#!/usr/bin/env python

# coding=UTF-8

import sys

try:

import pygtk

pygtk.require("2.4")

except:

pass

try:

import gobject

except:

sys.exit(1)

import gtk

import pyfprint

import thread

import time

import os

import threading

import MySQLdb

from gettext import gettext as \_

import locale

import gettext

import pyfprint\_swig as pyf

from modules.user import \*

from modules.configuration import \*

from modules.attendance import \*

from modules.event import EventSection

from modules.event import DBManager as EventDB

from modules.report import \*

APP\_NAME = "attendance\_system"

class SplashScreen():

def delete\_event(self, widget, event, data=None):

return False

def \_\_init\_\_(self):

self.window = gtk.Window(gtk.WINDOW\_TOPLEVEL)

self.window.set\_decorated(False)

self.window.set\_position(gtk.WIN\_POS\_CENTER)

self.window.set\_modal(True)

self.window.connect("delete\_event", self.delete\_event)

vbox = gtk.VBox(False,0)

i\_splash = gtk.Image()

i\_splash.set\_from\_file("icons/splash.jpg")

vbox.pack\_start(i\_splash,True,False,5)

self.\_\_progress\_fraction = 0

self.\_\_progressbar = gtk.ProgressBar()

self.\_\_progressbar.set\_fraction(self.\_\_progress\_fraction)

vbox.pack\_start(self.\_\_progressbar,True,False,5)

vbox.show\_all()

self.window.add(vbox)

self.window.show\_all()

def close(self):

self.window.destroy()

def update\_progressbar(self, message):

self.\_\_progress\_fraction += 0.25

self.\_\_progressbar.set\_fraction(self.\_\_progress\_fraction)

self.\_\_progressbar.set\_text(message)

while gtk.events\_pending(): gtk.main\_iteration()

class AttendanceSystem:

def delete\_event(self, widget, event, data=None):

if self.is\_admin or self.admin\_ok(title=\_("EXIT THE APPLICATION")):

return False

else:

return True

# Another callback

def destroy(self, widget, data=None):

try:

if self.dev != None:

self.dev.close()

pyfprint.fp\_exit()

self.\_\_user\_section.close()

except:

pass

try:

self.\_\_attendance\_section.close()

except:

pass

try:

self.\_\_cursor.close()

self.\_\_db.close()

except:

pass

try:

os.remove(TMP\_USER\_IMAGE\_FILE\_NAME)

except:

pass

gtk.main\_quit()

def \_\_init\_\_(self, finger\_print\_device, database, db\_cursor,\

config\_manager, config\_param):

self.window = gtk.Window(gtk.WINDOW\_TOPLEVEL)

self.window.connect("delete\_event", self.delete\_event)

self.window.connect("destroy", self.destroy)

self.window.set\_title(\_("Assistance Control System "))

self.window.set\_default\_size(800,600)

self.window.set\_position(gtk.WIN\_POS\_CENTER)

self.window.set\_border\_width(5)

self.window.set\_icon\_from\_file("icons/clock.png")

self.is\_admin = False

self.\_\_user\_section = None

self.\_\_config\_manager = config\_manager

self.\_\_config\_param = config\_param

self.\_\_db = database

self.\_\_cursor = db\_cursor

self.dev = finger\_print\_device

self.admin\_password = self.\_\_config\_param["admin"]["password"]

self.\_\_attendance\_section = None

main\_box = gtk.VBox(False, 0)

self.window.add(main\_box)

#TOP

self.top\_box = gtk.VBox(False, 0)

main\_box.pack\_start(self.top\_box,False,False,0)

self.create\_main\_toolbar()

self.top\_box.pack\_start(self.main\_toolbar,True,False,0)

separator = gtk.HSeparator()

self.top\_box.pack\_start(separator,True,False,0)

self.middle\_box = gtk.VBox(False, 0)

self.bottom\_box = gtk.VBox(False, 0)

main\_box.pack\_start(self.middle\_box,True,True,0)

main\_box.pack\_start(self.bottom\_box,False,False,0)

#MIDDLE

choose\_label = gtk.Label(\_("Choose one of the following options "))

self.middle\_box.pack\_start(choose\_label,True,False)

#BOTTOM

separator = gtk.HSeparator()

self.bottom\_box.pack\_start(separator,True,False,0)

b\_url = gtk.LinkButton("http://www.lemontruck.com", label=\_("Desarrollado por Lemontruck"))

self.bottom\_box.pack\_start(b\_url,True,False,0)

self.window.show\_all()

def admin\_ok(self, title=\_("Administrador")):

dialog = gtk.Dialog(title, self.window, \

gtk.DIALOG\_MODAL, (gtk.STOCK\_OK, gtk.RESPONSE\_ACCEPT, \

gtk.STOCK\_CANCEL, gtk.RESPONSE\_REJECT))

dialog.set\_default\_response(gtk.RESPONSE\_ACCEPT)

dialog.set\_resizable(False)

label = gtk.Label(\_("Enter the Administrator password "))

dialog.vbox.pack\_start(label,False,False,5)

e\_password = gtk.Entry()

e\_password.set\_visibility(False)

dialog.vbox.pack\_start(e\_password,False,False,10)

dialog.vbox.show\_all()

response = dialog.run()

r\_password = e\_password.get\_text()

dialog.destroy()

if response == gtk.RESPONSE\_ACCEPT:

if r\_password == self.admin\_password:

self.is\_admin = True

return True

else:

e\_dialog = gtk.MessageDialog(self.window, gtk.DIALOG\_MODAL, gtk.MESSAGE\_ERROR, \

gtk.BUTTONS\_OK, message\_format=\_(" Incorrect password!"))

response = e\_dialog.run()

e\_dialog.destroy()

return False

def exit\_cb(self, button):

if self.is\_admin or self.admin\_ok(title=\_(" Exit the application ")):

if self.\_\_user\_section != None:

if not self.\_\_user\_section.unsave\_datas():

self.destroy(button)

else:

q\_dialog = gtk.MessageDialog(self.window, gtk.DIALOG\_MODAL, \

gtk.MESSAGE\_QUESTION, gtk.BUTTONS\_OK\_CANCEL, \

" There are data without saving, if you leave the application data will be lost, do you want to continue?")

q\_dialog.set\_title(\_("Assistance Control System "))

response = q\_dialog.run()

if response == gtk.RESPONSE\_OK:

self.destroy(button)

q\_dialog.destroy()

else:

self.destroy(button)

def \_show\_error\_dialog(self, message=""):

e\_dialog = gtk.MessageDialog(self.window, gtk.DIALOG\_MODAL, \

gtk.MESSAGE\_ERROR, gtk.BUTTONS\_OK, message)

e\_dialog.set\_title(\_("Assistance Control System "))

e\_dialog.run()

e\_dialog.destroy()

def \_users\_cb(self, button):

self.remove\_widgets\_middle\_box()

self.\_\_user\_section = UserSection(self.window, self.middle\_box,\

self.\_\_config\_manager.get\_configurations(),\

self.\_\_cursor, self.dev)

def \_configurations\_cb(self, button):

self.remove\_widgets\_middle\_box()

ConfigSection(self.window, self.middle\_box, self.\_\_config\_manager,\

self.dev)

def \_events\_cb(self, button):

self.remove\_widgets\_middle\_box()

EventSection(self.window, self.middle\_box, self.\_\_config\_manager.get\_configurations(),\

self.\_\_cursor)

def \_reports\_cb(self, button):

self.remove\_widgets\_middle\_box()

ReportSection(self.window, self.middle\_box, self.\_\_config\_manager.get\_configurations(), \

self.\_\_cursor)

def admin\_cb(self, button):

if self.is\_admin or self.admin\_ok():

if (self.\_\_attendance\_section):

self.\_\_attendance\_section.close()

button.grab\_focus()

self.remove\_widgets\_middle\_box()

table = gtk.Table(rows=4, columns=2, homogeneous=True)

table.set\_row\_spacings(10)

admin\_box = gtk.HBox(False,0)

self.\_\_config\_param = self.\_\_config\_manager.get\_configurations()

self.b\_add\_user = gtk.Button(\_("Usuarios"))

self.b\_add\_user.connect('clicked', self.\_users\_cb)

self.b\_events = gtk.Button(\_("Eventos"))

self.b\_events.connect('clicked', self.\_events\_cb)

self.b\_report = gtk.Button(\_("Informes"))

self.b\_report.connect('clicked', self.\_reports\_cb)

self.b\_config = gtk.Button(\_("Configuraciones"))

self.b\_config.connect('clicked', self.\_configurations\_cb)

table.attach(self.b\_add\_user,1,2,0,1)

if self.\_\_config\_param["app"]["events\_attendance"] == "True":

table.attach(self.b\_events,1,2,1,2)

table.attach(self.b\_report,1,2,2,3)

table.attach(self.b\_config,1,2,3,4)

else:

table.attach(self.b\_report,1,2,1,2)

table.attach(self.b\_config,1,2,2,3)

i\_add\_user = gtk.Image()

i\_add\_user.set\_from\_file("icons/people.png")

i\_report = gtk.Image()

i\_report.set\_from\_file("icons/phonebook.png")

i\_config = gtk.Image()

i\_config.set\_from\_file("icons/build.png")

i\_events = gtk.Image()

i\_events.set\_from\_file("icons/calendar.png")

table.attach(i\_add\_user,0,1,0,1)

if self.\_\_config\_param["app"]["events\_attendance"] == "True":

table.attach(i\_events,0,1,1,2)

table.attach(i\_report,0,1,2,3)

table.attach(i\_config,0,1,3,4)

else:

table.attach(i\_report,0,1,1,2)

table.attach(i\_config,0,1,2,3)

admin\_box.pack\_start(table,True,False,0)

admin\_box.show\_all()

self.middle\_box.pack\_start(admin\_box,True,False,0)

self.b\_add\_user.grab\_focus()

def \_\_show\_select\_event\_window(self):

dialog = gtk.Dialog("Assistance Control System ", self.window,\

gtk.DIALOG\_MODAL, (gtk.STOCK\_OK, gtk.RESPONSE\_OK, \

gtk.STOCK\_CANCEL, gtk.RESPONSE\_CANCEL))

dialog.set\_default\_size(640,480)

dialog.vbox.pack\_start (gtk.Label(\_("Select an event and press Ok ")), False, False, 5)

#Search Options

search\_vbox = gtk.VBox()

frame\_search = gtk.Frame(\_("Search "))

frame\_search.add(search\_vbox)

dialog.vbox.pack\_start(frame\_search,False, False, 0)

event\_search\_entry = gtk.Entry()

event\_search\_entry.connect("activate", self.\_\_search\_event\_cb, event\_search\_entry)

search\_vbox.pack\_start(event\_search\_entry,False,False, 0)

search\_buttons\_hbox = gtk.HBox()

search\_vbox.pack\_start(search\_buttons\_hbox,False,False, 0)

b\_search = gtk.Button(\_("Buscar"))

b\_search.connect("clicked", self.\_\_search\_event\_cb, event\_search\_entry)

search\_buttons\_hbox.pack\_start(b\_search,False,False, 0)

b\_clean = gtk.Button(\_("Clean "))

b\_clean.connect("clicked", self.\_\_fill\_events\_list, event\_search\_entry)

search\_buttons\_hbox.pack\_start(b\_clean,False,False, 0)

#List

self.\_\_events\_list = gtk.ListStore(int,str)

#Row container

events\_treeview = gtk.TreeView()

# Container Model

events\_treeview.set\_model(self.\_\_events\_list)

#Add column to list

events\_treeview.append\_column(gtk.TreeViewColumn(\_("First name "), gtk.CellRendererText(), text=1))

scroll = gtk.ScrolledWindow()

scroll.add(events\_treeview)

self.\_\_fill\_events\_list()

dialog.vbox.pack\_start (scroll, True, True, 0)

dialog.show\_all()

result = dialog.run()

if result == gtk.RESPONSE\_OK:

model, itera = events\_treeview.get\_selection().get\_selected()

if not itera == None:

event\_id = model.get\_value(itera,0)

dialog.destroy()

return event\_id

else:

dialog.destroy()

return False

else:

dialog.destroy()

return -1

def \_\_search\_event\_cb(self, widget, entry):

m\_events = EventDB(self.\_\_cursor)

events = m\_events.search\_by\_name(entry.get\_text())

self.\_\_events\_list.clear()

for event in events:

self.\_\_events\_list.append((event["id"],event["name"]))

def \_\_fill\_events\_list(self, widget=None, entry=None):

self.\_\_events\_list.clear()

if not entry == None:

entry.set\_text("")

m\_events = EventDB(self.\_\_cursor)

events = m\_events.get\_events()

for event in events:

self.\_\_events\_list.append((event["id"],event["name"]))

def attendance\_cb(self, button):

if (self.\_\_user\_section):

self.\_\_user\_section.close()

button.grab\_focus()

self.is\_admin = False

event\_id = -1

self.remove\_widgets\_middle\_box()

if self.\_\_config\_param["app"]["events\_attendance"] == "True":

event\_id = False

while not event\_id:

event\_id = self.\_\_show\_select\_event\_window()

self.\_\_attendance\_section = AttendanceSection(self.window, self.middle\_box, self.\_\_cursor,\

self.\_\_config\_manager, self.dev, event\_id)

else:

self.\_\_attendance\_section = AttendanceSection(self.window, self.middle\_box, self.\_\_cursor,\

self.\_\_config\_manager, self.dev, event\_id)

def create\_main\_toolbar(self):

self.main\_toolbar = gtk.Toolbar()

self.main\_toolbar.set\_orientation(gtk.ORIENTATION\_HORIZONTAL)

self.main\_toolbar.set\_style(gtk.TOOLBAR\_BOTH)

accel\_group = gtk.AccelGroup()

self.window.add\_accel\_group(accel\_group)

#Attendance

self.tb\_attendance = gtk.ToolButton(label=\_(" Assistance "))

self.tb\_attendance.set\_expand(True)

self.tb\_attendance.connect('clicked', self.attendance\_cb)

self.tb\_attendance.set\_flags(gtk.CAN\_FOCUS)

self.tb\_attendance.set\_tooltip\_text(\_("Control Assistance (Ctrl+A)"))

self.tb\_attendance.add\_accelerator('clicked', accel\_group, ord('a'), \

gtk.gdk.CONTROL\_MASK, gtk.ACCEL\_VISIBLE)

i\_attendance = gtk.Image()

i\_attendance.set\_from\_file("icons/pen.png")

self.tb\_attendance.set\_icon\_widget(i\_attendance)

self.main\_toolbar.insert(self.tb\_attendance, -1)

separator = gtk.SeparatorToolItem()

self.main\_toolbar.insert(separator, -1)

#Admin Options

self.tb\_config = gtk.ToolButton(label=\_(" Administration "))

self.tb\_config.set\_expand(True)

self.tb\_config.connect('clicked', self.admin\_cb)

self.tb\_config.set\_tooltip\_text(\_("Configure the Application (Ctrl+D)"))

self.tb\_config.set\_flags(gtk.CAN\_FOCUS)

self.tb\_config.add\_accelerator('clicked', accel\_group, ord('d'), \

gtk.gdk.CONTROL\_MASK, gtk.ACCEL\_VISIBLE)

i\_config = gtk.Image()

i\_config.set\_from\_file("icons/interact.png")

self.tb\_config.set\_icon\_widget(i\_config)

self.main\_toolbar.insert(self.tb\_config, -1)

separator = gtk.SeparatorToolItem()

self.main\_toolbar.insert(separator, -1)

#Exit

self.tb\_exit = gtk.ToolButton(label=\_(" Exit "))

self.tb\_exit.connect('clicked', self.exit\_cb)

self.tb\_exit.set\_expand(True)

self.tb\_exit.set\_tooltip\_text(\_("Exit the application (Ctrl+E)"))

self.tb\_exit.set\_flags(gtk.CAN\_FOCUS)

self.tb\_exit.add\_accelerator('clicked', accel\_group, ord('e'), \

gtk.gdk.CONTROL\_MASK, gtk.ACCEL\_VISIBLE)

i\_exit = gtk.Image()

i\_exit.set\_from\_file("icons/arrow.png")

self.tb\_exit.set\_icon\_widget(i\_exit)

self.main\_toolbar.insert(self.tb\_exit, -1)

def remove\_widgets\_middle\_box(self):

middle\_widgets = self.middle\_box.get\_children()

for widget in middle\_widgets:

self.middle\_box.remove(widget)

class SetupApplication():

#English and Spanish languages support

def \_\_set\_up\_translation(self):

#Get the local directory

local\_path = os.path.join(os.path.dirname(sys.argv[0]),"locale")

#Check the default locale

lc, encoding = locale.getdefaultlocale()

#Set the defaul locale

lang = ["en\_US"]

if lc and lc.find("es") != -1:

lang = ["es\_ES"]

gettext.bindtextdomain(APP\_NAME, local\_path)

gettext.textdomain(APP\_NAME)

set\_lang = gettext.translation(APP\_NAME, local\_path, languages=lang)

set\_lang.install()

\_ = set\_lang.gettext

def \_\_init\_\_(self, splash\_screen):

self.\_\_set\_up\_translation()

splash\_screen.update\_progressbar(\_("Cargando las configuraciones..."))

self.\_\_config\_manager = ConfigurationManager()

self.\_\_config\_manager.load\_configurations()

self.\_\_config\_param = self.\_\_config\_manager.get\_configurations()

self.\_\_setup\_errors = {"database":False, "mdb":"", "fingerprint":False, "mfp":"",\

"config":False, "mcfg":""}

def start(self, splash\_screen):

if self.\_\_config\_param != None:

splash\_screen.update\_progressbar(\_("Starting Database..."))

ok\_db = self.\_\_init\_database(self.\_\_config\_param["database"])

if ok\_db:

splash\_screen.update\_progressbar(\_("Searching Fingerprint Device..."))

self.\_\_init\_finger\_print\_device()

if self.\_\_config\_param["app"]["attendance\_by\_finger\_print"] == "True":

if self.\_\_dev != None:

splash\_screen.update\_progressbar(\_("Initiating Fingerprint Device..."))

self.\_\_dev.open()

else:

return self.\_\_setup\_errors

else:

self.\_\_setup\_errors["config"] = True

self.\_\_setup\_errors["mcfg"] = \_(" There was a problem when trying to access the settings, the application will close")

return self.\_\_setup\_errors

def \_\_init\_database(self, db\_param):

return self.\_\_connect\_db(db\_param)

def \_\_init\_finger\_print\_device(self):

try:

pyfprint.fp\_init()

self.\_\_dev = pyfprint.discover\_devices()[0]

except:

self.\_\_dev = None

def \_\_open\_finger\_print\_device(self):

try:

self.\_\_dev.open()

except:

self.\_\_setup\_errors["fingerprint"] = True

self.\_\_setup\_errors["mfp"] = \_(" There was a problem when trying to start the fingerprint device, the application will not work correctly ")

def \_\_connect\_db(self, db\_param):

try:

self.\_\_db = MySQLdb.connect(host=db\_param["host"], user=db\_param["user"],\

passwd=db\_param["password"], db=db\_param["name"])

self.\_\_cursor = self.\_\_db.cursor()

return True

except MySQLdb.Error, e:

self.\_\_setup\_errors["database"] = True

self.\_\_setup\_errors["mdb"] = \_(" There was a problem when trying to connect to the Database, the application will close")

return False

def get\_finger\_print\_dev(self):

return self.\_\_dev

def get\_db(self):

return self.\_\_db, self.\_\_cursor

def get\_config(self):

return self.\_\_config\_manager, self.\_\_config\_param

def show\_error\_dialog(parent\_window, message):

e\_dialog = gtk.MessageDialog(parent\_window, gtk.DIALOG\_MODAL, \

gtk.MESSAGE\_ERROR, gtk.BUTTONS\_OK, message)

e\_dialog.set\_title(\_("Assistance Control System "))

e\_dialog.run()

e\_dialog.destroy()

def main():

gtk.main()

if \_\_name\_\_ == "\_\_main\_\_":

initial\_screen = SplashScreen()

while gtk.events\_pending(): gtk.main\_iteration()

setup = SetupApplication(initial\_screen)

error\_setup = setup.start(initial\_screen)

initial\_screen.close()

while gtk.events\_pending(): gtk.main\_iteration()

if error\_setup["config"]:

show\_error\_dialog(initial\_screen.window, error\_setup["mcfg"])

elif error\_setup["database"]:

show\_error\_dialog(initial\_screen.window, error\_setup["mdb"])

else:

gtk.gdk.threads\_init()

finger\_print\_device = setup.get\_finger\_print\_dev()

database, db\_cursor = setup.get\_db()

config\_manager, config\_param = setup.get\_config()

AttendanceSystem(finger\_print\_device, database, db\_cursor,\

config\_manager, config\_param)

main()