Instalar na máquina

https://ubuntu-mate.org/raspberry-pi/

formato do dicionário de contexto

{
app:{window_id:444444, entreprise: 'bla bla', logotipo: 'bla.png', outras informações globais},
history:{1:(obj1, list), 2: (obj1, edit), 3:(obj2, edit), pilha com o caminho percorrido pelo utilizador},
actual_form:1, formulário em que o utilizador está agora
obj1:{
app:{name:'obj1', title:'Object 1', outras informações do objecto}
list:{selected_ids:[], outras informações do objecto quando apresentado neste tipo de formulário}
edit:{key:'34324242423', list_field_1:{selected_ids:[], name:'obj_da_list_field'},
}
calendar:{}
popup:{}
}
obj2:{
}
}

Como definir um novo Objecto

```
# !/usr/bin/env python3
# -*- encoding: utf-8 -*-
"""
ERP+
"""
__author__ = ''
__credits__ = []
__version__ = "1.0"
__maintainer__ = ""
__status__ = "Development"
__model_name__='terceiro.Terceiro'
        a primeira parte é o nome em minusculas com underscore a dividir as palavras, é util

import base_models
        Só tenho que importar o base_models se quiser utilizar sequencias para documentação
from orm import *
from form import *
```

```
Qualquer módulo (objecto) que necessite de utilizar no ficheiro a criar deverá ser importado
try:
    from my_plano_contas import PlanoContas
except:
    from plano_contas import PlanoContas
Desta forma garantimos que se alguem quiser modificar o objecto para o seu uso, o objecto mo

a classe é iniciada assim:
class Terceiro(Model, View):
    def __init__(self, --kargs):
        Model.__init__(self, --kargs)
        self.__model_name__ = __model_name__

e se for uma classe herdada, ou seja de um objecto herdado:

from terceiro import Terceiro as TerceiroOriginal
class Terceiro(TerceiroOriginal):
    def __init__(self, --kargs):
        TerceiroOriginal.__init__(self, --kargs)
```

Propriedades do Objecto

self.name = ”

```
  Primeira parte do model_name
```

self.title = ”

```
  Titulo do formulário
```

self.list_edit_mode = ‘edit’

```
  Define o tipo de formulário, pode ser 'inline', 'edit' ou 'popup'
```

self.on_create_sql = ”

```
    Sempre que for necessário correr um determinado sql após a criação da tabela
```

self.__order_by__ = ''

```
    Define a ordem pela qual os registos são apresentados nas listas, podem ser varios campo
```

self.workflow = ()

```
Define o Workflow
Ex:
 self.__workflow__ = (
            'estado', {
                'Rascunho':['Activar', 'Gera Periodos'],
                'Activo':['Encerrar', 'Cancelar'],
                'Encerrado':['Cancelar'],
                'Cancelado':['Rascunho']
                }
            )
```

self.workflow_auth = {}

```
Gestão de Permissões para o workflow. Só aparecem os botões se o utilizador tiver autori
Ex:
    self.__workflow_auth__ = {
            'Gera Periodos':['Contabilista'],
            'Activar':['Contabilista'],
            'Encerrar':['Contabilista'],
            'Rascunho':['Gestor'],
            'Cancelar':['Gestor'],
            'full_access':['Gestor']
            }
```

self.workflow_context = {}

```
Utilizado para inibir botões no workflow se determinada condição não for cumprida
```

self.records_view = []

```
Utilizado para mostrar ou inibir a visualização de registos mediante determinada condiçã
Inibe todos os registos cujo campo (1º valor da lista), for igual a um dos valores do tu
desde que não tenham uma das funções (perfis) da lista(3º valor da lista) (auth) .
o 4º valor da lista pode ser 'in' ou 'not in' dependendo se é para mostrar os registos c
Ex:
self.__records_view__ = [
    ('estado', ('Pago', 'Validado'), ['Gestor'], 'not in'),
    ]
self.__records_view__ = [
    ('estado', 'get_records_view_values()', ['Gestor'], 'in'),
    ]
```

```
def get_records_view_values(self):
    return('Teste', 'bla')
```

self.tabs = []

```
Permite organizar os campos em tabs sempre que o formulário for do tipo edit
```

self.no_edit = []

```
Não permite editar quando um determinado campo tiver os valores da lista
Ex:
    self.__no_edit__ = [('estado', ['Encerrado','Cancelado'])]
```

self.get_options = []

```
Esta opção permite definir o campo ou os campos que servirão para enviar para as combobo
```

self.join_columns = {}

```
Esta opção serve para definir que colunas deveremos considerar em caso de JOIN depois me
```

self.record_colors = []

```
Esta opção serve para colorir os registos conçoante o valor de um determinado campo ou c
[('estado',{'Novo':'Black', 'Enviado':'Red'})]
```

self.force_db = False

```
No caso de eu fazer algo a partir da lista utilizando o sidebar tenho que definir o __fo
```

self.db_mode = 'Table'

```
pode ser 'Table', 'View' ou 'None' e serve para definir o comportamento em relação à bas
```

self.auth = {}

```
Define ue funções tem que direitos no Modelo, não necessita de pôr "Administrator" pois
Ex:
    self.__auth__ = {
        'read':['All'],
        'write':['All'],
        'create':['All'],
        'delete':['All'],
        'full_access':['all']
    }
```

self.side_menu = []

define um menu lateral que facilita a navegação no objecto quando em modo lista, principalmente quando tem hierarquias
Ex:
self.**side_menu** = [‘menu1’, ‘menu2’, ‘menu3’, ‘menu4’]

self.breadcrumbs_navigation = False

permite a navegação “migalhas de pão” ou seja uma barra superior que apresenta o caminho em que navegamos num objecto que tem hierarquia

Opções dos campos

unique

```
Falta implementar isto nos campos para sempre que não queira-mos valores duplicados!
```

size

```
Define o tamanhõ do campo no formulário
Ex:
    size = 60
```

view_order

```
Define a ordem de apresentação do campo no formulário
Ex:
    view_order=1
```

name

```
Nome do Campo, é utilizado na Label no formulário
Ex:
    name='Nome'
```

value

args

```
Qualquer argumento do html5 que seja aceite pelo input que o tipo de field cria
Ex:
    args = 'required'
    args = "rows=20" utilizado no text_field para garantir o numero de linhas desejado
    args = 'autocomplete="on"'
    args = 'style:visibility="hidden"'
```

edit_ok

options

```
Permite definir opções num combo_field, pode ser uma lista de tuples fixa ou uma função que
Ex:
    options = [('activo','Activo'), ('cancelado','Cancelado')]
    options = 'model.get_terminal()'
```

parent

source

nolabel

onlist

search

```
Permite activar ou não a possibilidade de fazer pesquisas pelos valores deste campo, se o va
Ex:
    search = False
```

default

```
Permite definir um valor por defeito
Ex:
    default = datetime.date.today()
    default = time.strftime('%H:%M')
    default = "session['user']"
    default = 'Aberta'
```

onchange

dynamic_atrs

Quando o valor de um campo muda muda atributos html5 nos widgets do formulário

ex: self.estado = info_field(view_order=3, name='Estado', size=40, default='Rascunho', dynamic_atrs = 'teste_dynamic_atrs')

def teste_dynamic_atrs(self, key, window_id): # teriamos que mudar aqui as outras funções que recebem record
print('estou no teste_dynamic_atrs')
import json
bottle.response.content_type = 'application/json'
result = {'sql':'readonly', 'totais':'readonly'}
#print(result)
return json.dumps(result)

field_name

model

```
Utilizado no combo_field, no choice_field e no parent_field para definir o modelo (objecto)
Ex:
    model = 'terminal'
```

model_name

```
Utilizado no list_field para definir o modelo da relação(objecto)
Ex:
    model_name = 'linha_caixa.LinhaCaixa'
```

condition

```
Utilizado no list_field para definir a condição (Where) que permite definir que registos da
Ex:
    condition = "caixa = '{id}'"
    condition = "documento='consumo' and num_doc={numero}"
```

fields

```
Utilizado nos list_field e nos many2many para restringir os campos que devem aparecer na lis
Ex:
    fields = ['nome', 'rede', 'diametro', 'comprimento']
```

show_footer

```
Utilizado nos list_field para optar se a lista deve ter um footer ou não
Ex:
    show_footer = False
```

sum

```
Se colocado numa coluna que devolve valores numéricos, no rodapé das listas aparece o somató
Ex:
    sum = True
```

hidden

column

```
Utilizado no parent_field e no combo_field para definir a coluna da relação
Ex:
    column = 'nome'
```

list_edit_mode

```
Utilizado nos list_field para definir como deve o registo ser editado, pode ser de um dos ti
Ex:
    list_edit_mode='inline')
```

simple

```
Utilizado nos list_field para definir se o design da lista deve ser normal ou simplificado
Ex:
    simple=True
```

Campos

separator

```
Gera um separador no formulário
```

new_line

```
Obriga o próximo campo a aparecer na próxima linha
```

many2many

```
Ex:
    self.reservatorio = many2many(view_order=8, name='Reservatórios', fields=['nome'], model
```

list_field

```
Ex:
    self.contacto = list_field(view_order = 12, name = 'Contactos', condition = "terceiro =
    self.ocorrencia = list_field(view_order=9, name ='Ocorrências', fields=['numero', 'data_
    list_field(view_order=5 , name='Estação de Bombagem', fields=['nome'], simple=True, show
```

function_field

```
Define campos cujo valor é devolvido por uma função
Ex:
    self.total = function_field(view_order = 10, name = 'Total Em Caixa', size = 20, sum = T
```

string_field

```
Ex:
    self.nome = string_field(view_order = , name = '', args = '', size = )
```

email_field

Um input do tipo email

image_field

Ex:

```
    image_field(view_order =, name = '', size = 100, onlist = False)
```

password_field

Um input onde os caracteres introduzidos são substituidos da forma como é definido na norma

combo_field

Ex:

```
    self.estado = combo_field(view_order = 3, name = 'Estado', size = 40, default = 'activo'
```

choice_field

Ex:

```
    self.a_receber = choice_field(view_order = 6, name = 'A Receber', size = 80, model = 'pl
    self.contador_in = choice_field(view_order=5 , name='Cont.Entrada', size=60, model='cont
```

parent_field

Ex:

```
    parent_field(view_order=1 , name='Terceiro', hidden=True, nolabel=True, onlist=False, mo
```

boolean_field

Ex:

```
    self.cliente = boolean_field(view_order = 8, name = 'Cliente?', default = True)
```

integer_field

Input do tipo numérico inteiro
Ex:

```
    self.num_doc = integer_field(view_order = 4, name = 'Número Documento', args = 'required
```

float_field

```
Input do tipo numérico decimal binário
```

currency_field

```
Input para ser utilizado em campo que emvolvem valores monetários
Ex:
    self.credito = currency_field(view_order = 5, name = 'Crédito', size = 50)
```

percent_field

```
Input para percentagens
Ex:
    self.desconto = percent_field(view_order = 4, name = 'Desconto', size = 50)
```

decimal_field

```
Input para valores numéricos decimais não binários
Ex:
    self.capacidade = decimal_field(view_order=4 , name='Capacidade M3', size=20)
```

date_field

```
Input para introdução de datas, por enquato só o chrome o suporta completamente mas sendo um
Ex:
    self.data_inicial = date_field(view_order=3, name='Data Inicial', size=60, args='require
```

time_field

```
Input para introdução de horas, por enquato só o chrome o suporta completamente mas sendo um
Ex:
    self.hora_inicial = time_field(view_order=5, name='Hora Inicial', size=60, args='require
```

text_field

```
Input para introdução de texto corrido, em breve terei que colocar em pé o WYSIWYG para util
Ex:
    self.descricao = text_field(view_order=8, name='Descrição', size=100, args="rows=20", or
```

message_field

```
Previsto para o Email
```

info_field

```
Campo que simplesmente apresenta informação mas não permite escrita
Ex:
    self.numero = info_field(view_order=1, name='Número', size=30)
    self.estado = info_field(view_order=7, name ='Estado', hidden=True, nolabel=True, defaul
```

Funções habituais utilizadas no ERP+

Função para devolver os valores para o campo options de um combo_field ou choice_field

```
Ex:
    (tem que importar o objecto antes de usar esta função)
    def get_terminal(self):
        return Terminal().get_options()

    def get_opts(self, model):
        return eval(model + '().get_options()')
```

Função para devolver o valor de um function_field

```
O nome deve ser "get_" seguido do nome do campo
Ex:
    def record_lines(self, key):
        #esta função é uma função intermédia para evidar multiplos hit's na base de dados, c
        def get_results():
            try:
                from my_linha_caixa import LinhaCaixa
            except:
                from linha_caixa import LinhaCaixa
            record_lines = LinhaCaixa(where = "caixa = '{caixa}'".format(caixa = key)).get()
            return record_lines
        return erp_cache.get(key = self.__model_name__ + str(key), createfunc = get_results)

    def get_total(self, key):
        value = to_decimal(0)
        record_lines = self.record_lines(key)
```

```
        if record_lines:
            for line in record_lines:
                value += to_decimal(line['entrada']) - to_decimal(line['saida'])
        return value
```

Função para um botão do workflow

```
Ex:
    (básico, limita-se a mudar o estado)
    def Cancelar(self, key, window_id):
        #Cancela a folha de caixa
        self.kargs = get_model_record(model = self, key = key)
        self.kargs['estado'] = 'Cancelada'
        self.put()
        return form_edit(window_id = window_id).show()




#   def estado_dyn_attrs(self, value):
#       # permite devolver atributos dinamicos ou seja atributos html que mudam consoante po
#       result = {}
#       #if value == 'Rascunho':
#       #   result = {'total':'hidden', 'data':'hidden'}
#       #elif value in ['Confirmado','Facturado']:
#       #   result = {'total':'disabled', 'data':'disabled'}
#       return result
#   def get_options(self, cliente=None):
#       #no get_options se eu acrescentar argumentos além do habitual "self" tenho que os te
#       options = []
#       opts = self.get()
#       for f in self.__fields__:
#           if f[0] == 'cliente':
#               field=f
#       for option in opts:
#           if cliente:
#               if str(option['cliente']) == str(cliente):
#                   nome_cliente = get_field_value(record=option, field=field, model=self)
#                   options.append((str(option['id']), '{numero}'.format(data=str(option['da
#           else:
#               nome_cliente = get_field_value(record=option, field=field, model=self)['fiel
#               options.append((str(option['id']), '{numero}'.format(data=str(option['data']
#       return options
```