Sistema de Catraca Eletrônica Utilizando o PIC e Interface Computacional em Python

André G. G. Pinto, Danubia Macedo

***Abstract*—Entre as milhares de aplicações das tecnologias, elas possuem um papel importante na segurança das pessoas. Um sistema que é bem comum para a segurança de estabelecimentos e edifícios é a catraca eletrônica. Assim, este projeto tem o objetivo de desenvolver um sistema de catraca eletrônica utilizando o PIC e a interface computacional em Python. No final os resultados são apresentados e avaliados. Os sistemas funcionam de maneira isolada e a atualização das informações entre eles não foi realizada completamente.**

***Index Terms*—Python, PIC, SQLite, Comunicação Serial, CRUD.**

I. Introdução

AS Tecnologias surgem a todo momento e servem muitas vezes para melhorar a qualidade de vida das pessoas. Dentre suas milhares de aplicações, elas possuem um papel importante na segurança de edifícios, empresas e sistemas de transporte público, entre outros. Um sistema que é bem comum nesses lugares funciona em conjunto com a catraca eletrônica. Com esse tipo de sistema o estabelecimento consegue gerenciar os usuários, saber quais são os usuários pagantes e não pagantes e assim tomar medidas de controle de acesso ao local [1].

A eletrônica dispõe-se de diversas ferramentas que possibilitam cada vez mais a integração do ser humano ao mundo rodeado de novas tecnologias. Os microcontroladores possuem um papel fundamental para o desenvolvimento da interação humano-máquina. Uma das suas principais características que facilitam isso é a facilidade de integração com diversos sistemas e o baixo custo. No microcontrolador PIC16F877A por exemplo, é possível realizar a comunicação com diversos periféricos e *softwares*.

Portanto, o objetivo deste projeto é desenvolver um sistema de catraca eletrônica utilizando o microcontrolador PIC16F877A e a interface computacional em PyQt5 com a persistência de dados sendo realizada na memória externa EEPROM(Electrically Erasable Programmable Read-Only Memory) e banco de dados SQLite, respectivamente.

As seguintes seções serão estruturadas da seguinte maneira: a seção II descreve os principais conceitos utilizados, já na seção III serão mostrados os materiais e métodos. Na seção IV as discussões e análises do que foi feito, finalizando com a V, os resultados do projeto desenvolvido e VI a conclusão com as devidas considerações finais.

II. A Teoria

Esta seção tem por objetivo apresentar os conceitos basilares para realizar o desenvolvimento do sistema da catraca eletrônica.

A. O microcontrolador PIC16F877Q

O PIC16F877A é um microcontrolador da família 16F que possibilita o controle de diversos dispositivos de forma barata e simples. Formado por uma unidade lógica aritmética (ULA), memória, portas I/O, temporalizadores, porta de comunicação externa, *clock*, interruptores possuem as seguintes características [2]:

- CPU RISC
- Velocidade de operação de 20MHz
- Três *timers*
- Módulo PWM
- Comunicação serial e I2C
- Entrada analógica
- 40 pinos
- Alimentação 5V

Tais características nos permite realizar uma grande gama de operações com esse simples microcontrolador.

B. Software CCS C Compiler

O software CCS C Compiler é um compilador em C desenvolvido pela Microchip a mais de 20 anos, para programar em microcontroladores PIC.

C. Simulador PICSimLab

Para realizar as simulações necessárias para a execução desse projeto, foi necessário a utilização de um simulador. Desenvolvido em 2008 por Luis Claudio Gambôa Lopes, o PICSimLab é um simulador de placas com a integração do depurador MPLABX/avr-gdb [3].

Sua principal proposta é simular um hardware real. Para isso, conta com 8 tipos de placas e em ambas integradas diversos componentes e ainda com possibilidade de expansão.

A comunicação do código fonte do PIC com a placa é feito através da funcionalidade de carregamento de código hexa. A IDE CSS C Compiler, após realizar a compilação do código, gera um arquivo hexadecimal, podendo dessa forma ser utilizado no PICSimLab.

D. Qt Designer e Python

Desenvolvido em 2007 pela empresa Qt Project, o Qt Designer é uma IDE multi-plataforma que permite a construção de GUIs (*Graphical User Interface*) de forma fácil e ágil [4].

Desenvolvido inicialmente para C++, sua facilidade de criação de telas interativas foram tanta que diversas linguagens também o aderiram, como o Python.

Embora o Qt Designer gerasse arquivos em formato ".ui" para geração de aplicações GUI, é possível converte-los em

para a linguagem Python através do comando **pyuic5 -o [NomeArquivoPython.py] [NomeArquivoQtDesigner.ui]**. Essa conversão é realizada através do PyQt. O PyQt é um empacotador da linguagem Python para estrutura de aplicativos Qt [5].

E. Banco de dados SQLite

O SQLite é uma biblioteca em C, que permite acesso a banco de dados SQL. Ele faz com que não seja necessário o uso de um SGBD(Sistema de Gerenciamento de Banco de Dados), sendo assim sem servidor, sem configuração e transacional[6].

III. Materias e Métodos

Esta seção apresenta os componentes e a metodologia utilizada para o desenvolvimento deste artigo como descrito a seguir.

A. Programas Utilizados

Nome	**Tipo**
PICSimLab	Simulador
CCS C Compiler	IDE
PyQt5	GUI
Qt Designer	GUI
SQLite	Banco de Dados

O projeto é feito inteiramente via software. O PIC é simulado utilizando o programa PICSimLab [3] e o compilador CSS gera os códigos para rodar no PIC, que inclusive também pode ser utilizado de maneira física. A placa utilizada é a PIC-Genios [7] que contém o *display* LCD necessário para exibir as informações e, finalmente foi adicionado uma *spare part* do teclado. Vale notar que para o uso do LCD e teclado foram utilizados *drivers* que podem ser encontrados no repositório [8].

B. Arquitetura do sistema

O sistema proposto é baseado na criação de partes que permitem a manutenção e autenticação de pessoas. As duas partes funcionam de maneira independente, mas podem realizar a atualização dos dados entre eles de forma assíncrona. Portanto, este trabalho se concentra no funcionamento isolado de ambos os sistema e só então a realização da comunicação entre eles.

Os sistemas possuem as mesmas funções: criar, buscar, deletar, editar e excluir, representando um CRUD (*Create Read Update Delete*). O que muda entre eles são as linguagens de programação, o modo de lidar com essas telas e a gravação dos dados. No PIC, por exemplo é bem mais limitado, principalmente para o gerenciamento da memória e a interação com o teclado.

*1) **Microcontrolador PIC**:* O sistema é controlado pelo microcrontrolador PIC PIC16F877A e utiliza uma memória externa EEPROM 24256 de 256 Bytes.

A memória EEPROM é conectada com o protocolo i2c por padrão no drive da memória nos pinos SDA (Pino C4) e SCL (Pino C3) do PIC e são os mesmos utilizados na placa PICGenios no PICSimLab.

Para iniciar a memória externa, deve-se chamar a função **"init_ext_eeprom()"** e ter importado a biblioteca "24256.c" que é inclusa na instalação do compilador CCS.

Utilizando o *driver* da memória, existem outras funções que foram utilizadas, a de escrever e ler. Para escrever é utilizado o **"write_ext_eeprom()"** que recebe os parâmetros do endereço da memória e o valor a ser armazenado. Para ler a rotina é **"read_ext_eeprom()"**, recebe o endereço da memória, retornando o valor naquele endereço. Na Fig. 1, possui a escrita e leitura da memória.

O teclado matricial de 4x4 possui 16 teclas e é incluso no PICSimLab. O *driver* do teclado faz uma varredura completa e identifica se uma tecla foi apertada. Isso acontece quando a função **"tc_tecla()"** é chamada, como parâmetro deve-se passa o tempo limite dessa varredura e caso não seja pressionada tecla alguma o retorno é **255**. É possível observar um exemplo do teclado na Fig. 1.

```
//------------------------------------
tmp = tc_tecla(1000); // ms
//------------------------------------
if(tmp≠255){

    write_ext_eeprom(0, tmp);
    delay_ms(50);
    tmp_result = read_ext_eeprom(0);
    delay_ms(50);
    // tmp_result = tmp;
    printf (lcd_escreve,"\f Button: %c", tmp_result);

  }else{

    printf (lcd_escreve,"\f   TECLADO  ");
  }
```

Fig. 1: Exemplo da aplicação do teclado e a memória externa.

O LCD é de 16 colunas e já é incluso na placa PICGenios. Aqui também é utilizado um *driver* para o LCD e configurado os pinos para a placa. Para a impressão é chamado o **"printf"** e passa-se como parâmetro a variável "lcd_escreve", o restante é utilizado todas as particularidades do **"printf"** para imprimir na tela. A configuração pode ser vista na Fig. 2.

```
#ifndef lcd_enable
   #define lcd_enable      pin_e1
   #define lcd_rs          pin_e2
   //#define lcd_rw        pin_e2
   #define lcd_d4          pin_d4
   #define lcd_d5          pin_d5
   #define lcd_d6          pin_d6
   #define lcd_d7          pin_d7
#endif

#include "mod_lcd.c"
```

Fig. 2: Definição dos pinos do LCD de acordo com a placa PICGenios utilizada.

Como o PIC utiliza a memória EEPROM externa é preciso criar soluções para conseguir gerenciar essa memória.Assim, foi definido o tamanho de bytes utilizado para cada usuário. Para buscar as informações em uma ordem exata foi preciso que as informações sempre forem salvas respeitando a mesma ordem, como o ID estando nas primeiras posições.

Para registrar um usuário por exemplo é preciso definir qual o último endereço disponível. Com os dados coletados do teclado passar para uma *struct* e gravar os dados na sequência que foi convencionada. A partir do endereço inicial é somada a próxima posição até ocupar o número de bytes do bloco. Como pode ser observado na Fig. 3, dentro da rotina **"saveUser"**.

```
User user;
user.id[0] = id[0];
user.id[1] = id[1];
user.pass[0] = pass[0];
user.pass[1] = pass[1];
user.pass[2] = pass[2];
user.pass[3] = pass[3];
user.status = status;

write_ext_eeprom(address, user.id[0]);
write_ext_eeprom(address+1, user.id[1]);
write_ext_eeprom(address+2, user.pass[0]);
write_ext_eeprom(address+3, user.pass[1]);
write_ext_eeprom(address+4, user.pass[2]);
write_ext_eeprom(address+5, user.pass[3]);
write_ext_eeprom(address+6, user.status);
```

Fig. 3: Código dentro da rotina "saveUser". Antes disso, o endereço definido foi determinado por uma função que retorna a última posição disponível.

Outro recurso que vale a pena mencionar é de exclusão de usuários. A rotina **"deleteUser"** recebe como parâmetro o ID do usuário, com ele a função **"getAddressByID"** é responsável por retornar o endereço inicial do usuário registrado. antes de ir para a particularidade de cada condição da memória, tem-se o endereço do usuário atual e o próximo. Com essas informações, existem dois casos:

- **Caso 1**: Se **não** existir o próximo bloco contendo o usuário, ou seja o endereço for vazio (-1), pegue o endereço do usuário obtido e apague os dados da memória.

- **Caso 2**: Se existir o próximo bloco contendo o usuário, ou seja o endereço for diferente de vazio (-1), faça um *loop* que subescreva o usuário atual com as informações do próximo e continue até que a última posição seja vazia. O resultado é que os usuários são movidos um bloco à esquerda na memória. A codificação desses casos pode ser vista na Fig. 4.

```
int previous_block = address;
int next_block = address + BLOCK_SIZE;
//Case 1: without the next block
if (read_ext_eeprom(next_block) == -1){
   erase_program_eeprom(previous_block);
   return 1;
}
//Case 2: Check whether the data in the next block addr
while(read_ext_eeprom(next_block) ≠ -1){
   //data_temp[index] = read_ext_eeprom(address);
   for(int i=0; i < BLOCK_SIZE; i++){
      //Read the data from the next block
      data_temp = read_ext_eeprom(next_block + i);
      //overwrite the previous block with the data fro
      write_ext_eeprom(previous_block + i, data_temp);
   }
   previous_block = next_block;
   next_block += BLOCK_SIZE;
}
//Go back to erase the block that's duplicated
next_block -= BLOCK_SIZE;
```

Fig. 4: Código dentro da rotina "deleteUser".

Por último, para receber as informações do usuário é necessário encontrar uma maneira de aguardar a entrada do usuário e guardar em uma variável. As informações que precisam se gravadas são o ID (2 bytes), senha (4 bytes) e o status (1 byte). A representação do usuário em formato *struct* pode ser observado na Fig. 5.

```
typedef struct {
   int id[2];
   int pass[4]; //password
   int status; //0-Unpaid,1-Paid, 3-Admin
}User;
```

Fig. 5: Formato de cada usuário no sistema do PIC.

Sempre que o teclado precisar ser utilizado para digitar alguma informação do usuário, é criado uma *loop* que se repete até o tamanho total de cada campo. Por exemplo, no caso da senha a contagem vai até quatro e só termina assim que o usuário tenha inserido quatro valores válidos. A cada valor digitado é colocado em um *array* temporário que é usado por outra função, como a **"inputKeyboardUser()"** para posteriormente salvar os dados na memória. Este código pode ser observado na Fig. 6.

```
while(i < max){

   printf(lcd_escreve,"\f%s", msg);
   delay_ms(50);
   option = readKeyboard();
   if(option != 255){
      printf(lcd_escreve,"\n\rTyped:%c", option);
      delay_ms(400);
      unsigned char destination[2];
      //Convert string from char and return
      //to the left array of char(str_pass)
      strfromchar(destination,option);
      temp = strToInt(destination);
      data[i] = temp[0];
      i++;
   }
}
```

Fig. 6: Função que aguarda os valores digitados pelo usuário/admin e retorna um vetor com a sequência de números digitados. No caso da senha o parâmetro "max" recebe quatro e a msg é "Digite a senha:".

A função para aguardar e obter a sequência digitada é utilizado por muitas outras funções no sistema. O restante se preocupa em criar menus e formas de organizar o CRUD e o login no sistema. Após entender o funcionamento das funções anteriores o restante do sistema apenas faz algumas modificações. Na Fig. 7 é possível ver a rotina **adminMenu()** que possui as opções para realizar o CRUD e na Fig. 8 o login do sistema com algumas condições baseada nas informações digitadas.

```
switch(option){
   case '1':
      inputKeyboardUser();
      break;
   case '2':
      temp = inputId();
      id[0] = temp[0];
      id[1] = temp[1];
      searchUser(id);
      break;
   case '3':
      temp = inputId();
      id[0] = temp[0];
      id[1] = temp[1];
      signed int success = deleteUser(id);
      printf(lcd_escreve,"\fSuccess → %d", success);
      delay_ms(500);
      (success ≥ 1)?
      printf(lcd_escreve,"\fusuario deletado"):
      printf(lcd_escreve,"\fusuario N Existe");
      delay_ms(500);
      break;
   case '4':
      editUser();
      break;
   default:
      printf(lcd_escreve,"\fDigite um valor");
      printf(lcd_escreve,"\r\nValido!");
      delay_ms(500);
```

Fig. 7: Rotina que gerencia as opções, dependo da opção digitada um recurso do CRUD é chamado.

```
int address = getAddressByID(id);
if(address == -1){
   printf(lcd_escreve,"\fID N Existe");
   printf(lcd_escreve,"\r\nTente de novo");
   delay_ms(1000);
   return -1;
}
unsigned int * temp;
unsigned int pass[4];

char msg [] = "Digite a senha: ";
int max = 4;
temp = inputToKeyboard(msg, max);
pass[0] = temp[0];
pass[1] = temp[1];
pass[2] = temp[2];
pass[3] = temp[3];
int result_pass = checkPassword(address,pass);
if(result_pass != 0){
   printf(lcd_escreve,"\fProcure a secretaria");
   printf(lcd_escreve,"\r\nP/ resolver");
   delay_ms(1000);
   printf(lcd_escreve,"\fNADA eh Ligado!");
   delay_ms(500);
   return -1;
}

int show = 0;
int status = getUserStatus(address,show);
```

Fig. 8: Função para logar no sistema, a cada informação digita é conferido se elas estão corretas.

A função login faz uso da função **”inputToKeyboard()”** que foi mencionada anteriormente. Primeiramente, para fazer login as informações precisam estar cadastradas no sistema. Dessa forma, é realizado a busca do ID, caso não existir é retornado o valor ”-1”.Após isso, a senha deve ser igual, caso não seja é apresentado uma mensagem ao usuário e a catraca não é liberada, obviamente.

No final é obtido o status do usuário dado o seu endereço inicial e setado para a opção para não exibir nada (show=0). O tratamento do estado é então utilizado fora da função, caso seja 1 (pagante) ou 3 (admin) a catraca é liberada, caso seja 0 (não pagante) a conta existe, mas não foi paga. Isso pode ser observado na Fig. 9. O restante das funções se encontra no repositório.

```
if(status == 1 || status == 3){
   printf(lcd_escreve,"\fBem Vindo(a)!");
   delay_ms(1000);
   printf(lcd_escreve,"\fLiga Led e Rele");
   delay_ms(500);
}else if(status == 0){//Unpaid
   printf(lcd_escreve,"\fConta Existe");
   printf(lcd_escreve,"\r\r,Mas Falta Pagar!");
   delay_ms(1000);
}
```

Fig. 9: Tratamento dos status, dependendo do status o login não é permitido.

*2) **Interface computacional**:* A interface computacional tem como finalidade funcionar de forma independente do microcontrolador. Diante disso, foi desenvolvido telas utilizando o PyQt5 para realizar login e operações CRUD conectadas ao banco de dados SQLite.

Inicialmente é necessário que seja realizado o login do usuário, essa ação esta vinculada à um GUI e ao banco de dados, sendo o método responsável por realizar essa verificação apresentado na Fig. 10.

```
try:

    #Loading Data
    self.connection = sqlite3.connect("./Interface PC/db/academy.db")
    self.c = self.connection.cursor()
    query = "SELECT * FROM User WHERE  id = {} and password = {}".format(int(user), int(password))
    resultQuery = self.c.execute(query)
    row = resultQuery.fetchone()

    if (row[2] == 3):
        self.window = MainWindow()
        self.window.show()
    elif (row[2]  ==  0):
        QMessageBox.warning(QMessageBox(), 'Error', 'Acesso não liberado!\n Por favor, verique sua mensalidade.')
    else:
        QMessageBox.information(QMessageBox(), 'Successful', "Acesso liberado.\n Bem-vindo! ")
        #ENVIA DADO PARA LIBERAR ACESSO

    self.connection.commit()
    self.c.close()
    self.connection.close()


except Exception:
    QMessageBox.warning(QMessageBox(), 'Error', 'Usuário não encontrado.')
```

Fig. 10: Código para busca de usuário em banco de dados e verificação para login.

Após realizado login, caso o usuário seja um administrador será redirecionado à uma tela principal, caso seja um usuário padrão sera liberado ou não sua entrada (condicionado ao pagamento).

Na tela principal é realizado a listagem dos usuários cadastrados, para isso é feito um select no banco de dados conforme apresentado na Fig. 11.

```
def loadData(self):
    try:

        #Loading Data
        self.connection = sqlite3.connect("./Interface PC/db/academy.db")
        self.c = self.connection.cursor()
        query = "SELECT * FROM User"
        result = self.c.execute(query)
        self.connection.commit()


        self.ui.table_list.setRowCount(0)
        for row_number, row_data in enumerate(result):
            self.ui.table_list.insertRow(row_number)
            for column_number, data in enumerate(row_data):
                self.ui.table_list.setItem(row_number, column_number,QTableWidgetItem(str(data)))

        self.c.close()
        self.connection.close()
    except sqlite3.OperationalError:
        #os.mkdir('db')
        QMessageBox.warning(QMessageBox(), 'Error', 'Could not add user to the database.')
```

Fig. 11: Listagem de dados na tela principal.

Ainda na tela principal, é possível encontrar outras ações para serem realizar. A inserção de novos usuários nos encaminha à uma nova tela de inserção, onde realizamos um insert user com seus valores conforme apresentado a Fig. 12.

```
try:
    self.conn = sqlite3.connect("./Interface PC/db/academy.db")
    self.c = self.conn.cursor()

    if id_user == "" or password_user == "" or status_user == "":
        QMessageBox.information(QMessageBox(),"Unsuccess", "Please, check the data entries! ")
    else:

        self.c.execute("INSERT INTO User (id, password, status) VALUES ('{}','{}','{}')"
        .format(int(id_user), int(password_user), int(status_user) ))
        QMessageBox.information(QMessageBox(),"Successful", "Data registered successfully ")

    self.conn.commit()
    self.c.close()
    self.conn.close()
except Exception:
        QMessageBox.warning(QMessageBox(), 'Error', 'Could not add user to the database.')
```

Fig. 12: Inserção de um novo usuário.

É importante ressaltar que é necessário que cada digito digitado não ultrapasse o valor de máximo de 99, logo é realizado uma verificação dos campos de inserção de ID e senha Fig. (13).

```
#Maximum value is 99
if any([int(self.ui.input_id1.text()) > 99, int(self.ui.input_id2.text()) > 99,
 int(self.ui.input_senha1.text()) > 99, int(self.ui.input_senha2.text()) > 99 ,
 int(self.ui.input_senha3.text()) >99, int(self.ui.input_senha4.text()) > 99])  :
    QMessageBox.warning(QMessageBox(), 'Error', 'Cada campo deve ter o valor máximo de 99 .')
```

Fig. 13: Controle de valor máximo de cada campo de inserção do ID e senha.

Esse tipo de validação é importante para que não ocorra conflito de valores com os dados do microcontrolador. A próxima opção possível de interação é a edição de usuários, no qual é necessário realizar a busca do usuário indicado e assim que alterado é realizado um update. Essas ações necessárias é feita dentro dos métodos searchQuery e updateQuery Fig. (14).

```
def updateQuery(self):

    id_user = self.ui.input_id1.text()
    password_user = self.ui.input_senha1.text()
    status_user = self.ui.input_status.text()

    try:
        self.conn = sqlite3.connect("./Interface PC/db/academy.db")
        self.c = self.conn.cursor()
        self.c.execute("UPDATE User SET id = {}, password = {}, status = {} WHERE id  = {}"
        .format(int(id_user), int(password_user), status_user, int(id_user) ))
        self.conn.commit()

        QMessageBox.information(QMessageBox(),"Successful", "Data Update successfully ")


        self.c.close()
        self.conn.close()
    except Exception:
            QMessageBox.warning(QMessageBox(), 'Error', 'Could not update user to the database.')
```

Fig. 14: Método updateSearch: atualização do usuário.

O método searchQuery realiza apenas um select condicionado ao ID especificado pelo usuário e o retorna atualizando os campos do formulário de edição, observe a Fig. 15.

```
try:

    #Loading Data
    self.connection = sqlite3.connect("./Interface PC/db/academy.db")
    self.c = self.connection.cursor()

    query = "SELECT * FROM User WHERE  id = " + self.ui.lineEdit.text()
    resultQuery = self.c.execute(query)
    row = resultQuery.fetchone()

    self.ui.input_id1.setText(str(row[0]))
    self.ui.input_senha1.setText(str(row[1]))
    self.ui.input_status.setText(str(row[2]))

    self.connection.commit()
    self.c.close()
    self.connection.close()

except Exception:
    QMessageBox.warning(QMessageBox(), 'Error', 'Could not Find User from the database.')
```

Fig. 15: Método searchQuery: Pesquisa de usuário condicionado ao seu ID e atualização dos campos do formulário.

Além de edição é possível buscar um usuário pelo ID, esse método é semelhante ao método searchQuery apresentado anteriormente, entretanto não ira atualizar nenhum campo, apenas apresentar o resultado da busca em um QMessageBox, podendo ser um box de informação caso encontre o usuario Fig. 16 ou de erro caso não seja encontrado Fig. 17.

```
serachresult = "Usuario encontrado!"+ "\n ID : "
+str(row[0])+'\n'+"Status: "+str(row[2])+'\n'

QMessageBox.information(QMessageBox(), 'Successful',
serachresult)
```

Fig. 16: Código de retorno da busca por um usuário.Busca de usuário com sucesso.

```
QMessageBox.warning(QMessageBox(), 'Error',
 'Could not Find User from the database.')
```

Fig. 17: Código de retorno da busca por um usuário.Busca de usuário sem sucesso

Ainda na tela principal, podemos chamar uma nova janela para realizar a exclusão de um usuário. Para isso basta que seja realizado um delete condicionado ao ID do usuário (Fig. 18). É importante que se tenha certeza dessa ação, pois não existe confirmação ou forma de voltar atrás da ação.

```
def queryDelete(self):
    try:

        self.connection = sqlite3.connect("./Interface PC/db/academy.db")
        self.c = self.connection.cursor()

        query = "DELETE from User WHERE  id = " + self.ui.input_id.text()

        resultQuery = self.c.execute(query)
        self.connection.commit()
        QMessageBox.information(QMessageBox(),'Successful','Deleted User {} From Table Successful'
        .format(self.ui.input_id.text()))

        self.c.close()
        self.connection.close()
        self.close()

        return resultQuery
    except Exception:
        QMessageBox.warning(QMessageBox(), 'Error', 'Could not Delete student from the database.')
```

Fig. 18: Método queryDelete: Deleta o usuário especificado e apresenta uma mensagem de confirmação ou erro.

É necessário que após a inserção, edição ou exclusão do usuário, a listagem de usuários presentes na tela principal seja atualizada. Para isso foi desenvolvido a opção para que o usuário atualize a qualquer momento a listagem. Desse modo, é de forma mais ágil foi utilizado a IDE Qt Designer. Nela foi possível realizar a construção das janelas sem que se tenha de escrever o código de fato.

3) ***Atualização via comunicação serial****:* A comunicação entre a interface computacional e o microcontrolador PIC é feita via comunicação serial. Como estamos trabalhando em um *software* de simulação do microcontrolador PIC, é necessário que se faça o uso do *software Virtual Serial Ports Emulators* (VSPE).

O programa VSPE é responsável por criar portas virtuais dentro do computador para que se possa realizar simulações de comunicação entre elas. Nessa situação temos de criar duas portas que se comunicam entre si para que possam ser utilizadas na interface computacional e no *sofware* PICSimLab (simulador do PIC).

Para iniciar a comunicação da interface com o microcontrolador é necessário que o usuário possa se conectar à uma porta. A Fig. 19 demonstra como é feito a procura por portas ativas no computador. Com a listagem de portas é possível que o usuário escolha e se conecte à porta.

```python
def loadPort(self):

    if sys.platform.startswith('win'):
        ports = ['COM%s' % (i + 1) for i in range(256)]

    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        # this excludes your current terminal "/dev/tty"
        ports = glob.glob('/dev/tty[A-Za-z]*')

    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')
    else:
        raise EnvironmentError('Unsupported platform')

    result = []
    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass

    return result
```

Fig. 19: Método loadPort: responsável por buscar as portas ativas no computador e retornar para inserção no combo box.

A conexão com a porta é feita utilizando o PySerial [9], uma biblioteca de comunicação serial em Python. Para realizar a conexão, obtemos a porta selecionada pelo usuário, definimos a velocidade, tamanho do byte e paridade dos dados, observe a Fig. 20.

```python
ser = serial.Serial('{}'.format(self.ui.comboBox_COMS.currentText()),
baudrate=9600, bytesize=8, timeout=2, stopbits=serial.STOPBITS_ONE)
```

Fig. 20: Conexão com a porta utilizada pelo usuário.

Após a conexão, pode-se enviar ou receber dados. O envio de dados para o PIC é feito a partir do carregamento de todos os dados do banco de dados, e em seguida, utiliza-se o comando write para que se escreva as informações na porta COM de comunicação, o código é apresentado na Fig. 21 .

```python
#Loading Data
self.connection = sqlite3.connect("./Interface PC/db/academy.db")
self.c = self.connection.cursor()
query = "SELECT * FROM User"
resultQuery = self.c.execute(query)

#Data: IFMTID;SENHA;STATUS;# Example: IFMT;01;0545;3#
user=""
print(enumerate(resultQuery))
for i in resultQuery:
    user+="IFMT"
    for data in i:
        user += str(data)
        user+=";"
    user = user[:-1]
    user+="#"

print(user)
ser.write(user.encode())
self.c.close()
self.connection.close()
```

Fig. 21: Busca e escrita de dados na porta serial.

A implementação do recebimento de dados e atualização do banco de dados não foi implementado.

O PIC por sua vez precisar ser habilitado a sua porta serial e a interrupção **RDA** para quando receber a mensagem na porta. Para fazer isso é preciso adicionar o **rs232**, configurar os pinos de transmissão e recepção e o cabeçalho com a função **"RDA_isr"**, conforme mostrado na Fig. 22. Para que funcione é necessário habilitar a interrupção no **main**, utilizando o **enable_interrupts** como na Fig. 23.

```
#use rs232(baud=9600,parity=N,xmit=PIN_C6,
           rcv=PIN_C7,bits=8,stream=Wireless)

#int_RDA
void RDA_isr(void){
   //Interrupção
   //Quando receber
   //Mensagem
}
```

Fig. 22: Declaração da comunicação serial (rs232) e da interrupção RDA para interromper assim que chegar uma mensagem pela porta serial.

```
enable_interrupts(INT_RDA);
enable_interrupts(GLOBAL);
```

Fig. 23: Habilitando a interrupção no PIC.

A interrupção RDA recebe apenas caracteres por vez. Dessa forma, é utilizado a função **getc** para obter cada caractere e guardá-los em uma posição do vetor **rx_buffer** na Fig. 24.

```
#int_RDA
void RDA_isr(void){
   rx_buffer[rx_wr_index] = getc();
   rxd = rx_buffer[rx_wr_index];
   rx_wr_index++;
   if(rx_wr_index > RX_BUFFER_SIZE){
      rx_wr_index = 0;
   }
```

Fig. 24: Recebimento dos caracteres via porta serial na interrupção RDA.

As informações que são transmitidas via porta serial podem ser perdidas ou dessincronizadas. Por este motivo, é criado um preâmbulo ("IFMT"), uma sequência de caracteres que deve ser respeitada antes que a verdadeira mensagem seja enviada. Na Fig. 25 é possível observar as condições confirmando essa sequência. Quando finalmente é aprovado, a posição do vetor é zerada e permite que a mensagem verdadeira seja guardada no vetor.

```
//Look for unique ID: "IFMT"
if(rxd == 'I' && lock_pos == 0){
   lock_pos++;
}
else if(rxd == 'F' && lock_pos == 1){
   lock_pos++;
}
else if(rxd == 'M' && lock_pos == 2){
   lock_pos++;
}
else if(rxd == 'T' && lock_pos == 3){
   lock_pos=0; //Reset the "combination lock"
   got_id = TRUE;
   read = rxd;
   //get ready to count the number of data bytes
   valid_data_count = 0;
   //buffer is reset to index 0
   rx_wr_index = 0;
}else {
   lock_pos = 0;
}
```

Fig. 25: Condições do preâmbulo confirmando o caractere e sua devida posição. No final se passar neste teste a mensagem está sincronizada.

O próximo passo é limitar a coleta dos caracteres até o tamanho máximo do *buffer* estabelecido. Por exemplo, no caso de esperar uma mensagem com 7 bytes, o limite é de 7 bytes e quando for permitido uma *flag* é ligada e confirma que toda a mensagem foi recebida. Isso é possível ser visto na Fig. 26.

```
if(got_id && valid_data_count++ >= BLOCK_SIZE){
   data_avail = TRUE;
   got_id = FALSE;
}
```

Fig. 26: Contagem para setar a variável que permite buscar este dado no programa.

A partir da mensagem recebida, cabe-se a criar uma função para que salve este dado na memória. Mesmo com a interrupção é recomendado esperar por este dado, assim a rotina **waitUpdate** faz exatamente isso, ela limpa a memória, e caso o dado esteja disponível (data_avail) é recebido esses dados, impresso na tela LCD e escrito na memória. Esta rotina é utilizada para receber a atualização dos dados da interface computacional para o PIC e pode ser observada na Fig. 27.

```
void waitUpdate(){
   resetMemory();
   printf (lcd_escreve,"\fEm Modo Espera");
   printf (lcd_escreve,"\r\nde dados ... ");
   delay_ms(100);
   User user;
   while(true){
      if(data_avail){
         if(rx_buffer[0] =='#' && rx_buffer[2] =='#'){
            printf (lcd_escreve,"\fFim da Comunicacao");
            delay_ms(1000);
            break;
         }
         data_avail = FALSE;
         user = receiveUser();
         printUser(user);
         overwriteUser(user);
         printf (lcd_escreve,"\fPIC em modo Espera");
         printf (lcd_escreve,"\r\nEsperando dados ... ");
         delay_ms(100);
      }
   }
}
```

Fig. 27: Rotina waitUpdate, responsável por esperar pelo dados atualizados da interface computacional e subscrever a memória.

IV. Análise e discussões

Esta seção apresenta os resultados do dos sistemas de catraca eletrônica utilizando PIC e a Interface em PyQt5 propostos.

1) ***Microcontrolador PIC****:* Devido o uso do microcontrolador no PICSimLab de forma simulada não houve tantos problemas no sentido de configuração e utilização. O sistema foi desenvolvido em partes, primeiro tentando entender como salvar e deletar da memória para assim ter a interação com o teclado, por exemplo.

Existem outras placas disponíveis no PiCSimLab, mas a utilizada neste projeto é a PICGenious que pode ser observada na Fig. 28. Os componentes como a memória EEPROM e PIC já são inclusos nesta placa, veja um zoom na Fig. 29. Finalmente, o teclado matricial precisou ser incluso externamente, é possível ver o teclado na Fig. 30.

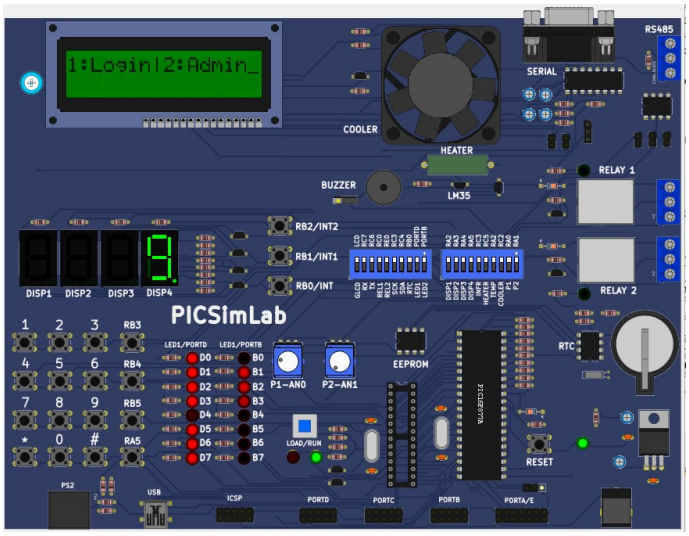


Fig. 28: Resultado do uso da placa PICGenios. O único componente que não possui na placa é o teclado matricial, assim ele é incluído separadamente como uma *spare part*.

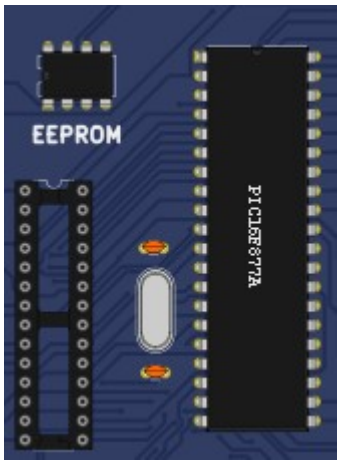


Fig. 29: Zoom na na placa PICGenios com a memória EEPROM e o PIC.

Fig. 30: Resultado do uso do teclado matricial externamente à placa PICGenios.

O sistema funciona da seguinte maneira: na primeira vez que ele é utilizado deve-se cadastrar um conta do administrador para prosseguir. Tendo ao menos um administrador cadastrado, a tela na Fig. 31 com a opção de logar ou visualizar o menu do administrador é exibida. Sempre que houver opções a mensagem é exibida e a função para a varredura do teclado é chamada e espera pela interação.

Fig. 31: Tela inicial do PIC. Ela é exibida se existir ao menos um admin cadastrado.

Ao selecionar o menu do administrador o id e a senha são requisitados, como é possível ser visto na Fig. 32 e Fig. 33, respectivamente. Caso o admin seja identificado, o menu na Fig. 34 é exibido para fazer a manutenção dos usuários no sistema.

Fig. 32: Tela para a espera dos números que compõem o ID.

Fig. 33: Tela para a espera dos números que compõem a senha.

O menu do admin na Fig. 34 possui as opções de cadastrar, buscar, deletar, editar e sair. Para testes do sistema, selecionando a opção 1 para cadastrar, foi cadastrado um usuário com o id igual à 2222, a senha igual à 1234 e o status de não pagante (0). Então, para confirmar que o usuário foi cadastrado é utilizado a função para buscar, o resultado é exibido na Fig. 35 e pode-se perceber que é exatamente o que foi cadastrado. Também há a opção de confirmar diretamente na memória EEPROM na Fig. 29, clicando sobre ela.

Fig. 34: Tela de menu do admin. Contem opções para o CRUD completo.

Fig. 35: Ao buscar um ID existente a PIC exibe no LCD os dados do ID e o status atual do usuário cadastrado.

Na opção 3 na Fig. 34, a deleção é feita apenas informando um id válido. Como teste foi adicionado o ID do usuário que acabou de ser cadastrado com o ID igual à 2222. Neste momento é buscado o id e a partir do endereço os dados são excluídos da memória, como planejado no caso 1 da função **"deleteUser()"**.

Finalmente, na opção 4 na Fig. 34 é escolhido o usuário com id igual à 3333 que foi salvo antes de iniciar o programa e a senha foi editada de 4321 para 1234. A edição é bem similar com a função de ler os dados do id, senha e status e depois o uso do **"saveUser()"**.

E ainda, saindo do menu do administrador é selecionado o login na Fig. 31. Do login é pedido o id e senha como em Fig. 32 e Fig. 33, mas a diferença é que aqui apenas liga o LED e o relé para a abertura da catraca quando o status é 1 de pagante ou 3 de administrador. Caso o status seja 0 de não pagante, o usuário deve contatar a secretaria.

*2) **Interface computacional**:* A construção da interface computacional não surgiu grandes problemas. Ao iniciar a aplicação, veremos uma tela de login conforme apresentado na Fig. 36. Essa tela de login é formada por quatro funcionalidades: realizar login administrador e usuário padrão, conectar à porta COM, enviar e receber dados.

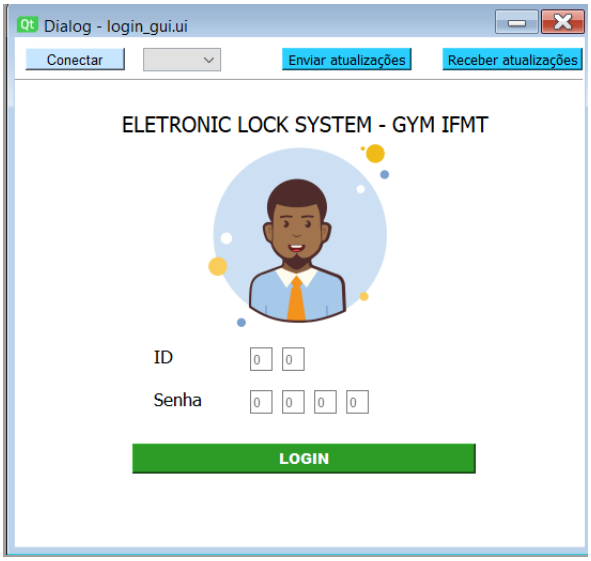


Fig. 36: Ao iniciar a aplicação obtemos a tela de login com suas funcionalides.

Ao realizar o login temos três possíveis resultado: caso seja um usuário comum, poderá aparecer que o acesso liberado ou usuário não liberado por falta de pagamento, conforme a Fig. 37. Já se o usuário for administrador, será encaminhado para uma nova janela de administração.

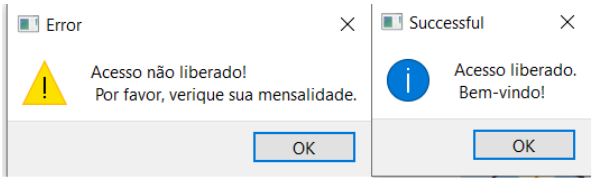


Fig. 37: Usuário comum: possíveis resultados ao realizar login.

O usuário administrador, ao realizar o login se encontrará com a janela demonstrada na Fig. 38. Nela temos algumas funcionalidades: Cadastrar, Editar, Remover, Buscar e atualizar.

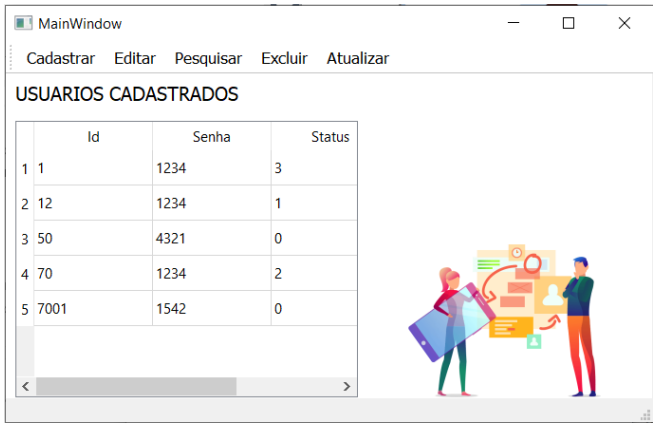


Fig. 38: Tela de gerenciamento do administrador.

Ao acessar a opção de cadastro, será obtido uma tela com todas os campos para ser preenchido, conforme apresentado na Fig. 39.

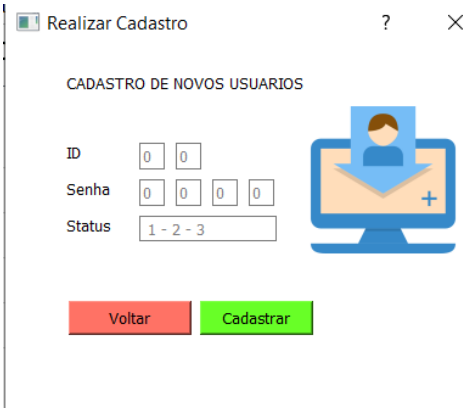


Fig. 39: Tela de cadastro de usuário.

A tela de edição do usuário apresentara um campo para que seja informado o ID, a partir desse ID é realizado uma busca e atualizado os campos contido na janela de edição (Fig.40).

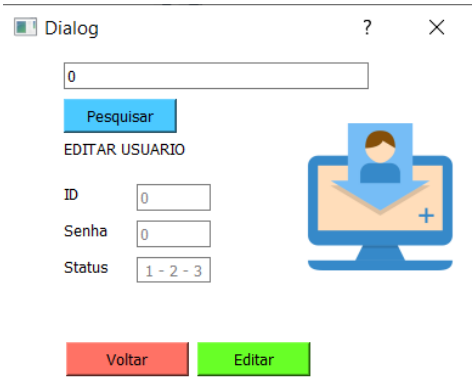


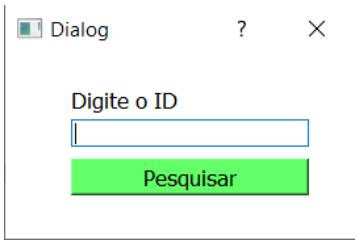
Fig. 40: Tela de edição.

A busca por um usuário é feita também através da inserção de um ID (Fig. 41a), aparecera uma tela para inserção do ID e em seguida uma mensagem informando se foi encontrado o usuário ou não conforme as figuras **??** e 41c.

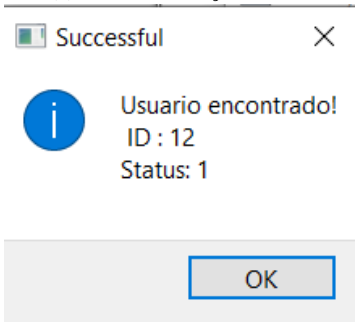
A exclusão do usuário funciona de maneira semelhante, basta apenas informar o ID e aparecerá a informação do a exclusão do usuário (Fig. 42).

O botão de atualizar tem a finalidade apenas de atualizar a listagem aparecida na tela gerenciamento do administrador. A ação é necessária sempre que realizar alguma alteração no cadastro de algum usuário.

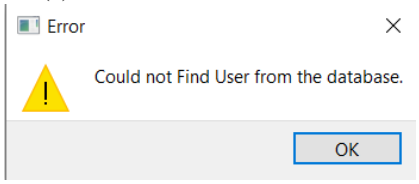
3) ***Atualização via comunicação serial****:* A interface computacional implementa a comunicação com o microcontrolador na tela de login, nela temos a opção para escolher a porta e se conectar. Após conectado é liberado as opções de envio e recebimento de dados. A opção para enviar dados foi implementada com sucesso, apresentando na porta COM que seria conectado o PIC as informações necessárias para



(a) Tela de inserção do ID

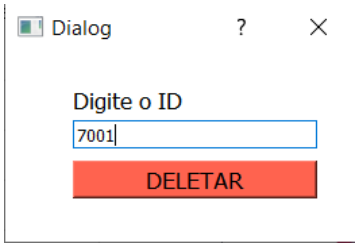


(b) Busca de usuário com sucesso

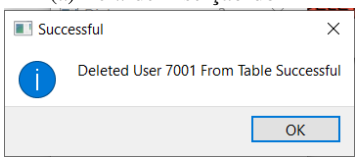


(c) Busca de usuário sem sucesso

Fig. 41: Tela de busca por usuário.



(a) Tela de inserção do ID



(b) Informação de confirmação de exclusão do usuário

Fig. 42: Tela para deletar usuário.

atualizar os dados. Entretanto o recebimento de dados não foi implementado, pois não houve implementação do envio de dados no microcontrolador.

No PIC a comunicação serial em conjunto com a interrupção RDA funcionou parcialmente. Ao receber o dado todo junto a mensagem é recebida tranquilamente. No entanto, o id deve ser no máximo 9999, sendo cada dois dígitos guardado em uma posição diferente. Além disso os dados tiveram que ser convertidos de ANSCII para inteiro e assim respeitar o padrão estabelecido no sistema.

No final na comunicação do PIC, tentou-se criar maneiras para concatenar e separar os números em pares de dígitos no id criando mais complexidades em outros dados como a senha e o status que não seguiam essa regra. O resultado não foi bem sucedido como planejado, fazendo com que essa funcionalidade no PIC não fosse implementada corretamente na prática.

V. Conclusão

A tecnologia surgiu para melhorar a qualidade de vida da humanidade. Supriu as necessidades em diversos setores importante, tais como, comunicação, saúde, segurança e empresas. O controle das informações é fundamental em estabelecimentos, e a eletrônica nos dispõe de tais tecnologias. Desse modo, foi desenvolvido um sistema de catraca gerenciado pelo microcontrolador PIC e interface computacional PyQt5.

O sistema de catraca eletrônica apresentado cumpriu seu propósito de realizar a manutenção completa e controle de acesso dos usuários.Faltando apenas a comunicação via serial para a atualização dos dados entre eles, que ainda sim foi feita parcialmente de um lado dos sistemas. Portanto, foi alcançado cerca de 85% do planejamento do projeto. Como proposta de continuação, seria importante finalizar a parte de comunicação e inserir novas funções para facilitar a interação do usuário.

References

[1] J. S. Marcondes. (2021) Catraca Controle Acesso: O que é? Objetivos, tipos, como funciona. [Online]. Available: https://gestaodesegurancaprivada.com.br/catraca-controle-acesso-o-que-e-objetivos-tipos-como-funciona/

[2] ALLDATASHEET. (2021) PIC6F87XA Data Sheet. [Online]. Available: https://pdf1.alldatasheet.com/datasheet-pdf/view/82338/MICROCHIP/PIC16F877A.html.

[3] L. C. G. Lopes. (2021) PICSimLab. [Online]. Available: https://lcgamboa.github.io/

[4] t. Qt Company.

[5] R. Computing. (2021) What is PyQt? [Online]. Available: https://riverbankcomputing.com/software/pyqt/intro

[6] D. R. Hipp. (2000) About SQLite. [Online]. Available: https://www.sqlite.org/about.html

[7] L. C. G. Lopes. (2021) PICGenios. [Online]. Available: https://lcgamboa.github.io/picsimlab_docs/stable/PICGenios.html#x22-210004.5

[8] G. André and M. Danubia, "Repositório: Eletronic Lock System," 2021. [Online]. Available: https://github.com/DanubiaM/eletronic-lock-system

[9] Pyserial. (2020) pySerial's documentation. [Online]. Available: https://help.figma.com/hc/en-us/articles/1500004362321-Guide-to-FigJam