* ERP : est l'acronyme de entreprise ressource planning, solution logicielle qui regroupe en son sein les principaux composants fonctionnel de l'entreprise.

* Point fort ERP :

- un système unifié permet de faire travailler des utilisateurs de diff métiers dans un environnement applicatif identique.
- 1 seul BSD cohérente, homogénéité des data.
- globalisation de la formation
- intégrité et unicité du SI, non redondance.

* Point faible ERP :

- coût élevé
- couvre rarement tous les besoin,
- couverture fonctionelle plus large que le besoin

* Odoo : un prologiciel open source de gestion intégré comprenant de très nombreux module permettant de répondre à de nombreux besoin de gestion des entreprise.

→ architecture technique :

- Serveur de base de donnée PostgreSQL
- Serveur d'application odoo
- client web.

→ architecture logicielle :

orienté par l'architecture MVC :

- Modèle : objet déclaré dans odoo = table dans postgreSQL
- Vue : fichier XML dans odoo.
- Contrôleur : objet python controlant les requête client-serveur

→ Protocole de communication XML-RPC, NET-RPC.

- fichier conf :

sert à la configuration des repertoire des modules, liste de build et le numero de port utilisé ...

* fichier Log :

c'est un fichier .txt qui liste les evenements exécuter, utile pour trouver les erreurs en cas de bug.

* n'importe quel modification python on doit redemarrer le serveur.

* Structure d'un module odoo :

- nom du module
  - models
  - static
  - views
  - report
  - wizard
  - security
  - /*__init__.py
  - /*__manifest__.py

ERP : acronyme de : ressource planning entreprise
solution logicielle qui regroupe les principale composant fonctionnel d'entreprise
Odoo : un prologiciel open source de gestion integré

→ manifest : déclare le répertoire comme un module odoo, contient la description du module.

architecture technique : XML-RPC NET-RPC

→ init : initialise le module, en important tous les repertoire contenant .py

→ views : contient les vues du module.

→ models : contient les fichier python.

→ wizard : pour les popups

→ report : pour les rapport

→ security : pour les droit d'accés

point fort ERP :
- 1 seul BDD, cohérence des data
- integrité et unicité du SI.
- globalisation de la formation
- coût et délai de mise en route de 3 à 36 mois

point faible ERP
- coût élevé
- couvre rarement tous les besoins
- nécissité une bonne connaissance des processus d'entreprise.

* Contenu du manifest :

- name : nom du module.
- Data : liste des chemins des fichiers installé ou màj avec le module.
- Depends : contient les autres modules chargé avant notre module soit car il les modifie ou utilise leurs fct.

* objet dans odoo:

chaque objet est représenté par une classe python qui hérite de « models.Model »

exemple:

```
from odoo import models, fields

class Session(models.Model):
    _name = 'formation.session'
    nom = fields.Char(string='Identifiant', required=True)
```

_name → nom technique du model dans la bdd on trouve formation_session

nom → nom technique se trouve dans la bdd

fields.Char → type de champs

string → ce qui va être affiché pour le user

exemple de type:
Char, Date, Integer, Boolean, Text, Selection([('...','...'), ('cash','Cash')])

exemple d'attribut pour champs:
default, Help, String, required, ondelete...

* Les interfaces:

Composé de: Menu, action, vue, créer à travers des fichier XML.

1 → les vues: form view:

```
<record id="view_session_form" model="ir.ui.view">
    <field name="name">Session</field>
    <field name="model">formation.session</field>
    <field name="arch" type="xml">
        <form string="Session">
            <header>
                <field name="etat" widget="statusbar"
                statusbar_visible="brouillon,en_cours,termine" />
            </header>
            <sheet>
                <group string="Informations de la session">
                    <field name="nom" />
                    <field name="date" />
                </group>
            </sheet>
        </form>
    </field>
</record>
```

unique

name = nom du fichier pyadéquat

champs dans fichier python

optionnel

Label de form

champs dans model python fichier

* Tree views :

appelé aussi List view :

```
<record id="view_session_tree" model="ir.ui.view">
    <field name="name">Session</field>
    <field name="model">formation.session</field>
    <field name="arch" type="xml">
        <tree string="Session">
            <field name="nom" />
            <field name="date" />
        </tree>
    </field>
</record>
```

2 → l'action : événement declanché suite à un click.

```
<record id="session_action" model="ir.actions.act_window">
    <field name="name">Session</field>          ← obligatoire
    <field name="model">formation.session</field>   ← le model qui va être utilisé après l'action
    <field name="view_type">form</field>        ← optionnel.
</record>
```

⇒ La vue est structuré comme ça :

```
<?xml version="1.0" encoding="utf-8" ?>
<odoo>
    <data>
        <!-- les balise record ici -->
    </data>
</odoo>
```

3 → Menu : `<menuitem id="gestion_formation_menu" name="Gestion des formations" sequence="1" />`

← menu père déclaré une seule fois dans le dossier views dans n'importe fichier

sequence="1" → pour être affiché en 1er.

<menuitem id="session_menu"
name="Session"
parent="gestion_formation_menu" ← référence au menu père.
action="session_action" ← id de l'action à déclencher
Sequence="1" />

* Les champs relationnel.

→ Many2one.

fields.Many2one('object.name', string='Fieldname', ...)
nom de l'objet destination - classe python - ; libellé du champ

par exemple : une formation peut avoir plusieurs session donc dans la classe session on rajoute :

formation_id = fields.Many2one('formation.formation', string='Formation' ...)
dans la classe session

→ one2many : relation virtuelle vers plusieurs enregistrements.
pour chaque many2one il doit y avoir one2many dans l'autre coté.

fields.One2many('objectname', 'field_id (m2o)', string='Fields_name'...)

exemple : session_ids = fields.One2many('formation.session', 'formation_id', string='Sessions' ...).
on l'a mis dans formation

→ Many2many :

exemple : une formation a plusieurs mot clé, un mot clé peut appartenir à plusieurs formation.

mot_clé_ids = fields.Many2many('formation.mot_clé', string='Mots clés')

* Les champs fonctionnels : c'est des champs calculables à partir d'une fonction via l'attribut compute.
ces champs ne seront pas stockés dans la bdd.

* Les méthode & fct :

```
@api.one
def compute_duree(self):
    if self.date_debut and self.date_fin:
        date_debut = datetime.strptime(self.date_debut, '%Y-%m-%d')
        :
```

* API (decorateurs):

→ @api.multi : self est un ou plusieurs enregistrement.

→ @api.one : self est un et un seul enregistrement.

→ @api.onchange('field_name'): la méthode est exécutée lors du changement du field_name.

→ @api.depends('field_name'): utiliser dans les méthodes compute pour spécifier les dépendances du champs calculé.

→ @api.contrains('field_name'): declanche une exception.

* par exemple pour dire que le prix d'une formation doit être > 1000

```
@api.contrains('prix')
def _check_price(self):
    if self.prix < 1000:
        raise ValidationError("Le prix de formation doit être > 1000")
```

* pour dire qu'un champs doit être unique :

```
-sql_contrains = [
    ('contraint1', 'UNIQUE(nom)', 'Nom de formation doit être unique'),
]
```

le nom de la contrainte ↑ la specification de contrainte avec le champs spécifié ↑ description de la contrainte.

* Wizard :

utilisent des enregistrement temporels pour ne pas remplir la système

exemple :

```
class Attestation_Wizard(models.TransientModel):
    _name = 'formation.attestation_wizard'
    date_deliverance = fields.Date(string='Date de deliverance', required=True)
    ;

@api.multi                                  ← récupérer tous les inscription de cette session et créer des attestation pour eux.
def action_done(self):
    active_id = self.env.context.get('active_id')
    session = self.env['formation.session'].browse(int(active_id))
    for c in session:
        for inscription in c.inscription_ids:
            self.env['formation.attestation'].create({'date_deliv': self.date_
                'formation': c.formation_id.nom,
                'candidat': inscription.candidat_
            })
```

→ vue de wizard :

```
<?xml version="1.0" encoding="UTF-8">
<odoo>
  <data>
    <record id="attestation_wizard_form_view" model="ir_ui_view">
      <field name="name">Attestation</field>
      <field name="model">formation.attestation_wizard</field>
      <field name="type">form</field>
      <field name="arch" type="xml">
          <form string="attestation">
              <sheet>
            <group string="creation d'attestation">
              <field name="..." >
            </group></sheet>
```

```
<footer>
  <button name="action_done" string="Confirmer" type="object"
          class="oe_highlight" />
  <button string="Annuler" class="oe_highlight"
          special="cancel" />
</footer>
</form>
</field>
</record>

<!-- action -->
<record id="attestation_wizard_action" model="ir.actions.act_window">
    <field name="name">Attestation</field>
    <field name="res_model">formation.attestation_wizard</field>
    <field name="view_type">form</field>
    <field name="target">new</field>
</record>
</data>
</odoo>.
```

pour appeler la méthode action_done

pour afficher le wizard en avant et laisser le reste dispo en background.

=> et puis on rajoute un bouton pour déclencher le wizard:

```
<button name="%(attestation_wizard_action)d" type="action"
    string="Créer Attestation" class="oe_button">
```

pour appeller une action xml.

- Les fct ORM:

-> self.env['object.name']: retourne l'objet object.name.

-> self.env['object.name'].browse(id): retourne l'enregistrement de l'objet object.name avec l'identificateur id.

→ self.env['object.name'].create({dict}) : Créer un enregistrement dans l'objet object.name avec les valeurs entre { }

→ .... .write(id, {dict}) : modifie un enregistrement précis

→ .... .unlink(id) : supprime un enregistrement avec l'identificateur id.

« Les rapport :

pour créer un rapport on doit définir : format de papier, report, report template

→ format papier :

```
<record id="paperformat_frenchcheck" model="report.paperformat">
    <field name="name">Impression</field>
    <field ..  ="format">A4</field>
    <field ..  ="orientation">Portrait</field>
    <field ..  ="margin_top">15</field>
    <..  ..  ..margin_left> ...
                :
</record>
```

→ Report :

```
<report
    id="report_inscription"          ← id unique
    file="Fiche d'inscription"       ← nom du fichier significatif
    model="formation.attestation"    ← on appele le model associé.
    ~~name="gestion.formation~~
    name="gestion_formation.fiche_inscription"   ← appel au template avec son id
    report_type="qweb-pdf"
    string="Fiche d'inscription"
    paperformat="gestion_formation.paperformat_frenchcheck"   ← appel au format de papier
/>
```

- Report template :

```
<template id="fiche_inscription">
    <t t-foreach="docs" t-as="doc">
        <t t-call="web.external_layout">
            <div class="page">
                <div name="header">
                    <center><h2>Fiche d'inscription</h2></center>
                </div>
            <br/>
            <div>
              <h3>                                    récupérer le champs
                <b>Numéro d'inscription :</b>  ↙
                <span t-field="doc.numero_inscription"/>
                  ⋮
            </div>
            <div name="footer" />
        </t>
    </t>
</template>
```

- Droit d'accès :

1- Créer des groupes dans fichier xml :

```
<?xml version ... >
<odoo>
  <data>
    <record id="gestion_formation.group_admin" model="res.groups">
        <field name="name">Administrateur</field>
    </record>
      ⋮
  </data>
</odoo>
```

pour affecter les propriété de permission aux groupe on crée un fichier

Csv : security/ir.model.access.csv

c'est une propriété

id, name, model_id:id, group_id:id, perm_read, perm_write, perm_create, perm_unlink

- id : id de la propriété
- name : nom de la propriété
- model_id:id : l'objet de la propriété (module.model_object_name) exemple : gestion_formation.model_formation_candidat
- group_id:id : id external du groupe : gestion_formation.group_admin