Universidade Federal de Santa Catarina

Relatório e Documentação de Desenvolvimento de Projeto

Monitoramento e Acesso de Portas UFSC Blumenau

Brunno Vanelli

Blumenau

Junho de 2017

Sumário

1 Introdução

1.1 Instalando o toolchain do EPOS

Para instalar as dependências necessárias e o toolchain do EPOS basta executar o seguinte script, substituindo o usuário correspondente para o repositório do SVN.

```
1
2 # Install dependencies
3
4 sudo apt update && sudo apt -y install lib32stdc++6 libc6-i386 libc6-dev-i386
    lib32ncurses5 lib32z1
5
6 # Install virtual host
7
8 sudo apt -y install qemu-system-arm
9
10 # Install EPOS Mote toolchain
11
12 cd /tmp
13 wget -O arm-gcc-4.4.4.tar.gz epos.lisha.ufsc.br/dl88
14 sudo mkdir -p /usr/local/arm
15 sudo tar xfvz arm-gcc-4.4.4.tar.gz -C /usr/local/arm
16
17 # Clone repository
18
19 sudo apt -y install subversion
20 cd $HOME && mkdir -p epos && cd epos
21 svn checkout https://epos.lisha.ufsc.br/svn/epos2/branches/arm/
22
23 # Automated compilation test
24 cd arm && make APPLICATION=hello
```

Quadro 1: Script de Instalação do toolchain do EPOS.

1.2 Traits

O EPOS utiliza um arquivo traits para moldar como o sistema vai se comportar e quais módulos serão utilizados. Os traits importantes são:

```
1 template<> struct Traits<Build>
2 {
3     enum {LIBRARY, BUILTIN, KERNEL};
4     static const unsigned int MODE = LIBRARY;
5
6     enum {IA32, ARMv7};
7     static const unsigned int ARCHITECTURE = ARMv7;
8
9     enum {PC, Cortex};
10    static const unsigned int MACHINE = Cortex;
11
12    enum {Legacy_PC, eMote3, LM3S811, Zynq};
13    static const unsigned int MODEL = eMote3;
14
15    static const unsigned int CPUS = 1;
```

```
16      static const unsigned int NODES = 1; // > 1 => NETWORKING
17  };
```

Quadro 2: Traits da arquitetura.

```
1  template<> struct Traits<Serial_Display>: public Traits<void>
2  {
3      static const bool enabled = true;
4      enum {UART, USB};
5      static const int ENGINE = USB;
6      static const int COLUMNS = 80;
7      static const int LINES = 24;
8      static const int TAB_SIZE = 8;
9  };
```

Quadro 3: Traits do debug USB.

```
1  template<> struct Traits<Hydro_Board>: public Traits<Machine_Common>
2  {
3      static const bool enabled = false; // enable on use
4
5      static const unsigned int INTERRUPT_DEBOUNCE_TIME = 100000; // us
6
7      // Enable/disable individual relays / ADCs
8      static const bool P3_enabled = true;
9      static const bool P4_enabled = false;
10     static const bool P5_enabled = false;
11     static const bool P6_enabled = false;
12     static const bool P7_enabled = false;
13 };
```

Quadro 4: Traits do Hydro Board.

```
1  template<> struct Traits<RFID_Reader>: public Traits<Machine_Common>
2  {
3      enum {MFRC522, W400};
4      static const int ENGINE = MFRC522;
5  };
```

Quadro 5: Traits do módulo MFRC522.

2 Documentação dos Componentes

2.1 MIFARE RFID-RC522

Leitor RFID para cartões MIFARE ISO/IEC 14443 Tipo A 13.56 MHz. Esse tipo de cartão apresenta armazenamento interno de 1KB e uma chave de 48 bits (padrão `0xFFFFFFFFFFFF`). O armazenamento interno se parece algo como o Quadro 6. O protocolo de comunicação com Arduino é o SPI, que necessita de 3 pinos: o **SCK**, **MISO** e **MOSI**. Deve-se conectar também o **3.3V**, o **GND** e os pinos **RST** e **SDA** (SS).

```
Card UID: D1 13 D5 35
Card SAK: 08
PICC type: MIFARE 1KB
```

Sector	Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AccessBits
15	63	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	62	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	61	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
14	59	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	58	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	57	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	56	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
13	55	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	54	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	53	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	52	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
12	51	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	49	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	48	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
11	47	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	46	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	45	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	44	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
10	43	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	42	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	41	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
9	39	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	38	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	37	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	36	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
8	35	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	34	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	33	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	32	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
7	31	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	29	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	28	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
6	27	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	26	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	25	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	24	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
5	23	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	22	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	21	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
4	19	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	18	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	17	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	16	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
3	15	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	14	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	13	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	12	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
2	11	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
1	7	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	5	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
0	3	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	0	D1	13	D5	35	22	88	04	00	85	00	B4	2E	F0	BB	6A	A8	[0 0 0]

Quadro 6: Bloco de Memória interno dos cartões MIFARE.

A pinagem para conexão do módulo RFID ao Epos Mote III utilizada, seguindo a implementação `config_SSI` descrita em `include/machine/cortex/emote3.h` foi:

- 3.3V ligado ao 3.3V
- RST ligado à porta PC6
- GND ligado ao GND
- MISO ligado a PA4
- MOSI ligado ao PA5
- SCK ligado a PA2
- SDA ligado à porta PB5

TODO:

- ☑ Resolver problema de o EPOS ser capaz de reconhecer as portas e inicializar o RFID_Reader, mas não faz a leitura dos cartões, que foi testado no Arduino.
- ☑ Testar as portas e comunicação SPI do EPOS no osciloscópio.
- ☑ Testar novamente o MFRC522 com o osciloscópio.
- ☐ Criar interface RJ45 para conexão do módulo ao eMoteIII.
- ☑ Criar esquemáticos das ligações.

2.2 Hydro Board

O módulo é do tipo shield e as interconexões são respectivamente, seguindo as especificações em `src/machine/cortex/hydro_board_init.cc`. O acionamento dos relés pode ser alterando o nível lógico nas portas via GPIO ou utilizando a biblioteca.

- Relé P3 ligado à porta PB0
- Relé P4 ligado à porta PB1
- Relé P5 ligado à porta PB2

TODO:

- ☐ Conseguir fonte de tensão regulável ou 12V para o módulo.

2.3 ESP8266

O módulo ESP8266 utiliza comunicação UART para intermediar o eMoteIII e o servidor. As conexões necessárias para comunicação são RX e TX. As portas são:

- RX do ESP8266 ligado à porta PA1.
- TX do ESP8266 ligado à porta PA0.
- Vin e CH do ESP8266 ligado à uma fonte externa de tensão 3.3V (da Hydro Board).
- Todos os GNDs compartilhados.

TODO:

☑ Reescrever ou fazer funcionar o código ESP8266 do Lisha.

☑ Criar regulador de tensão 3.3V para alimentação da ESP8266.

☑ Criar esquemáticos das ligações.

3 Camada de Nuvem

3.1 Banco de dados

O banco de dados utilizado setado para testes e armazenamento foi o Cassandra, utilizado com um driver para NodeJS. Para instalar localmente, basta:

```
1 # Cassandra
2 curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
3 echo "deb http://www.apache.org/dist/cassandra/debian 39x main" | sudo tee -a /etc/
    apt/sources.list.d/cassandra.sources.list
4 echo "deb-src http://www.apache.org/dist/cassandra/debian 39x main" | sudo tee -a /
    etc/apt/sources.list.d/cassandra.sources.list
5 sudo apt install cassandra
```

Quadro 7: Script de Instalação do Cassandra.

3.2 Servidor de Acesso

TODO:

- ☐ Conseguir um servidor permanente.
- ☐ Fazer deployment de um cluster Cassandra em nuvem.
- ☑ Instalar versão local no LABCOP.
- ☑ Implementar segurança.

3.3 Modelagem inicial do Banco de dados

Para modelar o banco de dados, pode-se usar a linha de comando ou usar algum software para gerenciar o banco. Foi utilizado o DBeaver 4.0.5 EE, que não possui versão comercial e está disponível no link `https://dbeaver.jkiss.org/files/4.0.5/dbeaver-ee_4.0.5_amd64.deb`.

Para criar uma tabela de acessos no Cassandra, foi utilizado um Keyspace utilizando o comando:

```
CREATE KEYSPACE dados
  WITH replication = { 'class': 'SimpleStrategy', 'replication_factor': '1' }
```

Cada nodo deve ter um `nodeid` para facilitar a identificação. Esse node ID será introduzido no arquivo de configuração e deve ser único para nodo. Assim, uma tabela para armazenas as relações nodo/sala será:

```
CREATE TABLE IF NOT EXISTS dados.salas_por_nodeid (
  updated timestamp,
  sala text,
  id text,
  access_token text,
  PRIMARY KEY (id, access_token)
);
```

Então, uma vez que se obtém a sala do nodo através da autenticação, acessa-se a tabela para armazenar as entradas de dados com os UIDs dos cartões magnéticos:

```
CREATE TABLE IF NOT EXISTS dados.uid_por_salas (
  updated timestamp,
  sala text,
  uid bigint,
  PRIMARY key (sala, uid)
);
```

Para armazenar as relações salas/UIDs, utilizar-se-á uma tabela com esse propósito. Como 'sala' é o PartitionKey, ela indexará a tabela para que as buscas por uma determinada sala sejam facilitadas.

Por fim, como log dos acessos às salas, uma tabela de controle de acesso indexada nos timestamps.

```
CREATE TABLE IF NOT EXISTS dados.log_acesso (
  timestamp timestamp,
  uid bigint,
  status_code int,
  id text,
  sala text,
  PRIMARY key ((uid), timestamp)
)
WITH CLUSTERING ORDER BY (timestamp DESC);
```

Além do cadastro dos usuários para os sistemas de recursos humanos:

```
CREATE TABLE IF NOT EXISTS dados.usuario_por_uid (
  updated timestamp,
  nome text,
  matricula text,
  uid bigint,
  PRIMARY key (matricula)
);
```

TODO:

- ☑ Criar modelo de banco de dados.
- ☑ Conversar com o Alex sobre os modelos de tabelas.
- ☐ Otimizar tabelas para satisfazer boas práticas: `https://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling`
- ☑ Criar tipo de dado UID no Cassandra (resolvido com 64 bits signed integer).

3.4 Métodos de Acesso

3.4.1 Servidor REST

Foi criado um servidor REST escrito em NodeJS para processar as requisições dos dispositivos e responder adequadamente, guardando os dados de log no banco de dados. A escolha do NodeJS foi devido ao fato de o PHP se mostrar difícil de instalar bibliotecas como o driver Cassandra. Como o Nodejs possui um gerenciador de pacotes e é rápido e escalável, optou-se por utilizá-lo. Para inicializar o servidor, basta:

```
cd rest-api/
npm install
npm start
```

3.4.2 Autenticação

A autenticação é realizada através do fornecimento de um par ID/token. O ID foi definido e é validado como um valor hexadecimal de 8 caracteres, em formato string, permitindo 4294967296

(4 bilhões) de nodos diferentes, com o conjunto de caracteres [0-9a-f]. O valor do token é definido como uma sequência em base 64, com o conjunto de caracteres [0-9a-zA-Z_-]. Toda a comunicação é encriptada do servidor ao ESP8266 por meio de SSL.

```
function authenticate(req, res, next, fn) {
    var sala = "";
    const query = 'SELECT sala FROM dados.salas_por_nodeid WHERE id=? AND
    access_token=?';
    const params = [ req.params.id, req.params.token ];
    client.execute(query, params, { prepare: true }, function (err, result) {
        if (err) {
            return next(new errors.InternalServerError());
        }
        if ( !(result.rows.length > 0) ) {
            return next(new errors.UnauthorizedError());
        }
        sala = result.first().sala;
        fn(sala);
    });
}
```

Quadro 8: Autenticação.

3.4.3 Acesso por GET

As informações de acesso são armazenadas no banco de dados. O nodo ESP8266 deve então fazer uma requisição para o servidor Apache com o parâmetro salas e seu token de acesso. Para proteger o token de acesso, além da criptografia, é importante utilizar um esquema de desafio, que ainda não foi implementado:

```
server.get('/api/v1/:id/:token', (req, res, next) => {

    res.contentType = "text/plain";

    if (!validate(req)) {
        return next(new errors.BadRequestError());
    }

    authenticate(req, res, next, (sala) => {
        if (!sala) {
            return next(new errors.NotFoundError());
        }
        var ret = "";
        const query = 'SELECT uid FROM dados.uid_por_salas WHERE sala=?';
        const params = [ sala ];
        client.eachRow(query, params, { prepare: true },
            function (n, row) {
                ret += row.uid;
                ret += "\r\n";
            },
            function (err) {
                if (err) {
                    return next(new errors.InternalServerError());
                }
                res.send(ret);
```

```
                return next();
        });
    });
});
```

Quadro 9: Caminho do método GET.

Pode-se testar a aplicação com curl:

```
curl -k "https://localhost/api/v1/abcdefgh/00000000000000000000"
```

Ou por meio das bibliotecas desenvolvidas no EPOS para acesso ao ESP8266:

```
ESP8266 esp8266;
char data[1500];
char * url = "localhost/api/v1/abcdefgh/00000000000000000000";
esp8266.get(url, data, sizeof(data));
```

A saída é um arquivo "text/plain"com as UIDs que são autorizadas à entrar na sala, separadas por um fim de linha e carriage return (`\r\n`). Formatos mais sofisticados como JSON ou XML poderiam ser usados, mas eles aumentam o tempo de processamento e uso de memória no microcontrolador.

3.4.4 Acesso por POST

Para enviar dados para serem persistidos na nuvem, o nodo pode fazer requisições para a nuvem utilizando seu ID e token.

```
server.post('/api/v1/:id/:token', (req, res, next) => {

    res.contentType = "text/plain";

    if (!validate(req)) {
        return next(new errors.BadRequestError());
    }

    authenticate(req, res, next, (sala) => {
        if (!sala) {
            return next(new errors.NotFoundError());
        }
        if ( !req.body || !req.body.uid || !req.body.timestamp || !req.body.code) {
            return next(new errors.BadRequestError());
        }
        if ( !validate_body(req) ) {
            return next(new errors.BadRequestError());
        }
        const uid = Number(req.body.uid);
        var timestamp = Number(req.body.timestamp);
        const code = Number(req.body.code);

        if (isNaN(uid) || isNaN(timestamp) || isNaN(code)) {
            return next(new errors.BadRequestError());
        }

        if (timestamp == 0) {
            timestamp = (new Date).getTime();
```

```
29          }
30
31          if (timestamp < 1445385600000) {
32              return next(new errors.ImATeapotError('https://i.imgur.com/KQu2mXz.jpg')
    );
33          }
34
35          const query = 'INSERT INTO dados.log_acesso (uid, timestamp, status_code, id
    , sala) VALUES (?, ?, ?, ?, ?) IF NOT EXISTS';
36          const params = [ uid, new Date(timestamp), code, req.params.id, sala];
37          client.execute(query, params, { prepare: true }, function (err, result) {
38              if (err) {
39                  return next(new errors.InternalServerError());
40              }
41              if ( !result.first()['[applied]']) {
42                  return next(new errors.ConflictError());
43              }
44              res.send('OK');
45              return next();
46          });
47      });
48  });
```

Quadro 10: Caminho do método POST.

O servidor aceita e valida três variáveis diferentes, **timestamp**, **code** e **uid**.

Tabela 1: Tabela de variáveis do servidor REST.

Variável	**Descrição**
uid	Unique identifier do cartão MIFARE, constituído (normalmente) de 4 bytes (32 bits), e enviado como um `unsigned int`.
code	Código de status do acesso, representado também por um `unsigned int`.
timestamp	Milissegundos decorridos desde 1 de Janeiro de 1970. Caso esse valor seja zero, o valor do servidor é usado.

Para testar essa função:

```
curl -k -d "uid=1&timestamp=0&code=1" "https://localhost:8080/api/v1/abcdefgh
    /00000000000000000000"
```

Ou por meio das bibliotecas desenvolvidas no EPOS para acesso ao NIC:

```
ESP8266 esp8266;
char * data = "uid=1&timestamp=0&code=1";
char response[1500];
char * url = "localhost/api/v1/abcdefgh/00000000000000000000";
```

```
5 esp8266.post(url, data, strlen(data), response, sizeof(response));
```

4 Descrição de Implementação

O sistema a ser implementado pode ser visto na Figura 1.

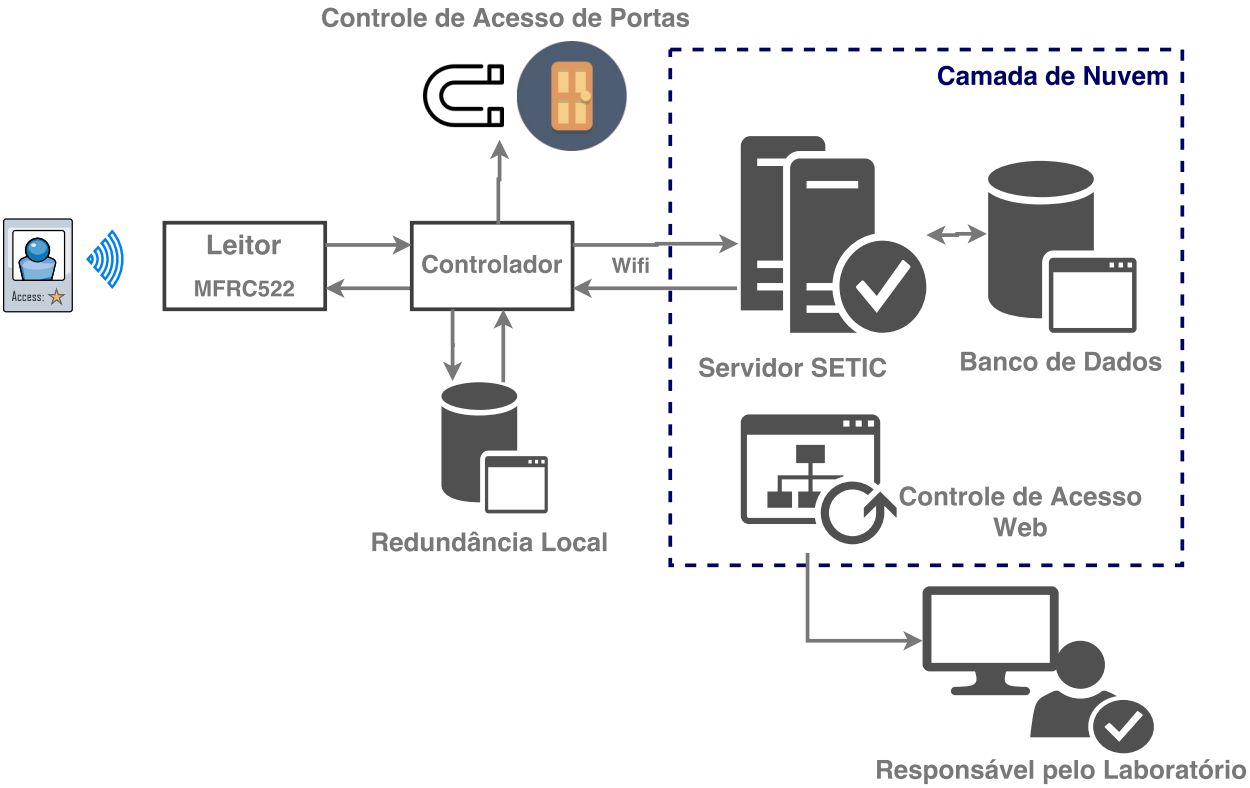

Figura 1: Sistema de controle de acesso.

4.1 ESP8266

Nesse projeto, foi utilizado um módulo Nodemcu por ser um modelo de ESP8266 mais avançada, possuir maior armazenamento interno e ter melhor conectividade com o computador, além de diversas portas disponíveis.

4.1.1 Conexão com a rede eduroam

Para conectar os módulos na rede, é necessário utilizar a última versão do SDK mantido pela Espressif. Para conexão com a eduroam (PEAP-MS-CHAPv2), é necessário que a SDK do Espressif seja no mínimo >2.0.0. No momento que está sendo escrito essa documentação, foi utilizado a versão da biblioteca Arduino 2.4.0-rc.1 commit `c730c0f`.

Pode-se executar as seguintes linhas:

```
cd /usr/local/arduino
cd hardware
mkdir esp8266com
cd esp8266com
```

```
git clone https://github.com/esp8266/Arduino.git esp8266

cd esp8266/tools
python get.py
```

Ou usar a seguinte plataforma no platformio.ini:

```
platform = espressif8266_stage
```

É também necessário mudar a identidade anônima que o ESP8266 usa para acessar a rede. Não tenho certeza quanto a verdadeira necessidade dessa mudança, mas a versão de testes que conectou-se à rede utilizava a versão modificada da biblioteca libwpa2.a, da SDK do Espressif. Para modificar esse arquivo, basta localizá-lo dentro da biblioteca instalada (`/usr/local/arduino/hardware/esp8266com/esp8266/tools/sdk/lib` para o Arduino e para Platformio `~/.platformio/packages/framework-arduinoespressif8266/tools/sdk/lib`) e executar os comandos:

```
cp libwpa2.a libwpa2.a.bk
bbe -e "s/anonymous@espressif.com/anonymous123456@ufsc.br/" -o libwpa2.a libwpa2.a.bk
```

4.1.2 Lista de Comandos AT

O ESP8266 foi programado para responder a comandos do tipo AT. Para enviar um comando, deve-se enviar `AT+COMANDO`, e para passar um parâmetro, `AT+COMANDO=PARAMETRO`. Todos os comandos devem ser terminados com um carriage return (
r). A resposta é sempre do tipo `OK` ou `ERROR`, seguida de uma descrição ou valor adicional. Exemplo:

```
AT+SYSTEMREADY\r
>OK\r
AT+GETFORMATTEDIP\r
>OK=192.168.0.25\r
AT+TIMESTAMP\r
>ERROR=CLOCKNOTSYNCED\r
```

A Tabela 2 mostra os comandos disponíveis para o ESP8266.

4.2 EPOS Mote III

Para implementação no nodo EPOS, existem três elementos básicos, a ligação e comunicação com o módulo ESP8266, a ligação com o módulo MFRC522 e a ligação com relé da Hydro Board.

Para fazer todo o protótipo funcionar foi necessário utilizar uma versão modificada da classe do EPOS `Persistent_Ring_FIFO`, e foi criada uma nova classe chamada `Persistent_Ring` que resolve esse problema, tendo tanto funções `pop_fifo()` como `pop_filo()`.

O modo de funcionamento do EPOS nesse cenário pode ser visto na Figura 2.

Tabela 2: Tabela de Comandos do firmware do ESP8266

Comando	**Retorno**
AT+SYSTEMREADY	Retorna se o filesystem e a conexão WiFi estão funcionais.
AT+CONNECTWIFI	Tenta se conectar ao WiFi padrão do firmware (executado por padrão no início).
AT+TIMESTAMP	Retorna a string de um inteiro de 32 bits representando o Epoch time.
AT+FTIMESTAMP	Retorna o timestamp completo formatado segundo a norma ISO 8601.
AT+HEAPSIZE	Retorna o tamanho da memória heap do ESP8266.
AT+GET=url	Envia uma requisição do tipo HTTP GET para a url e retorna o conteúdo sem os headers.
AT+GETS=url	Envia uma requisição do tipo HTTPS GET para a url e retorna o conteúdo sem os headers.
AT+PAYLOAD=payload	Cria um payload para requisições do tipo POST, e é consumido após chamar o comando POST.
AT+POST=url	Envia uma requisição do tipo HTTP POST para a url com o conteúdo do PAYLOAD e retorna o conteúdo sem os headers.
AT+POSTS=url	Envia uma requisição do tipo HTTPS POST para a url com o conteúdo do PAYLOAD e retorna o conteúdo sem os headers.
AT+IP	Retorna a string de um inteiro de 32 bits representando o IP do nodo.
AT+FIP	Retorna o IP formatado do nodo.
AT+RSSI	Retorna o RSSI do wifi_station.
AT+CHANNEL	Retorna o canal do Wifi.

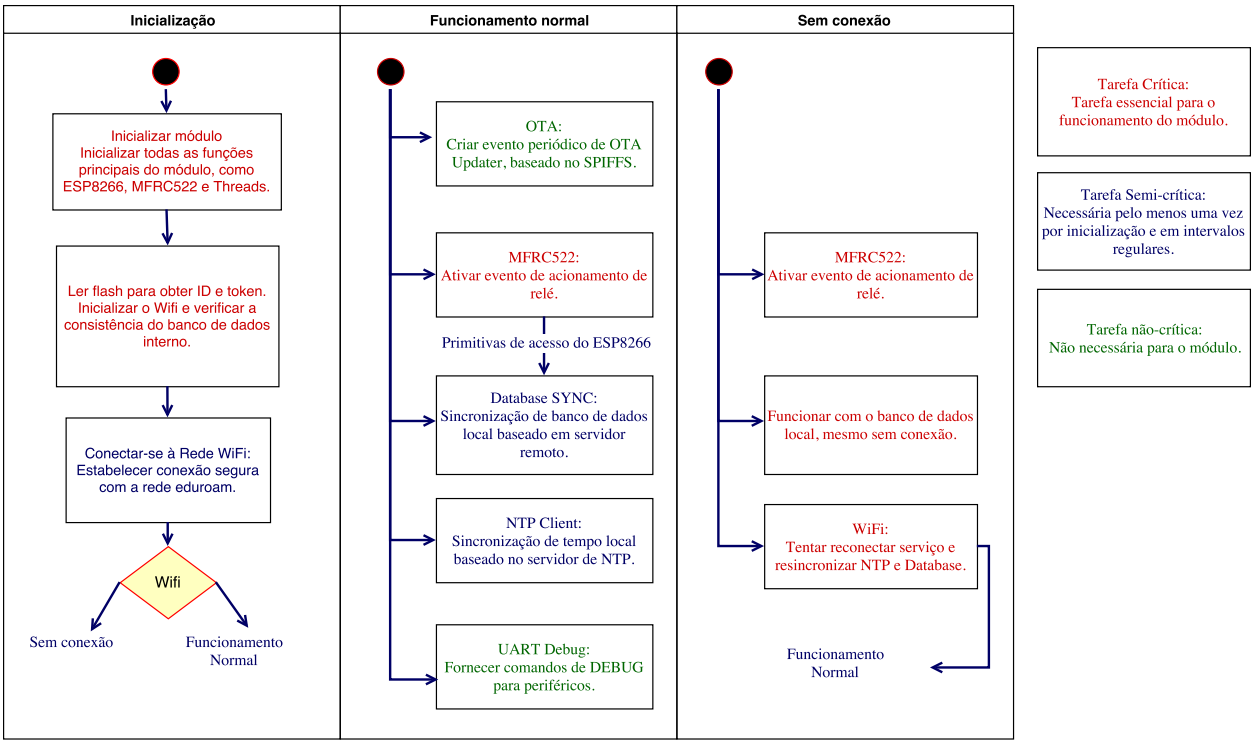

Figura 2: Diagrama de funcionamento do eMote3.

4.2.1 ESP8266

O ESP8266 aceita os requests do EPOS do tipo serial (baud 115200) e **não é uma classe síncrona, ou seja, não trata concorrência**. Ainda não foi estudado os impactos da forma insegura de transmissão de dados via serial.

A ligação dos módulos pode ser vista na Figura 3.

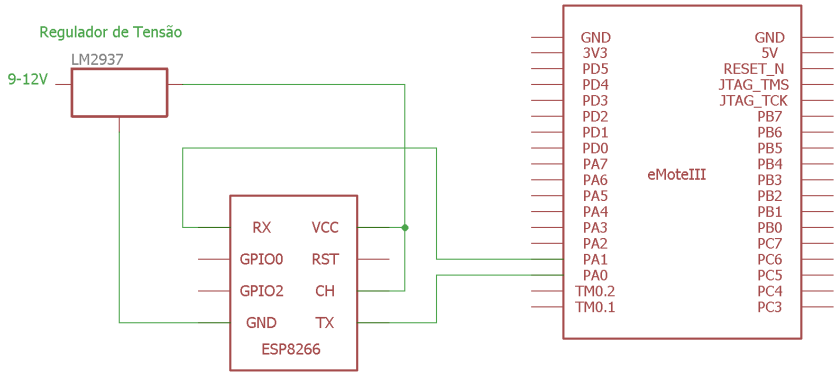

Figura 3: Diagrama de ligação eMote3-ESP8266.

4.2.2 MFRC522

O módulo RFID MFRC522 roda em uma Thread que lê os cartões, lê o banco de dados na presença de um cartão, gera uma chamada de acesso (permitido ou negado) e despacha uma nova entrada no banco de dados local, para depois ser persistido pela rotina de sincronização.

A ligação dos módulos pode ser vista na Figura 4.

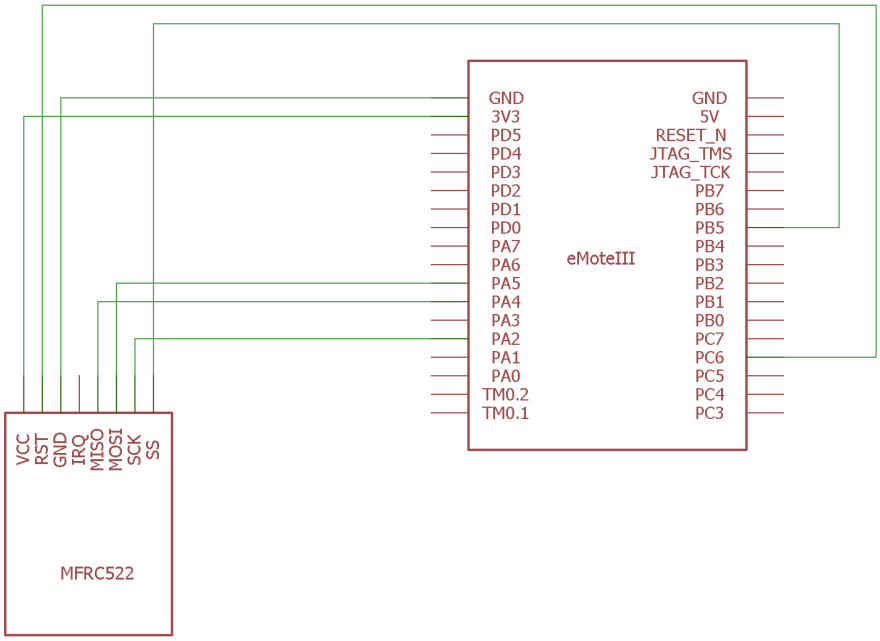

Figura 4: Diagrama de ligação eMote3-MFRC522.

4.2.3 Hydro Board

A Hydro Board ainda não foi testada. Um LED está sendo usado em seu lugar para simular o relé.

4.2.4 Banco de Dados local

Como já dito, o banco de dados local foi implementado em `Persistent_Ring`. Seria preferível implementá-lo utilizando o sistema de RIFFS (filesystem) que está sendo produzido para o EPOS, mas ele se mostrou instável e com funções faltantes, e era muito complexo para ser adaptado ao cenário.

Para isso, foi implementado um sistema de armazenamento interno, que respeita a seguinte lógica de funcionamento:

1. Um evento é chamado em uma função `rfid_reader()`, empurrando uma nova entrada no armazenamento interno.

2. A função de manutenção do banco de dados roda em intervalos específicos, utilizando a classe `Periodic_Thread` do EPOS.

3. Quando a função de manutenção é acordada, ela realiza dois passos em ordem:
   (a) Persistir todas as entradas de dados na nuvem, que possuem nível de importância maior.
   (b) Atualizar o banco de dados interno com a relação de UIDs que tem acesso à sala.
4. Caso o acesso ao meio esteja comprometido e o nodo não consiga atualizar-se com a nuvem, ele espera novamente o tempo de ciclo e volta ao passo 3.

4.2.5 Instalando o firmware no dispositivo

O dispositivo ainda não conta com uma build automatizada. Logo, para fazer sua própria build, siga os seguintes passos:

```
cd arm/
# alterar o id e o token no codigo em app/rfid_door_blumenau.cc
make APPLICATION=rfid_door_blumenau flash
# conectar o EPOS Mote
# fazer as conexoes fisicas
```