Gustavo de Oliveira Viana
Luan Batista da Silva

DESENVOLVIMENTO DE UM SISTEMA QUE POSSIBILITE O CONTROLE RESIDENCIAL VIA REDE LOCAL E INTERNET, UTILIZANDO BLACK BOARD E RASPBERRY PI

FACULDADES UNIFICADAS DE TEÓFILO OTONI
TEÓFILO OTONI – MG
2018

Gustavo de Oliveira Viana
Luan Batista da Silva

DESENVOLVIMENTO DE UM SISTEMA QUE POSSIBILITE O CONTROLE RESIDENCIAL VIA REDE LOCAL E INTERNET, UTILIZANDO BLACK BOARD E RASPBERRY PI

Monografia apresentada ao curso de Sistemas de Informação das Faculdades Unificadas de Teófilo Otoni como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação.

Área de Concentração: Automação.

Prof. Orientador: Luiz Fernando Alves Sousa

FACULDADES UNIFICADAS DE TEÓFILO OTONI
TEÓFILO OTONI – MG
2018

rede de ensino DOCTUM

FACULDADES UNIFICADAS DE TEÓFILO OTONI
NÚCLEO DE TCC / SISTEMAS DE INFORMAÇÃO
Autorizado pela Portaria 4.012 de 06/123/2004 – MEC

FOLHA DE APROVAÇÃO

A monografia intitulada: *Desenvolvimento de um sistema que possibilite o controle residencial via Rede Local e Internet, utilizando Black Board e Raspberry PI,*

elaborada pelos alunos Gustavo de Oliveira Viana
Luan Batista da Silva,

foi aprovada por todos os membros da Banca Examinadora e aceita pelo curso de Sistemas de Informação das Faculdades Unificadas de Teófilo Otoni, como requisito parcial da obtenção do título de

BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Teófilo Otoni, 5 de dezembro de 2018

Professor Orientador: Luiz Fernando Alves Souza

Professora Examinadora: Yvssa Carneiro Desmots Eliote

Professor Examinador: Marinho Soares de Souza

Não existe nada mais fatal para o pensamento que o ensino das respostas certas. Para isso existem as escolas: não para ensinar as respostas, mas para ensinar as perguntas. As respostas nos permitem andar sobre a terra firme. Mas somente as perguntas nos permitem entrar pelo mar desconhecido.

(Rubem Alves)

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

RESUMO

A presente monografia de conclusão do curso de bacharelado em Sistemas de Informação é baseada na área de automação, tendo como tema o desenvolvimento de um sistema que possibilite o controle residencial via rede local e internet, utilizando *Black Board* e *Raspberry Pi*, buscando proporcionar conforto e comodidade ao realizar tarefas rotineiras, localmente ou remotamente. Além de manter um investimento financeiro viável, com a utilização de recursos de baixo custo aquisitivo. Para o desenvolvimento deste trabalho, foi necessária pesquisa bibliográfica com assuntos relacionados a criação do sistema tais como: Automação, Desenvolvimento *Web*, *Python*, *Django*, C++, entre outros. A conclusão deste projeto resulta no desenvolvimento e implementação de um sistema de automação, em que se mostram as vantagens e desvantagens da adoção do mesmo em um ambiente residencial.

Palavras-Chave: Automação, *Python*, *Django*, *Black Board*, *Raspberry Pi*.

INTRODUÇÃO

A presente monografia concentra-se na área de automação residencial, tem como objetivo o desenvolvimento e implantação de um sistema de automação residencial que auxilie na comodidade de administração de uma residência por meio da rede local e internet, possibilitando uma melhor gestão das atividades domésticas rotineiras.

Atualmente à medida que a tecnologia avança, tudo ao seu redor a acompanha. O ser humano vai ficando cada vez mais apressado em relação a vida e suas atividades, já não se vive como antes e isso é um fato! Se várias coisas do cotidiano da maioria das pessoas são controladas e monitoradas 24 horas por dia remotamente, por que não controlar a própria casa?

Existem no mercado diversas tecnologias que possibilitam algum tipo de automação residencial, como por exemplo o Google Home e a Amazon Echo Dot Alexia, porém esta tecnologia não é tão acessível quanto a proposta deste projeto, que visa reduzir uma redução de custos.

Tendo em vista as diversas soluções já existentes nesta área, viu-se a necessidade de investigar a seguinte problemática: Seria viável o desenvolvimento de um sistema que possibilite o controle residencial via rede local ou internet, utilizando tecnologia de baixo custo?

Para este trabalho foram declarados e discorridos os seguintes objetivos específicos: (1) Reunir informações sobre tecnologias *WEB*, para o desenvolvimento da aplicação ou plataforma responsiva para controle da residência; (2) Analisar recursos que promovam comodidade e praticidade em uma residência utilizando tecnologia de automação; (3) Fazer um levantamento sobre as alterações na parte elétrica e eletrônica da residência que serão necessárias para implantação do

sistema; (4) Verificar tecnologias semelhantes no mercado, fazendo um levantamento de custos, analisando os equipamentos utilizados pelos mesmos; (5) Investigar como os sistemas de automação podem facilitar as tarefas residenciais, mantendo o investimento financeiro viável comparado com os disponíveis no mercado; (6) Desenvolver um sistema de automação que permita acesso via internet ou rede local, que promova mudança na realização das tarefas rotineiras em uma residência, agregando conforto e comodidade aos usuários.

Na criação deste projeto foram levantadas a seguintes hipóteses, buscando obter respostas e validação:

H0: Não seria viável o desenvolvimento de uma plataforma de automação residencial, pois essa tecnologia necessita de enormes investimentos financeiros, dispensando o interesse do público alvo.

H1: Seria viável o desenvolvimento de uma plataforma de automação residencial utilizando tecnologia de baixo custo, pois a mesma seria mais acessível comparada com recursos similares disponíveis no mercado.

H2: Seria viável o desenvolvimento de um sistema de automação residencial, pois o mesmo traria comodidade e praticidade.

H3: Não seria viável o desenvolvimento de uma plataforma de automação residencial, pois essa tecnologia requer muitas modificações na parte elétrica do local e poderia não ser aceita pelos proprietários.

H4: Seria viável o desenvolvimento de um sistema de automação residencial com gestão a distância utilizando tecnologia de baixo custo aquisitivo, pois o mesmo traria segurança adicional para a residência, comodidade e um possível auxílio as pessoas com deficiências relacionadas à mobilidade.

Para validar ou não as hipóteses levantadas, o sistema foi desenvolvido e implantando na área de lazer de uma casa, contando com a automação de alguns dispositivos distribuídos em vários cômodos. Esta área conta com um longo corredor, área gourmet, sinuca, lavanderia e jardim.

Quanto aos fins, as metodologias utilizadas foram aplicada, descritiva e laboratorial, pois gerou o conhecimento para aplicação e uma parte prática dirigida à solução do problema, visando uma utilidade econômica e social, através de um custo benefício válido, que possa compensar os investimentos. Para isto foram necessárias pesquisas na área tecnológica e científica, além de receber opiniões e sugestões do usuário final, gerando um *feedback* para o auxílio do desenvolvimento, a utilização de

artifícios e ferramentas para exemplificar o modo como o projeto foi concebido. Como método de teste do sistema, para evitar falhas no momento da implantação em um ambiente real, foi utilizado o teste laboratorial, permitindo a identificação de falhas ou demais modificações que se mostraram necessárias para uma melhor implantação.

Quanto aos meios, a pesquisa se classifica em dois tipos: exploratória, bibliográfica. Na área exploratória, trata-se de pesquisa sobre o assunto para aprofundar os conhecimentos, aquisição de conteúdo e experiência, auxiliando no desenvolvimento da solução para o problema em questão. No que tange a área documental, engloba-se o uso de *site*s, vídeo aulas, fotografias para agregar valor e conhecimento, para propor uma solução viável baseada em tais informações, para o problema descrito.

Nesta pesquisa fez-se o uso do método indutivo, partindo do pressuposto de que já existem várias tecnologias relacionadas a automação residencial no mercado. Estas tecnologias presentes no mercado foram analisadas, no que gira em torno dos equipamentos utilizados, recursos disponibilizados e custo final de implantação, considerando que o êxito da pesquisa depende de testes e de suas variáveis.

A organização deste trabalho foi feita em três capítulos, sendo eles: o primeiro concentra-se no referencial teórico, servindo como embasamento para o desenvolvimento deste projeto e foi disposto em subcapítulos que consideram as tecnologias e teorias para o desenvolvimento do sistema, como automação residencial, *Python*, *Django*, comunicação *serial*, dentre outros; o segundo apresenta ferramentas e técnicas para o desenvolvimento do sistema, bem como os testes realizados no decorrer implementação; o terceiro apresenta os resultados obtidos e analisados após a implantação, servindo para análise das hipóteses propostas.

1 REFERENCIAL TEÓRICO

1.1 Automação Residencial

Inicialmente é preciso entender sobre automação residencial, que se trata do foco principal deste projeto.

Segundo Muratori e Dal Bó (2011, p. 1), é possível entendê-lo como um conjunto de serviços possibilitados por meio de sistemas tecnológicos integrados, com o objetivo de satisfazer de maneira mais efetiva as necessidades básicas de comunicação, segurança, gestão elétrica e conforto de uma residência.

> A automação residencial também é conhecida como domótica (do latim “Domus” que significa casa e “robótica” controle automatizado de algo), casa inteligente, casa automática ou *retrofitting* (ato de introduzir uma modificação em algo já construído). Alguns autores costumam achar mais adequado o termo domótica, por ser mais abrangente e largamente utilizado na Europa, porém no Brasil é mais comum o uso da tradução literal do termo *home automation*, denominação americana mais restrita, ou automação residencial (MORATORI; DAL BÓ, 2011 apud BATISTELLO, 2014, p.19).

Com isto, é possível identificar como automação residencial quaisquer tecnologias que permitem tornar automáticas as atividades e tarefas dentro de uma residência.

De acordo com Prudente (2011), a automação pode ser aplicada em todas as atividades de uma residência, desde um simples controle de iluminação, até sistemas de segurança e vigilância mais complexos.

Vantagens obtidas como consequência da aplicação e utilização de uma automação residencial segundo Prudente (2011):

- Torna o ambiente mais acolhedor e agradável, agregando conforto e qualidade de vida para os envolvidos;

- Possibilita a identificação de incidentes podendo tomar as devidas precauções, garantindo maior segurança na habitação;
- Maior economia, pois garante uma gestão eficaz da instalação, com controle da iluminação, sistema de ar-condicionado, entre outros.
- Podem trazer benefícios para pessoas com dificuldades de locomoção.

Muitas pessoas ainda pensam nessa questão com certa visão futurista, o potencial de crescimento deste mercado é enorme, tanto que através dele surgiram novas profissões, como técnico em automação e engenheiro de automação (MORATORI; DAL BÓ, 2011 apud BATISTELLO, 2014, p.20).

Existem diversas tecnologias que podem ser utilizadas no desenvolvimento de sistemas de automação, como a solução proposta pela *Google*, o *Google Home*, o seu concorrente *Amazon Echo*. Neste trabalho pretende-se utilizar um microcomputador *Raspberry Pi* e um microcontrolador *Black Board*.

1.2 Microcontrolador

Microcontroladores no geral são ferramentas utilizadas para executar tarefas constantes visto que são facilmente programáveis e de baixo custo. Em sua atualidade estão presentes em diversos tamanhos e formas para atender quase todas as áreas de mercado industrial e mais recentemente chegando às casas. (TAVARES, 2013, p. 1).

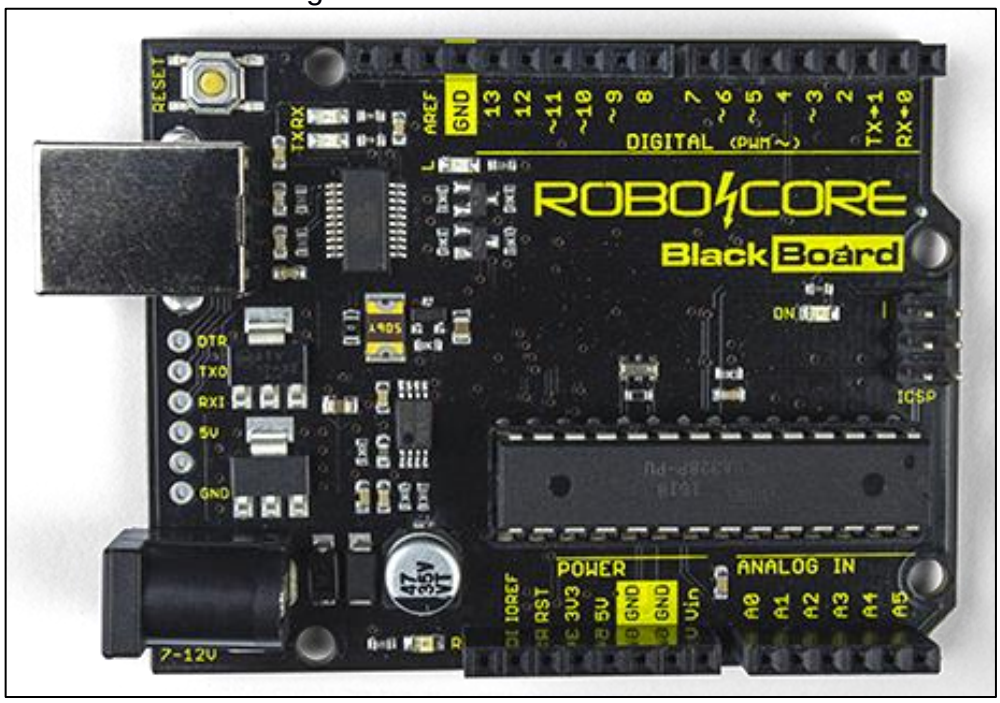
Dentro do ambiente computacional e da eletrônica digital nota-se a crescente utilização dos microcontroladores, tanto nas indústrias (circuitos para automação e gerenciamento, ferramentas de bancada como multímetros e frequencímetros, dentre outros) quanto em residências (portões eletrônicos, alarmes, impressoras, telefones). Porém não é dada a devida atenção, ou por falta de conhecimento ou por ser tratado como um simples Circuito Integrado (CI). Um microcontrolador é um CI capaz de efetuar processos lógicos com extrema rapidez e precisão. A grande vantagem deste CI é a sua possibilidade de programação, o que o torna adaptável à finalidade desejada, e que possibilita seu ajuste de acordo com a tarefa que deverá executar (ASSIS, 2004, p. 15).

1.2.1 *Black Board*

O microcontrolador *Black Board* pode ser visto como o irmão brasileiro do famoso italiano *Arduino*. Em sua essência, a *Black Board* é totalmente compatível com *softwares* e incrementos utilizados pelo irmão italiano, porém brasileira e de código aberto (ROBOCORE, 2016)[1].

A *Black Board* foi lançada no início de 2014 pela RoboCore[2], com o intuito de ser uma alternativa para o *Arduino* Uno no Brasil. O principal impulso para a fabricação *Black Board* foi o alto custo da comercialização das placas italianas, devido à alta carga tributária que incide sobre importação das mesmas. Outro incentivo é o rigoroso controle sobre a produção e os testes mantidos pela empresa nacional, o que não é possível na revenda dos modelos importados (ROBOCORE, 2018).

Figura 1 – *Black Board* Uno R3



Fonte: <https://www.robocore.net/loja/produtos/*Arduino-Black Board*.html#descricao>

A seguir no Quadro 1 é possível visualizar algumas especificações técnicas da *Black Board*.

---

1 Disponível em: <https://www.robocore.net/loja/produtos/arduino-blackboard.html#descricao>
2 Disponível em: <https://www.robocore.net/>

Quadro 1 – Características *Black Board* Uno R3

Tamanho:	5,3cm x 6,8cm x 1,0cm
Microcontrolador:	ATmega328
Tensão de operação:	5V
Tensão de entrada (recomendada):	7-12V
Tensão de entrada (limites):	6-20V
Pinos de entrada/saída (I/O) digitais:	14 (dos quais 6 podem ser saídas PWM)
Pinos de entrada analógicas:	6
Corrente DC por pino I/O:	40mA
Corrente DC para pino de 3,3V:	50mA
Memória Flash:	32KB (dos quais, 0,5KB são usados pelo bootloader)
SRAM:	2KB
EEPROM:	1KB
Velocidade de Clock:	16MHz
Temperatura de operação:	de 10º a 60ºC

Fonte: <https://www.embarcados.com.br/placa-*Arduino*-da-robocore-*Black Board*/>

Segundo a RoboCore (2016)[3] algumas diferenças notáveis entre a *Black Board* e o *Arduino* que estão em sua construção são:

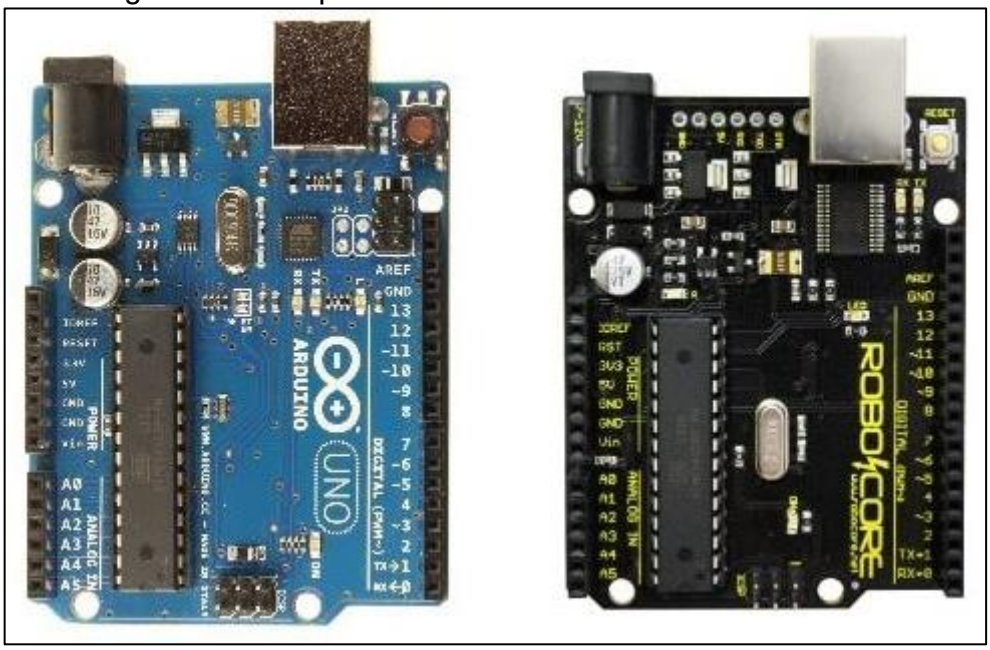
- Possui saídas de 5 volts que suportam mais conexões por porta, aumentando assim a quantidade de periféricos e servos que podem ser ligados por saída.
- É capaz de fornecer maior corrente em seu pino de alimentação 5 Volts para circuitos externos quando comparada com as placas *Arduino* UNO;
- Conta com um driver FTDI mais robusto e mais confiável que o italiano *Arduino*, fazendo assim a comunicação com o computador ocorrer mais fluidamente e com menos erros de compatibilidade com o sistema operacional;
- É fabricado no Brasil, o que gera um custo benefício muito maior tornando-a mais acessível e barata, além de valorizar a indústria brasileira.

Como a *Black Board* é muito compatível com o *Arduino*, tanto que recebe a classificação de “versão brasileira do *Arduino*”, será tomada como base a

[3] Disponível em: <https://www.robocore.net/loja/produtos/arduino-blackboard.html#descricao/>

documentação do *Arduino*, por conta da documentação disponibilizada pela Robô Core ainda ser muito simplificada/básica a respeito da *Black Board*.

Figura 2 – Comparativo *Black Board* Uno e *Arduino* Uno



Fonte: <https://www.embarcados.com.br/placa-*Arduino*-da-robocore-*Black Board*/>

Como foi demonstrado na Figura 2, os microcontroladores são muito semelhantes até mesmo fisicamente.

Segundo a documentação no *site oficial*[4], a programação do *Arduino* é dividida em três partes principais: Estruturas, valores (variáveis e constantes) e funções. Existindo assim uma gama enorme de recursos e funções já embutidos e disponibilizados no *site oficial*. O *Arduino* pode ser programado com o *Arduino Software IDE*.

De acordo com a documentação *Arduino* (2018), o *Arduino* possui pinos digitais e analógicos, que poder ser lidos e controlados, servindo como base para utilização de recursos/dispositivos que são associados a eles. A seguir tem-se a apresentação e definição das principais funções, segundo o *site oficial*[5]:

analogRead(): esta função lê o valor de um pino analógico especificado. A placa possui um conversor analógico-digital 6 canais, isso significa que este irá mapear tensões entre 0 e 5 volts para inteiros entre 0 e 1023. **Sintaxe:** analogRead(pino).

---

4 Disponível em: <https://www.arduino.cc/reference/pt/#>
5 Disponível em: <https://www.arduino.cc/reference/pt/>

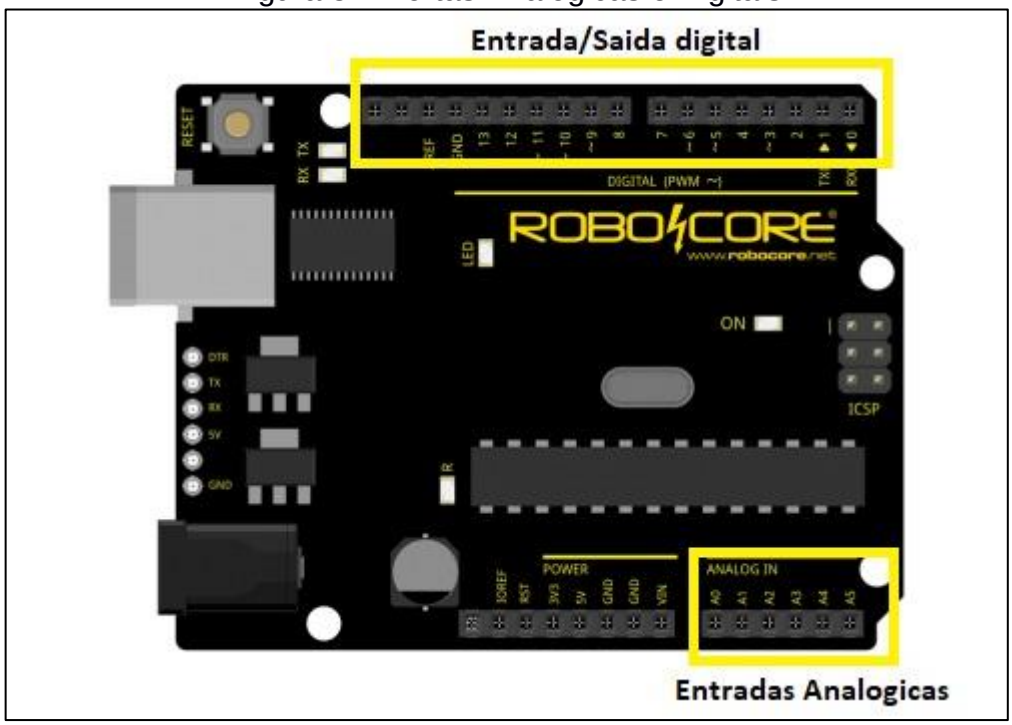
analogWrite(): a função aciona uma onda em um pino, possibilitando variar o brilho de uma LED ou até mesmo acionar um motor a diversas velocidades por exemplo. **Sintaxe:** analogWrite (pino, valor).

digitalRead():esta função lê o valor de um pino digital especificado, que pode ser HIGH ou LOW identificando se o mesmo está ativo ou inativo. **Sintaxe:** digitalRead(pino).

digitalWrite esta função aciona um valor *HIGH* ou *LOW* em um pino digital ativando ou desativando o mesmo. **Sintaxe:** digitalWrite(pino, valor) onde o valor somente pode receber *HIGH* ou *LOW*.

Após expor sobre a leitura das portas analógicas e digitais, na Figura 3 a seguir é possível visualizar os pinos e sua disposição no microcontrolador *Black Board*.

Figura 3 – Portas Analógicas e Digitais



Fonte: <https://www.robocore.net/>

A placa pode ser alimentada via conexão USB ou com uma fonte de alimentação externa. O intervalo indicado para as fontes externas que podem ser utilizadas na placa deve ser de 7 a 12 volts (*ARDUINO*, 2018).

Existe uma série de recursos para prover a comunicação entre a placa e um computador ou até mesmo com outros microcontroladores. A placa possui um método

que canaliza a comunicação serial via USB e aparece como uma porta de comunicação virtual para o computador, para isso não é necessário nenhum drive externo. A comunicação serial é feita através dos pinos 0 (RX) e 1 (TX), portanto não será possível utilizar a comunicação USB ao mesmo tempo dos pinos 0 e 1 como entradas ou saídas digitais. Para monitorar a comunicação é possível utilizar a ferramenta **Monitor Serial**, disponível no *Arduino Software IDE* (*ARDUINO*, 2018).

Neste projeto será utilizada a comunicação serial, entre a *Black Board* e o *Rapberry Pi*; Com isso tem-se a documentação completas das funções que operam sobre a comunicação serial no *site oficial* do *Arduino*[6], onde as mais comuns são:

Serial.avaliable(): Retorna o número de *bytes* disponíveis para leitura da porta serial, dados estes que já foram recebidos e guardados no *buffer*[7].

Serial.read(): Lê os dados recebidos na porta serial.

Serial.print(): Imprime na porta serial em texto ASCII[8] (facilmente legível, diferente dos valores binários).

No Quadro 2 nota-se um exemplo de utilização da comunicação Serial.

Quadro 2 – Comandos Serial *Black Board* Uno R3

```
int incomingByte = 0;   // variável para o dado recebido

void setup() {
        Serial.begin(9600);     // abre a porta serial, configura a taxa de transferência para 9600 bps
}

void loop() {

        // apenas responde quando dados são recebidos:
        if (Serial.available() > 0) {
                // lê do buffer o dado recebido:
                incomingByte = Serial.read();

                // responde com o dado recebido:
                Serial.print("I received: ");
                Serial.println(incomingByte, DEC);
        }
}
```

Fonte: <https://www.*Arduino*.cc/reference/pt/language/functions/communication/serial/read/>

Como a maioria das plataformas de programação, o ambiente *Arduino* pode ser estendido através do uso de bibliotecas, as bibliotecas fornecem funcionalidade extra para o desenvolvimento. O *Arduino Software IDE* já traz consigo várias bibliotecas

---

6 Disponível em: <https://www.arduino.cc/reference/pt/language/functions/communication/serial/>

7 É uma região de memória física utilizada para armazenar temporariamente os dados enquanto eles estão sendo movidos de um lugar para outro. Disponível em: <https://www.merriam-webster.com/dictionary/buffer/>

8 ASCII é a representação numérica de um caractere como 'a' ou '@' ou uma ação de algum tipo. Disponível em: <http://www.asciitable.com/>

pré-definidas ao ser instalado. Existem também diversas bibliotecas que são desenvolvidas e disponibilizadas livremente. Para facilitar e até mesmo encorajar os desenvolvedores no *site* do *Arduino* é possível encontrar um tutorial/procedimento para que seja possível desenvolver suas próprias bibliotecas[9] (*ARDUINO*, 2018).

O *Arduino* pode ser programado em duas linguagens um pouco semelhantes, C ou C++. Existem diversos projetos que utilizam *Arduino* e são disponibilizados livremente[10] no GitHub[11]. Observando tais projetos é possível identificar uma superioridade na utilização da linguagem C++. Além disso as melhorias que C++ disponibiliza em relação à linguagem C favoreceram a escolha de C++ como linguagem para ser utilizada neste projeto (*ARDUINO*, 2018).

1.2.1.1 Módulo Relé Serial

O módulo relé serial é indicado para projetos envolvendo *Arduino*/*Black Board* onde haja a necessidade de controlar vários dispositivos, reduzindo o uso das portas dos microcontroladores. Este módulo consegue controlar quatro relés da placa independente, utilizando apenas dois pinos do microcontrolador. É possível escolher exatamente qual relé a ser acionado através da comunicação serial. Além disso existe a possibilidade de adicionar mais módulos relés serial um atrás do outro, tornando possível acionar ainda mais relés usando as mesmas duas portas. Para utilizar este módulo com o microcontrolador, o mesmo deve estar obrigatoriamente alimentado com uma fonte 12v, e por conta das bobinas dos relés serem de 12v, esta alimentação é feita por meio do pino ***Vin*** (RoboCore, 2016).

---

9 Disponível em: <https://www.arduino.cc/en/Hacking/LibraryTutorial>
10 Disponível em: <https://github.com/search?q=Arduino>
11 Disponível em: <https://github.com/about>

Figura 4- Módulo Relé Serial

Fonte: https://www.robocore.net/loja/produtos/modulo-rele-serial.html#descricao

Para controlar este módulo, a própria RoboCore (2015) disponibiliza uma biblioteca de forma livre[12]. As principais funções e as únicas utilizadas neste projeto são:

GetState(): Que retorna individual de um dos relés do módulo, esse estado do relé é apenas a nível de código, pois no módulo não existe um meio de identificação dos estados individuais dos relés.

SetModule(): É responsável por definir um estado para o relé (ligado ou desligado), independe de sua situação atual, se tudo ocorrer bem a função retorna um valor verdadeiro, caso contrário retorna *false*.

Com isso tem-se um dos principais componentes deste projeto, que é o microcontrolador *Black Board*, sendo ele o principal responsável pela automação proposta, possibilitando o controle/gestão dos equipamentos desejados de acordo com a necessidade.

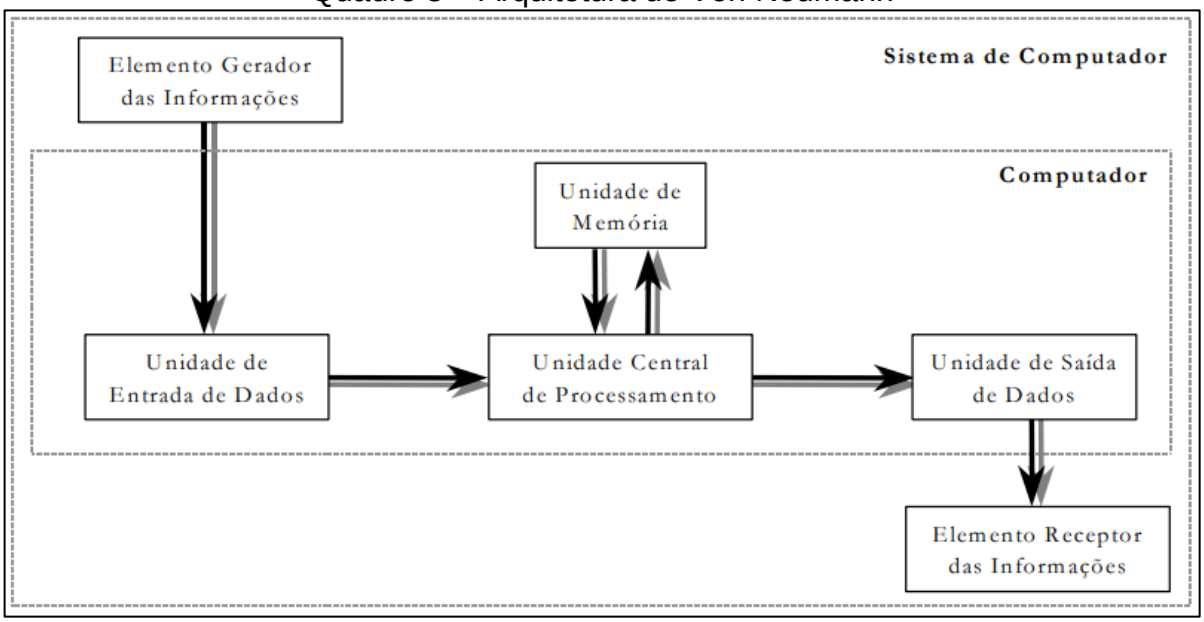
1.3 Microcomputadores

[12] Disponível em: <https://github.com/RoboCore/SerialRelay>

Mesmo com as enormes transformações na área de Eletrônica, os microcomputadores atuais ainda mantêm a mesma concepção funcional dos primeiros computadores eletrônicos (concepção conhecida como *Arquitetura de Von Neumann*), porém cada vez menores fisicamente (CATHARINA, 2000, p.1).

> Uma unidade central de processamento recebe informações através de uma unidade de entrada de dados, processa estas informações segundo as especificações de um programa armazenado em uma unidade de memória, e devolve os resultados através de uma unidade de saída de dados (CATHARINA, 2000, p.1).

Quadro 3 – Arquitetura de Von Neumann



Fonte: CATHARINA, 2000, p.1.

O Quadro 3 demonstra a arquitetura de computador que se caracteriza pela possibilidade de uma máquina digital, com entrada – processamento – saída.

1.3.1 *Raspberry Pi*

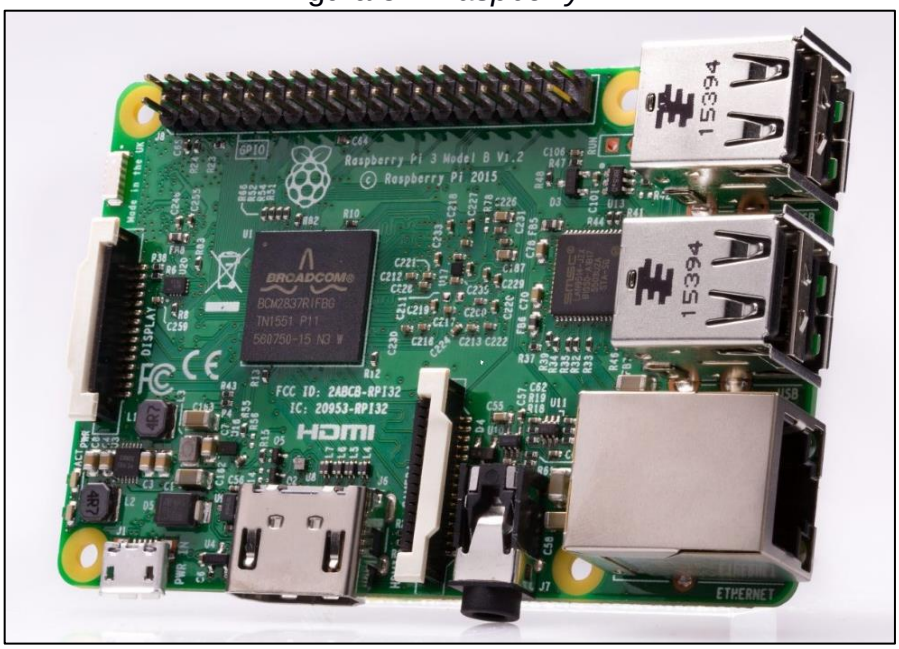
Os computadores já estão presentes no cotidiano de muitos e podem certamente ser considerados indispensáveis no uso profissional e muitas vezes no pessoal. Eles fazem parte de diversas tarefas diárias, auxiliando na comunicação, otimização e automação de tarefas (NETTO, 2013).

O *Raspberry Pi* pode ser compreendido como um dos menores computadores do mundo, com seu tamanho comparado ao de um cartão de crédito, o Pi é derivado do py que é abreviação de *Python* e tendo o Linux como sistema operacional padrão. Ele é indicado para realização das tarefas de um computador comum. O desenvolvimento dessas máquinas teve início em 2006, visando um computador flexível o bastante para ser uma ferramenta de aprendizado, com baixo custo de fabricação, porém o projeto não parou aí, o seu potencial foi descoberto no decorrer do seu desenvolvimento, e o seu uso deixou de ser apenas acadêmico (NETTO, 2013).

> O *Raspberry Pi* nada mais é do que um computador completo classificado como SBC – Single Board Computer, ou seja, um computador montado em uma única placa com processador, memória, I/O e outros componentes necessário para seu funcionamento (MERCES, 2013, p. 7).

Segundo Merces (2013, p.8), comparando o tamanho com a capacidade, o *Raspberry Pi* não foi criado com o propósito de realizar tarefas que envolvem enormes recursos de processamento, porém se comparado com tempos atrás, onde ninguém imaginava que seria possível carregar no bolso um computador, ele se torna muito útil, podendo até mesmo realizar tarefas de um servidor. Atualmente são disponíveis duas versões no mercado, o modelo pi zero, pi 1 modelo A e A+, pi 1 modelo B e B+ e o pi 3 modelo B e B+. A variação de preço entre as versões é aproximadamente 10 dólares. Além das diferenças na memória RAM, somente o modelo B possui interface de rede embutida. As dimensões deste equipamento são 8.6cm x 5.4cm x 1.5cm (Comprimento, Largura e Altura, respectivamente) podendo variar de acordo com o modelo. Segue a Figura 5 representando o modelo B para melhor entendimento:

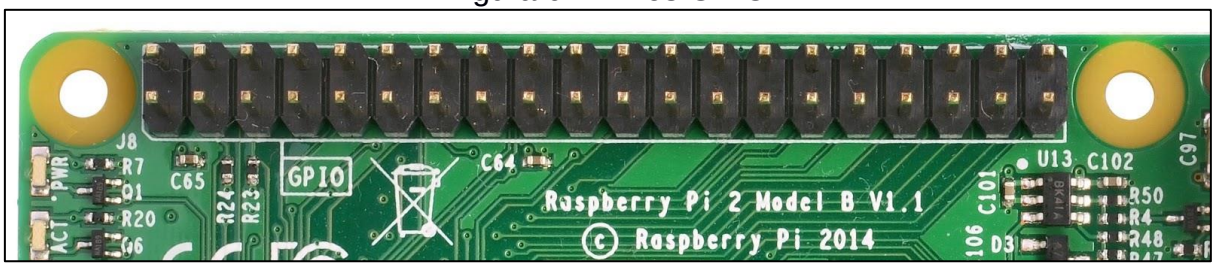
Figura 5 – *Raspberry Pi*



Fonte:<https://www.raspberrypi.org/products/>

O *Raspberry Pi* é alimentado por uma fonte externa de socket micro USB com voltagem padrão de 5v (volts) e 2.5 A (amperes) o que é relativamente uma voltagem baixa comparada com voltagens de computadores tradicionais, possui uma entrada de cartão micro SD que trabalha como armazenagem de dados primária, ou seja, todos os dados do sistema e futuros *downloads* ou configurações são armazenados no mesmo, além de componentes gerais de qualquer computador convencional como placa *wifi*, placa de *bluetooth*. A grande sacada por assim dizer do Raspberry são os pinos GPIO (entrada e saída de propósito geral) auto configuráveis como exemplificado na figura 6 (MONKY, 2016, p.17).

Figura 6 – Pinos GPIO

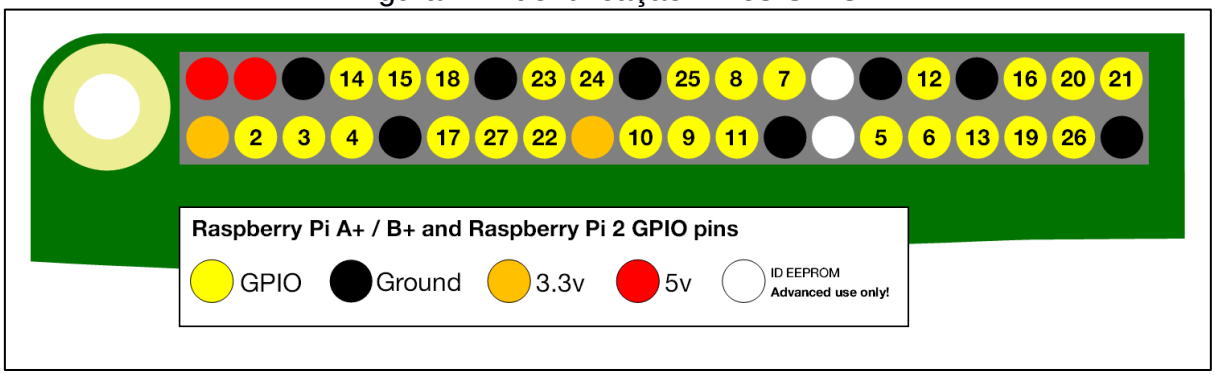


Fonte: <https://www.raspberrypi.org/documentation/usage/gpio/>

Criado no reino unido com o propósito de ser de baixo custo e de ensinar conceitos básicos de computação, principalmente em *Python*, tinha-se a necessidade de uma área programável na prática por assim dizer. Os pinos GPIO servem

justamente para demonstrar o uso da programação para integrar o Raspberry com outras tecnologias como sensores e servos. A figura 7 demonstra melhor sobre as funções dos pinos GPIO (*RASPBERRY PI* FUNDATION, 2018).

Figura 7 – Identificação Pinos GPIO



Fonte: <https://www.raspberrypi.org/documentation/usage/gpio/>

Sinalizados de amarelo, os pinos GPIO tem por padrão entrada e saída de informações em uma corrente padrão de 3,3V; em vermelho estão os pinos de corrente 5V que atende a periféricos com maior demanda de energia; em preto, os pinos GND ou *ground* que em uma explicação mais simples são o terra da placa onde se liga o lado negativo da corrente; os pinos marcados com laranja e não numerados são pinos apenas de voltagem 3.3V para alimentação de periféricos; em branco dois pinos reservados para finalidades avançadas da placa (*RASPBERRY PI* FUNDATION, 2018).

1.3.1.1 *Raspbian*

Segundo Oliveira, Carissimi e Toscani (2010, p. 22-23), Sistema Operacional é a camada de *software* colocada entre o *hardware* e as aplicações de usuário. É ele que possibilita a interação/comunicação entre o *hardware* (periféricos) e o *software*, além de gerenciar os recursos computacionais da máquina. Ele é o responsável pelo acesso aos periféricos. Quando alguma aplicação necessita de determinada operação de entrada e saída, será solicitada ao sistema operacional. Dessa forma não é responsabilidade do programador conhecer detalhes do *hardware* e desenvolver especificamente para cada um individualmente. Informações de como se comunicar

com cada periférico ficam “escondidas” dentro do Sistema Operacional. Com isso fica o Sistema Operacional responsável por identificar e gerir qual aplicação está acessando determinado recurso, conseguindo assim uma distribuição justa e eficiente dos recursos da máquina.

Segundo a documentação oficial do *Raspbian*[13], ele é uma distribuição Linux[14] criada para rodar nos *Raspberry Pi*. Derivada do Debian, essa distribuição é considerada o sistema operacional padrão do computador da *Raspberry Foundation*. Completo e com diversos *softwares* de desenvolvimento, tem profundo controle sobre o *hardware* da placa, além de ferramentas de acesso à Internet, de escritório na forma do *LibreOffice* e de entretenimento, o *Raspbian* é o ponto de partida ideal para quem está começando a usar a placa.

A compilação inicial de mais de 35.000 pacotes *Raspbian*, otimizada para melhor desempenho no *Raspberry Pi*, foi concluída em junho de 2012. Além disso conta com versão gráfica facilitando a utilização. No entanto, *Raspbian* ainda está em desenvolvimento ativo (*RASPBIAN*, 2012).

O *Raspbian* não possui participação na Fundação *Raspberry Pi*, ele foi desenvolvido por uma equipe pequena e dedicada de programadores e entusiastas do *hardware Raspberry Pi*, além de adeptos dos objetivos educacionais da *Raspberry Pi* Foundation (IDEM)

1.4 Rede de Computadores

Segundo Tanenbaum e Wetherall (2011, p.17), uma rede de computadores é formada pela troca de informações entre várias máquinas, podendo também haver compartilhamento de recursos entre as mesmas, sendo essa rede interligada por um sistema de comunicação de meios de transmissão e protocolos.

Tendo em vista a necessidade atual, as Redes de Computadores são de extrema importância tanto residencial como empresarial, pois possibilitam

---

13 Disponível em: <https://www.raspberrypi.org/documentation/>

14 O Linux é um sistema operacional completo, desenvolvido pelo finlandês Linus Torvalds. Um clone do sistema Unix, sua primeira versão a 0.01 foi liberada em 1991, essa versão continha 9.300 linhas de códigos escritos em C e mais 950 linhas em linguagem Assembly (TANENBAUM, 2009).

comunicação e envio de arquivos rapidamente além de recursos como impressoras, *scanners* e outros dispositivos (NASCIMENTO, 2011).

Com tanto avanço das redes foram criadas denominações para as mesmas. Dentre diversas encontram-se as redes locais, nas quais é possível comunicar com máquinas conectadas ao mesmo ponto central, seja ele um servidor ou um *switch*, as LANs (Local Area Networks), como são conhecidas, podem ser conexões locais com fios, sem fios ou mistas (TANENBAUM, 2011).

1.5 Banco de Dados

Segundo Elmasri e Navathe (2005, p.3), “um Banco de dados é um conjunto de dados relacionados. Os dados são fatos que podem ser gravados e alterados e que possuem um significado subentendido”.

Os bancos de dados são geralmente formados e alimentados por dados acerca de um objetivo específico, possuindo usuários com uma hierarquia de permissões definidas e algumas aplicações pré idealizadas de acordo com a necessidade dos usuários para garantir a confidencialidade das informações, mantendo somente os acessos necessários para cada usuário (ELMASRI; NAVATHE, 2011, p.3).

De acordo com os mesmos autores, um banco de dados é um conjunto de dados com um significado implícito. Um exemplo bem utilizado para exemplificar os bancos de dados são agendas de contatos telefônicos. Trazendo para a atualidade seria uma agenda de contatos no celular onde se encontra dados sobre pessoas conhecidas e cada dado tem um valor subentendido como, endereço, número de telefone, nome, idade, local de trabalho, telefone secundário dentre outros dados.

1.5.1 Sistema Gerenciador de Banco de Dados – SGBD

Os bancos de dados dão suporte para as operações de um determinado projeto, eles são vitais para o andamento de diversos processos. Por exemplo: se for levada em conta uma organização moderna, os BDs são essenciais para as que

pretendem se manter competitivas no mercado, devido ao cenário atual de extrema concorrência (CARVALHO, 2017, p.4)

Os SGBDs surgiram da evolução dos sistemas de arquivos de armazenamento em disco, criando novas estruturas de dados com o objetivo de armazenar informações, passando a utilizar diferentes formas de representação, ou modelos de dados, para que fosse possível descrever a estrutura das informações contidas em seu banco. Os modelos de dados mais utilizados pelos SGBDs são: modelo Hierárquico, modelo de redes, modelo relacional (amplamente usado) e o modelo orientado a objetos (TAKAI, 2005, p. 6).

Os bancos de dados gerenciam de forma automatizada os dados nele armazenados, o modelo de banco de dados relacional é o mais difundido, principalmente por sua capacidade de manter a integridade dos dados após alguma alteração nas estruturas das tabelas (CARVALHO, 2017, p.5)

1.5.2 Modelo de Dados

Um modelo de dados entende-se por conjuntos de conceitos que podem ser usados para atribuir descrição a estrutura de Banco de Dados (NAVATHE, 2005, p.19).

Segundo Navathe (2011, p. 20, 90), a estrutura pode ser descrita como a representação de algo real quem tem importância diante da aplicação, atributos “correspondem a alguma propriedade que ajuda a descrever uma entidade”, e relacionamento sendo a comunicação de uma ou mais entidades. Como neste projeto pretende-se utilizar o modelo relacional, serão expostos alguns conceitos do mesmo. Uma entidade representa um objeto do mundo real, como um funcionário, que são descritos no banco de dados, os atributos são propriedades ou características das entidades. Relacionamento entre duas ou mais entidades representam a associação entre estas, como por exemplo a entidade funcionário trabalha em um projeto, onde funcionário e projeto são entidades.

O modelo foi utilizado neste projeto é o relacional por conta dos seus benefícios e da familiaridade dos desenvolvedores deste projeto com o mesmo. O modelo de dados relacional pode ser entendido como um conjunto de tabelas relacionadas,

contando com uma estrutura básica com **colunas**, **linha** e **campos**. Os mecanismos que interligam as tabelas no modelo relacional agregam segurança ao trabalhar com um banco de dados (CARVALHO, 2017, p.5-6)

1.6 Programação *Web*

Segundo Macedo (2014, p.3) as aplicações desenvolvidas para o ambiente *web*, através de solicitações cliente servidor, representam um conceito baseado em acessibilidade e portabilidade. Podem ser consideradas aplicações *web*, quaisquer sistemas executados por meio de um navegador, seja através da internet ou redes locais, localizadas em um servidor *web*.

> O navegador solicita ou envia informações a ser processada no servidor, este devolve ao cliente onde o navegador imprime o resultado, se difere de um *site*, pois existe um nível significativo de regras e interações das informações entre os lados de servidor e cliente (TURBAN; MACLEAN; WETHERBE, 2002. apud MACEDO, 2014, p. 11).

1.6.1 Linguagem de Marcação - HTML5

Segundo Pedroso (2007, p. 3) HTML é uma linguagem de hipertextos, normalmente empregada na construção de páginas da Internet. E de acordo com Carvalho (2004, p.3), HTML – *HyperText Markup Language*, ou a tradução Linguagem de Formatação de Hipertexto, é o fruto da junção dos padrões *HyTime* e SGML.

A linguagem HTML (*Hypertext Markup Language*) tem o objetivo de formatar textos através de marcações especiais denominadas *tags*, para que possam ser exibidos de forma conveniente pelos clientes *Web*, também denominados navegadores ou browsers. Além disso, esta linguagem possibilita a interligação entre páginas da *Web*, criando assim documentos com o conceito de hipertexto (PEDROSO, 2007 p.5).

As principais características do HTML segundo Pedroso (2007, p.5) são:

- São arquivos de texto escritos em ASCII;
- Trata-se de uma linguagem que não é *Case Sensitive*, ou seja, não faz diferença entre letras maiúsculas e minúsculas em suas marcações.
- Nem todas suas marcações são suportadas por qualquer navegador.

HyTime - Hypermedia/Time-based Document Structuring Language: *HyTime* (ISO 10744:1992) - padrão para representação estruturada de hipermídia e informação baseada em tempo. Um documento é visto como um conjunto de eventos concorrentes dependentes de tempo (áudio, vídeo, etc.), conectados por *webs* ou *hiperlinks*. O padrão *HyTime* é independente dos padrões de processamento de texto em geral. Ele fornece a base para a construção de sistemas hipertexto padronizados, consistindo de documentos que aplicam os padrões de maneira particular (CARVALHO, 2004, p.3).

Outro conceito que torna-se necessário para o entendimento do funcionamento do HTML é, o SGML:

SGML - *Standard Generalized Markup Language*: Padrão ISO 8879 de formatação de textos; não foi desenvolvido para hipertexto, mas torna-se conveniente para transformar documentos em hiperobjetos e para descrever as ligações. SGML não é padrão aplicado de maneira padronizada: todos os produtos SGML têm seu próprio sistema para traduzir as etiquetas (*tags*) para um particular formatador de texto (IBIDEM).

DTD - *Document Type Definition*: Define as regras de formatação para uma dada classe de documentos. Um DTD ou uma referência para um DTD deve estar contido em qualquer documento conforme o padrão SGML, com isso o HTML é definido como um DTD de SGML (IBIDEM).

Segundo Carvalho (2004, p. 3), todo documento HTML apresenta elementos parênteses angulares (< e >). Assim esses elementos definem as *tags*/etiquetas do HTML, que são comandos de formatação da linguagem.

Com o HTML, não é possível manipular ou processar dados, nem mesmo enviá-los ao servidor ou a outra máquina qualquer, para estas tarefas, surge a necessidade de utilizar um programa que seja capaz de manipular e processar tais dados em páginas *Web* (SILVA, 2006, p.23).

Para criar uma página HTML não é necessário nenhum *software* específico, muito menos um computador potente. Basta um computador que suporte um editor de texto simples e um navegador de internet. Veja no Quadro 5 abaixo um exemplo de uma página HMTL.

Quadro 5 – Exemplo HTML

```
<html>
<head>
<title>Meu site</title>
</head>
<body>
Alô Mundo
</body>
</html>
```

Fonte: QUIERELLI, 2012, p.7

A *tag* **<hmtl>** inicia e encerra o documento, o cabeçalho da página é definido pela *tag* <**head>**, onde são inseridas informações que facilitam a localização/identificação da página por indexadores. A parte que será visível para os usuários da página é definida pela *tag* **<body>**, este trecho do código também é identificado com corpo do documento (QUIERELLI, 2012, p.7-8)

1.6.2 Linguagem de Programação JavaScript

JavaScript é uma linguagem desenvolvida pela *Netscape* em parceria com a *Sun Microsystems*, com a finalidade de trabalhar com aplicações interativas nas páginas *web* (HTML). A sua primeira versão recebeu o nome de Java Script 1.0, e foi lançada em 1995 (SILVA, 2006, p.23).

Os comandos JavaScript são embutidos nas páginas HTML e interpretados pelo navegador, não necessitando de nenhum processo de compilação (LIMA, 2006, p.9).

Segundo Silva (2006, p.23), Java Script é uma linguagem desenvolvida para rodar no lado do cliente, ou seja, a interpretação e o funcionamento da linguagem dependem de funcionalidades hospedadas no navegador do usuário. Isto somente é possível porque existe um interpretador Java Script hospedado no navegador.

A linguagem JavaScript de modo geral é uma linguagem de programação dinâmica, sendo capaz de realizar virtualmente qualquer tipo de aplicação, possuindo características do paradigma de orientação a objetos (GRILLO; FORTES, 2008, p. 3)

O código JavaScript pode ser mantido tanto dentro do documento HTML quanto externamente em um arquivo separado. Se o mesmo for mantido internamente, será necessário adicionar o código entre as *tags* **<script>** e **</script>**, caso seja mantido de forma externa o arquivo deverá ser salvo com a extensão **.js** e um *link* para ele deve ser adicionado na página HTML. Como existem outras linguagens de script do lado cliente, é necessário especificar a linguagem de script que está sendo utilizada, sendo assim a *tag* passa a ser **<script type=”text/javascript>**. Veja o Quadro 6 abaixo, mostra um exemplo da utilização de JavaScript para exibir a data atual no formato “dd/mm/aaaa” (PRESCOTT, 2016, p.5-6)

Quadro 6 – Exemplo JavaScript

```
<html>
<head>
<title> meu primeiro código JavaScript interno!!!</title>
<meta charset="UTF-8">
<script type="text/javascript">
var hoje = new Date();
var dd = hoje.getDate();
var mm = hoje.getMonth()+1;
var aaaa =  hoje.getFullYear();
if(dd<10)
dd = '0' + dd;
if(mm<10)
mm = '0' + mm;
hoje = dd+ '-'+mm+'-'+aaaa;
document.write("<b>" + hoje + "</b>");
</script>
</head>
<body>
</body>
</html>
```

Fonte: PRESCOTT, 2016, p.6

1.6.3 Linguagem de Folhas de Estilo – CSS3

Cascading Style Sheet (CSS), traduzido para o português como folhas de estilo em cascata tem sua definição mais precisa e simples como mecanismo para adicionar estilos aos documentos *web* (SILVA, 2011, p.24).

Segundo Quierelli (2012, p.10-11), o CSS permite, além de formatar textos e imagens, criar divisões que podem ser utilizadas no *layout* da página, facilitando a disposição dos elementos na mesma e agregando flexibilidade para o desenvolvedor.

O CSS tem como objetivo proporcionar ao HTML/XML que são linguagens de estruturação, a inclusão de aspectos visuais como cores de fontes, tamanho dos textos, posicionamento, dentre outros. Estes recursos não são responsabilidade do HTML, mas sim do CSS; resumindo: o “HTML para estruturar e o CSS para apresentar” (SILVA, 2011, p.24-25).

O arquivo CSS que servirá para determinada página pode ser criado utilizando um editor de texto comum devendo ter a extensão **.css** e ser importado da maneira correta na página, também podendo ser feito diretamente na página HTML como no exemplo no Quadro 7 a seguir (QUIERELLI, 2012, p.9).

Quadro 7 – Exemplo CSS

```
<html>
<head>
<title>PHP</title>
<style type="text/css">

Comentário: Cria uma caixa com o nome “principal” com borda largura de 900px.
#principal {
      border:solid 1px #CCC;
      width:900px;
      margin:auto;
}
</style>
</head>
<body>

Comentário: Faz a chamada da caixa “principal” centralizada dentro da tag div.
<div align="center" id="principal">
      Alô Mundo!
</div>
</body>
</html>
```

Fonte: QUIERELLI, 2012, p.9

1.7 Frameworks

Frameworks são ferramentas utilizadas para o auxílio da programação, essas ferramentas já vêm sendo usadas há algum tempo, existem *Frameworks* para quase todas as linguagens de programação existentes.

Frameworks em sua essência consistem em conjuntos de funções ou classes implementadas em uma linguagem de programação específica para facilitar a programação, auxilia também na redução de código escrito já que contém várias funções prontas (GABARDO, 2017, p14).

1.7.1 Django

Segundo o *site oficial*, o *Django* é um *Framework* totalmente desenvolvido na linguagem *Python* com uma visão de desenvolvimento rápido para *Web*, utilizando o padrão de trabalho *model-Template-view*. O *Django* foi liberado ao público em 2005 com a licença de *software* livre que impactou em grande sucesso perante a comunidade *Python* mundial.

Segundo a documentação oficial o modelo de trabalho do *Django* é subdividido em três áreas, sendo elas *Model*, *Template* e Views, cada uma dessas áreas tem um objetivo especifico.

- ***Model*:** Nesta parte escrevem-se todas as classes relacionadas ao banco de dados, ou seja, a parte *model* em si funciona com manipulações de objetos ORM (Mapeamento de Objeto Relacional). A escrita de linhas SQL é mínima, e contem suporte aos principais bancos. O ORM do *Django* tem suporte a MySQL, PostgreSQL, SQLite e até mesmo Oracle.
- ***View:*** Na área de views, por mais que o nome remeta a algo a ser visto, não é nesta área que se adicionam esses artifícios. Nesta área é onde se escrevem as regras de apresentação, ou seja, cria-se funções que tem parâmetros de um objeto, e que aguarda uma resposta, ou seja, a view é o “meio termo” entre o *backend* e o *frontend* da aplicação.
- ***Template*:** Os códigos escritos na área do *Template* são os códigos da página *web* por assim dizer. Mas não apenas HTML ou CSS, podem-se

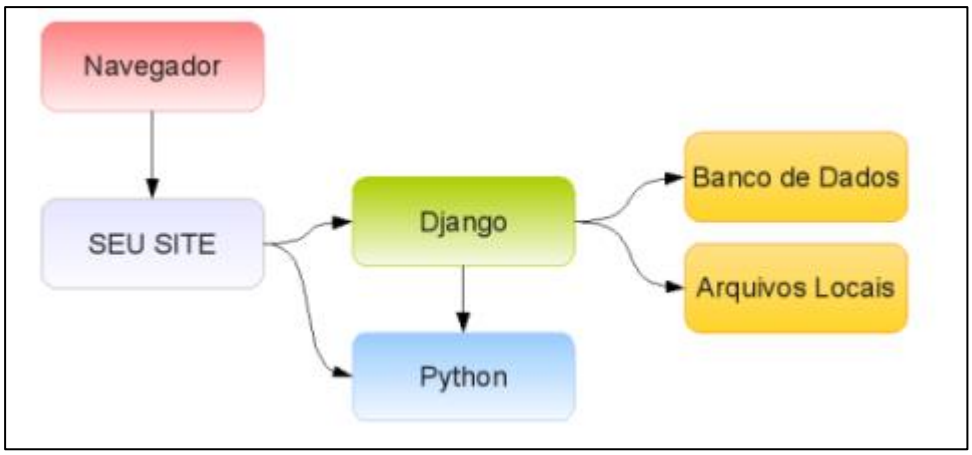
escrever vários outros tipos de linguagem nesta área como JavaScript, JSON, XML entre outros. O *Template*, em palavras mais descomplicadas, é a cara da aplicação, a parte que o usuário final tem acesso.

Ao iniciar aplicações *Django* é necessário a criação de arquivos padrão de configurações e de acesso. Segundo o *site oficial*, o *Django* cria esses arquivos. O usuário tem que apenas editá-los. Esses arquivos têm nomes específicos e configurações diferente sendo eles:

- **__init__.py:** Um arquivo vazio que diz ao *Python* que esse diretório deve ser considerado como um pacote *Python*.
- **manage.py:** Um utilitário de linha de comando que permite a você interagir com esse projeto *Django* de várias maneiras.
- **urls.py:** As declarações de URLs para este projeto *Django*; um "índice" de seu *site* movido a *Django*.

O *Django*, apesar de ser um *Framework*, trabalha não só na fase de criação do projeto, durante o uso do projeto ele está presente. A Figura 8 exemplifica melhor o modo de trabalho deste *Framework*.

Figura 8 – Modelo de Trabalho



Fonte: BRANDÃO, 2009, p 14

O autor ainda completa exemplificando de forma simplória que o navegador acessa seu *site* e o *Django* faz a ligação entre os comandos *Python*, banco de dados e arquivos locais, que neste caso são os códigos de *frontend*.

Após a criação do projeto e configurações necessárias para o funcionamento da aplicação, as requisições a esse projeto são feitas por meio de *request* ou (*HttpRequest*) que não é nada mais que pedido de informação à aplicação. Algumas informações são solicitadas em cada *request* e estão contidas na própria URL com exemplificado no Quadro 6.

Quadro 8 - *Request*

http://localhost:8000/admin/?nome=Mychell&idade=15

- Protocolo: **http**
- Domínio: **localhost**
- Porta: **8000**
- Caminho: **/admin/**
- Parâmetros: **nome = Mychell e idade = 15**

Fonte: BRANDÃO, 2009, p.40.

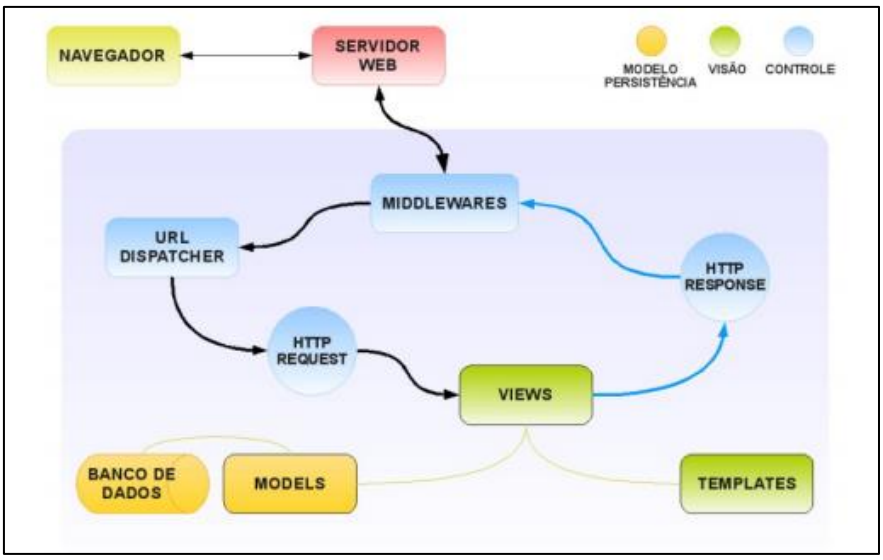
Outras informações que podem vir junto a esse *request* são: IP do usuário, localização, navegador utilizado, tudo depende da configuração realizada na aplicação (BRANDÃO, 2009, p.41);

Segundo Brandão (2009, p.41-44) após a requisição feita, são desencadeadas ações para atingir uma resposta e os executadores dessas ações são:

- ***Handle***: *Handle* é um manipulador do *HttpRequest* que direciona os caminhos por onde a requisição e a resposta passarão para que a comunicação seja fluida.
- ***Middlewares****:* São alguns trechos de códigos que analisam a resposta e a requisição, além de serem configuráveis para desempenhar diversas funções como: garantir a segurança do usuário, armazenar cache para acesso mais rápido, memorizar as ações do usuário para melhor atendê-lo na próxima requisição dentre outras funções.
- **URL *Dispather***: Após passar pelos *Middlewares* as requisições são analisadas *pleo dispather* que verifica o endereço e caminho das mesmas e se existir ele verifica o arquivo urls.py para determinar qual *view* será chamada para a resposta requisição.

A Figura 9 exemplifica as definições e o funcionamento dos componentes citados no parágrafo anterior

Figura 9 - Componentes



Fonte: BRANDÃO, 2009, p.43.

1.7.2. Materialize

Segundo o *Site oficial*[15] o Materialize é um projeto do Google que combina as principais técnicas de *site oficial* obtidas com as inovações tecnológicas. O principal objetivo deste projeto é desenvolver um sistema de *Site Oficial* que permita uma experiência de usuário unificada em todos seus produtos mesmo que em diferentes plataformas.

Ainda segundo as informações oficiais, sua equipe de desenvolvimento é composta por quatro estudantes da Universidade *Carnegie Mellon* localizada no EUA, são eles: *Alvin Wang, Alan Chang, Alex Mark e Kevin Louie*. A versão atual da ferramenta é a 1.0.0, que pode ser utilizada em duas formas diferentes:

- **Materialize:** Versão padrão que vem com arquivos CSS e JavaScript, necessitando de pouca configuração.

15 Disponível em: <https://materializecss.com/about.html>

- **Sass:** Esta é uma versão que contém arquivos SCSS de origem, disponibilizando maior controle sobre os componentes a serem utilizados, porém com isso requer uma configuração profunda e um compilador de Sass.

É possível localizar no *site oficial* um guia para obtenção[16], que pode ser feito através de CDN (*Content Delivery Network*), onde tudo é disponibilizado em seu projeto através da Rede, dispensando o *download* dos componentes, ou também por *download*, necessitando que os arquivos baixados sejam extraídos em um diretório do seu projeto, após isso basta vincular os arquivos corretamente em sua página *web*. Os arquivos ***min*** são comprimidos para reduzir o tempo de carregamento da página, outra recomendação importante para reduzir este tempo é incluir os arquivos JavaScript no final do corpo da página.

1.8 Programação de Computadores

Os computadores são instruídos por uma linguagem particular. Para que eles se comportem de acordo com o desejado, basta que sejam comandados a partir de uma linguagem que seja capaz de entender.

> Diferentemente do que ensina o senso o comum, os computadores não possuem inteligência. Seu único trabalho é processar dados, conforme uma sequência de instruções que fazem parte do vocabulário da linguagem que eles conseguem compreender. A ilusão de que eles realizam tarefas de forma inteligente é proporcionada através desse conjunto ordenado de instruções, que é denominado de algoritmo (SOUZA; JÚNIOR; FORMIGA, 2014, p. 2).

Os autores acima dizem que algoritmos representam passos a serem seguidos para que certas atividades sejam realizadas. Como por exemplo, algoritmos podem citar as instruções para montagem de equipamentos, para utilização de cosméticos, receitas culinárias, entre outras.

---

16 Disponível em: <https://materializecss.com/getting-started.html>

Algoritmo pode ser definido como uma sequência finita, ordenada e não ambígua de passos para solucionar determinado problema ou realizar uma tarefa. Portanto é possível compreender um algoritmo como um conjunto de passos executáveis que definem um processo finito (SOUZA; JÚNIOR; FORMIGA; 2014, p.2).

1.8.1 Linguagens de Programação

Uma aplicação/programa pode ser compreendida como um conjunto de instruções ou comandos que passam instruções para o computador para realizar certas operações. Estas aplicações, para que cumpram seu papel, necessitam que suas instruções sejam processadas e o responsável por isto é o processador. Porém o processador compreende somente linguagem de máquina, e esta linguagem varia de acordo com a arquitetura do processador. Um programa escrito em linguagem de máquina consiste em uma série de bits binários (DOBAY, 2012, p.4).

Segundo Dobay (2012), as Linguagens de Programação foram desenvolvidas pois não seria viável utilizar a Linguagem de Máquina para desenvolver programas, por conta do alto nível de complexidade. Com isso as Linguagens de Programação servem como intermediário entre a linguagem humana e a linguagem de máquina.

> Uma linguagem de programação deve ser, por um lado, legível e compreensível para um humano, e, por outro lado, suficientemente simples para que possa ser compreendida, completamente e sem ambiguidades por um computador (DOBAY, 2012, p.4-5).

Segundo Dobay (2012, p.5), as Linguagens de Programação são compostas por: palavras-chave (normalmente em inglês), símbolos e um conjunto de regras que podem ser combinadas e utilizadas.

As Linguagens de Programação podem não ser entendidas diretamente pelo computador, para isso existem programas específicos, feitos para entender e traduzir essas linguagens para linguagem de máquina. Estes programas são divididos em duas classificações, compiladores (que apenas traduz o programa para linguagem de máquina), e interpretadores (que lê o que está escrito em certa linguagem de programação e imediatamente executa as instruções) (DOBAY, 2012, p.5).

1.8.1.1 *Paradigmas de Programação*

A medida que as várias linguagens de programação começaram a existir criaram seus próprios padrões de programação, escrita e sintaxe para solucionar os problemas propostos a seu uso. Logo não poderiam funcionar igual ou terem a mesma filosofia de funcionamento, tendo em vista que essas informações criaram paradigmas, ou seja, conjuntos e modelos padrões que separam as linguagens para suas funções e denominações (JULIO, 2011, p.4-13).

O fato de linguagens terem paradigmas diferentes implica diretamente no modo como algumas linguagens enxergam suas variáveis e seus tipos. Tem-se como exemplo o paradigma orientado a objetos, onde é possível criar classes e objetos, coisas que não são possíveis em paradigmas como o estrutural, e isso implica diretamente no modo como a variável vai agir ou ser usada dentro de uma função (IDEM).

Os paradigmas de programação são um meio de classificar as linguagens de acordo com suas funcionalidades. Após anos de pesquisa, vários paradigmas foram surgindo ao redor das novas linguagens de programação, que foram aparecendo de acordo com a necessidade computacional. Os principais paradigmas são os paradigmas estruturados e os orientados a objetos. Apesar de um paradigma ditar diretamente o funcionamento de uma linguagem, não necessariamente essa linguagem se prende a apenas um paradigma. As principais linguagens que são usadas como exemplos são a C e C++ que, apesar de uma se derivar da outra, trabalham em paradigmas diferentes. Além do paradigma estrutural, a linguagem C também é dotada de outro paradigma, o procedural, enquanto o C++ é classificado como orientado a objetos e pode ser usado também de forma estrutural (DAVID, 2016, p10-11).

1.8.1.2 *Linguagem de Programação C++*

De acordo com Baltarejo e Santos (2006, p.1), a Linguagem C teve seu desenvolvimento realizado entre 1969 e 1973, foi atribuído a ela este nome por conta

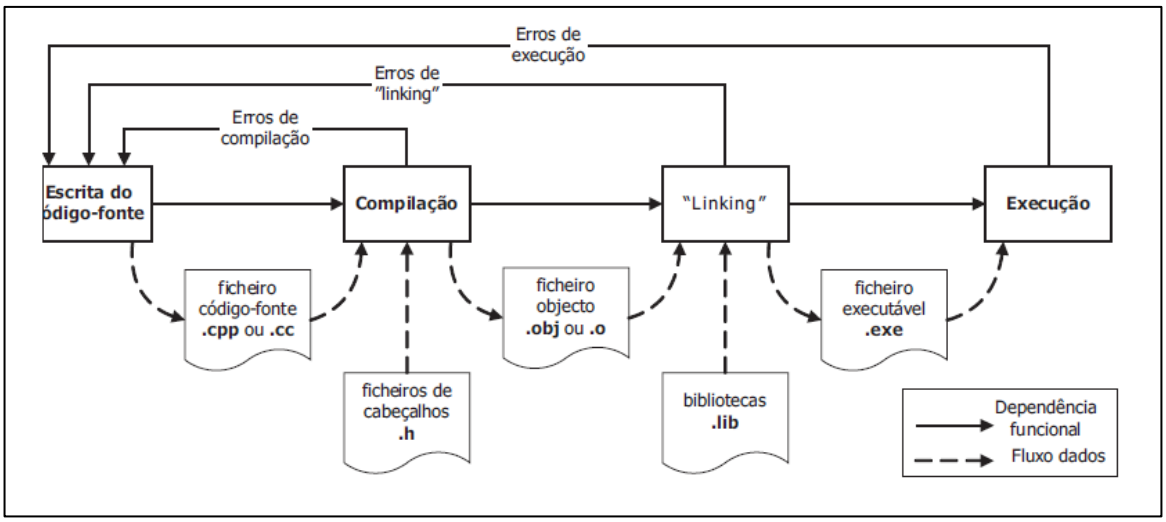
da sua derivação de uma já existente, a linguagem “B”. Um acontecimento que agregou bastantes recursos para a linguagem foi o suporte à grande parte do *Kernel Unix* que havia sido anteriormente escrito em *assembly*. Essa descrita linguagem C serviu como base para a C++ que realmente interessa a este projeto. Inicialmente destinada apenas para o ambiente UNIX, seus criadores foram Dennis Ritchie e Kenneth Thompson. A linguagem criada por eles atualmente é considerada bastante simples. Por sua simplicidade ficou bastante conhecida nos anos 80, chegando a possuir diversos compiladores e muitas arquiteturas, porém desde seu início até hoje existem muitos problemas de compatibilidade.

Derivada da linguagem C, a linguagem C++ chegou ao mercado trazendo a velha C como base com bastante melhorias, porém, utilizando o paradigma de programação orientado a objeto. A mesma surgiu através dos aprimoramentos feitos pelo estudioso Bjarne Stroustrup na antiga linguagem chamada C (VOTRE, 2016, p.3, 19).

Segundo Mendes (2010, p. 12), a linguagem C++ possui elementos importantes que possibilitam a compreensão de classes, objetos e consequentemente pode ser classificada como Programação Orientada a Objetos (POO), possui um padrão de entrada e saída de dados melhorado, comparando com o C. É nítido que as alterações feitas na antiga linguagem agregaram flexibilidade e facilidade para os programadores. Atualmente encontra-se na versão (14882:2017), lançada em 15 de dezembro de 2017.

O desenvolvimento de programas em linguagem C++ é um processo que possui quatro fases: escrita, compilação, “*linking*” e execução (BALTAREJO, SANTOS, 2006, p.3).

Quadro 9 – Fases do Processo de Desenvolvimento em C++



Fonte: BALTAREJO; SANTOS, 2006, p.3.

Escrita: é a fase inicial do processo de desenvolvimento, onde ocorre a criação/edição do(s) ficheiro(s) de texto contendo código-fonte; normalmente é realizado com a utilização de um editor de texto ou algum ambiente de desenvolvimento que possibilite tal recurso. O arquivo/ficheiro criado nesta fase tem que ter a extenso **.cpp** ou **.cc**.

Compilação: A segunda fase é realizada com auxílio de um compilador específico; nesta fase os erros de sintaxe serão detectados e reportados com identificação da linha onde o erro se encontra e uma breve descrição do erro. Caso não exista nenhum erro de sintaxe, o compilador produzirá um programa pronto a ser executado.

"*Linking*": Nesta fase o *linker* será responsável por substituir os cabeçalhos por um código executável; caso ocorra erro nesta fase, o processo retorna à primeira fase.

Execução: A última fase só ocorrerá caso todas as anteriores terminem com sucesso.

Segundo Baltarejo e Santos (2006, p.6), a utilização de variáveis num programa C++ requer a sua declaração prévia respeitando o padrão: tipo e o nome da variável. No Quadro 10 abaixo são descritos os tipos aceitos na linguagem.

Quadro 10 – Tipos de Variáveis C++

Nome	Tam.	Descrição	Gama de valores
`char`	1	Caracter ou inteiro de 8 bits de comprimento	c/sinal: -128 a 127 e s/sinal: 0 a 255
`short`	2	Inteiro de 16 bits de comprimento	c/sinal: -32768 a 32767 e s/sinal: 0 a 65635
`long`	4	Inteiro de 32 bits de comprimento	c/sinal: -2147483648 a 2147483647 e s/sinal: 0 a 42949967295
`int`	*		ver `short` e `long`
`float`	4	Número real	3.4e±38
`double`	8	Número real, virgula flutuante, dupla precisão	1.7e±308
`long double`	10	Número real longo, virgula flutuante, dupla precisão	1.2e±4932
`bool`	1	Valores lógicos - booleanos	`true` e `false`
`wchar_t`	2	Caracter extendido, necessário para a representação de caracteres internacionais	Caracteres (incluindo internacionais)

Fonte: BALTAREJO, SANTOS, 2006, p.7.

Através da palavra-chave **const** é possível declarar constantes de um determinado tipo, conforme demonstrado a seguir:

Quadro 11 – Declaração de Constantes C++

```
const int width = 100;
const char tab = \t;
const minhaConstante=12440;
```

Fonte BALTAREJO; SANTOS, 2006, p.10.

De acordo com Baltarejo e Santos (2006, p.12), a linguagem C++ possui além do **=** que realiza a atribuição direta para a esquerda (e nunca o inverso), os seguintes operadores lógicos:

Quadro 12 – Declaração de Variáveis C++

a=10;
b=4;
c=a;
	O valor da variável c será 10

Fonte: BALTAREJO, SANTOS, 2006, p.12.

Quadro 13 – Operadores Aritméticos C++

Operador	Nome	Exemplo	Resultado
+	soma	a+b	18
-	subtracção	a-b	8
*	multiplicação	a*b	65
/	divisão	a/b	2.6
%	resto da divisão inteira	a%b	5

Fonte: (BALTAREJO; SANTOS, 2006, p.13.

Existem também os operadores compostos permitindo alterar o valor de uma variável com um operador, como no Quadro 14:

Quadro 14 – Operadores Aritméticos Compostos C++

Operador	Nome	Exemplo	Significado
+=	soma/atribuição	a+=b	a=a+b
-=	subtração/atribuição	a-=b	a=a-b
*=	multiplicação/atribuição	a*=b	a=a*b
/=	divisão/atribuição	a/=b	a=a/b
%=	resto divisão inteira/atribuição	a%=b	a=a%b
++	incremento	a++	a=a+1
−−	decremento	b−−	b=b-1

Fonte: BALTAREJO; SANTOS, 2006, p.13.

Com os operadores relacionais é possível avaliar a comparação entre duas expressões, tento como resultado um valor do tipo ***bool***, sendo ele *true* ou *false*:

Quadro 15 – Operadores Relacionais C++

Símbolo	Significado
<	menor que
>	maior que
≤	menor ou igual que
≥	maior ou igual que
==	igual
! =	diferente

Exemplo	Resultado
(7==5)	falso
(a!=b)	verdade
(a<=7)	verdade
((a*b)>=c)	verdade

Fonte: BALTAREJO. SANTOS, 2006, p.14.

O resultado obtido com operações lógicas também é verdadeiro ou falso, mostrado na tabela com os operadores lógicos:

Quadro 16 – Operadores Lógicos C++

Símbolo	Significado
&&	conjunção
||	disjunção
!	negação

Exemplo	Resultado
((5==5) & & (3>6))	falso
((5==5) || (3>6))	verdade
!(5==5)	falso
!verdade	falso

Fonte: BALTAREJO; SANTOS, 2006, p.15.

Outro recurso da linguagem C++ são as instruções de decisão, que possibilitam a alternância entre um ou outro conjunto de ações após avaliação lógica de uma condição. Para uso estão disponíveis o ***if***, ***if-else*** e ***switch***. Já no caso de estruturas de repetição o C++ tem: ***while***, ***do-while*** e ***for*** melhor explicitado adiante.

O C++ também permite a criação de funções, de forma a estruturar o seu código de maneira modular, podendo ou não requerer argumentos, e o bloco de código presente no interior da função será executado sempre que a mesma for chamada. A função principal dos códigos em C++ é chamada “*main*” onde a execução é iniciada (BALTAREJO, SANTOS, 2006, p.15-36,37).

Quadro 17 – Sintaxe Funções C++

```
<tipo-de-dados>  <id-da-função>( <argumento 1>,<argumento 2>,...)
 {

    <bloco-de-instruções>
 }
```

Fonte: BALTAREJO. SANTOS, 2006, p.36.

A utilização da orientação a objetos pode beneficiar o desenvolvimento agregando robustez, extensibilidade, reuso e compatibilidade. Com isso é possível obter programas modulares, facilitando o entendimento e a produção, a continuidade e até mesmo a proteção no decorrer do processo de desenvolvimento (BORGES, CLINIO, 2006, p.8-9)

Segundo Borges e Clinio (2006, p.12-13), as classes em C++ são uma extensão de uma estrutura que além de dados também possui funções. No Quadro 18 a seguir é possível identificar uma estrutura (em C++: ***struct***) chamada ***Stack*** e uma função chamada ***empty*** que pertence à estrutura criada:

Quadro 18 – Declaração de Classes C++

```
struct Stack {
    // ...
    int empty() { return top == 0; }
};
```

```
struct Stack {
    // ...
    int empty();
};

int Stack::empty(void) { return top == 0; }
```

Fonte: BORGES; CLINIO, 2006, p.13.

Resumindo para o C++, classes são estruturas, objetos são variáveis do tipo de alguma classe (instância de alguma classe), métodos são funções de classes e todo objeto deve ser classificado de acordo com seu comportamento e não apenas como estrutura de representação.

A linguagem C++ é muito importante para este projeto, pois um dos dispositivos (*Black Board*) utilizados para o seu desenvolvimento, possui a mesma como linguagem padrão de funcionamento, sendo permitido o uso da linguagem C ou C++, porém por questão de escolha será utilizada C++. Acima fez-se uma breve descrição da linguagem, com as principais características para que seja possibilitado o entendimento de alguns dos processos no decorrer do projeto.

1.8.1.3 *Linguagem de Programação Python*

A linguagem *Python* foi concebida em meados dos anos 80 por *Guido Van Rossum*, neste período ele trabalhava na CWI (*Centrum Wiskunde & Informatica*, Centro de Matemática e Ciência da Computação) em Amsterdã, Holanda, no time de desenvolvimento da Linguagem ABC. Com o fim da linguagem ABC em 1987, Guido foi transferido para um projeto chamado AMOEBA - um sistema operacional, liderado por *Andrew Tanenbaum*. Logo Guido viu a necessidade de criar uma linguagem que trabalhasse entre a linguagem C (que na época era muito utilizada) e o *Shell Script*. Como Guido vinha de uma longa jornada trabalhando e desenvolvendo a Linguagem ABC, a *Python* herdou várias características da mesma, como indentação obrigatória que permeia até as versões mais atuais (SEVERANCE, 2013, p.2)

Em 1990 a primeira versão operacional do *Python* já estava em uso na própria CWI e para o espanto de todos já se mostrava mais usada que a própria linguagem ABC. O nome *Python* surgiu de uma tradição da CWI de colocar nomes de programas de televisão em seus projetos, *Rossum* era fã de um programa da época chamado *Monty Python's Flying Circus* então surgiu o nome *Python* vinculado a esse programa (SEVERANCE, 2013, p.5-6).

No ano de 1991 a primeira versão ao público foi lançada, denominada v.0.9.0, continha 21 partes compactadas em um arquivo **.tar**. Essa primeira versão de *Python* já continha classes, tratamento de exceções, herança, funções, sistemas de módulos e tipos de dados próprios como **list**, **dict** e **str** dentre outros. Como o projeto era subsidiado pela CWI, todas as versões lançadas dentro da empresa não continham licença GPL (Licença Pública Geral) mas sim a licença MIT que transferia todos os direitos a uma única empresa ou grupo. Dentro da CWI o Guido e sua equipe

permaneceram até 1995 onde aprimoraram a linguagem até sua versão 1.2 (ELKNER, 2015, p.5).

Segundo Downey, Elkner e Meyers (2017, p.4,5) a linguagem *Python* tem como objetivo simplificar a programação a fim de deixá-la mais fluida e simples de compreender. Por ser uma linguagem de alto nível era de se esperar que ela tivesse uma certa complexidade se comparada com suas "concorrentes" no mesmo segmento, porém sua proposta de facilitar é bem visível em algumas partes do código quando se começa a programação. Em Java por exemplo a brincadeira de imprimir "Alô mundo" em um de seus modos se escreve conforme o Quadro 19:

Quadro 19 – Alô Mundo em Java

```
class Hello {
    public static void main(String[] args) {
        System.ou.println("Alô, mundo.");
    }
}
```

Fonte: DOWNEY; ELKNER; MEYERS, 2017,p.4.

Na versão *Python*, segundo o Quadro 20 ele se transforma em:

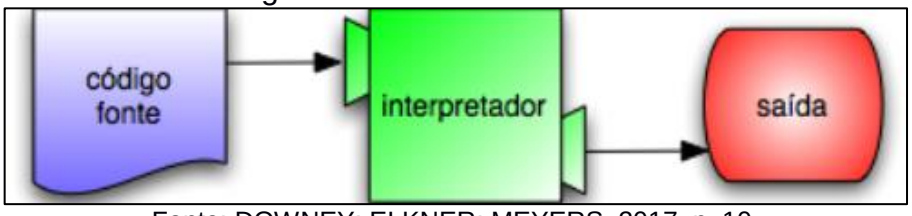
Quadro 20 – Alô, Mundo em *Python*

```
print "Alô, Mundo!"
```

Fonte: DOWNEY; ELKNER; MEYERS, 2017, p.5.

As linguagens de alto nível se comportam de forma diferente das de baixo nível não só pela sua escrita e métodos, mas também pela sua execução e como o computador interpreta seu resultado final, existem dois métodos (podendo ser visualizado nas Figuras 10 e 11), de se executar linguagens de alto nível. A Figura 10 mostra um processo mais simples, porém converte linguagem alto nível, em baixo nível usa de artifício um interpretador. Esse interpretador lê o código um pouco de cada vez e o executa uma linha de cada vez (DOWNEY; ELKNER; MEYERS, 2017, p. 10)

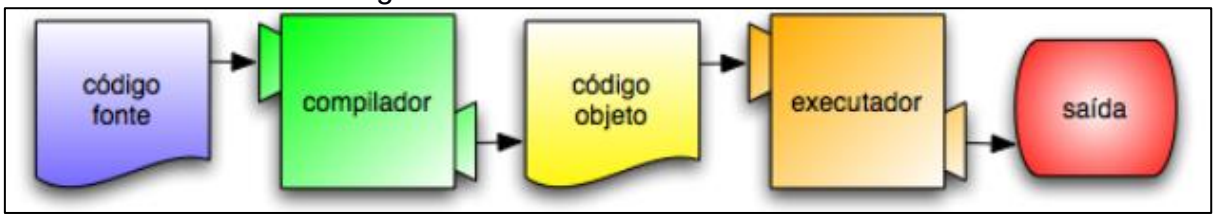
Figura 10 – Trabalho Alto Nível



Fonte: DOWNEY; ELKNER; MEYERS, 2017, p. 10.

Já o segundo método de execução dessa linguagem consiste em um compilador que lê o código fonte completamente antes de executá-lo, gerando um executável ou também conhecido como código objeto; uma vez compilado o usuário pode executar o programa de forma repetida sem a necessidade de compilá-lo novamente, como mostra a Figura 11.

Figura 11 – Trabalho de Alto Nível 2



Fonte: DOWNEY; ELKNER; MEYERS, 2017, p. 10.

Código Fonte: É a fase inicial do processo de desenvolvimento, onde ocorre a criação/edição de texto contendo código-fonte, geralmente feito por um programador utilizando ferramentas para facilitar a escrita do código (BALTAREJO, SANTOS, 2006, p.3).

Compilação: Essa fase, é realizada com auxílio de um compilador específico, nesta fase os erros de sintaxe serão detectados e reportados com identificação da linha onde o erro se encontra e uma breve descrição do erro. Caso não exista nenhum erro de sintaxe o compilador produzirá um programa pronto a ser executado (IBDEM)

Código objeto: Esse codinome diz respeito à aplicação pronta por assim dizer, ou seja, o executável do programa que pode ser distribuído a vários usuários.

Interpretador: O interpretador, ao contrário do compilador, roda o código escrito como sendo o executável, ele traduz o programa linha a linha, o programa vai sendo utilizado na medida em que vai sendo traduzido. Cada execução do programa precisa ser novamente traduzido e interpretado (DOWNEY; ELKNER; MEYERS; 2017, p. 11).

O *Python* é considerado uma linguagem interpretada pois a mesma necessita de um interpretador instalado para que possa ler os comandos escritos em código. Essa interpretação pode ocorrer de vários modos diferentes, mas os mais usados são por linhas de comando e no modo de *script,* no modo linha de comando se digita o comando *Python* e o interpretador mostra o resultado. Esse comando aceita como parâmetros nome de arquivos em extensão *Python*[.py], ou seja, o usuário pode

executar códigos diretamente no interpretador ou importar códigos feito em editores de texto como exemplo (DOWNEY; ELKNER; MEYERS; 2017, p. 11).

A primeira linha do quadro 23 demonstra a iniciação do interpretador *Python*; as três linhas que o seguem demonstram informações sobre o próprio interpretador e sistema operacional que está sendo usado no momento, ou seja, essas informações podem variar; os comandos da última e penúltima linha demonstram uma conta básica sendo executada, o sinal >>>, é usado para indicar que o interpretador está pronto para ser usado. Um script poderia ser aderido à primeira linha deste trecho de código, ou seja, um arquivo com extensão *Python* feito externamente ao interpretador e executado via linha de comando, como se refere o Quadro 21:

Quadro 21 – Linha de Comando

```
$ python
Python 2.7.3 (r252:60911, Apr 10 2012, 23:31:26)
[GCC 4.5.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 1 + 1
2
```

Fonte: DOWNEY; ELKNER; MEYERS; 2017, p. 11.

Quadro 22 – Executando Script py

```
$ python primeiro_programa.py
2
```

Fonte: DOWNEY; ELKNER; MEYERS; 2017, p. 11

Como linguagem de programação orientada a objetos o *Python* se destaca por sua simplicidade em sintaxe, definir variáveis, incluir métodos e bibliotecas, preza por simplicidade e organização o que é uma característica marcante dessa linguagem. Apesar de ser uma linguagem fracamente tipada as definições de variáveis contam com algumas regras de definições claras para melhor entendimento do interpretador como por exemplo, no Quadro 23, o não uso de número e caracteres especiais em seu nome:

Quadro 23 – Variáveis *Python*

```
>>> 76trombones = "grande parada"
SyntaxError: invalid syntax
>>> muito$ = 1000000
SyntaxError: invalid syntax
>>> class = "Ciencias da Computacao 101"
SyntaxError: invalid syntax
```

Fonte: DOWNEY; ELKNER; MEYERS; 2017, p. 18.

Essas definições de variáveis estão incorretas segundo o escopo padrão *Python*, pois a primeira começa com números e a segunda contém caractere especial que se enquadra em um conjunto de palavras reservadas existentes em *Python* juntamente com a terceira, ou seja, um conjunto de palavras ou símbolos que já contém uma predefinição própria em *Python*, todas com funções definidas. Sendo algumas delas representadas no Quadro 24:

Quadro 24 – Escopo Padrão

```
and       def       exec      if        not       return
assert    del       finally   import    or        try
break     elif      for       in        pass      while
class     else      from      is        print     yield
continue  except    global    lambda    raise
```

Fonte: DOWNEY; ELKNER; MEYERS; 2017, p. 21.

Em *Python* se utiliza bastante da matemática, já que um dos focos dessa linguagem é o auxílio na resolução de problemas relacionados a áreas mais exatas da ciência, os símbolos principais como +, - ou / e a utilização de parênteses para priorizar cálculos continua normalmente de acordo com a matemática clássica, porém as expressões mais complexas como potenciação e multiplicação utilizam de caractere específico: o * (asterisco); no caso da multiplicação uso de um, na potenciação uso de dois asteriscos (DOWNEY; ELKNER; MEYERS; 2017, p. 21-22).

O *Python* foi utilizado neste projeto para permitir ao usuário comandar o microcontrolador. Os scripts *py* serão responsáveis por trocar dados com a *Black Board*, dados estes que serão exibidos em uma interface de forma interativa para o usuário final. (DOWNEY; ELKNER; MEYERS; 2017, p. 18).

2 MATERIAS E MÉTODOS

2.1 Preparação do Ambiente de Testes

O *hardware* que foi utilizado para desenvolvimento da aplicação foram dois notebooks Acer, sendo a configuração do primeiro: processador com base em x64 Intel Core I5- 7200U CPU 2.50GHz com *bust* 3.50GHz, Memória RAM 8GB DDR4, Placa de Video NVIDIA 940MX com 2 GB GDDR5 com sistema operacional Linux Unbuntu 18,04 LTS de 64bits, com 250GB de SSD e 1TB de HD, e a configuração do segundo: processador com base em x64 Intel Core I7- 6200U CPU 2.4GHz com *bust* 3.3GHz, Memória RAM 8GB DDR3, Placa de Video NVIDIA 710M com 2 GB GDDR5 com sistema operacional Linux Xunbuntu 18,04 LTS de 64bits, com 1TB de HD.

2.2 Preparação do Ambiente de Produção

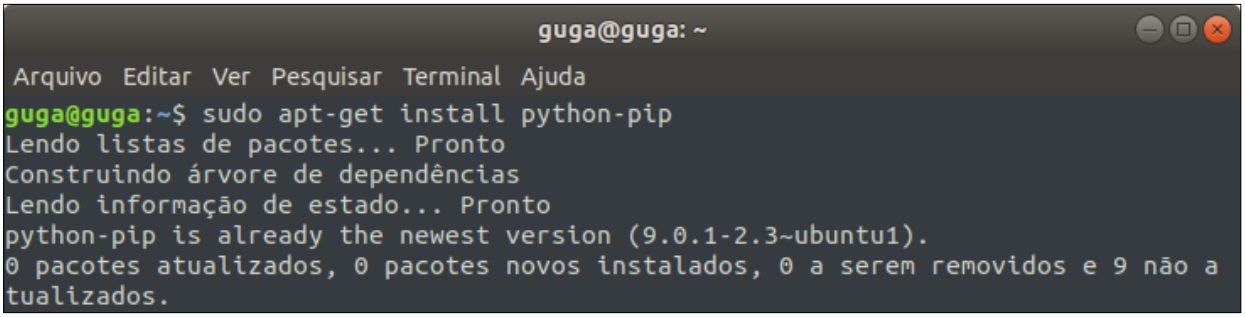
O *hardware* que foi utilizado para o ambiente de produção do projeto foi um *Raspberry Pi* 3 Modelo B + sendo a configuração: processador Broadcom BCM2837B0, Cortex-A53 (ARMv8) SoC de 64 bits a 1,4 GHz, Memória 1 GB de SDRAM LPDDR2 com sistema operacional Linux *Raspbian* de 64bits, com 16GB em um cartão SD, e uma *Black Board* modelo UNO R3 da fabricante nacional Robocore, está placa conta com pinagem digital e analógica sendo 12 pinos digitais e 5 pinos analógicos.

2.3 Ferramentas e Aplicações

2.3.1 *Python* 2.7.15

O *Python* foi utilizado neste projeto com o intuito de fazer a comunicação via USB do Raspberry para a *Black Board* e o *backend* da aplicação, em sistemas UNIX geralmente o *Python* 2.7 já vem instalado como padrão, porém, para instalação das bibliotecas e extensões é necessário a instalação do gerenciador de pacotes do *Python* o PIP, para e efetuar a instalação do mesmo é preciso apenas um comando como demonstra a Figura 12:

Figura 12 - Instalação gerenciador de pacotes pip



Fonte: dos próprios autores

2.3.2 Djagon 1.7.0

O *Framework Django*, para a linguagem *Python*, pode ser instalado pelo gerenciador de pacotes PIP, para tanto utiliza-se os comandos apresentados no Quadro 25. Onde o primeiro instalará o *Django* em sua última versão e o seguinte na versão 1.7.0, que foi utilizada neste projeto.

Quadro 25 – Instalação *Django*

```
sudo pip install Django
ou
sudo pip install Django==1.7.0
```

Fonte: dos próprios autores

2.3.3 *Pyserial*

Pyserial é uma biblioteca livre da linguagem *Python* que facilita a comunicação da linguagem com as portas serias do computador, por meio dela pode se comunicar mais facilmente e executar comando mais rapidamente nas serias[17]. Sua instalação é simples e feita através do gerenciador de pacotes do *Python* o PIP.

Quadro 26– Instalação *Pyserial*

```
sudo pip install pyserial
```

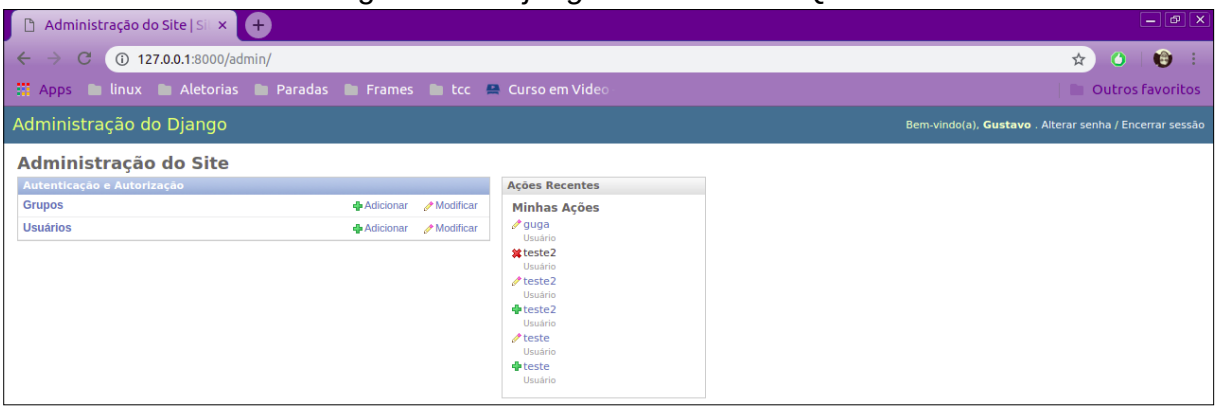
Fonte: dos próprios autores

2.3.4 SQLite

Essa biblioteca em linguagem C que implementa um banco de dados SQL[18] já vem com do *Django*, ou seja, não há necessidade de instalá-la no computador pois o *Django* já á utiliza como padrão, ao se iniciar uma aplicação *Django* automaticamente é importado algumas tabelas para a área administrativa onde se pode criar usuários com permissões de acesso diferentes sem necessariamente utilizar um processo de SGBD separado, esta base de dados é mantida localmente em arquivo Veja na Figura 13 o painel *Django* admin utilizando SQLite:

17 Disponível em: <https://pythonhosted.org/pyserial/>
18 Disponível em: <https://www.sqlite.org/amalgamation.html>

Figura 13 – *Django* Admin com SQLite

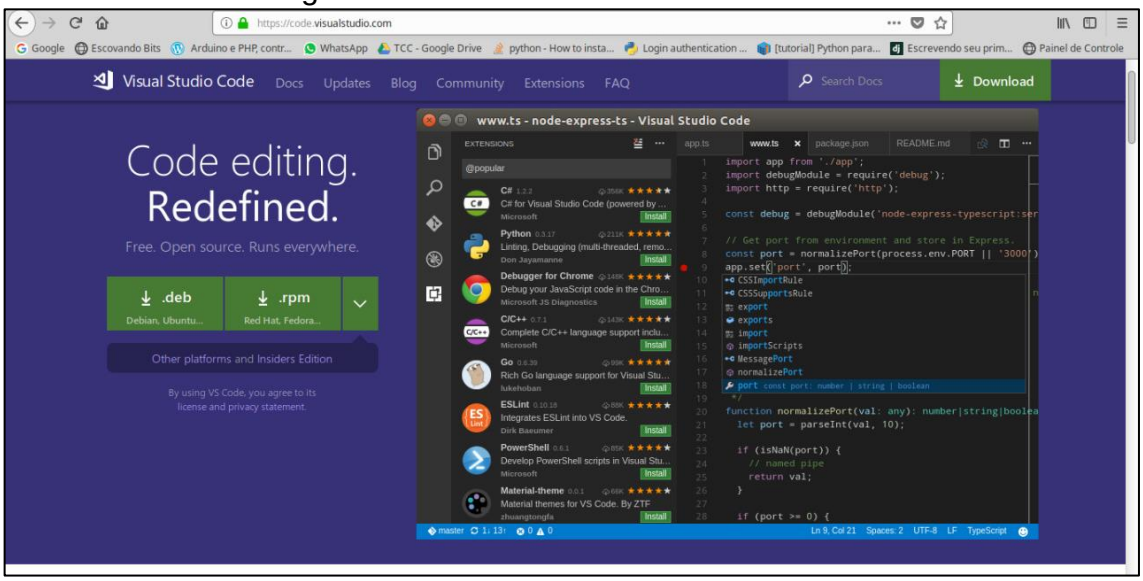


Fonte: dos próprios autores

2.3.5 Visual Studio Code

O Visual Studio Code é um editor de código, com suporte para operações de desenvolvimento, como depuração, execução de tarefas e controle de versão[19]. É necessário realizar o *download*, disponível em: <https://code.visualstudio.com/>, para as plataformas Linux, Windows e MacOS. Segue a Figura 14 para melhor visualização.

Figura 14 – *Download* Visual Studio Code



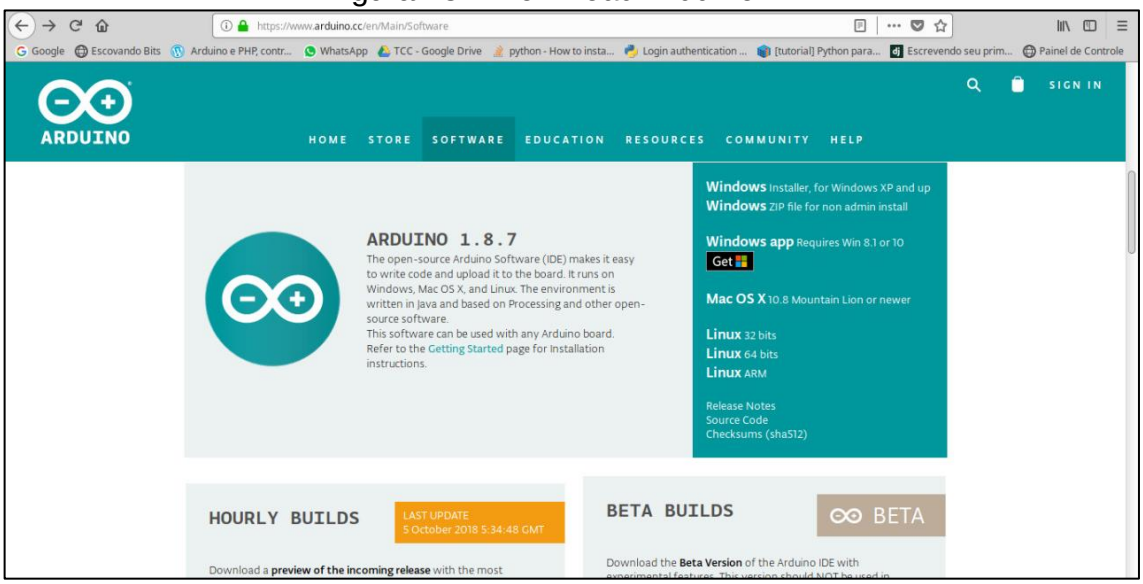
Fonte: dos próprios autores

19 Disponível em: <https://code.visualstudio.com/docs/supporting/faq>

2.3.6 *Arduino* IDE

O *Arduino* IDE é uma aplicação de código livre que facilita a gravação e upload de códigos para a placa (neste caso a *Black Board*)[20]. O *download* está disponível no *link*: < https://www.*Arduino*.cc/en/Main/*Software*>, para os sistemas operacionais: Windows 8.1 e 10, Linux e MacOS, como pode-se visualizar na Figura 15 a seguir:

Figura 15 – *Download Arduino* IDE



Fonte: dos próprios autores

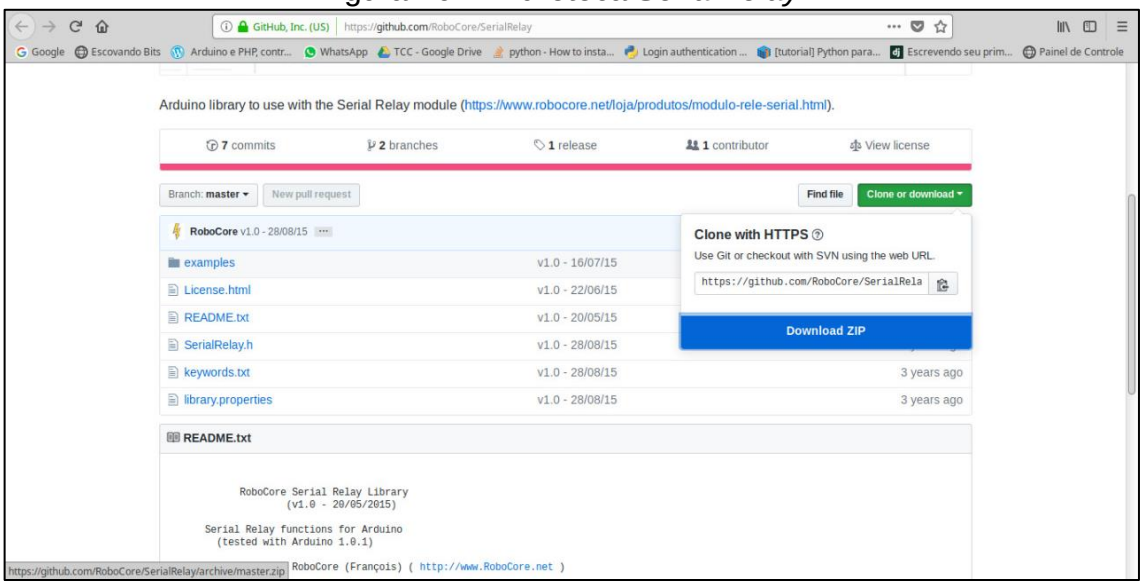
A versão utilizada neste projeto foi a 1.8.7, com o intuito de realizar o upload do código C++ para a *Black Board*. Após a instalação é necessário configurar a porta de comunicação com a placa, assim como o modelo da placa utilizada. Para isso, basta iniciar a aplicação, localizar o menu ferramentas na parte superior, nele estarão disponíveis as opções **Placa**, onde será selecionado o modelo; e **Porta**, onde deve-se escolher a porta utilizada pelo microcontrolador. As configurações utilizadas neste caso foram **Placa: "***Arduino*/Genuíno Uno"; **Porta:** "/dev/ttyUSB0"

2.3. 7 *SerialRelay*.h

[20] Disponível em: <https://www.arduino.cc/en/Main/Software>

Como dito no item *1.2.1.1* a *SerialRelay* é uma biblioteca livre, e será necessária para utilização do módulo relé serial. Para utilizar a biblioteca é necessário realizar o *download*, a mesma é disponibilizada no *link*: <https://github.com/RoboCore/*SerialRelay*>, como na Figura 16 a seguir:

Figura 16 – Biblioteca *SerialRelay*.h



Fonte: dos próprios autores

Após o *download* foi necessário a importação para que a o *Arduino* IDE consiga reconhecer os comandos da biblioteca e empurrá-los na *Black Board*. Para realizar esse procedimento basta iniciar a aplicação, localizar o menu Sketch na parte superior; logo após Incluir Biblioteca; e adicionar biblioteca .ZIP.

2.3.8. Materialize

Como exposto no item 1.7.2, o materialize facilitou a estilização da aplicação *web* proposta neste projeto, para isso é necessário realizar o *download*, disponível no *link*: <https://materializecss.com/getting-started.html>. No mesmo estão disponíveis as diferentes formas de utilização e vinculação dos recursos em sua página *web*. A versão utilizada neste projeto foi a 1.0.0 pelo método vinculação como pode ser visto na Figura 17 a seguir:

Figura 17 – Importação Materialize

```
<!DOCTYPE html> {% load static %}
<html lang="pt-br">

<head>
    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">



    <link type="text/css" rel="stylesheet" href="{%static 'css/materialize.min.css'%}" />
    <script type="text/javascript" src="{%static 'js/materialize.min.js'%}"></script>
```

Fonte: dos próprios autores

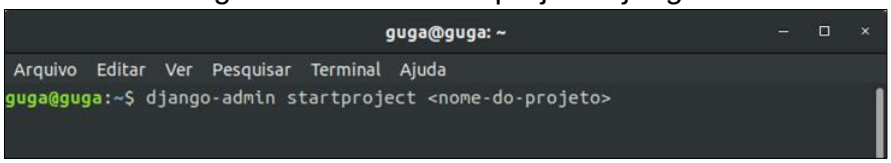
2.4 Metodologia

O desenvolvimento do sistema de automação foi realizado de forma incremental, no que diz respeito ao *software, Hardware* e fiação. Os testes foram realizados pelos desenvolvedores no ambiente de testes de acordo com o desenvolvimento de cada parte do sistema; já no ambiente real; os testes foram realizados pelo usuário com base no decorrer da implantação.

2.4.1 Desenvolvimento da aplicação

A aplicação para comunicação com a placa microcontroladora foi realizada através da linguagem *Python* por meio do *Django*, para iniciar um novo projeto executou-se o comando demonstrado abaixo na Figura 18.

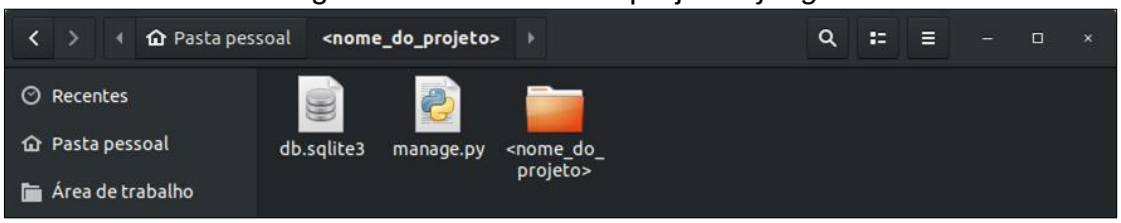
Figura 18 – Iniciando projeto *Django*



Fonte: dos próprios autores

Após efetuar a ação descrita acima o próprio *Framework* criou uma pasta com o nome escolhido e uma série de outros arquivos necessários para começar o desenvolvimento, arquivos estes editáveis e configuráveis, ou seja, uma base para o início do projeto como demonstra a Figura 19.

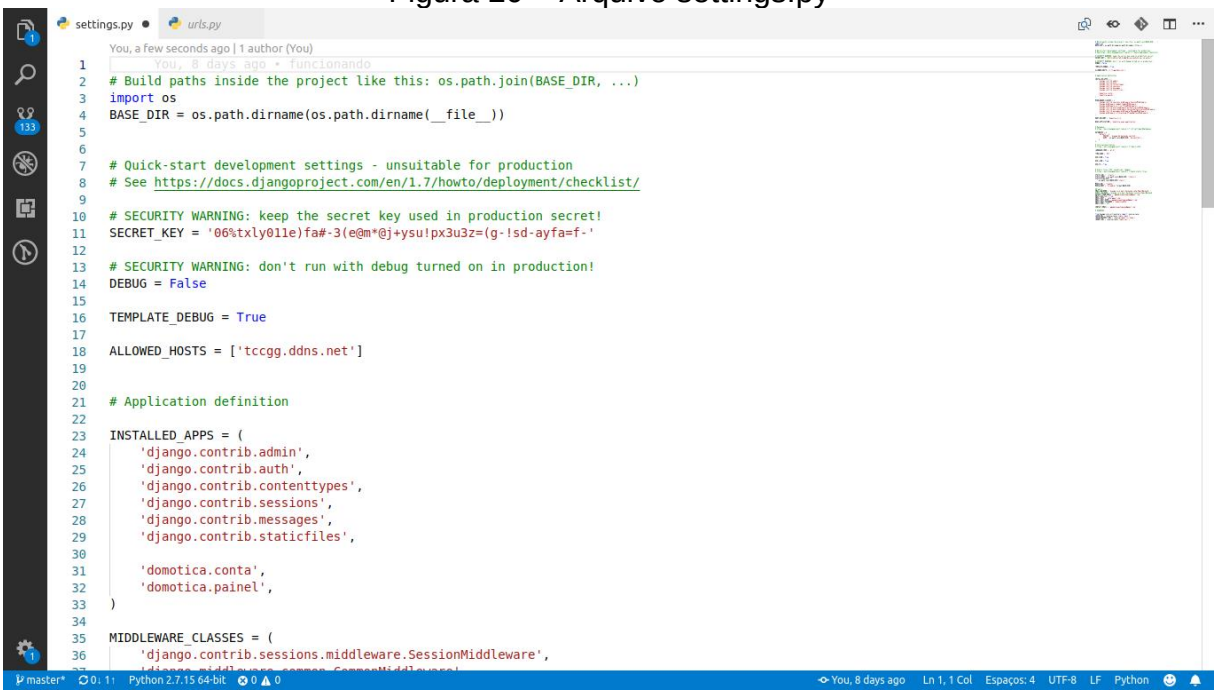
Figura 19 – Diretório do projeto *Django*



Fonte: dos próprios autores

Após a criação do projeto foi necessário a escolha de um editor para facilitar a modificação dos arquivos e criação de novos, como citado anteriormente, o *Visual Studio Code* foi usado com essa função. Os arquivos criados contem variais finalidades, mais como todo projeto em geral contém alguns de suma importância. A foto a seguir mostra o arquivo de configuração geral settings.py onde são importadas aplicações, APIS e todo tipo de configuração global do projeto. Como exposto na Figura 20:

Figura 20 – Arquivo settings.py



Fonte: dos próprios autores

Seguindo com arquivos importantes ao projeto tem-se o arquivo urls.py onde estão todos os “caminhos” do projeto, ou seja, onde existem as URLs de direcionamento para cara função do projeto. A Figura 21 abaixo demonstra com maior clareza essa sistemas e caminhos.

Figura 21 – Arquivo urls.py

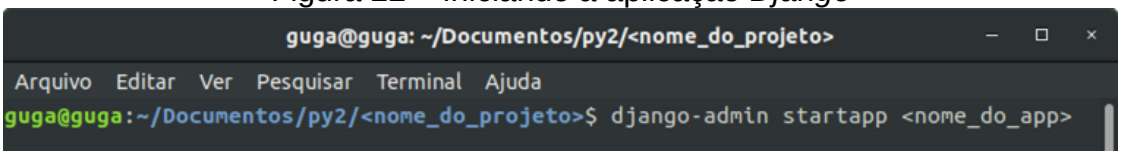
Fonte: dos próprios autores

Este arquivo da Figura 21 acima não necessariamente precisa ser o único, como demonstra a figura o mesmo está redirecionando os caminhos para outros dois arquivos que também contém URLs, mas como este se trata do principal tem essa função de reenviar as requisições aos outros arquivos.

Após a configuração dos arquivos principais, foi criado um APP, ou seja, uma aplicação dentro do projeto principal onde se contém as funções do *backend* na

linguagem *Python* e também contém os *Templates* onde estão contidos a parte HTML da aplicação. Para a criação deste APP em questão foi usado o comando demonstrado na Figura 22 abaixo:

Figura 22 – Iniciando a aplicação *Django*



Fonte: dos próprios autores

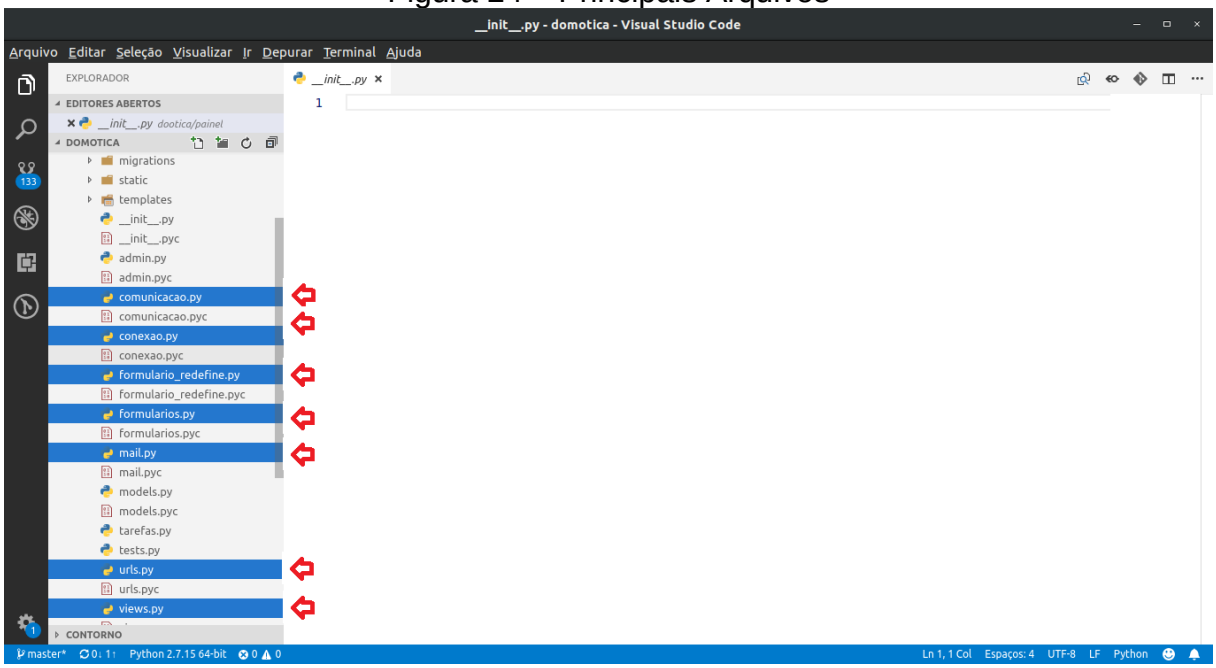
Após a criação do APP dentro do projeto original o *Django* criou uma pasta onde foi criado arquivos para a edição e o desenvolvedor necessita criar o resto dos arquivos necessários de acordo com sua aplicação. Houve a necessidade de criar um arquivo chamado views.py onde se encontrou várias funções *Python* que foram utilizadas pelo projeto. A Figura 23 seguir demonstra a sintaxe do arquivo com algumas funções.

Figura 23 – Arquivo views.py

Fonte: dos próprios autores

Houve necessidade de criação de vários outros arquivos para a finalização do projeto a figura abaixo demonstra seus nomes e quantidade para melhor visualização. Como na Figura 24 a abaixo:

Figura 24 – Principais Arquivos



Fonte: dos próprios autores

2.4.2 Programação da *Black Board*

Inicialmente foi necessário programar o microcontrolador, de modo que possibilitasse a utilização dos módulos e também o monitoramento geral do mesmo. Por escolha dos programadores a *Black Board* funcionará de forma ociosa, ou seja, ela ficará aguardando um comando externo (vindo da aplicação). Ao receber o comando ele será tratado, executará as funções requeridas por ele e gerará um retorno para a aplicação. Ações automáticas que não requerem intervenção do usuário são executadas internamente na *Black Board* enquanto a mesma aguarda uma requisição, essas ações geram apenas retornos para a aplicação.

A parte inicial do código do microcontrolador é composto por inclusão de bibliotecas (neste caso a única biblioteca externa utilizada foi a *SerialRelay*.h); pela declaração das funções que serão utilizadas posteriormente; e também a declaração das variáveis e constantes globais para que pudessem ser utilizadas em diferentes funções. Tudo isso pode ser visto na Figura 25 a seguir:

Figura 25 – Escopo inicial *Black Board*

```
#include <SerialRelay.h>

const byte NumModules = 2; // Número de Módulos Serial Relé
SerialRelay relays(4,5,NumModules); // Pinos Serial Relé: Data - 4, Clock - 5, 12v - Vin, GND - GND

char valor_recebido;

boolean a=false; // VARIÁVEL PARA ARMAZENAR O RETORNO DOS COMANDOS, SE TRUE/FALSE
boolean b=false; // VARIÁVEL PARA RETORNAR O ESTADO ATUAL DO RELÉ

const int LM35 = A0; // DEFINE O PINO QUE LERÁ O RETORNO DO SENSOR DE TEMPERATURA
float temperatura = 0; // VARIÁVEL PARA ARMAZENAR A TEMPERATURA APÓS CALCULADA
float voltagem = 0;
int sensorTemp = 0; // VARIÁVEL USADA PARA LER O VALOR DO SENSOR EM TEMPO REAL

const int luz = A1;  // PINO ANALÓGICO EM QUE O SENSOR ESTÁ CONECTADO
int sensorLuz = 0; // VARIÁVEL USADA PARA LER O VALOR DO SENSOR EM TEMPO REAL

// FUNÇÃO PARA LER O SENSOR E CALCULAR A TEMPERATURA
void Calc_temperatura(){
  sensorTemp = analogRead(LM35);
  voltagem = sensorTemp*(5.0/1023);
  temperatura = voltagem*100;

  Serial.println(temperatura);
}

// FUNÇÃO PARA LER O SENSOR E CALCULAR A LUMINOSIDADE
void Calc_luminosidade(){
  int sensorLuz = analogRead(luz); // LÊ A PORTA ANALOGICA DO SENSOR
  if(sensorLuz > 200){
    a = relays.SetRelay(1, SERIAL_RELAY_OFF, 2);
    Serial.println(sensorLuz);
  }else{
    a = relays.SetRelay(1, SERIAL_RELAY_ON, 2);
    Serial.println(sensorLuz);
  }
}
```

Fonte: dos próprios autores

A comunicação Raspberry-*Black Board* foi consolidada por meio da conexão USB-Serial, tal processo de comunicação necessitou ser iniciado na *Black Board*. Todas as ações de iniciação que devem ser executadas apenas uma vez ao início da execução, devem ser declaradas na função **setup**, já padrão do microcontrolador, como pode ser visto na Figura 26 a seguir:

Figura 26 – Função setup

```
void setup()
{
  Serial.begin(9600);
}
```

Fonte: dos próprios autores

Na função **loop**, outra função padrão do microcontrolador, são inseridas as ações que devem ser executadas constantemente, nela está declarada a tarefa responsável por realizar uma leitura da comunicação serial e aguardar uma variação para que possa prosseguir a execução. Após isso, o comando recebido é armazenado em uma variável e analisado para que as ações respectivas sejam executadas. Segue a Figura 27 para visualização do que foi exposto:

Figura 27 – Função Loop

```
void loop()
{
  while(!Serial.available()){
    // A BLACK BOARD FICA AGUARDANDO UMA TENTATIVA DE COMUNICAÇÃO SERIAL
  }

  // RECEBE A MENSAGEM VIA SERIAL E ARMAZENA NA VARIÁVEL CHAR
  valor_recebido = Serial.read();

  // VERIFICA O COMANDO RECEBIDO E REALIZA A AÇÃO RESPECTIVA E RESPONDE NA PORTA SERIAL O RETORNO DA OPERAÇÃO
  switch (valor_recebido){
    case 'a':
      b = relays.GetState(1, 1);
      if(b==true){
        a = relays.SetRelay(1, SERIAL_RELAY_OFF, 1);
        Serial.println(0);
      }else{
        a = relays.SetRelay(1, SERIAL_RELAY_ON, 1);
        Serial.println(1);
      }
    break;
    case 'b':
      b = relays.GetState(2, 1);
      if(b==true){
        a = relays.SetRelay(2, SERIAL_RELAY_OFF, 1);
        Serial.println(0);
      }else{
        a = relays.SetRelay(2, SERIAL_RELAY_ON, 1);
        Serial.println(1);
      }
    break;
```

Fonte: dos próprios autores

E assim por diante são analisadas as entradas recebidas pela porta serial da *Black Board*. Um único comando vindo da aplicação pode executar uma série de funções distintas na *Black Board*, e capturar seu retorno.

2.4.3 No-IP

O No-IP teve início na década de 90, e tem criador como CEO até os dias atuais. Sua principal funcionalidade é sanar o problema de troca de IPs que ocorre

nos provedores de internet, ou seja, o usuário final tem sempre seu IP alterado pelo provedor, o que dificulta a identificação de dispositivos locais de forma online. A tarefa do No-IP é basicamente acompanhar esta mudança, estando sempre com o novo IP disponível e referenciado, para que o acesso não seja perdido[21].

Para possibilitar o acesso ao sistema mesmo estando fora da residência foi utilizado o recurso citado acima, onde foi criado o *link* que pode ser acessado de qualquer local onde se tenha acesso à internet. Este *link* segue o IP geral da rede local onde o sistema está implantado, contendo uma porta especialmente configurada para permitir tal acesso. A parte visual do sistema foi desenvolvida através da linguagem de marcação HTML, pastes estas também conhecidas como *Templates*, como exposto anteriormente.

2.4.4 React Native

O *React Native* é um *Framework* de desenvolvimento criado pela gigante da tecnologia Facebook , baseado em uma biblioteca JavaScript com o nomenclatura de React que é usada na criação de interfaces de usuário pelo *Facebook* , o *React Native* tem a proposta diferente que é desenvolver para Android e IOS utilizando apenas JavaScript. Seu grande diferencial se comparado com seus “concorrentes” como o *Cordova* ou Manifold.js é que o *React Native* converte todo o código JS para o código nativo do sistema desejado, ou seja, se programa em JS e se converte para código nativo tornando o aplicativo nativo no sistema operacional[22].

Utilizando esta ferramenta foi desenvolvido um aplicativo hibrido simples, a fim de proporcionar ao usuário certa facilidade ao acessar o sistema, pois o mesmo é capaz de identificar se o dispositivo utilizado para acessar o sistema está ou não conectado à internet, podendo assim selecionar o modo local ou remoto do sistema. Se o usuário estiver conectado apenas na rede local sem acesso à internet o sistema encaminhará o mesmo para o *link* local caso contrário será utilizado o *link online* do sistema. Nos demais casos de conexão o sistema exibirá um erro, indicando que a página não está acessível.

---

21 Disponível em: <https://www.noip.com/pt-BR/about>

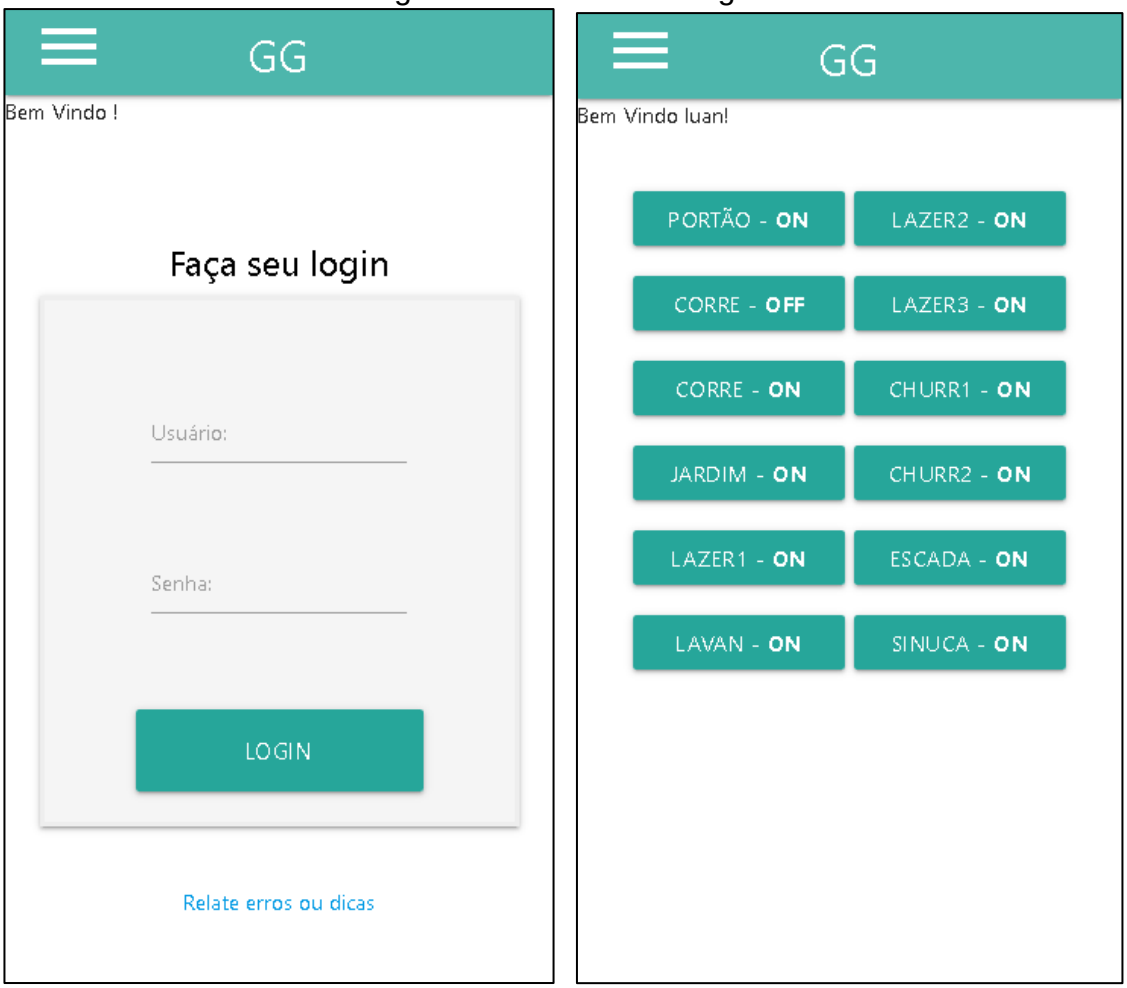
22 (EISENMAN,Bonnie. Learn React Native, Sebastopol ; O’Reilly Media, Inc.;2016.432p.)

2.5 Testes e Análise

O intuito deste tópico é expor os testes realizados e o que foi possível analisar através do uso do sistema.

O sistema de *login*, realiza uma busca na Base SQL criada e gerida através do *Django* Admin. O *login* somente é liberado caso usuário e senha sejam confirmados, a tela inicial é exibida mediante esse procedimento, conforme mostra a Figura 28:

Figura 28 – Efetuando login

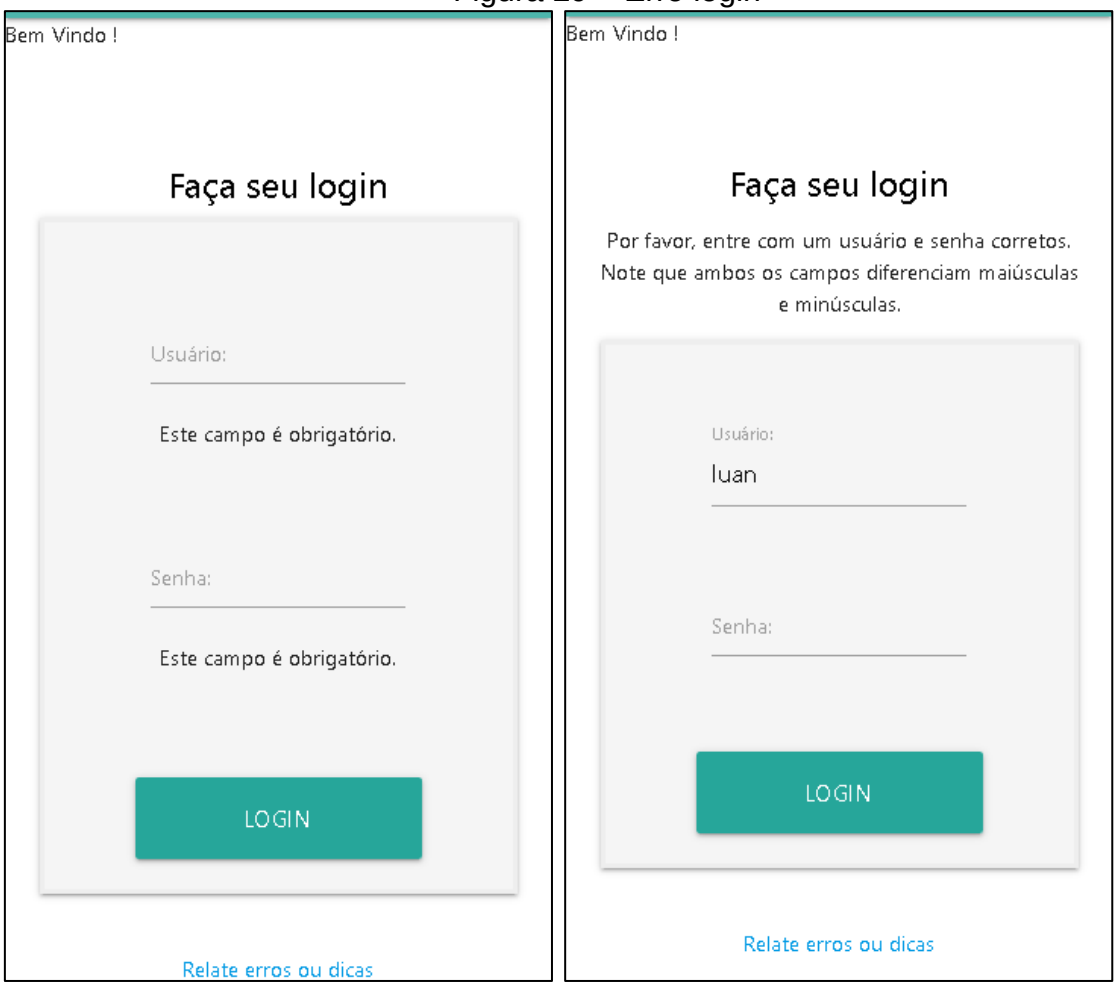


Fonte: dos próprios autores

O sistema verifica as credenciais inseridas, caso estejam divergentes, com dados inválidos ou até mesmo não preenchidos, um alerta de erro é emitido ao

usuário, não permitindo o acesso à página principal. Somente usuários previamente cadastrados conseguem acessar o sistema, como demostrado na Figura 29 a seguir:

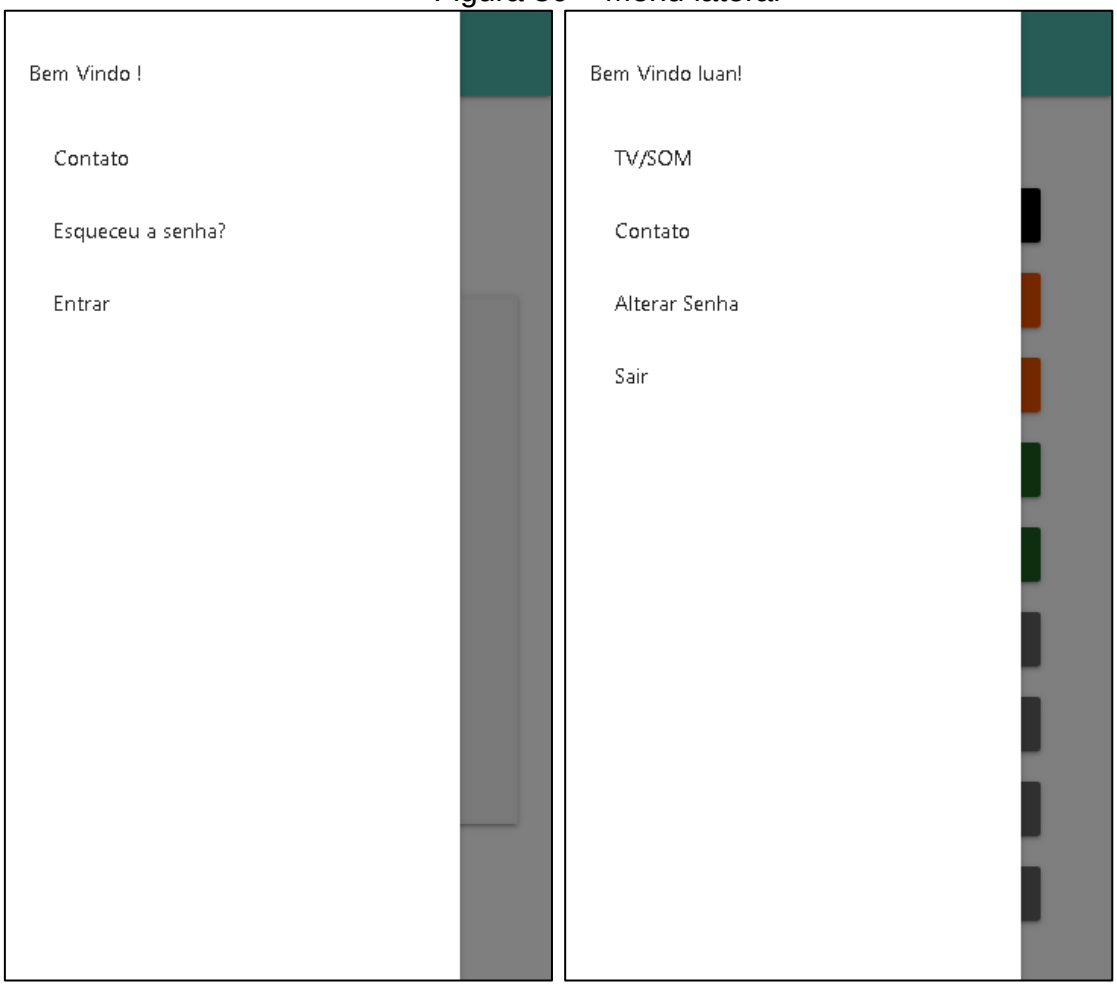
Figura 29 – Erro login



Fonte: dos próprios autores

Outro ponto muito importante é o menu, que varia de acordo com o usuário, se autenticado ou não. Usuários *logados* tem a possibilidade de alterar sua senha, já os usuários com problemas para acessar devem contatar o suporte solicitando uma redefinição de senha, como nota-se na Figura 30:

Figura 30 – Menu lateral



Fonte: dos próprios autores

2.6 Implementação

O sistema foi desenvolvido e implementado na área gourmet de uma residência. Esta área conta contém, corredor, jardim, espaço para churrascos, espaço para televisão e som, área de sinuca e lavanderia. Onde a sinuca e a lavanderia se encontram em um segundo piso (segundo andar), no jardim existe uma escada para acesso a este piso.

Foram automatizados, o portão para acesso ao corredor que leva a esta área, as lâmpadas deste corredor, juntamente com todas as lâmpadas dos demais espaços citados. No total foram, 2 (duas) lâmpadas no corredor, 3 (três) no jardim, 3 (três) no espaço para televisão e som, 3 (três) na área da churrasqueira, 2 (duas) na escada, 2 (duas) no espaço da sinuca, e 1 (uma) na lavanderia.

No espaço para televisão e som foi utilizado um sensor infravermelho[23] para mapear e comandar tanto a TV quanto o TVBOX que automaticamente controla o som. A página para controle do sistema é bastante funcional, composta apenas por botões que são destinados para determinada ação, como exibido na Figura 31:

Figura 31 – Controle Infravermelho

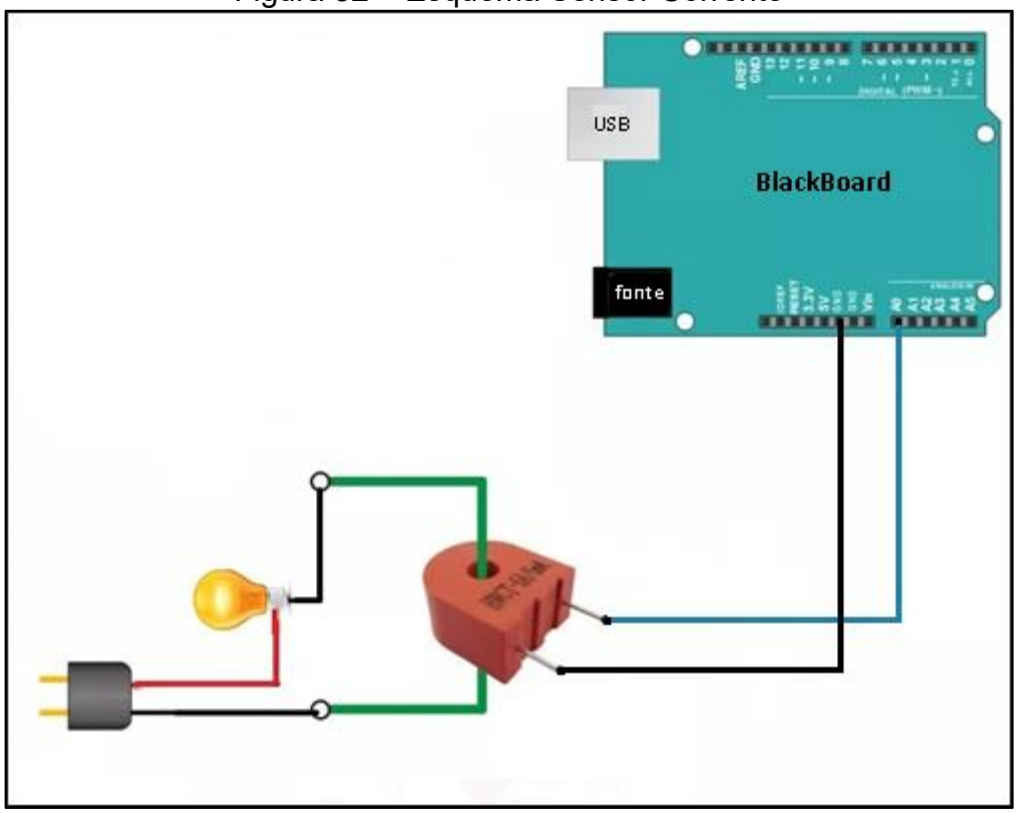
Fonte: dos próprios autores

Na criação do esquema de automação das luzes deste projeto viu-se a necessidade da utilização de um sensor de corrente não invasivo, ou seja, que não se tem a necessidade de cortar os fios a fim de capturar a corrente encontrada nos mesmos, facilitando assim a verificação de corrente. O Sensor em questão foi utilizado para a verificação entre o estado da lâmpada e o estado do relê, que tem a função de

[23] O emissor de luz infravermelho IR333C de alta intensidade, moldado em um pacote de plástico transparente. PT333-3B é um receptor alta sensibilidade e devido ao seu epóxi preto, o dispositivo é sensível à radiação infravermelha (ROBOCORE, 2018).

um interruptor, já que o padrão utilizado de ligações foi o *threeway*[24], por conta deste recurso ser limitado pelas portas analógicas da *Black Board* o mesmo foi utilizado em pontos estratégicos para identificar a passagem de corrente em alguns dispositivos agrupados. A figura 32 abaixo demonstra o sensor e como funciona com o controlador.

Figura 32 – Esquema Sensor Corrente

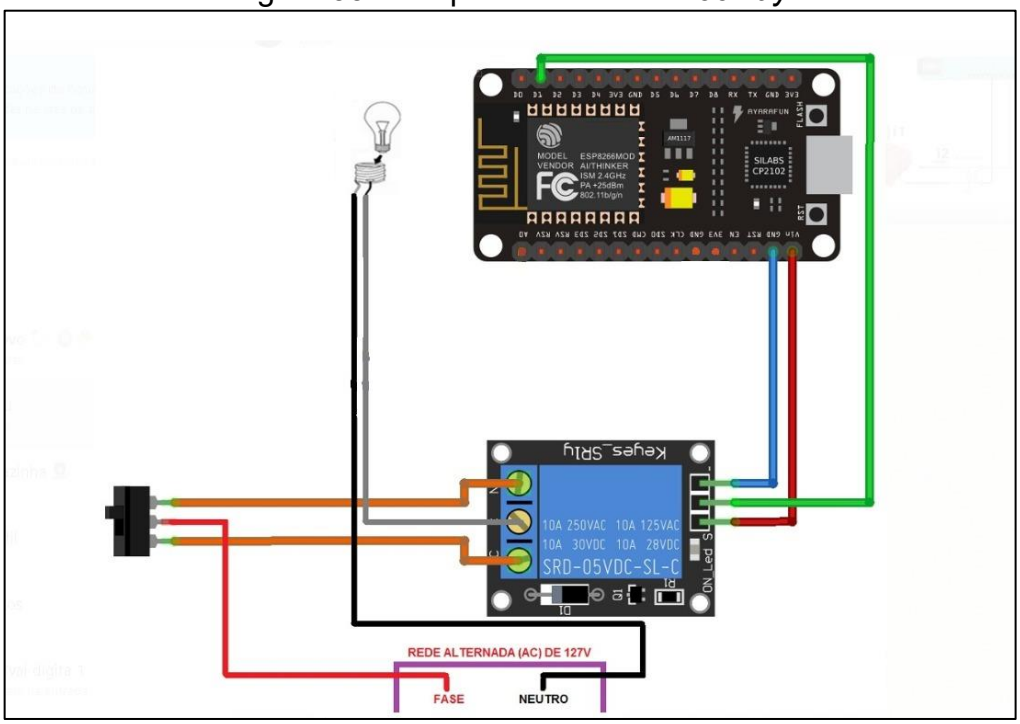


Fonte: Adaptada de: <https://http2.mlstatic.com/10-pcs-sensor-corrente-no-invasivo-5a5ma-*Arduino*-pic-ac-dc-D_NQ_NP_727811-MLB20651323836_032016-F.*web*p>

Na ligação citada acima, o *threeway* é uma ligação muito usada em projetos elétricos e que consiste basicamente em ter dois pontos de controle de uma lâmpada onde se possa acender de um lado e apagar de outro lado. Para não perder a mobilidade da área onde o sistema foi implantado, esse sistema foi a melhor opção. A Figura 33 demonstra como esse sistema de *threeway* funciona com o rele e o microcontrolador.

[24] Disponível em < https://www.mundodaeletrica.com.br/como-ligar-interruptor-paralelo-three-way/>

Figura 33 – Esquema Relé + *Threeway*



Fonte: Adaptada de < http://blogmasterwalkershop.com.br/embarcados/nodemcu/nodemcu-utilizando-com-interruptor-three-way/>

Para comandar a abertura do portão, é utilizado um cabo para enviar um pulso entre a *Black Board* e o dispositivo presente no portão.

2.6.1 Recursos utilizados

Para construir o sistema de automação foram necessários alguns recursos e equipamentos para que tudo fosse possível, segue uma relação com descrição e valores:

Quadro 27: Valor dos equipamentos

DESCRIÇÃO	VALOR
Black Board	R$ 85,00
Cartão MicroSD Classe 10 16gb	R$ 40,00
Rapberry	R$ 109,00
Módulo Relé Serial	R$ 49,00 x 3 = R$ 147,00
Sensor de Corrente	R$ 8,00 x 5 = R$ 40,00
Módulos IR	R$ 2,00
Plug Fêmea	R$ 6,00 x 2 = R$ 12,00
Interruptor *Threeway* Duplo	R$ 17,00 x 2 = R$ 34,00
Interruptor *Threeway* Triplo	R$ 25,00 x 3 = 75.00
Interruptor *Threeway* Individual	R$ 5,50
Fios de Energia 1,5mm	R$ 0,81 x 300 = R$ 243,00
Caixa de Passagem	R$ 43,00
Total	**R$ 835,50**

Fonte: dos próprios autores

Como um dos objetivos deste projeto é o baixo custo, fez-se necessária uma pesquisa de mercado, envolvendo o ambiente nacional onde os desenvolvedores vivem. Com tal pesquisa foram identificas as seguintes soluções similares e seus respectivos preços:

Quadro 28: Valores dos kits similares no mercado

DESCRIÇÃO	VALOR
Mini Google Home	R$ 269,90 [25]
Interruptor WIFI Individual	R$ 170,31[26]
Interruptor WIFI Duplo	R$ 177,12 [27] x 2 = R$ 354,12
Interruptor WIFI Triplo	R$ 197,96 [28] x 3 R$ 593,88
Sensor de Correte	Não Compatível
Fios de Energia	Não Requer
Módulo IR	Não Compatível
Total	**R$ 1.388,21**

Fonte: dos próprios autores

[25] Disponível em: <https://www.americanas.com.br/produto/31380776/google-home-mini-cor-chalk>

[26] Disponível em: <https://produto.mercadolivre.com.br/MLB-1057788026-interruptor-touch-wifi-smart-home-automation-_JM?quantity=1

[27] Disponível em: <https://produto.mercadolivre.com.br/MLB-1085741186-interruptor-wifi-sonoff-touch-t1-us-2-teclas-1152-_JM>

[28] Disponível em: <https://produto.mercadolivre.com.br/MLB-1085748443-interruptor-wi-fi-sonoff-touch-t1-us-3-teclas-1153-_JM>

Como é nítido nas informações apresentadas anteriormente, a mais simples solução disponível, identificada pelos desenvolveres deste projeto, possui um custo financeiro muito elevado se comparada à solução proposta por eles. Um único interruptor duplo compatível com a outra solução exposta tem o valor de aproximadamente 10 (dez) dos interruptores utilizados neste projeto. Além disso, os componentes necessários para esta solução são escassos no mercado nacional (Brasil), por serem fabricados no exterior. Já a maioria dos equipamentos utilizados neste projeto são de fabricação nacional, sendo facilmente encontrados em diversas lojas.

CONCLUSÃO

Este trabalho demonstrou a viabilidade da implantação de um sistema de automação residencial baseado em microcontroladores e a linguagem de programação *Python* para auxiliar a execução de tarefas rotineiras em uma residência.

De acordo com os procedimentos utilizados para alcance do objetivo geral deste trabalho, foram propostos os seguintes objetivos específicos:

- Reunir informações sobre tecnologias *WEB*, para o desenvolvimento da aplicação ou plataforma responsiva para controle da residência;

Para o desenvolvimento deste trabalho, foi necessário basear-se em fontes confiáveis para criação de um ambiente *WEB* que permitisse controlar a residência de forma interativa, como visto no tópico 1.6 e 1.7 do capítulo 1.

- Analisar recursos que promovam comodidade e praticidade em uma residência utilizando tecnologia de automação;

Para identificar os recursos necessários que promovessem conforto e comodidade, fez-se o estudo de tecnologias que possibilitassem tal resultado, como visto no tópico 1.1 do capítulo 1.

- Fazer um levantamento sobre as alterações na parte elétrica e eletrônica da residência que serão necessárias para implantação do sistema;

Foram necessários estudos no local de implantação para identificar as possíveis mudanças que se fariam necessárias neste processo, juntamente com os conhecimentos de elétrica dos desenvolvedores. Além dos esquemas elétricos necessários para o funcionamento do sistema, como visto no tópico 2.6 do capítulo 2.

- Verificar tecnologias semelhantes no mercado, fazendo um levantamento de custos, analisando os equipamentos utilizados pelos mesmos;

Viu-se a necessidade de realizar uma pesquisa de mercado para identificar tecnologias disponíveis no mercado, e verificar a viabilidade da criação de um sistema

próprio utilizando as tecnologias disponíveis visando o baixo custo, tal pesquisa foi exemplificada no capitulo 2 subcapitulo 2.6.1.

- Investigar como os sistemas de automação podem facilitar as tarefas residenciais, mantendo o investimento financeiro viável comparado com os disponíveis no mercado;

Foi-se necessário analisar o funcionamento das tecnologias de automação residencial disponíveis no mercado Brasileiro, e como as mesmas facilitam as tarefas rotineiras, e o quanto de recurso e investimento torna-se requisito para isto, tal pesquisa foi exemplificada no capitulo 2 subcapitulo 2.6.1.

- Desenvolver um sistema de automação que permita acesso via internet ou rede local, que promova mudança na realização das tarefas rotineiras em uma residência, agregando conforto e comodidade aos usuários.

Após reunir todas as informações e recursos necessários para criação do sistema, foi iniciada a implementação, onde foi possível identificar as dificuldades para implantar e adaptar o sistema em um ambiente real.

Tendo em vista a problemática deste trabalho, com base na viabilidade do desenvolvimento de um sistema que possibilite o controle residencial via rede local ou internet, utilizando tecnologia de baixo custo, foram realizados testes durante a implementação e após a implementação, ambos com a participação do usuário final, buscando identificar as melhorias ou pioras trazidas com o sistema, validando ou não as seguintes hipóteses:

H0: Não seria viável o desenvolvimento de uma plataforma de automação residencial, pois essa tecnologia necessita de enormes investimentos financeiros, dispensando o interesse do público alvo.

Esta hipótese **não foi validada**, visto que os investimentos financeiros requeridos não são elevados, tendo em vista a abrangência do sistema. Além de atrair o público alvo por conta dos recursos necessários serem facilmente encontrados no mercado nacional e o preço elevado das soluções similares.

H1: Seria viável o desenvolvimento de uma plataforma de automação residencial utilizando tecnologia de baixo custo, pois a mesma seria mais acessível comparada com recursos similares disponíveis no mercado.

Esta hipótese **foi validada**, visto que os sistemas disponíveis no mercado são em sua maioria importados tendo um preço elevado se encontrados no mercado nacional. Além da dificuldade relacionada à importação e impostos.

H2: Seria viável o desenvolvimento de um sistema de automação residencial, pois o mesmo traria comodidade e praticidade.

Esta hipótese **foi validada**, visto que o sistema após implantado facilita a execução de tarefas rotineiras, facilitando-as. Criando um novo leque de possibilidades para o usuário final, onde ele consegue realizar tais tarefas de forma ágil, sem necessitar de movimentação corporal.

H3: Não seria viável o desenvolvimento de uma plataforma de automação residencial, pois essa tecnologia requer muitas modificações na parte elétrica do local e poderia não ser aceita pelos proprietários.

Esta hipótese **não foi validada**, visto que as alterações na parte elétrica da residência foram mínimas, e não atrapalharam as atividades rotineiras dos moradores. Além disto não foi necessário realizar nenhum procedimento que requeresse alterar a estrutura do local.

H4: Seria viável o desenvolvimento de um sistema de automação residencial com gestão a distância utilizando tecnologia de baixo custo aquisitivo, pois o mesmo traria segurança adicional para a residência, comodidade e um possível auxílio as pessoas com deficiências relacionadas à mobilidade.

Esta hipótese **foi validada**, visto que após a implementação várias atividades puderam ser realizadas de forma otimizada, tornou-se possível monitorar o funcionamento de alguns dispositivos mesmo estando distante geograficamente do local. Como no ambiente de implantação alguns espaços se localizavam em um segundo piso, se algum usuário estivesse impossibilitado ou com dificuldade de locomover-se, o mesmo conseguiria realizar a gestão de alguns recursos e equipamentos sem que fosse necessário subir a escada. Conforme relatos do usuário final no que diz respeito ao uso da aplicação, foi notado que a mesma facilita a gestão dos equipamentos da residência, proporcionando conforto e comodidade.

A conclusão deste projeto resultou no desenvolvimento e implantação de um sistema de automação residencial, servindo para que os moradores consigam gerenciar dispositivos a distância, e também localmente de maneira otimizada, podendo até mesmo auxiliar pessoas com dificuldades relacionadas à mobilidade.

Devido ao pouco tempo disponível para tal projeto, optamos por não implementar o sistema em toda a residência, e alguns dispositivos com menos importância foram deixados para uma futura inclusão. A aplicação necessita de algumas melhorias na parte visual, para se tornar mais interativa e bela para o usuário.

Este trabalho é relevante estabelecendo ganhos para a formação dos desenvolvedores. No âmbito acadêmico, trouxe valor e conhecimento para a formação acadêmica, e também serviu de base para futuras pesquisas para alunos. É inegável o grande ganho para o mercado profissional, já que a tendência mundial é facilitar serviços por meio de automações. Certamente a pesquisa trouxe grande ganho pessoal para os autores, que aprenderam novas tecnologias que estão em alta. No ganho social agregou qualidade de vida, sendo uma solução viável e acessível.

Para desenvolvimentos futuros ficam a finalização de algumas pendências da aplicação como: (1) finalizar o sistema a fim de torná-lo mais reativo as respostas em tempo real dos pontos controlados da residência; (2) abranger a instalação do sistema no restante da residência; (3) desenvolver um módulo de notificações e vigilância para ocorrência de ativação ou desativação de itens de segurança como câmeras e alarmes; (4) permitir a criação de relatórios de acordo com necessidade dos usuários e demais moradores; (5) desenvolver um módulo de abertura automática de portões utilizando a geolocalização do usuário para trazer maior comodidade.

REFERÊNCIAS

ASSIS, Pietro. *Microcontrolador*. 2004. 92f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) - Universidade Presidente Antônio Carlos Faculdade de Ciência da Computação e Comunicação Social, Barbacena, 2004. Disponível em: <http://www.unipac.br/*site*/bb/tcc/tcc-f6cceedfa3f6307211208b80c790c6e3.pdf>. Acesso em: 20 mai. 2018.

ARDUINO. *Documentation*, 2018. Disponível em: <https://www.*Arduino*.cc/reference/en/>. Acesso em: 25 de jun de 2018

_____. *Arduino Uno* R3, 2018. Disponível em: < https://store.*Arduino*.cc/usa/*Arduino*-uno-rev3>. Acesso em: 02 de Jul de 2018

BALTEJO, Paulo; SANTOS, Jorge. *Apontamentos de Programação em C/C++.* São Paulo: Versão Draft, 2006. 108p.

BATISTELLO, Rodrigo. *Automação Residencial Utilizando Raspberry Pi e Android*. 2014. 83f. Trabalho de Conclusão de Curso (Sistemas de Informação) - Universidade do Oeste de Santa Catarina Unoesc – Unidade Chapecó, Chapecó, 2014. Disponível em: <https://www.*web*artigos.com/artigos/automacao-residencial-utilizando-raspberry-pi-e-android/128340/> Acesso em: 26 mai. 2018.

BORGES, Renato; CLINIO, André. *Programação Orientada a Objetos com C++*. 2000.104f. Disponível em: <https://*web*server2.tecgraf.puc-rio.br/ftp_pub/lfm/CppBorgesClinio.pdf>. Acesso em: 30 de set de 2018

CAMPOS, Augusto. *O que é Linux*: BR-Linux. Florianópolis, 2006. Disponível em <http://br-linux.org/faq-linux>. Acesso em: 14 mai. 2018.

CARVALHO, Flávia. *Apostila de Introdução à Linguagem HTML*. Taquara: 2004. 33f. Disponível em <http://correio.fdvmg.edu.br/*download*s/DET475/apostila_html.pdf>. Acesso em: 14 mai. 2018.

CATHARINA, Anna. *Arquitetura de Computadores I*. Natal: 2000. 33f. Disponível em <http://www.dee.ufrn.br/nicolau/ApostArqIok.pdf>. Acesso em: 26 mai. 2018.

DAVIS, Stephen. *C++ Para Leigos*. 7ed. Rio de Janeiro: Alta Books, 2016. 472p.

DJANGO DOCUMENTATION. Disponível em: <https://*Python*.org.br/*web*/>. Acesso em: 07 de set de 2018

DJANGO SOFTWARE FUNDATION. Disponível em: <https://docs.*Django*project.com/en/2.1/>. Acesso em: 06 de set de 2018

DOBAY, Eduardo. *Programação em C*. São Paulo: 2012. 103p. Disponível em <http://fig.if.usp.br/~esdobay/c/c.pdf>. Acesso em: 29 abr. 2018.

DOWNEY, Allen; ELKNER, Jeffrey; MEYERS, Chris. *How to Think Like a Computer Scientist: Learning with Python*. 1. ed. Wellesley, MA: Green Tea Press, 2017.

ELMAN, Julia; LAVIN, Mark. *Django Essencial*. São Paulo: Novatec, 2015.312p.

FILHO, Antonio. *Introdução à Programação Orientada a Objetos com C++*. Rio de Janeiro: Elsevier, 2010. 312p.

GABARDO, Ademir. *Laravel para ninjas*. 1ed. São Paulo: Novatec, 2017. 184p.

GRILLO, Filipe; FORTES, Renata. *Aprendendo JavaScript*. São Carlos: 2008. 47f. Disponível em <http://conteudo.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_ND_72.pdf>. Acesso em: 25 mai. 2018.

JULIO, Kessede. *Apostila de Paradigmas de Programação*. Campinas: 2011. 80f. Disponível em <http://maradentro.com.br/wp-content/uploads/2014/09/Paradigmas_apostila.pdf>. Acesso em: 30 abr. 2018.

LIMA, Adriano. *JavaScript – Aplicações Interativas para a Web*. Belo Horizonte: 2006. 234f. Disponível em <http://pavesys.com.br/*download*/JavaScript.pdf>. Acesso em: 05 mai. 2018.

MACEDO, Felipe. *Programação Web*. 2014. 32f. Trabalho de Conclusão de Curso (Técnico em Informática) – Qwerty Escola de Educação Profissional, Dom Pedrito, 2014. Disponível em < http://escola.qwerty.com.br/artigos/pdf_tcc/tcc_felipe_soares.pdf>. Acesso em: 12 mai. 2018.

MERCES, Ricardo. *Raspiberry Pi: Conceito e Prática*. Rio de Janeiro: Editora Ciência Modema, 2013. 18p.

MORAES, Juliano et al. *Programação Web*. Várzea Grande: 2007. 4f. Disponível em <https://www.inf.pucrs.br/~gustavo/disciplinas/sd/material/Artigo_*Webs*ervices_Conceitual.pdf>. Acesso em: 16 mai. 2018.

MONK, Simon. *Movimento, Luz e Som Com Arduino e Rapberry Pi.* São Paulo: Novatec, 2016. 352p

MURATORI, José; DAL BÓ, Paulo. Capítulo I - Automação residencial: histórico, definições e conceitos. In: MURATORI, José Roberto; DAL BÓ, Paulo Henrique. *Automação Residencial: Conceitos e Aplicação*. 3ed.São Paulo: Editora Edurece, 2011. 70-77. Disponível em: <http://www.instalacoeseletricas.com/*download*/Automacao_residencial1.pdf>. Acesso em: 15 mai. 2018.

NASCIMENTO, Edmar. *Introdução às Redes de Computadores*. Petrolina, Universidade Federal do Vale do São Francisco, 2011. Notas de Aula. Disponível em: <http://www.univasf.edu.br/~edmar.nascimento/redes/redes_20112_aula02.pdf>. Acesso em: 10 de jun de 2018

NETTO, Daniel. *Saber Eletrônica*: O uso do *Raspberry Pi* pelos profissionais de eletrônica, São Paulo, v. 48, n. 468, abr. 2013. Disponível em: <https://pt.slideshare.net/gertech/saber-eletrnica-465>. Acesso em: 28 abr. 2018.

OLIVEIRA, Rômulo; CARISSIMI, Alexandre; TOSCANI, Simão. *Sistemas Operacionais*. 4ed. Porto Alegre: Bookman, 2010. 259p.

PREDOSO, Roberta. *Apostila de HTML*. Niterói: 2007. 101f. Disponível em <https://www.telecom.uff.br/pet/petws/*download*s/apostilas/HTML.pdf>. Acesso em: 16 mai. 2018.

PRESCOTT, Preston. *Programando em JavaScript*. Babelcube, Inc, 2016 Traduzido Por Mayara Ávila. 35p.

PRUDENTE, Francesco. *Automação Predial e Residencial*: Uma Introdução. Rio de Janeiro: LTC, 2011. 228p.

Raspberry Pi. *Manual do Usuário*. 2015. Acesso em: 25 de ago de 2018

_____. *Site oficial*. Disponível em: <https://www.raspberrypi.org/>. Acesso em: 12 de ago de 2018

ROBÔ CORE. *Site oficial*. Disponível em: <https://www.robocore.net/>. Acesso em: 20 de ago de 2018

QUIERELLI, Davi. *Criando Sites com Html Css Php*. Joinville: Clube dos Autores, 2008. 91p.

_____. *CSS3: desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das css3* 1ed. São Paulo: Novatec, 2011. 496p.

_____. *JavaScript: Guia do Programador: Guia completo das funcionalidades de linguagem JavaScript.* 1ed. São Paulo: Novatec, 2015. 608p.

SEVERANCE, Charles*. Python for Informatics: Exploring Information*. Ann Arbor, MI, USA: CreateSpace Independent Publishing Platform, 2013. 224p.

SOUZA, Bruno; JÚNIOR, José; FORMIGA, Andrei. *Introdução a programação*. João Pessoa: UFPB, 2014. 103p.

TAKAI, Osvaldo. *Introdução a Banco de Dados*: 2005. 124f. Disponível em < https://www.ime.usp.br/~jef/apostila.pdf>. Acesso em: 23 mai. 2018.

TANENBAUM, Andrew; WETHERALL, David. *Redes de Computadores*. 5ed. São Paulo: Pearson, 2011. 600p.

TAVARES, Luís. *Uma Solução com Arduino para Controlar e Monitorar Processos*. 2013. 4f. Trabalho de Conclusão de curso (Engenharia de Sistemas Eletrônicos, Automação e Controle Industrial) – Instituto Nacional de Telecomunicações, Santa Rita. Disponível em: <https://www.inatel.br/biblioteca/pos-seminarios/seminario-de-

automacao-industrial-e-sistemas-eletro-eletronicos/i-saisee/9390-uma-solucao-com-*Arduino*-para-controlar-e-monitorar-processos-industriais/> Acesso em: 26 mai. 2018.

VOTRE, Vilmar. *C++ Explicado e Aplicado*. Rio de Janeiro: Alta Books, 2016. 664p.

ZAVALIK, Claudimir. *Integração de Sistemas de Informação Através de Web Service*. Monografia (Mestre em Informática) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2004. Disponível em: <https://www.lume.ufrgs.br/bitstream/handle/10183/4560/000457638.pdf?sequence=1> Acesso em: 26 mai. 2018.

APÊNDICE I

Configurações globais da aplicação - settyngs.py

```
"""
Django settings for domotica project.

For more information on this file, see
https://docs.Djangoproject.com/en/1.7/topics/settings/

For the full list of settings and their values, see
https://docs.Djangoproject.com/en/1.7/ref/settings/
"""

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))


# Quick-start development settings - unsuitable for production
# See https://docs.Djangoproject.com/en/1.7/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '06%txly011e)fa#-3(e@m*@j+ysu!px3u3z=(g-!sd-ayfa=f-'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

TEMPLATE_DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = (
    'Django.contrib.admin',
    'Django.contrib.auth',
    'Django.contrib.contenttypes',
    'Django.contrib.sessions',
    'Django.contrib.messages',
    'Django.contrib.staticfiles',

    'domotica.conta',
    'domotica.painel',
```

```
)

MIDDLEWARE_CLASSES = (
    'Django.contrib.sessions.middleware.SessionMiddleware',
    'Django.middleware.common.CommonMiddleware',
    'Django.middleware.csrf.CsrfViewMiddleware',
    'Django.contrib.auth.middleware.AuthenticationMiddleware',
    'Django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'Django.contrib.messages.middleware.MessageMiddleware',
    'Django.middleware.clickjacking.XFrameOptionsMiddleware',
)

ROOT_URLCONF = 'domotica.urls'

WSGI_APPLICATION = 'domotica.wsgi.application'


# Database
# https://docs.Djangoproject.com/en/1.7/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'Django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# Internationalization
# https://docs.Djangoproject.com/en/1.7/topics/i18n/

LANGUAGE_CODE = 'pt-br'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.Djangoproject.com/en/1.7/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIR = (
    os.path.join(BASE_DIR,"staic")
)
MEDIA_URL = "/media/"
MEDIA_ROOT = "{}/media".format(BASE_DIR)
```

```
#E-mails
EMAIL_BACKEND = 'Django.core.mail.backends.smtp.EmailBackend'
#EMAIL_BACKEND = 'Django.core.mail.backends.console.EmailBackend'
DEFAULT_FROM_EMAIL = 'ggempresaautomacao@gmail.com'
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'ggempresaautomacao@gmail.com'
EMAIL_HOST_PASSWORD = 'domotica2018'
EMAIL_PORT = 587

CONTACT_EMAIL = 'ggempresaautomacao@gmail.com'

# USUARIOS

from Django.core.urlresolvers import reverse_lazy
LOGIN_URL = reverse_lazy('login:login')
LOGIN_REDIRECT_URL = reverse_lazy('tcc:home')
LOGOUT_URL = reverse_lazy('login:sair')
```

APÊNDICE II

Programação *Black Board*

```
#include <ir_Lego_PF_BitStreamEncoder.h>
#include <boarddefs.h>
#include <IRremote.h>
#include <IRremoteInt.h>

#include <SerialRelay.h>
#include <IRremote.h>

const byte NumModules = 2; // Número de Módulos Serial Relé
SerialRelay relays(4,5,NumModules); // Pinos Serial Relé: Data - 4, Clock - 5, 12v - Vin, GND - GND

char valor_recebido;
int q, y;
int sensorValue;

boolean a=false; // VARIÁVEL PARA ARMAZENAR O RETORNO DOS COMANDOS, SE TRUE/FALSE
boolean b=false; // VARIÁVEL PARA RETORNAR O ESTADO ATUAL DO RELÉ



// FUNÇÃO PARA VERIFICAR O STATUS DE CADA DISPOSITIVO COM O SENSOR DE CORRENTE
#define ELECTRICITY_SENSOR A0
```

```
float amplitude_current; //amplitude da corrente
float effective_value; //valor efetivo da corrente

// FUNÇÃO PARA LER O SENSOR DE CORRENTE
int corrente(int cor){
  sensorValue = analogRead(cor);
  if (sensorValue > 4000)
  {
    return 1;
  }else{
    return 0;
  }
}


IRsend irsend; //instanciando um emissor IR

//########################################################################
#####################################
void setup()
{
  Serial.begin(9600);
}


//########################################################################
#####################################
void loop()
{
  while(!Serial.available()){
```

```
    // A Black Board FICA AGUARDANDO UMA TENTATIVA DE COMUNICAÇÃO SERIAL
  }


// RECEBE A MENSAGEM VIA SERIAL E ARMAZENA NA VARIÁVEL CHAR
  valor_recebido = Serial.read();


  // VERIFICA O COMANDO RECEBIDO E REALIZA A AÇÃO RESPECTIVA E RESPONDE NA PORTA SERIAL O RETORNO DA OPERAÇÃO
  switch (valor_recebido){
// COMANDOS PARA OS RELÉS DAS LÂMPADAS
    case 'a':
      a = relays.SetRelay(1, SERIAL_RELAY_OFF, 1);
      delay(1500);
      a = relays.SetRelay(1, SERIAL_RELAY_ON, 1);
      Serial.println(0);
    break;
    case 'b':
      b = relays.GetState(2, 1);
      y = corrente(1);
      if(y==0){
        b=false;
      }
      if(b==true){
        a = relays.SetRelay(2, SERIAL_RELAY_OFF, 1);
        Serial.println(0);
      }else{
        a = relays.SetRelay(2, SERIAL_RELAY_ON, 1);
        Serial.println(1);
      }
```

```
break;
case 'c':
  b = relays.GetState(3, 1);
  y = corrente(2);
  if(y==0){
    b=false;
  }
  if(b==true){
    a = relays.SetRelay(3, SERIAL_RELAY_OFF, 1);
    Serial.println(0);
  }else{
    a = relays.SetRelay(3, SERIAL_RELAY_ON, 1);
    Serial.println(1);
  }
break;
case 'd':
  b = relays.GetState(4, 1);
  y = corrente(3);
  if(y==0){
    b=false;
  }
  if(b==true){
    a = relays.SetRelay(4, SERIAL_RELAY_OFF, 1);
    Serial.println(0);
  }else{
    a = relays.SetRelay(4, SERIAL_RELAY_ON, 1);
    Serial.println(1);
  }
break;
```

```
case 'e':
  b = relays.GetState(1, 2);
  y = corrente(4);
  if(y==0){
    b=false;
  }
  if(b==true){
    a = relays.SetRelay(1, SERIAL_RELAY_OFF, 2);
    Serial.println(0);
  }else{
    a = relays.SetRelay(1, SERIAL_RELAY_ON, 2);
    Serial.println(1);
  }
break;
case 'f':
  b = relays.GetState(2, 2);
  y = corrente(5);
  if(y==0){
    b=false;
  }
  if(b==true){
    a = relays.SetRelay(2, SERIAL_RELAY_OFF, 2);
    Serial.println(0);
  }else{
    a = relays.SetRelay(2, SERIAL_RELAY_ON, 2);
    Serial.println(1);
  }
break;
case 'g':
```

```
  b = relays.GetState(3, 2);
  if(b==true){
    a = relays.SetRelay(3, SERIAL_RELAY_OFF, 2);
    Serial.println(0);
  }else{
    a = relays.SetRelay(3, SERIAL_RELAY_ON, 2);
    Serial.println(1);
  }
break;
case 'h':
  b = relays.GetState(4, 2);
  if(b==true){
    a = relays.SetRelay(4, SERIAL_RELAY_OFF, 2);
    Serial.println(0);
  }else{
    a = relays.SetRelay(4, SERIAL_RELAY_ON, 2);
    Serial.println(1);
  }
break;
case 'i':
  b = relays.GetState(1, 3);
  if(b==true){
    a = relays.SetRelay(1, SERIAL_RELAY_OFF, 3);
    Serial.println(0);
  }else{
    a = relays.SetRelay(1, SERIAL_RELAY_ON, 3);
    Serial.println(1);
  }
break;
```

```
case 'j':
  b = relays.GetState(2, 3);
  if(b==true){
    a = relays.SetRelay(2, SERIAL_RELAY_OFF, 3);
    Serial.println(0);
  }else{
    a = relays.SetRelay(2, SERIAL_RELAY_ON, 3);
    Serial.println(1);
  }
break;
case 'k':
  b = relays.GetState(3, 3);
  if(b==true){
    a = relays.SetRelay(3, SERIAL_RELAY_OFF, 3);
    Serial.println(0);
  }else{
    a = relays.SetRelay(3, SERIAL_RELAY_ON, 3);
    Serial.println(1);
  }
break;
case 'l':
  b = relays.GetState(4, 3);
  if(b==true){
    a = relays.SetRelay(4, SERIAL_RELAY_OFF, 3);
    Serial.println(0);
  }else{
    a = relays.SetRelay(4, SERIAL_RELAY_ON, 3);
    Serial.println(1);
  }
```

```
    break;
//#########################################################################
#####################################

// COMANDOS IR TV E TVBOX


    case 'A':
      irsend.sendNEC(0x2FD48B7, 32); // POWER TV
    break;
    case 'B':
      irsend.sendNEC(0x2FD58A7, 32); // VOLUME + TV
    break;
    case 'C':
      irsend.sendNEC(0x2FD7887, 32); // VOLUME - TV
    break;
    case 'D':
      irsend.sendNEC(0x2FDF00F, 32); // INPUT TV
    break;
    case 'E':
      irsend.sendNEC(0x807F02FD, 32); // POWER TVBOX
    break;
    case 'F':
      irsend.sendNEC(0x807F18E7, 32); // VOLUME + TVBOX
    break;
    case 'G':
      irsend.sendNEC(0x807F08F7, 32); // VOLUME MENOS TVBOX
    break;
    case 'H':
      irsend.sendNEC(0x807F8877, 32); // HOME TVBOX
    break;
```

```
case 'I':
  irsend.sendNEC(0x807F9867, 32); // VOLTAR TVBOX
break;
case 'J':
  irsend.sendNEC(0x807FC837, 32);// OK TVBOX
break;
case 'K':
  irsend.sendNEC(0x807F6897, 32); // SETA CIMA TVBOX
break;
case 'L':
  irsend.sendNEC(0x807F58A7, 32); // SETA BAIXO TVBOX
break;
case 'M':
  irsend.sendNEC(0x807F8A75, 32);// SETA ESQUERDA TVBOX
break;
case 'N':
  irsend.sendNEC(0x807F0AF5, 32); // SETA DIREITA TVBOX
break;
case 'O':
  irsend.sendNEC(0x807F32CD, 32); // OPÇÕES TVBOX
break;
case 'P':
  irsend.sendNEC(0x807F00FF, 32); // ATIVAR/DESATIVAR CURSOR TVBOX
break;
case 'Q':
  irsend.sendNEC(0x807FC23D, 32); // CONFIGURAÇÕES TVBOX
break;
case 'R':
  irsend.sendNEC(0x807FF00F, 32); // APLICATIVOS TVBOX
```

```
    break;


//#########################################################################
#####################################
    case 'z':
      Serial.println(1);
      q = 0;
      while(q<5){
        while(!Serial.available()){
       // A Black Board FICA AGUARDANDO UM VALOR PARA SABER QUAL PINO
LER
        }
        int pinoC = Serial.read();
        int z = corrente(pinoC);
        Serial.println(z);
        q++;
      }
    break;


  // SE O COMANDO NÃO ATENDER NENHUM DOS TESTES ANTERIORES
RETORNA 0/FALSE PARA IDENTIFICAR UMA POSSÍVEL FALHA
    default:
      a = false;
      Serial.println(a);
    break;
  }


}
```

APÊNDICE III

Página de Login – login.html

```
<title>Login</title>
<main>
  <div class="center-align">
    <img class="responsive-img" style="width: 250px;" src="" />

    <h5 class="black-text">Faça seu login</h5>

    {{ form.non_field_errors }}
    <div class="z-depth-1 grey lighten-4 row" style="display: inline-block; padding: 50px 60px 0px 60px; border: 3px solid #EEE;">

      <form class="col s12" method="post">
        {% csrf_token %} {% for field in form %}
        <div class='row'>
          <div class='input-field col s12'>
            {{field.label_tag}} {{field}} {{field.errors}}
          </div>
        </div>
        {% endfor %}
        <br/>
        <div class="center-align">
          <div class='row'>
            <button type='submit' name='bt_login' class='col s12 btn btn-large waves-effect'>Login</button>
          </div>
        </div>
      </form>
    </div>
    <p class="text-center">
      <a href="{% url 'tcc:contato' %}">Relate erros ou dicas</a>
    </p>
  </div>
</main>
```

APÊNDICE IV

Principais comandos *Black Board*

Entradas e Saídas Digitais	Funções Matemáticas	Números Aleatórios
digitalRead()	abs()	random()
digitalWrite()	constrain()	randomSeed()
pinMode()	map()	
	max()	Bits e Bytes
Entradas e Saídas Analógicas	min()	bit()
analogRead()	pow()	bitClear()
analogReference()	sq()	bitRead()
analogWrite()	sqrt()	bitSet()
		bitWrite()
Apenas Zero, Due e Família MKR	Funções Trigonométricas	highByte()
analogReadResolution()	cos()	lowByte()
analogWriteResolution()	sin()	
	tan()	Interrupções Externas
Entradas e Saídas Avançadas		attachInterrupt()
noTone()	Caracteres	detachInterrupt()
pulseIn()	isAlpha()	
pulseInLong()	isAlphaNumeric()	Interrupções
shiftIn()	isAscii()	interrupts()
shiftOut()	isControl()	noInterrupts()
tone()	isDigit()	
	isGraph()	Comunicação
Funções Temporizadoras	isHexadecimalDigit()	Serial
delay()	isLowerCase()	Stream
delayMicroseconds()	isPrintable()	
micros()	isPunct()	USB
millis()	isSpace()	Keyboard
	isUpperCase()	Mouse
	isWhitespace()	

APÊNDICE V

App.js – *React Native*

```
import React, {Component} from 'react';
import { WebView,NetInfo } from 'react-native';

export default class App extends Component {
  constructor (props) {
    super(props)
    this.state = {
      uri: 'https://www.google.com.br'
    }
  }

  componentWillMount(){

    fetch(this.state.uri)
      .then((response) => {
        if (!response.status === 200) {
          this.setState({ uri: "http://192.168.0.1" })
        }
      })
      .catch((error) => {
          console.log('network error: ' + error);
      })

  }

  render() {
    return (

        <WebView
        source = {{ uri: this.state.uri }}
        style={{marginTop: 20}}
        />

    );
  }
}
```