placa, tais como: motores, braços mecânicos, entre outros componentes. Tornou-se muito utilizada nas escolas e faculdades para iniciação de alunos e acadêmicos nas disciplinas de Robótica e Programação (RIOS, 2012).

A Arduino possui diversos modelos com características peculiares, o que permite a escolha da placa ideal levando em consideração as especificações para cada projeto. A Arduino Mega 2560 se destaca pela considerável quantidade de portas, a Arduino Nano que é uma pequena versão da placa semelhante a Arduino uno se distinguindo pelas 2 entradas analógicas a mais e um jumper de +5V AREF (RIOS, 2012), Arduino Uno e *Duemilanove* é apresentada na Figura 6, logo abaixo.



Fonte: Arduino, 2014.

Figura 6. Placa Arduino Duemilanove.

Segundo documentação oficial Arduinno.cc (2014) e conforme pode ser observado na Figura 6, *Arduino Duemilanove* é uma placa micro controlador baseado no Atmega328. Possui quatorze (14) entradas/saídas digitais e seis (6) entradas analógicas, um cristal oscilador de dezesseis (16) MHZ, conexão USB, uma entrada para fonte, *soquets* para ICPS e um botão *reset*. Contem todo o suporte para usar o microcontrolador, simplesmente conecte-a a um computador com cabo USB ou ligue a com uma fonte AC/DC ou bateria.

A plataforma de desenvolvimento foi baseada nas linguagens C/C++. Os programas escritos para a Arduino são chamados *sketches*, necessitam de duas funções primordiais "*setup()*" e o "*loop()*" e são salvos com a extensão "*.ino". O ambiente de desenvolvimento permite ao usuário verificar se o código possui erros, compilar o código e fazer o upload para o Arduino, criar um novo *Sketche*, exibir menu com *sketches* existentes no seu *sketchesbook*, salvar uma *Sketche* e acesso ao monitor serial.

Apresenta como vantagens, linguagem simples e de fácil manipulação para usuários iniciantes, baixo preço, *software* livre e disponível para várias plataformas *Microsoft Windows, Linux* e *Mac OS X*. A alimentação é obtida através de uma fonte externa ou conexão USB com computadores (RIOS, 2012).

2.2.2. SENSORES

Nos primórdios da eletrônica, variáveis ambientais e incompatibilidade de componentes tinham grande interferência no preço e resultado dos sensores. Com o avanço da tecnologia, os sistemas de sensoriamento tornam-se mais sofisticados, ao desenvolver sensores insensíveis aos obstáculos impostos pelo meio de exposição, aumento da sensibilidade e frequência de resposta torna preciso o resultado, outra característica que populariza estes componentes cada vez mais é a capacidade de integra-los a diferentes tipos de aplicações (ADAMOWSKI, 2015). A Figura 7 exibe diversos modelos de sensores.

Fonte: SensoricsFabio, 2015.

Figura 7. Tipos de Sensores.

Sensores mecânicos são aqueles que detectam movimento e presença utilizando recursos mecânicos como, chaves. Sua ativação se dá quando ocorre o corte ou ativação de um circuito no modo liga/desliga, estes sensores possuem como desvantagem a fragilidade sendo sujeitos a quebra e desgaste e o repique que pode inverter a ação e enviar um sinal de forma errônea (WENDLING, 2010).

Fotoelétricos são os sensores que trabalham com luz, podem ser empregados em diversas ocasiões sendo divididos em: fototransistor, fotodiodo, fotocélula e foto-resistor. Para Wendling (2010) o uso de sensores fotoelétricos exerce grande vantagem em relação ao uso de sensores mecânicos, pois não são tão frágeis e tem o tempo de resposta muito mais rápido.

Sensores capacitores geram campos elétricos, ao detectar alterações no campo elétrico devido à presença de dado objeto no ambiente, o oscilador ativa o circuito de saída, comutando seu estado (WENDLING, 2010).

Indutores são compostos basicamente por uma bobina em torno de um núcleo. Esse tipo de sensor emite um sinal que detecta objetos metálicos que atravessam seu campo magnético, ao identificar a interferência o sensor comuta (WENDLING, 2010).

Os sensores ultrassônicos vem ganhando espaço na indústria, pois aliam precisão e baixo custo. Em aplicações que buscam mensurar distancias se destacam os sensores baseados no método pulso-eco, estes sensores calculam a distância entre dois pontos através do intervalo de tempo gasto pra a onda sair do ponto A(emissor), ir ao ponto B(receptor) e retornar ao ponto A(emissor) (BASTOS et.al, 2015).

$$D = ((T_{(a->b)} + T_{(b->a)})*V) / 2$$

Fonte: Elaborado pelo Autro, 2015.

Segundo Wendling (2010) há uma gama de variáveis e características que devem ser consideradas na hora da escolha pelo sensor mais adequado para cada ambiente. Tempo de resposta, preço e precisão do resultado apresentado, são fatores que devem ser observados para apontar qual sensor terá melhor desempenho.

2.3. ESTRUTURAS DE SOFTWARE

Software é a parte lógica, qualquer programa ou conjunto de programas que são interpretados por um dispositivo computacional e fornece instruções ao *hardware* (MICHAELIS, 2014B). Estão divididos em três grupos: *software* de sistema que é o Sistema Operacional (SO), responsável pela interação homem/máquina, o *Software* de Programação constituído pelo conjunto de ferramentas que um desenvolvedor possui para implementar sistemas computacionais e os *Softwares* de Aplicação são programas que permitem os usuário desempenhar suas atividades diárias, como por exemplo redigir um texto em um processador de texto.

2.3.1. APLICAÇÕES WEB

Os sistemas *web* tornaram se populares na década de 90, sendo que com o passar do tempo a internet foi se tornando robusta e saiu do âmbito acadêmico para ganhar novos patamares. Segundo Shepherd (2007) a *World Wide Web* criada em 1992 é a mais complexa rede de comunicação já criada pelo homem, está a cada dia mais intrínseca em seu cotidiano, desde campanhas publicitarias, ambientes

educacionais, vendas via *e-comerce*, até poderosos sistemas de gestão comercial e relacionamento pessoal.

Sequencia de passos para uma comunicação via internet (BEZERRA, 2013):

1- O usuário abre o navegador, e digita o endereço de uma pagina qualquer, ex: www.google.com;
2- A requisição é transmitida da camada de aplicação para a camada física;
3- Ao chegar ao servidor do provedor, a requisição sobe todas as camadas até a camada responsável por processar a informação ou repassar para outra maquina da internet;
4- A máquina a qual foi endereçada a requisição processa a solicitação e envia no caminho inverso os dados do objeto solicitado;
5- O *browser* (cliente) recebe os dados da página solicitada e exibe as informações.

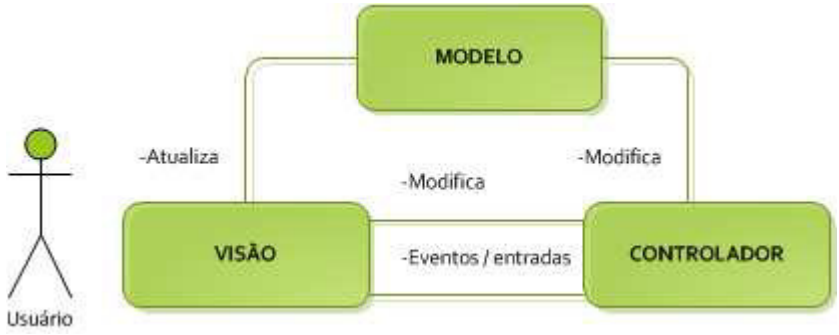
A comunicação na *web*, usuário/servidor é realizada pelo protocolo HTTP (*HyperText Transfer Protocol)*. Mas para que aconteça essa transferência de dados este protocolo precisa estar associado a mais dois (2) protocolos, são eles: TCP(*Transmition Control Protocol*) e IP (*Internet Protocol*) juntos formam o modelo TCP/IP que fornecem um conjunto de serviços bem definidos para o protocolo da camada superior (BEZERRA, 2013).

O protocolo HTTP é orientado a *resquest/response* onde o cliente faz uma requisição ao servidor que retorna uma resposta, os dois principais comandos do protocolo HTTP são *Get* e *Post*:

- *GET*: a requisição é feita via URL, possui limitação de dados a ser enviada, a "*string*" não pode exceder o tamanho de 255 caracteres.
- *POST*: os dados são enviados no corpo da requisição HTTP e não possui limitação de tamanho da mensagem.

Segundo Bond (2003) o modelo de aplicações baseadas na metodologia de três camadas, apresentado na Figura 8 tornou-se a arquitetura mais utilizada para sistemas *Web*. A possibilidade de separar as camadas de Aplicação, Logica de Negocio e Logica de Acesso a Dados, proporciona ao sistema maior flexibilidade, ou

seja, cada camada pode ser alterada sem possuir um relacionamento de dependência com as demais. Com este modelo o desenvolvedor não precisa ser perito nos outros níveis da aplicação, basta apenas conhecimento especifico da área em que trabalha.



Fonte: Mauricio Reale, 2014.

Figura 8. Modelo Três Camadas.

Para Camacho Júnior (2008) o desenvolvimento das aplicações deve começar das camadas mais internas para as mais externas, dessa forma orienta-se começar pela Camada de Acesso a Dados (*Data Access Layer-DAL*). Esta camada é responsável por manipular os métodos referentes às tabelas existentes em um projeto. A segunda camada a ser implementada é a Regras de Negocio, aqui o desenvolvedor define como o negocio funciona, estratégia de como tratará leis, regulamentações, objetivos e compromissos éticos sociais. A terceira camada é Apresentação (*User Interface* - UI), ela define a forma como os dados serão apresentados ao usuário.

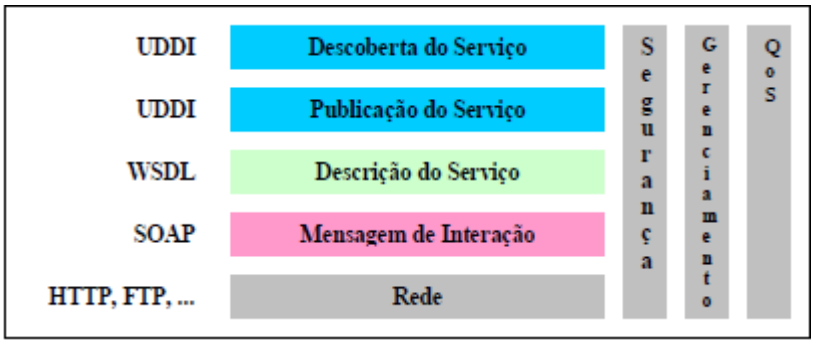
Entre as tecnologias para desenvolvimento *web* destacam-se as que são implementadas em PHP, *.NET* e linguagem de programação *Java*. A próxima seção abordara o assunto serviços disponibilizados na *web*.

2.3.2. WEB SERVICES

Os *web services* são definidos como uma solução para integrar aplicações de diferentes tecnológicas. Possibilita a comunicação de forma transparente e compatível entre as novas tecnologias com sistemas legados, executadas em arquiteturas que se tornaram obsoletas com o surgimento de novas aplicações. Muito utilizado na integração das aplicações corporativas um *web service* prove dados e serviços para outras aplicações ou serviços (FACUNTE, 2015).

O desenvolvimento de *web services* atualmente segue dois paradigmas principais SOAP e REST.

Segundo Martins (2007) *web services* embasados na arquitetura SOAP utilizam os protocolos *Hypertext Transfer Protocol* (HTTP), *Simple Object Access Protocol* (SOAP), *Web Service Description Language* (WSDL) e Universal Description, Descovery and Integration (UDDI) que juntos fornecem um pacote de padrões bem definidos que garantem uma comunicação com qualidade de serviços.



Fonte: Martins, 2007.

Figura 9. Camadas conceituais de Web Service SOAP.

Conforme se verifica na Figura 9, as literaturas descrevem *web service* baseado em SOAP sendo composto por cinco camadas conceituais. A de mais baixo nível é a de **Rede** onde se opera o protocolo HTTP. Um nível a cima fica a **Mensagem de Integração** neste instante o protocolo SOAP define o formato de transmissão dos *request/response* em linguagem XML (*eXtensive Markup Language*). No terceiro nível está a **Descrição do Serviço** com o protocolo WSDL que define os tipos de dados, operações disponíveis e informações sobre o protocolo de transporte que será utilizado. O quarto nível é a camada de **Publicação do Serviço,** e, por fim, o quinto **Descoberta do Serviço,** sobre os dois últimos níveis atua o UDDI que é responsável por descrever e integrar os *web services* (MARTINS, 2007).

A arquitetura *REpresentational State Transfer* (REST) foi proposta em uma tese de doutorado apresentada por Roy Fielding, coautor do protocolo mais utilizado na atualidade o HTTP. Na proposta desenvolveu o paradigma REST fortemente baseado no protocolo HTTP que é utilizado para transportar informação pela rede mundial de computadores, os *web services* que utilizam essa arquitetura são

considerados mais ágeis e leves, fator que os tornam primordiais para desenvolvimento de aplicações *mobile*, cenário em que os dispositivos apresentam recursos de *software* e *hardware* limitados (SAUDATE, 2015).

Para Saudate (2015) os *web services* que são implementados sobre o padrão REST podem utilizar como forma de representação das requisições o XML ou JSON, porém os que utilizam *JavaScript Object Notation* (JSON) apresentam maior agilidade por ser menos verbosa. Essa linguagem de marcação foi desenvolvida como uma vertente para o XML. JSON apresenta vantagem sobre o seu concorrente por ter tamanho reduzido fator que lhe proporciona melhor desempenho onde a largura de banda é modesta.

Fonte: ToolsWeb, 2015.

Figura 10. Requisição a Web Service.

A Figura 10 acima detalha como é feita a requisição de dados a um *web service*. O solicitante/cliente faz uma requisição, o *web service* realiza a operação e retorna os dados via XML para a aplicação, que recebe as informações e as disponibiliza para o cliente.

Os *Web Services* apresentam diversas vantagens tais como:

- Serviços embasados nos padrões predefinidos pela *Wide World Web Consortium* (W3C). Essa organização é responsável por padronizar e

regulamentar as boas pratica de desenvolvimento para aplicações *web*, a fim de alcançar um crescimento de longo prazo (W3C.BRASIL, 2015).

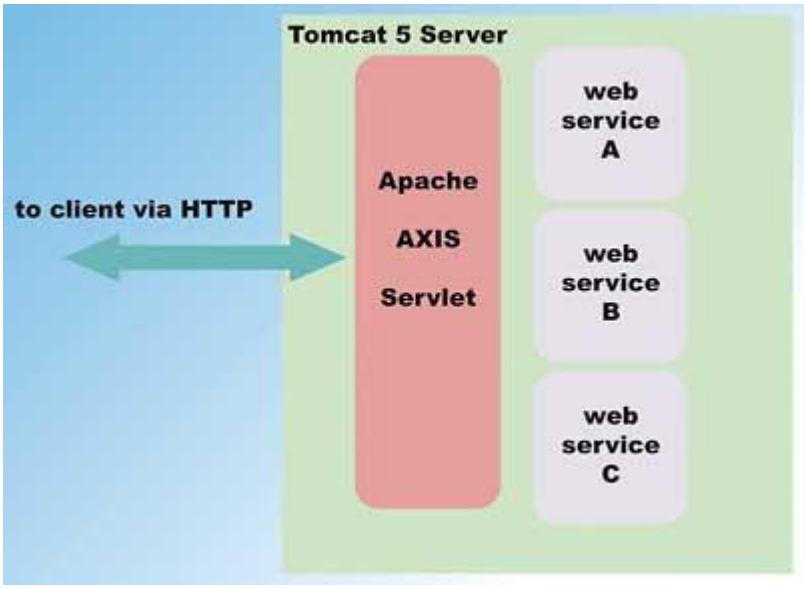
- Abdica de pré-configuração em *firewalls*, os dados via XML/JSON são encaminhadas a porta 80 de um servidor HTTP, que embora seja constantemente monitorada permite o trafego de informações (FACUNTE, 2015).
- Não há intervenção humana, uma vez que o serviço foi publicado e opera de forma estável, podem-se realizar milhares de operações, simultâneas e similares sem intervenção do desenvolvedor (MARTINS, 2007).
- Reutilização de componentes, uma empresa de e-comerce quer oferecer aos clientes o rastreamento de um pedido, para isso, ela ira apenas integrar os serviços oferecidos pelo *web service* dos Correios a sua aplicação. A capacidade de invocar um serviço já escrito é considera como reaproveitamento de módulos (SCHIMITZ, 2013).
- Independência de plataforma e linguagem de desenvolvimento, um aplicativo desenvolvido em *.Net* executado na plataforma *Microsoft Windows* pode interagir com uma aplicação desenvolvida em *Java* para um servidor *Linux* (SCHIMITZ, 2013).

Com o intuito de facilitar a desenvolvimento de *web services* surgiram *frameworks* para implementação destas aplicações. As seções que seguem apresentam estes *frameworks*.

2.3.2.1. APACHE AXIS

O *Apache Axis* é um *framework open source* para desenvolvimento de *web services*. É uma implementação do SOAP submetido ao W3C na busca pela interoperabilidade de sistemas (AXIS.APACHE, 2015). O *apache axis2* é a terceira geração do *axis*, oferece um motor de *web services* mais eficiente, mais modular e mais orientada a XML que seus antecessores.

Conforme pode ser observado na Figura 11, é apresentado como o *servlet axis* transforma o servidor *Tomcat* em um motor de *web services* de alta performance. A comunicação é feita via HTTP (SOAP/JAX-RPC), a solicitação é processada pelo *servlet* e despachada para os *web services* A, B e C.



Fonte: Singli, 2015.

Figura 11. Tomcat com web service engine

As principais vantagens de se utilizar o *axis* são: facilidade para criara um *web service* e realizar o *deploy* de um serviço, constante correção de problemas descobertos ao longo do tempo, comunidade de usuários que disponibiliza um rico acervo com questionamentos e duvidas sobre o *Apache Axis* (AXIS.APACHE, 2015).

2.3.2.2. RESTful

RESTEful são *web services* implementados com base no padrão REST. Segundo (SCHIMITZ, 2013) para que haja entendimento desta tecnologia é fundamental que se estude o conceito de REST.

Poderosa ferramenta na troca de informações entre clientes e servidores, o *RESTful* tem como seu conceito mais básico *request/response*. As requisições são feitas ao servidor via HTTP, que por sua vez retorna ao cliente a resposta no formato Texto Puro, XML ou JSON, costuma-se utilizar a terceira forma, pois ela é mais leve que as outras (SCHIMITZ, 2013).

Web services RESTful disponibilizam seus recursos através das URI's e utiliza os métodos do HTTP para consumi-los. A entrada pela URL é definida pelos seguintes métodos (SAUDATE, 2015):

- *GET* utilizado para recuperar dados identificados pela URL.

- *POST* cria um novo recurso.
- *PUT* utilizado para atualizar recursos.
- *DELETE* utilizado para apagar um recurso.

Cada método possui características particulares que tem importância de acordo com a necessidade do usuário.

```
{“endereco”: [
        {
        “rua”: Seis,
        “cidade”: Palmas,
        “estado”: TO
        }
]};
```

Fonte: Elaborado pelo Autor, 2015.

Quadro 1. Exemplo de Dados JSON.

O Quadro 1 apresenta um exemplo de dados JSON o objeto “endereco”. Para as atribuições ocorrerem com sucesso, a sintaxe segue uma estrutura bem definida, os caracteres chaves, ponto e virgula ({ };) delimitam o escopo do objeto, os colchetes ([ ]) indicam que se trata de um array, as aspas duplas sucedidas por dois pontos (“ ”:) identificam o atributo chave e após os dois pontos vem o valor atribuído a chave. Cada chave com seu valor é separada das demais pelo caractere vírgula (,).

A utilização de JSON torna as respostas do *web service* mais rápidas (SCHIMITZ, 2013), essa característica faz com que os dispositivos computacionais móveis despontem como grandes consumidores de aplicações *RESTful*.

2.4. DISPOSITIVOS COMPUTACIONAIS MÓVEIS

Mobilidade é o jargão utilizado para definir os dispositivos que oferecem aos usuários a capacidade de fácil deslocamento. Para que um dispositivo computacional se caracterize como móvel, ele deve ser portável, funcional, oferecer conectividade com a rede mundial de computadores e/ou outros dispositivos e possuir independência mínima de energia (LEE et al., 2005).

A flexibilidade oferecida por estes dispositivos deve se as suas características (LEE et al., 2005):

- **Portabilidade** é uma característica que está relacionada ao tamanho e peso dos dispositivos, que sofreram diversas alterações ao longo do tempo, pois os primeiros computadores e telefones portáveis erão de tamanho significativamente grande se comparados aos *notbooks* e smartphones da atualidade. Este termo refere-se a facilidade de transportar o dispositivo pelo meio em que o usuário se encontra.
- **Usabilidade** está diretamente ligada às características do usuário, características do ambiente e características de dispositivo. A primeira está relacionada ao grau de conhecimento e destreza do usuário em relação ao dispositivo. A segunda refere-se as condições do meio em que o usuário se encontra como calor, umidade, frio e seca. A terceira, os dispositivos oferecem características particulares como tempo de inicialização, integridade de dados, interface com usuário, robustez e resistência, que têm sua importância para cada tipo de usuário.
- **Funcionalidade** está intrínseca ao tipo de aplicação que o dispositivo oferece a cada usuário, são divididas em duas categorias: Independentes são as aplicações que não necessitam de interação com outro usuário ou sistema, e Dependentes são aquelas que precisão de conexão com algum tipo de serviço oferecido por terceiros para baixar e carregar informações.
- **Conectividade** dá-se por três modos, Nunca são as aplicações independentes, conexão Parcial com sistemas *back-end*, como serviço de previsão do tempo, e Sempre aplicações que estão o tempo todo conectadas a um sistema *back-end*, como o serviço de GPS.

Na atualidade os dispositivos computacionais móveis estão divididos em 4 grupos de destaque:

- *Wearables* são dispositivos computacionais desenvolvidos com o objetivo de agregar valor tecnológico aos acessórios vestíveis, como: pulseiras relógios e óculos (LANDIM, 2015). Entre os quais se destacam os *SmartWatch* e os *SmartGlasses*.

- *Tablet PCs* é um dispositivo computacional móvel que oferece a seus usuários a capacidade de escrever diretamente sobre o *display* e proporciona as mesmas funcionalidades de *PCs Desktop.* Eles surgiram no ano de 1989 com o lançamento do *GRidPad*, todos os anos são apresentados exemplares cada vez mais poderosos e sofisticados (LEE, 2015).
- *PCs Laptop* apresenta todas as características de um *PC Desktop* além de proporcionar a seus usuários a capacidade de deslocamento no ambiente em que se encontra. Para (LEE, 2015) provavelmente é o dispositivo móvel com as maiores dimensões aceitável pra aparelhos portáveis.
- *SmartPhones* a necessidade de estar conectado com a rede fez com que os telefones evoluíssem, ao combinar características primarias com outros dispositivos mais avançados deu-se origem a *smartphones* com alta capacidade de processamento e SOs surpreendentes (LEE, 2015).

Atualmente os usuários encontram no mercado diversos tipos e modelo de dispositivos móveis, o que facilita na escolha do melhor aparelho que se adeque as suas necessidades. Eles variam de tamanho, sistema operacional e capacidade de processamento, características que influenciam diretamente em seus preços, quanto maior a capacidade de processar informações e disponibiliza-las no *display* com alta qualidade mais elevado é o valor do produto.

O sistema operacional é responsável por prover as características logicas do dispositivo. A seguir uma breve introdução sobre os três SOs quem mais se destacam no mercado dos dispositivos computacionais *mobile*, *Android, IOS* e *Windows Phone*.

2.4.1. ANDROID

Android é um projeto *open source*, desenvolvido pelo *Android Inc* e adquirido pela *Google* em 2005. Atualmente é mantido por um consorcio de empresas *Google, Samsung, Intel, Sony*, entre outras. Essas gigantes da computação buscam em parceria desenvolver uma plataforma versátil, solida e com qualidade. O *Android Open Source Project* (AOSP) possui como foco os dispositivos moveis, S*martphones, Tablet´s* e *SmartWatch*. O android é distribuído sobre a licença Apache 2.0, e o sistema operacional foi concebido sobre o *Kernel* do *Linux*, fatores

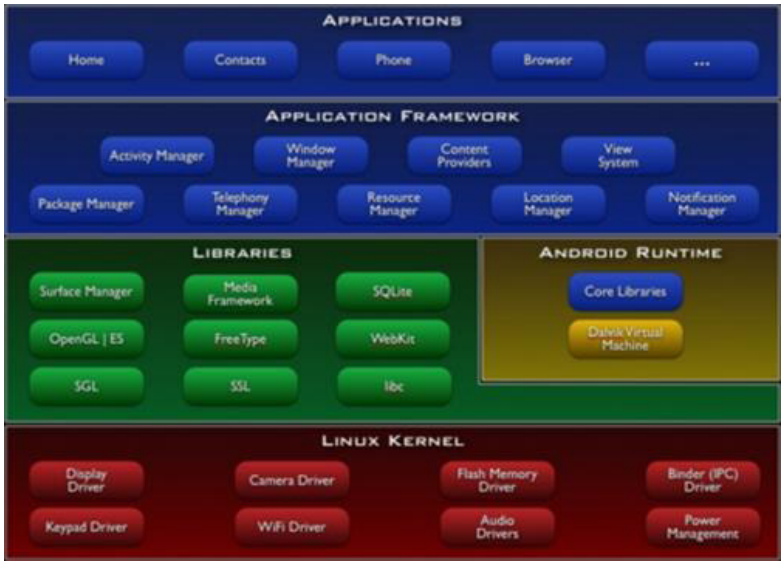
que possibilitam desenvolvedores em torno do mundo contribuir para o avanço da plataforma (AMORIM, 2011).

Android tornou-se o SO para mobile com a maior aceitação de mercado superando grandes concorrentes como IOS e *Windows Phone*. De acordo com os dados apresentados pelo IDC (*International Data Corporation*) no ano de 2014 85% dos dispositivos moveis fazem uso desta plataforma.

A *Google* disponibiliza os aplicativos no *Google Play*, que é a loja oficial na qual estão centralizadas todas as aplicações desenvolvidas para o *Android*. A loja oferece os *Apps* e Jogos distribuídos em duas categorias, gratuitos e pagos; assim seus clientes podem acessar a loja e fazer o download da aplicação que deseja onde quer que esteja (ANDROID, 2015).

2.4.1.1 ARQUITETURA ANDROID

Para o desenvolvedor A*ndroid* é fundamental conhecer a arquitetura da plataforma, compreender porque o android é constituído por uma pilha de *softwares* distribuídos em quatro camadas bem divididas, e quais as funcionalidades estão intrínsecas em suas características. Camadas: *Linux Kernel, Libraries and Android RunTime, Application Framework e Applications* (FREITAS, 2012).



Fonte: Euzébio, 2015.

Figura 12. Arquitetura Android.

Conforme se verifica na Figura 12, a camada de mais baixo nível é a *Linux Kernel.* Ela é a interface do SO responsável por prover os *drivers* necessários para

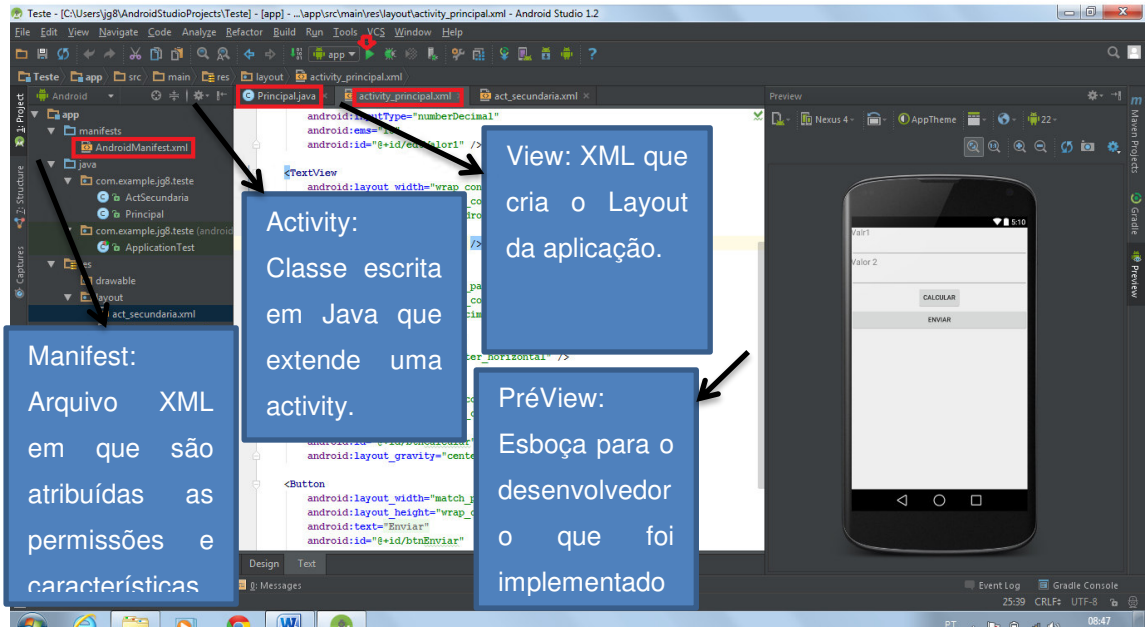
que *hardware* e *software* se comuniquem. Habilita de forma transparente os recursos mínimos necessários para que o dispositivo possa funcionar, como gerencia de memoria e gerencia de processos, configuração de *display*, rede *wifi*, áudio, entre outros. Em sua trajetória o *android* tem utilizado vários *kernels*, o primeiro foi o 2.6.25 e atualmente conta com a versão 3.4.

Na segunda camada estão presentes as *Libraries* e o *Android RunTime*. As bibliotecas presentes nessa camada são escritas em C ou C++ e executam diretamente no linux. Nesta camada também se encontram a maquina virtual *Dalvik* e o *Core Java Libraries* que compõem o *Android RunTime*.

A terceira camada compreende as APIs do *Android*, ela é responsável por abstrair o acesso das bibliotecas presentes na camada inferior nível 2. E na quarta camada encontram-se as aplicações nativas e os *Apps* fim que são aqueles desenvolvidas diretamente para os usuários, sejam eles para entretenimento ou negocio.

2.4.1.2 ESTRUTURA APLICAÇÃO ANDROID

Ao desenvolver aplicações para a plataforma *android* é necessária à utilização de um *Software Development Kit* (SDK), pode-se utilizar a IDE do *Eclipse* ou o *Android Studio*, a primeira atualmente passa por um processo de descontinuação, fator que leva os desenvolvedores migrarem para a segunda opção (ANDROID, 2015). O ambiente de desenvolvimento do *android studio* pode ser visualizado na Figura 13 logo abaixo.



Fonte: Elaborado pelo Autor, 2015.

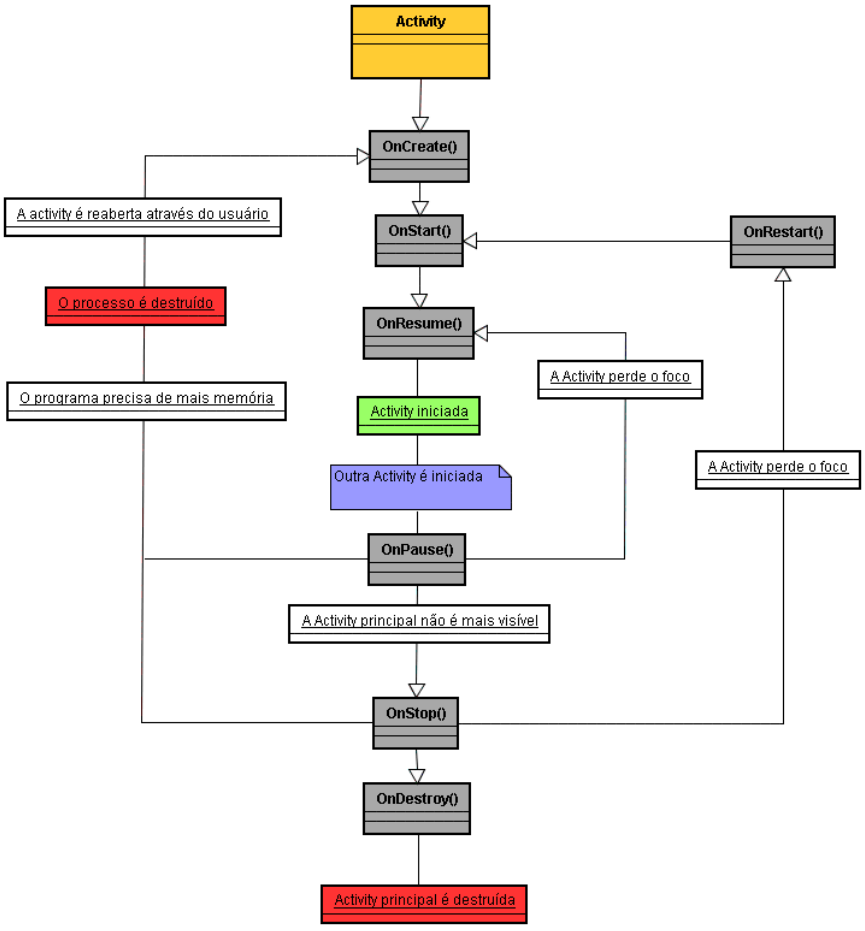
Figura 13. Android Studio.

Toda aplicação *android* é composta pelos seguintes componentes, *Services, Content Providers*, *Broadcast Receivers e Activities* (FREITAS, 2012):

- *Services* é o componente do *Android* que prover a capacidade de executar tarefas em segundo plano. Exemplo de *services,* quando o usuário aciona o *play* de musicas e logo em seguida abre o *App* do *Facebook* a musica continuará tocando até que ele volte ao aplicativo de musica e a pare.
- *Content Providers*, por questões de segurança o *Android* não permite que as aplicações acessem o banco de dados uma das outras, para isso ele disponibiliza o provedor de conteúdos, assim a aplicação pode compartilhar informações com n aplicações.
- O *Android* utiliza o *Broadcast* para enviar mensagem por todo o sistema anunciando que determinado evento aconteceu, o componente que tiver interesse em respondê-la é o *Broadcast Receivers*. Exemplo, o dispositivo informa que a bateria está fraca, o componente receptor é informado e aciona a caixa de dialogo que notifica ao usuário o estado da bateria.
- *Activity* é o componente do *android* que tem a responsabilidade de controlar os estados e comportamentos de uma tela do aplicativo.

Para criar uma *activity*, a classe *Java* deve estender da classe *android.app.Activity*. Ela fornece uma *view* com a qual o usuário pode interagir. Um aplicativo pode ter uma única atividade ou ser constituído por múltiplas atividades que executam ações diferentes, sempre que se começa uma nova atividade o sistema interrompe a *activity* anterior e a encaminha para a pilha de volta, onde fica acessível até que seja destruída (FREITAS, 2012).

A Figura 14 logo abaixo, representa o ciclo de vida da *activity* de aplicações *android,* nela podem-se observar os principais métodos de *callback* que o sistema chama quando ocorre transição entre *activitys* ou alteração em seus estados. Para Freitas (2012) é fundamental que o desenvolvedor conheça esse ciclo pra garantir que os aplicativos funcionem corretamente.



Fonte: Francisco Mendes, 2014.

Figura 14. Ciclo de vida Activity.

Conforme visualizado na figura 14 cada parte do ciclo de vida possui responsabilidades como (AMORIM, 2011):

- ***onCreate()*** Quando uma *activity* é lançada este é o primeiro método a ser chamado, é executado apenas uma vez durante a vida da *activity*, em geral é o responsável por carregar o arquivo XML que contem as configurações de *layout.*
- ***onStart()*** é chamado imediatamente após o *onCreate*, ou quando uma *activity* esta em *background* e volta a ter o foco principal.
- ***onResume()*** chamado logo após o *onStart*, quando uma *activity* volta a ter foco. Ela Diferencia-se do *onStart* pois ela sempre é chamada quando há uma retomada de foco, já o *onStart* é chamado quando a *activity* não estiver mais visível no display e volta a ter foco.
- ***onPause()*** quando a *activity* perde o foco é o primeiro método a ser chamado.
- ***onStop()*** este método tem reação similar ao *onPause*, e diferencia-se pois é chamado quando a *activity* fica totalmente encoberta por outra *activity*.
- ***onDestroy()*** mata a *activity*, deve ser sempre o ultimo método a ser lançado, uma vez executado não se pode mais relançar a *activity*.
- ***onRestart()*** é chamado antes do método *onStart* quando uma *activity* que estava em *background* volta a ter o foco principal.

Os *tablets* e *smartphones* usufruem da capacidade de alterar a orientação da tela nas configurações retrato e paisagem. Quando o usuário faz um movimento de rotação com o dispositivo, ele chama o método *onDestroy*() que destrói a atividade e logo em seguida é chamado o método *onCreate*() que recria a *activity*, o *layout* é redesenhado se ajustando a nova orientação do dispositivo em relação ao usuário.

2.4.2. IOS

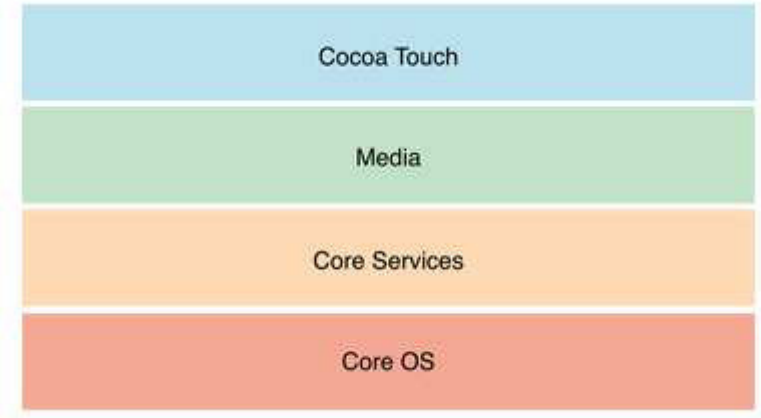
IOS é o sistema operacional para dispositivos moveis desenvolvidos pela *Apple*, feito inicialmente para o *iPhone* e posteriormente utilizado em *iPods, iPads* e *Apple TV*. As três primeiras versões foram batizadas com o nome *iPhone* seguido do numero de sua geração*,* em 2010 foi lançada a primeira versão chamada

simplesmente de “IOS” o IOS 4. Atualmente o sistema encontra-se operando de forma estável sobre a oitava versão, o IOS 8 (APPLE, 2015A).

O IOS é um sistema operacional proprietário exclusivo para os dispositivos *Apple*, o que permite os *Apps* aproveitarem totalmente os recursos oferecidos pelo *hardware*. As atualizações de SO são feitas de forma gratuita disponibilizadas anualmente pelo *iTunes* ou OTA. As aplicações desenvolvidas para o IOS estão todas disponibilizadas na *Apple Store,* lugar em que o usuário encontra mais de um milhão de aplicativos surpreendentes (APPLE, 2015B).

2.4.2.1. ARQUITETURA IOS

O IOS opera basicamente sobre quatro camadas de abstração: ***Core OS,*** ***Core Service***, **Mídia**, e a ***Cocoa Touch***. Ao projetar aplicações IOS os desenvolvedores devem investigar e compreender as tecnologias e os conceitos envolvidos em cada nível (APPLE, 2015C).



Fonte: Apple, 2015.

Figura 15. Camadas do IOS.

Conforme é evidenciado na Figura 15, a camada de mais alto nível é a *Cocoa Touch*. Possui papel fundamental na arquitetura deste SO, pois fornece a infra-estrutura necessária para suportar tecnologias-chave, tais como multitarefa, entrada com base em toque, notificações por *push*, entre outras (APPLE, 2015C).

O terceiro nível é composto pela *Media* que contem as interfaces gráficas, áudio e tecnologias de vídeo. Esta camada é responsável por prover e gerenciar os recursos de experiências multimídia (APPLE, 2015C).

A *Core Services* está no segundo nível, responsável por gerir serviços fundamentais que os aplicativos nativos utilizam. Nesta camada encontra-se as tecnologias para dar suporte as funcionalidades de localização, *iCloud*, ARC, entre outras (APPLE, 2015C).

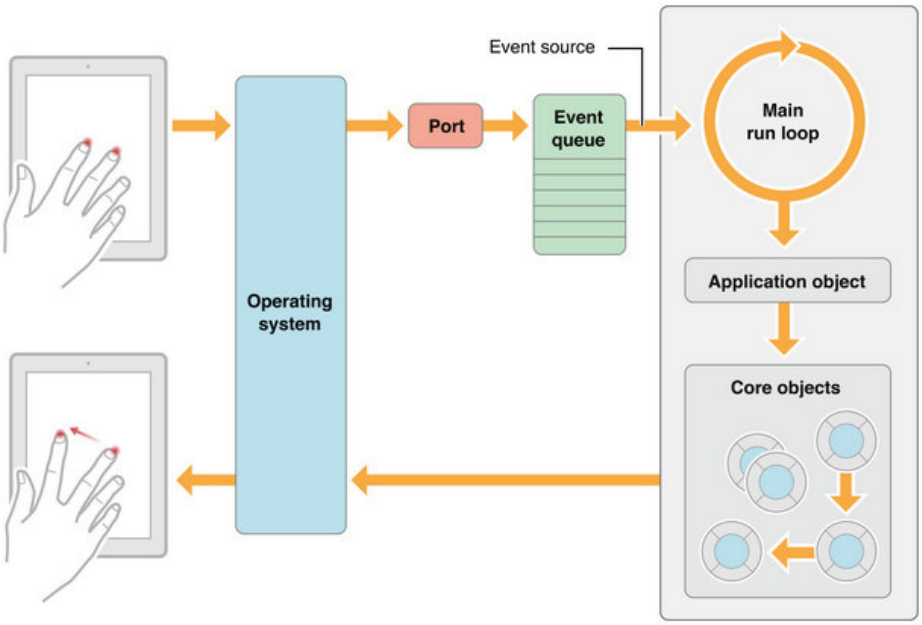
A camada *Core OS* é o *Kernel* do sistema operacional, compreende as características de baixo nível. A maioria das tecnologias presentes nas camadas superiores é construída sobre esta camada (APPLE, 2015C).

2.4.2.2. ESTRUTURA APLICAÇÃO IOS

Para desenvolvimento de aplicações IOS o desenvolvedor necessita de um aparato tecnológico mínimo, computador Macintosh Intel com SDK do IOS instalado. Segundo Apple (2015C) para transformar ideias em *Apps* é necessário que haja entendimento das interações que ocorrem entre o aplicativo e o sistema operativo.

Antes de submeter uma aplicação a loja da *Apple* o desenvolvedor deve certificar-se de que o App siga caminhos de execução bem definidos, deve-se realizar as transições entre primeiro plano e plano de fundo, sem que haja prejuízos no desempenho e performance do dispositivo. Aplicativos que estão associados a altos custos de energia passam uma experiência negativa para os usuários e são propensos a ser descartados (APPLE, 2015D).

Os *Apps* IOS fazem uso da arquitetura MVC, separando assim os dados do aplicativo, da logica de negócios e da apresentação visual. A utilização deste paradigma possibilita que aplicativos IOS possam ser executados em diferentes dispositivos com diversos tamanhos de *display* (APPLE, 2015D).



Fonte: Apple, 2015.

Figura 16. Ciclo de execução App IOS.

A Figura 16 esboça o ciclo de execução de *Apps* IOS. O usuário inicia a aplicação que como todo programa desenvolvido em linguagem C, tem como ponto de entrada a função *main*(). Está por sua vez invoca o método *UIApplicaitonMain*() que é responsável por iniciar a aplicação. É então instanciado o *application delegate* o método *applicationDidFinishLaunching* é invocado pelo objeto da aplicação e informa ao *delegate* da aplicação que já terminou o processo de lançamento da aplicação. A invocação deste método da inicio ao ciclo continuo da aplicação que fica a escuta de eventos externos. Para cada nova iteração do *Event Loop* é criada uma nova *autorelease pool* que é utilizada na gestão de memoria, sendo liberada no final do ciclo (FONSECA et. al. 2013).

Conforme o usuário interage com o *display* do dispositivo, eventos relativos as ações do usuário são gerados pelo sistema e entregues a aplicação pela porta criada pelo *UIKit*. Estes eventos são enfileirados internamente e despachados um a um para o laço principal de execução. O objeto *UIApplication* é o primeiro objeto a receber o evento e decidir a respeito do que deve ser feito (APPLE, 2015D).

2.4.3. WINDOWS PHONE

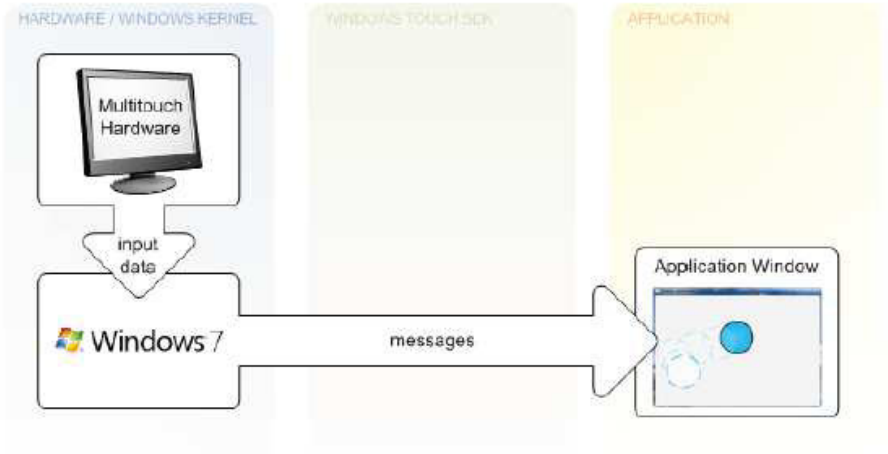
A *Microsoft* sempre encantou seus usuários com sistemas operacionais para dispositivos moveis que teve início com o *Pocket PC* que utilizava um *Windows CE*, que após 2003 evoluiu e passou a ser chamado de *Windows Mobile.* Este sistema após a primeira década do século XXI já não estava a altura de seus concorrentes *Android* e *IOS* e foi substituído pelo *Windows Phone 7* (LECHETA, 2013).

Em busca de espaço no mercado dos dispositivos moveis a empresa fechou parcerias com Nokia, HTC e Samsung, a primeira foi a maior investidora do SO e lança todos os dispositivos com Windows Phone (LECHETA, 2013). Atualmente o Sistema operacional da Microsoft encontra-se na versão Windows Phone 8 (MICROSOFT, 2015A).

Os *Apps* para *Windows Phone* são desenvolvidos em C#, que é a principal linguagem da plataforma *.NET*, o ambiente de desenvolvimento padrão é o *Visual Studio*. Segundo (LECHETA, 2013) a facilidade de publicar os aplicativos na *Windows Phone Store*, a plataforma da *Microsoft* ser uma das melhores do mercado e o ambiente de desenvolvimento ser simples, faz com que a empresa ganhe mais usuários e espaço na luta dos SOs *Mobile*

2.4.3.1. *ARQUITETURA* WINDOWS PHONE

Para abstração do conhecimento de como funciona um dispositivo executando o sistema operacional *Windows 7,* é necessário dividi-lo em três colunas, como esboçado na Figura 17 logo a baixo, que apresenta a visão da arquitetura *Windows*.



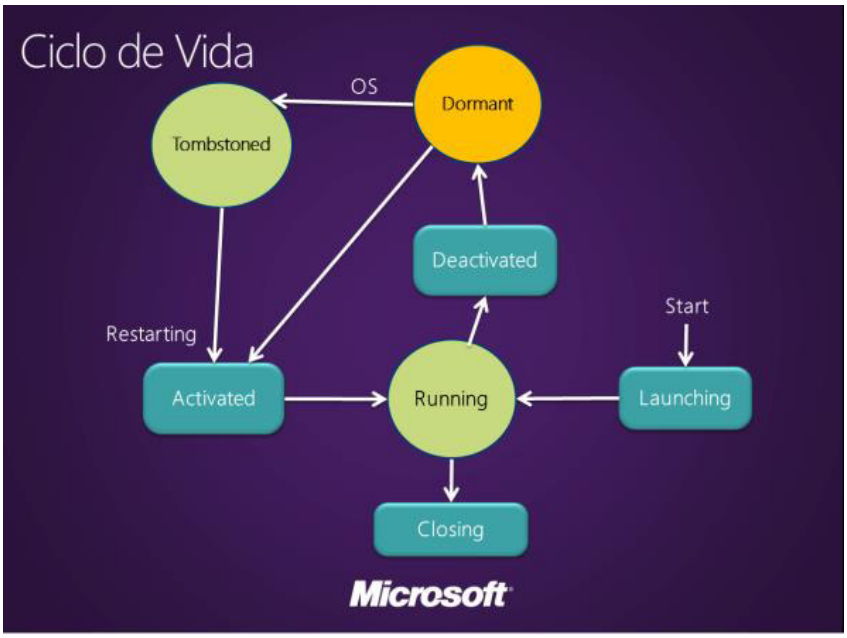
Fonte: Sergio Cavalcante, 2012.

Figura 17. Arquitetura Windows.

A coluna mais a direita compreende o *Hardware* e o *Windows Kernel*, a comunicação entre essas duas partes é feita por meio de um *drive*r. O *hardware* é sensível ao toque e ao ser tocado o sistema operacional gera um “WM_TOUCH” ou “WM_Gesture” como é abordado na coluna ao centro, está mensagem é então encaminhada para o “HWND” do aplicativo, e por fim na coluna a direita ao receber a mensagem a aplicação atualiza o conteúdo da interface de usuário (MICROSOFT, 2015B).

2.4.3.2. *ESTRUTURA APLICAÇÃO* WINDOWS PHONE

No Windows Phone apenas uma aplicação é executada em primeiro plano, segundo Microsoft (2015C) isto acontece para que o usuário tenha mais recursos disponíveis proporcionando uma experiência suave e intuitiva. O desenvolvedor de aplicações *Windows Phone* deve gerir os recursos dos dispositivos com cautela, tendo em vista que estes são finitos e concorridos. Para melhor entendimento a Figura 18, logo abaixo demonstra o ciclo de vida de um *WinApp*.



Fonte: Sergio Cavalcante, 2012.

Figura 18. Ciclo de Vida Aplicação Windows Phone.

Como observado na Figura 18, o *Start* é o momento em que o usuário inicia a aplicação por meio da tela inicial ou *App List*. O evento *Lauching* é responsável por persistir os dados da aplicação e exibir o estado inicial. O primeiro estado é o *Running* indica que o usuário inicializou o aplicativo e encontra-se pronto para uso. Caso clique no botão *Windows* ou abra outro *App* é ativado o evento *Deactivated*, a aplicação anterior vai para o segundo estado o de *Dormant*, A instância continua viva podendo consumir dados, porém o acesso aos recursos é reduzido.

Na hipótese da aplicação permanecer muito tempo no estado dormente o SO a coloca em *Tombstoned* que é o terceiro estado, neste momento o sistema enterra a aplicação preservando-a aberta na *App List*, mas apaga a instância da memoria. Para os casos de *Dormant* e *Tombstoned* ao chamar o evento de *Activated* a aplicação volta para o estado de *Runnig*, ou seja, a aplicação volta ao *foreground*. Ao clicar no botão *back* do dispositivo o usuário ativa o evento de *Closing* e encerra completamente a aplicação (MICROSOFT, 2015C).

3. METODOLOGIA

Para elaboração do presente trabalho a Priore foi realizada uma pesquisa minuciosa sobre diversas áreas de conhecimento relacionadas ao trabalho, tais como, conceitos de *Smart Cities*, Georreferenciamento, Arduino, *Web Services* e Aplicações para Dispositivos Computacionais Móveis, entre outras. O acesso a estas fontes de informação deram-se principalmente através de livros e artigos publicados na internet.

Após foi trabalhada a aquisição de componentes para a parte de prototipagem do *hardware* necessário para o sensoriamento, são eles: sensor Utrasónico, Placa Arduino, *Protoboard* e painel Fotovoltaico.

Diante disso, foi necessário a instalação e configuração dos ambientes de desenvolvimento e demais tecnologias associadas à elaboração do projeto. O dispositivo computacional utilizado para prover os serviços, sendo nele instaladas as seguintes ferramentas:

- Para o desenvolvimento do sensor com Arduino utilizou-se o IDE Arduino;
- No desenvolvimento do *Web Service* foi utilizado o Eclipse Luna;
- Para desenvolver a aplicação que comunica-se com a porta serial do Arduino utilizou-se a IDE *NetBeans*;
- No desenvolvimento do Banco de Dados que ira armazenar as informações coletadas pelo sensor ultrassônico utilizou-se o MySQL *Workbench*;
- E para o desenvolvimento da aplicação *mobile* foi utilizado o *Android Studio* com SDK *Android*.

Com os ambientes de desenvolvimento preparados o próximo passo foi desenvolver projetos de testes para familiarização e mensuração do potencial das IDEs. Por fim, fora realizada abordagem do tema energias renováveis, onde o objetivo foi alimentar a placa Arduino a partir de um painel solar.

Para prototipagem do *hardware* necessário, foram utilizados os componentes listados a seguir:

- Placa *Arduino Duemilanove* – Alimentação a Energia.

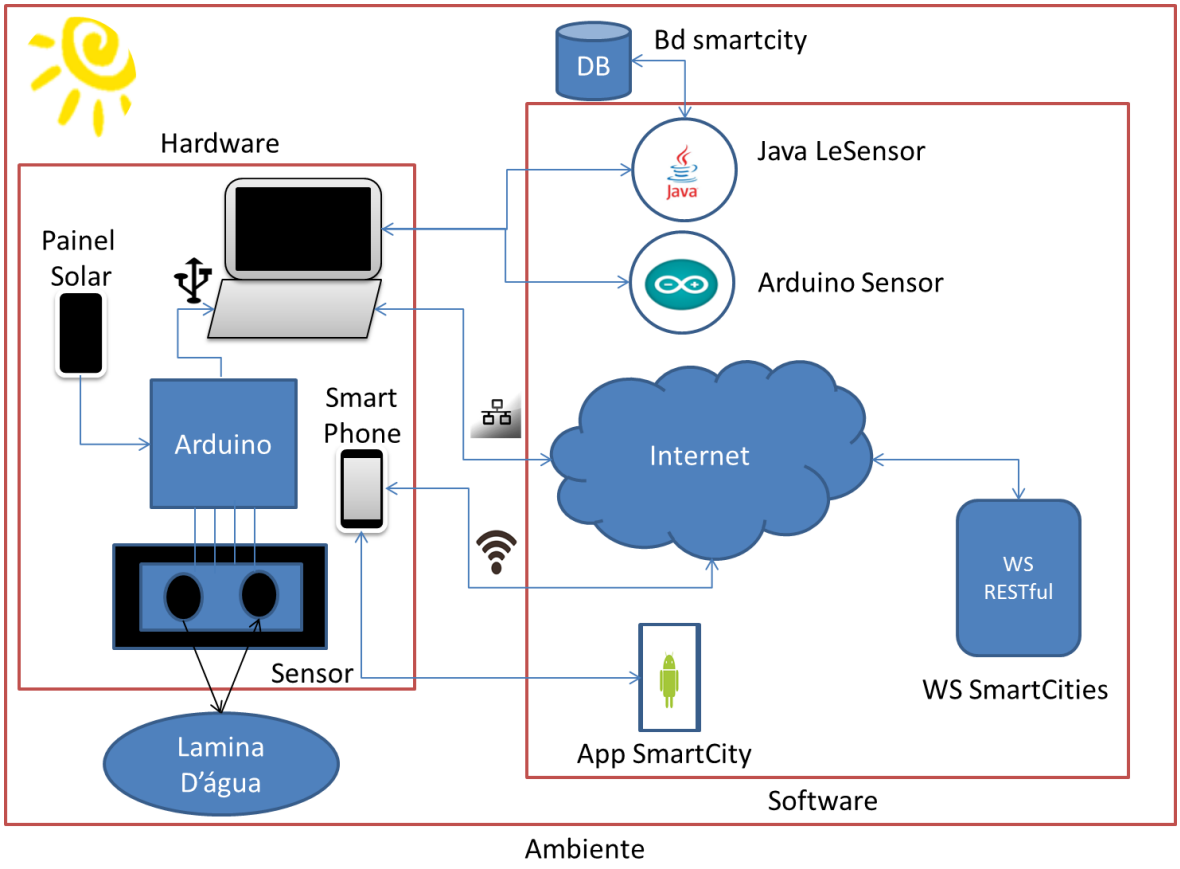
- *Jumper* – Fios com hastes metálicas nas extremidades para ligação dos circuitos eletrônicos.
- *Protoboard* – É uma base para criação de protótipos Arduino, esta repleta de furos onde são inseridos os *Jumpers* que energizam a *prootopoard* e os componentes que se deseja ligar a ela.
- Sensor Ultrassônico HC-SR04 – É composto por quatro hastes: VCC, *Trig*, *Echo* e GND. A primeira é responsável pela energização do sensor. A segunda dispara uma ação. A Terceira é responsável por receber a resposta de reflexão de dado objeto. E por ultimo o pino terra.
- Painel fotovoltaico – Este componente será abordado mais a frente na seção 4.2.3.

O trabalho foi desenvolvido nas dependências da Fundação Universidade do Tocantins, Sala de Aula e alternando hora no Laboratório de *Hardware* hora na Biblioteca. Mais detalhes da implementação desta aplicação serão descritas no próximo capítulo, que trata do processo de construção e resultados de cada *software* desenvolvido.

4. DESENVOLVIMENTO

Este trabalho empenha-se em possibilitar que a partir de um dispositivo computacional móvel com sistema operacional *Android*, o usuário possa ser notificado por meio de serviços disponibilizados por um web servisse, se determinada via de transito da cidade esta alagada. Para obtenção de resultados foi necessário trabalhar em estudos de quatro áreas primordiais desta pesquisa: Arduino, *Web Service*, Dispositivos Móveis *Android* e Cidades Inteligentes.

O Sistema esta dividido em duas partes distintas: *Hardware* e *Software* sendo a segunda subdividida em quatro partes, Arduino Sensor, Java LeSensor, WS SmartCities e App SmartCity, como pode ser conferido na Figura 19, logo abaixo.



Fonte: Elaborado pelo Autor, 2015.

Figura 19. Composição Sistema.

As próximas seções descrevem as aplicações a estrutura dos projetos e as principais funcionalidades.

4.1. IMPLEMENTAÇÃO SENSOR COM ARDUINO

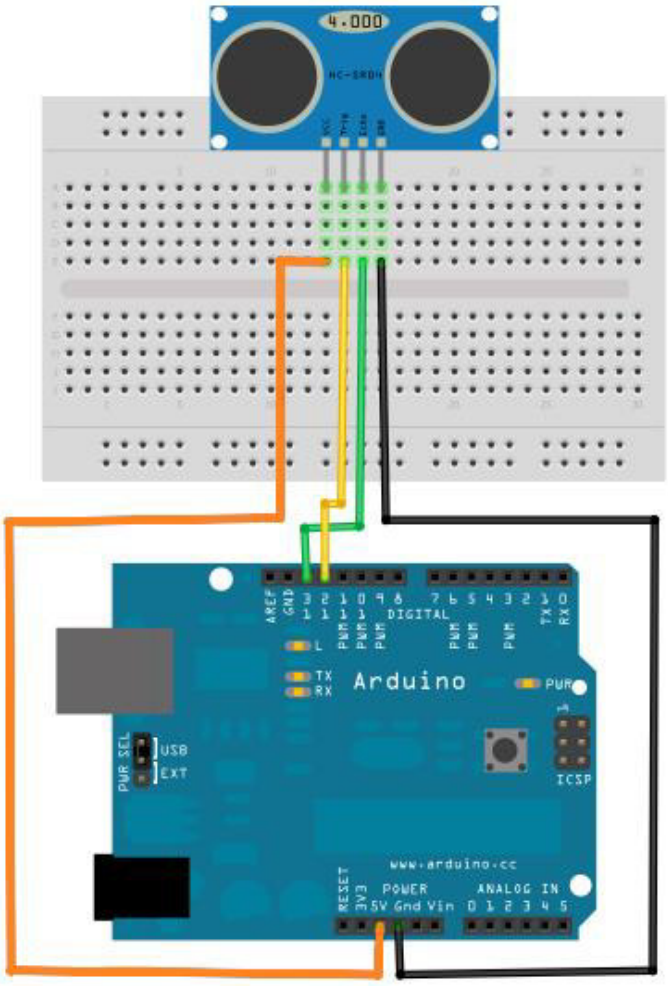
Para uma melhor metodologia de trabalho a parte que envolve a tecnologia Arduino esta dividida em duas partes: *Hardware* e *Software*, com ênfase no conceito de energias renováveis.

4.1.1. HARDWARE ARDUINO

A placa Arduino tem relevante prestigio no âmbito educacional, trata-se de um aparato de baixo custo e com curva de aprendizagem relativamente atrativa. A partir deste microcontrolador acadêmicos alcançam grandes resultados sem utilizar tecnologias de ultima geração. Tendo em vista estas informações a *Arduino Duemilanove* foi a placa escolhida para agregar valor a este trabalho.

Diversos sensores foram analisados e o escolhido para fazer parte desse projeto foi o sensor ultrassônico, pois alia a capacidade de identificar objetos com a possibilidade de calcular a distância do mesmo.

A Figura 20 logo abaixo, ilustra a estrutura do protótipo de *Hardware* desenvolvido para obtenção da altura da lamina d'água.



Fonte: Núbia Sousa, 2013.

Figura 20. Arduino com Sensor Ultrassônico.

A placa Arduino esta conectada a *protoboard* por quatro *jumpers*, sendo:

- 1º - Jumper de cor laranja está ligado ao pino 5V da Arduino e ligado ao VCC do sensor.
- 2º - Jumper de cor amarela esta ligado ao pino 12 da Arduino e ligado ao pino *Trig* do sensor.
- 3º - Jumper de cor verde esta ligado ao pino 13 da Arduino e ligado ao pino *Echo* do sensor.
- 4º - Jumper de cor preto esta ligado ao pino GND da Arduino e ligado ao pino GND do sensor.

Após o *hardware* ser montado, deu-se início a parte de implementação do *software*, os procedimentos adotados encontram-se descritos na próxima seção.

4.1.2. SOFTWARE ARDUINO

A implementação do código para a plataforma Arduino foi realizada na IDE Arduino. Como a sintaxe de desenvolvimento é baseada nas linguagens C/C++ não houve nenhuma dificuldade neste ponto. Porém, a necessidade de se configurar a porta serial aparece como obstáculo para aqueles que não têm a documentação dessa tecnologia em mãos. Geralmente utiliza-se a porta COM3, para defini-la, abra a IDE Arduino vá em, *Tools* → *Serial Port* → COM3.

Para alcançar o objetivo de mensurar a distância do sensor em relação ao solo, pontos de referência a partir dos quais se pode calcular a altura da lamina d'água. Partiu-se do conhecimento que os sensores ultrassônicos emitem ondas sonoras que ao entrar em contato com um objeto refletem as de volta ao sensor. Considera-se que a velocidade do som no ar é ~340 m/s, logo para calcular a distância entre o ponto emissor e o obstáculo utiliza-se a seguinte formula:

$$D = (T * V) / 2$$

Onde as variáveis são:

- D = Distância entre o ponto $A_{(sensor)}$ e o ponto $B_{(obstáculo)}$;
- T = Tempo gasto para a onda do sensor, ir do ponto A até o ponto B e retornar ao ponto A; e
- V = Velocidade do som (~340 m/s).

A expressão é dividida por dois, pois a variável T correspondente ao tempo gasto desde a emissão até o retorno da onda. Ao multiplicar o tempo pela velocidade, e dividir por 2, o sensor calcula a sua distância exata em relação ao objeto.

```
laminaDagua = 100;

tempo = pulseIn(echoPin, HIGH);

distanciaCm = (tempo / 58);

laminaDagua = (laminaDagua - distanciaCm);
```

Fonte: Elaborado pelo Autor, 2015.

Quadro 2. Código para calcular distância em cm.

O Quadro 2 logo acima, exibe o código necessário para calcular a distância entre dois pontos utilizando o sensor HC-SR04. As variáveis “tempo”, “distanciaCm” e “laminaDagua” são do tipo *float*, a primeira recebe o tempo de ida e volta da onda do sinal, a segunda recebe o valor da distância entre o ponto A(sensor) e o ponto B(obstáculo) em centímetros, e por fim a terceira é inicializada com 100cm e no decorrer das linhas de comando recebe o resultado da diferença entre a distância do sensor em relação ao solo e a distância do sensor em relação ao objeto, que neste exemplo é a água.

Como o volume das águas é informado em milímetros (mm) ocorreu a necessidade de transformar o resultado de centímetro para milímetros. Na escala métrica 1cm compreende 10mm, logo se a lamina d’água tem 2cm basta multiplicar esse valor por 10, e obter o resultado de 20mm.

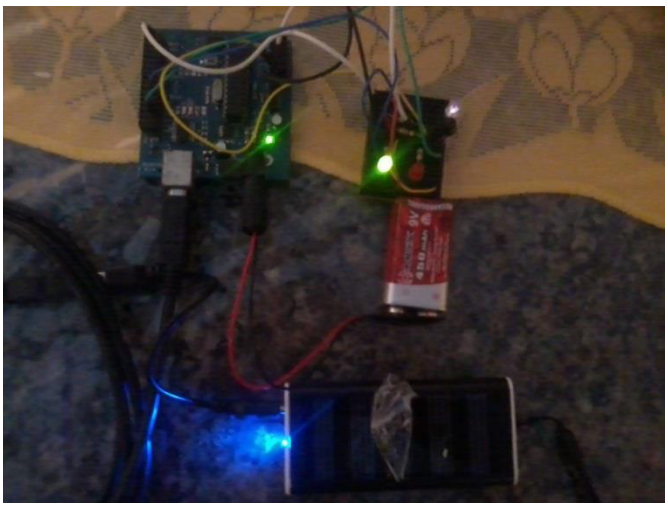
O protótipo de *hardware* com *software* para testes de mensuração de distancias apontou resultados satisfatórios, identificando a altura da lamina d’água tanto em centímetros quanto em milímetros. Para concluir a implementação da tecnologia Arduino basta apenas alimenta-la a partir de uma fonte de energia renovável, isto será abordado no próximo tópico.

4.1.3. ALIMENTAÇÃO DA PLACA ARDUINO COM PAINEL FOTOVOLTAICO

Embasado pelo estudo introdutório de *Smart Cities*, chegou-se ao entendimento que uma aplicação que tem por meta tornar a cidade mais inteligente, tem por obrigação abdicar dos conceitos energéticos tradicionais e fazer o uso de fontes de energias renováveis.

Para tornar a placa Arduino um utilizador de energia limpa, pesquisou-se qual a forma de energia renovável melhor se encaixaria neste trabalho. A escolhida foi a fotovoltaica. Logo para este protótipo utilizou-se um painel fotovoltaico de silício policristalino, com saída de 5,5 Volt, 1600 miliamperes que gera 8,8 Wp (lê-se “watt pico” de potência).

Este painel foi escolhido devido a fácil aquisição e o custo beneficio satisfatório. Foi fixado com uma inclinação de 20 graus em relação ao plano horizontal, apontando para o Norte geográfico, mostrou-se eficaz ao realizar os testes e forneceu carga energética suficiente para alimentar a tecnologia Arduino, conforme pode ser observado na Figura 21.



Fonte: Elaborado pelo Autor, 2015.

Figura 21. Arduino alimentada por painel solar.

Como a aplicação terá uso primordial em dias chuvosos com baixa incidência de luminosidade e utilização nos períodos noturnos, a estratégia utilizada para evitar que por falta de energia a aplicação pare, foi o uso de baterias para armazenar energia gerada pelo painel nos dias de sol intenso.

4.2. IMPLEMENTAÇÃO, APLICAÇÃO JAVA PARA COMUNICAÇÃO COM PORTA SERIAL DO ARDUINO E INSERÇÃO NO BANCO DE DADOS

Os dados produzidos pelo sensor serão disponibilizados em um *web service* e consumidos por uma aplicação *mobile*. Em cenário ideal, as informações emitidas pelo sensor deveriam ser enviadas de forma direta para o *web service* através de uma *Ethernet Sheild,* porém devido a falta de recursos de *hardware* para a aquisição

deste componente ocorreu a necessidade de encontrar outros meios para que se estabeleça a comunicação.

Para solucionar este obstáculo foi utilizado um *notebook* com conexão a internet, com isso originou-se a necessidade de um *software* que fique ouvindo a porta serial da Arduino e envie estes dados para um banco de dados, lugar em que o *web service* buscará as informações.

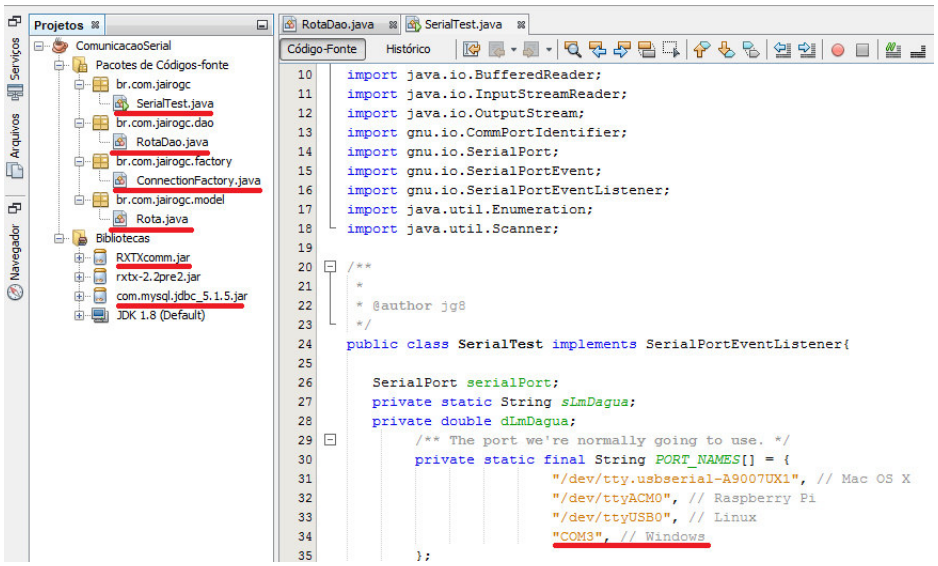
O código para que uma aplicação *Java* possa comunicar-se com a porta serial da Arduino esta disponível no site da comunidade ARDUINO.CC. Para que haja a possibilidade dessa comunicação entre a porta serial e a aplicação *Java* é fundamental a utilização das bibliotecas RXTXcomm.jar e a rxtxSerial.dll, sendo que estas devem ser compatíveis com a arquitetura do sistema operacional em uso 32-*bits* ou 64-*bits*.

A biblioteca rxtxSerial.dll deve ser colocada nas seguintes pastas:

- C:\Program Files\Java\jdk1.8.0_45\bin, onde 1.8.0_45 é a versão do JDK;
- C:\Program Files\Java\jre1.8.0_45\bin onde 1.8.0_45 é a versão do JRE;
- C:\Windows\SysWOW64; e
- C:\Windows\System32 (Caso do sistema operacional for 32-*bits*).

O código obtido no ARDUINO.CC foi incorporado ao pacote *br.com.jairogc* de uma aplicação *Java* desenvolvida na IDE *Netbeans*, esta por sua vez além de ouvir a porta serial da arduino tem a responsabilidade de conectar-se ao banco de dados e realizar as operações de SQL. Para conexão com o BD utilizou-se a classe ConnectionFactory.java que em nível organizacional encontra-se dentro do pacote *br.com.jairogc.factory*.

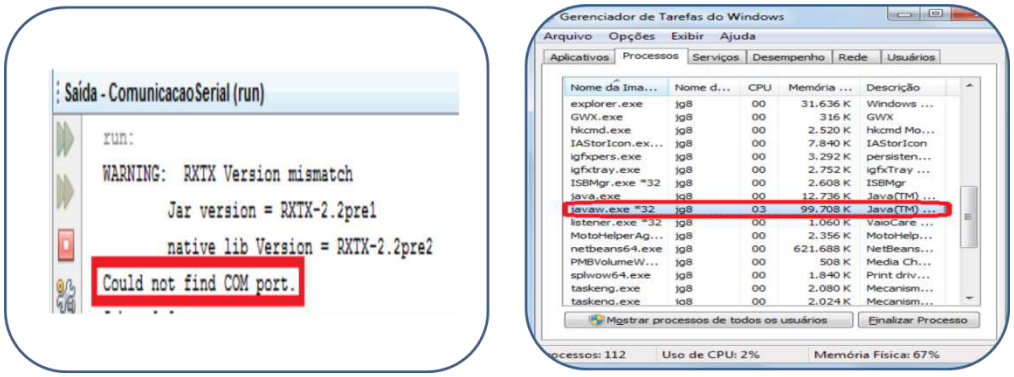
A classe Rota.java presente no pacote *br.com.jairogc.model* é a representação da tabela rota do banco de dados. Já a classe RotaDao.java tem a responsabilidade de realizar as operações SQL de inserção e edição no banco de dados. Também foi necessário utilizar os Jars RXTX.comm.jar para comunicação serial e com.mysql.jdbc_5.1.5.jar para conexão com o banco de dados. A Figura 22, logo abaixo, esboça a estrutura do projeto.



Fonte: Elaborado pelo Autor, 2015.

Figura 22. Estrutura Aplicação Java que faz a comunicação Arduino/BD.

Acontecimento que vale destacar é a aplicação não conseguir identificar a conexão USB entre Computador e placa Arduino, isso faz com que a aplicação não encontre a porta serial designada, gerando assim uma exceção.



Fonte: Elaborado pelo Autor, 2015.

Figura 23. Não encontra porta COM / Matar processo javaw.exe*32.

Conforme é verificada na Figura 23 logo a cima, a solução encontrada para este problema foi abrir o gerenciador de tarefas do *Windows* e finalizar o processo "javaw.exe*32", a partir dai a aplicação funcionou normalmente.

Outra função fundamental desta aplicação é a inserção e/ou atualização das coordenadas do sensor e altura da lamina d'água no banco de dados. As

informações de localização são inseridas manualmente pelo usuário administrador do sistema.

Para armazenar os dados coletados pelo sensor utilizou-se um banco de dados MySQL e o MySQL Workbench como ferramenta de manipulação do BD.

4.3. IMPLEMENTAÇÃO WEB SERVICE RESTful

O *Web Service* implementado se alimenta dos dados coletado pelo sensor, que envia três dados primordiais para o funcionamento de todo este aparato tecnológico, são eles: latitude, longitude e altura da lamina d'água. Os dois primeiros tratam da localização geográfica do sensor e o segundo se a região encontra-se alagada ou não.

4.3.1. ESTRUTURA DA APLICAÇÃO

Com base nos conceitos estudados para a revisão literária chegou-se ao entendimento que o *web service* utilizado para este trabalho será um WS REST, portanto *RESTful* com JSON, implementado utilizando a tecnologia *Java*, embasado pelo padrão estrutural MVC.

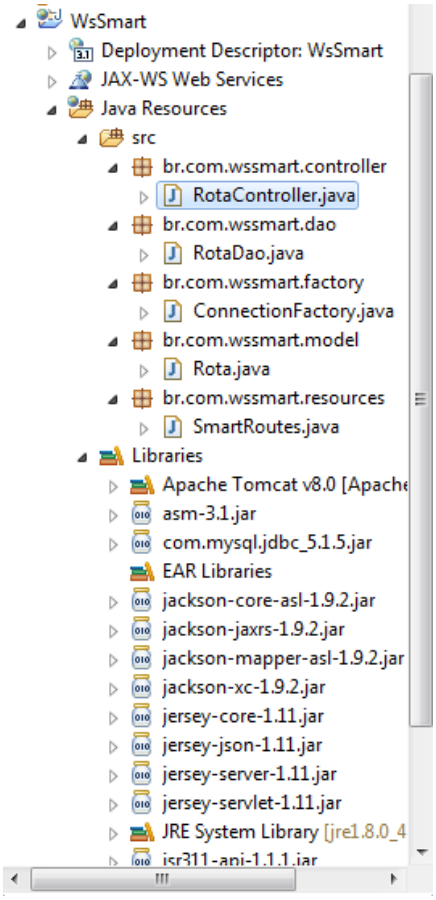
Conforme pode ser observado na Figura 24, o pacote terminado em .*factory* compreende a classe responsável por fazer a conexão com o banco de dados, o pacote .*model* contem as classes espelho do banco de dados, o pacote .dao engloba as classes que manipulam as SQL´s, *.controller* é responsável por gerenciar as classes do dao, e por fim o pacote terminado em *.resources* que tem as classes que armazenam e disponibilizam os recursos na *web*.

Para o desenvolvimento de *web services* implementados sob a arquitetura REST como é o caso do *RESTful*, foi criada a especificação JAX-RS API (JSR-311) que tem como objetivo através das anotações simplificar a construção das aplicações embasadas por essa tecnologia. Entretanto, a JAX-RS é apenas uma especificação e torna-se necessário uma implementação, que pode ser a *Jersey, framework open source* mantido pela *Oracle* (ORACLE, 2015). Este por sua vez é composto por cinco Jars fundamentais são eles:

- asm;

- Jersey-core;
- Jersey-server;
- Jersey-servlet; e
- Jsr311-api.

Para este projeto foi utilizada a versão 1.11 do *Jersey*. Os *Jars* citados anteriormente são indispensáveis para uma aplicação *Jersey*. Caso o desenvolvedor queira agregar mais funcionalidades ao sistema podem-se adicionar outros *Jars.* Em ocasiões que haja a necessidade de trabalhar com JSON, deve-se incluir o *Jersey*-json.



Fonte: Elaborado pelo Autor, 2015.

Figura 24. Estrutura web service WsSmart.

Para execução deste *web service* foi utilizado o *Container Tomcat.* Para este projeto a utilização de servidores de aplicações como o *JBoos*, *Glassfish* também apresentariam resultados semelhantes.

Ao tentar retornar a lista de rotas mapeadas, através do recurso "/routes/listarCoordenadasDaRota", a aplicação lançou uma exceção conforme é observado no Quadro 3. Isso se deu, pois a aplicação não encontrava o *application/json*.

```
        com.sun.jersey.spi.container.ContainerResponse
logException
        GRAVE: Mapped exception to response: 500 (Internal Server
Error)
```

Fonte: Elaborado pelo Autor, 2015.

Quadro 3. Exceção no retorno do JSON.

Para resolver este problema foi preciso utilizar os seguintes *Jars*: jackson-core-asl-1.9.2.jar, jackson-jaxrs-1.9.2.jar, jackson-mapper-asl-1.9.2.jar e jackson-xc-1.9.2.jar, a partir dai a aplicação funcionou normalmente.

4.4. IMPLEMENTAÇÃO APPLICAÇÃO ANDROID

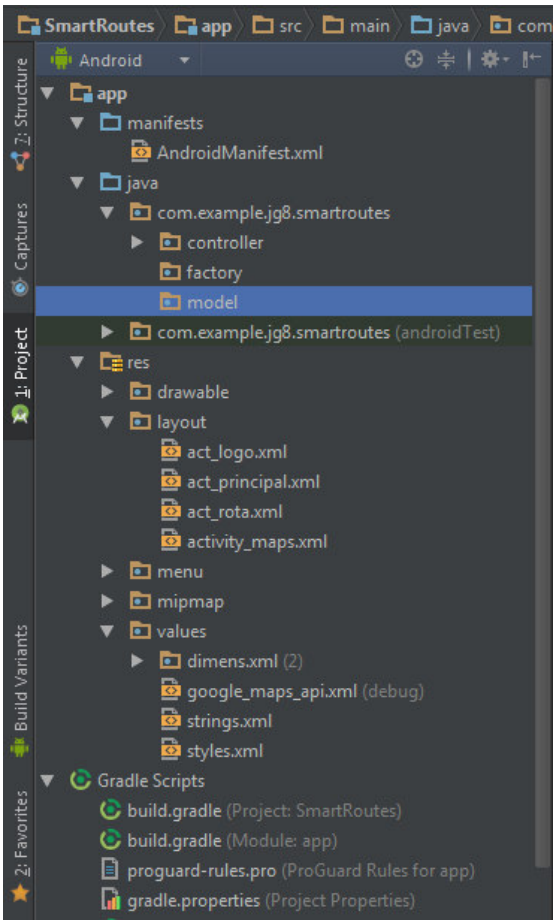
A escolha do sistema operacional ideal para o desenvolvimento do aplicativo levou em consideração a pesquisa realizada pelo IDC no ano de 2014, que apontou o SO *Android,* como o sistema que alcança o maior público alvo dos dispositivos computacionais móveis.

Para implementar a Aplicação, foi criado um projeto *android* com versão mínima 2.3. Sua estrutura esta embasada no padrão MVC, na qual se separa cada arquivo de forma a agrupa-los em pacotes de acordo com suas respectivas funcionalidades. Está estrutura pode ser observada na Figura 25.

Dentro do pacote *com.example.jg8.smartroutes.factory* esta a classe que se conecta com o *web service* citado anteriormente, no pacote *com.example.jg8.smartroutes.controller* estão contidas as classes *Activity.* No *com.example.jg8.smartroutes.model* estão as classes que servem de representação dos objetos.

O pacote "res" engloba os recursos que a tecnologia *android* disponibiliza para seus desenvolvedores. O primeiro pacote é o *drawable* que é responsável por armazenar os arquivos de mídia, como imagens. O Segundo pacote é o *layout* e

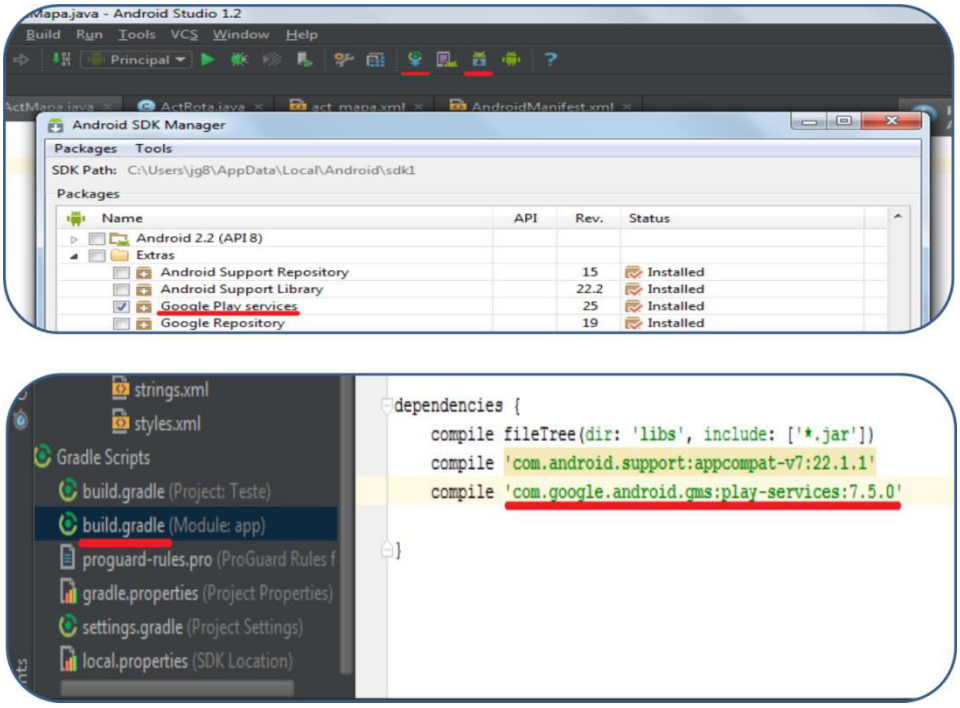
contém os arquivos XML que montam a interface gráfica de usuário, os arquivos presentes nos demais pacote são para configuração e apresentação do *layout*.



Fonte: Elaborado pelo Autor, 2015.

Figura 25. Estrutura projeto Android.

Como este projeto trabalha com o *Google Maps*, exige-se a instalação do *Google Play Services*, para isso, basta abrir o *SDK Manager*, ir ao *folder Extras* e fazer o *download*. Após este passo deve-se adicionar a API as dependências do projeto e sincronizar o projeto com os arquivos do *gradle*, conforme é demonstrado na Figura 26, logo abaixo.



Fonte: Elaborado pelo Autor, 2015.

Figura 26. SDK Manager e Gradle.

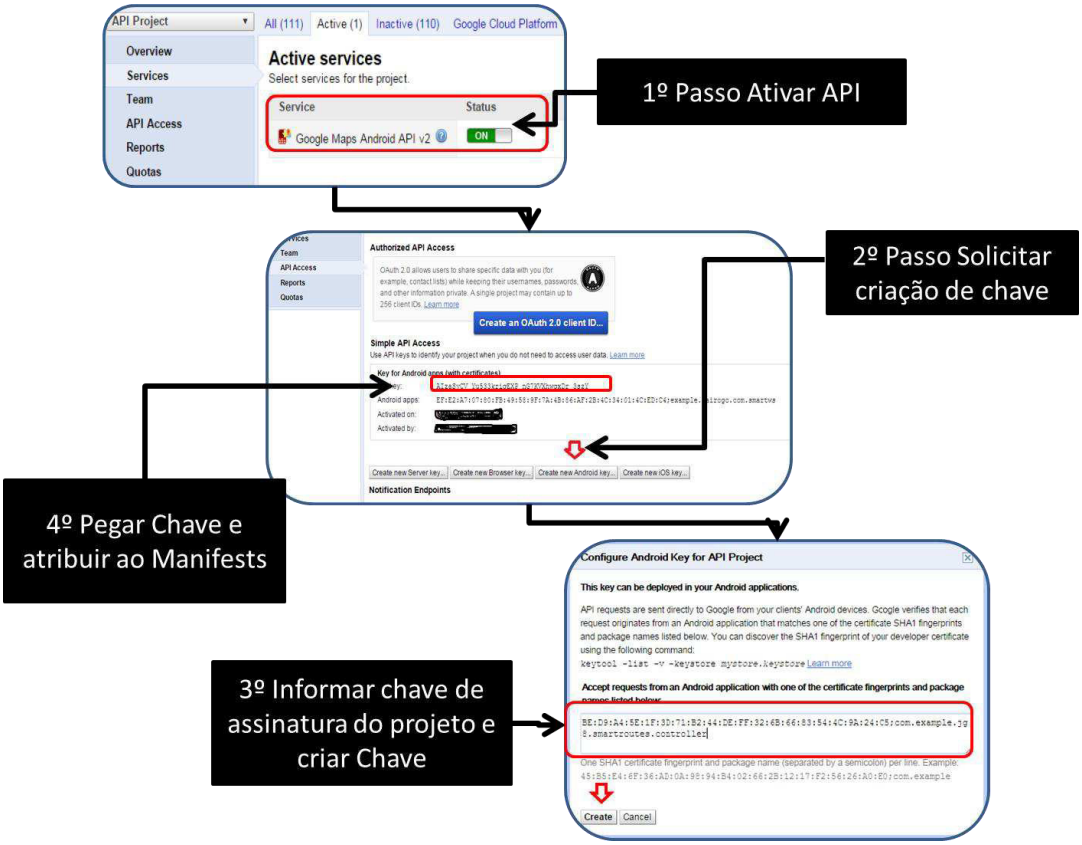
Com o *Google Play Services* instalado foi necessário declarar as permissões no *manifests*, apresentado no Quadro 4. A primeira permissão é a de "*INTERNET*" que é trivial, toda aplicação *android* que tem acesso a *web* deve utilizar esta permissão. A segunda "*ACCESS_NETWORK_STATE"* permite ao aplicativo acessar a rede através da classe *NetworkInfo*. A terceira é a "*WRITE_EXTERNAL_STORAGE",* permite gravar a aplicação no *sdcard* externo. A quarta é a "*READ_G SERVICES*" que permite ao *App* ler os dados dos *web services* do *Google.* A quinta "*ACCESS_COARSE_LOCATION* permite que a aplicação possa acessar dados de rede WiFi. E por ultimo "*ACCESS_FINE_LOCATION*" permite que a aplicação acesse dados do GPS.

```
        <uses-permission        android:name="android.permission.INTERNET"        />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission  android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
  <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"      />
  <!--
The  ACCESS_COARSE/FINE_LOCATION  permissions  are  not  required  to  use
    Google  Maps  Android  API  v2,  but  are  recommended.
  -->
  <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="SmartRoutes"
    android:theme="@style/AppTheme" >
    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="AIzaSyAO2Du3fGO0nLgxHzKtnL_Pzg3hrw1esKg" />
       .
       .
       .
```

Fonte: Elaborado pelo Autor, 2015.

Quadro 4. Inclusão de permissões no manifesto.

Ainda, dentro do *Manifests,* especificamente na *tag application* deve-se declarar a *tag meta-data* e especificar os valores para "*android:name" e "android:value"* que permitem a utilização de mapas do *Google,* neste caso o *google maps v2*. Para isso é necessário que se tenha a chave de acesso da API em questão. Para conseguir a chave o desenvolvedor deve ir ao *link:* https://code.**google**.com/apis/**console**/?noredirect e acessar com uma conta *gmail,* ativar a *Google Maps API v2*, e após a ativação clicar em criar uma chave *Android*. Com isso, será aberta uma janela que pede o *fingerprints*, para obtê-lo basta ir no *Android Studio* em, *File →New → Google →Google Maps Activity*, no pacote *values* será criado o arquivo "*google_maps_api.xml". Deve* obter no arquivo o código *fingerprints,* na pagina "*code.google"*, como esboçado na Figura 27 logo abaixo. Pode-se excluir a *activity* criada para o *google maps* ela é irrelevante.



Fonte: Elaborado pelo Autor, 2015.

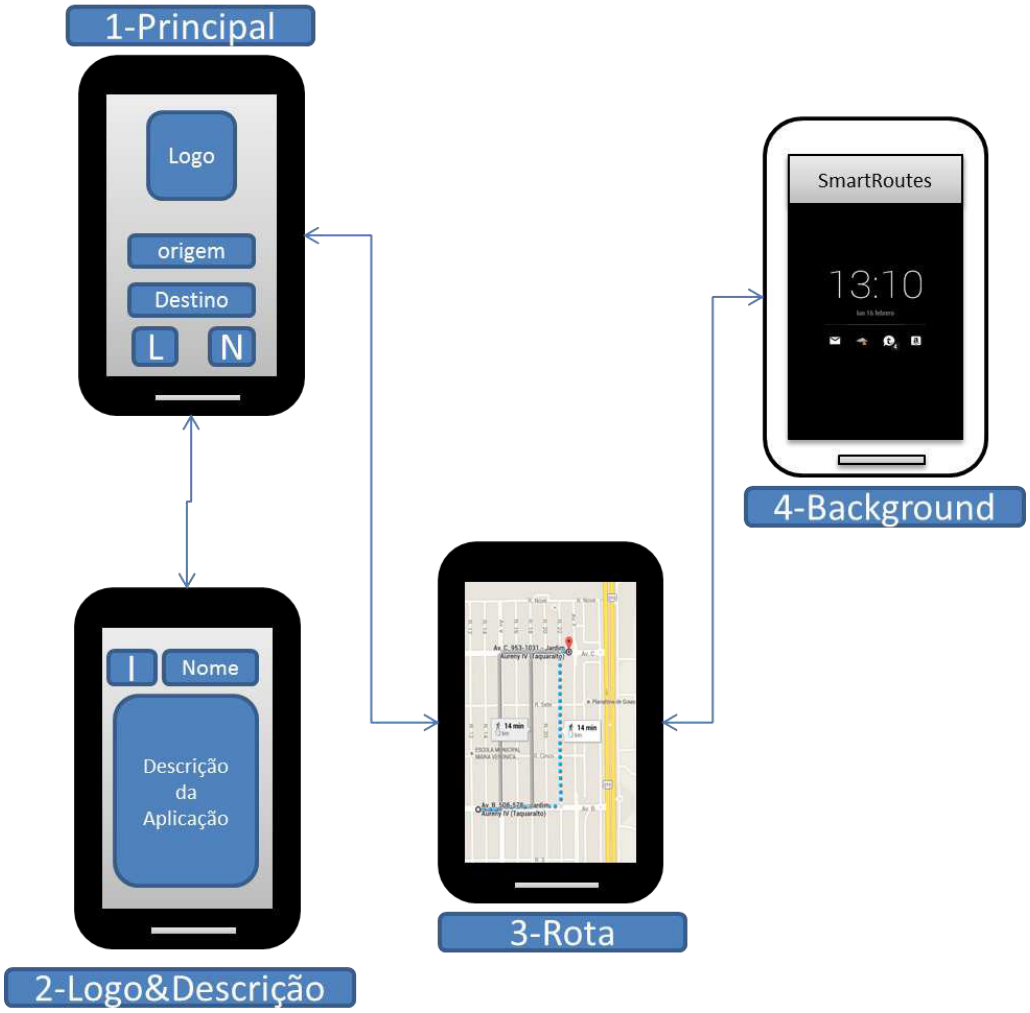
Figura 27. Chave API Google Maps v2.

Com estes procedimentos citados acima o usuário pode dar inicio ao desenvolvimento de aplicações que fornecem serviços de geolocalização utilizando-se das características oferecidas pela *Google Maps API v2.* Vale ressaltar que o emulador padrão do *Android Studio* não possui o *google play service* instalado, alguns sites da internet fazem a abordagem de como solucionar este obstáculo, porém existem outras duas soluções mais viáveis que são a utilização do emulador *Genymotion* ou o uso de um dispositivo real.

4.4.1. A INTERFACE

A proposta da interface deste aplicativo é possibilitar ao usuário uma experiência simples com funcionalidades intuitivas, conforme é mostrado na Figura 28. A visão inicial do aplicativo apresenta a logo marca do *App*, dois campos de *inputText* para o usuário informar o ponto de origem e o ponto de destino que ele deseja monitorar, logo mais abaixo aparecem dois botões Limpar Rota e Nova Rota. Caso o utilizador da aplicação clique na funcionalidade limpar rota e não existir

nenhum percurso pré-definido é retornado um *feedback* informando que não existe rota traçada, se existir, um *pop up* é aberto solicitando a confirmação da exclusão do caminho definido anteriormente.



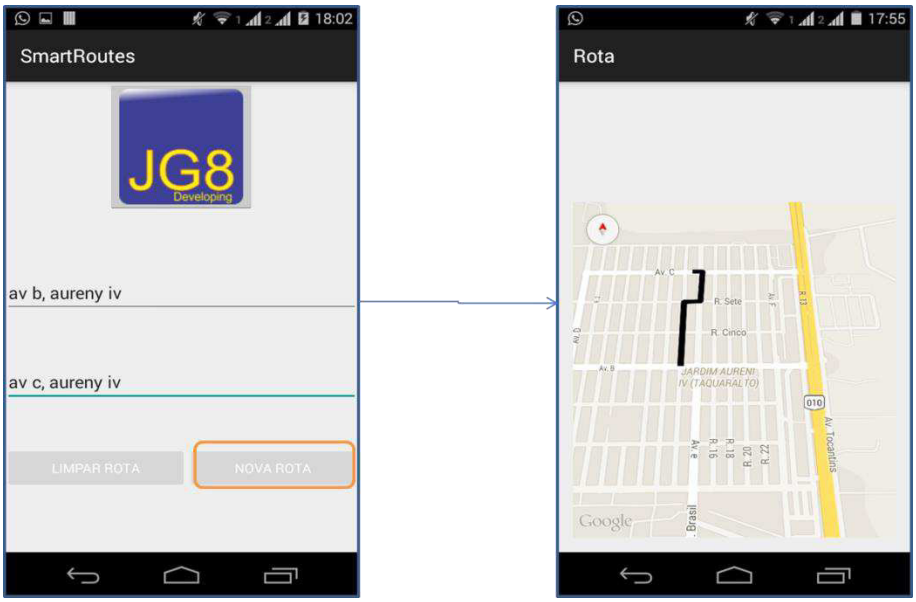
Fonte: Elaborado pelo Autor, 2015.

Figura 28. Storyboard App SmartRoutes.

A logo marca é um *imageButton* que ao ser clicado leva a segunda tela do aplicativo, lugar em que são apresentados um componente *imageView* com a foto do desenvolvedor seguido de um *texView* e o seu nome, logo abaixo é exibido *texView* com uma breve descrição pertinente as funcionalidades do sistema.

Ao informar os pontos de origem e destino e clicar no botão nova rota a aplicação é encaminhada para a terceira tela, lugar no qual, apresenta-se um mapa

mostrando um ponto de origem ligado ao ponto de destino por uma rota, informando ao usuário o percurso existente entre os dois pontos, conforme é verificado logo abaixo na Figura 29.



Fonte: Elaborado pelo Autor, 2015.

Figura 29. Inserção da origem e destino e plotagem do percurso no mapa.

Se a aplicação estiver em *background* e o sensor identificar um ponto alagado no percurso definido pelo usuário o sistema envia uma notificação que ao ser clicado traz a aplicação para *foreground* e informa ao usuário qual ponto ele deve evitar para não pegar uma via de transito alagada.

4.4.2. RECURSOS

Com base na ideia central do projeto o recurso que este aplicativo oferece é a possibilidade de o usuário traçar uma rota cotidiana e a partir destes dados informar se algum ponto do percurso traçado encontra-se alagado. O usuário informará um caminho a aplicação que é salvo em um arquivo “*.txt”, este por sua vez é responsável por persistir os dados informados até que o utilizador do aplicativo decida excluir a rota ou alterar o percurso.

Ao definir um ponto de origem e um ponto destino, a API devolveu o percurso existente entre as extremidades, este retorno é um objeto JSON. Acessa-

se o Array "*routes*", como neste caso foi definida apenas uma rota, pegou-se a primeira posição do *array*.

Dentro de *routes* acessou-se o *Array* "*legs*", como não foi definido nenhum wayPoints, foi pego a primeira posição do *array*, que contem o objeto "*distance*" responsável por armazenar os valores de distância entre a origem e destino, tanto em quilômetros quanto em metros, o objeto "*durations*" que trás a informação do tempo necessário para percorrer a rota definida, entre outras informações.

Ainda dentro de "*legs*" acessou-se o *Array* "*steps*", que contém as informações primordiais para o funcionamento deste aplicativo. O *steps* compreende os passos para montar a rota, suas informações são: *distance, duration, endLocation, polyline* e o *startLocation*.

Vale ressaltar que ao construir o aplicativo, caso sejam utilizadas as coordenadas de *endLocation* e *startLocation*, a rota perderá a fidelidade e poderá não apresentar as pequenas curvas do percurso. Para resolver este empasse, pegou-se o valor do atributo "*points*" dentro do objeto *polyline*, este resultado é um código *hash* que contém todos os pontos entre *startLocation* e o *endLocation*. Para decodificar a resposta e transforma-la em latitude e longitude, foi utilizado o método "*decodePolyline"* disponibilizado pela comunidade *stackoverflow* na *web*. Após extrair as coordenadas de latitude e longitude, ficam legíveis ao entendimento humano e podem ser manipulas conforme o desenvolvedor desejar.

4.5. TESTE DA INTEGRAÇÃO DOS SOFWARES

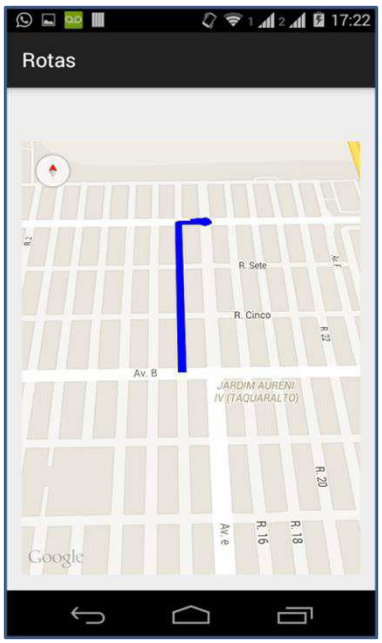
Como já citado o trabalho aborda três diferentes áreas de conhecimento: arduino, *web service* e aplicações *android*. O desenvolvimento de sistemas para cada tecnologia citada anteriormente foi fundamental para alcançar os objetivos isolados, porém para alcançar o objetivo geral proposto por este projeto foi necessário integrar essas tecnologias e realizar testes para analise da veracidade de seus resultados.

4.5.1. TESTES TECNOLOGIA ARDUINO

Para os testes foi escolhido um percurso no bairro Aureny IV na cidade de Palmas-TO, o ponto de origem é a Avenida B e o ponto de destino é a Avenida C.

Com a rota já definida a Figura 30, logo abaixo, exibe as coordenadas de cada sensor através dos atributos latitude e longitude ao longo das vias de transito.

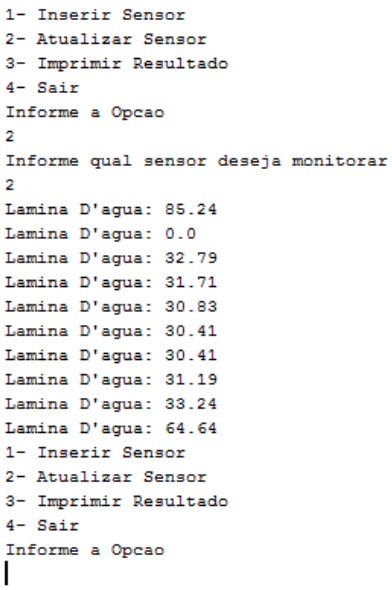
id	latitude	longitude
1	-10.30402	-48.30327
2	-10.29793	-48.30267
3	-10.29797	-48.30254
4	-10.29794	-48.3025
5	-10.29786	-48.30248
6	-10.29782	-48.30249
7	-10.29777	-48.30253
8	-10.29776	-48.30257
9	-10.29777	-48.30264
10	-10.2978	-48.30267
11	-10.29782	-48.30299



Fonte: Elaborado pelo Autor, 2015.

Figura 30. Coordenada dos Sensores.

Devido às limitações impostas por não estar no período das chuvas, dependência de computador para realizar a comunicação entre sensor e *web service* e a aquisição de um único sensor, os testes realizados foram simulados em um cenário de ensaio. Para isso registrou-se onze coordenadas de latitude e longitude que representam uma localização no mapa, cada uma destas é associada a um sensor.



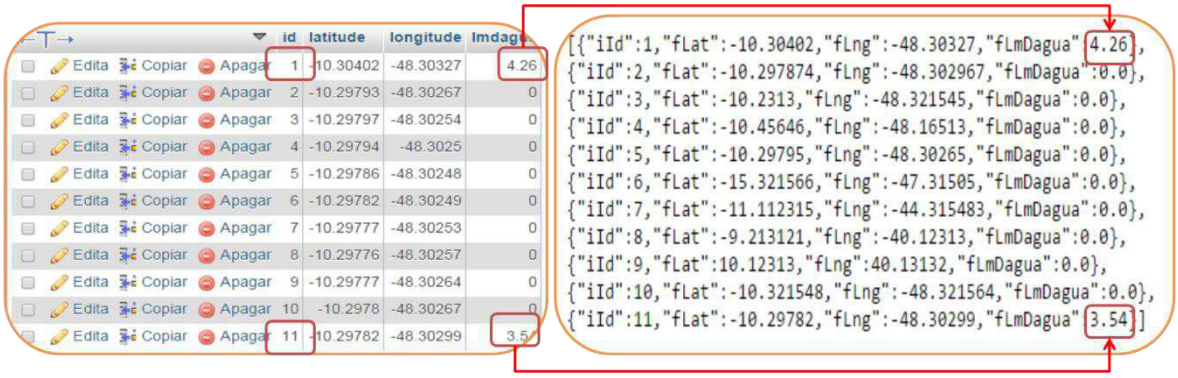
Fonte: Elaborado pelo Autor, 2015.

Figura 31. Leitura da lamina d'água realizada pelo sensor 2.

Conforme é exibido na Figura 31, logo acima, foi escolhido o sensor de número 2 para ser monitorado, este representativamente está colocado nas coordenadas de latitude: -10.29793 e longitude: -48.30267. Conforme se inseriu a água no recipiente o sensor fixado sobre este vasilhame a 100 cm de altura identificou que o valor da lamina d'água foi se elevando até alcançar a grandeza de 64.64 cm.

4.5.2. TESTES COM WEB SERVICE

Os dados colhidos pelo sensor foram inseridos no banco de dados, e estão disponíveis pelo *web service* "WsSmart", através do recurso "*listarTodasCoordenadas"*, que é do tipo "*GET*" e retornou um "JSON" informando o atributo "id" que identifica o sensor, os atributos "latitude" e "longitude" que demarcam a localização do sensor no mapa, e o atributo "lmdagua" que apresenta em tempo real a altura que a lamina d'água se encontra.



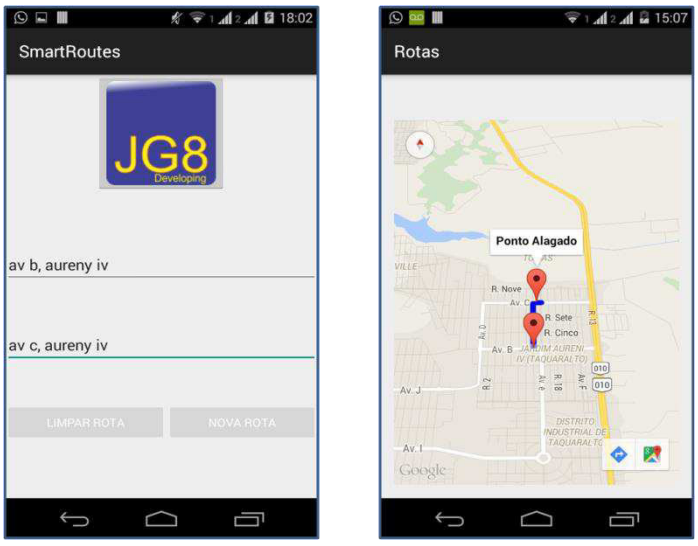
Fonte: Elaborado pelo Autor, 2015.

Figura 32. Leitura da lamina d'água realizada pelo sensor 2.

A Figura 32 logo a cima, esboça os resultados colhidos pelo sensor, na tabela a esquerda estão os dados armazenados no banco de dados e a direita a resposta do recurso "*listarTodasCoordenadas*" em formato JSON. Note que foi identificada alteração da lamina d'água nos sensores 1 e 11.

4.5.3. TESTES APLICAÇÃO ANDROID

O usuário informou no campo de origem a "av b, aureny iv", e no campo de destino a "av c, aureny iv" conforme é exibido na Figura 33. O aplicativo consumiu os recursos disponibilizados pelos servidores do *Google Maps* e plotou o percurso entre estes dois pontos.



Fonte: Elaborado pelo Autor, 2015.

Figura 33. Identificação de área alagada.

O Aplicativo cruzou os dados recuperados do *Web Service* “WsSmart” que contém as coordenadas de cada sensor, com os dados recuperados dos servidores do *Google* que compreende todos os pontos dentro de uma rota definida. Caso alguma coordenada onde foi fixado o sensor informe o valor da lamina d’água superior a 25mm que é 2,5 cm, a aplicação entende que está via encontra-se alagada e notifica ao usuário o ponto como uma região de alagamento, conforme é ilustrado pelos balões vermelhos na Figura 33. O valor de 25mm foi atribuído com o objetivo de discriminar algumas variáveis inconsistentes no ambiente, como poeira, lama e vegetação.

5. CONSIDERAÇÕES FINAIS

Com o intuito de tornar as cidades cada vez mais inteligentes, identificou-se que o uso de dispositivos computacionais móveis é um excelente recurso para propagação, disseminação e gestão de informações. As cidades modernas fazem uso de centrais de informação que provem diversos serviços disponíveis aos habitantes a fim de tornar o cotidiano das pessoas mais tranquilo e ágil com ênfase em conceitos de energias sustentáveis.

Com isso o estudo do acervo bibliográfico abordou paradigmas e ferramentas que podem ser utilizadas para o desenvolvimento de tecnologias que aliam a infraestrutura das cidades com as TIC´s. A partir deste estudo abstraiu-se conhecimento necessário para a escolha das tecnologias utilizadas neste projeto tais como: Arduino, Sensor Ultrassônico, *Web Service RESTful*, Sistema Operacional *Android* e Painel Fotovoltaico

Desta maneira foi desenvolvido um aplicativo para dispositivos computacionais móveis com SO *Android*, com o intuito de identificar as condições de trafego de uma via de transito terrestre de determinada cidade em dias chuvosos. Através de sensores ultrassônicos colocados em coordenadas estratégicas, podem-se coletar dados pluviométricos das ruas e disponibiliza-los para que as pessoas possam gerenciar seus percursos e evitar rotas alagadas.

A escolha por trabalhar com *web service* como modelo cliente servidor, deu-se por sua característica de interoperabilidade entre diferentes tecnologias. Ao disponibilizar um recurso por meio de *web services*, este ficará acessível tanto para aplicações *web* quanto para aplicativos implementados para sistemas operativos como IOS e *Windows Phone*. Conforme descrito na revisão literária *web services* construídos com base no padrão arquitetural REST apresentam índices de desempenho superiores aos embasados na arquitetura SOAP quando se trata de clientes *mobile*.

5.1. PONTOS FRACOS

Neste trabalho a detecção da altura da lamina d'água é realizada através de um sensor de distancia ultrassônico que embora tenha alcançado resultados

satisfatórios nós testes, onde variáveis inconsistentes foram desprezadas, não consegue diferenciar padrões de imagens, logo se, outro objeto entrar em seu caminho ele o identificará como água e poderá gerar resultados enganosos.

Outro ponto fraco identificado foi o envio de dados da placa arduino para o *web service*, para isso este trabalho utilizou um computador conectado a internet. Vale ressaltar que essa solução mostrou-se satisfatória nos testes de laboratório onde o cenário desprezou variáveis inconsistentes, mas se torna inviável caso o desenvolvedor queira aplicar esta tecnologia em cenário real, haja vista, que em cada ponto que se colocar um sensor haverá a necessidade de um computador com acesso a rede mundial de computadores.

O tópico a seguir aborda algumas funcionalidades que poderão deixar essa tecnologia com mais recursos e retornar resultados com maior grau de confiabilidade para seus usuários.

5.2. TRABALHOS FUTUROS

Com o desenrolar da implementação dos *softwares e hardwares* que compõe este aparato tecnológico, percebeu-se a necessidade de algumas funcionalidades que irão agregar valor ao aplicativo e enriquecer a experiência de uso. Logo abaixo são abordadas algumas delas:

- Inserção de origem e destino via toques no mapa, afim de, minimizar os esforços do usuário na aplicação;
- Implementação de sensor que identifique a lamina d'água por padrão de imagens;
- Realizar comunicação arduino com *web service* a partir de uma *Ethernet Shield*;
- Identificar lamina d'água por sistemas de padrão de imagens ;
- Sistema *Web* desenvolvido em JSF para gerenciar o estado dos sensores e a veracidade de seus resultados;
- Diversificar as opções de sistemas operativos, como IOS e *Windows Phone*; e

- Ampliação dos recursos fornecidos pela aplicação, como exemplo informar a melhor rota alternativa após identificar um ponto alagado no percurso original.

Ao identificar possíveis melhorias e novas funcionalidades para o sistema aqui desenvolvido, pode-se agregar valor e tornar os resultados deste aplicativo mais satisfatório a seus usuários além de poder despertar a curiosidade em outros pesquisadores. Considerando que a busca pela inovação e conquistas no âmbito tecnológico é cotidiana, esperamos que este trabalho também sirva de subsidio teórico para outros acadêmicos e pesquisadores.

6. REFERÊNCIAS

ADAMOWSKI, Julio Cezar**. Cap. 1- Introdução**. São Paulo: Escola Politécnica da USP. São Paulo. Disponível em: < http://www.atcp.com.br/imagens/produtos/ceramicas/artigos/Sensores_Teoria_e_Aplicacoes.pdf >. Acessado em: 01/02/2015.

AMORIM, Anderson Duarte de. **Android** – Uma Visão Geral. 2011.

ANDRADE, Claudenir C. **NFC-e Unlocked**. Disponível em: < http://desenvolvedoresdaruma.com.br/home/downloads/Site_2011/NFCe/STKs/NFCe_Unlocked_ClaudenirAndrade.pdf>. Acessado em: 13/09/2014.

ANDROID. **Android Studio**. Disponível em: < http://developer.android.com/sdk/index.html >. Acessado em: 03/05/2015.

ANELL. **Energia Solar.** cap. 3, p: 29-42. Disponível em: < http://www.aneel.gov.br/aplicacoes/atlas/pdf/03-Energia_Solar(3).pdf >. Acessado em: 04/09/2014.

APPLE. **IOS 8**. Disponível em: < http://www.apple.com/br/ios/ >. Acessado em: 10/05/2015A.

APPLE. **IOS 8 –** O sistema operacional móvel mais avançado do mundo. Disponível em: < https://www.apple.com/br/ios/what-is/ >. Acessado em: 10/05/2015B.

APPLE. **IOS Technology Overview**. Disponível em: < https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007898-CH1-SW1 >. Acessado em: 10/05/2015C.

APPLE. **The App Life Cycle**. Disponível em: < https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007072-CH1-SW1 >. Acessado em: 10/05/2015C.

ARDUINNO.CC. **Pagina oficial Arduino,cc –** Arduino Duemilanove. Disponível em: < http://arduino.cc/en/Main/arduinoBoardDuemilanove >. Acessado em: 04/10/2014.

AXIS.APACHE. **Documentation Index**. Disponível em: < http://axis.apache.org/axis2/java/core/docs/contents.html >. Acessado em: 15/03/2015.

BASTOS, Teodiano F. et. Al. **USO DE SENSORES ULTRA-SÔNICOS NA MEDIÇÃO DE PARAMETROS EM ROBÓTICA E OUTRAS APLICAÇÕES**. Madrid: Instituto de Automática Industrial. Disponível em: < http://www.sba.org.br/revista/volumes/v3n1/v3n1a06.pdf >. Acessado em: 03/02/2015.

BEZERRA, Adonel. **Guia de Estudos para Analise de Vulnerabilidades**. Manaus, 2013.

BOND, Matyin. **Aprenda J2EE em 21 Dias**. São Paulo: Pearson Education, 2003.

BRASIL, Ministério do Meio Ambiente (MMA). **Carvão.** Brasília 2014. Disponível em: <http://www.mma.gov.br/clima/energia/fontes-convencionais-de-energia/carvao>. Acessado em: 25/09/2014.

CAMACHO JÚNIOR, Carlos Olavo de Oliveira. **Desenvolvimento em Camadas com C# .NET**. Florianópolis: Visual Books, 2008.

CARDOSO, Manoel Antônio. **Geografia**: Texto Experimental. 1ª. ed. Minas Gerais, 2006.

DORNELAS, José Carlos Assis. **Empreendedorismo – Transformando Idéias em Negócios**. 2. ed, Rio de Janeiro: Campus, 2005.

FACUNTE, Emerson. **Delphi Internet e Banco de Dados 7** – Cap.13 Webservices. Disponível em: <http://www.facunte.com.br/2009/07/02/todos-os-capitulos-do-livro-delphi-7-internet-e-banco-de-dados-em-pdf-gratuito/>. Acessado em: 10/03/2015.

FASCIONI, Ligia. **GPS Para Curiosos** – Coisas que você queria saber e não tinha para quem perguntar. 1ª. ed. E-book: Brasil, 2013.

FONSECA, Nuno. et. al. **Desenvolvimento em IOS**. Iphone, Ipad e Ipood Touch. 2ª. ed. Editora de Informática, 2013.

FREITAS, Lucas Medeiros de. **Android –** Desenvolvendo Aplicativos para Dispositivos Moveis. Hachi Tecnologia, 2012.

GOOGLE. Conheça o Google Maps. Disponível em: <https://www.google.com/intl/pt-BR/maps/about/>. Acessado em: 26/05/2015.

HUERTA, Eduardo, et. al. **GPS** Posicionamento Satelital. 1ª. ed. Rosário: UNR Editora, 2005.

LECHETA, Ricardo R. **Desenvolvendo para Windows 8**. Aprenda a Desenvolver para Windows Phone 8 e Windows 8. 1ª. ed. São Paulo: Novatec, 2013.

LEE, Valentino. et. al. **Aplicações Móveis** – Arquitetura, projeto e desenvolvimento. São Paulo: PEARSON, 2005.

MARTINS, Geomar Machado. **Princípios de Automação Industrial**. Santa Maria: Universidade Federal de Santa Maria, 2012.

MARTINS, Rogerio Samuel. **Composição Dinâmica de Web Services** – Dissertação de Mestrado Universidade Vale do Rio do Sino. São Leopoldo, 2007.

MENDES, Roberto Damiani apud EAN Brasil. **Desenvolvimento de um Sistema de Automação Comercial para Ambiente WEB**. 56 f. Tese (Monografia Ciência da Computação)-Universidade Federal de Lavras, 2005.

MICHAELES. **Hardware**. Disponível em: < http://michaelis.uol.com.br/moderno/portugues/index.php?lingua=portugues-portugues&palavra=hardware >. Acessado em: 20/09/2014.

MICHAELES. **Software**. Disponível em: < http://michaelis.uol.com.br/moderno/portugues/index.php?lingua=portugues-portugues&palavra=software >. Acessado em: 08/10/2014.

MICROSOFT. **Windows Phone 8**. Disponível em: < http://www.windowsphone.com/pt-br/how-to/wp8 >. Acessado em: 20/05/2015A.MICROSOFT. **Architetural Overview**. Disponível em: <

https://msdn.microsoft.com/en-us/library/windows/desktop/dd371413(v=vs.85).aspx > acessado em: 20/05/2015B.

MICROSOFT. **App activation and deactivation for Windows Phone 8** . Disponível em: < https://msdn.microsoft.com/en-us/library/windows/apps/ff817008(v=vs.105).aspx >. Acessado em: 20/05/2015C.

ORACLE. **Welcome to the Community Wiki for Project Jersey**. Disponível em: < https://wikis.oracle.com/display/Jersey/Main >. Acessado em: 12/03/2015.

RIOS, Jefferson, et al. **Introdução ao Arduino**. FACOM-UFMS, 2012.

RÜTHER, Ricardo. **Edifícios Solares Fotovoltaicos**. 1. ed. Santa Catarina: UFSC/LABSOLAR, 2004.

SANTOS, Aldri Luiz, at al. **Cidades Inteligentes: Desafio para a Computação**. In: Congresso Sociedade Brasileira de Computação - CSBC . Fortaleza. Anais: CSBC-Fortaleza, 2013. p.9.

SAUDATE, Alexandre. **REST** – Construa API´s inteligentes de maneira simples. São Paulo: Casa do Código, 2015.

SCHIMITZ, Daniel. **Criando Sistemas RESTful com PHP e JQUERY –** Uma abordagem pratica na criação de um sistema de vendas. São Paulo: Novatec, 2013.

SELADA, Catarina, at al. **Índice de Cidades Inteligentes**. Portugal. Europress-Industria Grafica, 2012.

SHEPHERD, Geoge. **Microsoft ASP.NET 2.0 – Passo a passo**. Tradução Cláudio Belleza Dias - Porto Alegre: Bookman, 2007.

SONY. **SmartWatch 3 –** Com tecnologia AndroidWear. Disponível em: < http://www.sonymobile.com/br/products/smartwear/smartwatch-3-swr50/features/#tabs >. Acessado em: 25/05/2015.

TAURION, Cezar. **Cidades Inteligentes**: O desafio de preparar as cidades para as próximas décadas. Rio de Janeiro, 2014.

WENDLING, Marcelo. **Sensores**. 2ª versão. São Paulo: UNESP-Guaratinguetá, 2010.

W3C, Brasil. **Sobre o W3C**. Disponível em:< http://www.w3c.br/Sobre >. Acessado em: 06/12/2014.

LANDIM, Wikerson. Wearables: será que essa moda pega? Disponível em: < http://www.tecmundo.com.br/tecnologia/49699-wearables-sera-que-esta-moda-pega-.htm > Acessado em: 28/06/2015.

7. ANEXO

Esta seção compreende os códigos e conceitos implementados por outros desenvolvedores e pesquisadores que foram fundamentais para se chegar aos objetivos conquistados.

7.1. ANEXO – ARDUINO AND JAVA

Trata-se do código desenvolvido em *Java* “SerialTest.java” que possibilita a comunicação entre o *software* no computador e a arduino. Este código está acessível na página oficial do ARDUINO.CC.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import gnu.io.SerialPortEvent;
import gnu.io.SerialPortEventListener;
import java.util.Enumeration;

public class SerialTest implements SerialPortEventListener {
 SerialPort serialPort;
      /** The port we're normally going to use. */
 private static final String PORT_NAMES[] = {
                        "/dev/tty.usbserial-A9007UX1", // Mac OS X
                "/dev/ttyACM0", // Raspberry Pi
                        "/dev/ttyUSB0", // Linux
                        "COM3", // Windows
 };
 /* A BufferedReader which will be fed by a InputStreamReader
 * converting the bytes into characters
 * making the displayed results codepage independent*/
 private BufferedReader input;
 /** The output stream to the port */
 private OutputStream output;
 /** Milliseconds to block while waiting for port open */
 private static final int TIME_OUT = 2000;
 /** Default bits per second for COM port. */
 private static final int DATA_RATE = 9600;
 public void initialize() {
            // the next line is for Raspberry Pi and
            // gets us into the while loop and was suggested here was suggested
http://www.raspberrypi.org/phpBB3/viewtopic.php?f=81&t=32186
            System.setProperty("gnu.io.rxtx.SerialPorts", "/dev/ttyACM0");
              CommPortIdentifier portId = null;
              Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();
              //First, Find an instance of serial port as set in PORT_NAMES.
              while (portEnum.hasMoreElements()) {
                        CommPortIdentifier currPortId = (CommPortIdentifier)
portEnum.nextElement();
                        for (String portName : PORT_NAMES) {
                                  if (currPortId.getName().equals(portName)) {
                                            portId = currPortId;
                                            break;
                                  }
                        }
            }
```

```
                if (portId == null) {
                        System.out.println("Could not find COM port.");
                        return;
                }

                try {
                        // open serial port, and use class name for the appName.
                        serialPort = (SerialPort) portId.open(this.getClass().getName(),
                                        TIME_OUT);

                        // set port parameters
                        serialPort.setSerialPortParams(DATA_RATE,
                                        SerialPort.DATABITS_8,
                                        SerialPort.STOPBITS_1,
                                        SerialPort.PARITY_NONE);

                        // open the streams
                        input = new BufferedReader(new
InputStreamReader(serialPort.getInputStream()));
                        output = serialPort.getOutputStream();

                        // add event listeners
                        serialPort.addEventListener(this);
                        serialPort.notifyOnDataAvailable(true);
                } catch (Exception e) {
                        System.err.println(e.toString());
                }
        }

        /**
         * This should be called when you stop using the port.
         * This will prevent port locking on platforms like Linux.
         */
        public synchronized void close() {
                if (serialPort != null) {
                        serialPort.removeEventListener();
                        serialPort.close();
                }
        }

        /**
         * Handle an event on the serial port. Read the data and print it.
         */
        public synchronized void serialEvent(SerialPortEvent oEvent) {
                if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
                        try {
                                String inputLine=input.readLine();
                                System.out.println(inputLine);
                        } catch (Exception e) {
                                System.err.println(e.toString());
                        }
                }
                // Ignore all the other eventTypes, but you should consider the other ones.
        }
```

```
        public static void main(String[] args) throws Exception {
                SerialTest main = new SerialTest();
                main.initialize();
                Thread t=new Thread() {
                        public void run() {
                                //the following line will keep this app alive for 1000
seconds,
                                //waiting for events to occur and responding to them
(printing incoming messages to console).
                                try {Thread.sleep(1000000);} catch
(InterruptedException ie) {}
                        }
                };
                t.start();
                System.out.println("Started");
        }
       }
```

7.2. ANEXO – METODO DECODEPOLYLINE

Este método é responsável por capturar a *hash* atribuída a chave *points* dentro do objeto *polyline* que a API do *google maps* retornar, após obter o código o método "*decodePolyline*" irá decodificar a informação afim de obter as coordenadas latitude e longitude entre os "*steps*" no mapa.

```
        //decode polyline pega os pontos entre steps

private List<LatLng> decodePoyline(String encoded){
    List<LatLng> pointsList = new ArrayList<LatLng>();
    int index =0, len = encoded.length();
    int lat =0, lng =0;

    while (index <len){
        int b, shift =0, result =0;
        do{
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        }while (b >= 0x20);
        int dLat = ((result & 1) != 0 ? ~(result >> 1): (result >>
1));
        lat += dLat;

        shift = 0;
        result =0;
        do{
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        }while (b >= 0x20);
        int dLng = ((result & 1) != 0 ? ~(result >> 1): (result >>
1));
        lng += dLng;


        LatLng p  = new LatLng(((double)lat / 1E5)),(((double)lng
/1E5)));
        Log.i("Script", "POL: LAT: "+p.latitude+"| LNG"+p.longitude);
        pointsList.add(p);
    }
    return  pointsList;
}
```

8. APÊNDICE

Esta seção compreende alguns códigos e resultados implementados pelo autor da obra, que foram fundamentais para se chegar aos objetivos conquistados.

8.1. CÓDIGO SOFTWARE ARDUINO

Abaixo segue o código implementado par a placa Arduino que possibilita o sensor realizar a leitura de um objeto e mensurar a que distancia se encontra do sensor.

```
const int iGatilhoPin = 9; const int iEchoPin = 8;
float fTempo, fDistancia_cm, fLaminaDagua;
void setup() {
  Serial.begin(9600); // Inicia a porta serial
  pinMode(gatilhoPin, OUTPUT); // define o pino 9 como saida (envia)
  pinMode(echoPin, INPUT); //define o pino 8 como entrada (recebe)
}
void loop() {
  fLaminaDagua = 100;
  digitalWrite(iGatilhoPin, LOW);
  delayMicroseconds(2);
  digitalWrite(iGatilhoPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(iGatilhoPin, LOW);
  delayMicroseconds(2);
  fTempo = pulseIn(iEchoPin, HIGH);
  fDistancia_cm = (fTempo / 58);
  fLaminaDagua = (fLaminaDagua - fDistancia_cm);
  Serial.print("A altura da lamina em cm e: ");
  Serial.println(fLaminaDagua);
  delayMicroseconds(1000);
}
```

8.2. JSON RETORNADO DOS SERVIDORES GOOGLE PARA A APLICAÇÃO ANDROID

O JSON apresentado logo abaixo contém o retorno das coordenadas latitude e longitude do *google maps* existentes entre o ponto de origem e o ponto de destino definido nos testes seção 4.5.3.


```
"routes" : [
   {
         "steps" : [
            {
            "polyline" : {
            "points" : "bo{}@
            },
   "polyline" : {
   "points" : "xuz}@rfyeHCwAAo@"
   },
   "polyline" : {
   "points" : "ruz}@jbyeH?SiE@aE@"
        },
       }
       ]
}
```