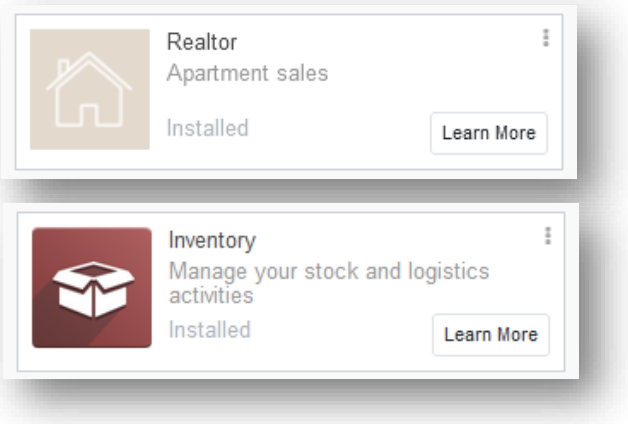
PROJET - REALTOR

ODOO - DJANGO

GESTION IMMOBILIÈRE : LE MODULE REALTOR

- Création d'un module de gestion immobilière
- Application de vente en ligne
- Gestion d'offre d'achat
- Prise en charge de la gestion des stocks



RÉPARTITION DU TRAVAIL RÉALISÉ

- Chacun réalise un point de l'énonce
- Utilisation des branches Git et issues
- Travail en binôme sur les tâches complexes
    - Pair Programming

BUSINESS OBJECT : UN APPARTEMENT

```python
class Apartment(models.Model):
    _name = 'apartment'
    _description = 'Apartment'
    _sql_constraints = [('unique_name', 'unique(name)', 'An apartment with the same name exist')]

    name = fields.Char(string="Name")
    description = fields.Text(string="Description")
    image = fields.Image(max_height=500, max_width=500, string="Picture")
    available_date = fields.Datetime(string="Available date")
    price = fields.Integer(string="Price")
    surface_apartment = fields.Integer(string="Surface of the apartment")
    surface_terrace = fields.Integer(string="Surface of the terrace")
    total_surface = fields.Integer(compute='_calculate_total_surface', string="Total surface")
    buyer = fields.Char(string="Buyer with the best offer", readonly=True, default=None, compute='_find_buyer')
    offer = fields.Integer(string="Highest offer", readonly=True, default=0)

    def _calculate_total_surface(self):
        for record in self:
            record.total_surface = record.surface_apartment + record.surface_terrace

    def _find_buyer(self):
        for record in self:
            record.buyer = None
            record.offer = 0
            min_offer = (record.price / 100) * 90
            buyers = self.env['res.partner'].search([("apartment", "in", record.name)])
            best_buyer = None
            offer = 0
            for buyer in buyers:
                if buyer.offered_price > offer and buyer.offered_price >= min_offer:
                    offer = buyer.offered_price
                    best_buyer = buyer.name
            record.buyer = best_buyer
            record.offer = offer
```

LES CONTRAINTES ...

```python
@api.constrains('price')
def _check_price(self):
    for record in self:
        if record.price <= 0:
            raise ValidationError('Price must be greater than 0')

@api.constrains('surface_apartment')
def _check_surface_apartment(self):
    for record in self:
        if record.surface_apartment <= 0:
            raise ValidationError('Surface apartment must be greater than 0')

@api.constrains('surface_terrace')
def _check_surface_terrace(self):
    for record in self:
        if record.surface_terrace <= 0:
            raise ValidationError('Surface terrace must be greater than 0')

@api.constrains('available_date')
def _check_available_date(self):
    for record in self:
        if record.create_date.year == record.available_date.year and record.create_date.month + 3 > record.available_date.month:
            raise ValidationError('Available date must be minimum 3 month after the creation of the apartment')
```

Un appartement …

Name	Ciao
Description	Apartment of Ciao !
Picture	
Available date	05/29/2023 02:00:00
Price	175,000

Surface of the apartment	88
Surface of the terrace	8
Total surface	96
Buyer with the best offer	Yacine Mamlouk
Highest offer	170,000

COMMENT L'OFFRE EST FAITE ?

EXTENSION DU RES PARTNER

```
class Offer(models.Model):
    _inherit = 'res.partner'

    apartment = fields.Many2one('apartment', string='Apartment', ondelete='cascade')
    offered_price = fields.Integer(string="Offered price", default=0)
```

- ✓ Rajout de caractéristiques supplémentaires au partner
  - ✓ Un appartement et une offre pour celui-ci

LES VUES ASSOCIÉES AUX APPARTEMENTS

Vue du formulaire

```xml
<record id="apartment_view_form" model="ir.ui.view">
    <field name="name">Apartment form</field>
    <field name="model">apartment</field>
    <field name="arch" type="xml">
        <form>
            <sheet>
                <group col="2">
                    <group col="2">
                        <field name="name">Name</field>
                        <field name="description">description</field>
                        <field name="image" widget="image"/>
                        <field name="available_date">Available date</field>
                        <field name="price">price</field>
                    </group>
                    <group col="3">
                        <field name="surface_apartment">Surface of apartment</field>
                        <field name="surface_terrace">Surface of terrace</field>
                        <field name="total_surface">Total surface</field>
                        <field name="buyer">Buyer</field>
                        <field name="offer">Offer</field>
                    </group>
                </group>
            </sheet>
        </form>
    </field>
</record>
```

LES VUES ASSOCIÉES AUX APPARTEMENTS

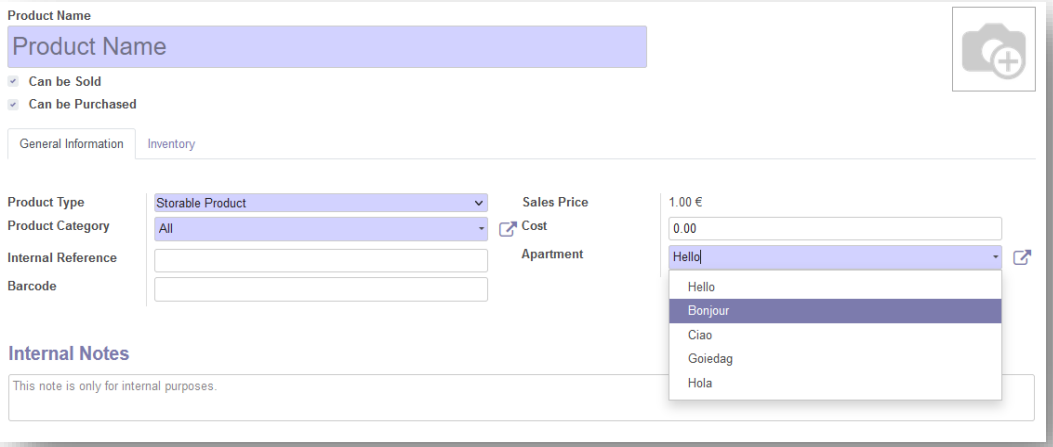
Vue en Arborescence

```xml
<record id="apartment_view_tree" model="ir.ui.view">
    <field name="name">Apartment tree</field>
    <field name="model">apartment</field>
    <field name="arch" type="xml">
        <tree decoration-muted="description,available_date,price">
            <field name="name" decoration-info="name" string="Name"/>
            <field name="description" string="description" width="3"/>
            <field name="available_date" string="available_date" width="3"/>
            <field name="price" string="price" width="3"/>
        </tree>
    </field>
</record>
```

Name	description
Hello	Apartment of Hello World !
Bonjour	Apartment of Bonjour !
Ciao	Apartment of Ciao !
Goiedag	Apartment of Goiedag !
Hola	Apartment of Hola !

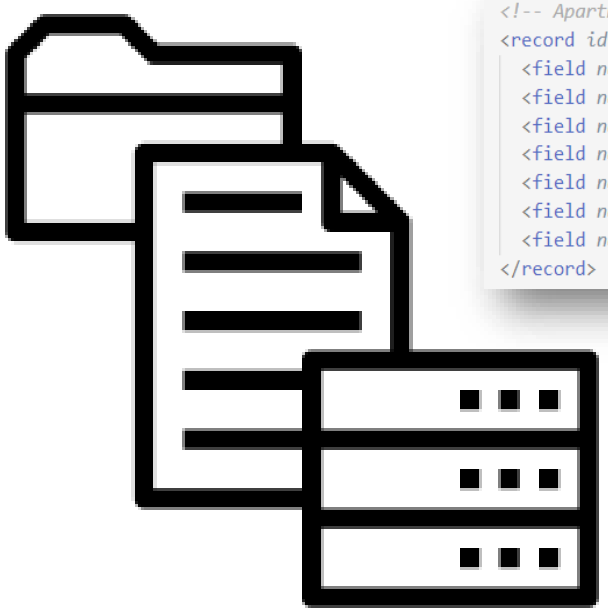
LES VUES ASSOCIÉES AUX PRODUITS

Vue formulaire - héritage




```xml
<record id="apartment_product_form" model="ir.ui.view">
    <field name="name">Apartment product form</field>
    <field name="model">product.template</field>
    <field name="inherit_id" ref="product.product_template_form_view"/>
    <field name="arch" type="xml">
        <field name="product_variant_id" position="after">
            <field name="apartment_product">Apartment</field>
        </field>
    </field>
</record>
```

Les données chargées



```xml
<!-- Apartments records -->
<record id="apartment0" model="apartment">
  <field name="name">Hello</field>
  <field name="description">Apartment of Hello World !</field>
  <field name="image" type="base64" file="realtor/static/description/img/hello_world_img.jpg"></field>
  <field name="price">125000</field>
  <field name="available_date">2023-04-12</field>
  <field name="surface_apartment">150</field>
  <field name="surface_terrace">15</field>
</record>
```

- Données chargées dès l'installation des modules

- Permet de remplir nos modules avec des données déjà existantes

LA GESTION DES STOCKS

La notion de produit

- Le module stock est automatiquement installé

```
# any module necessary for this one to work correctly
'depends': ['base', 'stock'],
```

```
class ProductApartment(models.Model):
    _inherit = 'product.template'

    apartment_product = fields.Many2one('apartment', string="Apartment", ondelete="cascade", required=True)
    list_price = fields.Float(compute='_product_price')

    def _product_price(self):
        for record in self:
            record.list_price = record.apartment_product.price
```

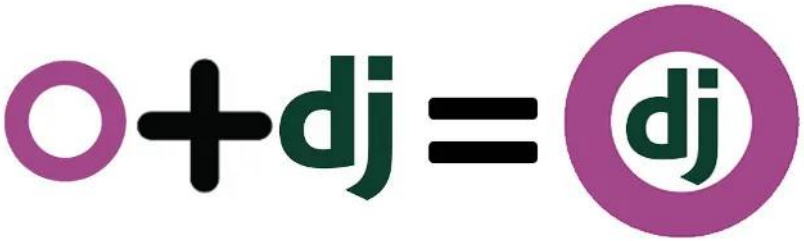
- Extension du produit de base , ajout d'un champ pour le choix de l'appartement

Alimentation initiale du stock

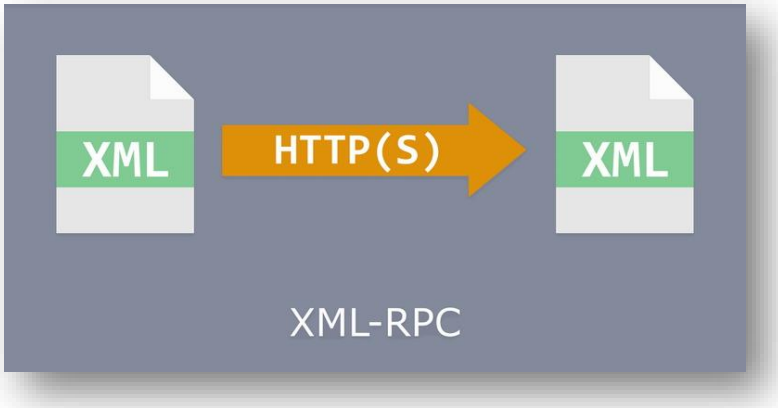
```xml
<record id="product_4" model="product.template">
  <field name="name">Apartment Hola </field>
  <field name="apartment_product" ref="apartment04" />
  <field name="type">product</field>
</record>

 <!--stock inventory-->
<record id="stock_apart_0" model="stock.inventory">
  <field name="name">Apartment Inventory</field>
  <field name="product_ids"  eval="[(6, 0,
   [ ref('product_0'),ref('product_1'),ref('product_2'),
  ref('product_3'),ref('product_4')])]"/>
</record>


<!--stock inventory line-->
<record id="stock_inventory_line_1" model="stock.inventory.line">
  <field name="product_id" ref="product_0" />
  <field name="product_uom_id" ref="uom.product_uom_unit" />
  <field name="inventory_id" ref="stock_apart_0" />
  <field name="product_qty">1</field>
  <field name="location_id" model="stock.location"
    eval="obj().env.ref('stock.warehouse0').lot_stock_id.id" />
</record>
```

- Création de produits, d'un inventaire et enregistrement dans un inventaire
- Liaison des produits créer à l'inventaire et ajout des quantités

DJANGO - REALTOR



Connexion Odoo - Django



- Utilisation de l'API ODOO
- Connexion réalisée à l'aide d'XML-RPC
- Django s'occupera de la partie cliente

Application de connexion à Odoo

- Application de configuration de la connexion Odoo
- Permet de tester le bon fonctionnement de la connexion vers Odoo

Redéfinition de l'utilisateur par défaut

```
class User(AbstractUser):
    username = None
    email = models.EmailField(unique=True)
    password_odoo = models.CharField(max_length=20)

    objects = UserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []
```

- Permet de modifier les champs de base
- L'email devient le nom d'utilisateur

Connexion Odoo et création de l'utilisateur

```python
def login(request):
    email = request.POST['email']
    password = request.POST['password']

    if connect(email, password, 'dev01'):
        if not User.objects.filter(email = email).exists():
            User.objects.create_user(email = email, password = password)
            User.objects.filter(email = email).update(password_odoo = password)

        user = authenticate(request, email=email, password=password)
        if user is not None:
            auth_login(request, user)

    return redirect('auth_odoo:index')
```

- Le client se connecte
- Création si non existant

- Connexion vers Odoo

```python
# Connects using given credential
def connect(username, password, db):
    global url
    common = xmlrpc.client.ServerProxy('{}/xmlrpc/2/common'.format(url))
    try:
        uid = common.authenticate(db, username, password, {})
        if not uid:
            return False
        else:
            return True
    except ConnectionRefusedError:
        return False
    except xmlrpc.client.Fault:
        return False
```

Récupération des produits

```python
# Fetch all products
def fetch(email, password, db):
    global url
    models = xmlrpc.client.ServerProxy('{}/xmlrpc/2/object'.format(url))
    common = xmlrpc.client.ServerProxy('{}/xmlrpc/2/common'.format(url))
    uid = common.authenticate(db, email, password, {})
    result = []
    products = models.execute_kw(db, uid, password, 'product.template', 'search_read', [[]])
    for product in products:
        apartment = models.execute_kw(db, uid, password, 'apartment', 'search_read', [[
            ['id', '=', product['apartment_product'][0] ]
            ]])
        apartment[0]['qty_available'] = product['qty_available']
        result.append(apartment[0])
    return result
```

- Le client est connecté
- Récupération des produits

Application d’achat d’appartements

Affichage des appartements grâce au script XML –RPC

Realtor Apartments

Accueil Liste des appartements Se déconnecter

Appartement	Description	Prix	Quantité disponible dans le stock	Montant maximale proposer	Meilleur acheteur
Bonjour	Apartment of Bonjour !	133000 €	1.0	134000 €	Jeff Bezos
Ciao	Apartment of Ciao !	175000 €	1.0	170000 €	Yacine Mamlouk
Goiedag	Apartment of Goiedag !	189000 €	1.0	187000 €	Mark Zuckerberg
Hello	Apartment of Hello World !	125000 €	1.0	0 €	Pas de meilleur acheteur pour le moment
Hola	Apartment of Hola !	112000 €	1.0	110000 €	Oumar Magomadov

Proposition d'une offre d’achat

```python
# Submit offer
def submit(email, password, db, name, offer, apartment):
    models = xmlrpc.client.ServerProxy('{}/xmlrpc/2/object'.format(url))
    common = xmlrpc.client.ServerProxy('{}/xmlrpc/2/common'.format(url))
    uid = common.authenticate(db, email, password, {})

    fetched_apartment = models.execute_kw(db, uid, password,'apartment', 'search_read', [[['name', '=', apartment]]])
    apartment_id = [fetched_apartment[0]['id'], fetched_apartment[0]['name']]

    if fetched_apartment[0]['offer'] < int(offer) and ((fetched_apartment[0]['offer'] / 100) * 90) < int(offer):
        search_user = models.execute_kw(db, uid, password,'res.partner', 'search_read', [[['name', '=', name]]])
        if len(search_user) == 0:
            models.execute_kw(db, uid, password, 'res.partner', 'create', [{'name': name}])
            search_user = models.execute_kw(db, uid, password,'res.partner', 'search_read', [[['name', '=', name]]])
        models.execute_kw(db, uid, password, 'res.partner', 'write', [search_user[0]['id'], {'apartment' : apartment_id[0], 'offered_price': int(offer)}])
```

- ✓ L'offre est vérifiée et ajoutée à l'utilisateur connecté

Proposition d'une offre d’achat

Formulaire de proposition d'offre sur un appartement