

Business Overview

A property management company oversees multiple residential and commercial buildings. Tenants often raise service requests (e.g., plumbing, elevator failure, pest control). The company wants a full internal management system in Odoo to track these requests from tenants, assign them to staff, and report on service efficiency.

Core Models

`property.building`

- `name`: Char
- `address`: Char
- `manager_id`: Many2one (`res.users`)

`property.unit`

- `name`: Char (e.g., Unit A1)
- `floor`: Integer
- `building_id`: Many2one to `property.building`
- `unit_type`: Selection (residential, commercial)

`property.tenant`

- `name, email, phone`: Basic contact fields
- `unit_id`: Many2one to `property.unit`
- `building_id`: Related field from `unit_id.building_id`

- Smart button to view related service requests

property.service.category

- Custom model to manage categories like Plumbing, HVAC, Cleaning

property.service.request

- `title, description, request_date, status`: Char/Date/Selection (draft, in_progress, resolved, rejected)
- `tenant_id`: Many2one to `property.tenant`
- `unit_id, building_id`: Related from tenant
- `category_id`: Many2one to `property.service.category`
- `assigned_to`: Many2one (`res.users`)
- `resolution_note`: Text
- `priority`: Selection (low, medium, high, urgent)
- `sla_deadline`: Computed Date based on request date and priority
- `is_late`: Boolean (True if `sla_deadline` passed and not resolved)



Business Logic

Dynamic Assignments

- Selecting `tenant_id` automatically populates `unit_id` and `building_id`.
- Field `sla_deadline` is computed based on priority:

- low: +5 days
- medium: +3 days
- high: +1 day
- urgent: same day

Buttons

- “Assign to Me” — assigns the request to current user.
- “Mark Resolved” — moves request to `resolved` and requires `resolution_note`.

Constraints & Validations

- Cannot resolve if `assigned_to` is empty.
 - Cannot delete resolved requests.
 - SLA alerts: Set `is_late=True` via cron job or automated compute if `sla_deadline < today` and status is not resolved.
-



Security & Access Rights

Groups:

- `Service Staff`: Can view and manage assigned service requests.
- `Service Manager`: Full access to all requests and reports.
- `Tenant Portal`: Can view their own requests (portal access only).

Rules:

- Tenants can only read their own requests.

- Service staff can only write requests assigned to them.
- Only managers can delete or reject requests.



Views to Create

◆ `property.building`

Views to create:

- List View: Show `name`, `address`, `manager_id`
- Form View:
 - Tab 1: Building Info
 - Tab 2: Smart button showing total Service Requests linked via units
- Search View: Add filters for `manager_id`

Smart Buttons:

- “Service Requests” → One2many inverse to `property.service.request` via unit
-

◆ `property.unit`

Views to create:

- List View: Show `name`, `floor`, `unit_type`, `building_id`
- Form View: Show details with building info
- Search View: Filter by building, unit type

◆ **property.tenant**

Views to create:

- List View: `name, unit_id, building_id, email, phone`
- Form View:
 - Tab 1: Tenant Info
 - Tab 2: Smart button → Service Requests by this tenant
- Search View: Filter by building, unit, phone

Smart Buttons:

- “My Service Requests” → One2many to `property.service.request`
-

◆ **property.service.category**

Views to create:

- Simple Tree and Form Views (admin only)
 - Used in dropdown for request categories
-

◆ **property.service.request**

This is the core model. It needs **several views**:

Tree View

- Columns: `title, tenant_id, building_id, category_id, status, priority, assigned_to, is_late`

Form View

- Top Section: `title, request_date, status, priority`
- First Tab (Details): `tenant_id`, auto-filled `unit_id, building_id, category_id, description`
- Second Tab (Assignment & Resolution): `assigned_to, resolution_note, sla_deadline, is_late`
- Chatter: Track status changes and assignment history
- Buttons:
 - “Assign to Me” (visible if not assigned)
 - “Mark as Resolved” (visible if in progress)

Kanban View

- Grouped by `status`
- Show `title, priority, tenant_id`, and assigned user
- Color by priority
- Include drag-and-drop for status

Calendar View (Optional)

- `request_date` as the date field
- Show service title, assigned staff

Search View

- Filters: `status, priority, assigned_to, is_late, building_id`
- Group By: `building_id, assigned_to, category_id, status`

Menu Structure

Main Menu: Property Management

- **Buildings**
 - Menu: `Buildings → property.building`
 - Menu: `Units → property.unit`
- **Tenants**
 - Menu: `Tenants → property.tenant`
- **Service**
 - Menu: `Service Requests → property.service.request` (default to Kanban)
 - Submenu: `Service Categories → property.service.category`

Freelance Project Scope: Automated PDF Merge and Email Distribution

Project Overview: Develop a standalone Python script that automatically merges daily incoming PDF files and emails the combined PDF to designated stakeholders.

Deliverables

Core Functionality

- **Automated PDF Merging:** Script that monitors a designated folder/location for incoming PDF files and merges them into a single combined PDF
- **Email Distribution:** Automatically send the merged PDF to a configurable list of stakeholders
- **Scheduling Capability:** Daily automated execution at specified time(s)
- **File Management:** Organize processed files (move to archive folder or maintain processing history)

Configuration Requirements

- **Flexible Input Sources:** Support monitoring local folders or network drives for incoming PDFs
- **Configurable Recipients:** Easy-to-update stakeholder email list without code modification
- **Customizable Settings:**
 - Email subject and body text
 - Merge frequency (daily timing)
 - File naming conventions for merged PDFs
 - Source and destination folder paths
- **Error Handling:** Logging and notification for failures (missing files, merge errors, email delivery issues)

Implementation Requirements

- **Modular Design:** Separate components for:
 - PDF collection and validation
 - PDF merging operations
 - Email sending functionality
 - Scheduling and automation
 - Logging and error handling
- **Configuration File:** All parameters (email credentials, folder paths, recipients, timing) managed through configuration file
- **Documentation:** Clear setup and usage instructions
- **Duplicate Detection:** Identify and skip duplicate files based on filename

Deliverables Package

1. Complete Python script(s) organized by functionality
2. Dependency specifications (requirements file)
3. Configuration file template
4. Documentation including:
 - Installation and setup guide
 - Configuration instructions
 - Scheduling setup (cron job or Task Scheduler)
 - Troubleshooting guide
5. Example usage and testing instructions

Out of Scope

- Web interface or dashboard
- Production server deployment

Success Criteria

The delivered script should:

- Successfully merge multiple PDFs into a single file
- Send emails with merged PDF attachment
- Run automatically on schedule
- Handle common error scenarios gracefully
- Be configurable without code changes
- Include clear logs for troubleshooting