



GOVERNO DO ESTADO DO RIO DE JANEIRO
SECRETARIA DE ESTADO DE CIÊNCIA E TECNOLOGIA E INOVAÇÃO
FUNDAÇÃO DE APOIO À ESCOLA TÉCNICA
FACULDADE DE EDUCAÇÃO TECNOLÓGICA DO ESTADO DO RIO DE JANEIRO
FAETERJ/PETRÓPOLIS

Implementação de um sistema de controle de acesso com Arduino

Victor Ferra Maximiano

Petrópolis - RJ

Dezembro, 2019

Victor Ferra Maximiano

Implementação de um sistema de controle de acesso com Arduino

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Tecnólogo em Tecnologia da Informação e da Comunicação da Faculdade de Educação Tecnológica do Estado do Rio de Janeiro Faeterj/Petrópolis, como requisito parcial para obtenção do título de Tecnólogo em Tecnologia da Informação e da Comunicação.

Orientador:

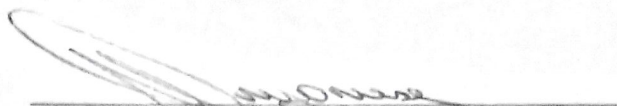
Alberto Torres Angonese

Petrópolis - RJ

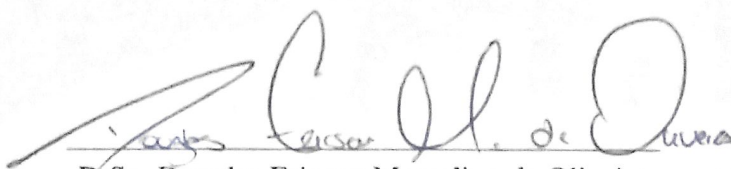
Dezembro, 2019

Folha de Aprovação

Trabalho de Conclusão de Curso sob o título “*Implementação de um sistema de controle de acesso com Arduino*”, defendida por Roberto Higor Matos dos Anjos e aprovada em 04 de Dezembro de 2019, em Petrópolis - RJ, pela banca examinadora constituída pelos professores:



D.Sc. Alberto Torres Angonese
Orientador



D.Sc. Douglas Ericson Marcelino de Oliveira
FAETERJ/Petrópolis



D.Sc. Eduardo Krempser da Silva
FIOCRUZ

Declaração de Autor

Declaro, para fins de pesquisa acadêmica, didática e tecnico-científica, que o presente Trabalho de Conclusão de Curso pode ser parcial ou totalmente utilizado desde que se faça referência à fonte e aos autores.

Victor Ferra Maximiano
Petrópolis, em 04 de Dezembro de 2019

Dedicatória

Dedico esta, aos meus pais Márcia e Roberto, e a minha companheira Julia que com seu apoio e incentivo constante me ajudou a chegar onde eu estou.

Agradecimentos

Primeiramente a Deus que me permitiu chegar onde estou.

A minha companheira Julia, pelo amor, pelo incentivo e pelo apoio constante que foram muito importantes para o desenvolvimento do projeto.

À instituição – funcionários, direção, administração e seu corpo docente pelo suporte prestado, por fornecer o material necessário para a realização do trabalho e que se mantém firme nos ideais em que acredita para proporcionar um ensino público de qualidade. Em especial aos amigos do laboratório Gabriel, Matheus e Vitor pelo suporte nos testes e grande ajuda na montagem do projeto.

E por último, mas não menos importante, ao professor D.Sc. Alberto Angonese, meu orientador, pela oportunidade, pela compreensão, pela amizade e pelo incentivo que garantiram conclusão deste trabalho.

Epígrafe

"To succeed, planning alone is insufficient. One must improvise as well"

(ASIMOV, Isaac; Foundation, 1951)

Resumo

A Internet das coisas trouxe novas possibilidades quanto ao acesso e geração de novos dados através vários tipos de dispositivos. Por ser um tema amplo, ela pode ser aplicada em diversas áreas, como nesse caso em segurança e controle. Diante disso, esse trabalho implementou um protótipo de controle de acesso ao laboratório da FAETERJ/Petrópolis. Foi implementado uma aplicação *desktop* responsável pela a comunicação entre o Arduino e o banco de dados, além de uma aplicação *web* que permite um acesso remoto ao sistema. A aplicação *web* também é responsável pelo cadastro dos usuários e por fornecer uma interface de usuário para visualizar os dados coletados. Foi utilizado no desenvolvimento das aplicações a linguagem de programação Python. Além disso foi utilizado ferramentas como o *framework* Django e o *Bootstrap 4* para ajudar no desenvolvimento *web*. Na implementação do Arduino foi utilizado a sua própria linguagem semelhante ao C++. O projeto poderá ser expandido no futuro para todas as salas visto que há uma infraestrutura na instituição. A arquitetura do sistema foi estruturada de forma que no futuro tire proveito da Internet das coisas, tornando o Arduino um dispositivo inteligente. Atualmente o sistema pode ser acessado apenas por dispositivos na rede local da instituição. Na parte eletrônica foi utilizado uma placa Arduino Uno R3 com um *shield Ethernet*. Foi também utilizado um módulo relé ligado a fechadura eletrônica, um teclado para receber os dados do usuário e um *buzzer* para emitir sons nas teclas. Foram feitos testes para a validação e através dos resultados obtidos foi possível gerar novas informações e comprovar a eficácia do sistema.

Palavras-chave: Internet das coisas. Controle de acesso. Arduino. Python

Abstract

The Internet of things brought new possibilities of access and data generation across various kinds of devices. Because it's a broad theme, it can be applied in several fields, in this case security and control. Thus, the present work implemented an access control prototype in the laboratory of FAETERJ/Petrópolis. It used a desktop application responsible for the communication between the Arduino and the database plus a web application that allows remote access to the system. The web application also allows the register of new users and provides a user interface to visualize the data collected. To develop the applications it used the Python programming language besides tools like the framework Django and Bootstrap 4 to ease the web development. The Arduino hardware is programmed using its built-in language similar to C++. The project can expand in the future to all the classrooms in the institution, since the infrastructure is already installed. The system architecture is structured in a way that, in the future, it can take advantage of the Internet of things, turning the Arduino in a smart device. Currently, it only can be accessed through devices inside the institution's local network. The electronic part of the implementation was built using an Arduino Uno R3 board with an Ethernet shield. It uses a relay module connected to the electronic lock, a membrane keyboard to receive the user's data and a buzzer to make the key's sound. The system was tested to confirm the work and through the results it was possible to generate new information and prove the efficiency of the system.

Keywords: Internet of things. Access control. Arduino. Python

Lista de Figuras

2.1	Esquemático de uma placa Arduino	p. 21
2.2	Placa Arduino Uno	p. 23
2.3	Ilustração do <i>shield Ethernet</i>	p. 23
3.1	Instalação da placa <i>Arduino</i> ligada ao disjuntor	p. 31
3.2	Arquitetura do sistema de monitoramento de meio ambiente	p. 32
4.1	Diagrama do funcionamento do sistema.	p. 35
4.2	Diagrama das ligações do Arduino	p. 37
4.3	Arduino e telado instalados na parede	p. 37
4.4	Diagrama do banco de dados.	p. 40
4.5	Página de registro da aplicação.	p. 44
4.6	Página mostrando o acesso.	p. 44
4.7	Exemplo de página responsivo.	p. 45
5.1	Módulo de teclado acoplado a parede.	p. 50
5.2	Página inicial da aplicação contendo os dados coletados.	p. 50

Lista de Tabelas

3.1	Comparação do Arduino.	p. 33
3.2	Comparação da aplicação.	p. 34
3.3	Comparação de funcionalidades com o Arduino.	p. 34
4.1	Campos da tabela Usuário	p. 40
4.2	Campos da tabela Acesso	p. 41
4.3	Campos da tabela Registro	p. 41
5.1	As horas com mais acessos.	p. 51
5.2	Os dias com mais acessos.	p. 51
5.3	O horario onde ocorreu o acesso mais cedo na semana.	p. 51
5.4	O horário onde ocorreu o acesso mais tardio na semana.	p. 51

Lista de trecho de códigos

4.1	Matriz do teclado	p. 38
4.2	Estrutura condicional utilizada para permissionamento	p. 42
4.3	Método da <i>view</i> responsável por obter os registros de acesso	p. 42
4.4	<i>Model do acesso</i>	p. 43
4.5	Método utilizado para a verificação de senha	p. 46
4.6	Método utilizado para checar se existe um usuário com a senha digitada . . .	p. 48
A.1	ArduinoBanco.ino	p. 56
B.1	base.html	p. 60
C.1	wsgi_windows.py	p. 63
D.1	mainEthernet.py	p. 64
E.1	UsuarioDao.py	p. 67
F.1	views.py	p. 70

Lista de abreviaturas

ANSI American National Standards Institute

BNDES Banco Nacional de Desenvolvimento

DAO Data Access Object

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

IoT Internet of Things

MVC Model-View-Controller

ORM Object-Relational Mapping

RFID Radio Frequency Identification

SGBD Sistema de Gerenciamento de Banco de Dados

SPI Serial Peripheral Interface

SQL Structured Query Language

SSL Secure Sockets Layer

URL Uniform Resource Locator

USB Universal Serial Bus

WAMP Windows, Apache, MySQL, and PHP

WSGI Web Server Gateway Interface

WSN Wireless Sensor Network

Sumário

1	Introdução	p. 16
1.1	Justificativa	p. 17
1.2	Motivação	p. 18
1.3	Objetivos	p. 18
1.3.1	Objetivo geral	p. 18
1.3.2	Objetivo específico	p. 19
1.3.3	Organização dos capítulos seguintes	p. 19
2	Fundamentação teórica	p. 20
2.1	Arduino	p. 20
2.1.1	O que é o Arduino	p. 20
2.1.2	Arduino Uno	p. 22
2.1.3	Módulo <i>shield Ethernet</i>	p. 22
2.2	Django	p. 24
2.2.1	Framework web	p. 24
2.2.2	O que é o Django	p. 25
2.2.3	Organização dos arquivos	p. 26
2.3	Sistema de gerenciamento de banco de dados	p. 27
2.4	Internet das Coisas	p. 27
3	Revisão bibliográfica	p. 30
3.1	Medidor de energia elétrica	p. 30

3.1.1	Arquitetura do sistema	p. 30
3.1.2	Arduino	p. 30
3.1.3	Aplicação web	p. 31
3.2	Monitoramento de meio ambiente	p. 32
3.2.1	Arquitetura do sistema	p. 32
3.2.2	Aplicação <i>web</i>	p. 33
3.3	Comparação entre as implementações	p. 33
4	Implementação	p. 35
4.1	Apresentação do Sistema	p. 35
4.2	Arduino	p. 36
4.2.1	Hardware	p. 36
4.2.2	Software	p. 37
4.3	Aplicação <i>web</i>	p. 39
4.3.1	Disponibilização dos dados	p. 39
4.3.2	Desenvolvimento do sistema <i>web</i>	p. 41
4.3.3	Configuração do servidor	p. 45
4.4	Aplicação <i>desktop</i> em Python	p. 46
4.4.1	Comunicação com o Arduino	p. 46
4.4.2	Comunicação com o banco de dados	p. 47
5	Experimentos e resultados	p. 49
6	Conclusão e trabalhos futuros	p. 52
	Referências	p. 54
	Anexo A – Código implementado na fechadura eletrônica	p. 56

Anexo B – Template base do site	p. 60
Anexo C – Arquivo WSGI utilizado no Windows	p. 63
Anexo D – Código principal da aplicação Python	p. 64
Anexo E – Comunicação da aplicação Python com o banco de dados	p. 67
Anexo F – Arquivo de <i>views</i> da aplicação de monitoramento	p. 70

1 *Introdução*

A evolução tecnológica dos dias atuais tornou possível o acesso à dispositivos com um maior poder de processamento a um baixo custo e tamanho reduzido, como é no caso dos microcontroladores, sendo um deles a placa Arduino. Também nos trouxe aplicações web que permitem um acesso mais facilitado por dispositivos diferentes. Com a revolução da Internet das Coisas (do inglês: *Internet of Things*), foi possível que diferentes dispositivos, como os microcontroladores, consigam acesso a internet tendo assim um grande potencial de uso nas mais diversas áreas das atividades humanas. (SANTOS et al., 2016).

Hoje em dia as aplicações web se popularizaram devido ao fácil acesso através de qualquer dispositivo sem que seja preciso instalar fisicamente a aplicação, além de deixar o processamento para o servidor. É possível combinar as vantagens das aplicações web com as possibilidades do Arduino. Uma dessas possibilidades é a de fornecer dados de controle para que o usuário consiga visualizar um registro de dados enviados do Arduino para o banco de dados.

Por estes fatores, foi possível unir o *hardware* do Arduino e a comunicação dos protocolos de rede para se obter um envio e recebimentos de dados com outros dispositivos ou aplicações. O Arduino pode ser utilizado para diversas funções através de módulos que ajudam a expandir suas funcionalidades. Foi aproveitado essas possibilidades para utilizar o Arduino como um recurso de segurança, para controlar o acesso ao laboratório de robótica além de no futuro utilizar em outras salas ou departamentos da instituição, sendo importante para proteger os ativos e controlar quem possui acesso a eles. Dentro dos módulos que foram utilizados para este objetivo inclui-se o relé, responsável por fornecer um controle de energia, um o módulo de teclado, utilizado para captar os dados digitados pelo usuário e um *shield Ethernet*, para permitir com que o Arduino se comunique via um cabo *Ethernet*.

Diante deste contexto, percebeu-se a possibilidade de utilização dos microcontroladores presentes na instituição para que seja realizado um melhor controle de acesso nas salas da instituição. O presente trabalho propõe o desenvolvimento de um sistema de controle utilizando-se de um banco de dados para armazenar os cadastros de cada usuário além dos registros de acesso

fazendo o maior uso das capacidades do Arduino junto da *Internet of Things (IoT)* com uma aplicação responsável por realizar a comunicação entre o Arduino e o banco de dados. Também foi implementada uma aplicação web para fornecer ao usuário uma interface de cadastro, visualização de senhas além dos registros obtidos pelo Arduino. A visualização, por ser em uma aplicação web, pode ser realizada tanto por computadores quanto por *smartphones* presentes na rede da instituição.

1.1 Justificativa

O estudo se justifica em virtude da carência dados e de desenvolvimento a respeito da Internet das coisas no Brasil. Hoje em dia o número de dispositivos eletrônicos capazes se conectar à Internet vem crescendo rapidamente, não mais exclusivamente celulares ou computadores já que até dispositivos mais simples como uma lâmpada ou uma geladeira possuem acesso. Estes dispositivos suportam controle remoto, possuem a capacidade de gerar dados para serem analisados, além da possibilidade de automação, tanto doméstica ou industrial (SANTOS et al., 2016)

A disseminação da Internet das coisas irá transformar tanto a economia quanto o dia a dia da população de maneira tão ou mais impactante do que a robótica avançada, tecnologias *Cloud* ou até mesmo do que a internet móvel. De acordo com o professor da universidade de Harvard Porter e Heppelmann (2015), a IoT é a mudança mais substancial na produção de bens desde a Segunda Revolução Industrial.

Ao perceber o aumento em pesquisas e uso da Internet das coisas, o Banco Nacional de Desenvolvimento (BNDES) criou um plano de ação para alavancar as pesquisas e desenvolvimentos sobre IoT a fim de permitir que o Brasil seja capaz se posicionar de forma destacada nesta corrida tecnológica global.

Como foi constatado pelo relatório, a Internet das coisas trará uma revolução muito importante para os países que efetuarem uso desta tecnologia, havendo também um grande interesse por parte das multinacionais e *startups* que vem investindo em pesquisas e no desenvolvimento de soluções (BNDES, 2017). É importante então que o Brasil consiga se posicionar de forma destacada nesta corrida tecnológica a fim de conseguir um maior aproveitamento, não ficando atrás desta nova tendência global.

Um estudo do McKinsey Global Institute feito por Manyika et al. (2015) estima que o impacto da IoT na economia global será de 4% a 11% do produto interno bruto do planeta em 2025. Até 40% desse potencial será capturado pelas economias emergentes e no caso específico

do Brasil, a estimativa é de 50 a 200 bilhões de dólares de impacto econômico anual em 2025.

Além disso, a Internet das coisas não oferece exclusivamente benefícios econômicos como também proporciona benefícios para a sociedade auxiliando o mundo a atingir seus objetivos sociais, como foi comunicado em junho de 2017 pela União Internacional de Telecomunicações, agência da Organização das Nações Unidas para o setor de comunicações (BNDES, 2017).

Portanto, conforme os dados vistos acima, é possível perceber a necessidade de estudos sobre a Internet das coisas considerando que esta nova tecnologia será de grande importância para o futuro, podendo trazer benefícios tanto econômicos quanto sociais em especial para a instituição FAETERJ.

1.2 Motivação

O Arduino permite hoje em dia com que pessoas com poucos conhecimentos em eletrônica consigam fazer projetos simples sem dificuldades e com baixo custo. A proposta apresentada é a de trazer um produto computacional de segurança para a faculdade utilizando-se do Arduino. O uso de recursos de segurança é importante para a proteção e controle dos ativos da instituição. A proposta também irá trazer para a instituição um conhecimento quanto a criação de um sistema de senhas utilizando um microcontrolador, *socket* e desenvolvimento web que foi obtido durante a fase de pesquisa e implementação do projeto.

O trabalho se seguiu para atender uma demanda da própria faculdade, a fim de contribuir no projeto que consiste na implementação de fechaduras eletrônicas controladas por senha. O projeto da faculdade é de implementar em todas as salas da instituição utilizando-se de placas Arduino já adquiridas. O propósito desse projeto é o de se obter um controle de acesso mais rigoroso, a princípio para o laboratório de robótica, para no futuro trazer mais segurança para a FAETERJ/Petrópolis. A proposta implementada neste trabalho se diferencia do projeto atual por fornecer a parte de comunicação com o banco de dados, além da aplicação web, pois o projeto atual utiliza senhas já armazenadas no Arduino, sem um controle de acesso sobre elas.

1.3 Objetivos

1.3.1 Objetivo geral

O objetivo geral deste trabalho constitui-se na pesquisa e desenvolvimento de um sistema de segurança com controle de acesso a sala do laboratório da FAETERJ/Petrópolis utilizando a

plataforma Arduino conectada via um cabo *Ethernet* a um computador rodando uma aplicação utilizando a linguagem de programação Python responsável por enviar os dados para serem armazenados no banco de dados. Os dados serão lidos por uma aplicação web implementada utilizando-se do *framework* web Django quem também será utilizado para realizar os cadastros dos usuários e fornece um painel de acesso para os usuários.

1.3.2 Objetivo específico

- Implementar o software do Arduino responsável por receber os dados de um teclado numérico e os enviar para uma aplicação em Python;
- Implementar a comunicação do Arduino com o Python utilizando o módulo *Ethernet shield* e uma comunicação por *socket*;
- Efetuar a liberação de uma fechadura eletrônica dependendo da resposta recebida;
- Implementar uma aplicação Python que irá se comunicar via *sockets* com o Arduino e realizar a autenticação no banco de dados;
- Guardar no banco de dados o registro de cada acesso realizado;
- Implementar um sistema web utilizando o *framework* Django que permita o cadastro de usuários e o registro de acessos apenas para os administradores;
- Realizar testes no laboratório da FAETERJ/Petrópolis utilizando um computador da sala como servidor e um Arduino instalado na porta;
- Apresentar os resultados obtidos.

1.3.3 Organização dos capítulos seguintes

Este Trabalho de Conclusão de Curso está organizado da seguinte forma: no **Capítulo 2** é apresentada uma visão geral da fundamentação teórica necessária para compreender os conceitos envolvidos das técnicas utilizadas neste trabalho. No **Capítulo 3** é apresentada a revisão bibliográfica comparando a proposta apresentada com trabalhos semelhantes na área. No **Capítulo 4** é apresentada a implementação utilizada para realizar a tarefa de controle de acesso utilizando o Arduino, as aplicação com *sockets* em Python e a aplicação web em Django propostas neste trabalho. No **Capítulo 5** é apresentado os experimentos realizados e os resultados obtidos. No **Capítulo 6** é apresentado as considerações finais do trabalho e discussões sobre trabalhos futuros.

2 *Fundamentação teórica*

Neste capítulo serão apresentados os conceitos teóricos que dão embasamento para o desenvolvimento deste trabalho em que são abordados conceitos de engenharia de software (PRESSMAN; MAXIM, 2021)

2.1 **Arduino**

Nesta parte será apresentada os conceitos sobre a placa Arduino utilizada na construção do componente eletrônico da proposta.

2.1.1 **O que é o Arduino**

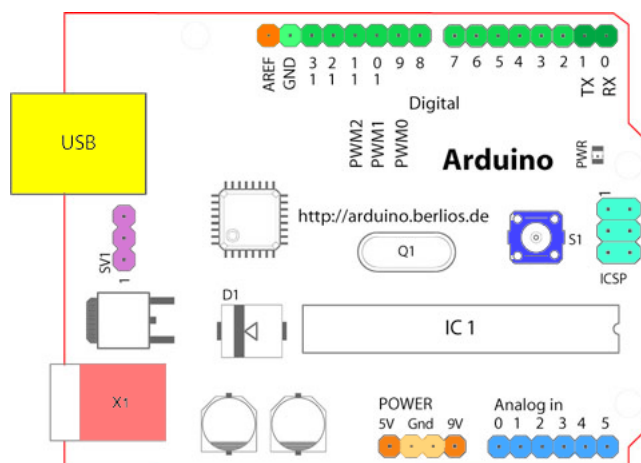
O Arduino é uma plataforma de prototipagem eletrônica *open source* com o intuito de fácil utilização tanto em relação a seu *hardware* quanto a seu *software*. As placas Arduino são capazes de ler entradas e transformá-las em uma saída através de seu microcontrolador. O Arduino suporta diversos tipos de entrada, desde módulos de botões e teclados como também sensores e *LEDs* enquanto as saídas podem ser desde o envio de um sinal de energia ou uma mensagem para um módulo conectado a placa. As instruções sobre como tratar as entradas é feito utilizando a *Arduino programming language* no ambiente de desenvolvimento *Arduino Software Integrated Development Environment* (IDE) (ARDUINO.CC, c2019d).

O Arduino surgiu em 2005 como um *fork* do projeto *Wiring* feito por um dos professores da banca do Wiring. Este *fork* no futuro veio a se tornar o projeto Arduino. O *Wiring* tinha o objetivo de oferecer um *hardware* controlado por *software* acessível para não especialistas, permitindo com que qualquer pessoa elaborasse protótipos e projetos de eletrônica sem dificuldades (BARRAGÁN, 2004).

O Arduino possui diversos pinos esses separados em digital e analógico, como demonstrado na 2.1. Nesses pinos serão conectados os componentes adicionais para fornecer uma entrada

ou receber uma saída. Os pinos digitais podem ser utilizados com entradas e saídas de propósito geral com os métodos *pinMode()*, *digitalRead()* e *digitalWrite()* disponíveis na *Arduino programming language*. Cada pino pode ser ligado ou desligado através dos argumentos *HIGH* para ligado e *LOW* para desligado no método *digitalWrite()*. Já os pinos analógicos suportam uma conversão utilizando o método *analogRead()* em que permite com que a maioria deles seja utilizada como se fossem digitais (ARDUINO.CC, c2019b)

Figura 2.1: Esquemático de uma placa *Arduino*¹.



A Placa pode ser alimentada via conexão Universal Serial Bus (USB), por uma bateria externa ou por uma fonte externa além de fornecer energia para outros componentes através dos pinos VIN, 5V, 3V3 e GND.

A *Arduino programming language* utilizada para programar as funcionalidades do Arduino é baseada em C++, tendo inclusive suporte às bibliotecas do C++. A linguagem do Arduino é bastante similar a utilizada no projeto *Wiring* (ARDUINO.CC, c2019d) pois tanto a linguagem quanto a IDE foram baseadas nas respectivas versões desenvolvidas para o *Wiring* e assim como as do *Wiring*, elas também são *open source* e multiplataformas.

Os comandos escritos na linguagem do Arduino são enviados para a placa do Arduino através do *Arduino IDE* e em seguida armazenados na memória flash da placa Arduino (ARDUINO.CC, c2019e) que por ser uma memória não-volátil, retém os comandos mesmo caso haja uma perda de energia.

As principais vantagens do Arduino são:

- Baixo Custo - As placas Arduino são muito mais baratas comparadas a outras plataformas microcontroladoras.

¹<https://www.arduino.cc>

- **Multiplataforma** - A *Arduino IDE* funciona no Windows, Macintosh OSX e sistemas operacionais Linux.
- **Ambiente de programação simples** - A *Arduino IDE* é fácil de ser utilizada por iniciantes porém avançada para usuários mais avançados.
- **Software open source e extensível** - A *Arduino IDE* é publicada como ferramenta *open source*, disponível para programadores experientes, podendo ser expandida através de bibliotecas C++.
- **Hardware open source e extensível** - O projeto das placas Arduino são publicados sobre a licença *Creative Commons*, permitindo com que seja possível criar sua própria versão.

2.1.2 Arduino Uno

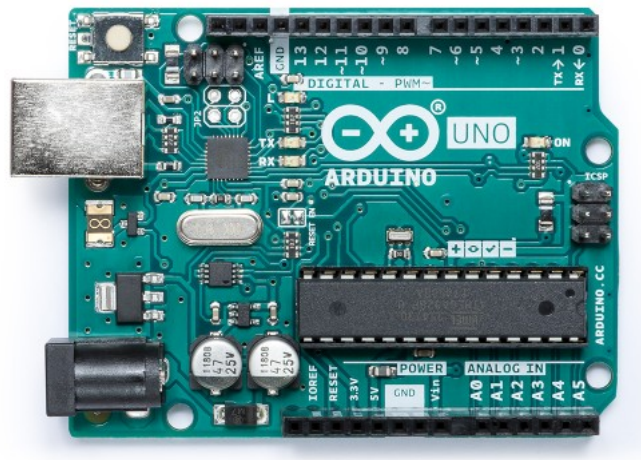
Por ser *open source*, existe uma pluralidade de placas Arduino, dispondo de diversos modelos de placas, módulos e *shield*. A placa Arduino utilizada na proposta foi a Arduino Uno (Um em italiano) que segundo a descrição do site oficial (ARDUINO.CC, c2019a), foi escolhido para marcar o lançamento da versão 1.0 do Arduino IDE, servindo como modelo de referência para a plataforma Arduino, sendo a primeira placa da série de placas Arduino com conexão USB. Desde então, o Arduino Uno vem recebendo diversas revisões, sendo a utilizada na proposta a revisão 3. Foi também utilizado o módulo *shield ethernet W5100*.

O Arduino Uno visto na Figura 2.2 é baseado no microcontrolador *ATmega328P*, dispondo de 14 pinos digitais para entrada/saída, 6 pinos analógicos, conexão USB, um oscilador de cristal de *16MHz*, uma entrada para a fonte de alimentação, um botão de *reset* e um *ICSP header*.

2.1.3 Módulo *shield Ethernet*

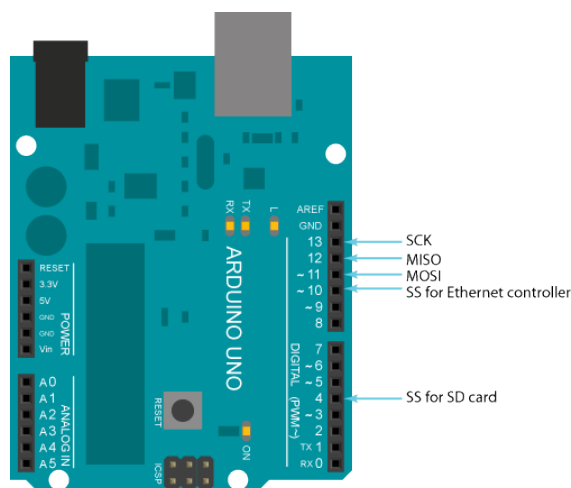
Os módulos *shield* são placas que podem ser conectadas em cima da placa *Arduino* para estender suas capacidades. Existem diversos *shield* disponíveis para o *Arduino* feito por terceiros, como o *Xbee* que permite com que múltiplos *Arduino* se comuniquem através de uma conexão sem fio, o *Motor Control shield* que permite o controle de máquinas de corrente contínua e o *Critical Velocity Accelerometer shield* que integra um acelerômetro de 3 eixos (D'AUSILIO, 2012).

²<https://www.arduino.cc>

Figura 2.2: Placa Arduino Uno².

No projeto foi utilizado o *shield Ethernet R3 W5100*³ para permitir uma conexão do *Arduino* com o servidor na rede através do cabo Ethernet utilizando o padrão IEEE 802.3. O cabo é conectado no roteador, podendo enviar ou receber dados de qualquer computador da rede ou até mesmo na internet.

O Arduino se comunica com o *shield* e o *slot* de cartão SD via Serial Peripheral Interface (SPI) através do *ICSP header*. O SPI necessita de pinos, sendo eles o 10, 11, 12 e 13 além do pino 4 para o *slot* de cartão SD (ARDUINO.CC, c2019c) como pode ser visto na 2.3. Esses pinos então são reservados para o *shield* e não podem ser utilizados por módulos. O *shield* suporta os seguintes protocolos TCP/IP: TCP, UDP, ICMP, IPv4 ARP, IGMP, PPPoE e *Ethernet*. As especificações completas do *shield* podem ser encontradas em seu *datasheet* citado pelo trabalho.

Figura 2.3: Ilustração do *shield Ethernet*⁴.

³https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_datasheet_v1.16.pdf

2.2 Django

Nesta parte será apresentado os conceitos teóricos do *framework web* Django utilizado para facilitar a programação de aplicações *web* utilizando o Python.

2.2.1 Framework web

Um *framework* é um conjunto de ferramentas prontas que fornecem uma infraestrutura para o desenvolvimento de aplicações sem que o desenvolvedor necessite resolver novamente problemas que já foram solucionados pela comunidade ou pelo próprio desenvolvedor. Isso permite um maior foco no desenvolvimento de códigos mais limpos e que possam ser mantidos ao mesmo tempo em que se desenvolve de maneira mais rápida (HOLOVATY; KAPLAN-MOSS, 2009).

Um *framework web* nesse caso são as ferramentas para aplicações *web*. Hoje em dia já existem diversos conjuntos ferramentas prontas que são utilizadas por diferentes linguagens de programação, sendo o Django que será discutido em seguida, uma delas voltado para a linguagem de programação Python.

Os *framework* dispõem de diversos tipos ferramentas como por exemplo o Object-Relational Mapping (ORM) que foi utilizado no trabalho para realizar um mapeamento objeto-relacional dos modelos da aplicação Django, facilitando na criação das tabelas do banco de dados por conta de gerar as consultas Structured Query Language (SQL) conforme está modelado as classes do sistema além de permitir uma fácil migração da aplicação para diferentes tipos de banco de dados por não precisar escrever o SQL específico.

Existem também ferramentas de segurança, que conseguem abstrair esta parte da aplicação como no caso do Django que fornece formulários, telas e tabelas de usuário já prontos, não precisando com que o desenvolvedor se preocupe em implementar sistemas de criptografia ou tabelas para um permissionamento pois o Django já realiza essas tarefas, ainda fornecendo opções para o desenvolvedor personalizar do jeito que for necessário.

Há diversos outros tipos de ferramentas, tendo o desenvolvedor a capacidade de criar novas a partir de suas necessidades ou descobertas e as compartilhar com a comunidade. Uma das limitações de *framework* é o fato de ser necessário seguir a estrutura de projetos propostas por ele, o que pode ser um fator positivo ou negativo dependendo do projeto.

⁴<https://www.arduino.cc>

2.2.2 O que é o Django

O Django é um *framework web* para a linguagem de programação Python que fornece ferramentas para facilitar o desenvolvimento de aplicações *web* com Python provendo uma abstração de alto-nível para padrões comuns e atalhos para tarefas frequentes de desenvolvimento a fim de diminuir o retrabalho. O Django traz também convenções claras de como resolver os problemas, porém permitindo com que o desenvolvedor modifique um pouco do escopo do *framework* (HOLOVATY; KAPLAN-MOSS, 2009).

O surgimento do Django foi de forma orgânica em 2003 enquanto desenvolvedores Adrian Holovaty e Simon Willison que já utilizavam o Python para desenvolver aplicações *web* para o *Lawrence Journal-World Newspaper* tiveram a necessidade de encontrar uma forma de conseguir cumprir os prazos criados pela *World Online Team* para a implementação de novas funcionalidades em aplicações existentes ou até que novas aplicações fossem criadas do zero, o que acabou exigindo muito mais dos seus desenvolvedores.

O Desenvolvimento do *framework web* se tornou a única forma de cumprir esses prazos, eliminando a necessidade de criar novas funcionalidades do zero, aproveitando as já desenvolvidas, o que resultou em uma grande economia de tempo. Em 2005 eles decidiram lançar esse *framework* publicamente utilizando-se de uma licença *open source*, o nomeando Django.

De acordo com Holovaty e Kaplan-Moss (2009), pelo fato do Django ter surgido no desenvolvimento de sites de notícias, ele possui muitas ferramentas como sua página de *admin* que funciona muito bem com sites como a *Amazon*, *Craigslist* e o *The Washington Post* que fornecem um conteúdo dinâmico e movidos a um banco de dados. Apesar disso, o Django não é limitado a apenas esses modelos de aplicação *web*, podendo ser utilizado em diversos outros tipos de aplicações.

Pelo fato do Django não ter surgido no mundo acadêmico ou ter sido um produto comercial, e sim surgido no dia a dia, ele é precisamente focado em resolver os problemas do desenvolvimento *web* com Python no qual os próprios desenvolvedores já vivenciaram. De acordo com (HOLOVATY; KAPLAN-MOSS, 2009), isso acaba também motivando que esses desenvolvedores, por necessidade, contribuem também com o projeto Django com o interesse de garantir com que o Django ajude a diminuir o tempo de desenvolvimento, a produzir aplicações que são fáceis de se manter e que executem bem sobre alta demanda.

2.2.3 Organização dos arquivos

O Django segue vagamente a organização de acordo com o padrão de arquitetura Model-View-Controller (MVC), tendo arquivos de código separados de acordo com sua funcionalidade no sistema a fim de manter uma melhor organização dos arquivos e de suas funcionalidades. O MVC de acordo com Holovaty e Kaplan-Moss (2009) é uma maneira de se desenvolver *software* de forma com que os códigos de definição e acesso os dados (o *model*) esteja separado da lógica de requisição e roteamento (o *controler*), que por consequência esteja separado da interface do usuário (a *view*).

A grande vantagem dessa abordagem de organização é que os componentes ficam desacoplados, permitindo que seja realizado modificações sem que afetem o funcionamento dos outros componentes. Um designer pode por exemplo mudar a aparência de uma página sem olhar o código responsável por a renderizar. Um administrador de banco de dados pode renomear uma tabela e especificar a mudança em apenas um lugar ao invés de precisar achar e modificar em diversos arquivos diferentes.

No Django, a organização dos arquivos é feita da seguinte maneira, separando os assuntos nos seguintes arquivos principais:

- **models.py:** É o arquivo que contém as descrições de tabelas do banco de dados, cada tabela sendo representada por uma classe Python chamada de *model*. Através das *model* criadas, é possível executar as funções no banco de dados utilizando apenas código simples em Python, não sendo necessário escrever queries SQL pois o próprio Django irá gerá-los através de seu ORM.
- **views.py:** Contém a lógica de negócio para a página. A página pode ser uma função ou classe que será chamada em determinado endereço, dependendo de como for implementado.
- **urls.py:** É onde é especificado qual *view* carregar para uma determinada *Uniform Resource Locator (URL) pattern*. Por exemplo, uma URL `"/sobre/"` irá chamar a função (ou classe) `view_sobre()` mapeada neste endereço que irá então renderizar o conteúdo da página.
- **template.html:** São os arquivos Hypertext Markup Language (HTML) em si, onde é descrito o design das páginas. Também é possível adicionar códigos em python dentro da expressão `"{% %}"`

2.3 Sistema de gerenciamento de banco de dados

Os Sistemas de Gerenciamento de Banco de Dados (SGBD) são um conjunto de softwares que facilitam no gerenciamento de banco de dados. Existem diversos SGBD no mercado porém o escolhido para o projeto foi o PostgreSQL pelo fato dele ser grátis e de possuir o código aberto.

O PostgreSQL é um banco de dados objeto-relacional que se originou em 1986 baseado no POSTGRES, um projeto da Universidade de Califórnia em Berkeley e acabou conquistando uma grande reputação por sua arquitetura comprovada, confiabilidade, integridade e pela dedicação da comunidade *open source* além de funcionar na maioria dos sistemas operacionais (POSTGRESQL, c2019).

Os SGBD trabalham com uma linguagem de *scripts* chamada SQL utilizada para modificar os dados armazenados no banco de dados além de executar consultas e inserções de dados. Cada SGBD possui suas particularidades e comandos próprios porém existe um consenso quanto ao SQL padronizado pelo American National Standards Institute (ANSI). Existem ferramentas que geram *scripts* SQL específicos para cada banco, chamados de ORM como o utilizado no Django. Apesar disso, alguns *scripts* foram escritos para a aplicação em Python pois não foi utilizado um ORM na aplicação responsável pela comunicação com o Arduino que realiza algumas inserções e consultas.

2.4 Internet das Coisas

A Internet das coisas é um novo paradigma da computação, onde o foco é a possibilidade de diversos tipos de dispositivos se conectarem à internet. Atualmente o número de novas publicações acadêmicas sobre IoT vem crescendo com a ajuda de diversos campos diferentes, como os de sensores sem fio, novas tecnologias de redes de comunicação e o de nanotecnologia (MADAKAM et al., 2015).

O termo IoT envolve a junção de “internet” com “coisas”. Coisas são os objetos reais no mundo físico, podendo ser qualquer objeto do dia a dia. As “Coisas” incluem dispositivos eletrônicos como geladeiras, celulares, máquina de lavar-roupas etc., não se limitando porém apenas a esses dispositivos que utilizamos diariamente, mas também incluindo objetos não eletrônico como: roupas, móveis, materiais, pontos turísticos, monumentos e pinturas (KOSMATOS et al., 2011). De acordo com Madakam et al. (2015) na IoT também podem ser considerados como objetos seres vivos, desde pessoas até diversos tipos de animais e plantas.

A IoT possui diversas definições dependendo de quem a está utilizando. Apesar dessas diferenças, existe um consenso na definição de que a Internet das coisas é a versão da internet onde os dados não são apenas criados por pessoas, mas também por objetos do dia a dia, que agora têm a capacidade de se conectarem a internet. Segundo Madakam et al. (2015), a definição comum de IoT na literatura é ser uma rede aberta e compreensível de objetos inteligentes que possuem a capacidade de auto-organização, compartilhar informações, dados e recursos, além de reagir e agir na face de situações e mudanças no ambiente.

De acordo com Madakam et al. (2015), pelo fato do termo não possuir um consenso quanto sua definição mas sim diversas interpretações diferentes pelos múltiplos campos que o utilizam, a IoT não possui uma padronização universal, tendo um ambiente heterogêneo que depende de quais tecnologias estão sendo utilizadas. Portanto a IoT necessita de uma padronização para que cada usuário não apenas crie o sistema do seu próprio jeito (SANTOS et al., 2016).

Outro problema da Internet das coisas é devido a segurança, visto que os dispositivos podem ser acessados de qualquer lugar. Por conta de serem dispositivos com baixo poder de processamento, eles não possuem um nível alto de segurança. Além disso, eles acabam dependendo da internet e podem sofrer com problemas de latência ou queda de conexão. Também tem problemas de privacidade pelo fato de ser centralizado, pois a Internet das Coisas acaba coletando muitos dados, que podem ser atacados em um só lugar.

O termo foi utilizado pela primeira vez em uma apresentação feita por Ashton et al. (2009) enquanto trabalhava na P&G. Nessa apresentação, o termo era associado a tecnologia de Radio Frequency Identification (RFID), tendo Ashton sua própria interpretação do que é a Internet das coisas, não controlando como o termo é interpretado hoje em dia. No entanto, o que contribuiu para a popularização do termo foram os avanços nas Wireless Sensor Network (WSN), que utiliza sensores interconectados em diversas áreas como no meio ambiente ou na saúde (XU et al., 2014).

A IoT trouxe também avanços em diversas áreas industriais como monitoramento ambiental, agricultura e vigilância de segurança (XU et al., 2014). Além disso trouxe também avanços para o consumidor comum, como sistemas de casas inteligentes onde é possível utilizar de diversos sensores em diferentes dispositivos que nos fornecem informações sobre estes objetivos ou também pode nos dar controle de funções básicas (KELLY et al., 2013).

Com cada vez mais dispositivos sendo capazes de nos fornecer dados, é possível gerar dados de forma mais frequentes, o que antes sem a utilização de dispositivos, era feito de modo limitado e de forma, além de ser descentralizado. A IoT também permite a coleta de dados que antes não eram possíveis, como diversos sensores que conseguem se comunicar e enviar informações para

um servidor central, como por exemplo, sensores individuais de um automóvel que fornecem vários tipos de informações como temperatura e consumo de combustível. Juntos, esses dados podem revelar outras informações como sobre a performance do carro comparado com sua especificação oficial (PORTER; HEPPELMANN, 2015).

Através da análise desses novos dados, obtém-se uma maior gama de possibilidades de informações que podem trazer benefícios tanto para a indústria quanto para o consumidor normal, nos fornecendo poderosos *insights* para identificar padrões em milhares de dados que podem ser obtidos em vários produtos durante o tempo. Esses dados agora que agora podem vir em um nível imprescindível de variedade e volume.

3 *Revisão bibliográfica*

Neste capítulo será mostrado os trabalhos acadêmicos e artigos avaliados que possuem características semelhantes à proposta ou que foram utilizados como inspiração para o desenvolvimento.

3.1 Medidor de energia elétrica

O projeto analisado foi desenvolvido por Guedes (2018) como trabalho de conclusão de curso e consiste em um medidor de consumo elétrico utilizando placa Arduino para coletar e enviar dados que serão armazenados em um banco de dados na nuvem.

3.1.1 Arquitetura do sistema

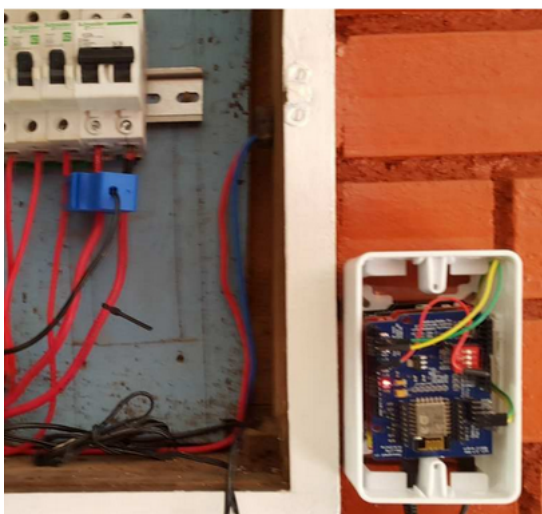
O Arduino é conectado a um disjuntor através de um módulo sensor de corrente. Os dados coletados pelo Arduino são enviados para o servidor web por requisições Hypertext Transfer Protocol (HTTP) e então armazenados em um banco de dados na nuvem, diferenciando-se do sistema de controle de acesso onde o envio dos dados é para uma aplicação *desktop* em Python, que irá armazenar os dados no banco de dados, sendo então separado da aplicação *web*.

3.1.2 Arduino

O Arduino utilizado no projeto possui uma conexão sem fio utilizando-se de um *shield* que é conectado via serial, enquanto o sistema controle de acesso utiliza um *shield Ethernet* conectado via SPI e necessita de um cabo RJ-45 conectado diretamente ao roteador. O trabalho de GUEDES utiliza a biblioteca *WiFiEsp* para abstrair o controle do módulo sem fio, ao invés de utilizar a biblioteca *socket* do Python e a *Ethernet* no Arduino para facilitar a comunicação via *socket*. No trabalho de GUEDES foi necessário um *shield* adicional contendo um circuito de acoplamento para o sensor de corrente.

A instalação do sistema controle de acesso possui semelhanças com a utilizada por GUEDES, também consistindo em uma placa Arduino preso a parede em uma caixa de plástico, que pode ser visto mostrado na Figura 4.3. Nesse caso o Arduino de GUEDES está conectado ao disjuntor, ao invés de uma fechadura eletrônica, como pode ser visto na Figura 3.1. Também foi utilizado no sistema controle de acesso um teclado acoplado no outro lado da parede para receber os dados dos usuários, ao invés de um medidor de corrente que coleta os dados do disjuntor e envia automaticamente para o Arduino.

Figura 3.1: Instalação da placa *Arduino* ligada ao disjuntor¹.



3.1.3 Aplicação web

A aplicação *web* foi construída utilizando a linguagem de programação PHP e utiliza o SGBD MySQL, enquanto a aplicação *web* do sistema de controle de acesso foi construída utilizando Python com o *framework WEB* Django e o SGBD PostgreSQL, porém com a possibilidade de ser migrado para outros SGBD por conta das ferramentas do Django.

O servidor *web* nesse caso é o responsável por receber os dados do medidor e inseri-los no banco de dados além de prover uma página para a visualização desses dados, se diferenciando do sistema de controle de acesso onde a aplicação web é apenas responsável pela visualização dos dados, tendo a aplicação *desktop* em Python a responsabilidade de receber os dados do Arduino e armazená-los no banco de dados.

¹(GUEDES, 2018)

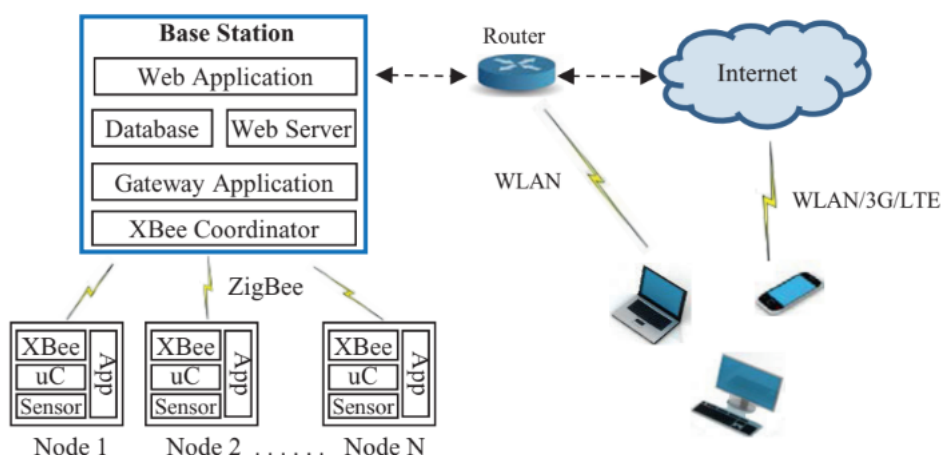
3.2 Monitoramento de meio ambiente

Esta implementação feita por Ferdoush e Li (2014) e consiste em um sistema para o monitoramento de meio ambiente utilizando-se do Arduino devido a ser um *hardware* de baixo custo e baixo consumo de energia, além de suportar sensores de solo, sendo então uma alternativa barata para realizar esta tarefa.

3.2.1 Arquitetura do sistema

A arquitetura do sistema pode ser vista na Figura 3.2, consistindo em uma estação base sendo executada em um microcomputador Raspberry Pi de baixo custo que possui um processador ARM equivalente a um *Pentium II 300MHz*, enquanto a implementação do sistema de controle de acesso é executada em um computador pessoal disponível na instituição para a realização dos testes.

Figura 3.2: Arquitetura do sistema de monitoramento de meio ambiente².



A estação base recebe os dados de diversos nódulos Arduino que estão conectadas a ela via uma conexão sem fio. Estes dados são armazenados no banco de dados que está sendo executado na própria estação junto da aplicação *web*.

Assim como no sistema de controle de acesso, a estação base utiliza uma aplicação em Python que efetua a comunicação com o Arduino diferente, por exemplo, da implementação realizada no projeto do GUEDES onde a placa Arduino se comunica diretamente a aplicação *web*. A aplicação Python de FERDOUSH utiliza a biblioteca *XBee 2.1.0* para a comunicação sem fio.

²(FERDOUSH; LI, 2014)

Cada nó visto na Figura 3.2 é composto por uma placa Arduino contendo um sensor de umidade e temperatura e um módulo *XBee S2B* utilizando o padrão de comunicação *ZigBee* para efetuar uma transferência de dados sem fio com a estação base além de poder controlar de forma simples o comportamento dos sensores. Foi utilizado a biblioteca *Xbee* no Arduino abstrair a comunicação (FERDOUSH, 2014)

3.2.2 Aplicação web

A aplicação *web* de FERDOUSH foi desenvolvida em PHP assim como a de GUEDES. No entanto, assim como o sistema de controle de acesso, a aplicação *web* de FERDOUSH permite apenas uma visualização dos dados armazenados no banco de dados, enquanto utiliza aplicação separada em Python responsável pelo tratamento e armazenamento dos dados. A principal vantagem dessa abordagem utilizada no sistema de controle de acesso, é o fato da aplicação *web* funcionar independentemente da aplicação em *desktop*, podendo ser alterada sem que afete o funcionamento do sistema. Além disso, possui a capacidade de ser executada em outra estação ou até mesmo na nuvem sem muita dificuldade. No entanto, diferente dos 2 trabalhos aqui comparados, o sistema de controle de acesso funciona a princípio somente na rede local da instituição.

3.3 Comparação entre as implementações

De modo a facilitar o entendimento de trabalhos relacionados em comparação com o trabalho apresentado, a Tabela 3.1 compara o módulo utilizado para conexão além das bibliotecas utilizadas no software do Arduino, levando em consideração que os projetos utilizaram a placa Arduino UNO R3.

Tabela 3.1: Comparação do Arduino.

Trabalho	Tipo de conexão	Bibliotecas
Guedes	Sem fio através do módulo ESP8266	DHT e Xbee
Ferdoush	Sem fio através do módulo XBee Pro S2B	Emonlib, WiFiEsp e SoftwareSerial
Nossa proposta	Cabo RJ45 com o <i>shield Ethernet</i> RE W5100	Ethernet e Keypad

Enquanto isso, na Tabela 3.2 é possível ver as diferenças da aplicação utilizada para realizar a comunicação com o Arduino além das bibliotecas utilizadas. Já na Tabela 3.2 é mostrado as

diferenças da aplicação utilizada para realizar a comunicação com o Arduino além das bibliotecas utilizadas.

Tabela 3.2: Comparação da aplicação.

Trabalho	Linguagem	Comunicação	Bibliotecas
Guedes	PHP	Requisição HTTP	Nenhuma
Ferdoush	Python	Utilizando o padrão ZigBee	PySerial 2.7, MySQL-python 1.2.5, APScheduler 2.1.2 e XBee 2.1.0
Nossa proposta	Python	Bytes via Sockets TCP	psycpg2 2.8.2

Por último, na Tabela 3.3 é demonstrado as funcionalidades implementadas no Arduino, nesse caso se ele envia os dados, recebe uma resposta em tempo real após o envio dos dados e se é possível controlar o Arduino remotamente.

Nesse caso nossa proposta possui uma resposta em tempo real os dados enviados para a aplicação são processados e retornados imediatamente por utilizar um canal de comunicação contínuo. No caso de controle, somente a implementação de Ferdoush possui um nível de controle remoto, conseguindo desligar sensores remotamente ou alterar o tempo de envio de dos dados.

Tabela 3.3: Comparação de funcionalidades com o Arduino.

Trabalho	Envio de dados	Resposta em tempo real	Controle remoto
Guedes	Sim	Não	Não
Ferdoush	Sim	Não	Sim
Nossa proposta	Sim	Sim	Não

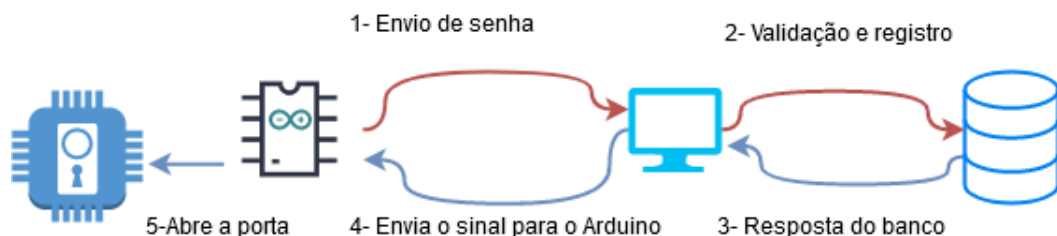
4 Implementação

Neste capítulo será descrito como foi feita a implementação de cada parte do sistema, tanto do *hardware* quanto do *software* além de uma breve explicação de como o sistema funciona e como as diversas aplicações estão relacionadas.

4.1 Apresentação do Sistema

O sistema de controle e monitoramento das fechaduras eletrônicas está organizado da seguinte maneira como é possível ver na Figura 4.1, onde o fluxo se inicia no Arduino enviando a senha para a aplicação desktop.

Figura 4.1: Diagrama do funcionamento do sistema.



No passo (1) após o usuário digitar sua senha pessoal no módulo de teclado ligado ao Arduino, essa senha é enviada para a aplicação desenvolvida em Python responsável pela comunicação entre o Arduino e o banco de dados. Em seguida (2) os dados são enviados para o banco de dados para que seja verificado se a senha digitada consta no banco de dados e caso exista, é feito um registro do usuário associado com a senha digitada, a sala que foi liberada e o horário da liberação. O banco de dados (3) envia a resposta para a aplicação em Python que envia um sinal (4) para o Arduino dependendo da resposta do banco de dados. Caso tenha encontrado um registro, é enviado um sinal para liberar a porta que é recebido pelo Arduino e enviado (5) para o relé ligado a fechadura.

A aplicação *web* se conecta diretamente com o banco de dados, fornecendo uma interface visual para o usuário ver os relatórios criados pela aplicação além de poder criar seu perfil no

sistema com uma senha para a porta gerada automaticamente.

4.2 Arduino

Nesta parte será descrita a implementação do *Arduino* tanto da parte de software quanto da parte de hardware mostrando como está configurado os pinos e módulos utilizados.

4.2.1 Hardware

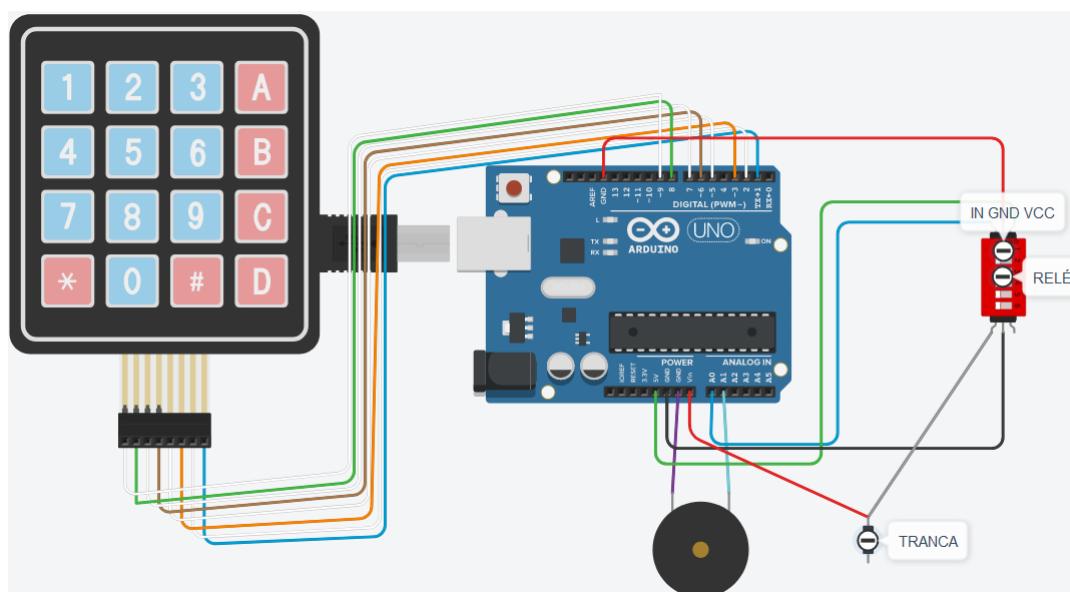
Uma das facilidades do *Arduino* é o fato de não ser necessário nenhuma montagem de circuitos. A ligação dos módulos com a placa do *Arduino* são feitas através dos seus pinos digitais e analógicos por cabos chamados de *jumper*.

O *Arduino* com os módulos e a fechadura eletrônica teve em média um custo total de 180 reais, preço consultado em 4 de dezembro de 2019. Porém antes de iniciar o projeto, a faculdade já possuía esta infraestrutura, sendo então reaproveitada. As fechaduras eletrônicas no mercado possuem uma alta variação no valor, podendo chegar até a casa de 2 mil reais. Uma das vantagens de ter se utilizado o *Arduino* se deve ao fato de poder autenticar a senha no banco de dados, suportando o armazenamento de diversas senhas, além de ter o controle total sobre a fechadura.

O *Arduino* utilizado se encontra ligado ao roteador via cabo *Ethernet* para conseguir se comunicar com o servidor que se encontra na mesma rede local. Sua alimentação é feita por uma fonte externa de 5V conectada na entrada de energia do próprio *Arduino*. Para a programação da placa é utilizada sua entrada Serial conectado diretamente com o computador que irá enviar o código para a placa através da *Arduino IDE*.

Foi utilizado um *shield Ethernet* para habilitar o envio de dados via cabo *Ethernet* para não utilizar a entrada serial e foi utilizado os seguintes módulos no *Arduino*: Um teclado de membrana com 4 linhas e 4 colunas conectado nos 8 pinos digitais do 9 até o 1, pulando o pino 4 por ser reservado para o cartão SD do *shield*; o módulo relé conectado em 2 entradas GND além da entrada 5V e Vin para alimentação e no pino analógico A0 para controle; o módulo *buzzer* foi ligado na entrada GND e no pino analógico A1.

A escolha dos pinos é utilizada na programação do software para indicar o *Arduino* quais pinos controlar. Na Figura 4.2 é possível ver o diagrama com esquema de ligação dos pinos enquanto na Figura 4.3 é possível ver o teclado utilizado para digitar a senha no lado exterior da sala além do *Arduino* acoplado com o *shield* ligado a parede no lado interior da sala.

Figura 4.2: Diagrama das ligações do Arduino¹.Figura 4.3: Teclado utilizado e o Arduino instalado com o *shield* e módulos ligados a ele.

4.2.2 Software

O software instalado no Arduino é responsável por receber os dados do módulo de teclado e enviá-los para a aplicação em Python, aguardando então sua resposta para em caso positivo, ativar o relé conectado a fechadura.

O código Arduino é separado em diversos blocos. O primeiro bloco é onde é declarado as bibliotecas utilizadas. O segundo bloco é responsável pela declaração das variáveis. O terceiro e quarto blocos fazem parte da *Arduino Programming Language*, sendo o método *Setup* que é chamada assim que o Arduino é executado, servindo para inicializar as variáveis, pinos e outras configurações que precisam serem executados apenas uma vez. O quarto bloco é o *Loop* que irá repetir o código que se encontra dentro da função, que é onde se encontra o código principal do Arduino. O quinto bloco foi utilizado para a declaração das funções. O Código completo se encontra no ANEXO A.

¹Ilustração elaborada no Thinkercad

Para a comunicação com o teclado foi utilizado a biblioteca *Keypad* que já se encontra dentro da IDE, sendo responsável por prover métodos que facilitam a comunicação com um teclado baseado em matriz. É necessário criar uma matriz mapeando os botões do teclado além dos pinos utilizados pelo teclado como pode ser visto no trecho de código 4.1.

Trecho de código 4.1: Matriz do teclado

```
// Configuração de teclado
const byte ROWS = 4;
const byte COLS = 4;

5 char hexaKeys[ROWS][COLS] = {
    { '1', '2', '3', 'A' },
    { '4', '5', '6', 'B' },
    { '7', '8', '9', 'C' },
    { '*', '0', '#', 'D' }
10 };

// Ordem dos pinos
byte rowPins[ROWS] = { 9, 8, 7, 6 };
byte colPins[COLS] = { 5, 3, 2, 1 };

15 Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
    COLS);
```

Para a conexão *Ethernet* foi utilizada a biblioteca *Ethernet* que também é inclusa na Arduino IDE. No bloco *Setup* é feito a configuração dos pinos além da conexão com o *socket* da aplicação em Python utilizando os métodos e classes da biblioteca *Ethernet*. O bloco *Loop* fica aguardando o recebimento de uma tecla e adicione ela em um vetor onde será armazenado a senha. Após pressionada a tecla asterisco, o vetor é enviado para a aplicação em Python e o Arduino e após recebido a resposta, o código entra no bloco onde será feita a leitura da resposta. Dependendo da resposta recebida, é enviado ou não um sinal para o relé para ser efetuado a liberação da porta.

Caso ocorra um erro de conexão, o Arduino entra em modo *offline* onde possui uma senha que pode ser utilizada somente quando não há conexão enquanto tenta se reconectar a cada segundo.

4.3 Aplicação web

Nessa parte será descrito a aplicação *web* assim como a configuração e organização do banco de dados utilizado para armazenar os dados gerados pelo Arduino além das credenciais de acesso do sistema.

4.3.1 Disponibilização dos dados

Antes do armazenamento e recebimento dos dados, o banco de dados deverá ser criado e configurado através dos seguintes passos principais:

- Criação do banco de dados;
- Configuração dos outros usuários e permissões;
- Criação das tabelas;

A criação das tabelas é feita pelo Django através de seu ORM utilizando os modelos referentes as tabelas do banco de dados, como visto na Subseção 2.2.1. A criação do banco de dados é feita pelos comandos *makemigrations* e *migrate* do arquivo *manage.py* do Django.

A figura 4.4 mostra um diagrama de como está organizado o banco de dados onde podem ser vistas as tabelas que serão criadas e suas relações. O banco de dados criado possui três tabelas: Usuário, Registro e Acesso. A seguir será descrito em detalhes cada tabela do sistema.

A Tabela usuário é criada pelo próprio Django e serve para armazenar as contas de usuário que irão acessar o sistema, incluindo a conta de administrador. A tabela está ligada a Acesso, onde cada usuário possui um Acesso além de também estar ligada a tabela de Registro, associando o usuário a um registro. Na Tabela 4.1 é mostrado os campos da tabela equivalente ao *auth_user* pois a biblioteca *Auth* do Django possui várias outras tabelas que não entram no escopo deste trabalho.

Figura 4.4: Diagrama do banco de dados.

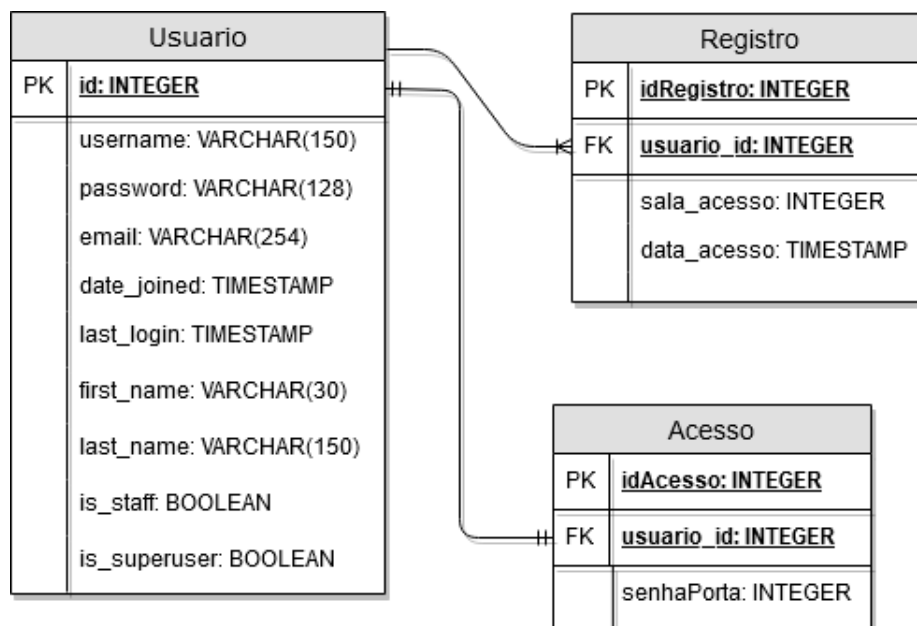


Tabela 4.1: Campos da tabela Usuário

Campo	Tipo de Dado	Descrição
id	INTEGER	Armazena o identificador único de usuário
username	VARCHAR(150)	Armazena o <i>username</i> utilizado para acessar o sistema
password	VARCHAR(128)	Armazena a <i>hash</i> SHA256 da senha do usuário
email	VARCHAR(254)	Armazena o email do usuário
date_joined	TIMESTAMP	Armazena a data de quando o usuário foi criado
last_login	TIMESTAMP	Armazena a data do último acesso do usuário
first_name	VARCHAR(30)	Armazena o primeiro nome do usuário
last_name	VARCHAR(150)	Armazena o sobrenome do usuário
is_staff	BOOLEAN	Armazena a variável dizendo se o usuário pertence a staff e possui acesso ao painel de <i>admin</i> .
is_superuser	BOOLEAN	Armazena a variável dizendo se o usuário é um super usuário.

A tabela Acesso é responsável por manter as senhas de acesso para a porta e é associada a um usuário. Cada usuário possui um acesso que é criado automaticamente pela página da aplicação web. A senha é gerada aleatoriamente para o usuário por motivos de segurança, pois pelo fato de ser uma senha única, ela poderia ser descoberta na etapa de criação do usuário, recebendo um alerta de senha já existente, o que significa que já existe essa senha de acesso, mas pertencendo a outro usuário. Foi escolhida uma coluna de senha única pois cada usuário

tem seu próprio acesso, não havendo o compartilhamento das senhas. Na tabela 4.2 é mostrado os campos da tabela.

Tabela 4.2: Campos da tabela Acesso

Campo	Tipo de Dado	Descrição
idAcesso	INTEGER	Armazena o identificador único de acesso.
usuario_id	INTEGER	Armazena a chave estrangeira do usuário associado ao acesso.
senhaPorta	INTEGER	Armazena a senha de acesso utilizada para liberar a porta.

A tabela Registro é responsável por manter um registro do acesso realizado as salas a fim de se obter o controle de acesso, associando cada registro ao usuário que liberou a fechadura. É armazenado o horário do registro assim como a sala acessada e quem acessou ela. Na tabela 4.3 é mostrado os campos da tabela.

Tabela 4.3: Campos da tabela Registro

Campo	Tipo de Dado	Descrição
idRegistro	INTEGER	Armazena o identificador único do registro.
usuario_id	INTEGER	Armazena a chave estrangeira do usuário associado ao registro criado.
sala_acesso	VARCHAR(20)	Armazena a sala onde foi feito o acesso.
data_acesso	TIMESTAMP	Armazena a data e a hora do acesso a sala.

O banco de dados opera no mesmo servidor que a aplicação *web* e os dados armazenados são centralizados, pois caso seja implementado múltiplas placas Arduino em outras salas da instituição, todos irão utilizar o mesmo banco de dados.

Seu uso na aplicação em Python consiste na execução de requisições ao banco de dados na hora de validar se o acesso digitado pelo usuário existe na base de dados além de inserir um registro com os dados do usuário e o horário que foi realizado o acesso a porta enquanto na aplicação *web*, o principal uso do banco de dados é para exibir os registros de acesso além do cadastro de usuários e senhas de acesso para as portas.

4.3.2 Desenvolvimento do sistema *web*

O sistema *web* foi desenvolvido utilizando o *framework* Django para facilitar o desenvolvimento do site responsável pelo cadastro de usuários e pela visualização dos registros. A

aplicação foi separada em duas aplicações menores, uma estrutura de organização do próprio Django.

A aplicação principal é a de monitoramento onde se encontram as principais páginas do site, chamadas no Django de *templates* e *views* e são responsáveis pela exibição dos registros e informações sobre o sistema.

A segunda aplicação é a de usuário que é gerada pelo próprio Django contendo sua tabela de usuário (chamada de *AUTH_USER* e é onde se encontram os *templates* e *views* relacionados a *login*, cadastro e *logout*. Os formulários do site são gerados pelo próprio Django com pequenas modificações feitas para a proposta e foi utilizado o pacote *django-crispy-forms*² utilizando o *Bootstrap 4* na configuração para aprimorar o visual de formulários como de cadastro e de *login*.

Através do Django, é possível executar códigos em Python dentro das páginas como consultas a bancos de dados ou estruturas condicionais, como pode ser visto no trecho de código 4.2 utilizado como base de todas as páginas em que os botões "Ver acesso" e registrar só podem ser acessados caso o usuário esteja autenticado no sistema e possua a permissão de super usuário pois se trata de páginas que não deve ser acessada por todo mundo. O código por completo pode ser visto no ANEXO B.

Trecho de código 4.2: Estrutura condicional utilizada para permissionamento

```
{% if user.is_authenticated %}
    {% if request.user.is_superuser %}
        <a class="nav-item nav-link" href="{% url 'site-acesso' %}">Ver
acesso </a>
        <a class="nav-item nav-link" href="{% url 'register' %}">Registrar
</a>
    {% endif %}
5   <a class="nav-item nav-link" href="{% url 'logout' %}">Sair </a>
{% else %}
```

Um exemplo de *view* criada no Django pode ser visto no trecho de código 4.3 onde é possível ver a classe responsável por retornar os registros do banco de dados para o *template* "home.html" além de ordená-los pela data. O arquivo *views.py* completo do aplicativo de monitoramento pode ser visto no ANEXO F

Trecho de código 4.3: Método da *view* responsável por obter os registros de acesso

```
class RegistroListView(LoginRequiredMixin, PermissionRequiredMixin,
    ListView):
```

²<https://github.com/django-crispy-forms/django-crispy-forms>

```

permission_required = "auth.change_user"
model = Registro
template_name = 'monitoramento/home.html'
5 context_object_name = 'registros'
paginate_by = 10

#Método para busca por sala
def get_queryset(self):
10     try:
        query = self.request.GET.get('q')
    except:
        query = ''
    if query:
15         object_list = Registro.objects.filter(sala_acesso__icontains=
query).order_by('-data_acesso')
    else:
        object_list = Registro.objects.all().order_by('-data_acesso')
    return object_list

```

Assim como o ORM é responsável por realizar as consultas para se obter informações, através da modelagem do *model* ele também é responsável por criar a tabela no banco de dados. A *model* responsável pela tabela Acesso pode ser vista no trecho de código 4.4.

Trecho de código 4.4: *Model do acesso*

```

class Acesso(models.Model):
    usuario_id = models.OneToOneField(User, on_delete=models.CASCADE)
    senhaPorta = models.IntegerField(unique=True, default = gerarAcesso)
5
    def __str__(self):
        return f'{self.usuario_id.username} Acesso'

```

Na Figura 4.5 é possível ver a página de registro também gerada pelo Django junto com os campos contendo validações, como o de senha com checagem de segurança e o de email.

Na página exibida na 4.6 é possível visualizar a senha de acesso gerada para o usuário após seu cadastro. Utilizando as credenciais cadastradas na aplicação web, é possível acessar esta página para visualizar a senha a qualquer momento no caso de esquecimento.

Parte do visual do site foi construído utilizando o *framework web* chamado *Bootstrap* que fornece componentes visuais prontos como botões e tabelas. Uma das vantagens de se utilizar o *Bootstrap* é o fato dele possuir um design responsivo, provendo suporte a *smartphone* e outros

Figura 4.5: Página de registro da aplicação *web* gerada pelo Django.

The screenshot shows a web application interface for 'Django monitoramento'. The header bar contains the site name and navigation links 'Home' and 'Sobre' on the left, and 'Ver acesso', 'Registrar', and 'Sair' on the right. The main content area is a registration form with the following fields and instructions:

- Usuário***: A text input field. Below it, a note states: 'Obrigatório. 150 caracteres ou menos. Letras, números e @/./+/-/_ apenas.'
- Email***: A text input field.
- First name***: A text input field.
- Last name***: A text input field.
- Senha***: A text input field. Below it, a list of password requirements:
 - Sua senha não pode ser tão parecida com suas outras informações pessoais.
 - Sua senha precisa conter pelo menos 8 caracteres.
 - Sua senha não pode ser uma senha habitualmente utilizada.
 - Sua senha não pode ser inteiramente numérica.
- Confirmação de senha***: A text input field. Below it, a note states: 'Informe a mesma senha informada anteriormente, para verificação.'

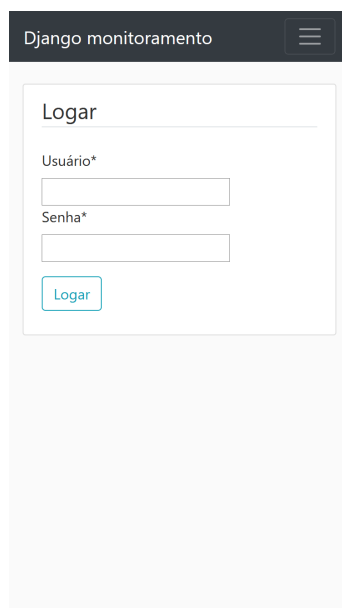
A blue 'Cadastrar' button is located at the bottom of the form.

Figura 4.6: Página mostrando o acesso gerado para o usuário cadastrado.

The screenshot shows the same 'Django monitoramento' header. The main content area displays the message: 'Sua senha de acesso: 159753'.

dispositivos além do computador como pode ser visto na Figura 4.7 mostrando como o site se comporta em um dispositivo móvel. O formulário de login é gerado pelo próprio Django para ser utilizado no template. Caso um usuário tente acessar o sistema sem estar autenticado, foi configurado para que ele seja redirecionado para a página de login.

Figura 4.7: Exemplo de página responsivo em um smartphone.



4.3.3 Configuração do servidor

A aplicação Django é executada em um servidor Apache incluída no pacote Windows, Apache, MySQL, and PHP (WAMP). Foi escolhido o WAMP pelo fato dele ser específico para o Windows que foi o sistema operacional utilizado durante todo o desenvolvimento. As aplicações Python utilizam o padrão Web Server Gateway Interface (WSGI) desenvolvido para que *frameworks* sejam compatíveis com diversos servidores disponíveis (KUBICA, 2010). O próprio Django gera um arquivo WSGI porém foi necessário criar outro pois o gerado por ele é feito para servidores Linux. Além disso, as variáveis de ambiente responsáveis por guardar as senhas de modo a proteger elas são armazenadas no arquivo WSGI *wsgi_windows.py* que pode ser visto no ANEXO C.

Foi utilizada o pacote *mod_wsgi*³ que inclui um módulo para executar uma aplicação Python em um servidor apache além de ser necessária para finalizar a configuração dos arquivos do *Apache*.

No arquivo *httpd_vhosts.conf* do *Apache* se encontra a localização de diversos arquivos da aplicação como o WSGI e a pasta *"/static/"*. Além disso o arquivo também possui algumas configurações de permissionamento, nome dos arquivos de *log* do servidor e a porta utilizada.

³https://github.com/GrahamDumpleton/mod_wsgi

4.4 Aplicação *desktop* em Python

Nesta parte será descrita a implementação da aplicação Python responsável pela comunicação entre a placa Arduino e o banco de dados.

4.4.1 Comunicação com o Arduino

A comunicação com o Arduino é realizada via cabo Ethernet utilizando-se do *shield* Ethernet e da biblioteca *socket*⁴ já inclusa no Python. A funcionalidade se baseia no envio de mensagens de resposta com um único caractere dependendo se a resposta foi de sucesso ou de falha. O código completo do programa contendo tanto a comunicação com o Arduino quanto com o banco de dados pode ser vista no ANEXO D.

O método "logar()" visto no trecho de código 4.5 demonstra um bloco de código *if* onde caso a senha digitada seja encontrada no banco de dados, é enviado um caractere "S" para o Arduino que irá ativar no Arduino o bloco de código responsável pela abertura da porta através do envio de um sinal *LOW* para o pino do relé. Após isso o método irá enviar o registro do acesso realizado para o banco de dados através do método *inserirRegistro()*.

Caso não seja encontrado nenhum registro contendo a senha digitada, será enviado um caractere "F" para o Arduino para sinalizar uma falha e a aplicação irá retornar sua execução normalmente sem liberar a porta.

Trecho de código 4.5: Método utilizado para a verificação de senha

```
def logar(senha):  
    usr = UsuarioDao()  
    #result = usr.logarUsuario(criptografarSenha(senha))  
    result = usr.logarUsuario(senha)  
  
    try:  
        if result is not None:  
            arduino.write('S'.encode())  
            print("Sucesso")  
            #Inserir no registro o usuário que possui a senha digitada  
            inserirRegistro(result)  
  
        else:  
            #print("falha: {}".format(result[0]))  
            arduino.write('F'.encode())
```

⁴<https://github.com/python/cpython/blob/master/Lib/socket.py>

```
        print("Falha")
    except Exception as err:
        print("Ocorreu um erro no envio da mensagem", err)
```

O método principal inicializa um *socket* com um IP fixo e aguarda a conexão do Arduino para então começar a comunicação em uma estrutura de repetição *while*. Caso a conexão seja perdida, ele entra em um bloco para se reconectar ao Arduino. Por limitações do método de *recv()* de recebimento de dados, foi utilizado um *timeout* pois este método suspende o sistema e pode não reconhecer caso haja uma falha na conexão. Além disso, está sendo utilizado o TCP para garantir que os dados não sejam perdidos no caminho, pois o TCP aguarda um sinal de confirmação de recebimento além de tentar várias vezes enviar os dados. Ele então inicia novamente o bloco esperando por dados, sempre enviando 1 byte para checar se o *socket* ainda está funcionando, jogando uma exceção caso contrário.

Portanto colocados vários blocos *try catch* para caso ocorra qualquer erro na aplicação ou com a comunicação com o Arduino, a aplicação conseguir se recuperar e continuar rodando sem interrupção.

4.4.2 Comunicação com o banco de dados

A principal tarefa da aplicação Python é por fazer uma intermediação entre o Arduino e o banco de dados, contendo uma classe Data Access Object (DAO) responsável por fazer esta conexão. Para a comunicação com o banco de dados foi utilizado a biblioteca *Psycopg2*⁵

A comunicação da aplicação Python com o banco de dados foi feita sem utilizar ORM, tendo então as consultas SQL de forma manual. O objetivo principal é permitir com que o Arduino consiga autenticar as senhas no banco de dados e que seja realizado um controle de acesso.

O trecho de código 4.6 demonstra o método responsável por checar se a senha digitada pertence a um usuário cadastrado, executando uma consulta SQL para realizar uma consulta de modo a descobrir se a senha digitada consta no banco de dados além de associar a chave estrangeira "usuario_id" da tabela acesso com o "id" do usuário dono da senha digitada com o objetivo de guardar este usuário em uma variável que será posteriormente enviado para a tabela registro. O código completo se encontra no ANEXO E.

A proposta inicial foi de se utilizar uma conexão via cabo serial, o que acabou gerando a arquitetura inicial do sistema que foi levemente alterada quando tornou-se possível a utilização

⁵<https://github.com/psycopg/psycopg2>

de um *shield Ethernet*. Foi também cogitado a utilização de um Raspberry Pi para executar a aplicação em Python, processando os dados recebidos do Arduino e armazená-los no banco de dados, não sendo mais necessário devido a possibilidade de múltiplas conexões remotas em um dispositivo ao contrário da limitação do cabo serial.

Trecho de código 4.6: Método utilizado para checar se existe um usuário com a senha digitada

```
def logarUsuario(self, senha):  
    try:  
        con = ConnectionFactory.conectar()  
        cursor = con.cursor()  
5        cursor.execute("""SELECT b.*  
                        FROM AUTH_USER AS b  
                        INNER JOIN  
                        (  
10                            SELECT "senhaPorta", usuario_id_id  
                            FROM users_acesso  
                            AS a  
                        )AS a  
                        ON b.id=a.usuario_id_id  
                        WHERE a."senhaPorta" = '%s'""" , (senha,))  
15  
    except (Exception, psycopg2.Error) as error:  
        print("Falha ao obter o registro: {}".format(error))  
    else:  
        return cursor.fetchone()  
20    finally:  
        con.close()  
        cursor.close()
```

5 *Experimentos e resultados*

Para testar a aplicabilidade do sistema foi realizado um experimento no período de 14 dias, que ocorreu no laboratório da FAETERJ/Petrópolis dos dias 24 de outubro de 2019 até o dia 7 de novembro 2019. O teste visou realizar um uso prolongado do sistema para checar a estabilidade do mesmo. Nesse período foi realizado a coleta dos dados dos alunos que utilizam a sala para estudo ou projetos, tendo seus dados cadastrados no sistema para que ganhem uma senha de acesso personalizada e que seja possível identificar quando foi feito um acesso a sala.

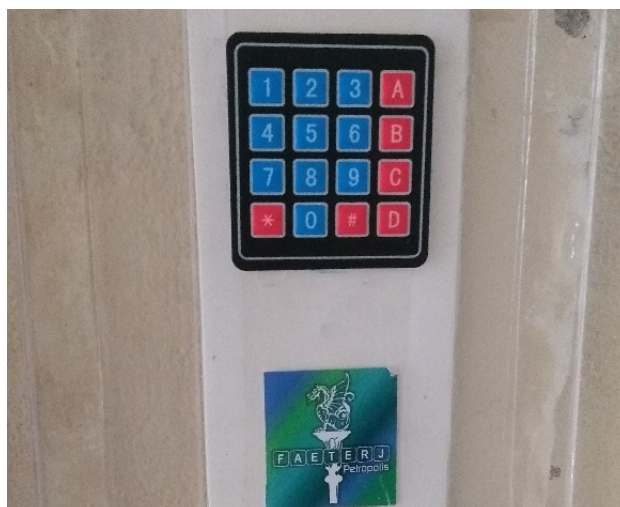
O sistema conseguiu se mostrar funcional, ocorrendo apenas um problema nesse período de testes, que nesse caso foi a reinicialização automática do computador, devido a uma atualização do sistema operacional Windows 10. Isso ocorreu pelo fato do computador não ser um servidor dedicado e sim um computador pessoal, sendo necessário no futuro transferir o servidor para o Linux. Para esses casos foi criado uma senha offline de emergência no Arduino que se mostrou funcional, permitindo a entrada na sala a fim de reiniciar o servidor. Além disso, a chave ainda funciona para abrir a porta caso haja uma queda de energia.

Foi criado também um *script* para executar as aplicações assim que o computador é ligado, porém o mesmo não funcionou devido a um problema no roteador utilizado na sala, sendo necessário reiniciá-lo manualmente. Houve outra pausa no experimento, porém foi apenas para a realização da coleta de outros dados e execução de testes.

O experimento foi feito utilizando uma placa Arduino que pode ser visto na Figura 4.3 conectado ao roteador da sala e utilizando a configuração e módulos descritos na seção 4.2.1. Na Figura 5.1 é possível ver o teclado utilizado para se digitar as senhas já acoplado a parede do laboratório.

Foi utilizado como servidor um dos computadores do laboratório para conseguir executar a aplicação *web*, o banco de dados e a aplicação Python, responsável por realizar a comunicação com o Arduino através de um *socket*. Na Figura 5.2 é possível ver a página da aplicação *web* responsável por exibir o registro de acessos preenchida com os dados coletados durante o período de testes, contendo o nome da pessoa, a sala que foi acessada e data/hora do acesso além

Figura 5.1: Módulo de teclado acoplado a parede.



de um formulário para se filtrar os resultados por nome.

Figura 5.2: Página inicial da aplicação contendo os dados coletados.

Monitoramento			Home	Sobre	Ver acesso	Registrar	Sair
Search		Filtrar					
Nome	Sala	Hora					
Alberto Angonese	Laboratório	12:39 08/11/2019					
Gabriel Aguiar	Laboratório	12:04 08/11/2019					
Matheus Cirino Soares	Laboratório	11:22 08/11/2019					
Gabriel Aguiar	Laboratório	11:04 08/11/2019					
Alberto Angonese	Laboratório	11:03 08/11/2019					
Alberto Angonese	Laboratório	7:58 08/11/2019					
Matheus Cirino Soares	Laboratório	13:18 07/11/2019					
Gabriel Aguiar	Laboratório	12:55 07/11/2019					
Matheus Cirino Soares	Laboratório	12:54 07/11/2019					
Matheus Cirino Soares	Laboratório	12:54 07/11/2019					
1	2	3	Proxima	Ultima			

Foi realizado alguns relatórios com os dados coletados pois é possível de se obter outras informações a partir desses dados. Na Tabela 5.1 é possível ver os 5 horários com mais acessos, sendo às 19h onde ocorreu mais acessos na semana.

Tabela 5.1: As horas com mais acessos.

	Hora acessada	Quantidade de acessos
1	14h	10
2	13h	9
3	21h	8
4	19h	7
5	15h	6

Na Tabela 5.2 é possível ver em quais dias ocorreram mais acessos ao laboratório, tendo o dia 28 de outubro 13 acessos e sendo o dia com mais acessos durante o período de testes.

Tabela 5.2: Os dias com mais acessos.

	Dia acessado	Quantidade de acessos
1	28/10/2019	13
2	24/10/2019	10
3	01/11/2019	9
4	31/10/2019	8
5	08/11/2019	6

Também foi possível identificar através dos dados coletados horários pouco comuns ou inusitados, nesse caso quando e por quem foi realizado o acesso no dia e hora mais cedo e mais tarde, como pode ser visto na Tabela 5.3 que demonstra o acesso mais cedo durante todo o período de teste e na Tabela 5.4 que demonstra o acesso mais tardio. Esse tipo de informação pode ajudar a identificar anomalias ou inconsistências de horários, servindo no futuro para um sistema de notificações de acesso ou até mesmo um controle simples de ponto, podendo gerar relatórios por usuário.

Tabela 5.3: O horario onde ocorreu o acesso mais cedo na semana.

Nome	Sobrenome	Horário do acesso
Alberto	Angonese	09:48 - 28/10/2019

Tabela 5.4: O horário onde ocorreu o acesso mais tardio na semana.

Nome	Sobrenome	Horário do acesso
Alberto	Angonese	22:02 - 24/10/2019

6 *Conclusão e trabalhos futuros*

Neste trabalho foi proposto e implementado um sistema de controle de acesso utilizando uma placa Arduino, em conjunto com uma aplicação web e uma *desktop* ambas feitas em Python, para monitorar os acessos do laboratório da FAETERJ/Petrópolis e armazená-los em um banco de dados.

Foi possível comprovar a eficácia do sistema apresentado durante a etapa de experimentos. A abordagem também se mostrou eficiente visto que o sistema funcionou apresentando somente uma falha relacionada ao computador utilizado como servidor e o roteador disponível no laboratório. Portanto a abordagem conseguiu cumprir sua proposta, sendo possível realizar o controle de acesso utilizando o Arduino instalado na instituição, que antes apenas executava a liberação com as senhas armazenadas no próprio código sem registro algum de acesso além de ser necessário o compartilhamento das senhas pois era único por sala.

A implementação teve um baixo custo, totalizando cerca de R\$180 porém os recursos necessários para a criação do sistema já se encontravam na instituição por conta do projeto das salas. Assim como o Arduino, grande parte as bibliotecas e *software* utilizados são *open source*. Pelos fatos dos *software* utilizados serem multiplataformas, pode-se migrar no futuro para o sistema operacional Linux. Durante a etapa de pesquisa foi possível ver diversas outras aplicações para o Arduino assim como diferentes tipos de arquitetura que podem ser utilizados para a IoT, mostrando as diversas possibilidades de projetos que podem ser construídos e que a IoT é um tema que ainda pode ser muito explorado.

Além disso, foi possível construir o sistema sem um conhecimento profundo de eletrônica mostrando que o Arduino consegue ser também uma porta de entrada no assunto. O fato de tanto a plataforma Arduino quanto seu *software* ser de código aberto ajuda também a reduzir o custo de desenvolvimento além de permitir a criação de diversas versões do *hardware* Arduino dependendo do que for necessário para o projeto.

Através dos dados coletados pode-se obter também um controle de ponto simples, entretanto visualizamos que não é possível identificar certos tipos de serviço, como por exemplo o

registro de saída devido a limitação da própria estrutura.

Este trabalho pode servir de base para outras ideias, possuindo também diversos pontos de melhoria como a utilização de um módulo WIFI ou um Arduino ESP8266 eliminando assim a necessidade do cabo *Ethernet*, diminuindo também o espaço ocupado pela placa. Em questões de segurança é possível também implementar o protocolo Secure Sockets Layer (SSL) no conjunto de aplicações WAMP conforme descrito na Seção 4.3.3. Existem outros módulos que podem ser utilizados para controle de acesso, não se limitando ao teclado utilizado no projeto, como por exemplo por impressão digital ou utilizando-se de um cartão RFID.

Existem também várias funcionalidades que foram pensadas para o futuro do projeto, porém não entraram no escopo deste trabalho. Dentre essas funcionalidades se encontra um sistema de notificações que pode avisar via e-mail caso ocorra um acesso em um horário inusitado a uma sala para a identificação de uma tentativa de invasão, o armazenamento de falhas ao digitar a senha e a visualização de relatórios básicos como os vistos no capítulo 5 dentro da aplicação web.

Também foi identificado oportunidades para expandir o projeto para mais salas a utilização das tabelas de grupos do próprio Django para criar permissões de usuário por sala, a centralização da aplicação Python através de da criação de socket com *thread* ou com a biblioteca *select* para que todos os Arduino se comuniquem com o mesmo servidor. Através dessa expansão também podem ser obtidos novas informações como se uma sala foi acessada em um dia ou está ocupada além das salas mais acessadas na semana.

Existe também no futuro a alternativa de utilizar uma comunicação com *web socket* com a proteção HTTPS através da biblioteca Django *Channels*, a fim de mesclar a parte de comunicação com a aplicação web, sendo necessário somente uma aplicação. Os testes foram realizados em uma rede local porém já existe a possibilidade de utilizar o servidor na nuvem, fornecendo um acesso para quem estiver fora da rede e realizando um melhor uso da IoT. Para isso no entanto deve-se implementar uma criptografia leve no Arduino para esconder o envio da senha, visto que durante a implementação não houve tempo para pesquisar uma criptografia.

Referências

- ARDUINO.CC. **Arduino Uno**. c2019. Disponível em: <<https://store.arduino.cc/usa/arduino-uno-rev3>>. Acesso em: 20 jul. 2019.
- ARDUINO.CC. **Arduino.cc Board**. c2019. Disponível em: <<https://www.arduino.cc/en/reference/board>>. Acesso em: 04 jun. 2019.
- ARDUINO.CC. **Arduino.cc Ethernet**. c2019. Disponível em: <<https://www.arduino.cc/en/Reference/Ethernet>>. Acesso em: 2 nov. 2019.
- ARDUINO.CC. **Arduino.cc Introduction**. c2019. Disponível em: <<https://www.arduino.cc/en/Guide/Introduction/>>. Acesso em: 04 jun. 2019.
- ARDUINO.CC. **Arduino.cc Memory**. c2019. Disponível em: <<https://www.arduino.cc/en/Tutorial/Memory>>. Acesso em: 04 jun. 2019.
- ASHTON, K. et al. That ‘internet of things’ thing. **RFID journal**, Jun, v. 22, n. 7, p. 97–114, 2009.
- BARRAGÁN, H. Wiring: Prototyping physical interaction design. **Interaction Design Institute, Ivrea, Italy**, 2004.
- BNDES. **Produto 8: Relatório do Plano de Ação**. 2017. Disponível em: <<https://www.bndes.gov.br/wps/wcm/connect/site/269bc780-8cdb-4b9b-a297-53955103d4c5/relatorio-final-plano-de-acao-produto-8-alterado.pdf?MOD=AJPERES&CVID=m0jDUok>>. Acesso em: 20 jul. 2019.
- D’AUSILIO, A. Arduino: A low-cost multipurpose lab equipment. **Behavior research methods**, Springer, v. 44, n. 2, p. 305–313, 2012.
- FERDOUSH, S.; LI, X. Wireless sensor network system design using raspberry pi and arduino for environmental monitoring applications. **Procedia Computer Science**, Elsevier, v. 34, p. 103–110, 2014.
- FERDOUSH, S. M. **A low-cost wireless sensor network system using Raspberry Pi and Arduino for environmental monitoring applications**. [S.l.]: University of North Texas, 2014.
- GUEDES, F. S. Internet das coisas (iot–internet of things)-implementação de um medidor de energia elétrica com conexão a internet. Universidade Federal Fluminense, 2018.
- HOLOVATY, A.; KAPLAN-MOSS, J. **The definitive guide to Django: Web development done right**. [S.l.]: Apress, 2009.
- KELLY, S. D. T.; SURYADEVARA, N. K.; MUKHOPADHYAY, S. C. Towards the implementation of iot for environmental condition monitoring in homes. **IEEE sensors journal**, IEEE, v. 13, n. 10, p. 3846–3853, 2013.

- KOSMATOS, E. A.; TSELIKAS, N. D.; BOUCOUVALAS, A. C. Integrating rfids and smart objects into a unified internet of things architecture. **Advances in Internet of Things**, Scientific Research Publishing, v. 1, n. 01, p. 5, 2011.
- KUBICA, M. **HOWTO Use Python in the web**. 2010.
- MADAKAM, S.; RAMASWAMY, R.; TRIPATHI, S. Internet of things (iot): A literature review. **Journal of Computer and Communications**, Scientific Research Publishing, v. 3, n. 05, p. 164, 2015.
- MANYIKA, J.; CHUI, M.; BISSON, P.; WOETZEL, J.; DOBBS, R.; BUGHIN, J.; AHARON, D. Unlocking the potential of the internet of things. **McKinsey Global Institute**, 2015.
- PORTER, M. E.; HEPPELMANN, J. E. How smart, connected products are transforming companies. **Harvard business review**, v. 93, n. 10, p. 96–114, 2015.
- POSTGRESQL. **PostgreSQL About**. c2019. Disponível em: <<https://www.postgresql.org/about/>>. Acesso em: 04 out. 2019.
- PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software-9**. [S.l.]: McGraw Hill Brasil, 2021.
- SANTOS, B. P.; SILVA, L. A.; CELES, C.; BORGES, J. B.; NETO, B. S. P.; VIEIRA, M. A. M.; VIEIRA, L. F. M.; GOUSSEVSKAIA, O. N.; LOUREIRO, A. Internet das coisas: da teoria à prática. **Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)**, 2016.
- XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. **IEEE Transactions on industrial informatics**, IEEE, v. 10, n. 4, p. 2233–2243, 2014.

ANEXO A – Código implementado na fechadura eletrônica

Trecho de código A.1: ArduinoBanco.ino

```

#include <Keypad.h>
#include <Ethernet.h>

#define LED 13
5 #define RELE A0 // Porta digital 6 PWM
#define BUZZER A1
#define SENHAMESTRE "INSERIR SENHA MASTER AQUI" // Apagar antes do commit

// Definindo as configurações da conexão
10 byte mac[] = { 0xBE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 0, 200 };
byte server[] = { 192, 168, 0, 171 }; // IP do servidor
int tcp_port = 65432;
EthernetClient client;

15 // Configuração de teclado
const byte ROWS = 4;
const byte COLS = 4;

20 char hexaKeys[ROWS][COLS] = {
    { '1', '2', '3', 'A' },
    { '4', '5', '6', 'B' },
    { '7', '8', '9', 'C' },
    { '*', '0', '#', 'D' }
25 };

// Ordem dos pinos
byte rowPins[ROWS] = { 9, 8, 7, 6 };
byte colPins[COLS] = { 5, 3, 2, 1 };

```

```
30 Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
    COLS);

void setup() {
    // Inicializando a conexão com o mac e ip definidos.
35 Ethernet.begin(mac, ip);
    Serial.begin(9600);
    pinMode(LED, OUTPUT);
    pinMode(RELE, OUTPUT);
    pinMode(BUZZER, OUTPUT);
40 digitalWrite(RELE, HIGH); // Para começar com a porta fechada

    // Um segundo para o shield inicializar
    delay(1000);
    // Diminuindo o timeout do client.connect() para que não bloqueie o Arduino
45 client.setConnectionTimeout(100);
    conectar();
}

50 int contador = 0;
    char customKeyArray[10] = {};
    char liberou = 0;

void loop() {
55 // Reiniciar o contador se passar do numero máximo de dígitos.
    if (contador >= 7) {
        contador = 0;
    }

60 // Quando estiver recebendo dados. Entrar apenas caso esteja enviando uma senha
    if (client.available()) {
        Serial.println("Aguardando resposta...");
        char serialListener = client.read();
65 // Liberar caso receber um sinal 'S'
        if (serialListener == '1') {
            // fazer nada. Testar sem else if caso dê erro.
        } else if (liberou) {
            if (serialListener == 'S') {
70 tone(BUZZER, 400, 500);
                digitalWrite(RELE, LOW); // Liberar porta
            }
        }
    }
}
```

```

        delay(1000);
        digitalWrite(RELE, HIGH); //Fechar porta
        liberou = 0; //Impedir de entrar novamente nesse bloco até limpar o array
75      Serial.println("Senha correta");
        delay(2000); //Esperar 2 segundo para descansar.
    } else if (serialListener == 'F'){ //Exibir falha caso receber um sinal 'F'
        Serial.println("Senha áinvlida");
        for(int i=0;i<4;i++){ //SOM DE ERRO QUANDO A SENHA DIGITADA ESTÁ
INCORRETA
80          tone(BUZZER,300,200);
          liberou = 0;
        }
    }
} else {
85    //Pegar a tecla digitada
    char customKey = customKeypad.getKey();
    //Adicionar somente dígitos numéricos para o array
    if (customKey && isdigit(customKey)){
        //Serial.println(customKey);
90      customKeyArray[contador++] = customKey;
        tone(BUZZER, 400, 100);
    } else if (customKey == '*' && contador > 0){ //Somente enviar caso tenha sido digitado
1      numero e apertado o * para finalizar
        //Colocando um terminador de string
        customKeyArray[contador] = '\0';
95      Serial.println("Senha digitada: ");
        Serial.println(customKeyArray);
        client.write(customKeyArray);

        // @@@@ Não revelar a senha mestre.
100     if(!client.connected() && strcmp(customKeyArray, SENHAMESTRE) == 0){
        Serial.println("Utilizando çãliberao no modo offline");
        for(int i=0;i<4;i++){ //SOM DE QUANDO LIBERA NO MODO OFFLINE
            tone(BUZZER,300,500);
            digitalWrite(RELE, LOW); //Liberar porta
105          delay(1000);
            digitalWrite(RELE, HIGH); //Fechar porta
        }

        //Resetando o array e o contador
110      customKeyArray[0] = '\0';
        contador = 0;
        liberou = 1; //Permite entrar no bloco para receber uma resposta
    }
}

```

```
    }  
115  
    // Quando a conexão for perdida  
    if (!client.connected()) {  
        // Serial.println("Conexão perdida. Reconnectando...");  
        conectar();  
120    }  
}  
  
void conectar() {  
    client.stop();  
125    if (client.connect(server, tcp_port)) {  
        Serial.println("Conectado no ConectarUmaVez");  
    }  
}
```

ANEXO B – Template base do site

Trecho de código B.1: base.html

```

<!-- carregando o arquivo css da pasta static (o css)-->
{% load static %}

<!DOCTYPE html>
5 <html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
      shrink-to-fit=no">
10
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
      bootstrap/4.3.1/css/bootstrap.min.css" integrity="
      sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
      crossorigin="anonymous">

    <link rel="stylesheet" type="text/css" href="{% static 'monitoramento/
      main.css' %}">
15
    <!-- Se a pagina tiver um titulo, colocar ele óaps o titulo base. ãSeno,
      apenas utilizar o íttulo base-->
    {% if title %}
      <title>Monitoramento - {{ title }}</title>
    {% else %}
20      <title>Monitoramento</title>
    {% endif %}
  </head>
  <body>
    <header class="site-header">
25      <nav class="navbar navbar-expand-md navbar-dark bg-dark
        fixed-top">

```

```

        <div class="container">
            <a class="navbar-brand mr-4" href="{% url 'inicio-registros
' %}">Monitoramento</a>
            <button class="navbar-toggler" type="button" data-toggle="
collapse" data-target="#navbarToggle" aria-controls="navbarToggle"
aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
30 </button>
            <div class="collapse navbar-collapse" id="navbarToggle">
                <div class="navbar-nav mr-auto">
                    <!-- Pode usar a url direta no href (como /about) ou
usar o nome da ávarivel do arquivo urls.py -->
                    {% if request.user.is_superuser %}
35 <a class="nav-item nav-link" href="{% url '
inicio-registros ' %}">Home</a>
                    {% endif %}
                    <a class="nav-item nav-link" href="{% url '
site-about ' %}">Sobre</a>
                </div>
                <!-- Navbar Right Side -->
40 <div class="navbar-nav">
                <!-- Checando se o usuario esta autenticado com a
funcao do Django para exibir o ãboto certo-->
                {% if user.is_authenticated %}
                    <a class="nav-item nav-link" href="{% url '
site-acesso ' %}">Ver acesso</a>
                    {% if request.user.is_superuser %}
45 <a class="nav-item nav-link" href="{% url '
register ' %}">Registrar</a>
                    {% endif %}
                    <a class="nav-item nav-link" href="{% url '
changePassword ' %}">Alterar Senha</a>
                    <a class="nav-item nav-link" href="{% url 'logout '
%}">Sair</a>
                    {% else %}
50 <a class="nav-item nav-link" href="{% url 'login '
%}">Logar</a>
                    {% endif %}
                </div>
            </div>
        </nav>
55 </header>

```

```

<!-- Criando um bloco chamado content que pode ser sobrescrito por quem
herdar essa pagina -->
<main role="main" class="container">

    {% if messages %}
        {% for message in messages %}
            <!-- Nesse caso ele vai pegar qual alerta o django
enviou e utilizar a tag Bootstrap correta (sucesso, fala, etc) pois o
django funciona bem com o bootstrap -->
            <div class = "alert alert-{{ message.tags }}">
                {{ message }}
            </div>
        {% endfor %}
    {% endif %}
    {% block content %}{% endblock %}

</main>

<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8
abtTE1Pi6jizo" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/
umd/popper.min.js" integrity="
sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1 "
crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/
bootstrap.min.js" integrity="
sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
</body>
</html>

```

ANEXO C – Arquivo WSGI utilizado no Windows

Trecho de código C.1: wsgi_windows.py

```
import os, sys, site
from django.core.exceptions import ImproperlyConfigured

#Adicionar o activate do virtualenv
5 activate_this = 'C:/Users/SirLab/Roberto/tcc-roberto-web/env/Scripts/
  activate_this.py'

exec(open(activate_this).read(), dict(__file__=activate_this))

# Adicionar o site-packages do virtualenv
10 site.addsitedir('C:/Users/SirLab/Roberto/tcc-roberto-web/env/Lib/site -
  packages')

# Adicionar os diretórios da alpicção para o PYTHONPATH
sys.path.append('C:/Users/SirLab/Roberto/tcc-roberto-web/TccRobertoWeb')
sys.path.append('C:/Users/SirLab/Roberto/tcc-roberto-web/TccRobertoWeb/
  monitoramento')
15 sys.path.append('C:/Users/SirLab/Roberto/tcc-roberto-web/TccRobertoWeb/
  users')

# Pegando as variáveis de ambiente e enviando para o Django
os.environ['DJANGO_SETTINGS_MODULE'] = 'TccRobertoWeb.settings'
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'TccRobertoWeb.settings')
20 os.environ['ENV_ROLE'] = 'INSERIR_ROLE'
os.environ['SECRET_KEY'] = 'INSERIR_SECRET_KEY'
os.environ['DB_PASS'] = 'INSERIR_SENHA_DO_BANCO'

from django.core.wsgi import get_wsgi_application
25 application = get_wsgi_application()
```


ANEXO D – Código principal da aplicação Python

Trecho de código D.1: mainEthernet.py

```

import serial , socket , time

from UsuarioDao import UsuarioDao
from Usuario import Usuario

5
"""
    Variaveis
"""

10 IP = "192.168.20.2" #Ip do servidor. significa todos os ips do computador (local e de rede)
PORTA = 65432 #Portas não registradas > 1023
TIMEOUT = 120 #Tempo esperando por dados

"""

15     éMtodos=====
"""

#Inserir na tabela LOG quando a porta foi aberta
def inserirRegistro(result):
20     usr = UsuarioDao()
    usuarioRetorno = Usuario(result[0], result[1], result[4], result[5],
    result[6], result[7])
    usr.inserirLog(usuarioRetorno)

25 #Método para chegar se achou um registro com a senha digitada
def logar(senha):
    usr = UsuarioDao()
    result = usr.logarUsuario(senha)

30
try:

```

```

    if result is not None:
        conn.sendall('S'.encode())
        print("Sucesso")
        #Inserir no registro o usuário que possui a senha digitada
35         inserirRegistro(result)

    else:
        #print("falha: {}".format(result[0]))
        conn.sendall('F'.encode())
40         print("Falha")

except Exception as err:
    print("Ocorreu um erro no envio da mensagem", err)

"""
45 Main =====
"""

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    print("Bind no IP: {} e PORTA: {}".format(IP, PORTA))
    sock.bind((IP, PORTA))
50     print("Servidor ouvindo...")
    sock.listen()

    conn, addr = sock.accept()
    conn.setblocking(False)
55     conn.settimeout(TIMEOUT)
    print("ãConexo de {} aceita".format(addr))

    while(1):
        #Bloco para tratar erro nos dígitos recebidos ao invés de parar o programa
60         if(conn):
            try:
                print("Aguardando dados...")
                conn.send(b'I')
                #decode serve para transformar os bytes em caracteres
                #O comando -2 significa ir apenas até -2 para cortar o "
65                 senha = int(conn.recv(8).decode("UTF-8"))

                #Parar caso não receber mais dados
                if not senha:
70                     print("ãConexo perdida, sem mais dados")
                    conn = 0
                elif senha:
                    #Mostrar a senha para testes

```

```
75         print(senha)
        try:
            logar(senha)
        except OSError as err:
            print("Ocorreu um erro com o Socket", err)
        except Exception as err:
80             print("Ocorreu um erro ao tentar logar: '", err
    )

    except socket.timeout as err:
        print("Sem dados recebidos: ", err)

85    except socket.error as err:
        print("Erro na conexão: ", err)
        conn = 0
    except Exception as err:
        senha = 0
90        print("Ocorreu um erro com dígitos recebidos: '", err)

    else:
        #Tentar reconectar caso a conexão seja perdida
        print("Reconectando ...")
95        try:
            #Reiniciando servidor
            sock.listen()
            conn, addr = sock.accept()
            conn.setblocking(False)
            conn.settimeout(TIMEOUT)
            print("Conectado com sucesso")
            #Caso não encontre, espere 30 segundos para tentar novamente
        except Exception as err:
            print("Conexão perdida", err)
            print("Reconectando em 30 segundos...")
105            time.sleep(30)
```

ANEXO E – Comunicação da aplicação Python com o banco de dados

Trecho de código E.1: UsuarioDao.py

```

from ConnectionFactory import ConnectionFactory
import psycopg2, os
from psycopg2 import Error
from Usuario import Usuario
5
#Classe responsável pelos queries no banco de dados
class UsuarioDao:

    def inserirUsuario(self, novoUsuario):
10
        try:
            con = ConnectionFactory.conectar()
            cursor = con.cursor()
            cursor.execute("INSERT INTO Usuario VALUES (%s, %s, %s)", (
novoUsuario.nome, novoUsuario.usuario, novoUsuario.senha))
            #bloco de exceção
15
        except (Exception, psycopg2.Error) as error:
            print("Falha ao inserir: {}".format(error))
            #bloco que será executado caso tudo ocorra bem
        else:
            con.commit()
20
            #bloco que sempre será executado para fechar a conexão
        finally:
            con.close()
            cursor.close()

25
    def inserirLog(self, usuarioLogin):
        try:
            con = ConnectionFactory.conectar()
            cursor = con.cursor()

```

```
        cursor.execute("INSERT INTO monitoramento_registro (sala_acesso
, usuario_id) VALUES (%s, %s)", (os.environ['SALA'], usuarioLogin.
usuario_id,))
30     except (Exception, psycopg2.Error) as error:
        print("Falha ao inserir o registro: {}".format(error))
    else:
        con.commit()
    finally:
35         con.close()
        cursor.close()

def selecionarUsuario(self, usuarioConsulta):
    try:
40         con = ConnectionFactory.conectar()
        cursor = con.cursor()
        cursor.execute("SELECT * FROM AUTH_USER WHERE senha = '%s'", (
usuarioConsulta.senha,))

    except (Exception, psycopg2.Error) as error:
45         print("Falha ao obter o registro: {}".format(error))
    else:
        #return cursor.fetchall()
        return cursor.fetchone()
    finally:
50         con.close()
        cursor.close()

def logarUsuario(self, senha):
    try:
55         con = ConnectionFactory.conectar()
        cursor = con.cursor()
        #Inner join criado para o postgres devido a sua limitação.
        cursor.execute("""SELECT b.*
                        FROM AUTH_USER AS b
60                        INNER JOIN
                        (
                            SELECT "senhaPorta", usuario_id_id
                            FROM users_acesso
                            AS a
65                        )AS a
                        ON b.id=a.usuario_id_id
                        WHERE a."senhaPorta" = '%s'""", (senha,))
```

```
70     except (Exception, psycopg2.Error) as error:
71         print("Falha ao obter o registro: {}".format(error))
72     else:
73         return cursor.fetchone()
74     finally:
75         con.close()
76         cursor.close()
```

ANEXO F – Arquivo de views da aplicação de monitoramento

Trecho de código F.1: views.py

```

from django.shortcuts import render
from django.views.generic import ListView
from django.contrib.auth.mixins import LoginRequiredMixin,
    PermissionRequiredMixin
from .models import Registro
5 from django.contrib.auth.models import User

class RegistroListView(LoginRequiredMixin, PermissionRequiredMixin,
    ListView):
    permission_required = "auth.change_user"
    model = Registro
10    template_name = 'monitoramento/home.html'
    context_object_name = 'registros'
    paginate_by = 10

    #Método para busca por sala
15    def get_queryset(self):
        try:
            query = self.request.GET.get('q')
        except:
            query = ''
20        if query:
            #Retornando o id dos usuários que possuem os caracteres digitados no nome
            user_list = User.objects.filter(first_name__icontains=query).
            values('id')
            #Retornando do banco os registros onde o atributo usuario está contido no user list
            object_list = Registro.objects.filter(usuario__in=user_list).
            order_by('-data_acesso')
25        else:

```

```
        object_list = Registro.objects.all().order_by('-data_acesso')
        return object_list

30 def about(request):
    return render(request, 'monitoramento/about.html', {'title': 'About'})

def acesso(request):
    return render(request, 'monitoramento/acesso.html', {'title': 'Acesso'
    })
```