Université Abou Bekr Belkaid Tlemcen

Faculté Des Sciences

Département d'Informatique

TP ERP

1ER PARTIE

Master 2 GL

2017 - 2018

MESSABIHI Mohamed

GHITRI Mustapha Salim

Structure d'un module

- **__init__.py** Déclarer tout les fichier python utiliser par le module.
- **__manifest__.py** Déclaration et description du module.
- **static** Répertoire qui contient les images, CSS, JS, logo …
- **models** Répertoire des fichiers Python (Objets).
- **views** Répertoire des fichiers XML (Interfaces).
- **security** Répertoire des fichiers XML et CSV (Droit d'accès).
- **wizard** Répertoire des fichiers PY et XML destinés au wizard
- **report** Répertoire des fichiers XML (Qweb)
- **data** Répertoire des fichiers XML (enregistrements)

__manifest__.py

Ce fichier doit contenir un dictionnaire Pyhton avec les valeurs suivantes :

name	Le nom du module.
version	La version du module, sur deux chiffres (exp. 1.1 ou 2.3).
description	La description du module y compris la documentation sur l'utilisation du module.
author	L'auteur du module.
website	Le site Web du module.
licence	La licence du module (par défaut: GPL-2).
installable	True ou False. Indique si le module est installable ou non.
category	La catégorie dans laquelle le module va être placée.
application	True ou False. Pour que le module soit identifié comme application. Seul Odoo délivre les certificats qualifiant un module d'application.
data	Liste de fichiers xml contenant les interfaces
…	

Les Objets

- Chaque objet est représenté par une classe python qui hérite de la classe ***« models.Model »***.
- Le nom de l'objet est déclaré dans l'attribut « _name ».

Champs et Attributs

- Les champs peuvent êtres de plusieurs type (*Char, Text, Integer, Float, Boolean, Date, Selection, binary …*)
- Chaque champ est personnalisé par des attributs (*string, required, readonly, default, compute, related …*).

```
11  class session(models.Model):
12      _name = 'formation.session'
13
14      def compute_duree(self):
15          date_debut = datetime.strptime(self.date_debut,'%Y-%m-%d')
16          date_fin = datetime.strptime(self.date_fin,'%Y-%m-%d')
17          self.duree = str(date_fin - date_debut)
18
19      name = fields.Char(string='Identifiant', required=True)
20      nb_participant = fields.Integer(string='Nombre de participant')
21      date_debut = fields.Date(string='Date de début', required=True)
22      date_fin = fields.Date(string='Date de fin', required=True)
23      duree = fields.Char(string="Durée", compute="compute_duree")
24      state = fields.Selection([
25              ('bientot', 'Bientôt'),
26              ('en_cours', 'En cours'),
27              ('terminer', 'Terminer'),
28              ], string='Status', default='bientot')
29
```

Les interfaces

- Ces interfaces sont composées des : *Menus, Action, Vue …*
- La Création et la configuration de ces interfaces est faite a travers des fichiers « XML », avec le modèle ci-contre.

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <odoo>
4      <data>
5
6          Déclaration des éléments de l'interface
7
8      </data>
9  </odoo>
10
```

Menus

- Les menus ont une structure hiérarchiques, le menu qui n'a pas de parent est le « *TOP MENU* » *(généralement le nom du module)*
- La création d'un menu se fait par la balise « menuitem » avec les attributs suivant : (*id, name, parent, sequence, action …*)

Actions

- Une action est un événement déclenché suite à un click.
- Les actions sont des enregistrements dans le modèle « *ir.actions.act_window* ».
- Ce modèle possède des champs obligatoire et des champs optionnels (*name, res_model, view_type, view_mode*)

Vues

- Les objets sont affichés dans le navigateur par les vues standards de Odoo.
- Un objet peut être représenté dans une ou plusieurs vues (*form, tree, kanban, search, calendar, graph, gantt*).
- Les vues sont des enregistrements dans le modèle « *ir.ui.view* ».
  - La vue formulaire.
  - La vue arbre (Tree).
  - La vue Kanban.
  - La vue recherche.

Relation entre les objets

L'ORM offert par Odoo permet de définir des champs relationnels entre les objets.

Many2one

Etablir une relation à un objet parent, en utilisant une clé étrangère.

```
formation_id = fields.Many2one('formation.formation', string='Product', ondelete='restrict')
```

One2many

- Etablir une relation virtuelle vers plusieurs enregistrements d'un autre objets.
- Chaque champs One2many est lié a un champ Many2one dans l'objet référencé

```
session_ids = fields.One2many('formation.session', 'formation_id', string='Sessions')
```

Many2many

Etablir une relation bidirectionnelle, plusieurs à plusieurs.

```
candidat_ids = fields.Many2many('formation.formateur', 'sess_form' 'session_id', 'formateur_id')
```

Fonctions & Méthode

- Le traitement des données s'effectue a travers des fonctions python.
- La déclaration des fonctions est faites avec le paramètre « self » suivie par les paramètres spécifiques.
- Le type de retour peut être une variable, une action, comme il peut être nul.

```
def function(self):
    pass
```

Champ calculable

- Est un champ calculé a partir des autres champs.
- Le champ est spécifié avec l'attribut « compute ».
- La valeur de l'attribut « compute » est le nom de la fonction python dans la quelle le champs est calculé.
- Les champs calculés, ne sont pas stockés dans la BDD, ils sont recalculés a chaque fois.

```
duree = fields.Char(string="Durée", compute="compute_duree")
```

API (décorateurs)

@api.multi

Décore les méthodes où « self » est un ou plusieurs enregistrements.

@api.one

Décore les méthodes où « self » Devrait être 1 et un seul enregistrement.

@api.onchange('field_name')

- Décore les méthodes de changement des champs « onchange ».
- Chaque argument est un nom d'un champ.
- La méthode est exécutée lors d'un changement d'un champs des arguments.

@api.depends('field_name')

Utilisé dans les méthodes « compute » pour spécifier les dépendances du champ calculé.

@api.constraints('field_name')

- Décore les méthodes de vérification des contraintes.
- Ce décorateur va assurer l'exécution de cette fonction sur les opérations de *création*, *modification*, *suppression*.

Quelque fonctions de L'ORM

self.env['object.name']

Retourne l'object *'object.name'*

self.env['object.name'].search([Domaine])

Retourne l'enregistrement (1 ou n) qui satisfait la condition du domaine.

self.env['object.name'].browse(id)

Retourne l'enregistrement de l'objet *'object.name'* avec l'identificateur *'id'*.

self.env['object.name'].create({dict})

Crée un enregistrement dans l'objet *'object.name'* avec les valeurs du dictionnaire.

self.env['object.name'].write(id,{dict})

Modifie l'enregistrement avec l'identificateur *'id'* de l'objet *'object.name'* .

self.env['object.name'].unlink(id)

Supprime l'enregistrement avec l'identificateur *'id'* de l'objet *'object.name'* .

self.env.ref('Module.External_id')

Retourne l'enregistrement qui a comme id externe *'External_id'* . (celui du XML)

Wizard

- Un Wizard est un assistant utilisé pour développer des fonctionnalités d'une manière plus pratique.
- Les Wizards utilisent des enregistrements temporels pour ne pas alourdir le système. (l'enregistrement est supprimé après un certain temps, appeler transitoire)
- Pour déclarer un Wizard, il faut créer une classe qui hérite du (*models.TransientModel*).

```
class Mon_Wizard(models.TransientModel):
    _name = 'formation.mon_wizard'
```

- Un Wizard peut avoir des champs et des fonctions comme tous les objets.
- Un Wizard peut référencer des objets normaux avec un M2O, mais la réciproque n'est pas vraie.
- L'action du Wizard a un mode de vue formulaire, et une nouvelle cible.

```
<field name="view_mode">form</field>
<field name="target">new</field>
```

- Le pied de page de la vue formulaire du Wizard contient 2 boutons, un pour confirmer et lancer le traitement, et l'autre pour annuler et fermer le Wizard.

```
<footer>
    <button name="action_done" string="Confirmer" type="object" class="btn-primary"/>
    <button string="Cancel" class="btn-default" special="cancel" />
</footer>
```

Les Rapports

- Les rapports sont écrits avec le HTML/QWeb.
- Pour créer un rapport sur un objet, il faut définir :

Report :

La balise *<report/>* assure la création d'un rapport.

```
<report id="report_attestation"
    string="Attestation"
    model="formation.attestation"
    report_type="qweb-pdf"
    report_name="Attestation"
    name="gestion_formation.report_attestation_template"
/>
```

Report template :

- La personnalisation du modèle est faite dans le **template** du rapport .
- Le champ « name » du rapport contient l'ID externe du template.

```
<template id="report_attestation_template">
    <t t-call="report.html_container">
        <t t-foreach="docs" t-as="o">
            <t t-call="report.external_layout">
                <div class="page">
                    <h2>Attestation de participation</h2>
                    <p>Formation :<span t-field="o.formation_id.name"/></p>
                </div>
            </t>
        </t>
    </t>
</template>
```

Les Rapports

Format de papier:

Odoo permet la création d'un format de papier spécifique, (A4 par défaut)

```
<record id="paperformat_specifique" model="report.paperformat">
        <field name="name">Attestation format</field>
        <field name="default" eval="True"/>
        <field name="format">custom</field>
        <field name="page_height">210</field>
        <field name="page_width">270</field>
        <field name="orientation">Portrait</field>
        <field name="margin_top">8</field>
        <field name="margin_bottom">8</field>
        <field name="margin_left">4</field>
        <field name="margin_right">4</field>
        <field name="header_line" eval="False"/>
        <field name="header_spacing">3</field>
        <field name="dpi">90</field>
</record>
```

Le champ « paper_format » du rapport, contient l'ID externe du format de papier

```
<record id="gestion_formation.report_attestation" model="ir.actions.report.xml">
        <field name="paperformat_id" ref="gestion_formation.paperformat_specifique" />
</record>
```

Droit d'accès (security)

- Odoo gère les droit d'accès à travers les profiles utilisateurs qui peuvent appartenir a des groupes, et chaque groupe a ses propres propriétés de permission.
- Les Groupes sont des enregistrements dans la table « res.groups », et ils sont créés à l'aide d'un fichier XML.

```xml
<record id="group_gerant" model="res.groups">
    <field name="name">Gérant</field>
    <field name="category_id"
     ref="gestion_formation.centre_formation_categorie"/>
</record>
```

- Pour affecter les propriétés de permission (lecture, écriture, création, suppression) à ces groupes on passe par un fichier CSV comme le fichier suivant :

```csv
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_candidat_sec,candidat.secretaire,gestion_formation.model_formation_candidat,group_secretaire,1,1,1,1
access_candidat_ger,candidat.gerant,gestion_formation.model_formation_candidat,group_gerant,1,1,1,1
```

- id : du propriété.
- name : de la propriété.
- model_id:id : l'objet de la propriété *(module.model_**object_name**)*
- group_id:id : id externe du groupe.
- perm_read, perm_write, perm_create, perm_unlink : [lire, ecrire, crée, modifier] (0 ou 1) Vrai ou Faux.