

CIFP JUAN DE COLONIA

Sistemas de Gestión Empresarial

Memorias Proyecto Final Hospital

Roberto Tiron & Alexis López Briongos Dam2t

23/02/2024

Índice

1. Introducción	2
2. Contexto del Hospital	2
3. Diseño del módulo	3
4. Funcionalidades	8
5. Modelo de datos.....	13
6. Vistas y Formularios.....	20
7. Proceso de desarrollo	24
8. Conclusiones	25

1. Introducción

- El proyecto consiste en el desarrollo de un módulo para Odoo que se enfoca en la gestión integral de un hospital o ambulatorio. Este módulo tiene como objetivo optimizar los procesos administrativos y clínicos dentro de la institución médica, permitiendo una gestión eficiente de los recursos y una atención de calidad para los pacientes.

Objetivos:

- **Mejorar la eficiencia administrativa:** Automatizando tareas como la gestión de citas médicas, registro de pacientes, asignación de camas, entre otros, se busca agilizar los procesos administrativos del hospital.
- **Facilitar la atención médica:** Proporcionando a los profesionales de la salud herramientas para acceder de manera rápida y precisa a la información de los pacientes, historiales médicos, diagnósticos, tratamientos, etc., se pretende mejorar la calidad y la rapidez en la atención médica.
- **Centralizar la información:** Integrando todos los aspectos de la gestión hospitalaria en un único sistema, se busca centralizar la información y evitar la duplicación de datos, lo que contribuye a una toma de decisiones más informada y a una mejor coordinación entre los diferentes departamentos del hospital.

2. Contexto del Hospital

- El módulo se está desarrollando para su implementación en un entorno hospitalario moderno y dinámico.
- Este hospital puede variar en tamaño, desde pequeños ambulatorios hasta grandes centros médicos, y busca optimizar sus procesos mediante el uso de tecnología.
- Se espera que el sistema sea escalable y adaptable a las necesidades específicas de cada institución, permitiendo una fácil configuración y personalización según los requerimientos del hospital.

3. Diseño del módulo

- Este módulo se compone de 6 csv, 9 modelos y 10 vistas:

Estructura del proyecto

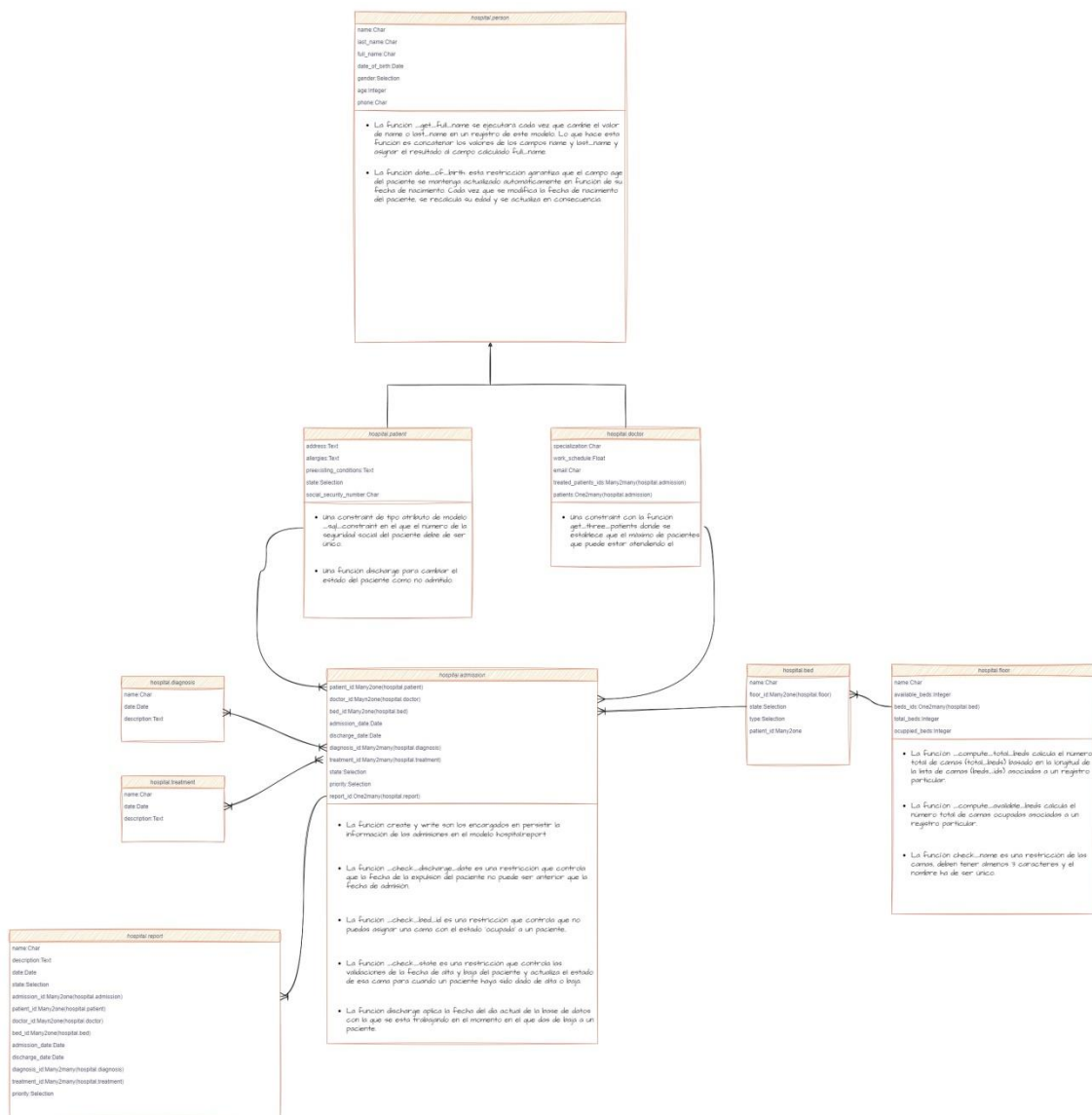
Hospital

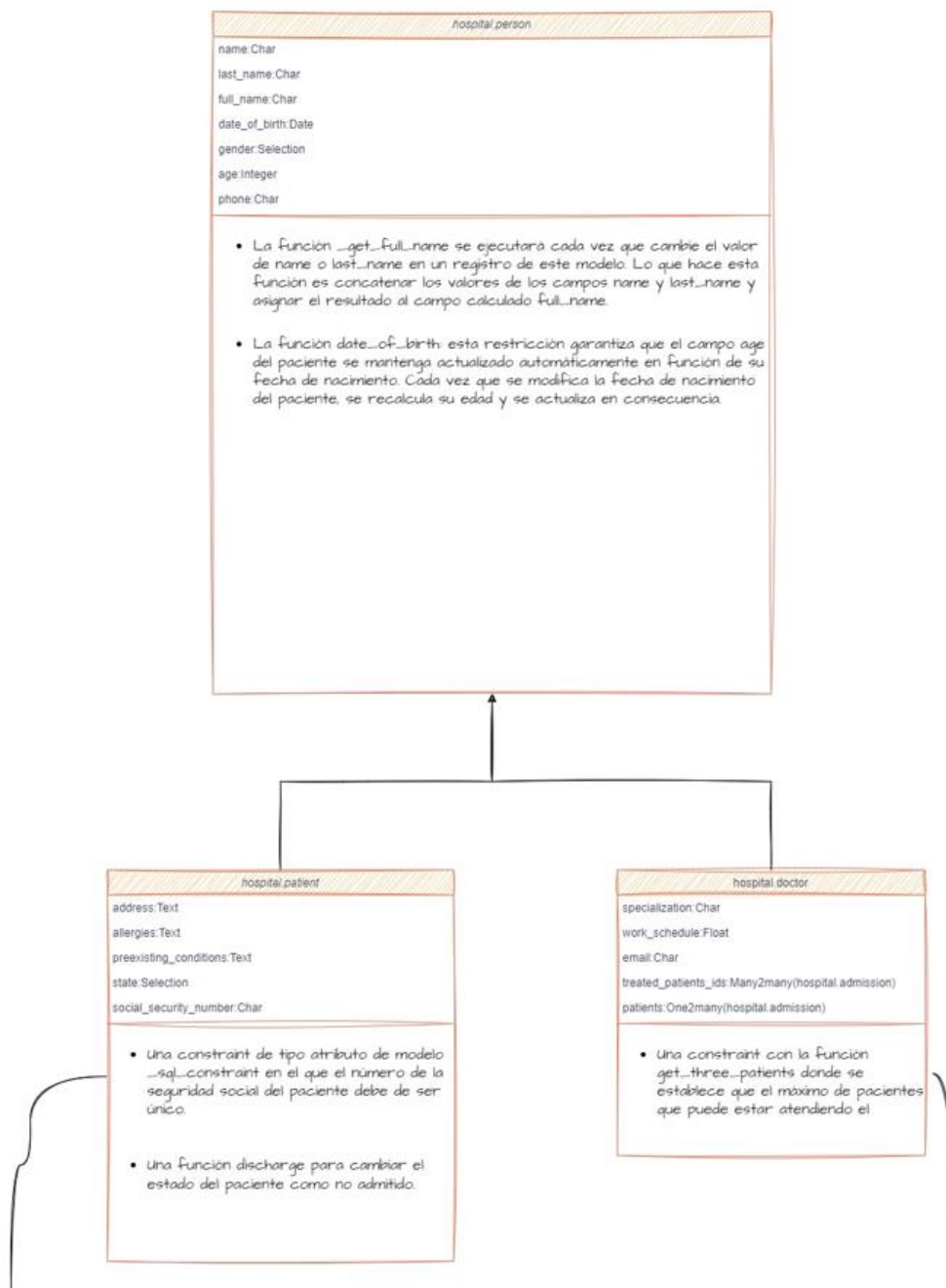
- ❖ controllers
 - `_init_.py`
 - `controllers.py`
- ❖ data
 - `hospital.bed.csv`
 - `hospital.diagnosis.csv`
 - `hospital.doctor.csv`
 - `hospital.floor.csv`
 - `hospital.patient.csv`
 - `hospital.treatment.csv`
- ❖ demo
 - `demo.xml`
- ❖ models
 - `_init_.py`
 - `Admission.py`
 - `Bed.py`
 - `Diagnosis.py`
 - `Doctor.py`
 - `Floor.py`
 - `Patient.py`
 - `Person.py`
 - `Report.py`
 - `Treatment.py`
- ❖ security
 - `ir.model.access.csv`
- ❖ static
 - `description`
 - `icon.png`
 - `src`
 - `scss`
 - `menus.scss`
 - `views.scss`
- ❖ views
 - `Admission.xml`
 - `Bed.xml`
 - `Diagnosis.xml`

- Doctor.xml
- Floor.xml
- Menu.xml
- Patient.xml
- Report.xml
- template.xml
- Treatment.xml

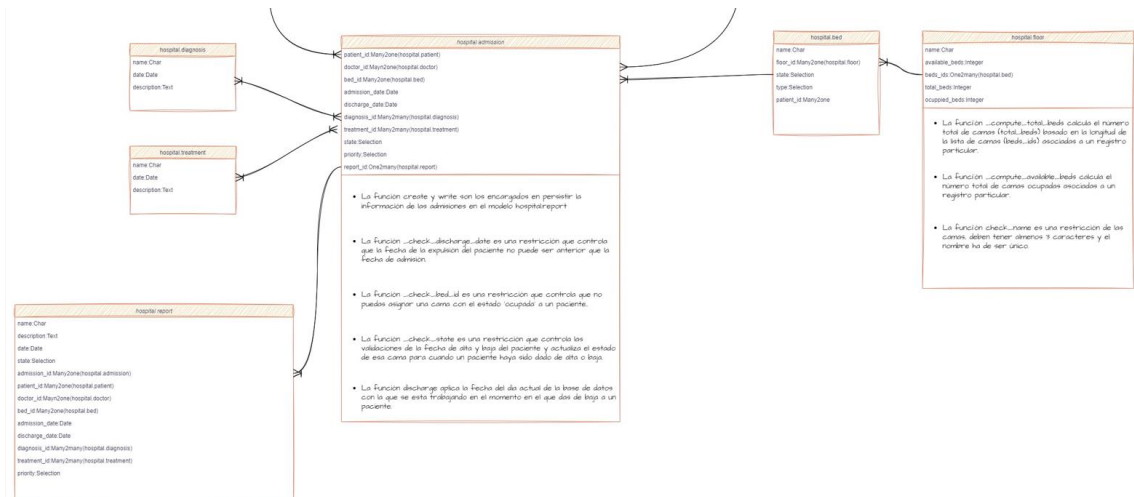
- ❖ _init_.py
- ❖ _manifest_.py
- ❖ pythonBedsGenerateCSV.py
- ❖ pythonDiagnosisGenerateCSV.py
- ❖ pythonDoctorsGenerateCSV.py
- ❖ pythonFloorGenerateCSV.py
- ❖ pythonTreatmentGenerateCSV.py

Diagrama UML

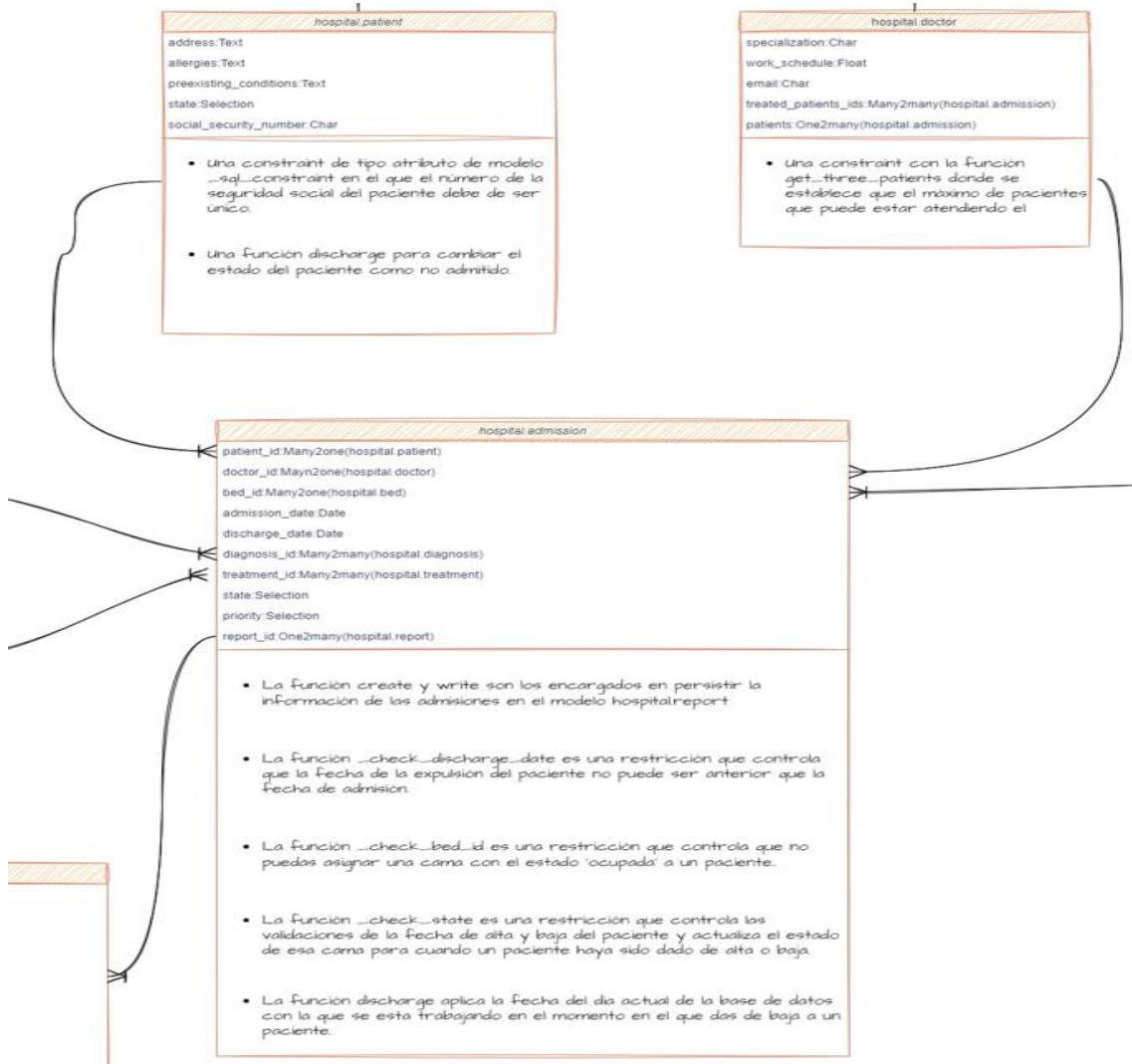




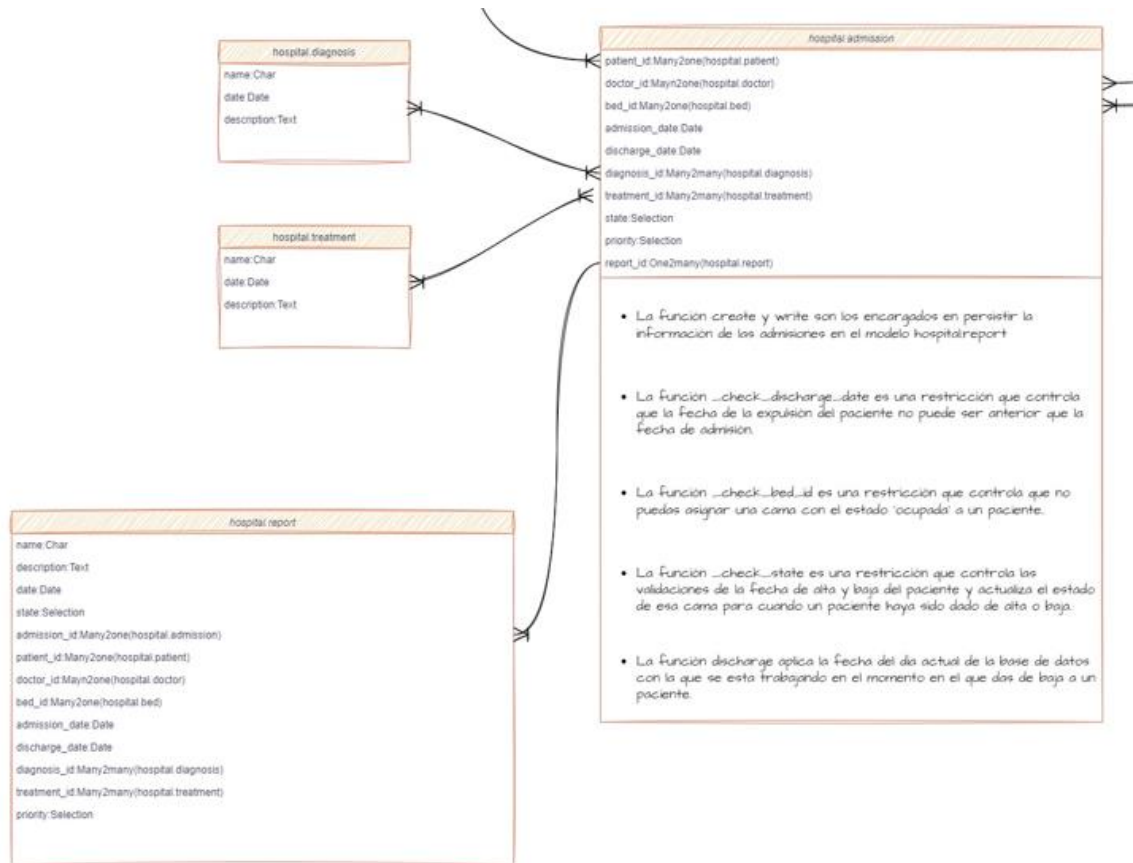
Los modelos *hospital.patient* y *hospital.doctor* heredando de *hospital.person*.



*El modelo **hospital.admission** relacionándose con todos los modelos necesarios para poder pasar una consulta y dar de alta a un paciente.*



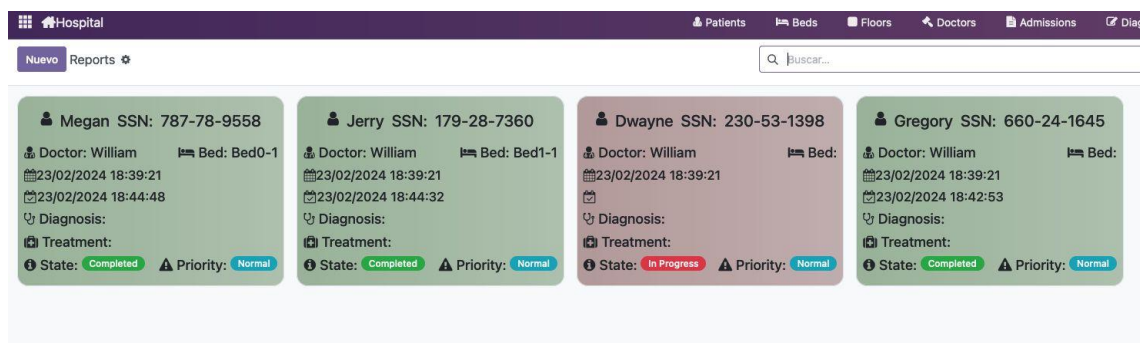
*El modelo **hospital.admission** relacionado con **hospital.patient** y **hospital.doctor**(Many2One)*



Relación entre el modelo hospital.admission y hospital.report para poder persistir los datos del historial de admisiones.

Diseño

- Todos los modelos de este módulo soportan todas las vistas disponibles en Odoo para mostrar las tablas de la base de datos, pero en este caso nos hemos centrado en desarrollar una interfaz atractiva e intuitiva para el usuario basado en la vista Kanban



Ejemplo de la vista kanban del modelo hospital.report

4. Funcionalidades

- Este módulo está organizado por áreas o módulos relacionados (por ejemplo, gestión de pacientes, citas médicas, historial médico, etc.).

The image shows a file explorer window with a 'data' directory. Inside, there are several CSV files: hospital.bed.csv, hospital.diagnosis.csv, hospital.doctor.csv, hospital.floor.csv, hospital.patient.csv, and hospital.treatment.csv. To the right, a Python script is shown with a list of file paths to be loaded.

```
# always loaded
'data': [
    'views/Patient.xml',
    'views/Floor.xml',
    'views/Bed.xml',
    'views/Doctor.xml',
    'views/Diagnosis.xml',
    'views/Treatment.xml',
    'views/Report.xml',
    'views/Admission.xml',
    'views/Menu.xml',
    'data/hospital.patient.csv',
    'data/hospital.floor.csv',
    'data/hospital.bed.csv',
    'data/hospital.doctor.csv',
    'data/hospital.diagnosis.csv',
    'data/hospital.treatment.csv',
]
```

The image shows a file explorer window with several Python scripts for generating CSV data:

- __init__.py
- __manifest__.py
- pythonBedsGenerateCSV.py
- pythonDiagnosisGenerateCSV.py
- pythonDoctorsGenerateCSV.py
- pythonFloorGenerateCSV.py
- pythonPatientsGenerateCSV.py
- pythonTreatmentGenerateCSV.py

Admission.py hospital.patient.csv

+ - 🔍 ↺

CSV ▾ ⬇ ⬆ ⬇

- Para obtener los datos de los registros utilizaremos documentos csv que deberán de estar denominados que igual que los modelos y que se cargarán cada vez que se ejecute el módulo añadiéndolos en el `__manifest__.py`.
- También hemos realizado la creación de una clase específica para generar los valores de los registros de cada modelo.

```
import csv
import random
from faker import Faker
from datetime import datetime, timedelta

fake = Faker()

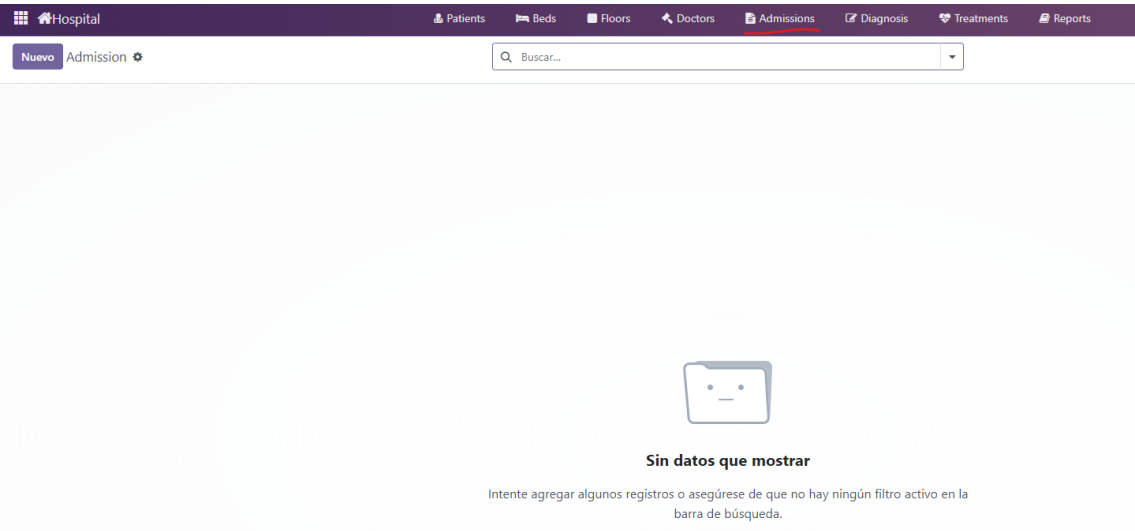
# Función para generar una fecha de nacimiento aleatoria
! usage -t rjt
def generate_date_of_birth():
    end_date = datetime.now() - timedelta(days=365 * 18) # Hace que los pacientes tengan al menos 18 años
    start_date = end_date - timedelta(days=365 * 90) # Hace que los pacientes tengan hasta 90 años
    return fake.date_between(start_date=start_date, end_date=end_date)

# Generar datos para 1000 pacientes extendidos
patients_data = []
for _ in range(20):
    name = fake.first_name()
    last_name = fake.last_name()
    date_of_birth = generate_date_of_birth()
    gender = random.choice(['male', 'female', 'other'])
    address = fake.address()
    phone = fake.phone_number()
    allergies = fake.sentence(nb_words=6)
    preexisting_conditions = fake.sentence(nb_words=8)
    social_security_number = fake.ssn()

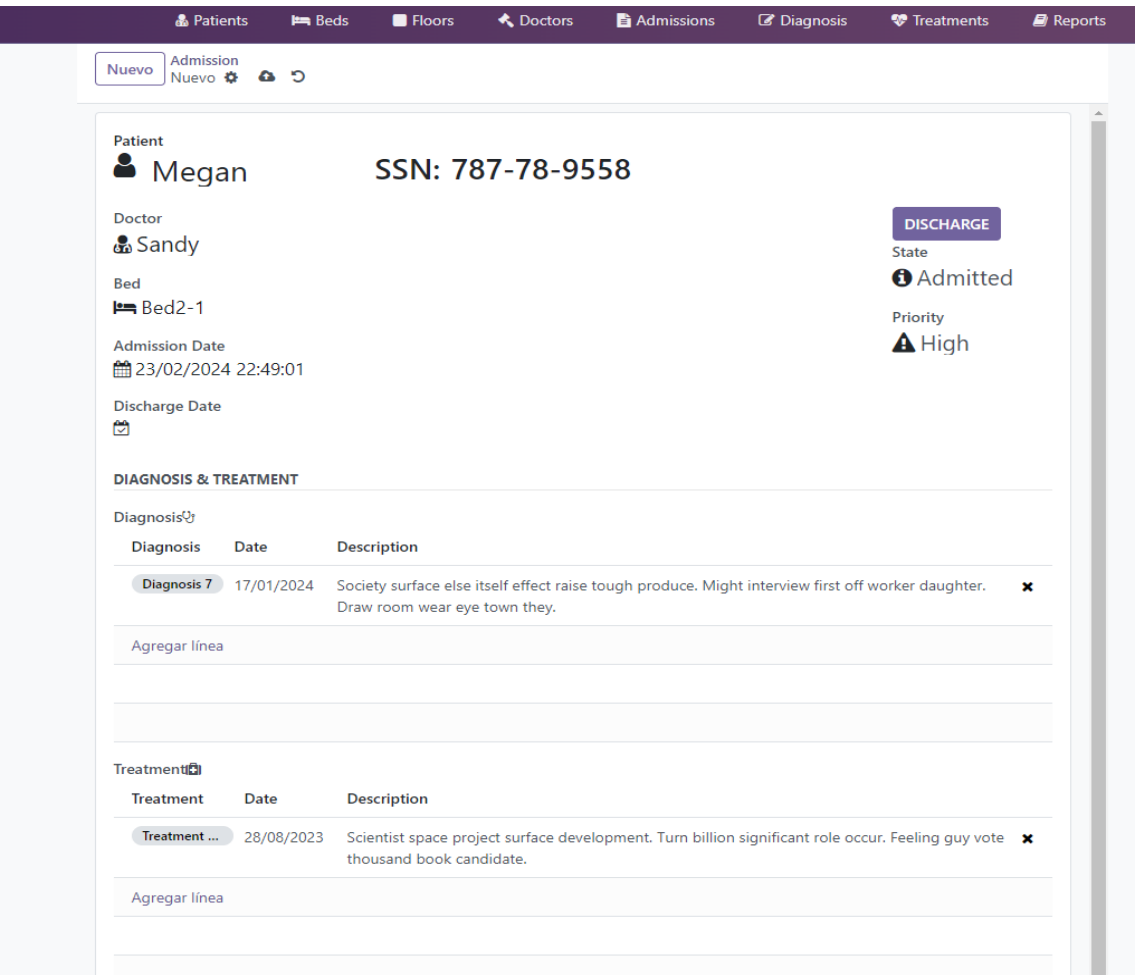
    patient = {
        'name': name,
        'last_name': last_name,
        'date_of_birth': date_of_birth,
        'gender': gender,
        'address': address,
        'phone': phone,
        'allergies': allergies,
        'preexisting_conditions': preexisting_conditions,
        'social_security_number': social_security_number,
    }
    patients_data.append(patient)
```

Clase encargada de generar valores para los registros del hospital.patient.csv

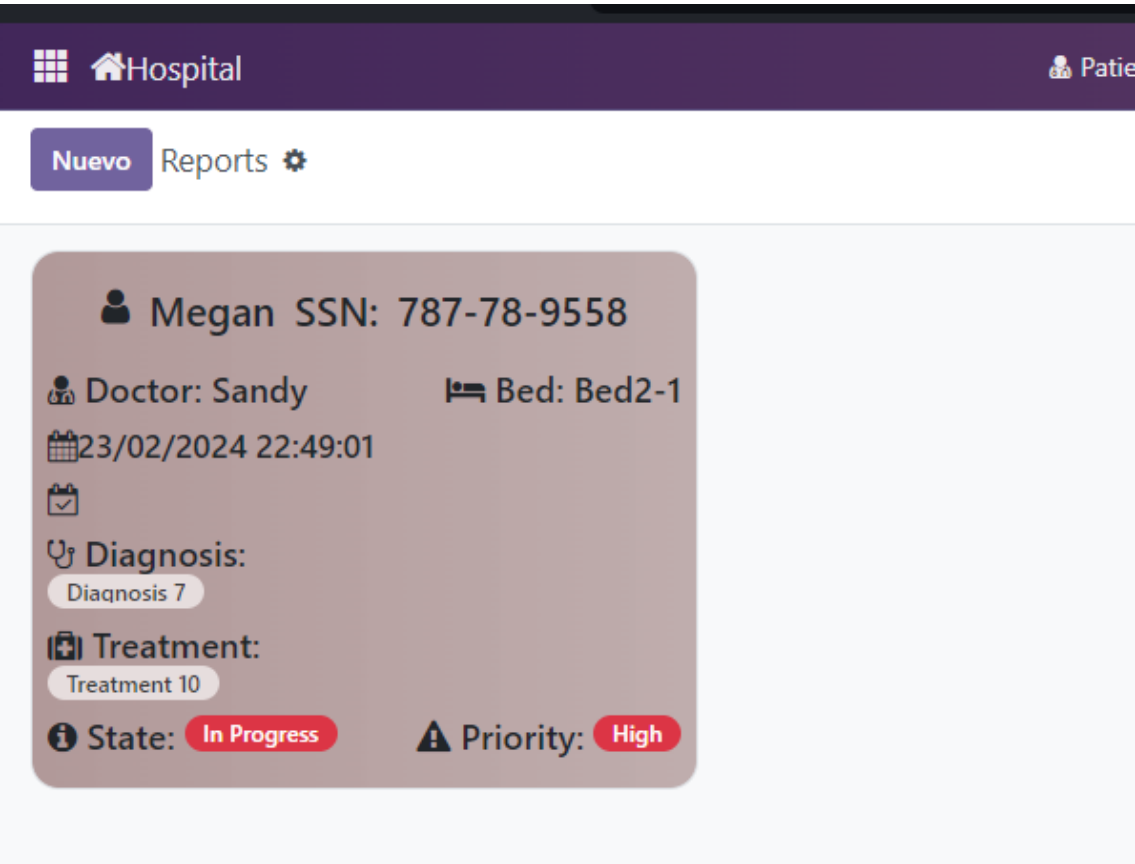
- Para poder generar los datos del csv utilizaremos la librería Faker.
- Esta librería tiene métodos para poder generar datos aleatorios pero específicos como pueden ser fechas, direcciones o seleccionar entre los valores que tú le indiques.



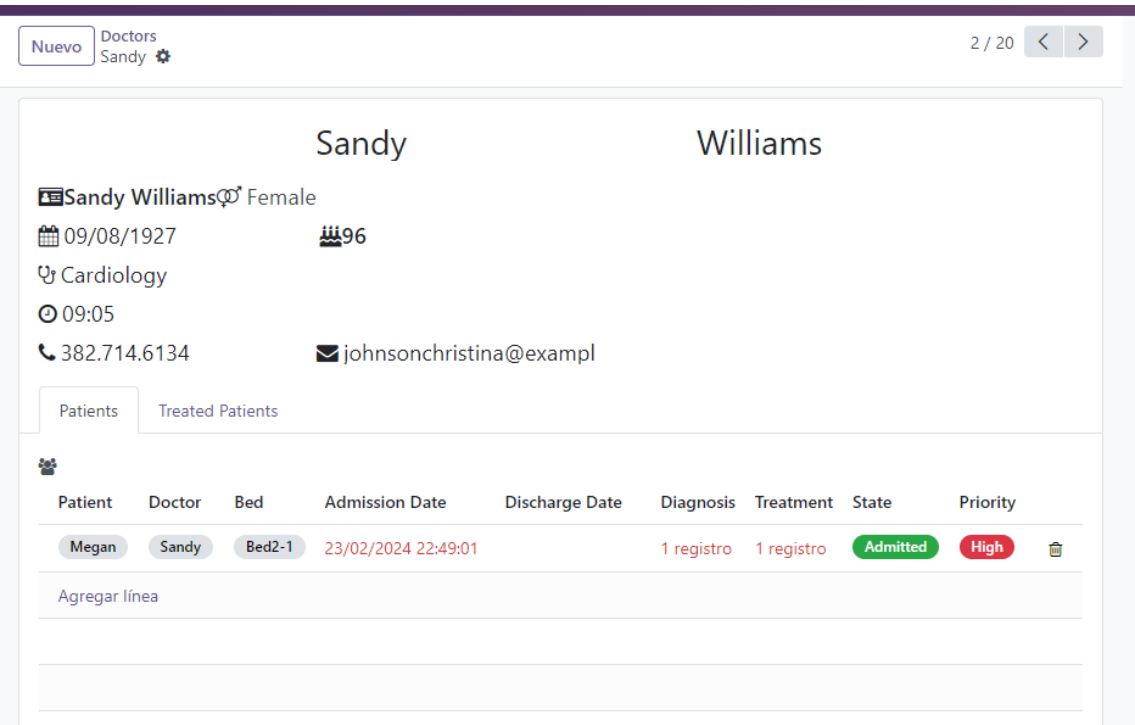
- Después de tener registros en todos los modelos anteriormente mencionados, la funcionalidad principal de este modelo es crear una solicitud de admisión para un paciente en el centro sanitario.



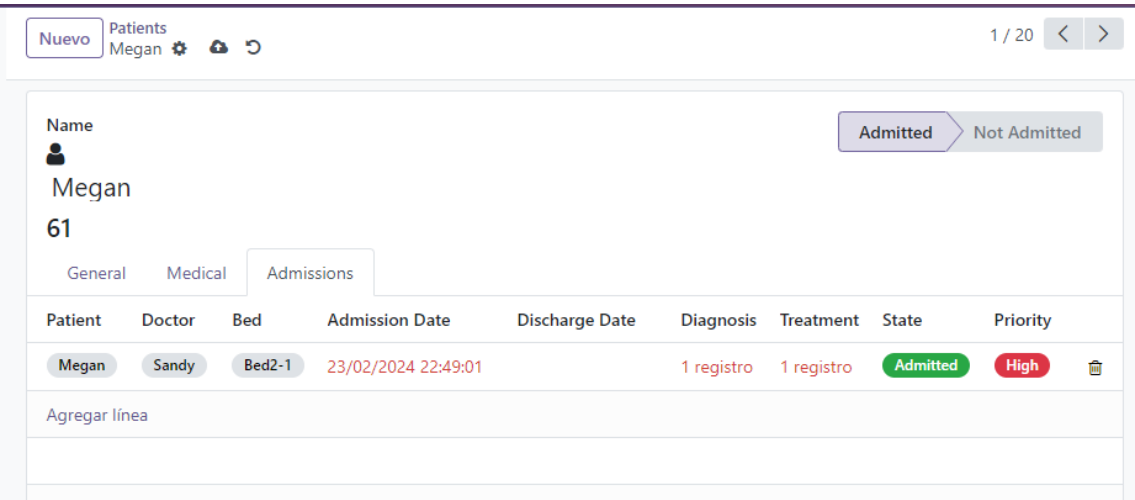
- Aquí podemos visualizar todos los campos necesarios como el doctor que lo atiende, la fecha, la prioridad, etc.
- Para realizar la gestión de una admisión de un nuevo paciente al centro sanitario.



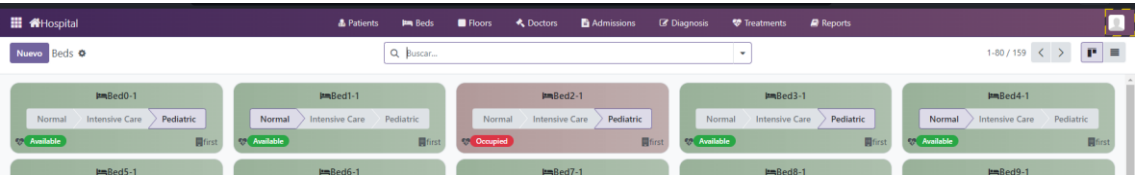
- Después en el modelo de hospital.report podemos visualizar las consultas de los pacientes.



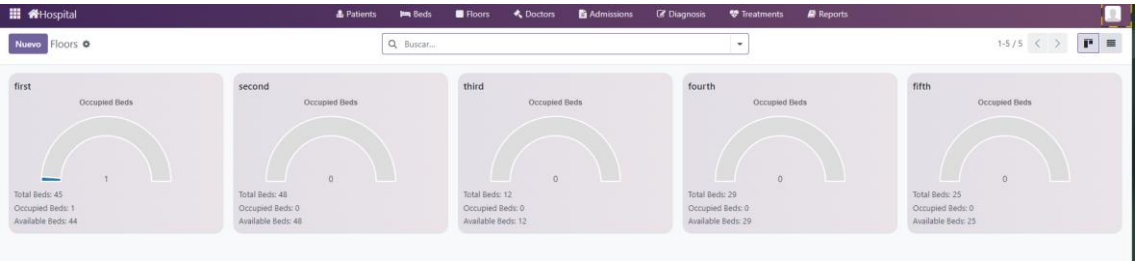
- En el modelo de hospital.doctor podemos visualizar los doctores todos los con pacientes activos.



- También podemos visualizar las admisiones activas en el modelo hospital.patient.



- En el modelo hospital.bed podemos ver que camas están ocupadas.

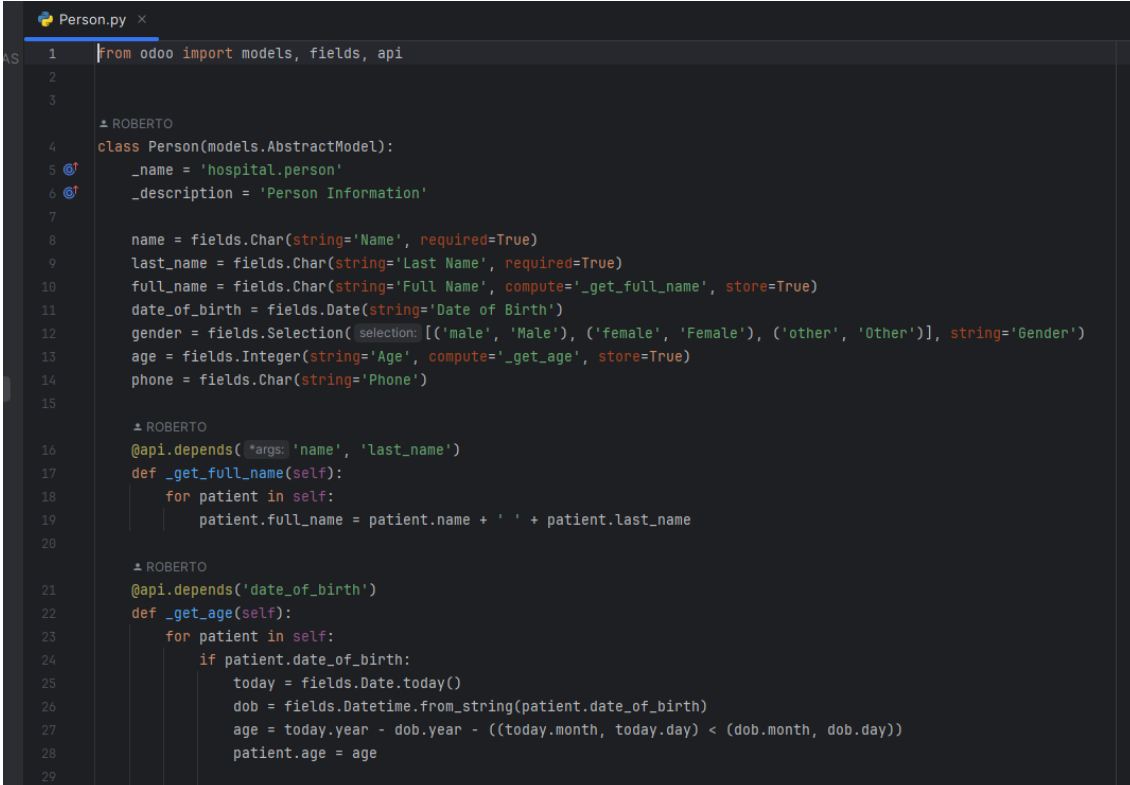


- En el modelo hospital.floor podemos visualizar todas las plantas y las capas ocupadas en cada planta.

5. Modelo de datos

- El módulo se compone de 9 modelos que iremos comentando a continuación.

hospital.person



```
Person.py x
1 from odoo import models, fields, api
2
3
4 class Person(models.AbstractModel):
5     _name = 'hospital.person'
6     _description = 'Person Information'
7
8     name = fields.Char(string='Name', required=True)
9     last_name = fields.Char(string='Last Name', required=True)
10    full_name = fields.Char(string='Full Name', compute='_get_full_name', store=True)
11    date_of_birth = fields.Date(string='Date of Birth')
12    gender = fields.Selection(selection=[('male', 'Male'), ('female', 'Female'), ('other', 'Other')], string='Gender')
13    age = fields.Integer(string='Age', compute='_get_age', store=True)
14    phone = fields.Char(string='Phone')
15
16    @api.depends('name', 'last_name')
17    def _get_full_name(self):
18        for patient in self:
19            patient.full_name = patient.name + ' ' + patient.last_name
20
21    @api.depends('date_of_birth')
22    def _get_age(self):
23        for patient in self:
24            if patient.date_of_birth:
25                today = fields.Date.today()
26                dob = fields.Datetime.from_string(patient.date_of_birth)
27                age = today.year - dob.year - ((today.month, today.day) < (dob.month, dob.day))
28                patient.age = age
29
```

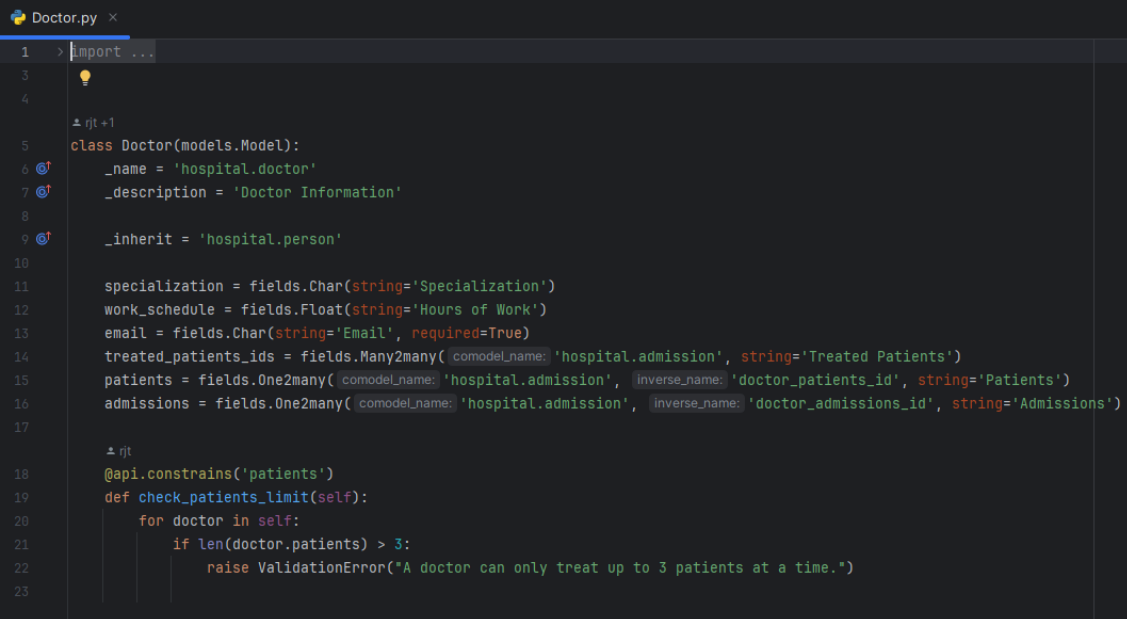
- La clase `hospital.person` proporciona una estructura básica para almacenar información sobre personas en un entorno hospitalario, incluyendo campos para atributos clave y métodos para calcular valores derivados.
- Las clases que heredan de esta clase pueden extenderla según sea necesario para adaptarse a requisitos específicos, como pacientes, médicos, personal administrativo, etc.

hospital.patient

```
1 from odoo import models, fields
2
3
4 class Patient(models.Model):
5     _name = 'hospital.patient'
6     _description = 'Patient Information'
7
8     _inherit = 'hospital.person'
9
10    address = fields.Text(string='Address')
11    allergies = fields.Text(string='Allergies')
12    preexisting_conditions = fields.Text(string='Preexisting Conditions')
13
14    state = fields.Selection(selection=[
15        ('admitted', 'Admitted'),
16        ('not_admitted', 'Not Admitted'),
17    ], string='State', default='not_admitted')
18
19    social_security_number = fields.Char(string='Social Security Number')
20
21    admission_ids = fields.One2many(comodel_name='hospital.admission', inverse_name='patient_id', string='Admissions')
22
23    # Creamos una condicion para que el numero de seguro social sea unico
24    _sql_constraints = [
25        ('social_security_number_unique',
26         'UNIQUE(social_security_number)',
27         'The Social Security Number must be unique.')
28    ]
29
```

- La clase Patient extiende la funcionalidad de la clase hospital.person para representar específicamente a los pacientes en un sistema hospitalario, agregando campos adicionales y estableciendo relaciones relevantes para gestionar la información de los pacientes de manera efectiva.

hospital.doctor



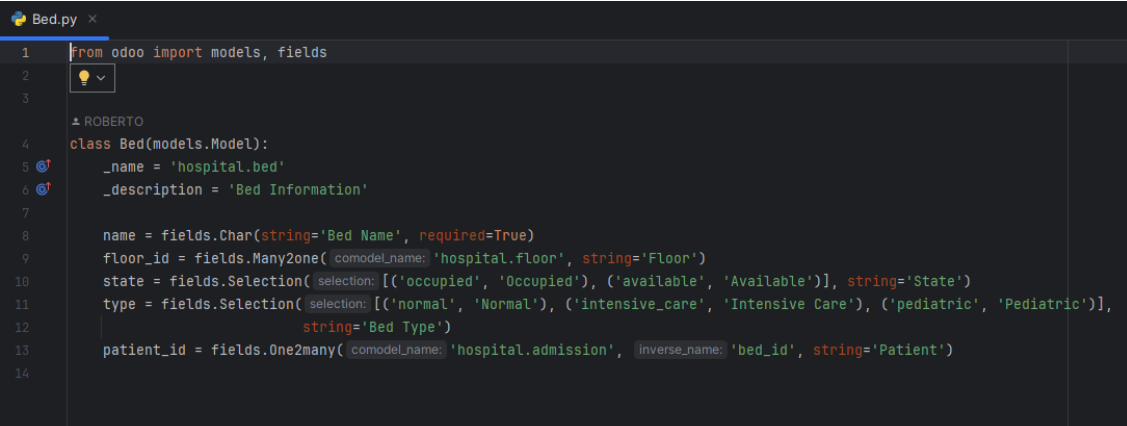
```

1  > import ...
2
3
4
5  class Doctor(models.Model):
6      _name = 'hospital.doctor'
7      _description = 'Doctor Information'
8
9      _inherit = 'hospital.person'
10
11     specialization = fields.Char(string='Specialization')
12     work_schedule = fields.Float(string='Hours of Work')
13     email = fields.Char(string='Email', required=True)
14     treated_patients_ids = fields.Many2many(comodel_name='hospital.admission', string='Treated Patients')
15     patients = fields.One2many(comodel_name='hospital.admission', inverse_name='doctor_patients_id', string='Patients')
16     admissions = fields.One2many(comodel_name='hospital.admission', inverse_name='doctor_admissions_id', string='Admissions')
17
18     @api.constrains('patients')
19     def check_patients_limit(self):
20         for doctor in self:
21             if len(doctor.patients) > 3:
22                 raise ValidationError("A doctor can only treat up to 3 patients at a time.")
23

```

- En resumen, la clase Doctor en el modelo de datos del hospital extiende la funcionalidad básica de una persona para representar a los médicos, agregando campos específicos de médicos y estableciendo relaciones relevantes para gestionar la información de los médicos y sus interacciones con los pacientes y las admisiones hospitalarias.

hospital.bed



```

1  from odoo import models, fields
2
3
4  class Bed(models.Model):
5      _name = 'hospital.bed'
6      _description = 'Bed Information'
7
8      name = fields.Char(string='Bed Name', required=True)
9      floor_id = fields.Many2one(comodel_name='hospital.floor', string='Floor')
10     state = fields.Selection(selection=[('occupied', 'Occupied'), ('available', 'Available')], string='State')
11     type = fields.Selection(selection=[('normal', 'Normal'), ('intensive_care', 'Intensive Care'), ('pediatric', 'Pediatric')], string='Bed Type')
12     patient_id = fields.One2many(comodel_name='hospital.admission', inverse_name='bed_id', string='Patient')
13
14

```

- La clase Bed en el modelo de datos del hospital representa las camas del hospital, proporcionando información sobre su nombre, estado, tipo y la relación con la planta del hospital en la que se encuentran, así como la información del paciente que ocupa la cama en un momento dado.

hospital.floor

```

1  from odoo import models, fields, api
2
3
4  class Floor(models.Model):
5      _name = 'hospital.floor'
6      _description = 'Floor Information'
7
8      name = fields.Char(string='Floor Name', required=True)
9      available_beds = fields.Integer(string='Available Beds', compute='_compute_available_beds')
10     beds_ids = fields.One2many(comodel_name='hospital.bed', inverse_name='floor_id', string='Beds')
11
12     total_beds = fields.Integer(string='Total Beds', compute='_compute_total_beds')
13     occupied_beds = fields.Integer(string='Occupied Beds', compute='_compute_occupied_beds')
14
15     @api.depends('beds_ids')
16     def _compute_total_beds(self):
17         for record in self:
18             record.total_beds = len(record.beds_ids)
19
20     @api.depends('beds_ids.state')
21     def _compute_occupied_beds(self):
22         for record in self:
23             record.occupied_beds = len(record.beds_ids.filtered(lambda bed: bed.state == 'occupied'))
24
25     @api.depends('beds_ids.state')
26     def _compute_available_beds(self):
27         for record in self:
28             record.available_beds = len(record.beds_ids.filtered(lambda bed: bed.state == 'available'))
29
30     @api.constrains('name')
31     def check_name(self):
32         for record in self:
33             if record.name and len(record.name) < 3:
34                 raise models.ValidationError("The name must have at least 3 characters.")
35             if record.name and self.env['hospital.floor'].search_count([('name', '=', record.name)]) > 1:
36                 raise models.ValidationError("The name must be unique.")
37

```

- La clase Floor en el modelo de datos del hospital representa los pisos del hospital, proporcionando información sobre el nombre del piso, el número total de camas, el número de camas ocupadas y disponibles, y la relación con las camas ubicadas en ese piso. Además, incluye restricciones para garantizar la integridad de los datos, como la unicidad del nombre del piso y la longitud mínima del nombre.

hospital.treatment

```
Treatment.py ×
1  from datetime import datetime
2
3  from odoo import models, fields
4
5
6  class Treatment(models.Model):
7      _name = 'hospital.treatment'
8      _description = 'Treatment Information'
9
10     name = fields.Char(string='Treatment', required=True)
11     date = fields.Date(string='Date', default=datetime.now())
12     description = fields.Text(string='Description')
```

- La clase Treatment en el modelo de datos del hospital almacena información sobre tratamientos médicos, incluyendo el nombre del tratamiento, la fecha en que se realizó y una descripción opcional del tratamiento. Esto permite llevar un registro de los tratamientos médicos administrados a los pacientes en el hospital.

hospital.diagnosis

```
Diagnosis.py ×
1  from datetime import datetime
2
3  from odoo import models, fields
4
5
6  class Diagnosis(models.Model):
7      _name = 'hospital.diagnosis'
8      _description = 'Diagnosis Information'
9
10     name = fields.Char(string='Diagnosis', required=True)
11     date = fields.Date(string='Date', default=datetime.now())
12     description = fields.Text(string='Description')
```

- La clase Diagnosis en el modelo de datos del hospital almacena información sobre diagnósticos médicos, incluyendo el nombre del diagnóstico, la fecha en que se realizó y una descripción opcional del mismo. Esto permite llevar un registro de los diagnósticos médicos realizados a los pacientes en el hospital.

hospital.admission

```

import ...

class Admission(models.Model):
    _name = 'hospital.admission'
    _description = 'Admission Information'

    name = fields.Char(string='Admission')
    patient_id = fields.Many2one(comodel_name='hospital.patient', string='Patient', required=True)
    patient_social_security_number = fields.Char(related='patient_id.social_security_number', string='Social Security')

    doctor_admissions_id = fields.Many2one(comodel_name='hospital.doctor', string='Doctor', required=True)
    doctor_patients_id = fields.Many2one(comodel_name='hospital.doctor', string='Doctor')

    bed_id = fields.Many2one(comodel_name='hospital.bed', string='Bed', domain=[('state', '=', 'available')])
    admission_date = fields.Datetime(string='Admission Date', default=datetime.now())
    discharge_date = fields.Datetime(string='Discharge Date')
    diagnosis_ids = fields.Many2many(comodel_name='hospital.diagnosis', string='Diagnosis')
    treatment_ids = fields.Many2many(comodel_name='hospital.treatment', string='Treatment')
    state = fields.Selection(selection=[('admitted', 'Admitted'), ('discharged', 'Discharged')], string='State',
                             default='admitted')
    priority = fields.Selection(selection=[('low', 'Low'), ('normal', 'Normal'), ('high', 'High')], string='Priority',
                               default='normal')

    report_id = fields.One2many(comodel_name='hospital.report', inverse_name='admission_id', string='Report')

```

```

Admission.py
29
30
31 @api.constrains('report_id')
32 def _check_report_id(self):
33     for admission in self:
34         if len(admission.report_id) > 1:
35             raise ValidationError('Only one report per admission')
36
37 @api.model
38 def create(self, vals):
39     record = super().create(vals)
40     record.name = "Admission for " + record.patient_id.full_name
41     report = self.env['hospital.report'].create({
42         'name': 'Report for ' + record.patient_id.name,
43         'admission_id': record.id,
44     })
45     record.patient_id.state = 'admitted'
46     if record.bed_id:
47         record.bed_id.state = 'occupied'
48
49     record.doctor_patients_id = record.doctor_admissions_id
50     record.report_id = [(0, 0, [report.id])]
51     return record
52
53 @api.model
54 def write(self, vals):
55     res = super().write(vals)
56     if 'state' in vals and vals['state'] == 'discharged':
57         for record in self:
58             if record.report_id:
59                 record.report_id.write({
60                     'state': 'completed',
61                 })
62     return res
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

└ ROBERTO
@api.constrains('bed_id')
def _check_bed_id(self):
    for admission in self:
        if admission.bed_id and admission.bed_id.state == 'occupied':
            raise ValidationError('Bed already occupied')

└ ROBERTO
@api.constrains('state')
def _check_state(self):
    for admission in self:
        if admission.state == 'discharged' and not admission.discharge_date:
            raise ValidationError('Discharge date is required')
        if admission.state == 'admitted' and admission.discharge_date:
            raise ValidationError('Discharge date must be empty')
        if admission.state == 'discharged' and admission.bed_id:
            admission.bed_id.state = 'available'
        if admission.state == 'admitted' and admission.bed_id:
            admission.bed_id.state = 'occupied'

```

- La clase Admission en el modelo de datos del hospital gestiona la información relacionada con las admisiones hospitalarias, incluyendo los pacientes admitidos, los médicos encargados de la admisión, las camas asignadas, los diagnósticos y tratamientos asociados, el estado de la admisión y los informes generados durante la admisión.
- También proporciona métodos para realizar acciones como la creación de una admisión, la actualización de su estado y el alta del paciente.

hospital.report

```

Report.py
1 from odoo import models, fields
2
3
4 class Report(models.Model):
5     _name = 'hospital.report'
6     _description = 'Report Information'
7
8     name = fields.Char(string='Report', required=True)
9     description = fields.Text(string='Description')
10    date = fields.Date(string='Date', default=fields.Date.today())
11    state = fields.Selection(selection=[('in_progress', 'In Progress'), ('completed', 'Completed')], string='State',
12                           default='in_progress')
13    admission_id = fields.Many2one(comodel_name='hospital.admission', string='Admission')
14    patient_id = fields.Many2one(comodel_name='hospital.patient', string='Patient', related='admission_id.patient_id', readonly=True)
15    patient_social_security_number = fields.Char(related='admission_id.patient_social_security_number',
16                                                string='Social Security', readonly=True)
17    doctor_id = fields.Many2one(comodel_name='hospital.doctor', string='Doctor', related='admission_id.doctor_admissions_id',
18                              readonly=True)
19    bed_id = fields.Many2one(comodel_name='hospital.bed', string='Bed', related='admission_id.bed_id', readonly=True)
20    admission_date = fields.Datetime(string='Admission Date', related='admission_id.admission_date', readonly=True)
21    discharge_date = fields.Datetime(string='Discharge Date', related='admission_id.discharge_date', readonly=True)
22    diagnosis_ids = fields.Many2many(comodel_name='hospital.diagnosis', string='Diagnosis.xml', related='admission_id.diagnosis_ids',
23                                   readonly=True)
24    treatment_ids = fields.Many2many(comodel_name='hospital.treatment', string='Treatment', related='admission_id.treatment_ids',
25                                   readonly=True)
26    priority = fields.Selection(selection=[('low', 'Low'), ('normal', 'Normal'), ('high', 'High')], string='Priority',
27                               related='admission_id.priority', readonly=True)

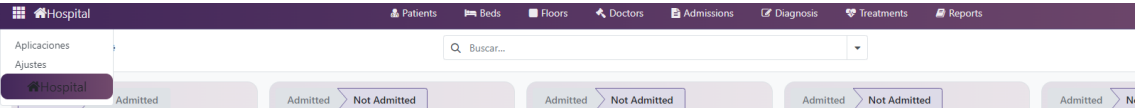
```

- La clase Report en el modelo de datos del hospital almacena información sobre los informes asociados a las admisiones hospitalarias. Esto incluye detalles sobre el paciente, el médico, la cama, la fecha de admisión y alta, los diagnósticos y tratamientos asociados, así como el estado del informe.

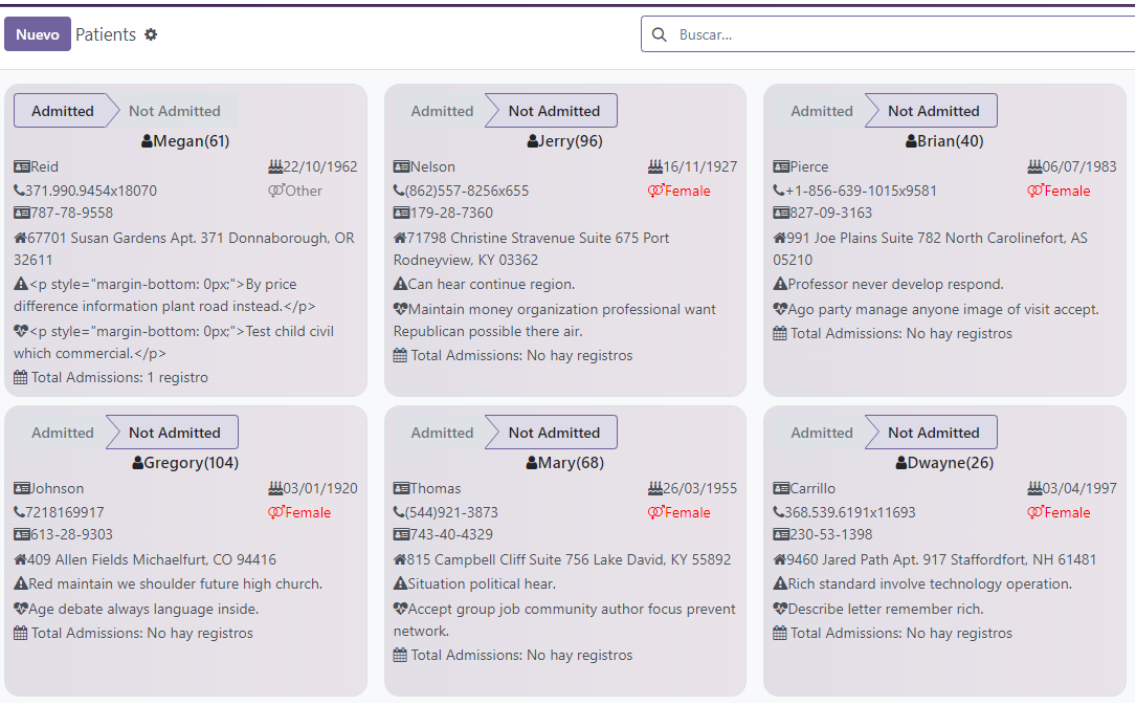
6. Vistas y Formularios

- Este módulo ofrece soporte para diferentes tipos de vistas, incluyendo listas y formularios. Sin embargo, nos centraremos en resaltar las vistas tipo kanban, ya que son visualmente más atractivas y ofrecen una presentación óptima de la información.

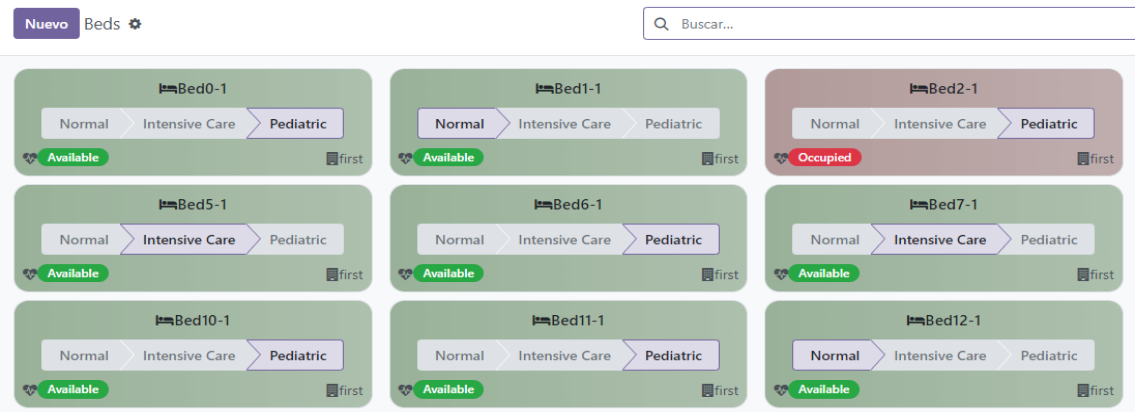
Hospital



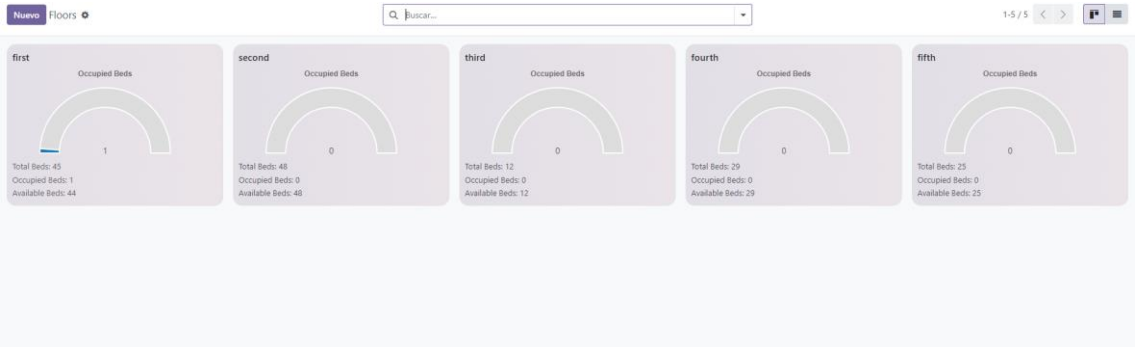
Patient



Bed



Floor



Doctor

Nuevo Doctors

Q

Buscar...

WilliamAllen(39)

Female

Gastroenterology

04:35

288-301-7392

cmartin@example.org

SandyWilliams(96)

Female

Cardiology

09:05

382.714.6134

johnsonchristina@example.org

MichaelBrock(82)

Male

Dermatology

04:10

001-300-729-6806x0645

daniel45@example.net

MichaelRhodes(38)

Other

Gastroenterology

08:40

798.396.2617

nielsenkayla@example.net

MichaelPace(27)

Female

Neurology

06:30

241.481.9457x65997

burchangela@example.org

AmyHarper(31)

Other

Cardiology

04:25

394.899.3880

kferguson@example.net

Página 21 de 25

Admission

Nuevo

Admission

Megan SSN: 787-78-9558

Doctor: Sandy

Bed: Bed2-1

23/02/2024 22:49:01

Diagnosis:

Diagnosis 7

Treatment:

Treatment 10

State: Admitted

Priority: High

Nuevo

Admission

Nuevo

Patient

SSN:

Doctor

Bed

Admission Date

23/02/2024 22:49:01

Discharge Date

DISCHARGE

State

Admitted

Priority

Normal

DIAGNOSIS & TREATMENT

Diagnosis

Diagnosis	Date	Description
Agregar línea		

Treatment

Treatment	Date	Description
Agregar línea		

Diagnosis

Nuevo

Diagnosis ⚙

Q

Buscar...

📅

24/02/2023

📝

Diagnosis 0

📌Success series size reality wear activity whatever. Once bring prepare sort water six would. Water think receive but various be this.

📅

02/03/2023

📝

Diagnosis 1

📌Phone that health end bring simply raise. Dinner religious itself want represent increase enjoy nature. Teacher part station best. Someone order yard always pick way piece.

📅

08/07/2023

📝

Diagnosis 2

📌Current realize region discover various. Down everybody view environment would use. Trouble parent believe manager save although street. Theory manage increase successful. Network store fish several.

📅

13/06/2023

📝

Diagnosis 5

📌Run respond effort detail would. Research son democratic knowledge less discover. After particularly book fast kind which. Activity travel listen.

📅

10/12/2023

📝

Diagnosis 6

📌Could political fall night out kitchen. She realize social thought. Role wind once. Message describe through film building. At production structure down central interesting huge.

📅

17/01/2024

📝

Diagnosis 7

📌Society surface else itself effect raise tough produce. Might interview first off worker daughter. Draw room wear eye town they.

Treatments

Nuevo

Treatments ⚙

Q

Buscar...

📅

02/12/2023

📝

Treatment 0

📌Authority west site approach within film game. Fall memory draw practice window politics dog second. Think among character collection military. Stuff treat day particular.

📅

25/06/2023

📝

Treatment 1

📌Often picture usually step large. Approach marriage avoid live represent to cell. Born radio situation machine drug shake scientist why.

📅

22/05/2023

📝

Treatment 2

📌Author service upon much he. Road near shake dream choice discussion it. Former law pass write apply manage cause.

📅

08/02/2024

📝

Treatment 5

📌Item defense fine sister. Child manager ok feeling. Land put game cultural must light. Try learn son particularly upon official. Political ask debate edge agent less candidate.

📅

15/02/2024

📝

Treatment 6

📌Car itself the too major agent over. Entire risk practice growth term both class. Red all data since parent likely.

📅

01/10/2023

📝

Treatment 7

📌Within town determine next situation parent. Any staff three. Class end well test bring ask itself turn. Treatment red effect get human huge. Child read game put.

Report

Nuevo

Reports ⚙

👤

Megan SSN: 787-78-9558

👤

Doctor: Sandy

🛏️

Bed: Bed2-1

📅

23/02/2024 22:49:01

📝

Diagnosis:

Diagnosis 7

📝

Treatment:

Treatment 10

📌

State: In Progress

⚠️

Priority: High

Nuevo

Reports
Report for Megan

1 / 1

Patient

Megan SSN: 787-78-9558

Doctor

Sandy

Bed

Bed2-1

Admission Date

23/02/2024 22:49:01

Discharge Date

State

In Progress

Completed

Priority

Low

Normal

High

DIAGNOSIS & TREATMENT

Diagnosis.xml

Diagnosis	Date	Description
Diagnosis 7	17/01/2024	Society surface else itself effect raise tough produce. Might interview first off worker daughter. Draw room wear eye town they.

Treatment

Treatment	Date	Description
Treatment 10	28/08/2023	Scientist space project surface development. Turn billion significant role occur. Feeling guy vote thousand book candidate.

MEDICAL REPORT

Admission Admission for Megan Reid

7. Proceso de desarrollo

- Metodología de desarrollo: Freestyle
- Herramientas utilizadas para el desarrollo:
 - Github
 - GitKraken
 - Pycharm
 - Docker Desktop
 - Slacker

8. Conclusiones

Resumen de los logros del proyecto:

- El proyecto ha logrado desarrollar un módulo integral para la gestión hospitalaria en la plataforma Odoo, que aborda de manera efectiva las necesidades de administración y atención médica en un entorno hospitalario.

Durante el desarrollo del proyecto, se han identificado varias lecciones importantes que pueden ser útiles para futuros proyectos similares:

- **La importancia de una planificación detallada:** La planificación adecuada de los requisitos del proyecto y la definición clara de los objetivos desde el principio son fundamentales para el éxito del proyecto.
- **La comunicación efectiva es clave:** Mantener una comunicación clara y constante entre todos los miembros del equipo de desarrollo, así como con los usuarios finales, es esencial para asegurar que el producto final cumpla con las expectativas y necesidades.
- **Flexibilidad para adaptarse a cambios:** Los requisitos del proyecto pueden cambiar a lo largo del tiempo, por lo que es importante mantenerse flexible y estar dispuesto a adaptarse a nuevas circunstancias y requerimientos.

A pesar de los logros alcanzados, existen áreas de mejora que podrían explorarse en futuras iteraciones del proyecto:

- **Ampliación de funcionalidades:** Se podrían agregar nuevas funcionalidades para cubrir aspectos específicos de la gestión hospitalaria que no fueron abordados en la versión inicial del módulo.