**PROJECT DOCUMENTATION**

Real Estate Management System Module for Odoo

Software Developer Intern Project

based on Odoo Developer Documentation

Culambot, Mark Renzkie C.

November 2023

Contents

[**A. MODULE OVERVIEW:** 3](#_Toc150991590)

[a.1. FEATURES AND FUNCTIONALITIES 3](#_Toc150991591)

[a.2. SPECIFICATIONS: 5](#_Toc150991592)

[**I. MODELS** 6](#_Toc150991593)

[1.1. estate\_property.py 6](#_Toc150991594)

[1.2. estate\_property\_offer.py 8](#_Toc150991595)

[1.3. estate\_property\_tag.py 10](#_Toc150991596)

[1.4. estate\_property\_type.py 11](#_Toc150991597)

[1.5. res\_users.py 12](#_Toc150991598)

[**II. SECURITY** 13](#_Toc150991599)

[2.1. security.xml 13](#_Toc150991600)

[2.2. security\_rules.xml 13](#_Toc150991601)

[**III. VIEWS** 14](#_Toc150991602)

[3.1. estate\_property\_views.xml 14](#_Toc150991603)

[3.2. estate\_property\_offer\_views.xml 15](#_Toc150991604)

[3.3. estate\_property\_tag\_views.xml 16](#_Toc150991605)

[3.4. estate\_property\_type\_views.xml 16](#_Toc150991606)

[3.5. estate\_menus.xml 17](#_Toc150991607)

[3.6. res\_users\_views.xml 18](#_Toc150991608)

[**IV. REPORTS** 19](#_Toc150991609)

[4.1. estate\_property\_reports.xml 19](#_Toc150991610)

[4.2. estate\_property\_template.xml 20](#_Toc150991611)

[4.3. estate\_property\_by\_salesperson\_template.xml 21](#_Toc150991612)

[**V. DEMO** 21](#_Toc150991613)

[5.1. estate\_property\_demo\_data.xml 21](#_Toc150991614)

[5.2. estate\_property\_demo\_data\_base\_on\_partner.xml 22](#_Toc150991615)

[5.3. estate\_property\_demo\_data\_x2many\_field.xml 23](#_Toc150991616)

[**VI. MANIFEST** 23](#_Toc150991617)

[6.1. manifest.py 23](#_Toc150991618)

[**B. SCREENSHOTS** 24](#_Toc150991619)

# **A. MODULE OVERVIEW:**

The Real Estate Management System Module is a comprehensive solution designed for managing real estate brokerage operations within the Odoo framework. Based on [Odoo Developer documentation](https://www.odoo.com/documentation/16.0/developer.html). This module enhances the capabilities of Odoo to effectively handle various aspects of real estate transactions, from property listings to offer management.

## a.1. FEATURES AND FUNCTIONALITIES

1. **Property Management:**
   * **Feature:** Manage various types of properties such as villas and trailer homes.
   * **Functionality:** Track details like name, description, location, and key property features.
   * **Benefit:** Easily organize and categorize different properties.
2. **Offer Management:**
   * **Feature:** Receive and manage offers from potential buyers.
   * **Functionality:** View and respond to offers, including accepting or refusing them.
   * **Benefit:** Streamlines the negotiation process with interested buyers.
3. **User Roles and Permissions:**
   * **Feature:** Assign different roles to users, such as Agents and Managers.
   * **Functionality:** Control access to specific features based on user roles.
   * **Benefit:** Enhances security and ensures that users have appropriate access.
4. **Property Types and Tags:**
   * **Feature:** Categorize properties into types and add tags for easy classification.
   * **Functionality:** Group properties based on predefined types and add custom tags.
   * **Benefit:** Simplifies property organization and search.
5. **Reports and Analytics:**
   * **Feature:** Generate reports for property details, offers, and sales.
   * **Functionality:** Access visual representations of data, such as graphs and pie charts.
   * **Benefit:** Gain insights into property performance and sales trends.
6. **Customizable Views:**
   * **Feature:** View property information in different formats, including lists and kanban.
   * **Functionality:** Customize the display of property details based on preferences.
   * **Benefit:** Provides a flexible and user-friendly interface.
7. **Menu Navigation:**
   * **Feature:** Intuitive menu structure for easy navigation.
   * **Functionality:** Access different sections like property listings and settings.
   * **Benefit:** Quick and easy access to relevant features.
8. **Demo Data for Testing:**
   * **Feature:** Pre-loaded demo data for showcasing module functionality.
   * **Functionality:** Use demo data to simulate real-world scenarios.
   * **Benefit:** Helps users understand and test the system before actual use.

## a.2. SPECIFICATIONS:

* **Compatibility:** Compatible with the base module of Odoo.
* **Documentation:** Detailed documentation for developers and users.
* **Version:** Currently at version 1.2.
* **Installability:** Easy installation and configuration.
* **Application Type:** Designed for Real Estate/Brokerage purposes.

# **I. MODELS**

## 1.1. estate\_property.py

1. **Class Definition:**
   * EstateProperty is a subclass of models.Model in the Odoo framework, representing a model for real estate properties.
   * \_name: Specifies the technical name of the model in the Odoo system ("estate.property").
   * \_description: Provides a human-readable description for the model ("Estate Property").
   * \_order: Specifies the default sorting order for records when queried from the database ("id desc").
2. **Fields:**
   * Various fields are defined using the fields module from Odoo, such as name, description, postcode, etc. These fields represent different attributes of a real estate property (e.g., title, description, bedrooms, etc.).
   * Some fields have specific properties, like required, readonly, default, and compute, which define their behavior and appearance in the Odoo user interface.
3. **Constraints:**
   * Two SQL constraints are defined to ensure that the expected price is positive and the selling price is non-negative.
   * A method \_check\_selling\_price is defined to check a condition regarding the relationship between the selling price and the expected price.
4. **Computed Fields:**
   * The \_total\_area method computes the total area of a property based on its living area and garden area.
   * The \_compute\_best\_price method computes the best offer price among the property's associated offers.
5. **Onchange Method:**
   * The \_onchange\_garden method is called when the "garden" field changes. It automatically sets the garden area and orientation based on whether the property has a garden.
6. **Business Logic Methods:**
   * The unlink method overrides the default behavior of deleting records and allows deletion only if the property is in the "new" or "canceled" state.
   * The make\_property\_sold and make\_property\_cancel methods update the state of a property to "sold" or "canceled," respectively, based on certain conditions.
7. **Exception Handling:**
   * The code uses exceptions (UserError, ValidationError) to handle errors and communicate issues with the user.

## 1.2. estate\_property\_offer.py

1. **Class Definition:**
   * **EstatePropertyOffer** is a subclass of **models.Model** in the Odoo framework, representing a model for offers made on real estate properties.
   * **\_name**: Specifies the technical name of the model in the Odoo system ("estate.property.offer").
   * **\_description**: Provides a human-readable description for the model ("Estate Property Offer").
   * **\_order**: Specifies the default sorting order for records when queried from the database ("price desc").
2. **Fields:**
   * Various fields are defined using the **fields** module from Odoo, such as **price**, **partner\_id**, **status**, etc. These fields represent different attributes of an offer (e.g., price, partner making the offer, status, etc.).
3. **Constraints:**
   * One SQL constraint is defined to ensure that the offer price is positive.
4. **Computed Field and Inverse Method:**
   * The **date\_deadline** field is computed based on the offer's creation date and validity period.
   * The **\_compute\_date\_deadline** method calculates the deadline based on the creation date and validity period.
   * The **\_inverse\_date\_deadline** method calculates the validity period based on the deadline.
5. **Create Method:**
   * The **create** method is overridden to perform additional checks when creating an offer.
   * It checks if the offer price is higher than the best offer price for the associated property. If not, it raises a **UserError**.
   * It updates the state of the associated property to 'received' and then calls the superclass method to create the offer.
6. **Business Logic Methods:**
   * The **make\_accept** method is used to mark an offer as accepted. It refuses other offers on the same property and updates the property's state, selling price, and buyer information.
   * The **make\_refuse** method is used to mark an offer as refused. If no other offer is accepted, it resets the property's selling price, buyer, and state.

## 1.3. estate\_property\_tag.py

1. **Class Definition:**
   * **EstatePropertyTag** is a subclass of **models.Model** in the Odoo framework, representing a model for tags associated with real estate properties.
   * **\_name**: Specifies the technical name of the model in the Odoo system ("estate.property.tag").
   * **\_description**: Provides a human-readable description for the model ("Estate Property Tag").
   * **\_order**: Specifies the default sorting order for records when queried from the database ("name").
2. **Fields:**
   * Various fields are defined using the **fields** module from Odoo, such as **name** and **color**. These fields represent different attributes of a property tag (e.g., name and color).
   * The **name** field is a required character field representing the name of the property tag.
   * The **color** field is an integer field representing the color associated with the tag.
3. **Constraints:**
   * One SQL constraint is defined to ensure that the name of a property tag is unique.
   * The constraint is named 'unique\_tag' and checks for the uniqueness of the 'name' field.

## 1.4. estate\_property\_type.py

1. **Class Definition:**
   * **EstatePropertyType** is a subclass of **models.Model** in the Odoo framework, representing a model for types of real estate properties.
   * **\_name**: Specifies the technical name of the model in the Odoo system ("estate.property.type").
   * **\_description**: Provides a human-readable description for the model ("Estate Property Type").
   * **\_order**: Specifies the default sorting order for records when queried from the database ("sequence, name").
2. **Fields:**
   * Various fields are defined for the model, such as **name**, **property\_ids**, **offer\_ids**, **offer\_count**, and **sequence**.
   * The **name** field is a required character field representing the name of the property type.
   * The **property\_ids** field establishes a one-to-many relationship with the **estate.property** model, linking property types to properties.
   * The **offer\_ids** field establishes a one-to-many relationship with the **estate.property.offer** model, linking property types to offers.
   * The **offer\_count** field is an integer field that computes the count of associated offers for each property type.
   * The **sequence** field is an integer field representing the sequence/order of the property types.
3. **Constraints:**
   * One SQL constraint is defined to ensure that the name of a property type is unique.
   * The constraint is named 'unique\_type' and checks for the uniqueness of the 'name' field.
4. **Computed Field and Method:**
   * The **offer\_count** field is computed based on the count of associated offers for each property type.
   * The **\_compute\_offer\_count** method calculates the offer count for each property type and updates the **offer\_count** field.

## 1.5. res\_users.py

1. **Class Definition:**
   * The class **ResUsers** is defined as a subclass of **models.Model** in the Odoo framework.
   * **\_inherit**: Specifies that this class is extending (or inheriting from) the existing model named "res.users."
2. **New Field:**
   * The **property\_ids** field is added to the **ResUsers** class.
   * This field establishes a one-to-many relationship with the model **estate.property**.
   * It links user records to property records where the user is designated as a salesman (**salesman\_id**).
   * The **domain** parameter restricts the linked property records to those with a state of either 'new' or 'received.'

# **II. SECURITY**

## 2.1. security.xml

1. **First Record - Agent Group (estate\_group\_user):**
   * A security group named "Agent" is defined using the **res.groups** model.
   * The group is assigned an ID (**estate\_group\_user**).
   * The group is categorized under the module category named "Real Estate Brokerage."
   * The group is implied to have the permissions of the 'account.group\_account\_invoice' group. Implied groups inherit the access rights of the specified groups.
2. **Second Record - Manager Group (estate\_group\_manager):**
   * Another security group named "Manager" is defined.
   * This group is assigned an ID (**estate\_group\_manager**).
   * It is also categorized under the "Real Estate Brokerage" module category.
   * This group is implied to have the permissions of the 'estate\_group\_user' group. This means that members of the "Manager" group will have the access rights of the "Agent" group as well.
   * The **users** field specifies that the administrator user (**base.user\_admin**) is a member of the "Manager" group.

## 2.2. security\_rules.xml

1. **First Record - User Access Rule (rule\_user\_see\_property\_base\_on\_saleperson):**
   * An access rule named "Estate: see property base on saleperson" is defined using the **ir.rule** model.
   * The rule is assigned an ID (**rule\_user\_see\_property\_base\_on\_saleperson**).
   * It applies to the model identified by the reference 'model\_estate\_property' (assumed to be the "Estate Property" model).
   * The rule is associated with the "Agent" group (**estate.estate\_group\_user**).
   * The **domain\_force** field specifies a domain filter for record visibility: properties where either the salesman is not set (**salesman\_id','=',False**) or the salesman is the current user (**salesman\_id','=',user.id**).
   * Permissions are set to read and write (**perm\_read=True** and **perm\_write=True**), while creation and deletion permissions are denied (**perm\_create=False** and **perm\_unlink=False**).
2. **Second Record - Manager Access Rule (rule\_manager\_see\_property\_base\_on\_saleperson):**
   * Another access rule with a similar name is defined, but this one applies to the "Manager" group (**estate.estate\_group\_manager**).
   * It has the same model association and a different domain force, which is always true (**[(1,'=',1)]**), meaning it allows managers to see all properties.
   * Managers have broader visibility with no restrictions on the salesperson field.

# **III. VIEWS**

## 3.1. estate\_property\_views.xml

1. **Tree View (estate\_property\_view\_tree):**
   * This view presents a list of properties in a tree structure.
   * The **decoration-success** attribute highlights properties with states 'received' or 'accepted' in green.
   * The **decoration-bf** attribute decorates properties with the 'accepted' state specifically.
   * The **decoration-muted** attribute decorates properties with the 'sold' state in a muted style.
   * It displays several fields like name, state (invisible), postcode, bedrooms, living area, expected price, selling price, and date availability.
2. **Form View (estate\_property\_view\_form):**
   * This view represents the detailed form for each property.
   * It includes buttons for marking a property as sold or canceled and printing sale information.
   * The **header** section contains buttons and a status bar for the property state.
   * The **sheet** section contains grouped information about the property, including tags, property type, postcode, date availability, expected price, best price, and selling price.
   * The **notebook** includes pages for description, offers, and other information, each containing relevant fields.
3. **Kanban View (estate\_property\_view\_kanban):**
   * This view represents properties in a kanban board format.
   * It defaults to grouping properties by their state.
   * Each card in the kanban displays the property name, expected price, best price, selling price, and property type.
4. **Search View (estate\_property\_view\_search):**
   * This view provides a search interface for filtering properties based on various criteria.
   * It includes fields for name, postcode, expected price, bedrooms, living area, facades, and a filter for available properties.
   * It also includes a group-by filter for postcodes.
5. **Graph View (estate\_property\_graph):**
   * This view presents a graph visualization for properties.
   * It displays the property name in a graph format.
6. **Action (estate\_property\_action):**
   * This action defines how users can interact with the "Estate Property" model.
   * It specifies the name of the action, the target model ("estate.property"), and the view modes (kanban, tree, form, graph).
   * The context is set to have the search default to properties with the 'new' state.

## 3.2. estate\_property\_offer\_views.xml

1. **Tree View (estate\_property\_offer\_view\_tree):**
   * This view presents a list of property offers in a tree structure.
   * The **editable="bottom"** attribute allows users to add new offers directly from the tree view.
   * The **decoration-success** attribute highlights offers with the 'accepted' status.
   * The **decoration-danger** attribute highlights offers with the 'refused' status.
   * It displays fields like price, partner (buyer), validity, date deadline, property type, and status.
   * Two buttons are provided for making accept and refuse actions on the offers.
   * The status field is made invisible in the tree view.
2. **Form View (estate\_property\_offer\_view\_form):**
   * This view represents the detailed form for each property offer.
   * It includes fields for price, partner (buyer), validity, date deadline, and status.
   * The fields are grouped in a sheet.
3. **Action (estate\_property\_offer\_action):**
   * This action defines how users can interact with the "Estate Property Offer" model.
   * It specifies the name of the action, the target model ("estate.property.offer"), and the view modes (tree and form).
   * The domain is set to filter offers based on the active property type.

## 3.3. estate\_property\_tag\_views.xml

1. **Tree View (estate\_property\_tag\_view\_tree):**
   * This view presents a list of property tags in a tree structure.
   * The **editable="bottom"** attribute allows users to add new tags directly from the tree view.
   * It displays the name field for each tag.
2. **Form View (estate\_property\_tag\_view\_form):**
   * This view represents the detailed form for each property tag.
   * It includes a field for the tag name.
   * The name field is presented within an **<h1>** header.
3. **Action (estate\_property\_tag\_action):**
   * This action defines how users can interact with the "Estate Property Tag" model.
   * It specifies the name of the action, the target model ("estate.property.tag"), and the view modes (tree and form).

## 3.4. estate\_property\_type\_views.xml

1. **Tree View (estate\_property\_type\_view\_tree):**
   * This view presents a list of property types in a tree structure.
   * It displays the sequence, which is a field used for ordering, and the name of each property type.
2. **Form View (estate\_property\_type\_view\_form):**
   * This view represents the detailed form for each property type.
   * It includes a button for accessing property offers related to this property type.
   * The form includes a sheet with a button box, a title with the property type name, and a notebook with a page for displaying associated properties.
   * The associated properties are displayed in a tree structure, showing their name, expected price, and state.
3. **Graph View (estate\_property\_type\_graph\_pie):**
   * This view presents a pie chart graph for property types.
   * It visualizes the distribution of properties across different property types.
4. **Action (estate\_property\_type\_action):**
   * This action defines how users can interact with the "Estate Property Type" model.
   * It specifies the name of the action, the target model ("estate.property.type"), and the view modes (tree, form, and graph).

## 3.5. estate\_menus.xml

1. **Root Menu (real\_estate\_root\_menu):**
   * This menu item represents the main entry point for the Real Estate module.
   * It has the name "Real Estate" and serves as the top-level menu.
2. **First Level Menu (real\_estate\_first\_level\_menu):**
   * This menu item represents a submenu under "Real Estate" called "Advertisements."
   * It is nested under the root menu (**real\_estate\_root\_menu**).
3. **Second Level Menu (real\_estate\_second\_level\_menu):**
   * This menu item represents a submenu under "Advertisements" called "Properties."
   * It is nested under the first-level menu (**real\_estate\_first\_level\_menu**).
   * It is associated with the action **estate\_property\_action**, which specifies how to display the properties.
4. **First Level Menu for Settings (estate\_type\_first\_level\_menu):**
   * This menu item represents a submenu under "Real Estate" called "Settings."
   * It is associated with the group **estate.estate\_group\_manager**, meaning that only users belonging to this group can see and access this menu.
5. **Second Level Menu for Property Types (estate\_type\_second\_level\_menu):**
   * This menu item represents a submenu under "Settings" called "Property Types."
   * It is associated with the action **estate\_property\_type\_action**, which specifies how to display the property types.
   * It is also associated with the group **estate.estate\_group\_manager**.
6. **Menu for Property Tags (estate\_tag\_menu):**
   * This menu item represents a submenu under "Settings" called "Property Tags."
   * It is associated with the action **estate\_property\_tag\_action**, which specifies how to display the property tags.
   * It is also associated with the group **estate.estate\_group\_manager**.

## 3.6. res\_users\_views.xml

1. **Root Menu (real\_estate\_root\_menu):**
   * Represents the main entry point for the Real Estate module.
   * Named "Real Estate."
2. **First Level Menu (real\_estate\_first\_level\_menu):**
   * Represents a submenu under "Real Estate" called "Advertisements."
   * Nested under the root menu (**real\_estate\_root\_menu**).
3. **Second Level Menu (real\_estate\_second\_level\_menu):**
   * Represents a submenu under "Advertisements" called "Properties."
   * Nested under the first-level menu (**real\_estate\_first\_level\_menu**).
   * Associated with the action **estate\_property\_action**, specifying how to display properties.
4. **First Level Menu for Settings (estate\_type\_first\_level\_menu):**
   * Represents a submenu under "Real Estate" called "Settings."
   * Associated with the group **estate.estate\_group\_manager**, allowing access only to users in this group.
5. **Second Level Menu for Property Types (estate\_type\_second\_level\_menu):**
   * Represents a submenu under "Settings" called "Property Types."
   * Associated with the action **estate\_property\_type\_action**, specifying how to display property types.
   * Also associated with the group **estate.estate\_group\_manager**.
6. **Menu for Property Tags (estate\_tag\_menu):**
   * Represents a submenu under "Settings" called "Property Tags."
   * Associated with the action **estate\_property\_tag\_action**, specifying how to display property tags.
   * Also associated with the group **estate.estate\_group\_manager**.

# **IV. REPORTS**

## 4.1. estate\_property\_reports.xml

1. **Estate Property Report (report\_estate\_property):**
   * **Name:** "Estate Property Report"
   * **Model:** **estate.property** - Associated with the Estate Property model.
   * **Report Type:** qweb-html - Specifies the type of report.
   * **Report Name:** **estate.estate\_property\_template\_report** - Refers to the QWeb template used for rendering the report.
   * **Report File:** **estate.estate\_property\_template\_report** - Refers to the QWeb template file.
   * **Print Report Name:** 'Property Offers' - Name displayed when the report is printed.
   * **Binding Model ID:** Reference to the model ID for **estate.property**.
   * **Binding Type:** report - Indicates that this action is a report.
2. **Final Sale Report (report\_final\_sale):**
   * **Name:** "Final Sale"
   * **Model:** **estate.property** - Associated with the Estate Property model.
   * **Report Type:** qweb-html - Specifies the type of report.
   * **Report Name:** **estate.estate\_property\_template\_report** - Refers to the QWeb template used for rendering the report.
   * **Report File:** **estate.estate\_property\_template\_report** - Refers to the QWeb template file.
   * **Print Report Name:** 'Final Sale' - Name displayed when the report is printed.
   * **Binding Type:** report - Indicates that this action is a report.
3. **Estate Property by Salesman Report (report\_estate\_property\_by\_salesman):**
   * **Name:** "Estate Property by Salesman"
   * **Model:** **res.users** - Associated with the User model.
   * **Report Type:** qweb-html - Specifies the type of report.
   * **Report Name:** **estate.estate\_property\_by\_saleperson\_template** - Refers to the QWeb template used for rendering the report.
   * **Report File:** **estate.estate\_property\_by\_saleperson\_template** - Refers to the QWeb template file.
   * **Print Report Name:** 'Estate Property by Salesman' - Name displayed when the report is printed.
   * **Binding Model ID:** Reference to the model ID for **res.users**.
   * **Binding Type:** report - Indicates that this action is a report.

## 4.2. estate\_property\_template.xml

1. **Estate Property Offer Template (estate\_property\_offer\_template\_report):**
   * This template is designed to display information about offers related to a property.
   * The template first displays the expected price and status of the property.
   * It then checks if the property has any associated offers (**property.offer\_ids**).
   * If offers exist, it creates a table displaying details of each offer, such as price, partner, validity, deadline, and status.
   * If no offers exist, it displays a message stating that the property has no offers yet.
2. **Estate Property Template (estate\_property\_template\_report):**
   * This template is a container that calls other templates and is meant to be used as part of a larger report.
   * It iterates over a list of documents (**docs**), treating each document as a property.
   * For each property, it displays the property's name and the name of the salesman assigned to it.
   * It then calls the **estate\_property\_offer\_template\_report** template to display information about offers related to the property.

## 4.3. estate\_property\_by\_salesperson\_template.xml

**Template (estate\_property\_by\_saleperson\_template):**

* This template is designed to be used as part of a larger report.
* It iterates over a list of documents (**docs**), treating each document as a user (salesperson).
* For each user, it displays the salesperson's name in a heading.
* It then sets a variable **properties** to the properties associated with the current user (**user.mapped('property\_ids')**).
* For each property in the list of properties, it creates a new page in the report and displays the property's name in a heading.
* It then calls the **estate.estate\_property\_offer\_template\_report** template to display information about offers related to the property.

# **V. DEMO**

## 5.1. estate\_property\_demo\_data.xml

* **Record (estate\_property\_1):**
  + It creates an estate property with the following attributes:
    - **name**: "Big Villa"
    - **state**: "new"
    - **description**: "A nice and big villa"
    - **postcode**: 12345
    - **date\_availability**: February 2, 2020
    - **expected\_price**: 1600000
    - **bedrooms**: 6
    - **living\_area**: 100
    - **facades**: 4
    - **garage**: True
    - **garden**: True
    - **garden\_area**: 100000
    - **garden\_orientation**: "south"
* **Record (estate\_property\_2):**
  + It creates another estate property with the following attributes:
    - **name**: "Trailer home"
    - **state**: "canceled"
    - **description**: "Home in a trailer park"
    - **postcode**: 54321
    - **date\_availability**: January 1, 1970
    - **expected\_price**: 100000
    - **selling\_price**: 120000
    - **bedrooms**: 1
    - **living\_area**: 10
    - **facades**: 4
    - **garage**: False

## 5.2. estate\_property\_demo\_data\_base\_on\_partner.xml

* **Records (res.partner):**
  + Two partner records are created:
    1. Partner with **name** "Azure Interior" (record ID: **partner\_1**).
    2. Partner with **name** "Deco Addict" (record ID: **partner\_2**).
* **Records (estate.property.offer):**
  + Three offers are created, associating partners and properties:
    1. Offer from "Azure Interior" for "Big Villa" (**estate\_property\_1**) with **price** 10,000 and **validity** of 14 days (record ID: **estate\_property\_offer\_1**).
    2. Another offer from "Azure Interior" for the same property with a higher **price** of 1,500,000 and **validity** of 14 days (record ID: **estate\_property\_offer\_2**).
    3. Offer from "Deco Addict" for the same property with a **price** higher than the previous offers (1,500,001) and **validity** of 14 days (record ID: **estate\_property\_offer\_3**).
* **Function (make\_refuse):**
  + A function is called to simulate refusing multiple offers. The **make\_refuse** function of the **estate.property.offer** model is invoked with a list of offer references (**[ref('estate\_property\_offer\_1'), ref('estate\_property\_offer\_2')]**). This function likely updates the state of the specified offers to "refused."

## 5.3. estate\_property\_demo\_data\_x2many\_field.xml

* **Record (estate.property):**
  + A new estate property record is created with the following details:
    - **name**: "Custom estate demo"
    - **state**: "new"
    - **description**: "Import from demo data"
    - **expected\_price**: 100,000
    - **bedrooms**: 1
    - **living\_area**: 10
    - **facades**: 4
    - **garage**: False
* **Field (offer\_ids):**
  + The **offer\_ids** field, which is a One2many field in the **estate.property** model, is populated using the **eval** attribute.
  + The value assigned to **offer\_ids** is a domain expression (**[(4, ref('estate\_property\_offer\_2'))]**).
  + The **[(4, ...)]** notation is used to add a new record to the One2many field. In this case, it indicates adding the offer with the reference **'estate\_property\_offer\_2'** to the **offer\_ids** field.

# **VI. MANIFEST**

## 6.1. manifest.py

* **Module Information:**
  + **name**: The name of the Odoo module, in this case, "Real Estate."
  + **description**: A brief description of the module's purpose, which includes a reference to Odoo Developer Technical Exercises.
  + **author**: The author or developer of the module (Mark Renzkie C. Culambot).
  + **version**: The version number of the module (1.2).
  + **category**: The category to which the module belongs, specified as "Real Estate/Brokerage."
* **Dependencies:**
  + **depends**: Lists dependencies on other Odoo modules. In this case, it depends on the 'base' module.
* **Data Files and Views:**
  + **data**: Lists XML files containing data records and configurations to be loaded when the module is installed or upgraded. This includes security definitions, access control rules, report templates, and various views related to estate properties, offers, types, tags, menus, and user views.
  + **demo**: Lists XML files containing demonstration data to showcase the functionality of the module.
* **Sequence and Installability:**
  + **sequence**: Specifies the sequence in which this module should be installed relative to other modules.
  + **installable**: Indicates whether the module is installable.
  + **application**: Indicates whether the module is an application module.

# **B. SCREENSHOTS**

MODULE APP INFORMATION VIEW

A screenshot of a computer

Description automatically generated

MODULE INSTALLATION VIEW

A screenshot of a computer

Description automatically generated

MODULE MAIN VIEW

A screenshot of a computer

Description automatically generated

MODULE NEW PROPERTY VIEW

A screenshot of a computer

Description automatically generated

PDF REPORT VIEW

A screenshot of a computer

Description automatically generated

PROPERTY TYPES

A screenshot of a computer

Description automatically generated

PROPERTY TAGS

A screenshot of a computer

Description automatically generated

OVERALL VIEW

A screenshot of a computer

Description automatically generated