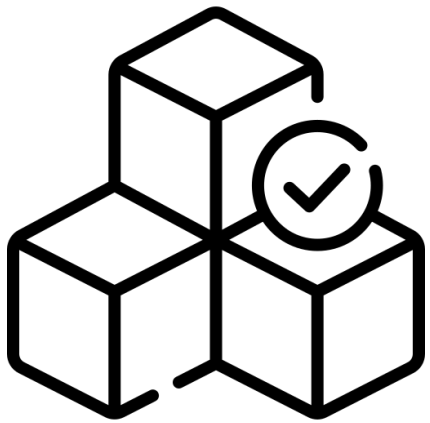
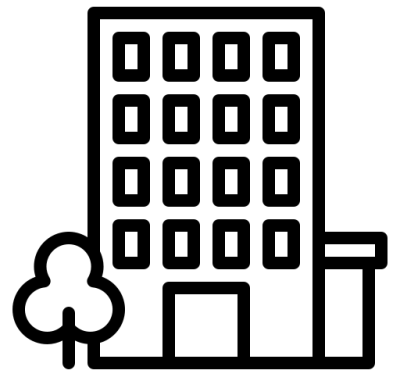
GESTION IMMOBILIÈRE

Oumar Magomadov

BRÈVE DESCRIPTION DU PROJET



Encodage des appartements dans le stock



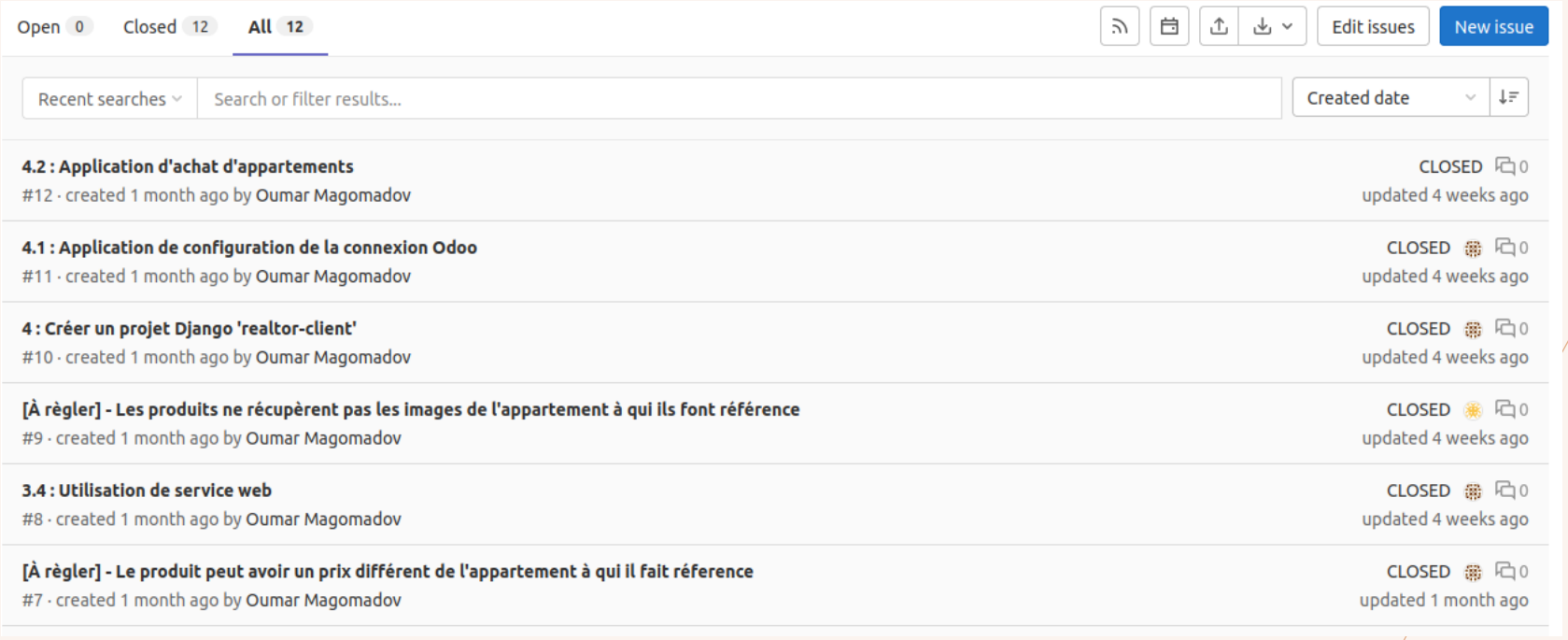
Proposition des appartements aux acheteurs

Proposition d’offres d’achats de la part des acheteurs

RÉPARTITION DES TÂCHES

Répartition de chaque point de l'énoncé entre développeur

Utilisation des issues sur GitLab

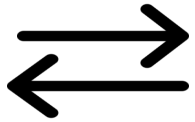


Travaille en binôme pour les tâches plus complexes ou qui sont dépendant l'un de l'autre

L’ARCHITECTURE DU PROJET

La solution logiciel est développée en utilisant un ERP (**Odoo**) et un framework web (**Django**)

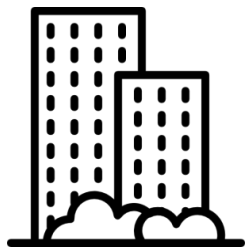
XML-RPC



La communication entre les 2 est faites à travers le **XML-RPC**

Partie Odoo : module Realtor

Apartment



business object

```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
```

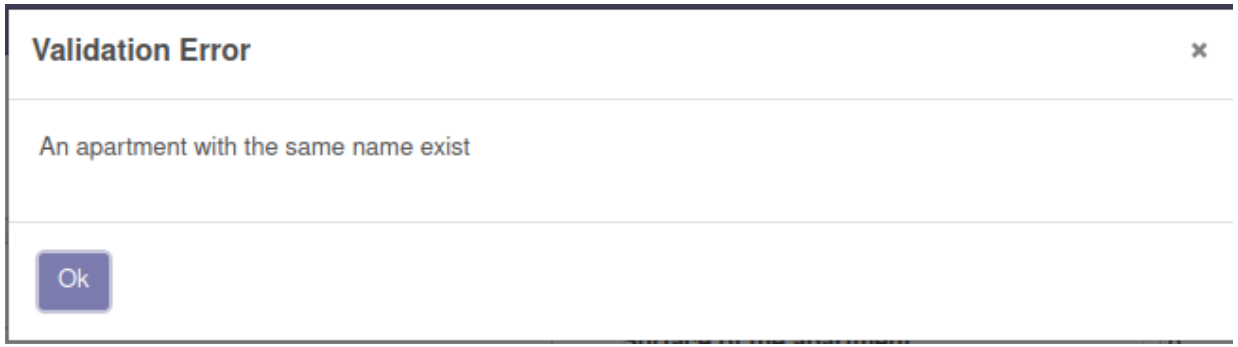
Contrainte **SQL**

+

Contrainte **Python**

L'unicité du nom d'appartement en utilisant une contrainte **SQL**

```
_sql_constraints = [('unique_name', 'unique(name)', 'An apartment with the same name exist')]
```



D'autres contraintes comme par exemple la vérification du prix (> à 0)
mais cette fois ce sont des contraintes **Python**

```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')
```

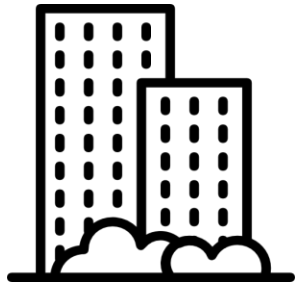
Name	Bonjour
Description	Apartment of Bonjour !
Picture	
Available date	06/01/2023 02:00:00
Price	133,000

Surface of the apartment	80
Surface of the terrace	10
Total surface	90
Buyer with the best offer	Jeff Bezos
Highest offer	134,000

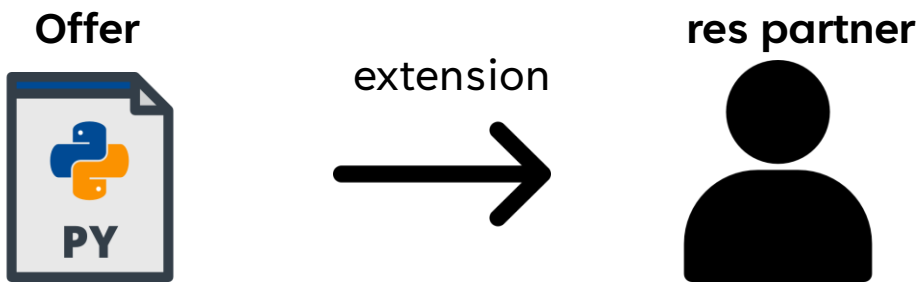
Un acheteur peut proposer une offre pour un appartement

Acheteurs

Appartement



Les acheteurs sont représentés comme des **res partner**



La classe **Offer** va faire une extension de la classe **res.partner** d'Odoo

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

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

Pas de nom technique (-> sinon héritage classique)

Le modèle **res partner** possèdera 2 nouvelles colonnes **apartment** et **offered_price**

Chaque **appartement** est un produit et il peut y'avoir plusieurs produits en stock

La dépendance au module **Stock**

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

La classe **ProductApartment** va faire une extension de la classe **product.template** de **Stock**

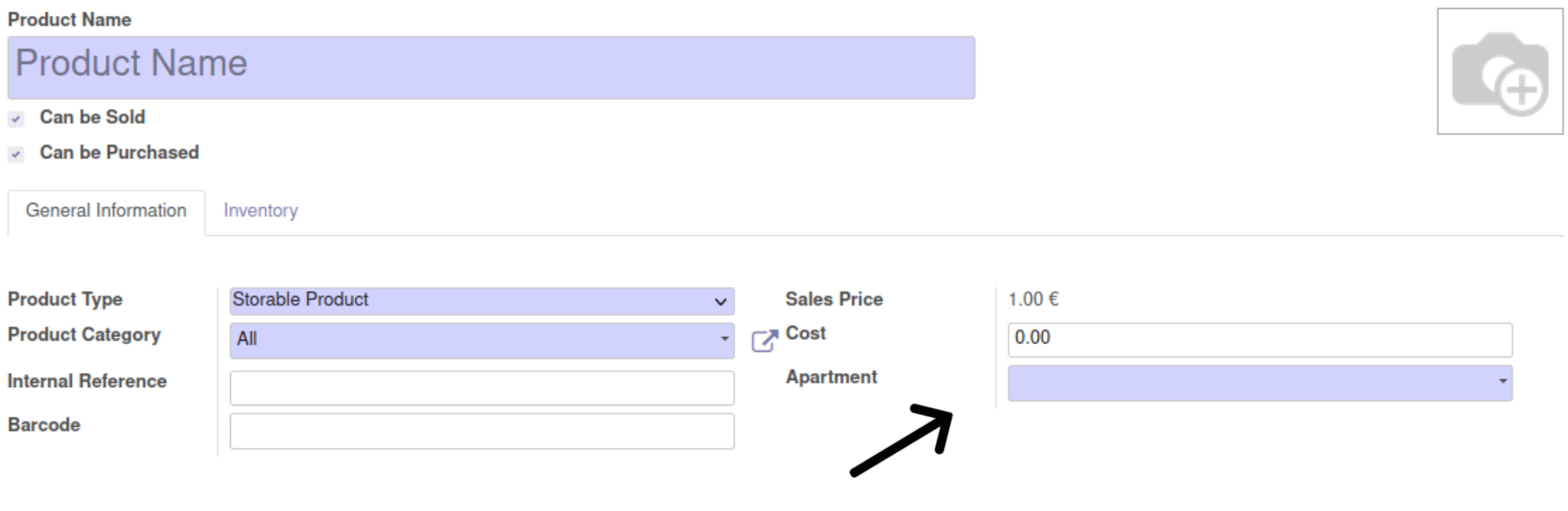
```
class ProductApartment(models.Model):
    _inherit = 'product.template'

    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
```

Un produit est lié à un appartement grâce à la relation *Many2One* vers **Apartment**

Réécriture de la colonne **list_price** de **Stock** pour que le prix de l'appartement soit associé au produit

Utilisation de *héritage de vues* pour le formulaire du **produit**

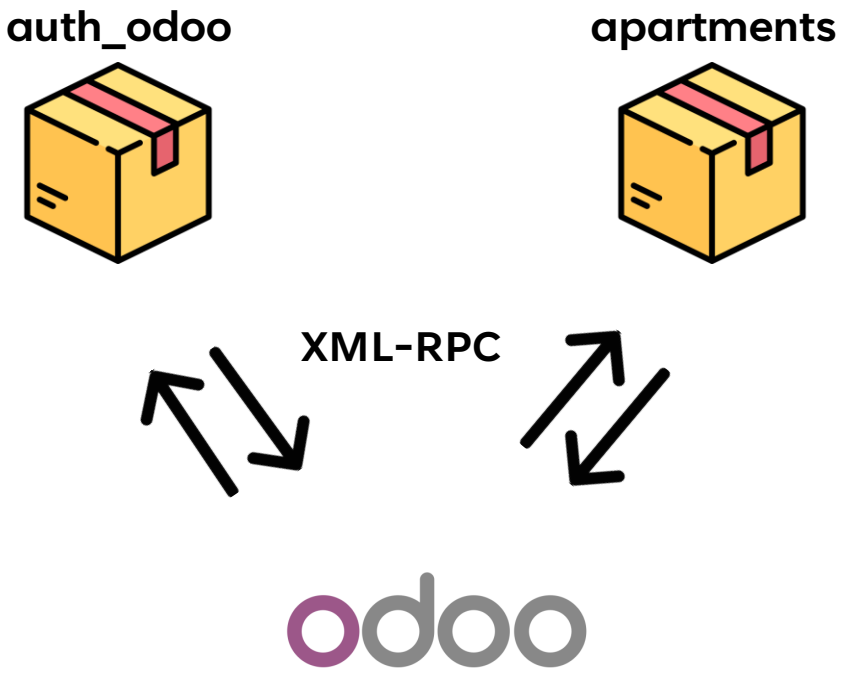
```xml
<record id="apartment_product_form" model="ir.ui.view">
    <field name="name">Apartment product form</field>
    <field name="model">product.template</field>
    <field name="inherit_id" ref="product.product_template_form_view"/>
    <field name="arch" type="xml">
        <field name="product_variant_id"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



Partie Django : realtor_client

L'application **auth_odoo** qui gère la connexion au site

L'application **apartments** qui gère l'affichage des appartements



Realtor Apartments

Accueil

Vous n'êtes pas connecter sur le site Realtor

Email*

Password*

Se connecter

Realtor Apartments | 2022

Mise en place d'un modèle **User**

Utilisation de **AbstractUser** pour pouvoir modifier le **User** de base de Django

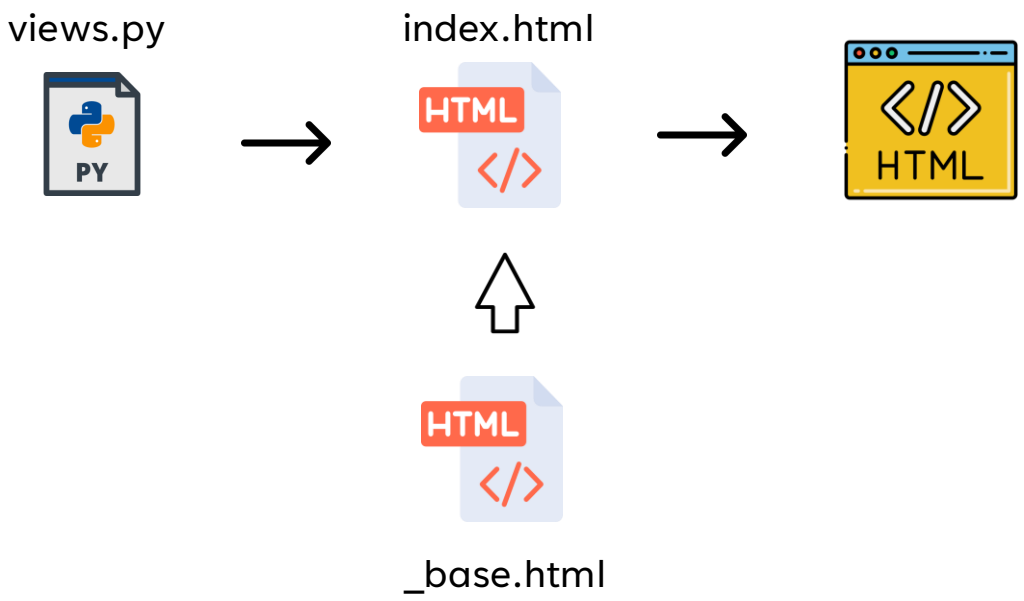
models.py

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

    objects = UserManager()

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

Remplacer le **username** par un **email**

Définition d'une **route** (/) qui fait appel à une **vue** (index)

```
app_name = 'auth_odoo'

urlpatterns = [
    path('', views.index, name='index'),
    path('login/', views.login, name="login"),
    path('logout/', views.logout, name="logout"),
]
```



Utilisation d'un **gabarit générale** (_base.html) pour les pages HTML

```
def index(request):
    return render(request, 'auth_odoo/index.html', {'form' : UserForm})
```

_base.html sera utiliser par **auth_odoo** & **apartments** (DRY)

La fonction **login** :

- Envoie de données au script **XML-RPC**
- Création de l'utilisateur dans la base de données

views.py

```python
from odoo.auth.xml_rpc import connect

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)
```

xml-rpc.py

```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
```

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

Une page qui affiche la liste des appartements

views.py

```python
from odoo.auth.xml_rpc import fetch as retrieve_data, submit as submit_offer

def index(request):
    if not request.user.is_authenticated:
        return redirect('auth_odoo:index')
    else:
        products = retrieve_data(request.user.email, request.user.password_odoo, 'dev01')
        return render(request, 'apartments/apartments.html', {'products' : products})
```

Vérifier si l'utilisateur est connecté

xml-rpc.py

```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
```

Vous pouvez proposer une offre pour un appartement

Le nom de l'acheteur

Montant

L'appartement choisi

Bonjour

Soumettre offre

Un acheteur peut proposer une offre pour un appartement

views.py

```python
def offer(request):
    submit_offer(request.user.email, request.user.password_odoo, 'dev01', request.GET['buyer'], request.GET['amount'], request.GET['apartment_name'])
    return redirect('apartments:index')
```

Si l'acheteur n'existe pas, il sera créé sur Odoo

xml-rpc.py

```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(offe
```