Biometric Access Control

Authors

- Gustavo Olegário -> *gustavo-olegario@hotmail.com*
- Johann Westphall -> *johannwestphall@gmail.com*
- Bruno Marques -> *brunomn95@gmail.com*
- Romane Gallier -> *romanegallier@gmail.com*

Motivação

Com a evolução da tecnologia de leitura de biometria, muitas aplicações do cotidiano começaram a aplicar esta técnica para autenticação de usuários. Essa metodologia, além de apresentar várias vantagens sobre tecnologias como cartões RFID, possuem um custo barato e podem ser integradas até mesmo em pequenos dispositivos como os da IoT.

Visto que dispositivos da IoT e similares tem se tornado cada vez mais comuns no dia-a-dia, fica nítido que pode-se implementar a tecnologia da biometria em aplicações como a validação de presença de alunos dentro de uma sala de aula.

Utilizando a biometria como forma de validar a presença de alunos, fica extremamente simples de acompanhar as atividades e comprometimento dos alunos em tempo real. Além disso, professores não precisam ficar sacrificando tempo de aula, ainda que mínimo, para realizar a chamada. Por último, a biometria traz uma segurança de que o aluno realmente está presente e que não é um outro indivíduo tentando burlar o sistema, como seria o caso de uma lista de chamada ou um cartão RFID.

Dessa forma, este trabalho busca integrar o EPOS MOTE III com um leitor biométrico, dentro de um cenário IoT, a fim de validar presença de alunos e dar uma maior comodidade tanto para aluno quanto para professores.

Objetivos

Esse projeto tem como objetivo mostrar a viabilidade do desenvolvimento de uma solução para permitir que a presença na UFSC seja validada com auxílio de um leitor biométrico e da plataforma EPOSMote III.
Objetivos gerais:

Software para comunicação entre EPOSMote III e leitor biométrico.
Software para enviar informações referentes à presença à base de dados.
Software que carrega um conjunto de impressões digitais no leitor biométrico para comparação posterior.

Metodologia

Usaremos o EPOS MOTE III e o leitor de impressão digital FPM10A a fim de possibilitar uma maneira de contabilizar a presença em aula através da impressão digital. Para isso, estabeleceremos a comunicação entre o Mote e o leitor e também entre o Mote e uma base de dados.

Tarefas

1. Desenvolvimento do plano de projeto.
2. Dispositivo de biometria funcionando com o EPOS.
    1. Integrar o leitor biométrico recomendado pelo professor junto com o EPOS.
    2. Comunicar ao usuário sobre o sucesso, ou não, sobre o 'matching' de digitais.
3. Envio de confirmação de presença para base de dados.
    1. Enviar a informação da confirmação da presença de um aluno para a base de que será fornecida.
4. Carregamento de base de dados dentro do leitor.
    1. Através do EPOS, carregar a base de dados, ou ao menos uma parte dela, para dentro do leitor.

Entregáveis

1. Detalhamento do Projeto, estudo contemplando manuais, livros, repositórios e demais materiais pertinentes ao trabalho encontrados na web.
2. Demonstração da viabilidade tecnológica.
3. Dispositivo de biometria integrado com o EPOS.
4. Envio de confirmação da presença da digital para a base de dados.
5. Carregamento de uma parte da base de dados para dentro do leitor.

Cronograma

Task	25/09	02/10	10/10	18/10	05/11	29/11
Task1	D1					
Task2	X	D2				
Task3		X	X	D3		
Task4				X	D4	
Task5					X	D5

How to

No atual momento deste projeto, já se obteve sucesso em capturar digitais, envia-las a uma base de daods e baixa-las utilizando um sensor biométrico, a plataforma do EPOS mote e um módulo WiFi. Esta secção servirá como um passo-a-passo do que deve ser feito para obter o mesmo resultado.

Requisitos

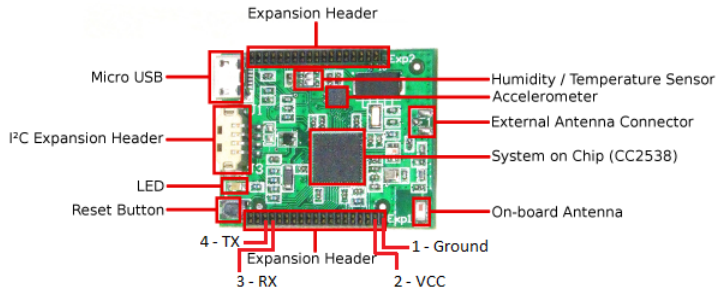
1. O sensor biométrico da Adafruit (o qual pode ser encontrado no próprio LISHA): https://www.adafruit.com/product/751
2. Um módulo Wi-Fi ESP8266 (o qual também pode ser encontrado no LISHA): http://espressif.com/en/products/hardware/esp8266ex/overview
3. Uma fonte adaptada para poder poder suprir a corrente necessária ao módulo Wi-Fi
4. Um módulo FTDI para fazer o deploy do certificado e do código para o ESP8266 (também disponível no LISHA): https://i.stack.imgur.com/21VZq.jpg
5. Para que o sensor funcione dentro da plataforma do EPOS também será necessário do código do sensor desenvolvido para Arduino. Este, por sua vez, pode ser encontrado neste repositório do GitHub: https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library
6. Uma plataforma Mote do EPOS.
7. Uma VM na SETIC ou em qualquer outro servidor que forneça um serviço similar
8. Cabos para conectar as portas entre o sensores e o Mote do EPOS.

Hardware

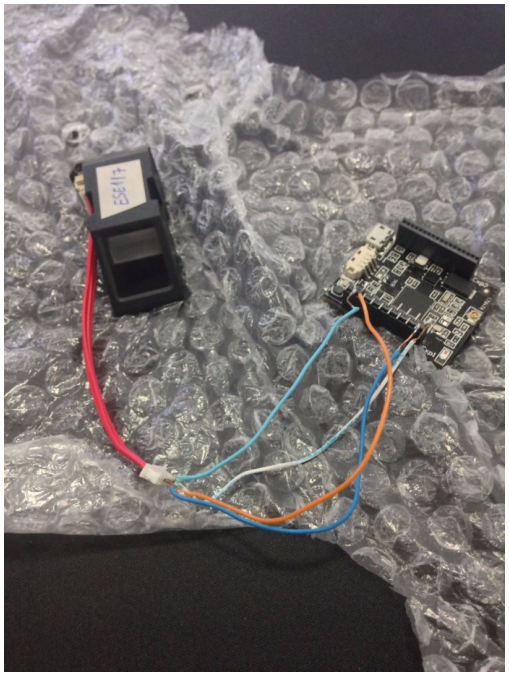
Módulo da biometria

Para fazer com que o sensor se comunique corretamente com o EPOS, deve-se ligá-lo fisicamente ao mote do EPOS. No projeto que foi realizado, utilizou-se um adaptador para as portas físicas do epos e os fios que foram utilizados para conectar o Mote e o Sensor foram soldados diretamente no adaptador, visto que estavam ocorrendo problemas relacionados com mal contato. Todavia, é possível realizar esta conexão sem o adaptador. As portas que serão utilizadas do sensor serão estas:

Deve-se conectar nas respectivas portas do Mote:



No final do processo, o resultado deve ser similar a este:



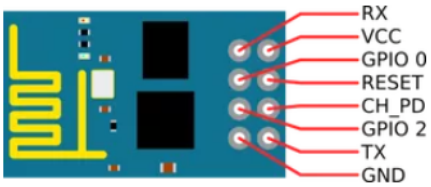
Módulo Wi-Fi ESP8266

Github das bibliotecas do ESP8266

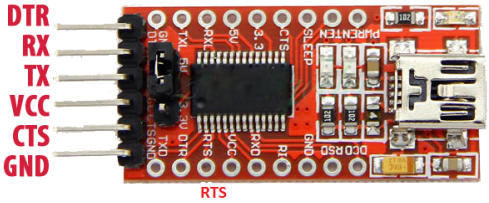
O módulo Wi-Fi é um periférico que já foi utilizado em trabalhos passados dentro da plataforma do EPOS. No escopo deste trabalho, ele será utilizado para enviar requisições HTTP: POST e GET para enviar e receber biometrias, respectivamente. Maiores informações para o módulo acesse: http://epos.lisha.ufsc.br/Wi-Fi+for+EPOSMote+III+with+the+ESP8266+Module .

Porém, algumas questões precisam ser tratadas para que tudo funcione de forma harmoniosa. O módulo ESP8266, por padrão, não vem com seu Firmware configurado, nem com certificado. Então, teremos que fazer o deploy destes de forma manual. Instale a IDE do Arduino em seu Linux e abre esta com as permissões de sudo (isto é necessário, pois precisaremos ter acesso as portas de USB)

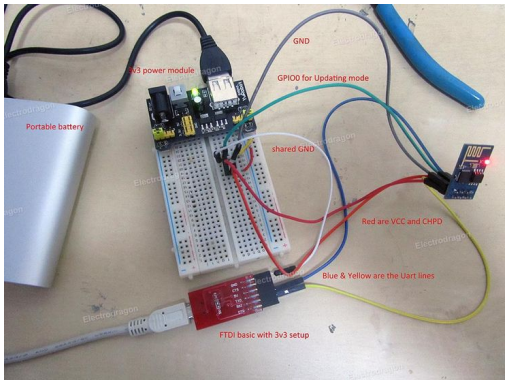
Aproveite o tempo enquanto a instalação é feita para ligar o módulo Wi-Fi com o FTDI. O padrão de pinos do ESP2866 é este:



Enquanto que o padrão de pinos do FTDI é este:



Eles devem ser ligados da seguinte forma:



Agora basta ligar a porta USB do FTDI com o computador.

Com a IDE do Arduino aberta vamos primeiro fazer o deploy do firmware da placa e em seguida o deploy dos certificados necessários. Para podermos fazer o deploy do código na placa, precisamos de um plugin específico:

1. Com a IDE já aberta, vá em File > Preferences.
2. No campo de Additional Boards Manager URLs coloque: http://arduino.esp8266.com/stable/package_esp8266com_index.json e clique em Ok.
3. Agora vá para Tools > Board: "Arduino/Genuino Uno" > Boards Manager
4. Na janela que irá surgir, digite esp8266 e instale o driver na versão 2.3.0
5. Selecione novamente Tools > Board: "Arduino/Genuino Uno" > Generic ESP2866 Module

Utilize o link abaixo para ter acesso ao código da placa:

https://github.com/Olegario96/INE-5424-Sistemas-Operacionais-II/blob/823e4a314605d9e30dfdb03e3a4917fa55b3b510/fpm/espv4/espv4.ino

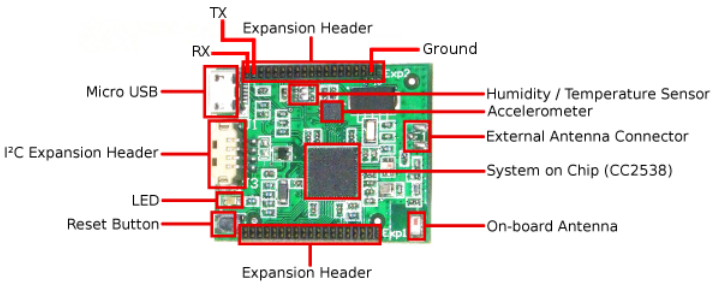
Destacamos aqui principalmente os atributos da classe IoT:

```
static const int SSID_MAX_SIZE = 32; static const int PASS_MAX_SIZE = 32; static const int HOST_MAX_SIZE = 128; static const int ROUTE_MAX_SIZE = 128; static const int MAX_MESSAGE_SIZE = 1024; static const int USER_MAX_SIZE = 64; static const int CONNECT_MAX_TRIES = 3; private: char ssidChar[SSID_MAX_SIZE+1]; String ssid; int ssid_size; char passChar[PASS_MAX_SIZE+1]; String pass; int pass_size; //char host[HOST_MAX_SIZE]; String host; int host_size; //char route[ROUTE_MAX_SIZE]; String route; int route_size; char username_char[USER_MAX_SIZE+1]; String username; int username_size; int port; bool user_connect; File certificate; File key; char * certificateBuffer; char * keyBuffer; enum state_t { CONNECTED, DISCONNECTED, HELLODONE, HELLONOTDONE }; int con_state; int server_con_state; unsigned long int last_time_recorded; unsigned long int last_response_time; HTTPClient http; ESP8266WiFiMulti WiFiMulti;
```

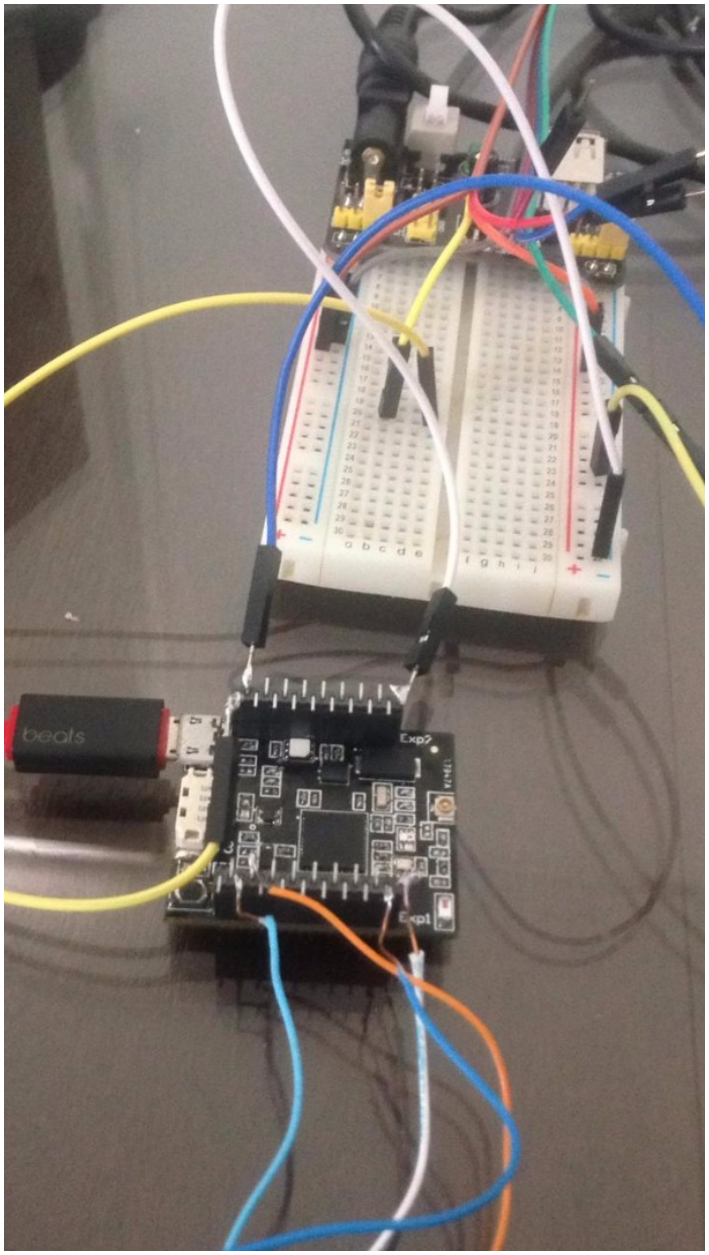
Cole o código do repositório na IDE do Arduino. Com o FTDI ainda desconectado do seu computador, certifique-se que o GPIO0 está conectado no Ground e que o FTDI e o ESP8266 estão com a mesma referência de Ground. Após isso clique no botão de upload, e conecte o USB do FTDI no computador. Caso tudo ocorra bem, o upload será iniciado. Para saber se o deploy obteve sucesso, abra o Serial Monitor e digite o comando AT+++. Caso um OK seja retornado, é porque a placa está funcionando corretamente.

OBS : Após o upload, desconecte o GPIO 0 do Ground para que o código seja gravado permanentemente na flash.

Para fazer o deploy dos certificados, acesse o Makers do ESP2866 ou Tutorial github ESP8266, para maiores informações.
Agora com o código e cerfiticados prontos na placa temos que conectar ela no EPOS. Como um lado do Mote já está sendo usando para conectar com o sensor da biometria, vamos utilizar o outro lado:



Ao final devemos ter algo similar a isto:



Software

A seguir existe um passo a passo do que deve ser alterado no código para que ele fique como foi feito no projeto desenvolvido. Porém, é muito importante que o leitor não se esqueça que ele deve escrever em todos os arquivos que forem utilizados, após realizar os includes:

```
using namespace EPOS;
```

Módulo da Biometria

1. A primeira coias que deve ser feita é adaptar a parte da serial dentro do código. Como o sensor foi desenvolvido para o Arduino, ele utiliza a serial padrão desta plataforma, que é a 'Software Serial'. Sendo assim, a Software Serial, deve ser trocada pelo UART, que é a serial utilizada no EPOS. A numeração das portas e o valor do boudrate devem ser mantidos. Como o UART tem como algumas nuâncias em relação ao Software Serial, mas é filosoficamente o mesmo componente, deve-se também alterar os seguintes métodos: de 'available' para 'ready_to_get', 'read' para 'get' e os 'prints' da Software serial devem ser alterados para o 'put' do UART. Ainda neste passo, é muito importante que quando o UART for instanciando no main, o objeto deve configurar o loopback para false, se não o funcionamento não ocorrerá da forma correta. Não esqueça que o uart deve ser importado.

2. Para que o código cotinue siga o padrão da bilbioteca, deve também importar o header:

alarm.h
, pois é neste arquivo que se localiza o método de delay. Dessa forma, nos trechos onde os delays estão configurados, deve ser feito:

Delay(TEMPO_EM_MICROSEG);

3. Na sequência, deve-se adaptar os couts, para que o log possa ser acompanhado num terminal que esteja rodando um Minicom. Para isso basta fazer o include:

#include <utility/ostream.h>
.E após ter definido o namespace do EPOS, deve ser declarado

OStream cout;

4. Na função

void FPM::writeRaw(uint8_t * data, uint16_t len){ uint16_t written = 0; while (len > packetLen){ writePacket(theAddress, FINGERPRINT_DATAPACKET, packetLen + 2, &data[written]); written += packetLen; len -= packetLen; } writePacket(theAddress, FINGERPRINT_ENDDATAPACKET, len + 2, &data[written]); getReply(); // sensor sends unkonwn data }

Adicionamos "+ 2" ao tamanho do pacote, para o envio de pacotes ao sensor possuir o tamanho correto e adicionamos um getReply ao final do envio a fim de esvaziar o buffer do UART, pois testes apontaram uma resposta desconhecida do sensor, que caso não lida, afetava negativamente a comunicação.

Biblioteca do EPOS para Módulo Wi-Fi ESP2866

Por diversas razões que não cabem no escopo desta página, usaremos uma versão do driver do código do ESP2866 diferente daquela produzida na página do Makers. A estrutura em si, será bem semelhante, mas com mudanças significativas. Como será utilzado o comando GET, que aparentemente não havia sido testado, há a necessidade de adicionar a sequencia '\r\n' para o seu correto funcionamento. Além disso o timeout do sensor não havia sido programado. Dessa forma, sugerimos que o código utilizado na placa seja este:

https://github.com/Olegario96/INE-5424-Sistemas-Operacionais-II/blob/master/fpm/esp8266.cc

Enviando biometria para o banco de dados

Nesta secção será explicado como o grupo desenvolveu um método para exportar as digitais capturadas pelo sensor para uma base de dados. Para isto, será necessário:

- Um servidor previamente configurado
- Uma aplicação que utilize a biometria e o ESP2866 rodando no EPOS

É importante destacar que no atual momento que este trabalho está sendo documentado, todos os alunos

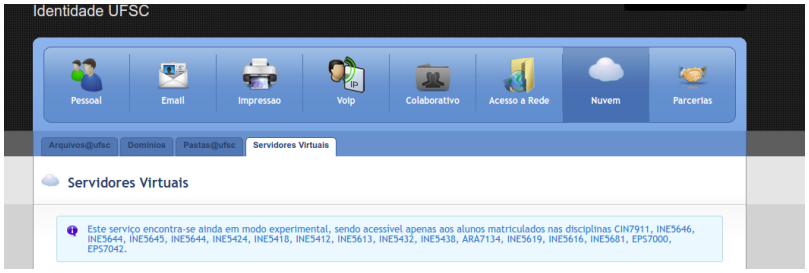
matriculados na disciplina de Sistemas Operacionais II tem direito a uma VM na SETIC. Sendo assim, primeiro será feito o passo a passo de como configurar o servidor e depois será apresentada as mudanças necessárias a serem feitas no main do EPOS.

Criando o servidor

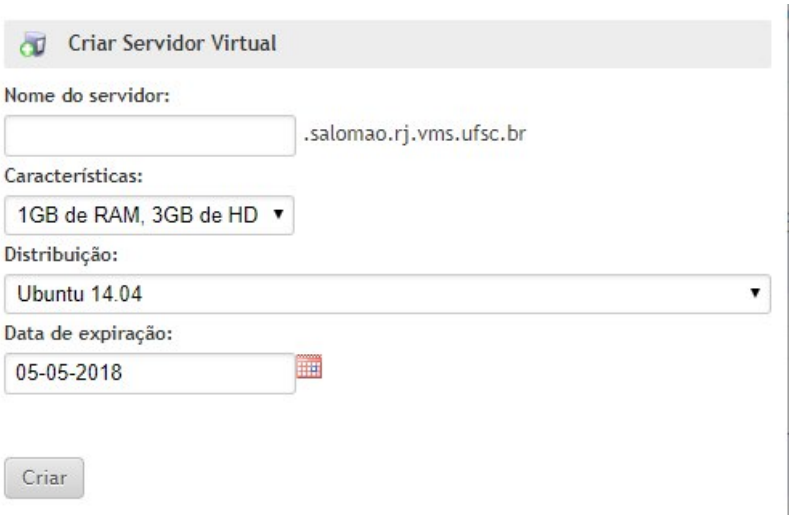
Para criar o servidor, inicialmente, deve-se acessar a seguinte página:

- https://idufsc.ufsc.br/cloud/seafile

Na secção de Nuvem, clicar em Servidores virtuais:



Em seguida, deve-se clicar na opção "Criar servidor". A seguinte janela surgirá:



Um nome deverá ser escolhido para o servidor. Em seguida, basta clicar em criar servidor. Durante todo este tutorial, sempre será destacado onde deve ser colocado o nome do servidor e nome do usuário escolhido pelo leitor.

Você será então redirecionado para a seguinte tela:

Clique no ícone em verde para iniciar o servidor. Caso deseje trocar a senha do servidor, basta clicar no ícone da chave.

Para conectar no servidor, será necessário utilizar uma conexão SSH. Porém, a conexão funcionará somente se estivermos na VPN da UFSC. Caso o leitor já esteja dentro da rede da própria UFSC, automaticamente, já estará na VPN. Caso contrário, será necessário configurar a conexão. Para criar a VPN da UFSC em distribuições Ubuntu, veja este tutorial:

- https://otrs.setic.ufsc.br/otrs/public.pl?Action=PublicFAQZoom;ItemID=1225

Após se certificar que a VPN da UFSC está ativa, abrir um terminal e digitar o seguinte comando:

```
$ssh nome_do_usuario@nome_do_servidor.nome_do_usuario.vms.ufsc.br
```

Se esta for a primeira vez que a conexão estiver sendo feita, o Linux irá perguntar se o usuário deseja adicionar o host a lista de hosts confiáveis. Digite 'yes'. Em seguida, digite a senha do servidor.

Configurando o servidor

A primeira coisa a ser feita é atualizar a máquina. Então, vamos rodar o comando

```
$sudo apt-get update -y && sudo apt-get dist-upgrade -y
```

Caso sua máquina durante atualização exiba o seguinte aviso:

```
perl: warning: Setting locale failed. perl: warning: Please check that your locale settings: LANGUAGE = (unset), LC_ALL = (unset), LC_TIME = "pt_BR.UTF-8", LC_MONETARY = "pt_BR.UTF-8", LC_CTYPE = "pt_BR.UTF-8", LC_ADDRESS = "pt_BR.UTF-8", LC_TELEPHONE = "pt_BR.UTF-8", LC_NAME = "pt_BR.UTF-8", LC_MEASUREMENT = "pt_BR.UTF-8", LC_IDENTIFICATION = "pt_BR.UTF-8", LC_NUMERIC = "pt_BR.UTF-8", LC_PAPER = "pt_BR.UTF-8", LANG = "en_US.UTF-8"
```

Execute os seguintes comandos:

```
$sudo apt-get install locales $sudo locale-gen pt_BR pt_BR.UTF-8 $sudo dpkg-reconfigure locales $sudo apt-get dist-upgrade -y
```

Agora começaremos a installar a nossa stack de ferramentas necessárias. Como a VM fornecida é bem simples, criaremos um sevidor utilizando o Flask (um framework de Python para micro-aplicações web) e o banco de dados será o SQLite em sua terceira versão. A versão do python também será a terceira.

Para instalar o SQLite, rode o comando:

```
$sudo apt-get install sqlite3 -y
```

Iremos também precisar de um editor de texto para fazer algumas tarefas. Os únicos editores disponíveis na distro são o Vi e o Vim. Caso o leitor se sinta confortável usando estas ferramentas, pode usar sem problema algum. Mas neste tutorial usaremos o nano. Vamos instála-lo, então:

```
$sudo apt-get install nano
```

Agora temos que configurar todo o nosso ambiente Python. O Ubuntu fornecido já possui tanto Python2 quanto Python3, porém o padrão da distribuição é a segunda versão. Como queremos que nossa aplicação fique estável, vamos criar um alias para que sempre que o Python seja executado, seja a versão mais atual.

```
$nano .bash_aliases
```

Dentro do arquivo criado escreva:

```
alias python=python3.4
```

E salve o arquivo. Para que a mudança já faça efeito execute:

```
$source .bash_aliases
```

Com o Python3 configurado, agora temos que instalar o gerenciador de pacotes pip.

```
$sudo apt-get update && sudo apt-get -y install python3-pip
```

Com o pip, poderemos criar virtualenvs. Desta forma, podemos instalar pacotes sem permissões de sudo e os pacotes não serão efetivamente instalados, mas sim instalados toda vez que a aplicação for executada dentro do virtualvenv e removidos imediatamente que ela for terminada. Desta forma, ao mesmo tempo que mantemos a máquina "limpa" economizamos espaço livre no disco. Vamos rodar os seguintes comandos para instalar o virtualenv e criar uma pasta para nosso projeto:

```
$sudo pip3 install virtualenv $mkdir biometric $cd biometric
```

Agora iremos criar e ativar nosso virtualenv para nosso projeto:

```
$virtualenv venv $. venv/bin/activate
```

Sempre que desejar fechar a virtualenv, rode o comando:

```
$deactivate
```

Agora vamos criar os arquivos e instalar os pacotes necessários para nossa aplicação:

$touch app.py requirements.txt $pip install Flask==0.10.1 $pip freeze > requirements.txt

Isso já será o suficiente para desenvolvermos o nosso backend, mas ainda temos que configurar as ferramentas de deploy, que neste caso será o Apache. Vamos instalá-lo usando:

$sudo apt-get install apache2 -y $sudo apt-get install libapache2-mod-wsgi -y

Agora vamos criar e editar os arquivos de configurações. Ainda na pasta 'biometric' crie o arquivo 'wsgi.py' e escreva dentro dele:

import sys sys.path.insert(0, "/home/nome_do_usuario/biometric") from app import app as application

Agora vamos editar os arquivos de host:

$sudo nano /etc/hosts

Crie uma nova linha entre a primeira e a segunda linha e digite:

127.0.0.1 nome_do_servidor.nome_do_usuario.vms.ufsc.br

Agora iremos editar os arquivos do Apache. Para isso:

$sudo nano /etc/apache2/sites-available/flask.conf

Dentro deste arquivo, escreva:

<VirtualHost *:80> ServerName nome_do_servidor.nome_do_usuario.vms.ufsc.br WSGIDaemonProcess flaskTest threads=5 WSGIScriptAlias / /home/nome_do_usuario/biometric/wsgi.py <Directory /home/nome_do_usuario/biometric/> WSGIProcessGroup flaskTest WSGIApplicationGroup %{GLOBAL} WSGIScriptReloading On Order deny,allow Allow from all </Directory> </VirtualHost>

Agora precisamos editar outro arquivo do apache:

$sudo nano /etc/apache2/conf-available/servername.conf

Dentro dele escreva:

ServerName nome_do_servidor.nome_do_usuario.vms.ufsc.br

Agora vamos para o diretório do apache e rodar os comandos para poder ativá-lo:

```
$cd /etc/apache2/sites-available $sudo a2enmod wsgi $sudo service apache2 restart $sudo service apache2 reload
```

Pronto! Agora o servidor está completamente configurado. Basta agora desenvolvermos nossa aplicação. Voltemos para dentro da pasta do nosso projeto e vamos editar nosso arquivo de aplicação

```
$cd ~/biometric/ $nano app.py
```

Dentro do arquivo escreva:

```
from flask import Flask, request import json as json import sqlite3 as lite app = Flask(__name__) @app.route('/INE5424', methods=['POST']) def index(): content = request.stream.read() saved = save_fingerprint(content) if saved: return 'Fingerprint saved!' else: return 'This fingerprint is already registred' @app.route('/INE5424/<fingerprint_id>', methods=['GET']) def export_fingerprints(fingerprint_id): if fingerprint_id == str(0): len_fingerprint = total_fingerprint() return str(len_fingerprint) else: fingerprint = export_fingerprint(fingerprint_id) return fingerprint def export_fingerprint(fingerprint_id): conn = lite.connect('./db/biometric.db') cursor = conn.cursor() query_template = "SELECT biometric FROM biometrics b WHERE b.id_biometric = '%s'" query_to_execute = query_template % fingerprint_id try: for row in cursor.execute(query_to_execute): fingerprint = bytes(json.loads(row[0])) break except Exception as e: print(str(e)) fingerprint = 'There is no fingerprint in the DB with this ID' return fingerprint def save_fingerprint(fingerprint): fingerprint_saved = json.dumps(list(fingerprint)) conn = lite.connect('./db/biometric.db') cursor = conn.cursor() query_template = "INSERT INTO biometrics (biometric) VALUES ('%s')" query_to_execute = query_template % fingerprint_saved try: cursor.execute(query_to_execute) saved = True except Exception: saved = False conn.commit() conn.close() return saved def total_fingerprint(): conn = lite.connect('./db/biometric.db') cursor = conn.cursor() query_to_execute = "SELECT COUNT(*) FROM biometrics" for row in cursor.execute(query_to_execute): return row[0] if __name__ == '__main__': app.run(host='0.0.0.0')
```

Nossa aplicação está quase pronta. Falta somente a base de dados. Vamos criar uma pasta específica pro banco e escrever um script em Python para criar nossa base:

```
$mkdir db $nano tables.py
```

Dentro do arquivo escreva:

```
import sqlite3 biometric = """CREATE TABLE IF NOT EXISTS biometrics ( id_biometric INTEGER PRIMARY KEY AUTOINCREMENT, biometric BLOB, CONSTRAINT biometric_unique UNIQUE (biometric) );""" if __name__ == '__main__': conn = sqlite3.connect('./db/biometric.db') cursor =
```

conn.cursor() cursor.execute(biometric) conn.commit() conn.close()

Agora basta executarmos:

$python tables.py $python app.py

E pronto, nosso aplicação está no ar. Porém, lembre-se que toda vez que for necessário mandar uma requisição para o servidor é necessário estar conectado na VPN da UFSC além de na requisição especificar a porta 5000, a qual é a padrão do Flask. Iremos agora adaptar nosso script Python fornecido pelo LISHA.

Aplicação do EPOS

Aqui abaixo temos um exemplo de uma aplicação que utiliza tanto o módulo da biometria quanto o ESP2866. É importante ressaltar, que caso você esteja usando a VM da SETIC para exportar as digitais ou alguma outra rede que utilize VPN, a conexão deve ser obrigatoriamente dentro da rede da UFSC ou na VPN. O grupo desconhece a metodologia de configurar a VPN para o módulo, mas é possível conectar em acess points como o do LISHA ou do LabSec, que por estarem configurados na rede da UFSC, já estão diretamente conectados na VPN.

Na sequência, tem-se o exemplo final que foi usado no trabalho deste grupo. Como prova de funcionamento, a lógica do código funciona da seguinte forma: inicialmente o EPOS instancia tanto o módulo Wi-Fi quanto o da biometria. Em seguida, é feita a conexão com o acess point do LISHA. Na sequência, o EPOS solicita o registro de uma digital no sensor. Essa digital é enviada para o banco de dados através de uma requisição HTTP POST e toda a memória do sensor é apagada. Por último, o EPOS solicita uma digital por vez para o banco utilizando requisições HTTP GET. É necessário solicitar uma biometria de cada vez, pois o EPOS não é robusto o suficiente para trabalhar com muitas biometrias em memória ao mesmo tempo.

int main() { Delay(5000000); mySerial = new UART(0, 9600, 8, 0, 1); wifiSerial = new UART(1, 9600, 8, 0, 1); GPIO rst('B', 3, GPIO::OUT); rst.set(true); wifi = new ESP8266(wifiSerial, &rst); finger = new FPM(); templateSend = new uint8_t[TEMPLATE_SIZE+2]; setup(); Delay(1000000); finger->emptyDatabase(); char host[] = "hammerfall.g.olegario.vms.ufsc.br"; cout << "CONFIGURANDO ENDPOINT" << endl; wifi->config_endpoint(5000, host, sizeof(host) - 1, "/INE5424", 8); cout << "Tentando conectar" << endl; wifi->connect("LISHA", 5, "LISHAPASS", 9); int vezes = 0; while(!wifi->connected()){ vezes++; cout << "Tentando conectar " << vezes << endl; Delay(3000000); wifi->connect("LISHA", 5, "LISHAPASS", 9); } cout << "CONNECTED" << endl; //Capturando biometrias enroll = 1; if(enroll){ finger->getTemplateCount(); while(finger->templateCount < 1){ finger->getTemplateCount(); loopEnroll(); } } else while(true) loopMatch(); //Enviando biometrias para base de dados sendAllTemplates(); finger->emptyDatabase(); Delay(5000000); //Processo de baixar as biometrias int size = 0, numBiometrics; char numBiometricsChar[7], biometricChar[7]; size = wifi->get(numBiometricsChar, "0", 1); while(strncmp("ERR", numBiometricsChar, 3) == 0) size = wifi->get(numBiometricsChar, "0", 1); numBiometricsChar[size - 2] = '\0'; cout << "Num biometrias CHAR: " << numBiometricsChar << endl; numBiometrics = atoi(numBiometricsChar); for (int i = 1; i <= numBiometrics; ++i){ size = 0; while(size != TEMPLATE_SIZE + 2){ itoa(i, biometricChar); cout << endl; cout << "Biometric Number String:" << biometricChar << endl; size = wifi->get((char *) templateSend, biometricChar, strlen(biometricChar)); sendTemplate(i-1); } } while(true) loopMatch(); return 0; }

Datasheet

Os comandos e informações gerais podem ser encontrados no datasheet da série do sensor biométrico: https://www.olimex.com/Products/Components/Sensors/SNS-FINGERPRINT/resources/ZFM-user-manualV15.pdf

O datasheet com informações mais específicas pode ser encontrado em: https://github.com/brianrho/FPM/blob/master/extras/documentation/Datasheet%201.pdf

Bibliografia

1.DALUZ, Moses, H.: **Fundamentals of Fingerprint Analysis** Disponível em: http://libgen.io/ads.php?md5=066C0D452E115A06E4CB33ECA4183787&key=3DJ476B3AMYRV7UE Acessado em: 17/09/2017.

2. MALTONI, D., MAIO, D., JAIN, A., PRABHAKAR, S. **Handbook of fingerprint recognition** Disponível em: http://libgen.io/ads.php?md5=D6483B209C4872E3A4CE6780C2F5BE01 Acessado em: 17/09/2017.

3. WIECLAW, L.: **A minutiae based matching algorithms in fingerprint recognition systems** Disponível em: https://www.researchgate.net/publication/228644313_A_minutiae-based_matching_algorithms_in_fingerprint_recognition_systems Acessado em: 17/09/2017

4. **Grove - Fingerprint Sensor Manual**. Disponível em: http://www.mouser.com/ds/2/744/Seeed_101020057-786538.pdf Acessado em: 17/09/2017.

5. **Adafruit-Fingerprint-Sensor-Library** Disponível em: https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library Acessdo em: 17/09/2017

6. TRIGGS, Robert. **How fingerprint scanners work: optical, capacitive, and ultrasonic variants explained** Disponível em: http://www.androidauthority.com/how-fingerprint-scanners-work-670934/ 18_serrau.pdf Acessado em 17/09/2017.

7. MAINGUE,J. **Introduction to biomectric** Disponível em: http://biometrics.mainguet.org/basics/introduction.htm Acessado em 17/019/2017.