UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DYEGO CANTÚ

SISTEMA WEB PARA MONITORAMENTO DE SENSORES DE TEMPERATURA E UMIDADE

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2013

DYEGO CANTÚ

SISTEMA WEB PARA MONITORAMENTO DE SENSORES DE TEMPERATURA E UMIDADE

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO

2013

ATA Nº: **220**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **DYEGO CANTú**.

Às 18:10 hrs do dia 19 de setembro de 2013, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Robison Cris Brito (Convidado) e Fábio Favarim (Convidado), para avaliar o Trabalho de Diplomação do aluno Dyego Cantú, matrícula 608874, sob o título **Sistema WEB para Monitoramento de Sensores**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 18:40 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Robison Cris Brito, M.Sc.
Convidado

Prof. Fábio Favarim, Dr.
Convidado

Prof.Eliane Maria De Bortoli Fávero, M.Sc.
Coordenadora do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

Bonito é melhor que feio.
Explícito é melhor que implícito.
Simples é melhor que complexo.
Complexo é melhor que complicado.
Linear é melhor do que aninhado.
Esparso é melhor que denso.
Legibilidade conta.
Casos especiais não são especiais o bastante para quebrar as regras.
Ainda que praticidade vença a pureza.
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados.
Diante da ambiguidade, recuse a tentação de adivinhar.
Deveria haver um– e preferencialmente só um –modo óbvio para fazer algo.
Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
Agora é melhor que nunca.
*Embora nunca frequentemente seja melhor que *já*.*
Se a implementação é difícil de explicar, é uma má ideia.
Se a implementação é fácil de explicar, pode ser uma boa ideia.
Namespaces são uma grande ideia – vamos fazer mais desses!

(O Zen do Python, por Tim Peters)

AGRADECIMENTOS

Agradeço a todos os gigantes que emprestaram seus ombros para que este trabalho fosse realizado.

À minha orientadora, Profa. Beatriz Terezinha Borsoi, por embarcar comigo nesta jornada, por me desculpar pelos atrasos, e pelo carisma. Não é a toa que os alunos adoram suas aulas!

Aos professores Fábio Favarim e Robison Cris Brito, pela disponibilidade e apoio que sempre me deram, e pelas críticas construtivas feitas em torno deste trabalho.

À minha família, por acreditar neste trabalho e entender as minhas ausências, ou, 'porque me tranquei no quarto'.

A minha namorada Juliane Cristina Dal Magro, por compartilhar todos os momentos comigo e me dar a certeza que tudo isso vale a pena.

Aos Hackers, que inspiram e divertem o mundo através da tecnologia. Python e Arduino são como 'aquele brinquedo' que todo developer gostaria de ganhar!

RESUMO

CANTÚ, Dyego. Sistema web para monitoramento de sensores de temperatura e umidade. 2013. 87 f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2013.

A Internet e os sistemas microcontrolados são dois exemplos evidentes dos avanços da tecnologia. A Internet possibilita a comunicação globalizada e o uso de aplicativos sem necessidade de instalação do software na máquina do usuário. Os sistemas no padrão cliente/servidor não mais necessitam de redes privadas e o acesso aos aplicativos é realizado por um navegador *web*, embora plugins e suporte a *scripts*, por exemplo, possam ser necessários para a execução da aplicação. Os sistemas microcontrolados possibilitam a automação de equipamentos e periféricos. O software contido nesses dispositivos pode executar tarefas predefinidas, como, por exemplo, as atividades pré-programadas de um forno micro-ondas; ou atuar de acordo com dados coletados do ambiente, como, por exemplo, fechar uma janela se começar a chover. Os dados coletados do ambiente por meio de sensores podem ser transmitidos pela Internet e manipulados por uma aplicação *web*. A manipulação se refere a esses dados serem apresentados ao usuário ou permitir ações em outros dispositivos, como acionar um atuador para, por exemplo, ligar ou desligar um equipamento. Considerando o amplo uso da Internet e as possibilidades que têm sido oferecidas por dispositivos microcontrolados e sistemas embarcados (que possuem o software incorporado ao hardware), neste trabalho um sistema web é desenvolvido utilizando o framework Django em linguagem Python para interfaceamento com sensores por meio de um sistema microcontrolado denominado Arduino. É uma forma de exemplificar o uso da tecnologia Arduino.

Palavras-chave: Microcontrolador Arduino. Sistema *web*. Linguagem Python. Linguagem Arduino.

ABSTRACT

CANTÚ, Dyego. Web system for monitoring temperature and humidity sensor. 2013. 87 f. Monografia de Trabalho de Conclusão de Curso - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Pato Branco, 2013.

The Internet and microcontroller systems are two evident examples of technology advances. The Internet enables the global communication and the use of applications without installation of software on the user's machine. Systems in the standard client/server no longer need access to private networks and applications is done by a web browser, though plugins and scripts support, for example, may be required for the application execution. The microcontroller systems enable the automation equipment and peripherals. The software contained in these devices can perform predefined tasks, for example, pre-programmed activities of a microwave oven, or act according to data collected from the environment, for example, close a window if it starts raining. The data collected from the environment through sensors can be transmitted over the Internet and manipulated by a web application. This manipulation refers to the data being presented to the user or enable actions in other devices such as trigger an actuator, for example, connecting or disconnecting a device. Considering the widespread use of the Internet and the possibilities that have been offered by devices microcontroller and embedded systems (which contain the software embedded in the hardware), in this work a web system is developed using the Django framework by Python language for interfacing with sensors through a system microcontroller called Arduino, thus characterizing a way to exemplify the use of this technology.

Key-words: Arduino. Web system. Python.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

ASP	*Active ServerPage*
CGI	*Common Gateway Interface*
CSS	*Cascading Stype Sheet*
DRY	*Don't Repeat Yourself*
EDA	*Electronic Design Automation*
EEPROM	*Electrically Erasable Programmable Read Only Memory*
EPROM	*Erasable Programmable Read Only Memory*
GLP	*General Public License*
GNOME	*GNU Object Model Environment*
HTML	*HyperText Markup Language*
IBSG	*Cisco's Internet Business Solutions Group*
IDE	*Integrated Development Environment*
IoT	Internet of Things
ISTAG	*European Commission's IST Advisory Group*
LED	*Light-Emitting Diode*
MTV	*Model-Template-View*
MVC	*Model-View-Controller*
MXML	*Maximum Experience Markup Language*
PCB	*Printed Circuit Board*
RAM	*Random Access memory*
RFID	*Radio-Frequency Identification*
RIA	*Rich Internet Applications*
SGBD	*Sistema gerenciador de banco de dados*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UCP	*Unidade Central de Processamento*
UML	*Unified Modeling Language*
URL	*Universal Resource Locator*
USB	*Universal Serial Bus*
W3C	*World Wide Web Consortium*
WSGI	*Web Server Gateway Interface*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, com o contexto do trabalho, os seus objetivos e a justificativa. Por fim está a estrutura do texto por meio da apresentação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

No ano de 2008, o número de dispositivos conectadas à Internet excedeu o número de pessoas existentes no planeta Terra (CISCO, 2012). E, segundo a previsão da *Cisco's Internet Business Solutions Group* (IBSG) haverá cerca de 15 bilhões de dispositivos conectados até o ano de 2015. Eles não são apenas *smartphones* e *tablets*, mas todo dispositivo capaz de transmitir dados pela rede, desde pequenos sensores para monitoramento de um ambiente a microcontroladores com Inteligência Artificial.

O vínculo da Internet com sistemas microprocessados possibilita o conceito de “Internet das coisas”, do termo original em inglês “Internet of Things” (HOMPEL, 2005 citado em HRIBERNIK et al. 2011). Esse conceito extrapola a ideia da Internet como uma rede de computadores global e interconectada, para descrever uma rede de coisas (dispositivos) interconectados, tais como objetos do dia a dia e produtos e objetos do ambiente (HRIBERNIK et al. 2011). Assim, a Internet das coisas representa uma base comum para diversas áreas recentes de desenvolvimento, como, por exemplo, Inteligência do Ambiente (DUCATEL et al. 2001), Computação Ubíqua (WEISER, 1991), Computação Pervasiva (GUPTA et al. 2001) e Auto Identificação (COLE, ENGELS, 2002). Como aspecto central desse conceito está a ideia que objetos (coisas) são capazes de processar informação, comunicar-se com outros objetos e com o ambiente e tomar decisões autonomamente (HRIBERNIK et al. 2011).

Tecnicamente a arquitetura da Internet das Coisas é baseada em ferramentas

de comunicação de dados. O objetivo da Internet das Coisas é facilitar a troca de informações sobre, entre outros, bens em uma rede de fornecimento global, isto é, a infraestrutura de tecnologia de informação deveria prover informações sobre essas “coisas” de maneira segura e confiável (WU et al., 2010). Expandido o escopo inicial, a Internet das Coisas poderia também servir como uma infraestrutura para a computação ubíqua, possibilitando aos ambientes reconhecer e identificar objetos e reter informações a partir da Internet para facilitar suas funcionalidades adaptativas (WEBER, 2009).

Esses sistemas são geralmente denominados de sistemas embarcados, quer conectados por meio de Internet, de outras formas de rede ou mesmo com funcionamento isolado (*standalone*). Os sistemas embarcados executam funções específicas para as quais eles foram programados. Eles estão presentes na maioria dos equipamentos eletrônicos. Esses equipamentos também podem interagir com sensores e atuadores obtendo dados do ambiente e interagindo, ou seja, enviando instruções ou comandos para outros dispositivos. Dentre as linguagem de programação utilizadas para desenvolver o software incorporado ao microcontrolador está a Assembly e C/C++.

Como forma de exemplificar a interação destes dispositivos conectados à Internet, este trabalho apresenta o desenvolvimento de um sistema *web* para o monitoramento de dados de um ambiente através do interfaceamento com uma plataforma microcontrolada responsável pela leitura de sensores de temperatura e umidade.

1.2 OBJETIVOS

O objetivo geral apresenta o resultado principal obtido com o desenvolvimento deste trabalho e os objetivos específicos o complementam.

1.2.1 Objetivo Geral

Desenvolver um sistema *web* usando o framework Django e linguagem Python para monitoramento de sensores através do interfaceamento com o kit de desenvolvimento Arduino.

1.2.2 Objetivos Específicos

São objetivos específicos deste trabalho:

- Exemplificar a forma de uso de sensores com o kit Arduino;
- Apresentar a linguagem de programação Python e o uso do *framework* Django no desenvolvimento ágil de sistemas web;
- Exemplificar o uso da linguagem Python, através do módulo pySerial para interação com o Arduino por meio de comunicação serial;
- Exemplificar o uso da biblioteca JavaScript Flot para apresentação de dados e geração de gráfico dinâmico.

1.3 JUSTIFICATIVA

Os microntroladores sob a forma de sensores e atuadores, por exemplo, possibilitam interação com objetos do ambiente de maneiras distintas e diversas. Essa interação pode ocorrer em aplicações industriais e mesmo domésticas. Na indústria, o amplo emprego que a robótica têm nas linhas de montagem industrial é um claro e efetivo exemplo de uso de microcontroladores. Nas residências as aplicações estão relacionadas a abertura e fechamento de janelas e portas, ligar e desligar lâmpadas, no controle de umidade (irrigação) de plantas, dentre muitos outros.

Sistemas *web* para interfacear dados provenientes de sistemas

microcontrolados facilita o acesso remoto aos dados que esses dispositivos coletam. Esse acesso é independente de software específico estar instalado, bastando o uso de um navegador *web*.

Este trabalho se justifica também pela oportunidade de exemplificar a interação com dispositivos conectados a Internet através do desenvolvimento de um sistema web para interfaceamento com uma plataforma microcontrolada. Essa plataforma é responsável pela leitura de sensores instalados em um ambiente.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta a introdução, com as considerações iniciais, os objetivos e a justificativa. O Capítulo 2 contém o referencial teórico e está dividido em aplicações *web* e microcontroladores. No Capítulo 3 estão os materiais e a sequência de passos para modelar e exemplificar o desenvolvimento com Arduino. No Capítulo 4 está o sistema modelado e implementado, que é o resultado deste trabalho. E por fim, no Capítulo 5, está a conclusão.

2 REFERENCIAL TEÓRICO

O referencial teórico está organizado em duas partes. Uma delas se refere ao desenvolvimento para ambiente Internet, são os aplicativos *web*. E a outra está centrada em sistemas microcontrolados.

2.1 APLICAÇÕES *WEB*

Os serviços *web* utilizam a infraestrutura de rede Internet para disponibilizar acesso a informação e a aplicativos. A arquitetura dos aplicativos que executam utilizando serviços Internet é denominada cliente/servidor. Um servidor pode disponibilizar acesso a um aplicativo ou conter hospedadas páginas de texto a serem exibidas no cliente. O cliente é o aplicativo denominado navegador para acesso ao conteúdo de páginas *web* ou para prover acesso a aplicativos desenvolvidos para execução na Internet.

A conexão entre o cliente e o servidor é, geralmente, realizada pelo protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*), pelo menos nas aplicações denominadas *web* tradicionais que são as baseadas em HTML (*HyperText Markup Language*). Em termos de serviços, de um lado está o cliente *web* (browser ou navegador) que solicita dados ao servidor *web*, que recebe as respostas, formata a informação e a apresenta ao usuário; do outro lado está o servidor *web* que recebe as requisições, lê dados armazenados no banco de dados do servidor, executa instruções e retorna dados para o cliente. Esses dados são apresentados ao cliente de diversas formas, como páginas HTML, seja puro ou agregado por *scripts* e outros, ou como componentes MXML (*Maximum Experience Markup Language*), por exemplo, que definem as interfaces para aplicações *web* denominadas RIA (*Rich Internet Applications*).

Em uma aplicação *web*, as páginas apresentadas ao usuário são dinâmicas. Assim denominadas porque permitem interação com o usuário. O usuário insere informações por meio do navegador utilizando formulários HTML ou definidos por

outras tecnologias. Essas informações são enviadas ao servidor *web* que executa instruções que compõem a aplicação. A partir do resultado da execução dessas instruções são compostas as páginas que são apresentadas ao cliente. Essa composição pode ser simples como a atualização de campos ou a abertura de outra página, ou mais complexa como a apresentação de um relatório a partir de parâmetros indicados pelo usuário por meio de um formulário.

Ressalta-se que em uma aplicação *web* não é somente o servidor quem realiza processamento. O *browser* pode executar programas denominados *scripts* que têm a capacidade de perceber os eventos originados pelo usuário e responder aos mesmos. Esses *scripts* são incorporados no código HTML e o seu código fonte pode ser visualizado a partir da própria página exibida pelo navegador. Além dos *scripts* podem ser executados aplicativos em linguagem de programação normalmente usadas para programas no lado servidor, como as linguagens *PHP Hypertext PreProcessor (*PHP) e Java.

As primeiras páginas *web* desenvolvidas eram estáticas. Sem animações e baseadas em cópias eletrônicas de texto. Na época, páginas estáticas eram adequadas considerando o objetivo de troca ou disponibilização de informação. Porém, com a utilização comercial da Internet, houve a necessidade de as páginas se tornarem dinâmicas e várias tecnologias surgiram e continuam surgindo para tornar a *web* um ambiente mais agradável, interativo e amigável (O'REILLY, 2005).

Atendendo aos padrões estabelecidos pela W3C (*World Wide Web Consortium*), as páginas de Internet que eram apenas texto, passaram a utilizar HTML. HTML é uma linguagem de marcação para apresentação de conteúdo, utilizando fontes, cores, recursos de formatação e imagens. As páginas definidas essencialmente com HTML eram estáticas. As animações ocorriam por meio de imagens animadas (*gifs* animados) ou textos que representavam o efeito de "piscar" e que se movimentavam na tela.

Uma das primeiras tecnologias utilizadas para tornar as páginas dinâmicas foi o CGI (*Common Gateway Interface*). GCI possibilitava, por meio do uso de *scripts*, executar tarefas no lado do servidor e apresentar o resultado para o cliente na Internet.

Complementado os GCIs diversas tecnologias de *scripts* surgiram para

executar no computador que acessa o sistema, o cliente, ou que executam no servidor, tais como: JavaScript, Servlets, ASP (*Active ServerPage*), PHP, JavaServer Page e VBScript.

A Figura 1 apresenta diversas tecnologias utilizadas para o desenvolvimento *web*. O relacionamento entre essas tecnologias permite entender o surgimento, as origens, das mesmas.

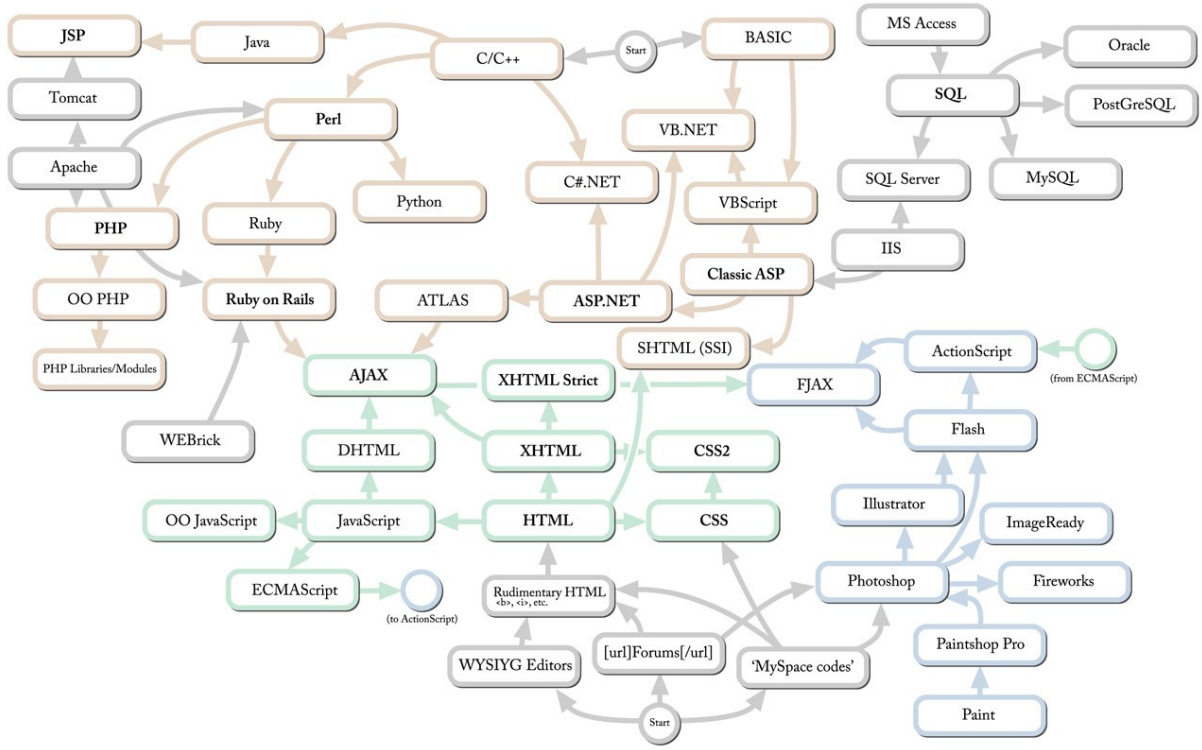

Figura 1 - Tecnologias para desenvolvimento web

Fonte: Brown (2006).

Brown (2006) ressalta que o projeto e o desenvolvimento de aplicações *web* é um campo complexo. Há centenas de tecnologias, linguagens de programação, padrões e ferramentas que podem ser utilizados. Isso pode ser verificado pela representação da Figura 1. A escolha da tecnologia ou das tecnologias a serem utilizadas depende de uma série de fatores incluindo a segurança necessária, os recursos oferecidos e a capacidade de investimento da empresa.

2.2 SISTEMAS MICROCONTROLADOS

As aplicações embarcadas possuem características diferenciadas tanto em *hardware*, que leva em consideração uma arquitetura miniaturizada de baixo consumo de energia e podendo operar em baixas frequências, quanto em *software*, que ao invés de interagir diretamente com usuários, é desenvolvido para controle de processos e manipulação dos recursos do dispositivo no qual atua.

Os sistemas embarcados são encontrados em praticamente todos os dispositivos eletrônicos digitais, como, por exemplo: celulares, sistemas de *airbag*, equipamentos para redes (roteadores, *modems*, *switches*), eletrodomésticos (máquinas de lavar, fornos micro-ondas) e equipamentos de controle industrial.

Um microcontrolador é um circuito integrado que em sua estrutura interna pode conter memória RAM (R*andom Access Memory*), *flash* e EEPROM *(Electrically Erasable Programmable Read Only Memory), timers*, contadores, conversores analógico/digital e protocolos de transmissão de dados (NASCIMENTO, 2009). O uso de microntroladores define o conceito de produtos denominados inteligentes.

2.2.1 Produtos Inteligentes

Produtos inteligentes são itens físicos, que podem ser transportados, processados ou usados e que possuem a habilidade de agir de forma inteligente (HRIBERNIK et al., 2011). McFarlane et al. (2003) definem produto inteligente como uma representação de um item baseada em informação ou aspectos físicos. Essa representação possui uma identificação única e é capaz de comunicar-se efetivamente com o ambiente, pode reter ou armazenar dados sobre si mesmo, implantar uma linguagem para apresentar suas características, requisitos de produção e etc.; além disso é capaz de tomar decisões relevantes para seu próprio destino.

O grau de inteligência desses produtos pode ser variado, de processamento de dados simples a comportamento pró-ativo complexo. De acordo com Hribernik et al. (2011) esse é o foco da definição constante em McFarlane et al. (2003) e

Kärkainen et al. (2003). Três dimensões que caracterizam produtos inteligentes são sugeridas por Meyer et al. (2009):

a) Nível de inteligência – descreve se o produto apresenta capacidade de tomar decisões, notificar problemas ou manipular informações;

b) Localização da inteligência – mostra se a inteligência é construída no objeto ou se está localizada na rede;

c) Nível de agregação de inteligência – descreve se o próprio item é inteligente ou se a inteligência é agregada a partir de outros itens.

Produtos inteligentes têm se mostrado aplicáveis em vários cenários e modelos de negócio. Por exemplo: Kärkkäinen et al. (2003) e Ventä (2007) descrevem a aplicação em problemas de gerenciamento de informação em cadeias de suprimentos; McFarlane, et al. (2003) no controle da manufatura; Wong, et al. (2002) na logística de armazenamento, distribuição e produção. Outro exemplo de uso é a casa inteligente proposta em Hribernik et al. (2011).

Usando o Arduino, Hribernik et al. (2011) apresentam um estudo de caso com o objetivo de criar uma casa inteligente. Os objetos do mundo real foram conectados ao mundo virtual de maneira a monitorá-los e controlá-los. Assim, uma primeira atividade foi a identificação dos requisitos e os objetos que seriam monitorados e controlados. Todos os objetos a serem manipulados foram equipados com sensores e atuadores. Nesse estudo de caso, os objetos controlados foram: porta controlada com RFID, interruptor de lâmpadas automático, controle de temperatura, controle das janelas e sensor para controlar a irrigação das plantas (umidade do substrato do vaso). Esses objetos estão numerados na Figura 2 e o controle foi realizado com o Arduino. Esses objetos foram integrados em uma arquitetura de um sistema *web.*

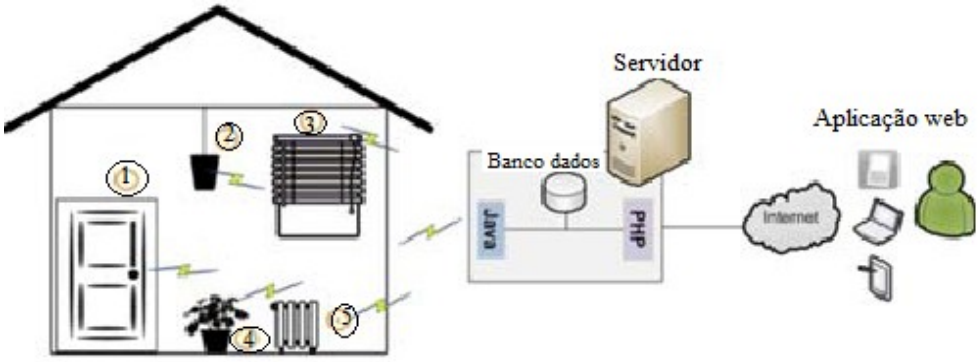

Figura 2 - Visão geral da casa inteligente

Fonte: Hribernik et al. (2011, p. 5)

As partes numeradas na Figura 2 representam:

1) Abridor de portas elétricas, leitor RFID;

2) Led, sensor de luz;

3) Servo motor, sensor de luz;

4) Pote de planta, sensor de umidade;

5) Aquecedor/ventilador, sensor de temperatura.

Uma casa inteligente representa um ambiente inteligente. Esse conceito foi introduzido em 2001 pela ISTAG (*European Commission's IST Advisory Group*) (KO, RAMOS, 2010) e provê uma visão da sociedade da informação onde pessoas são cercadas por interfaces intuitivas e inteligentes. Essas interfaces são incorporadas em todo tipo de objetos e estão inseridas em um ambiente que é capaz de reconhecer e responder a presença de diferentes indivíduos de forma não intrusiva. De certa forma, um ambiente inteligente refere-se a um ambiente digital que proativamente, mas sensivelmente, auxilia as pessoas no dia a dia de suas vidas (RAMOS, AUGUSTO, SHAPIRO, 2008).

2.2.2 Microcontroladores

Um microcontrolador é um elemento eletrônico, desenvolvido para executar tarefas específicas (NICOLOSI, 2000). Já Sica (2006) define um microcontrolador como um computador-num-*chip*. O propósito de um microprocessador é executar tarefas específicas armazenadas em sua memória (NICOLOSI, 2000).

Um microcontrolador contém basicamente processador, memória e funções de entrada/saída. Além dos componentes lógicos e aritméticos, usuais de um microprocessador de uso geral, eles podem incluir elementos adicionais como memória RAM, E*rasable Programmable Read Only Memory (EPROM)*/EEPROM, *flash*, dispositivos de entrada e saída, que podem ser desde um simples pino digital do componente a uma interface *Universal Serial Bus (USB)* ou Ethernet e controladores de interrupções para interfaces paralelas e seriais. Um microcontrolador típico tem instruções de manipulação de *bits*, acesso fácil e direto

aos dispositivos de entrada e saída e processamento eficiente de interrupções (MARINHO, MARINHO, 2001). O *software* escrito para sistemas embarcados, é muitas vezes chamado de *firmware* (SICA, 2006).

A Figura 3 é uma representação esquemática de um microcontrolador.

Figura 3 - Partes de um microcontrolador

Fonte: Chase (2007) citado em Nascimento (2009, p. 34).

As partes de um microcontrolador, como indicado na Figura 3, são (NASCIMENTO, 2009): unidade de memória, unidade de processamento, barramentos, unidade de entradas e saídas e Comunicação serial dos microcontroladores.

a) Unidade de memória – possui a função de armazenar os dados. Cada parte da memória tem seu endereço respectivo. Dois conceitos básicos estão envolvidos: endereçamento e memória. A memória é o conjunto de todos os locais de armazenamento de registros. Endereçamento é a identificação desses locais de memória, possibilitando buscar (ler o conteúdo) ou armazenar (escrever) um conteúdo em um local de memória. Isso significa que é possível saber o valor armazenado em um determinado endereço de memória, mais especificamente saber o conteúdo armazenado em cada local de memória que é individualizado por um endereço.

b) Unidade Central de Processamento (UCP) – a UCP controla todas as funções do sistema (MARINHO, MARINHO, 2001). A UCP de qualquer sistema computacional contém os seguintes grupos de unidades funcionais (NASCIMENTO,

2009):

b.1) Registradores – os registradores são utilizados para armazenamento temporário de dados de utilidade interna ou externa à UCP (NICOLOSI, 2000).

b.2) Contadores – são indicadores de endereço de memória externa. Um contador endereça a próxima instrução a ser lida (NICOLOSI, 2000).

b.2) Unidade lógica e aritmética – é a unidade funcional da UCP que executa as funções lógicas e aritméticas entre palavras binárias, gerando uma outra palavra na saída.

b.3) Unidade de controle e sincronização – essa unidade coordena e controla todas as unidades funcionais em uma sequência lógica e sincronizada.

A memória e a UCP estão interligadas. Quando a UCP solicita um registro que está alocado em algum endereço de memória ou para armazenar valor em endereços de memória, utiliza um "caminho" para transferência dos dados que é chamado de barramento ou *bus.*

c) Barramentos – é por eles que trafegam as informações e eles interligam todos os componentes que formam um microcontrolador, também é chamado de "*bus*". Existem três tipos de *bus* (NOBREGA FILHO, 1999):

c.1) O Barramento de dados (*Data bus*) – transmite dados entre as unidades. Portanto, um microprocessador de 8 bits requer um barramento de dados de 8 linhas para transmitir dados de 8 bits em paralelo. Semelhantemente, um microprocessador de 64 bits necessita de um barramento de dados de 64 linhas para transmitir dados de 64 bits em paralelo. Se o barramento de dados para um microprocessador de 64 bits fosse formado por 8 linhas, seriam necessárias oito transmissões sucessivas, tornando o sistema mais lento. O Barramento de dados é bi-direcional, isto é, pode transmitir em ambas as direções.

c.2) O Barramento de endereço (*Address bus)* – é usado para selecionar a origem ou o destino dos sinais transmitidos em um dos outros barramentos ou em uma de suas linhas. Esse barramento conduz endereços. Uma função típica do barramento de endereço é selecionar um registrador em um dos dispositivos do sistema que é usado como a fonte ou o destino do dado.

c.3) O Barramento de controle (*Control bus*) – sincroniza as atividades do sistema. Esse barramento conduz o *status* e a informação de controle de/para o

microprocessador.

Os barramentos podem ser posicionados como parte do circuito no próprio *chip* (barramentos internos) ou podem servir de comunicação externa entre os *chips* (barramentos externos). Os barramentos externos podem ser expandidos para facilitar a conexão de dispositivos especiais.

d) Unidade de entradas e saídas – a função desta unidade é ler (entrada) ou escrever (saída) um sinal digital (0 ou 1) em um pino de microcontrolador. É comum os microcontroladores oferecerem uma grande quantidade de entradas e saídas digitais. Registradores de configuração permitem definir os pinos usados para entrada, para saída e para outras funções. As entradas e saídas de um microcontrolador podem ser classificadas como:

d.1) Entradas digitais – Um sinal digital só lê valores compreendidos entre “0 e 1”. Um exemplo típico do uso de entradas digitais é para detectar se um botão está pressionado (QUADROS, 2008). No lugar do botão é possível ter qualquer tipo de sensor ou dispositivo com uma saída digital.

d.2) Saídas digitais – um uso básico de uma saída digital é acionar um LED (*Light-Emitting Diode*) (QUADROS, 2008). A capacidade de acionamento de uma saída digital normalmente é limitada; um transistor ou um relê pode ser utilizado para controlar cargas maiores.

d.3) Entradas analógicas – uma entrada analógica fornece um número que depende da tensão que está presente no pino correspondente (QUADROS, 2008). A quantidade de *bits* presente no número define a precisão da conversão; valores típicos estão correspondidos entre 8, 10, 12 e 16 bits. As aplicações são as mais variadas, como, por exemplo: verificar o nível da alimentação de uma bateria; verificar o nível de sinal recebido, para circuitos com comunicação via rádio; verificar o nível de água de um recipiente; e verificar o pH da água. Nas entradas e saídas analógicas, é comum que microcontroladores utilizem conversores para transformar um sinal analógico em digital. Nos comparadores analógicos, o microcontrolador indica somente se a tensão do pino é maior ou menor que um valor (fixo) armazenado em uma variável interna.

d.4) Saídas analógicas – em uma saída analógica a tensão de um pino é controlada escrevendo um valor em um registrador. O microcontrolador possui um

circuito de conversão de sinal digital para analógico (D/A) e utiliza o valor do registrador para aplicar uma tensão no pino configurado. Desta forma pode-se controlar a intensidade de luz de um LED ou a velocidade de um motor, por exemplo.

e) Comunicação serial dos microcontroladores – os microcontroladores se comunicam basicamente da forma serial. Segundo Matic (2003) eles trabalham apenas com três linhas: uma para enviar dados, outra para receber esses dados e a terceira é usada como referência para envio e recebimento dos dados. Esse conjunto de regras forma o protocolo RS-232. A comunicação RS-232 é chamada serial devido aos dados se moverem *bit* a *bit* em série. Em uma transmissão em paralelo, os dados podem ser enviados em palavras de 4, 8, 16, 32 bits ou outros múltiplos simultaneamente ao receptor.

A Assembly foi uma das primeiras linguagens utilizadas para a programação de microntroladores. A linguagem Assembly é orientada a máquina (para o processador). Essa linguagem utiliza instruções de baixo nível que operam diretamente com registradores e endereços de memória, ou seja, as instruções são executadas diretamente pelo processador. Atualmente há microcontroladores desenvolvidos para executarem software em linguagens de alto nível, como a linguagem C ou variações dessa linguagem, como a Arduino.

2.2.3 Arduino

Arduino pode ser denominado como um kit de desenvolvimento de prototipagem eletrônica de código fonte aberto que se baseia em *software* e *hardware* flexíveis e fáceis de usar (ARDUINO, 2012). É destinado à criação de objetos ou ambientes interativos. Seu intuito é ser uma tecnologia de baixo custo, que facilite a aprendizagem e o desenvolvimento de projetos e que possa ser estudada e modificada quando necessário. O projeto Arduino possui um site (www.arduino.cc) com manuais, bibliotecas, exemplos de códigos e conta com uma comunidade com diversas iniciativas em prática.

Arduino é uma plataforma de desenvolvimento de código aberto baseada na

família de microntroladores ATMega (MASSIMO, CUARTIELLES, 2012). Arduino pode receber dados do ambiente por meio de sensores e interagir com o meio por meio do controle de atuadores (AL-KUWARI et al., 2011), como motores e lâmpadas, por exemplo. Um exemplo de uso para o Arduino foi apresentado no evento Campus Party de 2012. Um jovem chileno de 14 anos de idade utilizando um sensor de terremotos, que custou US$ 50, ligado a um Arduíno e a um servidor criou um sistema para envio de mensagens ao Twitter para avisar de terremotos. O sistema emite alerta com cinco a trinta segundos de antecedência de ocorrência do evento (CARDOSO, 2012).

O microcontrolador é programado usando a linguagem de programação Arduino e um ambiente de desenvolvimento. Os projetos Arduino, incluem o *hardware* e o *software* desenvolvido para a funcionalidade específica, podem atuar sozinhos, no modelo *standalone*, ou comunicar-se com outros *software* em execução em um computador.

Para Hodges et al. (2012) Arduino tem se tornado o padrão de plataforma aberta para prototipagem física. Interação com objetos físicos por meio do controle de sensores e atuadores que estão conectados a microntroladores é chamada de computação física. E Arduino é uma plataforma amplamente aceita para computação física (KATO, 2010). Inclusive porque é de aprendizado fácil e amplamente usado em ensino, em pesquisa e por *hobby* (HODGES et al. 2012).

As placas Arduino podem ser construídas manualmente ou compradas pré-montadas. Os projetos de referência do *hardware* são disponibilizados sob a forma de licença *open-source*. E sendo dessa forma, o projeto pode ser adaptado livremente (ARDUINO, 2012).

Existem diversas plataformas de desenvolvimento que permitem uma prototipagem fácil e rápida de componentes de produtos inteligentes, tais como sistemas *embedded*, sistemas de posicionamento, sensores e redes sem fio (HRIBERNIK et al., 2011). Uma visão de *kits* de desenvolvimento e como eles podem ser aplicados ao usuário final pode ser encontrado em Cvijikl e Michahelles (2011). Esses autores focam na plataforma Arduino Duemilanove, que é um microcontrolador baseado em ATmega168 ou ATmega328.

2.2.3.1 Hardware do Arduino

A estrutura física do Arduino consiste em um microcontrolador com componentes complementares para facilitar o controle de entrada e saída de dados, que pode trabalhar em conjunto com um programa executado em um computador ou em modo *standalone* (autônomo). As suas especificações estão disponíveis sob licença da Creative Commons. A Figura 4 apresenta uma placa Arduino Uno.

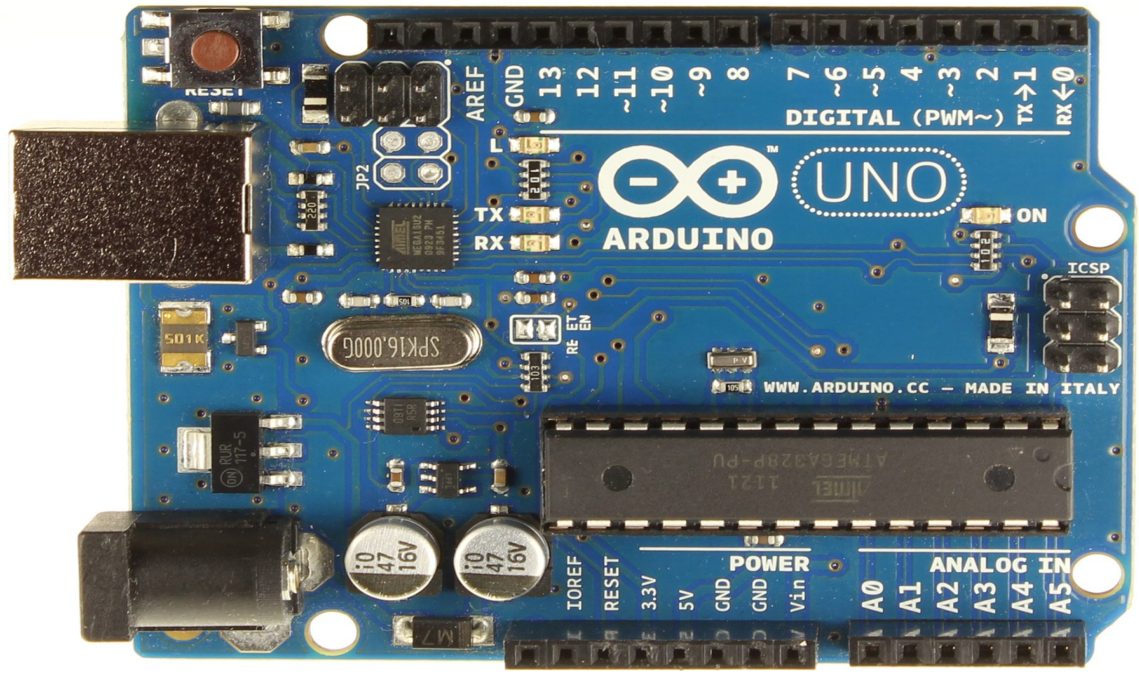

Figura 4 - Placa Arduino Uno

Fonte: http://arduino.cc/en/uploads/Main/ArduinoUno_R3_Front.jpg

O *hardware* do Arduino está organizado de forma que suas funções possam ser expandidas por meio de placas específicas (*shields*), que provêem outros recursos como conexão *ethernet* ou *wireless*. Atualmente o acesso à programação do microcontrolador é realizada por meio de conexão USB, dispensando o uso de dispositivos programadores específicos.

2.2.3.2 Software do Arduino

O ambiente de programação do Arduino consiste de uma IDE (*Integrated Development Environment*) desenvolvida em Java, que é baseada no Processing e na linguagem Arduino, derivada do Wiring, que é baseada em C/C++ (KATO, 2010). A Figura 5 apresenta o ambiente de desenvolvimento do kit Arduino com um exemplo de código escrito na linguagem de programação específica para o Arduino.

Figura 5 - IDE e Linguagem Arduino

O desenvolvimento ocorre pela criação do código do programa (*sketch*), compilação e gravação do mesmo no microcontrolador. Esse é o papel fundamental do ambiente de desenvolvimento que, utilizando ferramentas livres, verifica o código, transforma-o em C/C++ e em seguida faz a sua compilação gerando um arquivo *.hex* que será gravado no microcontrolador.

3 MATERIAIS E MÉTODO

Neste capítulo são apresentados os materiais e o método utilizados para desenvolver o sistema que é resultado deste trabalho.

3.1 MATERIAIS

A análise do sistema foi feita com o auxílio da ferramenta de criação de diagramas Dia (GNOME, 2013), a implementação com Python (PHYTON, 2013) com o banco de dados SQLite (SQLITE, 2013) e camada de aplicação através da utilização do framework Django (DJANGO, 2013). Assim como a utilização do módulo pySerial (PYSERIAL, 2013) para realização de comunicação serial.

Como interface do usuário foram utilizados recursos próprios do *framework* Django responsáveis pelo tratamento da camada de visualização. E uma biblioteca JavaScript (MOZILLA, 2013) para apresentação dos dados e geração de gráficos dinâmicos.

Os dados serão obtidos a partir da conexão serial com o kit Arduino (ARDUINO, 2013), o qual, realiza a leitura dos sensores do ambiente. O Arduino é uma plataforma embarcada que possui uma linguagem de programação própria, baseada em C/C++.

A seguir a lista das ferramentas e das tecnologias utilizadas para o desenvolvimento do projeto:

a) Dia: Para a modelagem baseada em UML (*Unified Modeling Language*) do sistema.

b) Fritzing: Ferramenta utilizada para criação do protótipo (esquema eletrônico).

c) Python: Linguagem de programação para implementação do sistema.

d) Django: *Framework web* responsável pela camada de aplicação.

e) SQLite: Como banco de dados para armazenamento e consulta das leituras.

f) Bootstrap: Framework que faz uso de HTML, *Cascading Stype Sheet* (CSS)

e Javascript para criação do front-end do sistema, camada de apresentação.

g) Flot: Biblioteca Javascript utilizada para criação de gráficos com Jquery.

h) DHT11: Sensor temperatura e umidade utilizado para obter os dados do ambiente. Foi utilizado junto com um resistor de 10k oms.

i) Arduino Uno: Kit de desenvolvimento para interação com o sensor utilizado.

j) pySerial (PYSERIAL, 2013): Módulo para realização de comunicação serial.

k) Cron: Gerenciador de tarefas do sistema operacional GNU/Linux.

O custo para desenvolvimento incluindo o Arduino Uno, o sensor DHT11 e o resistor de 10k é de aproximadamente R$100,00.

3.1.1 Dia

Dia é uma ferramenta para desenhar diversos tipos de diagramas, como diagramas UML, entidade relacionamento, fluxogramas, diagramas de rede, entre outros. Foi originalmente desenvolvido por Alexander Larsson para o ambiente de desktop GNU *Object Model Environment* (GNOME), sob licença General Public License (GPL). A Figura 6 apresenta a tela inicial do sistema e um exemplo de diagrama.

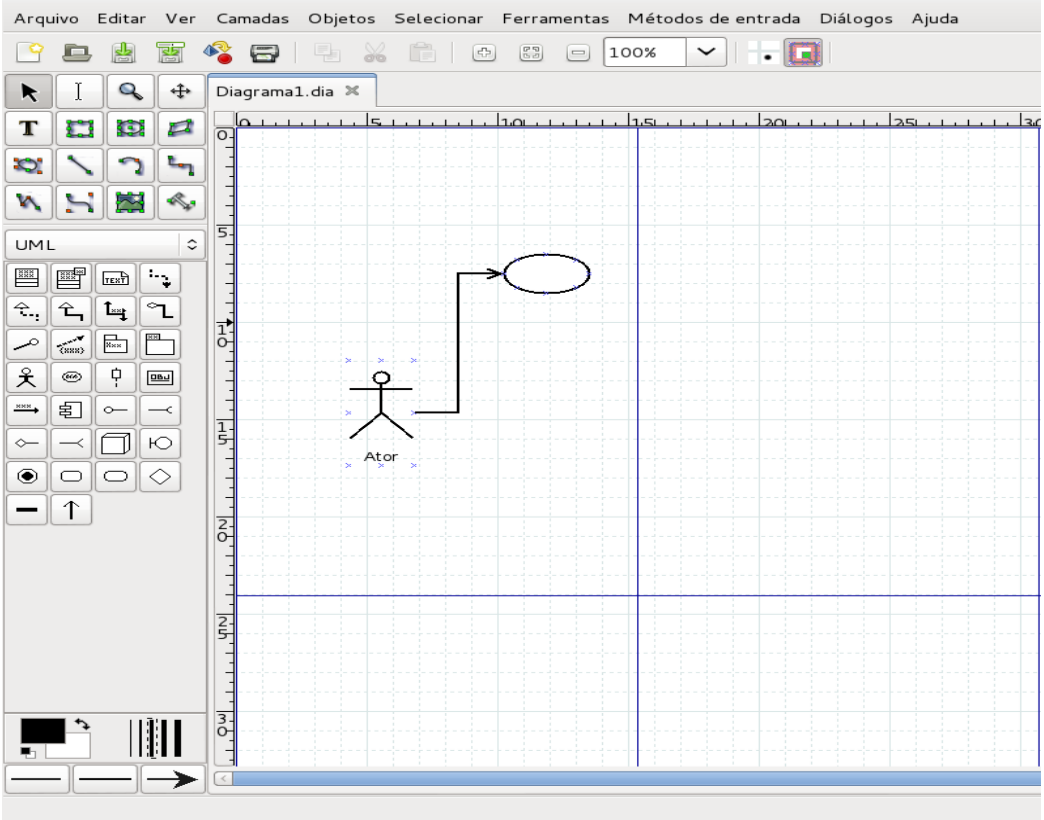

Figura 6 - Interface da ferramenta Dia

Além do menu e a barra de ferramentas, Dia, permite visualizar a caixa de objetos de diagrama de acordo com o item selecionado, neste caso foi selecionado 'UML'. Os diagramas gerados podem ser impressos ou exportados como imagem.

3.1.2 Fritzing

A ferramenta Fritzing é um software livre para modelagem de protótipos eletrônicos. Criado na Universidade de Ciências Aplicadas de Potsdam (Alemanha), a ferramenta Fritzing auxilia a criação de circuitos que integram a plataforma Arduino. A Figura 7 apresenta a *interface* dessa ferramenta.

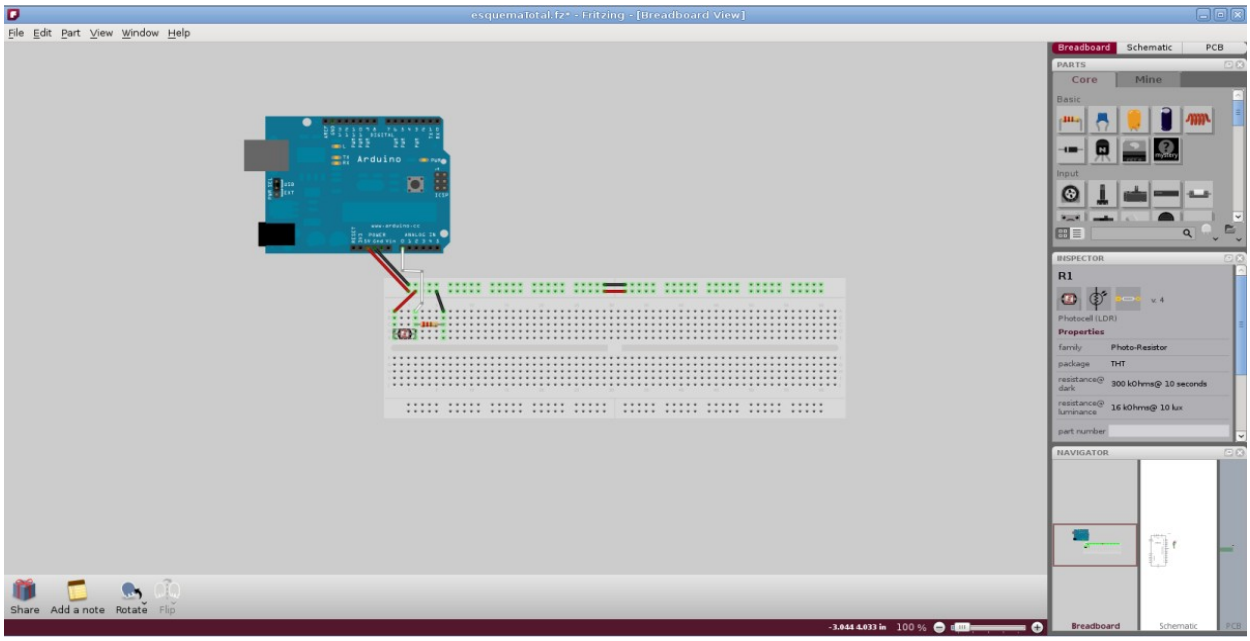

Figura 7 - Interface da ferramenta Fritzing

Além do menu de opções, destacam-se as seguintes áreas desta ferramenta (Figura 7):

a) Edição – área para a composição do projeto.

b) *Views* - na área superior a direita possibilitando a visualização do projeto, além da forma convencional EDA (*Electronic Design Automation*), também em seu esquemático (*schematic*) e PCB (*Printed Circuit Board*), facilitando a manufaturação dos circuitos do projeto.

c) *Parts* - abaixo das *Views*, que disponibiliza diversos componentes para incluir no projeto. Basta selecionar o componente desejado e arrastar para a área do projeto.

d) Inspector - seguindo abaixo da área *Parts*, cuja função é possibilitar a configuração dos componentes utilizados no projeto.

3.1.3 Linguagem Python

Python é uma linguagem de altíssimo nível (em inglês, Very High Level Language) orientada a objeto, de tipagem dinâmica e forte, interpretada e interativa

(BORGES, 2010).

O nome da linguagem origina-se do nome da série humorística britânica Monty Python's Flying Circus (SANTANA, 2010) e foi lançada em 1991 pelo holandês Guido van Rossum, sendo hoje mantida pela organização sem fins lucrativos Python Software Foundation (PSF) sob licença livre.

Originalmente implementada em linguagem C, o CPython é o padrão mais popular da linguagem Python, mas existem outros padrões como o Jython, implementado em linguagem Java, o IronPython em C#, e o PyPy, que é implementado com o próprio Python.

A linguagem Python possui recursos de introspecção, herança multipla, meta-classes, decorators e duck typing e possui uma extensa biblioteca padrão que aplica-se em diversos domínios. Além da possibilidade de utilização de pacotes de terceiros e de vários *frameworks* disponíveis para a linguagem.

3.1.4 Django

Django é um *framework* de desenvolvimento ágil para *web*, escrito em Python. Foi criado pelo grupo editorial The World Company dos Estados Unidos para a versão *web* do seus jornais. Posteriormente, em 2005, foi liberado sob licença BSD, tornando-se um software *open-source*.

Assim como outros *frameworks* ágeis, o Django utiliza o princípio DRY (*Don't Repeat Yourself*), fazendo com que o desenvolvedor faça reuso de código. O *framework* utiliza o padrão de arquitetura MTV (*Model-Template-View*), o qual é similar ao MVC (*Model-View-Controller*) e é apresentado na Figura 8.

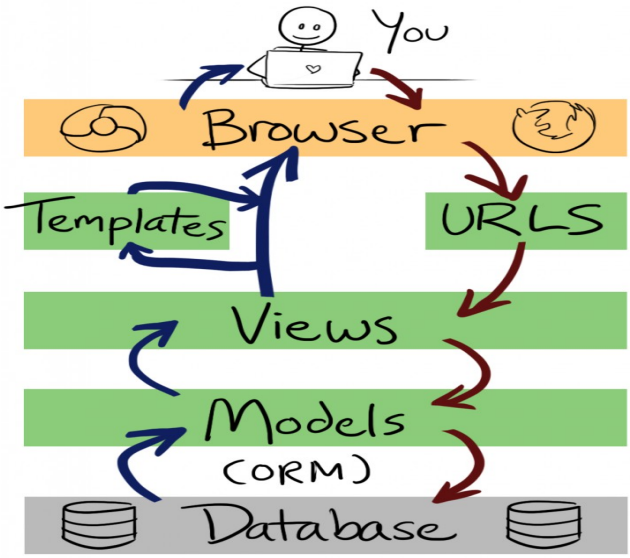

Figura 8 - Model-Template-View

Fonte: http://littlegreenriver.com/weblog/wp-content/uploads/mtv-diagram-730x1024.png

Conforme apresenta a Figura 8, o fluxo do sistema mo modelo *Model-Template-View*, se resume nos seguintes itens:

a) *URLs* (*Universal Resource Locator)* - mapeamento das rotas e recursos do sistema.

b) *View* - regras de negócios do sistema, atuando como intermediário entre o Model e Template. Obtêm o objeto de requisição (*request*) e retorna o objeto de resposta (*response*).

c) *Model* - abstração do banco de dados onde são definidos os modelos de dados do sistema.

d) *Template* - disposição e apresentação de dados ao usuário.

O Django é considerado um "*superframework*", pois é composto de vários *frameworks* (componentes) menores (SANTANA, 2010). Alguns destes componentes são : mapeamento objeto-relacional;sistema de template; sistema de administração; *Universal Resource Locator dispatcher*; Sistema de cache; internacionalização; formulários.

3.1.5 SQLite

SQLite é uma 'biblioteca em processo' que implementa um motor de banco de dados SQL transacional (SQLITE, 2013) que é auto-suficiente, ou seja, sem servidor e sem necessidade de configuração.

SQLite foi criada por Dwayne Richard Hipp no ano de 2000 e liberada em domínio público. É implementada em linguagem C, multiplataforma, e possui integração com diversas linguagens de programação. Programas que usam a biblioteca SQLite podem ter acesso a banco de dados SQL sem executar um processo SGBD separado.

O principal objetivo do projeto SQLite é ser simples e tem sido uma escolha popular como banco de dados integrado para armazenamento local / cliente no software aplicativo como navegadores *web*, sistemas operacionais e sistemas embarcados, entre outros.

3.1.6 Bootstrap

Bootstrap é um *framework open-source* lançado pela empresa Twitter em 2011 para auxiliar no desenvolvimento de websites e aplicações *web*. Atua no *front-end* (camada de visualização) através de um conjunto de componentes HTML, CSS e Javascript através da biblioteca JQuery. Um de seus recursos mais populares é o de *responsive design.*

3.1.7 Flot

Flot é uma biblioteca Javascript de geração de gráficos para JQuery. Lançada pelo dinamarquês Ole Laursen como um projeto *open-source*.

Com o uso do Flot é possível gerar diversos tipos de gráficos como, formato pizza, gráfico de barras, linhas em multiplos eixos e gráficos atualizados em tempo-real via Ajax.

3.1.8 Sensor DHT11

Sensores são dispositivos que respondem a um estímulo físico/químico de maneira específica e mensurável analogicamente.

Para a realização deste trabalho a escolha do sensor foi baseada na facilidade de se trabalhar com o sensor e sua eficiência. A Figura 9 apresenta uma imagem do sensor DHT11.

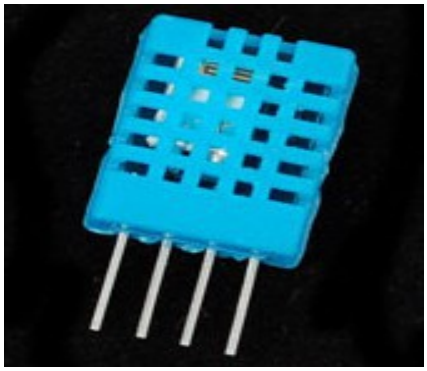

Figura 9 - Sensor DHT11

O DHT11 é composto por um sensor capacitivo de umidade relativa (%UR) e um termistor temperatura (ºC). Possui internamente um microcontrolador de 8 bits que facilita sua utilização tratando o sinal de resposta de forma digital. É composto por quatro terminais:o primeiro, da esquerda para direita, para alimentação (Vcc), o segundo para dados, o terceiro não é utilizado e o quarto para o terra (GND). A seguir, algumas especificações:

- Tensão de alimentação: 3 a 5.5 VDC (5VDC recomendado).
- Saída do sinal: digital de 1 fio.
- Faixa de medição: 20-90% RH; 0-50ºC .
- Precisão: Umidade+-4%RH (Max +-5%RH); Temperatura: +-2ºC .
- Resolução: Umidade 1%; Temperatura: 0.1ºC .
- Estabilidade a longo prazo: +-0.5%RH/ano.

Conforme recomendado por um dos fornecedores do sensor DHT11 deve-se utilizar um resistor de 10k ohms para calibragem do mesmo. O resistor é um componente elétrico capaz de limitar a corrente de um circuito que seja maior do que um componente espera receber, evitando a queima deste componente.

Para utilizar o sensor com o Arduino é necessário fazer download da biblioteca no site https://github.com/adafruit/DHT-sensor-library e salvar no diretório libraries da IDE Arduino.

3.1.9 Arduino

O Arduino como hardware e seu ambiente de desenvolvimento (IDE) foram apresentados no Capítulo 2 como parte de referencial teórico. Para esse trabalho foi utilizada uma placa Arduino Uno e a IDE Arduino versão 1.0.5.

3.1.10 Cron

O Cron é um serviço do Linux que é carregado durante o processo de *boot* do sistema. Trata-se de uma ferramenta que permite programar a execução de comandos e de processos de maneira repetitiva ou apenas uma única vez (ALECRIM, 2005).

Para executar as tarefas, o cron usa uma espécie de tabela denominada *crontab*. O arquivo crontab geralmente fica localizado no diretório /etc, mas também pode estar em um diretório que cria um crontab para cada usuário do sistema, geralmente em /var/spool/cron/ (ALECRIM, 2005).

Para configurar o Cron é necessário usar o comando crontab -e. Em seguida a tabela do Cron é aberta para edição devendo ser adicionada uma linha no formato: [minutos (0-59)] [horas (0-23)] [dia do mês (0-31)] [mês (1-12)] [dia da semana (0-7)] [usuário] [comando]. O campo usuário pode ser emitido se o usuário que irá executar o comando é o usuário em questão, os campos podem ser preenchidos com um * indicando repetição constante. A Listagem 1 apresenta um exemplo de configuração.

```
10 * * * * /home/dyego/swm/arduino.sh
```

Listagem 1 - Exemplo de configuração do Cron

Neste exemplo o comando é executado a cada 10 minutos em todas as horas

de todos os dias de todos os meses.

3.2 MÉTODO

O método define as principais atividades realizadas para o desenvolvimento deste trabalho. Como método foram utilizadas as fases do processo sequencial proposto em Pressman (2005). A seguir essas fases e o que foi realizado em cada uma delas.

a) Levantamento de requisitos

Inicialmente foi definido o escopo do sistema: um sistema web para monitoramento de dados (temperatura e umidade) de um ambiente através da interação com o microntrolador Arduino e o uso de sensor específico. Definiu-se que a ênfase do trabalho estaria em mostrar o funcionamento do Arduino, incluindo a atuação com sensores e a sua linguagem de programação.

b) Análise e projeto do sistema

Definição da modelagem do sistema. A modelagem é bastante simples, baseada em casos de uso e na forma de disposição do sensor e demais dispositivos eletrônicos na protoboard. Em seguida foi definido o diagrama de entidade-relacionamento do banco de dados.

c) Implementação (codificação) do sistema

A implementação se deu em duas etapas: inicialmente foi implementada a parte do sistema relacionada ao microcontrolador, sua interação com o sensor e comunicação serial com um software responsável por armazenar as leituras no banco de dados. A segunda parte foi o desenvolvimento do sistema web dentro da arquitetura MTV.

4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta o sistema *web* para monitoramento de sensores implementado como resultado do desenvolvimento deste trabalho. Da modelagem estão os casos de uso, protótipo do circuito eletrônico para uso do Arduino e o sensor, e diagrama de entidade-relacionamento definido.

4.1 APRESENTAÇAO DO SISTEMA

O sistema desenvolvido possibilita a aquisição de dados de um ambiente através da plataforma Arduino. Esses dados se referem a fenômenos físicos como temperatura e umidade. A interação do Arduino com o ambiente físico para a coleta dos dados desses fenômenos ocorre por meio de um sensor específico. O sensor fica conectado a uma placa Arduino e os dados são enviados, via comunicação serial, para um aplicativo dedicado que armazena-os em banco de dados.

O usuário interage com o sistema por meio de um navegador web. Essa interação se refere a visualização desses dados, atualizados a cada minuto na tela principal do sistema, visualização de relatório de leituras por período informado e gráfico do histórico das leituras realizadas.

O sistema possui área administrativa para cadastro de usuários e manipulação dos dados das leituras.

4.2 MODELAGEM DO SISTEMA

Como requisitos funcionais do sistema foram definidos:

a) Os usuários do sistema podem consultar dados de temperatura e umidade de um ambiente.

b) O sensor utilizado deve obter os valores de temperatura e umidade de um ambiente.

c) Os dados obtidos devem ser armazenados em banco de dados.

d) Os usuários do sistema podem visualizar o histórico de leituras realizadas em um período através de relatórios.

e) Os usuários do sistema podem visualizar o histórico de leituras realizadas através de um gráfico.

f) O sistema deve permitir cadastro de usuários.

E como requisitos não funcionais foram identificados os seguintes:

a) O sistema deve ser acessado via web.

b) O Arduino deve ficar aguardando uma requisição do sistema para realizar a leitura do sensor.

c) O Arduino deve retornar ao sistema os dados obtidos do sensor.

d) É necessário login para acesso ao sistema.

e) O cadastro de usuários do sistema é realizado pelo administrador.

A partir dos requisitos foram definidos os casos de uso representados na Figura 10.

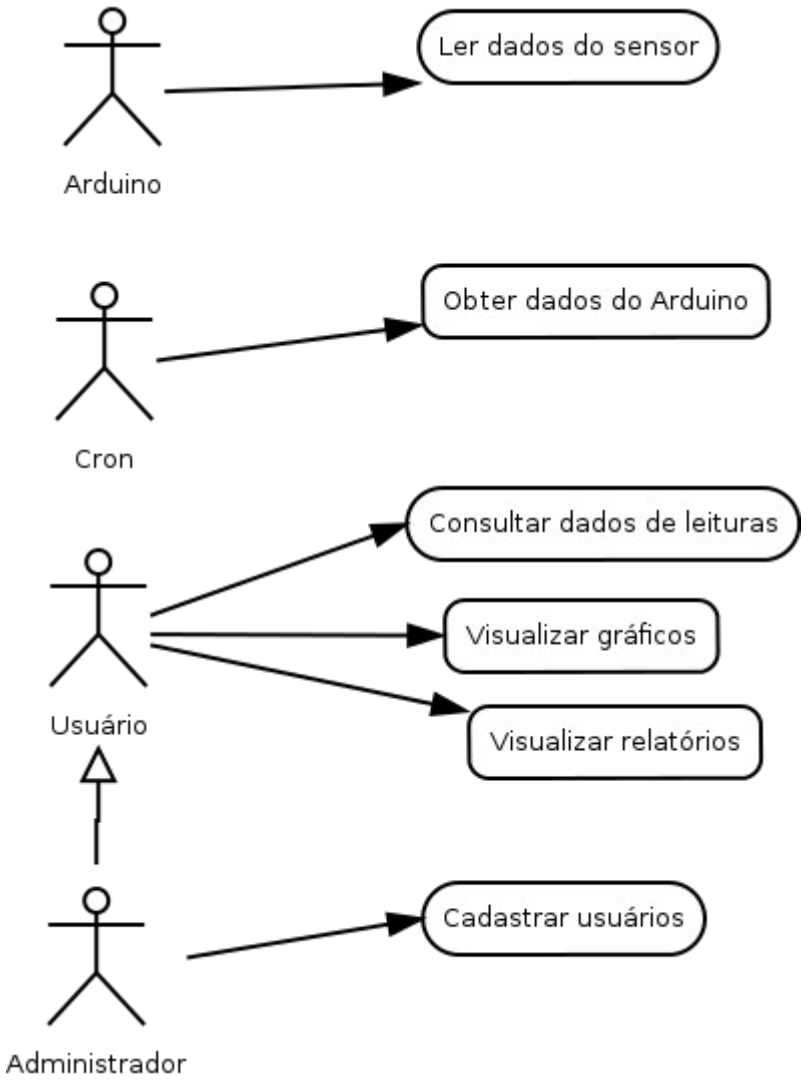

Figura 10 - Diagrama de casos de uso

Os quadros 1 a 5 a seguir apresentam a descrição dos casos de uso constantes na Figura 10.

Identificador do caso de uso: Ler dados do sensor. **Descrição**: O Arduino faz a leitura atual do sensor. **Evento Iniciador**: O sistema envia o sinal para de leitura. **Atores**: Arduino. **Pré-condição**: Não há pelo sistema, porque considera-se que o Arduino esteja ativo e os sensores em

condições de leitura dos fenômenos físicos.

Sequência de Eventos:

1. O sistema envia o sinal.
2. O Arduino faz a leitura do sensor.
3. O Arduino envia os dados obtidos ao sistema.
4. O sistema armazena os dados recebidos no banco de dados.

Pós-Condição:

Leitura salva no banco.

Extensões:

Não há.

Quadro 1 - Caso de uso ler dados do sensor

Identificador do caso de uso:

Obter dados do Arduino.

Descrição:

O Cron executa o aplicativo para obter os dados de leitura do Arduino.

Evento Iniciador:

O sistema envia o sinal para de leitura.

Atores:

Arduino.

Pré-condição:

O Cron deve estar configurado no intervalo de tempo desejado.

Sequência de Eventos:

1. É atingido o intervalo configurado.
2. O Cron executa o aplicativo de comunição com o Arduino.
3. O sistema envia o sinal.
4 O Arduino faz a leitura do sensor.
5. O Arduino envia os dados obtidos ao sistema.
6. O sistema armazena os dados recebidos no banco de dados.

Pós-Condição:

Leitura salva no banco.

Extensões:

Não há.

Quadro 2 - Obter dados do Arduino

Identificador do caso de uso: Consultar dados de leituras. **Descrição**: O usuário deseja visualizar os dados da leitura mais recente. **Evento Iniciador**: O sistema verifica no banco o registro mais recente. **Atores**: Usuário. **Pré-condição**: Deve existir registro no banco de dados. **Sequência de Eventos**: 1. O usuário faz login no sistema. 2. O usuário acessa a página principal. **Pós-Condição**: Leitura apresentada na tela.

Quadro 3 - Caso de uso consutar dados de leitura

Identificador do caso de uso: Visualizar gráficos. **Descrição**: O usuário deseja visualizar gráficos de leituras. **Evento Iniciador**: O usuário acessa o item do menu 'Gráfico'. **Atores**: Usuário. **Pré-condição**: Deve existir registro no banco de dados. **Sequência de Eventos**: 1. O usuário faz login no sistema. 2. O usuário acessa o menu. 3. O sistema apresenta o gráfico. **Pós-Condição**: Gráfico de leituras apresentado na tela. **Extensões**: Não há.

Quadro 4 - Caso de uso visualizar gráficos

Identificador do caso de uso:

Visualizar relatório.

Descrição:

O usuário deseja visualizar relatório de leituras.

Evento Iniciador:

O usuário acessa o item do menu 'Relatório'.

Atores:

Usuário.

Pré-condição:

Deve existir registro no banco de dados.

Sequência de Eventos:

1. O usuário faz login no sistema.
2. O usuário acessa o menu.
3. O usuário informa o período.
4. O sistema apresenta o relatório.

Pós-Condição:

Relatório de leituras apresentado na tela.

Extensões:

Não há.

Quadro 5 - Caso de uso visualizar relatório

Identificador do caso de uso:

Cadastrar usuário.

Descrição:

O administrador deseja cadastrar usuário.
Evento Iniciador:
O administrador acessa a área de administração do sistema.
Atores:
Administrador.
Pré-condição:
Inexistente.
Sequência de Eventos:
1. O administrador faz login.
2. O administrador acessa o menu 'Cadastrar usuário'.
3. O administrador informa os dados do usuário.
4. O administrador salva os dados.
Pós-Condição:
O usuário é cadastrado no sistema.
Extensões:
Não há.

Quadro 6 - Caso de uso cadastar usuários

O diagrama de entidade e relacionamentos determinado é simples com apenas uma tabela para armazenar os dados das leituras feitas com o Arduino. Essa tabela está na Figura 11.

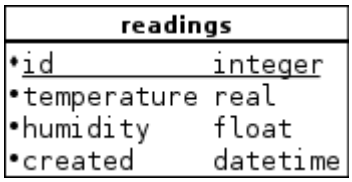

Figura 11 - Tabela do banco de dados

A entidade *readings* é composta pela temperatura (campo *temperature*), umidade (campo *humitity*) e data e hora da criação do registro (campo *created*).

As demais entidades e relacionamentos necessários são criadas e gerenciadas pelo *framework* Django, que mapeia a entidade *readings* e gera automaticamente a camada de modelo do sistema através de seu sistema ORM (*Object-Relational Mapping*).

Como parte da modelagem do sistema, foi realizado o esboço (*sketch*) do

circuito eletrônico utilizado (Figura 12). Esse esboço indica os elementos utilizados (sensor, resistor) e a ligação entre eles por meio de fios de energia e o terra, dispostos em uma *protoboard* e a conexão com o Arduino. Para o correto funcionamento desse sensor foi necessário utilizar um resistor de 10k ohms ligado entre os pinos 5V e entrada digital (pino 2).

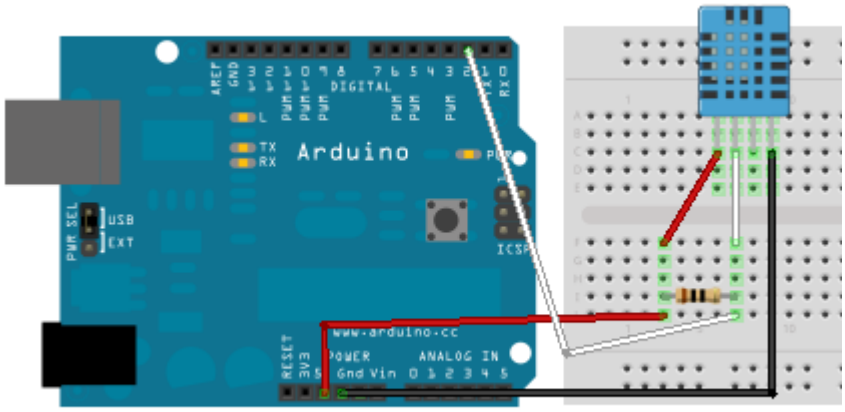

Figura 12 - Protótipo do circuito eletrônico

4.3 DESCRIÇÃO DO SISTEMA

Na Figura 13 está a tela de *login* ao sistema, responsável pela validação de acesso ao sistema por meio do nome de usuário e da respectiva senha.

Figura 13 - Tela de login no sistema

Após ser realizada a validação de acesso e todos os dados estarem de acordo com o seu cadastro no banco de dados, o sistema será aberto com a tela inicial (Figura 14).

Figura 14 - Tela inicial do sistema

Na parte superior da tela da Figura 14 encontra-se o menu de acesso as funções do sistema. Ao lado direito do menu aparece o botão “Sair” que desconecta o usuário do sistema. No lado esquerdo do menu encontram-se os itens: “Principal”, que mostra a tela principal do sistema, “Gráficos” e “Relatório” que serão explicados adiante.

Na mesma figura, ao centro, são apresentados os valores de temperatura e umidade lidos recentemente, assim como a data e hora da última atualização.

Ao clicar na opção “Gráficos” são apresentados os gráficos de temperatura e umidade, sendo que a variação do gráfico depende do intervalo de leitura, conforme é demonstrado na Figura 15.

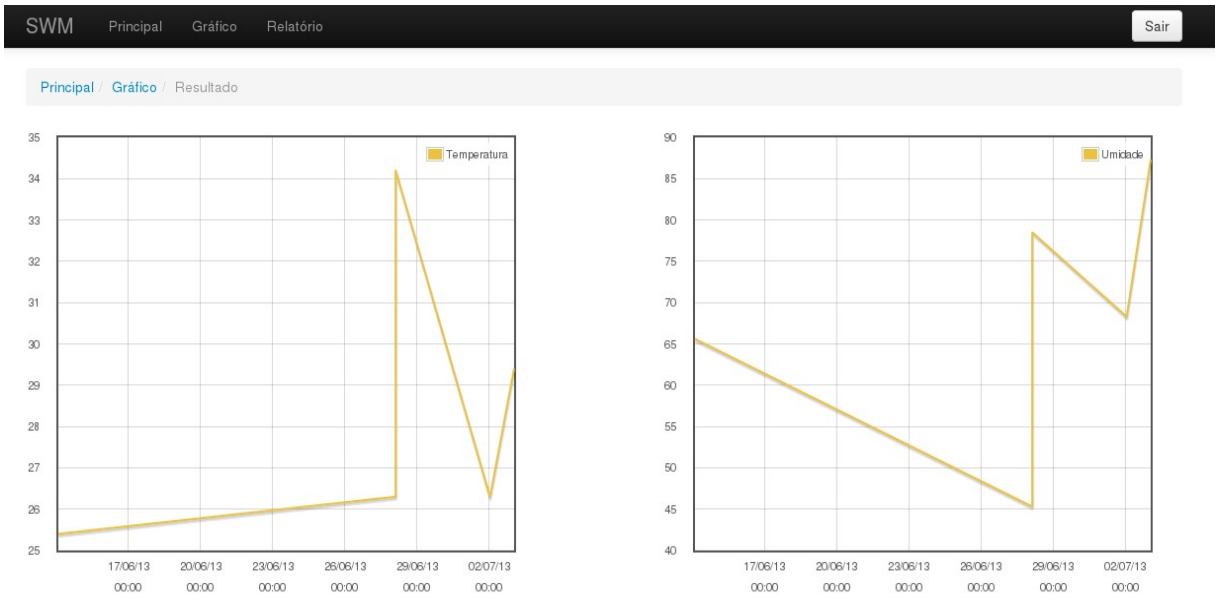

Figura 15 - Tela de gráficos

Já a opção “Relatório” mostra na tela um formulário para filtrar os resultados do relatório de acordo com o período informado pelo usuário. A tela de relatórios é apresentada na Figura 16.

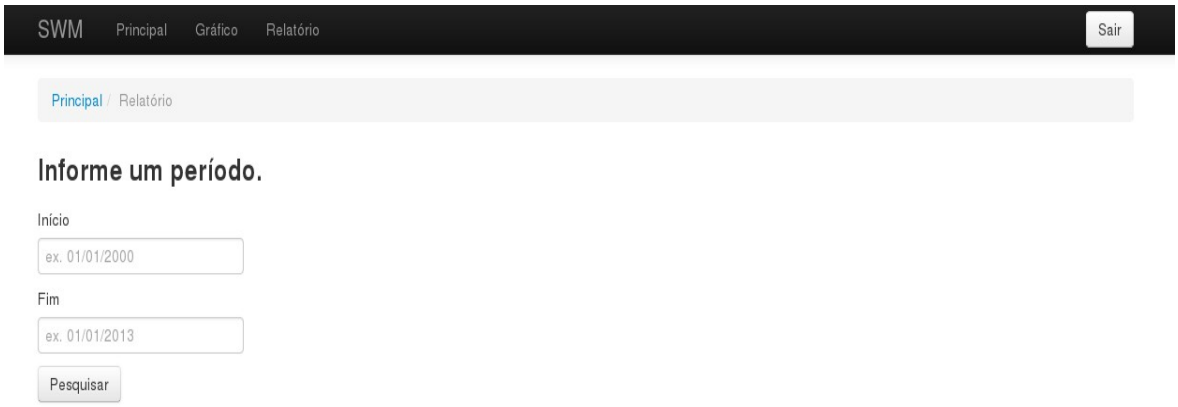

Figura 16 - Tela de filtros para o relatório

Na Figura 17 é apresentado o relatório após o usuário preencher o formulário de período. O relatório é composto pelos campos “ID”, que identifica o registro, “Temperatura”, “Umidade” e “Data/Horário” que mostra a data e hora da respectiva leitura do sensor.

ID	Temperature	Umidade	Data/Horário
1	21,2	55,3	31 de Julho de 2013 às 15:01
2	23,4	68,4	31 de Julho de 2013 às 15:01
3	21,3	65,4	8 de Agosto de 2013 às 17:38
4	21,3	75,6	8 de Agosto de 2013 às 17:39
5	26,5	82,1	8 de Agosto de 2013 às 17:39
6	20,3	84,5	8 de Agosto de 2013 às 17:39
7	24,5	71,4	8 de Agosto de 2013 às 17:39
8	23,4	64,9	8 de Agosto de 2013 às 17:39

Figura 17 - Tela de relatório

Ao acessar a área administrativa é apresentada a tela de login (Figura 18), acessível somente para usuários administradores do sistema.

Figura 18 - Tela de login para adminnistrador (Django)

Na parte superior da tela encontra-se o menu (Figura 19). No canto direito do menu é apresentado o usuário conectado e as opções de alterar senha e encerrar a sessão. Ao centro da tela, a esquerda são apresentadas as opções de gerenciamento como grupos e usuários, e os modelos do sistema. Ao lado direito tem-se as “Ações Recentes” realizadas pelo usuário administrador.

Figura 19 - Tela do administrador do sistema

Clicando em “Adicionar” nas opções de “Users” (usuários) é possível cadastrar um novo usuário no sistema, como é mostrado na Figura 20.

Figura 20 - Tela para incluir usuários

Ao clicar em "Read datas" na tela principal são apresentados todos os registros de leituras feitos, sendo possível editá-los ou adicionar novos clicando no botão "Adicionar read data" (Figura 21).

Figura 21 - Tela de gerenciamento de registros

4.4 IMPLEMENTAÇÃO DO SISTEMA

Este capítulo apresenta a implementação do projeto de software como resultado deste trabalho. O desenvolvimento foi separado em duas etapas:

a) Arduino: Apresentação da linguagem de programação do Arduino e apresentação do código do programa responsável pela leitura do sensor e envio dos dados obtidos ao sistema.

b) Django: Apresentação do framework Django e da linguagem Python. O Django utiliza o conceito de *app* (aplicativos) para cada conjunto de funcionalidades. Desta forma o trabalho realizado é incremental, começando pela criação do projeto Django e a configuração do framework, e em seguida são desenvolvidos os aplicativos (*apps*), cada qual implementando a estrutura MTV. Por último é apresentado o aplicativo que interfaceia o Arduino e o Django, sendo gerenciado pelo Cron.

O desenvolvimento deste projeto foi executado em sistema operacional Debian GNU/Linux e são apresentados os comandos para manipulação de arquivos e diretórios quando necessário.

As listagens de códigos são comentadas de acordo com a seguinte sintaxe, conforme a respectiva linguagem utilizada:

a) Arduino:

// comentário

b) Python/Django:

comentário

c) Template Django:

{# comentário #}

4.4.1 Arduino

No Arduino, todo código (*sketch*) deve respeitar uma estrutura geral que consiste em duas funções principais: *setup()* e *loop()*. Como é mostrado na Listagem 2.

```
void setup()
{
// código
}

void loop()
{
// código
}
```

Listagem 2 - Funções principais do Arduino

A função *setup()* tem o objetivo de configurar propriedades e declarar ou inicializar funções e variáveis que serão utilizadas pelo programa. Em seguida o programa entra em um laço de repetição infinito, o qual é definido pela função *loop().* Nessa função deve ocorrer o processamento em si, como a leitura e a escrita dos pinos digitais e analógicos.

A Listagem 3 mostra o escopo do e as funções do programa. Nesse código pode ser vista a inclusão da biblioteca responsável pelas funções do sensor, a declaração da velocidade de transmissão de dados por meio de *BAUD_RATE, a* definição do pino analógico número 2 (DHTPIN) que é utilizado pelo sensor, o modelo de sensor DHTTYPE DHT11 e o ID do Arduino, que servirá para identificar quando é feita uma requisição do sistema. É, ainda, criado o objeto sensor DHT dht(...) passando como argumentos o pino utilizado e o modelo definido e por último são criadas as variáveis que receberam os valores lidos de umidade e temperatura.

```
// Importa abiblioteca do sensor.
#include "DHT.h"

// Velocidade da comunicação serial.
#define BAUD_RATE 115200

// Define o pino do sensor e o seu tipo.
#define DHTPIN 2
#define DHTTYPE DHT11

// Define um caractere que indica leitura do sensor.
#define ID '1'

// Cria objeto sensor.
DHT dht(DHTPIN, DHTTYPE);

float humidity;
float temperature;

void setup()
{
  // Inicializa a comunicação serial.
  Serial.begin(BAUD_RATE);

  // Inicializa o sensor.
  dht.begin();
}

void loop()
{
  // Obtem dados do sensor.
  humidity = dht.readHumidity();
  temperature = dht.readTemperature();
}
```

```
// Rotina executada ao receber dados da serial.
void serialEvent()
{
  // Recebe o caractere da serial.
  char receive = Serial.read();

  // Se for igual ao ID cria uma string JSON com os dados da leitura e envia como resposta.
  if(receive == ID)
  {
    if(isnan(temperature) || isnan(humidity))
    {
      Serial.println("{\"error\": \"Failed to read from DHT\"}");
    }
    else
    {
      Serial.print("{");
      Serial.print("\"humidity\": ");
      Serial.print(humidity);
      Serial.print(", ");
      Serial.print("\"temperature\": ");
      Serial.print(temperature);
      Serial.print("}");
      Serial.println();
    }
  }
}
```

Listagem 3 - Programa Arduino

Como pode ser visto na Listagem 3, na função *setup()* é configurada a velocidade com a qual o Arduino se comunicará com a máquina através da porta USB e é iniciada a comunicação com o sensor. A função *loop()* obtêm os valores de umidade e temperatura constantemente e os armazena nas respectivas variáveis.

Na Listagem 3 também é apresentada a função *serialEvent()* que executa sua rotina sempre que houver uma requisição via serial, simulando uma interrupção.

Interrupção é utilizada por microcontroladores para aguardar um evento sem a necessidade de se implementar tal função, que teria que ser executada constantemente para efeito de capturar um evento. Assim, o programador define uma rotina que será chamada automaticamente quando ocorrer o evento. Neste caso, a rotina começa por ler um caractere do *buffer* serial, *Serial.read(),* e armazena-o na variável *receive.* Em seguida, o caractere recebido é testado e caso signifique o ID do Arduino é preparada a resposta para o sistema. A preparação da resposta começa com uma verificação se os valores de umidade e temperatura lidos anteriormente são válidos. Se os valores forem nulos ou lixo da memória é enviada uma *string* em formato JSON significando erro. Caso contrário, é criada outra *string* JSON com os valores da umidade e temperatura que é enviada via comunicação serial. A utilização do formato JSON facilita a troca e manipulação de dados entre sistemas.

4.4.2 Django

A implementação do sistema web inicia-se com a criação do projeto Django através do comando abaixo, sendo *swm (Sensor Web Monitor)* o nome do projeto (Listagem 4).

```
django startproject swm
```

Listagem 4 - Programa Arduino

O comando demonstrado cria a estrutura apresentada na Listagem 5.

```
swm
   |-- manage.py
   `-- swm
      |-- __init__.py
      |-- settings.py
      |-- urls.py
      `-- wsgi.py
```

Listagem 5 - Estrutura do projeto Django

A raiz do projeto, *swm*, possui o módulo *manage.py,* responsável por executar o ambiente de desenvolvimento e gerenciar os demais aplicativos. Dentro do subdiretório *swm* tem-se o arquivo *__init__.py,* que identifica que este é um programa Python, e *wsgi.py,* modulo que cria a instância de WSGI (Web Server Gateway), especificação que define uma interface simples e universal entre servidores web e aplicações web ou frameworks para a linguagem de programação Python. Já o arquivo *urls.py* ou *URLconf* como é chamado no Django, contem todas as rotas de URL's acessíveis atrvés do mapeamento com as respectivas funções da camada *View.* Para cada aplicativo criado é necessário criar um mapeamento no *URLConf. O* arquivo *settings.py* possui a configuração do projeto e é abordado adiante.

4.4.2.1 Configuração do framework

O próximo passo é a configuração do framework. Isso é realizado pela identificação do nome e a localização do banco de dados, URLs padrões e caminhos de arquivos que serão utilizados pelas aplicações do Django. Na Listagem 6 é mostrado o arquivo de configuração (*swm/swm/settings.py)*.

```
# Ao acessar o sistema o usuário é redirecionado para o login.
LOGIN_REDIRECT_URL = '/'

# Definição do caminho para login/logout.
LOGIN_URL = '/login/'  # caminho
LOGOUT_URL = '/logout/'

# Configura o sistema de banco de dados e o arquivo base.
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '/home/dyego/swm/swm/swm.db',
    }
```

```
}

# Define o local para arquivos estáticos (css, js).
STATIC_ROOT = '/home/dyego/swm/swm/static'
STATIC_URL = '//localhost:8000/static/'

# Define o diretório para as páginas a serem exibidas.
TEMPLATE_DIRS = (
    /home/dyego/swm/swm/'templates',
)

# Configura as apps utilizadas pelo Django.
INSTALLED_APPS += (
    'django.contrib.admin',
    'main',
    'reports',
    'charts',
)
```

Listagem 6 - Código de swm/swm/settings.py

A seção *INSTALLED_APPS* informa ao framework quais aplicativos serão utilizados no sistema. Na listagem acima já estão incluídos todos os aplicativos para simplificar a explanação do código, porem, a seção de aplicativos instalados deve ser preenchida conforme é criado cada aplicativo.

Em seguida é necessário criar as rotas de acesso do sistema. Essa configuração consiste em fazer o mapeamento das URLs, por meio de expressões regulares, e ligá-las à função da camada View do aplicativo responsável, como é apresentado na Listagem 7 (*swm/swm/urls.py*).

```
# Importa os módulos necessários.
from django.conf import settings
from django.conf.urls import patterns, include, url
from django.contrib import admin

# Habilita o sistema de administração.
```

```
admin.autodiscover()

# Mapeamento das rotas do sistema.
urlpatterns = patterns('',
    url(r'^login/$', 'django.contrib.auth.views.login', \
            {'template_name': 'login/login.html'}),
    url(r'^logout/$', 'django.contrib.auth.views.logout_then_login', \
            {'login_url': '/login/'}),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^charts/', 'charts.views.charts'),
    url(r'^charts_ajax/', 'charts.views.charts_ajax'),
    url(r'^reports/', 'reports.views.reports'),
    url(r'^$', 'main.views.index'),
)

# Mapeamento de arquivos estáticos.
if settings.DEBUG:
    urlpatterns += patterns('',
        url(r'^static/(?P<path>.*)$', 'django.views.static.serve', \
                {'document_root': settings.STATIC_ROOT}),
)
```

Listagem 7 - Código de swm/swm/urls.py

Conforme demonstrado na listagem acima, o módulo *URLConf* deve conter o mapeamento para cada aplicativo. Da mesma forma feita anteriormente com o *settings.py,* para simplificar a explanação do código o arquivo *urls.py* contem todos os mapeamentos feitos, porem, estes devem ser incluídos conforme é desenvolvido cada aplicativo.

4.4.2.2 Aplicativo principal

Tendo configurado o *framework*, é criado o aplicativo principal *main,* onde é definido o *Model* do sistema. Este será o único aplicativo que define classe de

modelo já que os outros aplicativos apenas processam dados existentes no banco. A Listagem 8 apresenta o comando para criação da aplicação *main*.

```
python manage.py startapp main
```

Listagem 8 - Criação da aplicação main

4.4.2.2.1 Model

Criada a aplicação *main*, é implementado o *Model* do sistema (*swm/main/models.py*), como apresentado na Listagem 9.

```
# Importa a classe de modelo do Django.
from django.db import models

# Define o modelo para as leituras feitas.
class ReadData(models.Model):
    humidity = models.FloatField()
    temperature = models.FloatField()
    created = models.DateTimeField()
```

Listagem 9 - Código de swm/swm/main/models.py

A camada de modelo do sistema é simples, com apenas uma classe que define a leitura realizada pelo Arduino cujo atributos respectivamente são umidade, temperatura e data e hora da leitura.

Para que posteriormente o administrador do sistema possa gerenciar os registros é necessário criar o módulo *admin.py* dentro da aplicação *main (swm/main/admin.py*), que registra a classe de modelo *ReadData*. A Listagem 10 apresenta esse arquivo.

```
# Importa a classe modelo.
from models import ReadData

# Importa o módulo de adminstração.
from django.contrib import admin

# Registra o modelo no subsistema de administração.
```

```
admin.site.register(ReadData)
```

Listagem 10 - Código de swm/main/admin.py

4.4.2.2.2 View

A função principal do sistema é a apresentação da leitura mais recente. A Listagem 11 mostra a *View* da aplicação *main* (*swm/main/views.py*).

```
# Importa módulos necessários.
from django.shortcuts import render_to_response
from django.contrib.auth.decorators import login_required
from models import ReadData

# Usa um 'decorator' indicando que a função requer que o usuário esteja logado.
@login_required
def index(request):
    try:
        read_data = ReadData.objects.latest('pk')
    except ReadData.DoesNotExist:
        read_data = None
    return render_to_response('main/index.html', {'read_data': read_data})
```

Listagem 11 - Código de swm/swm/main/views.py

A função *index* recebe o objeto de requisição e utiliza o 'decorator' *login_required* sendo executada apenas se o usuário estiver logado. Caso o login seja feito, é obtido registro mais recente do banco de dados através de *ReadData.objects.latest('pk')* . Caso ocorra uma exceção indicando que não existem registros no banco o objeto *read_data* recebe o valor *None.* Por último é retornada para o usuário a resposta através da função *render_to_response()* contendo o template index.html e o objeto *read_data.*

4.4.2.2.3 Template

Com a camada *View* implementada é necessário criar o *Template* que define

a página que será exibida ao usuário. Como o Django sugere a reutilização de recursos e antes de criar o *Template* da aplicação *main* é necessário criar os templates bases contendo o conteúdo HTML, CSS e JavaScript essencial para o sistema. Desta forma é criado o diretório *static* e os subdiretórios necessários que conterão os arquivos estáticos que serão utilizados pelos *templates* (Listagem 12).

```
mkdir -p swm/swm/static/css swm/swm/static/js
```

Listagem 12 - Criação do diretório static

Dentro do diretório *static* foram adicionados os arquivos do *framework* Bootstrap para criação do *front-end* do sistema e a biblioteca Flot para criação de gráficos com JavaScript.

O *framework* Django possui uma linguagem própria para criação de *templates* baseda no Python. Esta linguagem é incorporada ao HTML e simplifica a construção da página fornecendo rotinas para a lógica de apresentação de dados. A seguir é criado o diretório *templates* onde se encontrarão os *templates* do sistema (Listagem 13).

```
mkdir swm/swm/templates
```

Listagem 13 - Criação do diretório templates

Após isso são criados os templates básicos que serão herdados pelos templates específicos de cada aplicação. O primeiro template foi definido é o *base.html*, cujo código está na Listagem 14.

```
{# Carrega arquivos estáticos #}
{% load staticfiles %}

<!DOCTYPE html>
<html lang="pt-br">
   <head>
      <meta charset="utf-8"/>
      <meta name="viewport" content="width=device-width, initial-scale=1.0"/>

      {# Cria bloco meta #}
      {% block meta %}
```

```
    {% endblock meta %}

    {# Cria bloco title #}
    <title>{% block title %}SWM | Sensor Web Monitor{% endblock title %}</title>

    {# Obtem arquivos CSS #}
    <link href="{% static "css/bootstrap.min.css" %}" rel="stylesheet"/>
    <link href="{% static "css/bootstrap-responsive.min.css" %}" rel="stylesheet"/>
    <link href="{% static "css/base.css" %}" rel="stylesheet"/>

    {# Bloco destinado a arquivos de estilo adicionais #}
    {% block css %}
    {% endblock css %}

</head>
<body>

    {# Inclui o menu da pagina #}
    {% block menu %}
        {% include 'menu.html' %}
    {% endblock menu %}

    <div class="container">
        {# Bloco para o conteudo da pagina #}
        {% block content %}
        {% endblock content %}
    </div>

    {# Obtem arquivos Javascript #}
    <script src="{% static "js/jquery.min.js" %}"></script>
    <script src="{% static "js/bootstrap.min.js" %}"></script>

    {# Bloco destinado a arquivos Javascript adicionais #}
    {% block js %}
    {% endblock js %}
```

```
  </body>
</html>
```

Listagem 14 - Código de swm/swm/templates/base.html

Da mesma forma é criado o arquivo com o menu da página, *swm/swm/templates/menu.html,* como apresenta o código da Listagem 15.

```
{% block menu %}
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="navbar-inner">
      <div class="container">
        <button type="button" class="btn btn-navbar" data-toggle="collapse" data-
target=".nav-collapse">
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="brand" href="/">SWM</a>
        <div class="nav-collapse collapse">
          <div class="pull-right">
            <a href="/logout" class="btn">Sair</a>
          </div>
          <ul class="nav">
            <li><a href="/">Principal</a></li>
            <li><a href="/charts">Gráfico</a></li>
            <li><a href="/reports">Relatório</a></li>
          </ul>
        </div>
      </div>
    </div>
  </div>
{% endblock menu %}
```

Listagem 15 - Código de swm/swm/templates/menu.html

Como é apresentado na Listagem 15, o menu é apenas um bloco do template

com a estrutura em HTML. Com os templates básicos criados é possível criar o *Template* da aplicação *main*. Para isso é criado o diretório próprio para o aplicativo (Listagem 16).

```
mkdir swm/swm/templates/main
```

Listagem 16 - Criação do diretório para o aplicativo

A Listagem 17 apresenta o *template index.html.*

```
{# Herda o template base #}
{% extends 'base.html' %}

{# Carrega arquivos estaticos #}
{% load staticfiles %}

{# Inclui diretriz no bloco meta #}
{% block meta %}
<meta http-equiv="refresh" content="10"/>
{% endblock meta %}

{# Obtem arquivo de estilo próprio para a pagina #}
{% block css %}
<link rel="stylesheet" type="text/css" href="{% static "css/main.css" %}"/>
{% endblock css %}

{# Inclui conteudo na pagina #}
{% block content %}
<div id="main1">
   <div class="hero-unity">
      <h1 class="text-center">Sensor Web Monitor</h1>
   </div>
   <hr/>

{# Verifica se o contexto possui dados #}
{% if read_data %}
   <div class="row-fluid marketing">
      <div class="span6 read-data">
```

```
            <h3 class="text-center">Temperatura</h3>
            <p class="text-center" id="temperature">{{ read_data.temperature }}&deg;C</p>
        </div>
        <div class="span6 read-data">
            <h3 class="text-center">Umidade</h3>
            <p class="text-center" id="humidity">{{ read_data.humidity }}%</p>
        </div>
    </div>
    <hr/>
    <div class="jumbotron">
        <p class="lead text-center">Atualizado em {{ read_data.created}}h.</p>
    </div>
</div>
{% else %}
<p>Registro não encontrado.</p>
{% endif %}
{% endblock content %}
```

Listagem 17 - Código de swm/swm/templates/main/index.html

4.4.2.3 Criação do banco de dados

A Listagem 18 apresenta a geração do banco de dados do sistema a partir do *Model* definido no aplicativo *main*.

```
python manage.py syncdb
```

Listagem 18 - Criação do banco de dados

O comando *syncdb* além de criar o banco de dados sugere o cadastramento de um usuário administrador para o sistema (Listagem 19).

```
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_groups
```

```
Creating table auth_user_user_permissions
Creating table auth_user
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table django_admin_log
Creating table main_readdata

You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'dyego'): admin
Email address: admin@admin.org
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

Listagem 19 - Geração do banco e usuário administrador

Neste momento é possível acessar a administração do sistema e inserir registros no banco de dados para simular as leituras feitas pelo Arduino.

4.4.2.4 Relatórios

O próximo aplicativo a ser criado é responsável pelo relatório de leituras (*reports*) (Listagem 20).

```
python manage.py startapp reports
```

Listagem 20 - Criação dos relatórios

4.4.2.4.1 View

Para que o usuário possa visualizar as leituras feitas em um período é disposto um formulário de intervalo de data. Este formulário (código na Listagem 21) é um modelo *forms.py* que depois será manipulado pela *view* da aplicação.

```
# Importa o módulo de formulário do Django
from django import forms

# Define a classe de formulário de periodo.
class FormPeriod(forms.Form):
    start_date = forms.DateField(label='Inicio', \
            widget=forms.DateInput(format='%d/%m/%Y'), \
            input_formats=['%d/%m/%y', '%d/%m/%Y'])
    end_date = forms.DateField(label='Fim', \
            widget=forms.DateInput(format='%d/%m/%Y'), \
            input_formats=['%d/%m/%y', '%d/%m/%Y'])
```

Listagem 21 - Código de swm/reports/forms.py

A Listagem 22 apresenta a *view* da aplicação *reports.*

```
# Importa módulos necessários.
from django.shortcuts import render_to_response
from django.template import RequestContext
from django.contrib.auth.decorators import login_required
from main.models import ReadData
from forms import FormPeriod

# Exige login para esta função.
@login_required
def reports(request):

    # Verifica se a método utilizado é POST.
    if request.method == 'POST':
        form = FormPeriod(request.POST, request.FILES)
```

```
    # Verifica se os dados informados são válidos.
    if form.is_valid():
        period = form.cleaned_data

        # Obtem a lista de registros dentro do período.
        read_data_list = \
            ReadData.objects.filter(created__range=\
            (period['start_date'], period['end_date']))

        # Retorna o template e os registros obtidos.
        return render_to_response('reports/reports.html', \
                {'read_data_list': read_data_list})

# Se o método não for POST é criado um formulário.
else:
    form = FormPeriod()

# Retorna o template com um formulário.
return render_to_response('reports/form.html', {'form': form}, \
        context_instance=RequestContext(request))
```

Listagem 22 - Código de swm/reports/views.py

A *view* possui apenas uma função que verifica se a requisição contém o método *post*. Se positivo, é feita a consulta ao banco e uma filtragem dos registros de acordo com o período informado no formulário pelo usuário. Caso o método da requisição não seja *post* então é criado um formulário, instância da classe *FormPeriod*, como resposta ao usuário.

4.4.2.4.1 Template

Para criar os templates do aplicativo *reports* é criado um diretório específico (Listagem 23).

```
mkdir swm/swm/templates/reports
```

Listagem 23 - Criação do diretório para os relatórios

Na Listagem 24 está o template do formulário, arquivo *form.html.*

```
{# Herda o template base #}
{% extends 'base.html' %}

{# Insere o conteúdo da página #}
{% block content %}
<ul class="breadcrumb">
    <li><a href="/">Principal</a><span class="divider">/</span></li>
    <li class="active">Relatório</li>
</ul>
<h3>Informe um período.</h3>

{# Usa POST para a URL reports #}
<form action="/reports/" method="post">
    {% csrf_token %}
    <label>Início</label>
    <input type="text" name="start_date" placeholder="ex. 01/01/2000">
    <label>Fim</label>
    <input type="text" name="end_date" placeholder="ex. 01/01/2013">
    <br/>
    <button type="submit" class="btn">Pesquisar</button>
</form>
{% endblock %}
```

Listagem 24 - Código de swm/swm/templates/reports/form.html

Na Listagem 25 é apresentado o template do relatório. Esse template está no arquivo *reports.htm.*

```
{# Herda o template base #}
{% extends 'base.html' %}

{# Insere o conteúdo da página #}
```

```
{% block content %}
<ul class="breadcrumb">
   <li><a href="/">Principal</a><span class="divider">/</span></li>
   <li class="active">Relatório</li>
</ul>

{# Verifica se o contexto possui dados #}
{% if read_data_list %}
<table class="table table-hover">
   <thead>
      <tr>
         <th>ID</th>
         <th>Temperature</th>
         <th>Umidade</th>
         <th>Data/Horário</th>
      </tr>
   </thead>
<tbody>

{# Itera na lista de registros para montar a tabela #}
{% for item in read_data_list %}
   <tr>
      <td>{{ item.id }}</td><td>{{ item.temperature }}</td>
      <td>{{ item.humidity }}</td><td>{{ item.created }}</td>
   </tr>
{% endfor %}
</tbody>
</table>

{# Se a lista estiver vazia #}
{% elif read_data_list|length == 0 %}
<p>Nenhum registro encontrado.</p>
{% endif %}
{% endblock %}
```

Listagem 25 - Código de swm/swm/templates/reports/reports.html

4.4.2.5 Gráficos

Em seguida é criado o aplicativo para geração de gráficos (*charts*) por meio da instrução (Listagem 26).

```
python manage.py startapp charts
```

Listagem 26 - Criação do aplicativo para geração dos gráficos

4.4.2.5.1 View

A View da aplicação *charts* recebe a requisição do usuário e retorna o template respectivo. No template é feita uma chamada via Ajax para a *view* obter os dados das leituras e responder um objeto JSON, sendo este tratado no template e gerando os gráficos de temperatura e umidade. A Listagem 27 apresenta a *view* (*swm/charts/views.py*).

```
# Importa módulos necessários.
import time
from django.http import HttpResponse
from django.shortcuts import render_to_response
from django.contrib.auth.decorators import login_required
from django.utils import simplejson
from main.models import ReadData

# Função que obtem os registros de leituras.
# Retorna um objeto JSON.
def items():

    # Obtem todos os registros.
    data = ReadData.objects.all()

    # Cria um dicionário com duas listas: temperatura e umidade.
    items = {'charts': {'temperature': [], 'humidity': []}}
```

```
    # Cada registro é adicionado no dicionário.
    for item in data:
        created = int(time.mktime(item.created.timetuple())*1000)
        items['charts']['temperature'].append([created, item.temperature])
        items['charts']['humidity'].append([created, item.humidity])

    # Cria e retorna um objeto JSON.
    return simplejson.dumps(items)

# Exige login para executar.
@login_required
def charts(request):

    # Retorna o template.
    return render_to_response('charts/charts.html', {'data': items()})

# Função chamada pelo AJAX.
def charts_ajax(request):

    # Retorna o contexto (JSON) para o template.
    return HttpResponse(items(), content_type='application/json')
```

Listagem 27 - Código de swm/swm/charts/views.py

4.4.2.5.2 Template

Em seguida é criado o diretório de templates do aplicativo *charts* (Listagem 28).

```
mkdir swm/swm/templates/charts
```

Listagem 28 - Criação do diretório de templates

O código da aplicação *charts* é apresentado na Listagem 29.

```
{# Herda o template base #}
{% extends 'base.html' %}
```

```
{# Carrega os arquivos estáticos #}
{% load staticfiles %}

{# Insere um estilo para a página #}
{% block css %}
<link href="{% static "css/charts.css" %}" rel="stylesheet"/>
{% endblock css %}

{# Insere o conteúdo da página #}
{% block content %}
<ul class="breadcrumb">
    <li><a href="/">Principal</a><span class="divider">/</span></li>
    <li><a href="/charts/">Gráfico</a><span class="divider">/</span></li>
    <li class="active">Resultado</li>
</ul>

{# Verifica se existem dados no contexto #}
{% if data %}
<div id="temperature"></div>
<div id="humidity"></div>

{# Se não tiver dados, informar #}
{% elif data|length == 0 %}
<p>Nenhum registro encontrado.</p>
{% endif %}
{% endblock content %}

{# Insere biblioteca Flot para criação dos graicos #}
{% block js %}
<script src="{% static "js/jquery.flot.min.js" %}"></script>
<script src="{% static "js/flot/jquery.flot.time.js" %}"></script>

{# Javascript responsável pelos gráficos #}
<script>
    $(function() {
```

```
    {# Opções do gráfico #}
     var options = {
        xaxis: {
            mode: "time",
            timeformat: "%d/%m/%y %H:%M"
        }
     };

     {# Executada ao receber objeto JSON #}
     function onDataReceived(items) {
        $.plot("#temperature",[{data: items["charts"]["temperature"],
            label: "Temperatura"}], options);

        $.plot("#humidity", [{data: items["charts"]["humidity"],
            label: "Umidade"}], options);
     }

     {# Chamada da View via AJAX #}
     $.ajax({
        url: "/charts/charts_ajax/",
        type: "GET",
        dataType: "json",
        success: onDataReceived
     });
  });
</script>
{% endblock js %}
```

Listagem 29 - Código de swm/swm/templates/charts/charts.html

4.4.2.6 Aplicativo Python-Arduino

Neste momento é criado o aplicativo Python responsável por fazer a comunicação serial com o Arduino e obter os dados das leituras do sensor, salvando

os registros no banco. Este aplicativo utiliza o Django para obter o *Model* e manipular o banco de dados através do *ORM,* entretanto, o aplicativo não é parte do framework como os demais aplicativos e este é gerenciado pelo programa Cron.

A Listagem 30 apresenta o módulo *swm/arduino.py.*

```
#!/usr/bin/env python

# Importa os modulos de comunicação serial e de tempo.
import serial
import time

# Define a classe para comunicação com o Arduino.
class Arduino(object):

    def connect(self, port, baud_rate):
        self.arduino = serial.Serial(port, baud_rate)
        time.sleep(2)

    def write(self, data):
        self.arduino.write(data)

    def read(self):
        return self.arduino.readline()

# Indentifica que este é o programa principal.
if __name__ == '__main__':

    # Importa modulos para interação com o Django.
    import os
    import sys
    import json

    # Se identificar a configuração do Django importa a classe de modelo.
    if os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'swm.settings'):
        from main.models import ReadData
        from django.utils import timezone
```

```
else:
    raise
    sys.exit(1)

# Define a porta serial (USB) e velocidade de transmissão de dados.
SERIAL_PORT = '/dev/ttyACM0'
BAUD_RATE = 115200

# Cria um objeto Arduino.
arduino = Arduino()
arduino.connect(SERIAL_PORT, BAUD_RATE)

# Envia o caractere 1 indicando leitura ao Arduino.
arduino.write('1')

# Recebe dados da leitura.
receive = json.loads(arduino.read())

# Cria objeto leitura e alimenta seus atributos.
read_data = ReadData()
read_data.temperature = receive['temperature']
read_data.humidity = receive['humidity']
read_data.created = timezone.now()

# Salva a leitura no banco de dados.
read_data.save()
```

Listagem 30 - Código de swm/arduino.py

Para gerenciar o tempo das leituras a serem realizadas foi utilizado o programa Cron do GNU/Linux. Esta tarefa permite ao usuário configurar quando deve ser feita a leitura e o intervalo da mesma. O Cron executará um Shell Script cuja função é acessar o diretório do projeto e, através do Python, executar o módulo *arduino.py*. O script é apresentado na Listagem 31.

```
#!/usr/bin/env bash
```

```
cd /home/dyego/swm/
python /home/dyego/swm/arduino.py
```

Listagem 31 - Código de swm/arduino.sh

A primeira linha do código apresentado na Listagem indica que este é um script executado pelo interpretador Bash. As linhas seguintes são responsáveis por acessar o diretório do projeto e executar o módulo arduino.py.

Em seguida é configurado o Cron através do seguinte comando (Listagem 32).

```
crontab -e
```

Listagem 32 -Comando para configuração do Cron

No arquivo *crontab* é adicionada a seguinte linha, indicando para executar o script *arduino.sh* a cada 10 minutos com a instruçao:

10 * * * * /home/dyego/swm/arduino.sh.

Tendo configurado o Cron a aplicação já estará sendo executada no tempo determinado e o resultado é visível pelo sistema web.

Todos os códigos implementados neste trabalho podem ser reproduzidos e estão disponíveis no endereço https://github.com/dyegocantu/swm

5 CONCLUSÃO

O crescimento da rede mundial de computadores tem mudado muitos paradigmas tecnológicos, ampliando o próprio conceito da Internet. Dispositivos eletrônicos, dos mais variados, cada vez mais interagem entre si e com sistemas computacionais por meio da rede, sob o termo Internet das Coisas.

Como forma de exemplificar o uso da Internet das Coisas, o presente trabalho teve como objetivo a implementação de um sistema *web* utilizando o *framework* Django e a linguagem de programação Python para monitoramento de sensores com o uso da plataforma microcontrolada Arduino.

O trabalho realizado apresenta a forma de uso do Arduino para leitura de um sensor de temperatura e umidade. Os dados das leituras realizadas são armazenados em banco de dados por meio de um programa Python via comunicação serial. Estes dados são visualizados e manipulados por um sistema *web*, que possui uma interface de usuário e uma de administração.

As tecnologias utilizadas neste trabalho são livres e são adequadas tanto para o estudo acadêmico como para projetos comerciais. A plataforma Arduino facilita o aprendizado e a prototipação de circuitos eletrônicos devido a sua linguagem de programação simplificada e baseada em C/C++ e por possuir uma documentação consistente voltada ao público leigo em eletrônica. A linguagem Python se destaca por possuir uma sintaxe didática e pela riqueza de módulos incorporados, proporcionando uma rápida curva de aprendizado e possibilitando que o desenvolvedor mantenha o foco na solução do problema.

O Django é um *framework* para desenvolvimento *web* ágil. Apesar do Django exigir algum conhecimento de arquitetura *Model-Template-View*, o *framework* mostrou-se muito eficiente por ser bem documentado e fornecer os recursos que facilitaram o desenvolvimento do sistema *web* mantendo o prazo estabelecido. Entre os destaques do Django pode-se citar: sistema ORM, módulo de autenticação de usuários e sistema de administração.

Como complemento a este trabalho, explorando ainda mais a Internet das Coisas, sugere-se que o próprio Arduino envie dados pela rede ao sistema web. Isto pode ser feito utilizando um *shield*, placa que expande as funções do Arduino, com

conexão *ethernet* ou *wireless.* Desta forma, no sistema web, deverá ser implementada a rotina que aguarda conexões de rede para obter os dados do dispositivo, ou *coisa,* de acordo com o conceito de IoT.

REFERÊNCIAS

AL-KUWARI, Ali M.A.H.; ORTEGA-SANCHEZ, Cesar; SHARIF, Atif; POTDAR, Vidyasagar M. **User friendly smart home infrastructure: BeeHouse.** 5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011), 2011, p. 257-262.

ALECRIM, Emerson. **Usando cron e crontab para agendar tarefas**. 2005. Disponível em: http://www.vivaolinux.com.br/artigo/Usando-cron-e-crontab-para-agendar-tarefas. Acesso em 28 ago. 2013.

ARDUINO. **Arduino**. Disponível em: <http://www.arduino.cc/>. Acesso em: 14 fev. 2013.

BORGES, Luis E. **Python para desenvolvedores**. 2a. ed. Rio de Janeiro - RJ: Edição do Autor, 2010.

BROWN, Stuart. **W*eb* technology family tree**, 2006, Disponível em: http://modernl.com/article/*web*-tech-family-tree. Acesso em: 14 fev. 2013.

CARDOSO, Ismael. "**Não precisa dinheiro", diz jovem que criou alerta de tremor**. 2012. Disponível em: <http://tecnologia.terra.com.br/nao-precisa-dinheiro-diz-jovem-que-criou-alerta-de-tremor,160ae194c2bda310VgnCLD200000bbcceb0aRCRD.html>. Acesso em: 9 set. 2013.

CHASE, Otávio. A. **Sistemas embarcados**. 2007. Disponível em: <www.neoradix.com.br/_.../NEORADIX_01_Sistemas_Embarcados.pdf>. *A*cesso em: 12 jun. 2013.

CISCO. **Cisco visualizations**, 2011. Disponível em <http://share.cisco.com/internet-of-things.html>, acessado em: 03 abr. 2013.

COLE Peter H., ENGELS Daniel W. **Auto ID - 21st century supply chain technology**. Proceedings of AEEMA Cleaner Greener Smarter conference, October 2002, p. 1-7.

CVIJIKL, Irena P., MICHAHELLES, Florian. T**he Toolkit approach for end-user participation in the internet ofthings**. In: Uckelmann, D., Harrison, M., Michahelles, F. (Eds): Architecting the Internet of Things. Springer, 2011, p. 65-97.

DJANGO, 2013. **The Web framework for perfectionists with deadlines**. Disponível em: <https://www.djangoproject.com/>. Acesso em: 03 abr. 2013.

DUCATEL K., BOGDANOWICZ M., SCAPOLO F., LEIJTEN J., BURGELMAN J-C.

Scenarios for ambient intelligence in 2010. European Commission, Technical Report, 2001.

GNOME. **Dia a drawing program**. Disponível em: <http://projects.gnome.org/dia/>. Acesso em: 03 abr. 2013.

GUPTA, Sandeep K. S.; LEE, Wang-Chien, PURAKAYASTHA; Apratim; SRIMANI, Pradip K. **An overview of pervasive computing**. IEEE Pers Communication, v. 8, n. 4, p. 8-9, 2001.

HODGES, Steve; VILLAR, Nicolas; SCOTT, James; SCHMIDT, Albrectit. A **new era for ubicomp development**. Pervasive Computing, IEEE, v. 11, n. 1, p. 5-9, Jan. 2012.

HOMPEL Michael ten. **Das internet der dinge: status, perspektive, aspekte der aktuellen rfid-entwicklung**. Dortmunder Gespräche 2005. Fraunhofer Symposium RFID, Dortmund.

HRIBERNIK, Karl A.; GHRAIRI, Zied; HANS, Carl; THOBEN, K. **First experiences in the participatory design of intelligent products with arduino**. Proceedings of the 2011 17th International Conference on Concurrent Enterprising (ICE 2011).

KÄRKKÄINEN, Mikko; HOLMSTRÖM Jan; FRÄMLING, Kary; ARTTO, Karlos. I**ntelligent products - a step towards a more effective project delivery chain**, 2003 Comput in Industry, v. 50, p. 141-151.

KATO, Yoshiharu. **Splish: a visual programming environment for arduino to accelerate physical computing experiences**. 2010 Eighth International Conference on Creating, Connecting and Collaborating through Computing. IEEE Computer Society, 2010, p. 3-10.

KO, Hoon; RAMOS, Carlos A. **A Survey of context classfication for intelligent systems research for Ambient Intelligence.** 2010 International Conference on Complex, Intelligent and Software Intensive Systems. IEEE Computer Society, 2010, p. 746-751.

MARINHO, José Edson dos Santos, MARINHO, Ednaldo dos Santos. **Minicurso de microcontrolador.** Ed. Especial nº 2, Saber Eletrônica, Janeiro 2001. Disponível em: <http://www.ivair.com.br/download/minicurso.pdf>. Acesso em: 24 abr. 2013.

MASSIMO, Banzi, CUARTIELLES, David. **Arduino**. Disponível em <http://www.arduino.cc/.>. Acesso em: 06 fev. 2013.

MATIC, Nebojsa. **The PIC microcontrollers**, book 1, Editora*:* mikroElektronika, ano 2003.

MCFARLANE, Duncan; SARMA, Sanjay; CHIRN Jin L.; WONG, ChienYaw; ASHTON, Kevin. **Auto ID systems and intelligent manufacturing control**. Eng

Appl of Artif Intell, v. 16, p. 365-376.

MEYER, Gerben G., FRÄMLING, Kary, HOLMSTRÖM Jan. **Intelligent products: a survey,** Comput in Ind, v. 60, p. 137-148.

MOZILLA, 2013. **JavaScript**. Disponível em: <https://developer.mozilla.org/en-US/docs/JavaScript>. Acesso em: 03 abr. 2013.

NASCIMENTO, Erik B. **Aplicação da programação de microcontroladores 8051 utilizando linguagem C.** Trabalho de Conclusão de Curso. Bacharelado em Sistemas de Informação. Faculdade Sete de Setembro – FASETE, Paulo Afonso – BA, 2009.

NICOLOSI, Denys E.C. **Microcontrolador 8051 detalhado**. São Paulo: Érica, 2000.

NOBREGA FILHO, Raimundo G. **Fundamentos de hardware**. Disponível em: <http://www.di.ufpb.br/raimundo/ArqDI/Arq5.htm>. Acesso em: 6 mar. 2013.

O'REILLY, Tim. **What is web 2.0**. O'Reilly, 30 set. 2005. Disponível em: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>. Acesso em: 15 abr. 2013.

PRESSSMAN, Roger. **Engenharia de software**. McGraw-Hill, 2005.

PYSERIAL, 2013. **Welcome to pySerial's documentation**. Disponível em: <http://pyserial.sourceforge.net/index.html>. Acesso em: 03 abr. 2013.

PYTHON, 2013. **About Python**. Disponível em: <http://python.org/about/>. Acesso em: 03 abr. 2013.

QUADROS, Daniel. **Microcontroladores – parte 5**, 2008. Disponível em: <http://dqsoft.blogspot.com/2008/08/microcontroladores-parte-5.html>. Acesso em: 12 jun. 2013.

RAMOS, Carlos; AUGUSTO, Juan Carlos; SHAPIRO, Daniel. **Ambient intelligence the next step for artificial intelligence.** IEEE Intelligent Systems, v. 23, n. 2, Nov. 2008, p. 15–18.

SANTANA, Osvaldo; GALESI, Thiago. **Python e Django: desenvolvimento ágil de aplicações web**. São Paulo: Novatec, 2010.

SICA, Carlos. **Sistemas automáticos com microcontroladores 8031/8051**. 1a. ed. São Paulo - SP: Novatec, 2006.

SQLITE, 2013. **About SQLite**. Disponível em <http://www.sqlite.org/about.html>. Acesso em: 03 abr. 2013.

VENTÄ, Olli. **Intelligent and systems. Technology theme - final report**. VTT, Espoo: VTT Publications, 2007.

WEBER, Rolf H. **Internet of things - need for a new legal environment?**. Computer Law & Security Review, 2009, p. 522-527.

WEISER Mark. **The computer for the twenty-first century**. Sci Am, v. 265, p. 94-104, 1991.

WONG, ChienYaw; MCFARLANE, Duncan; ZAHARUDIN, Alia A.; AGARWAL, Vivek **The intelligent product driven supply chain.** In: IEEEInternational Conference on Systems, Man and Cybernetics, vol. 4, 2002.

WU, Miao; LU, Ting-lie; LING, Fei-Yang; SUN, Ling; DU, Hui-Ying. **Research on the architecture of Internet of things**. 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)., 2010, v. 5, p. V5-484 – V5-487.