



UNIVERSIDAD PRIVADA DE SANTA CRUZ DE LA SIERRA  
FACULTAD DE INGENIERÍA

**Estudiante y Número de Registro:**

Jassiel Ovando Franco — 2021117600

**Materia:**

Base de Datos — SI 314

**Docente:**

Carlos Wilfredo Egüez Terrazas

**Proyecto:**

Grupo 2: Veterinaria

**Fecha:**

Viernes, 8 de diciembre, 2023

# Índice

<b>1. Introducción</b>	<b>4</b>
1.1. Enunciado para la Veterinaria . . . . .	4
1.1.1. Ampliación: Servicio de Hotel de Mascotas . . . . .	4
1.2. Consideraciones y Decisiones de Implementación . . . . .	5
<b>2. Modelos</b>	<b>6</b>
2.1. Modelo Conceptual . . . . .	6
2.2. Modelo Lógico . . . . .	7
2.3. Modelo Físico . . . . .	9
2.3.1. Dbeaver . . . . .	10
2.3.2. PGadmin4 . . . . .	13
<b>3. Arquitectura de la Solución Construida</b>	<b>17</b>
3.1. Herramientas Utilizadas . . . . .	17
<b>4. Implementación</b>	<b>18</b>
4.1. Base de Datos . . . . .	18
4.1.1. Código en SQL . . . . .	18
4.2. Aplicación . . . . .	22
4.2.1. Django — CRUD . . . . .	22
4.2.2. Cobranza . . . . .	22
4.2.3. Reporte de Huéspedes entre 2 Fechas . . . . .	25
<b>A. Django</b>	<b>27</b>
<b>B. cobranza.py</b>	<b>38</b>
<b>C. reporte-por-periodo.py</b>	<b>40</b>
<b>D. models.py</b>	<b>41</b>

## Índice de figuras

1.	Modelo conceptual hecho en plantuml. . . . .	6
2.	Modelo lógico hecho en plantuml. . . . .	7
3.	Modelo lógico proporcionado por el profesor, como base. . . . .	8
4.	Modelo físico hecho en plantuml. . . . .	9
5.	Primera parte: Modelo físico generado por DBever. . . . .	10
6.	Segunda parte: Modelo físico generado por DBever. . . . .	11
7.	Tercera parte: Modelo físico generado por DBever. . . . .	12
8.	Primera parte: Modelo físico generado por PGAdmin4. . . . .	13
9.	Segunda parte: Modelo físico generado por PGAdmin4. . . . .	14
10.	Tercera parte: Modelo físico generado por PGAdmin4. . . . .	15
11.	Cuarta parte: Modelo físico generado por PGAdmin4. . . . .	16
12.	Menú django . . . . .	27
13.	Agregar cliente en django. . . . .	28
14.	Agregar una estancia en el calendario de vacunas en django. . . . .	29
15.	Agregar un encargado en django. . . . .	30
16.	Agregar una estancia en el hotel en django. . . . .	31
17.	Ver las personas en django. . . . .	32
18.	Generación de código de cliente en django (al momento de crear uno). . . . .	32
19.	Asignación de encargado en base a cliente y persona. . . . .	33
20.	Generación de código de mascota en django (al momento de crear una). . . . .	34
21.	10 registros de peso de mascota en django. . . . .	35
22.	10 registros de peso de mascota en django. . . . .	36
23.	Agregar una estancia en el hotel en django. . . . .	37
24.	Agregar dos requerimientos a una estadía . . . . .	37
25.	Ejecución de cobranza.py mostrando lista principal. . . . .	38
26.	Ejecución de cobranza.py mostrando el recibo. . . . .	38
27.	Revisión en django de que se agrego la fecha. . . . .	39
28.	Ejecución de reporte-por-periodo.py mostrando la lista de huéspedes atendidos. . . . .	40

# 1. Introducción

## 1.1. Enunciado para la Veterinaria

Un veterinario tiene como pacientes: animales, y como clientes: familias. Un cliente es un conjunto de personas que suelen corresponderse con una familia.

Cada cliente tiene: código, apellido paterno (cabeza de la familia), número de cuenta bancaria, dirección, teléfono, y nombres e identificación de personas correspondientes. No existe el límite de personas asociadas a un cliente, además, una persona puede estar dado de alta en varios clientes. *Ejemplo: Un hombre que vive con su esposa tiene un gato y como tal pertenece a un cliente, pero también está dado de alto en el cliente asociado con el perro de sus padres.* Los clientes pueden tener varias mascotas, cada mascota tiene: código, alias, especie, raza, color de pelo, fecha de nacimiento, peso (mantener un historial sobre el peso de la mascota por las últimas 10 visitas, además del peso actual). Así mismo se guarda un historial médico con cada enfermedad que tuvo y la fecha en la que se enfermó. Adicionalmente, cada mascota tiene un calendario de vacunación, en la que se llena el registro de: cada vacuna, y la enfermedad contra la que se está vacunando.

### 1.1.1. Ampliación: Servicio de Hotel de Mascotas

La veterinaria está desarrollando un nuevo negocio de *hotel de mascotas*, para lo cual necesitan ampliar su sistema teniendo en cuenta el funcionamiento deseado de este nuevo servicio:

- Se puede hospedar a cualquier mascota sea o no paciente de la veterinaria, pero debe registrarse la información para todos.
- Para el huésped, debe registrarse si tiene necesidades especiales en cuanto a medicación, alimentación o algún cuidado particular.
- La alimentación en general se cobra como parte del servicio de hospedaje, excepto cuando requiere alimentación especial, que se factura de manera adicional, lo mismo que medicinas y otros insumos que sean necesarios para su cuidado particular.

### Requerimientos

1. El servicio incluye también la higiene básica de un baño al ser recibido, y en estancias prolongadas un baño semanal adicional, cualquier servicio más allá de eso se considera extra en la cuenta.
2. Se debe construir la funcionalidad necesaria para el registro de los huéspedes y todas sus necesidades. Además de construir la funcionalidad para la verificación (check out) del huésped y la emisión de la nota de cobranza.
3. Se debe elaborar un reporte de los huéspedes atendidos en un periodo de entre 2 fechas, incluyendo los huéspedes que están siendo atendidos.

4. Por último, elaborar todas las interfaces ABMC/CRUD necesarias para el manejo de los datos.

## 1.2. Consideraciones y Decisiones de Implementación

Dado que se tiene una base para la implementación de la veterinaria, con el código ejemplar del profesor, además de el diagrama base, se hará uso del mismo donde se agregarán más elementos, tablas atributos o relaciones. Entonces, se tendrá una base de datos (veterinaria) que almacene las siguientes tablas:

- Cliente
- Mascota
- Persona
- Encargado
- Historial Médico
- Historial de Peso
- Calendario de Vacunas (más como un historial)
- Vacunas (para saber qué vacunas existen)
- Estadía (para registrar los huéspedes, para la ampliación del hotel)
- Requerimientos (que pedidos tienen los huéspedes)
- Servicios (que servicios se ofrecen en el hotel)

Cada tabla será mostrada más adelante en SQL y diagrama ERD, hecho en PlantUML y el generado por DBeaver.

Para las interfaces, se hace uso de Django, que mediante modelos, nos permite modificar, crear, eliminar, y ver los datos de la base de datos, mediante *modelos*, que representan las tablas. Aparte, el historial de pesos de mantener un registro de los 10 historiales de peso más nuevos, los viejos son descartados. Para esto se tiene un código en python que manipula la tabla en cada guardado.

Finalmente para los 2 procedimientos, *el registro del huésped con generación de factura*, y *el reporte de huéspedes atendidos en un periodo de 2 fechas*, se hace uso de programas de consola en python3, que se conectan a la base de datos, y realizan las consultas necesarias, para luego mostrar los datos en pantalla.

## 2. Modelos

Dado que las imágenes puede que no se muestren en una buena calidad por la compresión de la imagen para que quede en el documento, el enlace al repositorio donde están las imágenes usadas es <https://gitlab.com/jassiel-uni/si-314/>, en el directorio `proyectos/final/doc/img`

### 2.1. Modelo Conceptual

Para el modelo conceptual no se trata de idear mucho en la parte de la ampliación, por lo tanto contiene diferencias contra los demás modelos. Mientras que la parte base de la veterinaria, mantiene una similitud con los modelos base y demás modelos.

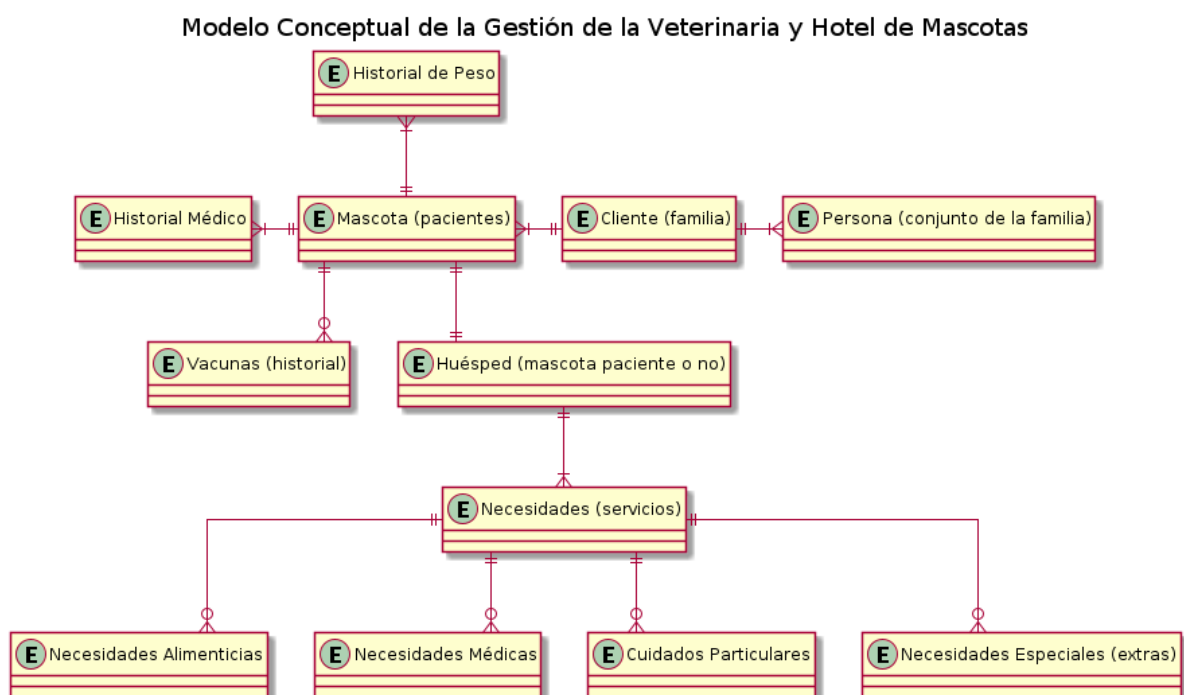


Figura 1: Modelo conceptual hecho en plantuml.

## 2.2. Modelo Lógico

En el modelo lógico ya se procede a realizar un mejor diseño para las tablas demás de considerar las las relaciones, las llaves foráneas, tipo de atributos (entero, cadena, fecha, etc), y demás.

Se consideran nuevas tablas por parte de la ampliación para el servicio de huéspedes y sus necesidades, las cuales deberán estar en sus requerimientos dependiendo de los servicios que se ofrecen.

La tabla principal, se demuestra como la mascota, y sub-principales como el cliente y la persona, que son los que tienen más atributos y relaciones con otras tablas.

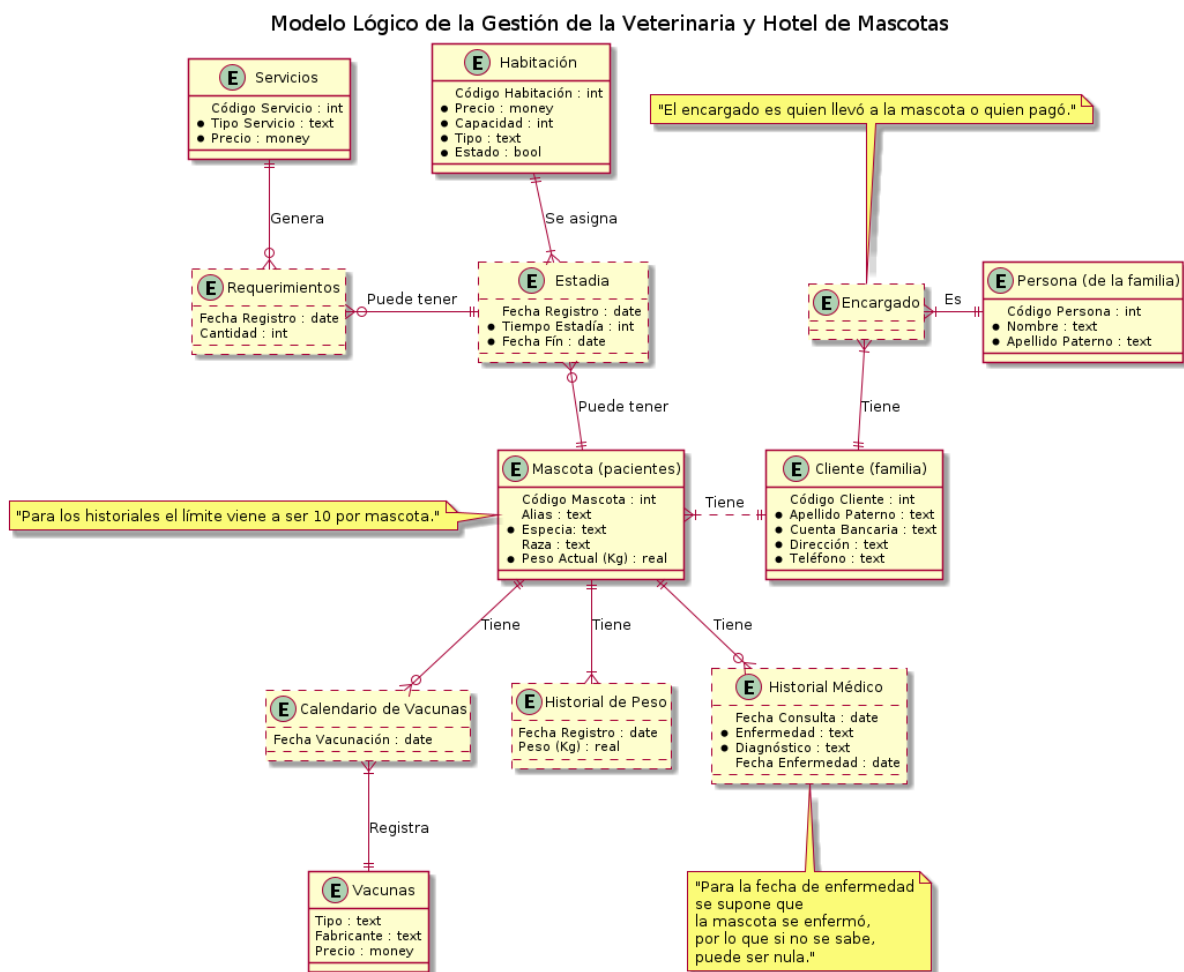


Figura 2: Modelo lógico hecho en plantuml.

El modelo base proporcionado por el profesor, se muestra a continuación, el cual sirve como ayuda para la creación de tablas y coherencia de un sistema mejor elaborado.

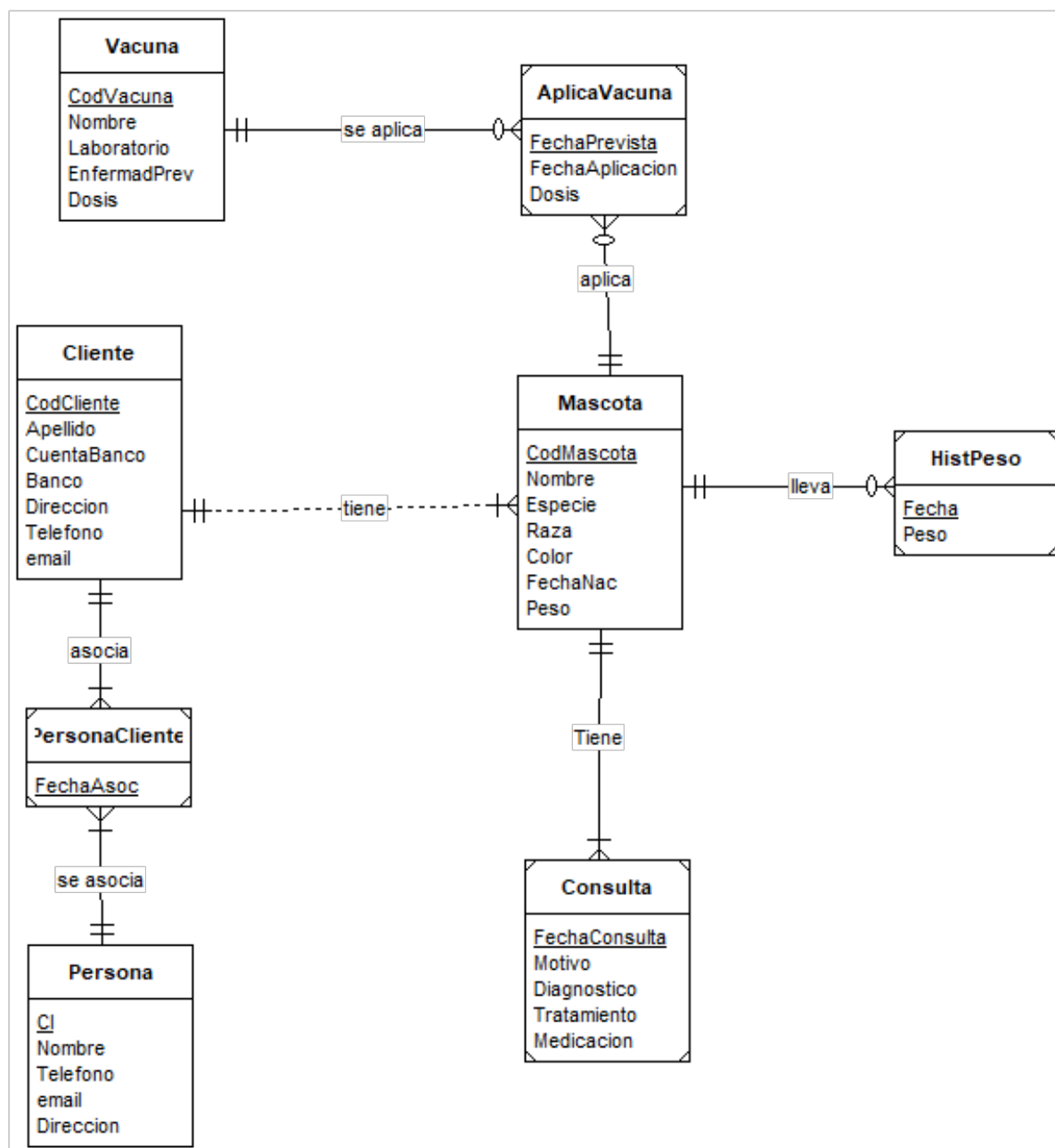


Figura 3: Modelo lógico proporcionado por el profesor, como base.



## 2.3. Modelo Físico

Una vez se hayan hecho todas las modificaciones finales, se procede a generar el modelo físico, que está en base a la base de datos ya hecha y funcional, la idea principal, prototipo se genera primeramente con PlantUML (sin haber hecho revisado la base de datos), pero luego con DBeaver y PGAdmin4, se genera el modelo físico final.

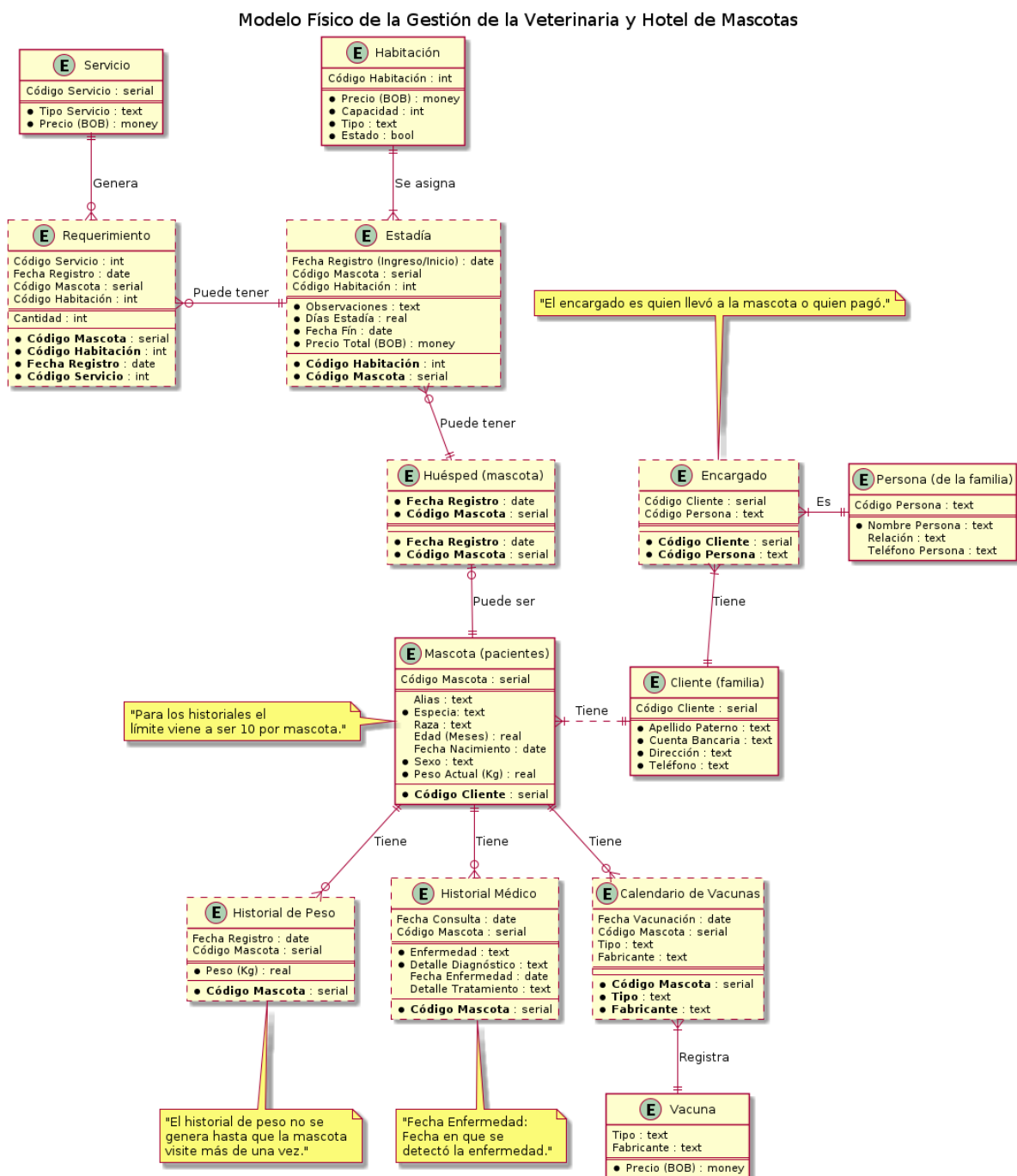


Figura 4: Modelo físico hecho en plantuml.

### 2.3.1. Dbeaver

Dbeaver proporciona un mejor modelo físico, una mejor generación y mejor detalle de las tablas, atributos, llaves primarias y foráneas, y demás.

Dado que se hace uso de django, se tienen tablas extras de autenticación con la base de datos y usuarios en la interfaz web de administración, las cuales no aparecen en ninguno de los otros modelos.

Debido a que el modelo físico es muy grande, se muestra en las siguientes páginas.

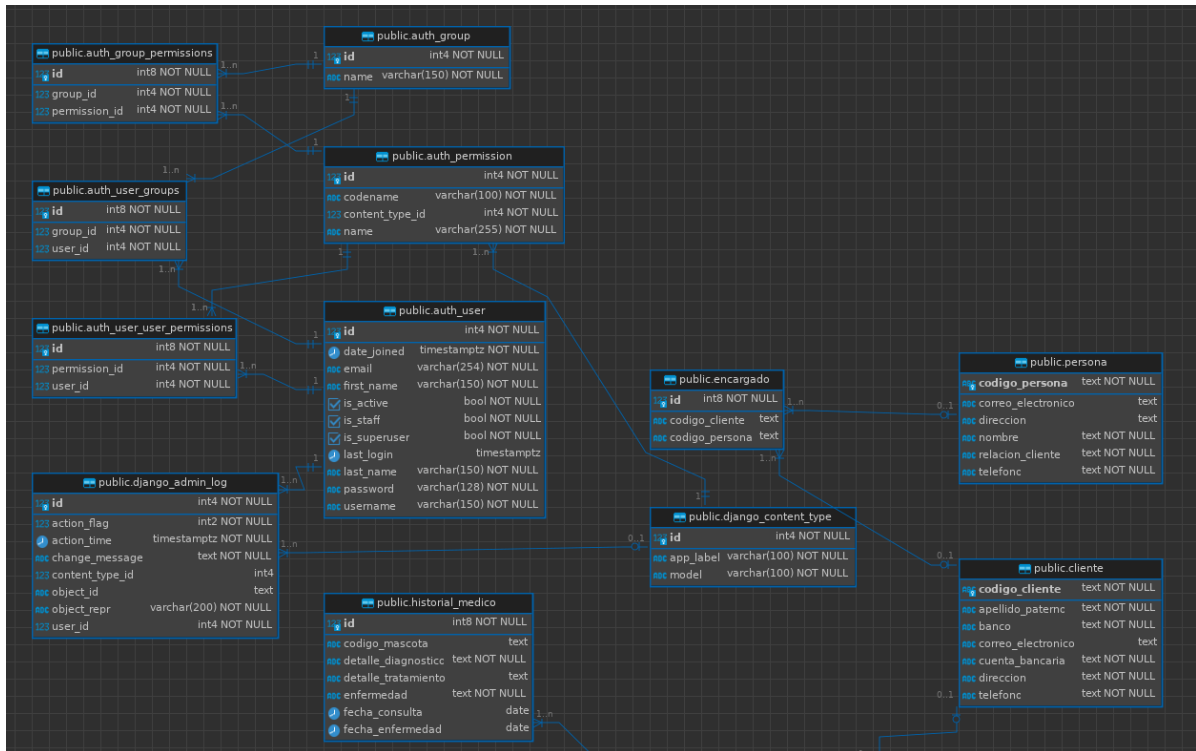


Figura 5: Primera parte: Modelo físico generado por DBeaver.

Aquí se aprecia más la parte de la autenticación generada por django, y algunas tablas de (cliente y persona) de la veterinaria.

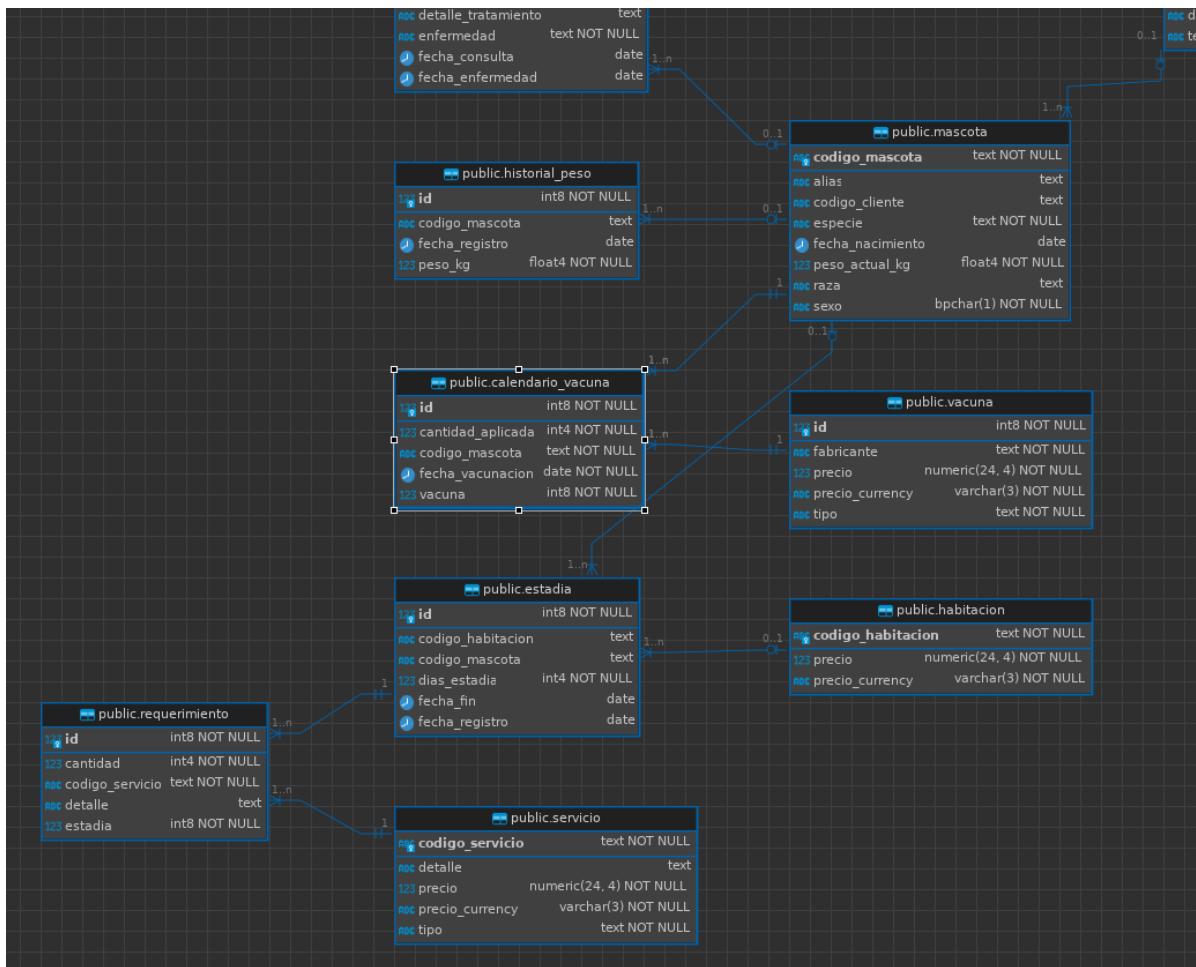


Figura 6: Segunda parte: Modelo físico generado por DBeaver.

Aquí ya se puede apreciar más las tablas principales de la veterinaria, así como la ampliación para el hotel.

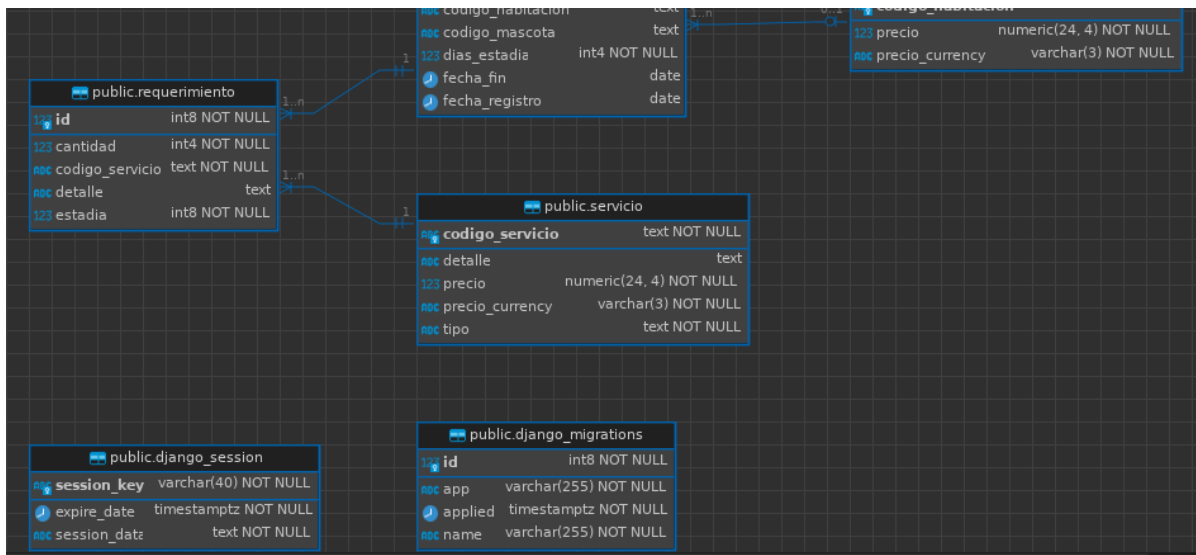


Figura 7: Tercera parte: Modelo físico generado por DBeaver.

Finalmente se aprecian las tablas de migración y sesión de django más los servicios y requerimientos de la ampliación.

### 2.3.2. PGadmin4

PGadmin4 al igual que DBeaver, proporciona un modelo ERD, con bastantes detalles, pero dado que su generación es muy incompleta, se hace uso de DBeaver para la generación del modelo físico, principalmente. De igual forma, se procede a mostrar el modelo generado por PGadmin4.

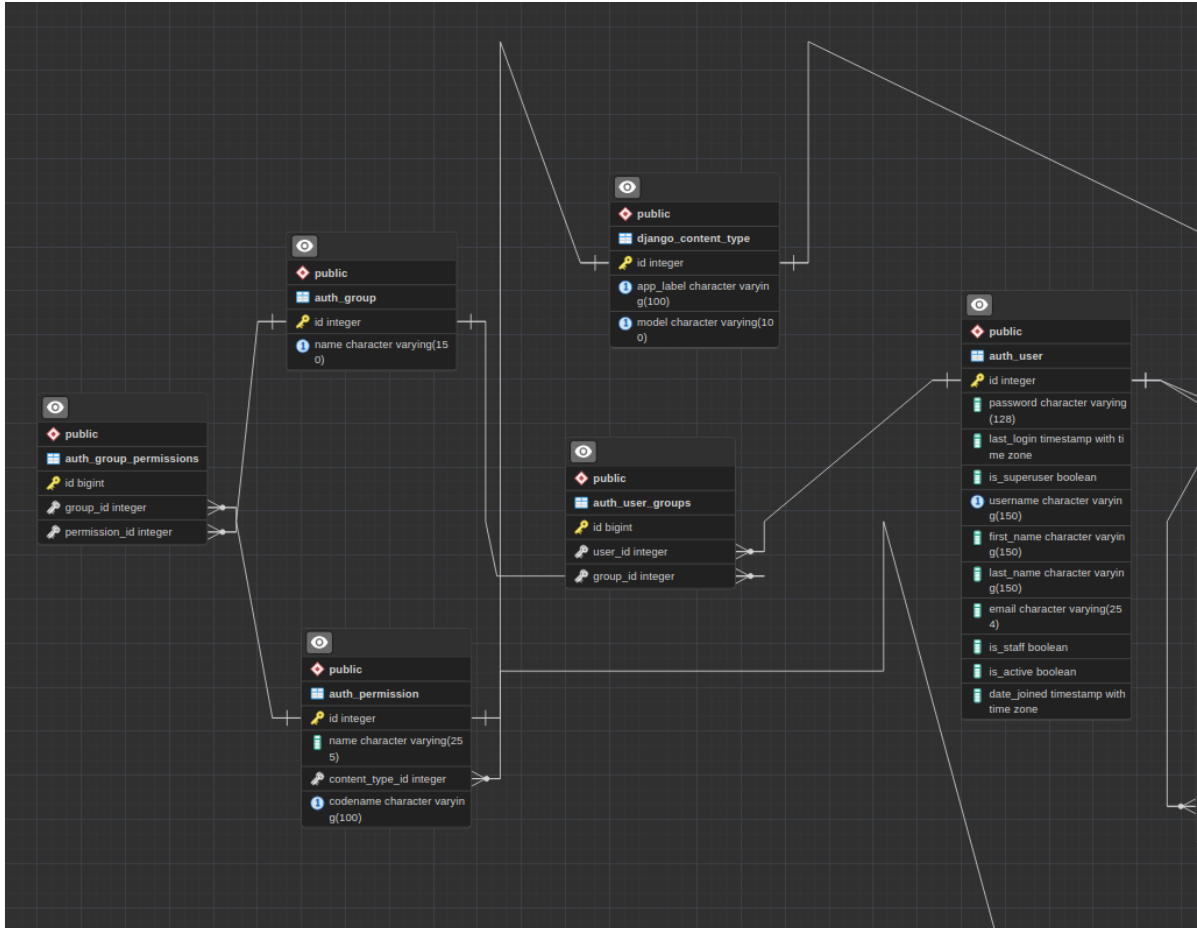


Figura 8: Primera parte: Modelo físico generado por PGadmin4.

Tablas de autenticación y conexión de django.

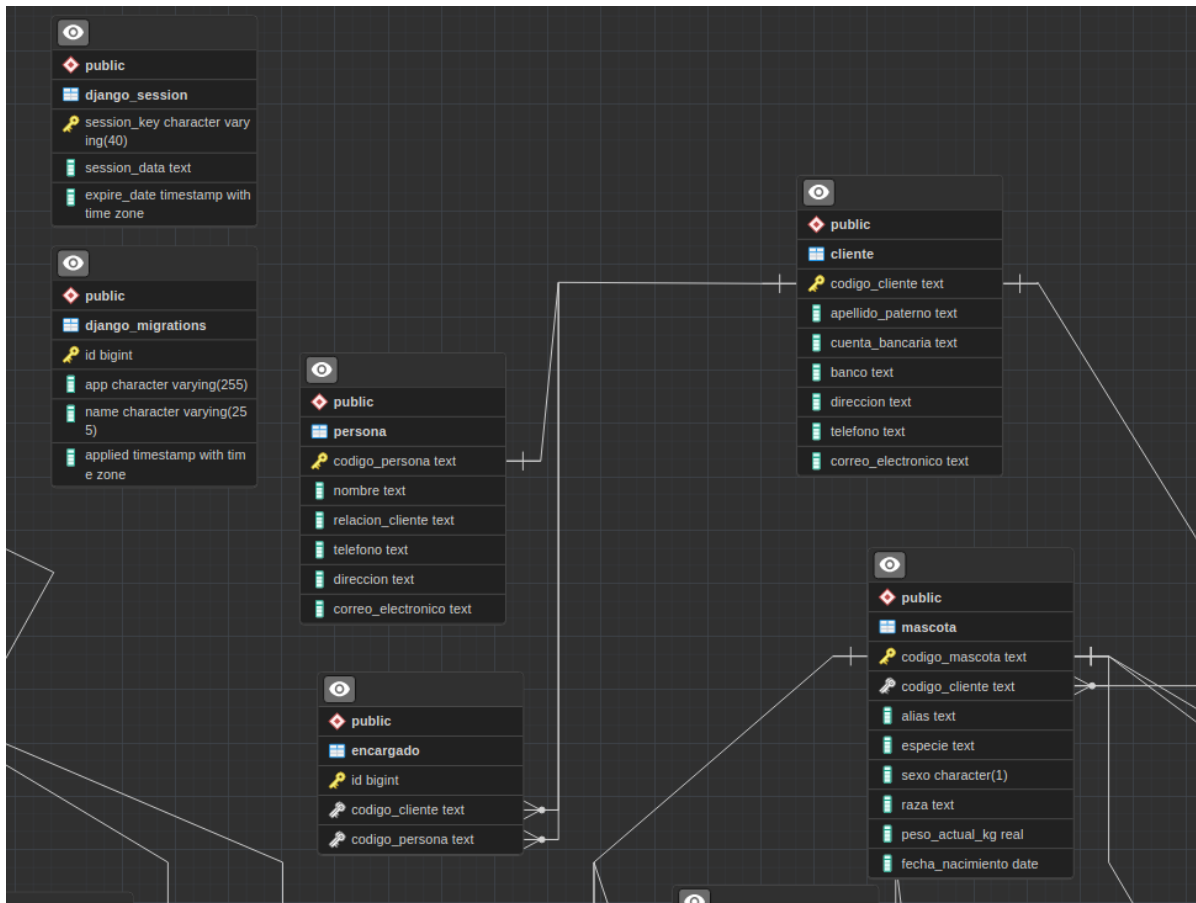


Figura 9: Segunda parte: Modelo físico generado por PGAdmin4.  
Tablas de migración y sesión, más tablas principales de la veterinaria.

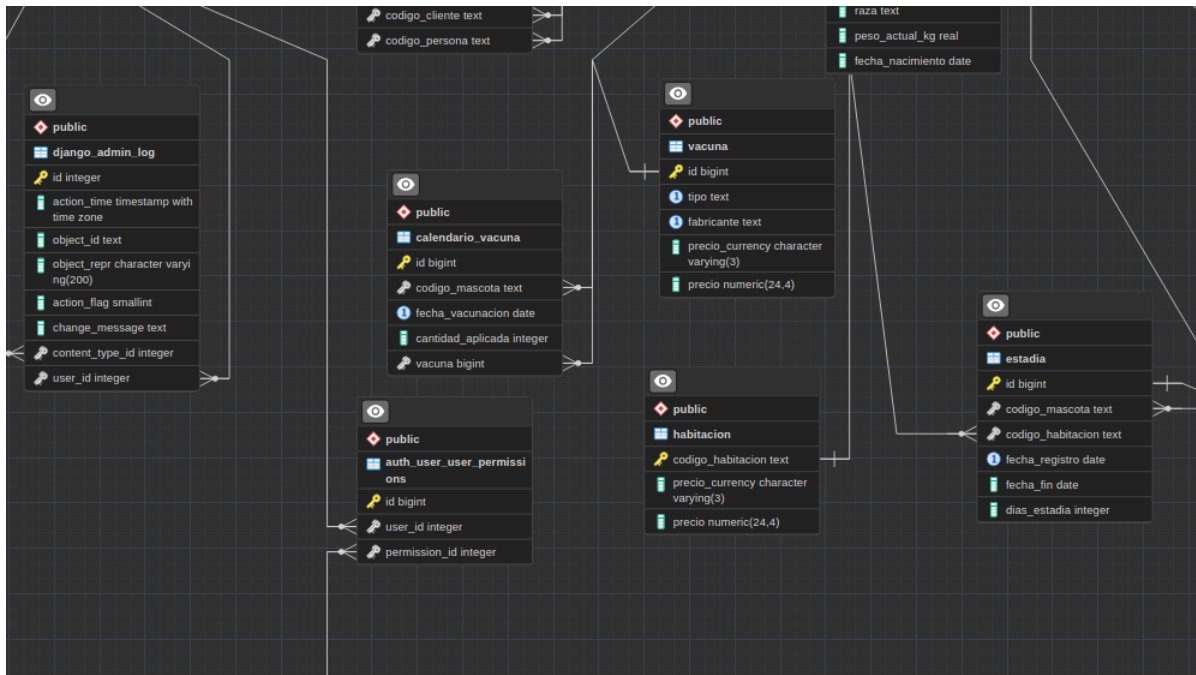


Figura 10: Tercera parte: Modelo físico generado por PGAdmin4.

Tablas de la ampliación del hotel.

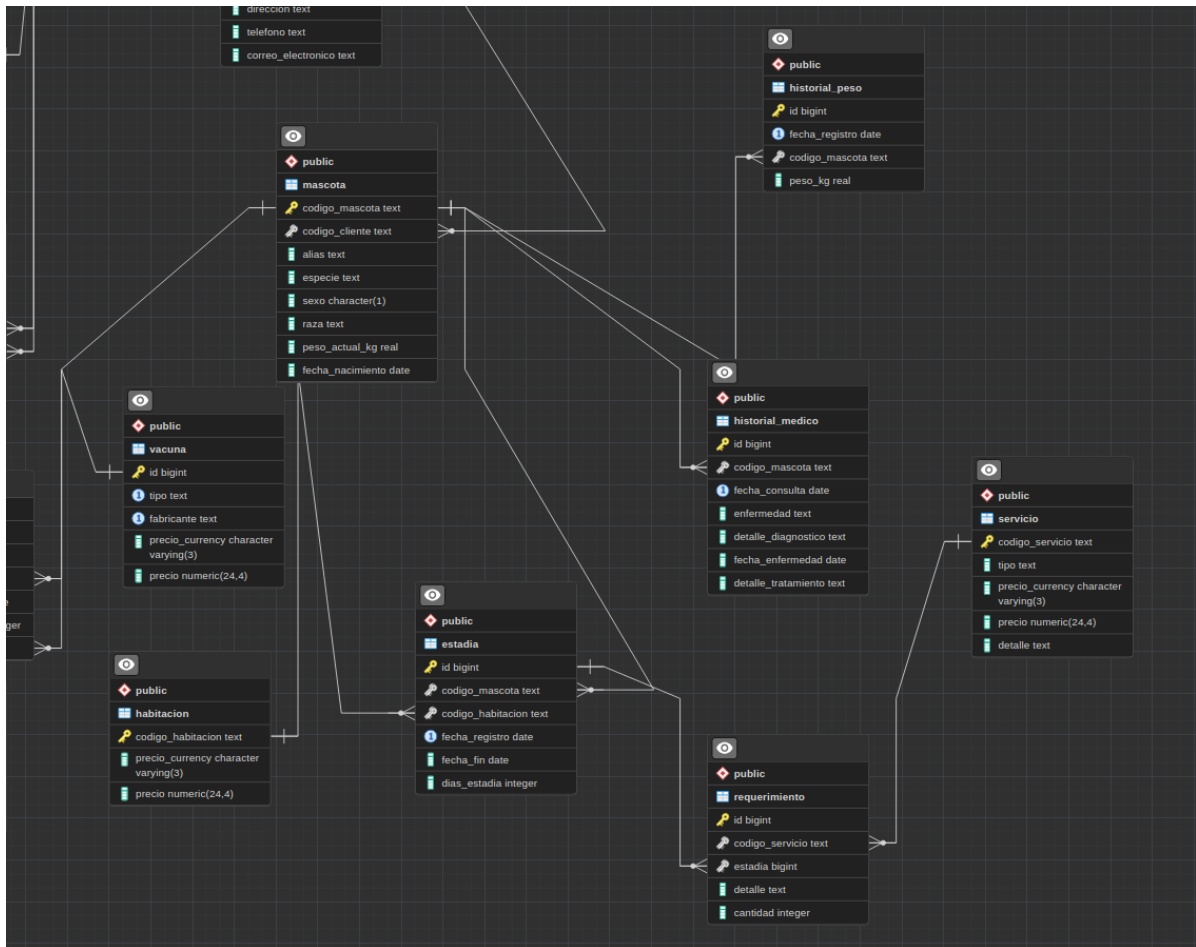


Figura 11: Cuarta parte: Modelo físico generado por PGAdmin4.

Ampliación e historiales bajo una vista más amplia.



### 3. Arquitectura de la Solución Construida

En cuanto al tipo de arquitectura que se usa, se considera una de 3 capas.

Primera capa: La base de datos, que es la que almacena los datos, y se encarga de la persistencia de los mismos.

Segunda capa: La aplicación, que es la que se encarga de la lógica de negocio, y la interacción con el usuario.

Tercera capa: La interfaz gráfica, que es la que se encarga de la presentación de los datos, y la interacción con el usuario.

Además de la interfaz gráfica, para las funciones extras mencionadas, como límite de historial de peso de 10 registros más nuevos, se hace uso de administración en `models.py` de django, y un código en python que se encarga de la manipulación de los datos.

Para la cobranza y *check-out* de los huéspedes, además del reporte de huéspedes atendidos en un periodo de 2 fechas, se hace uso de programas de consola de python3, que se encarga de conectarse a la base de datos, seleccionar, las condiciones, calcular los datos, cambiarlos, y finalmente mostrarlos en pantalla.

#### 3.1. Herramientas Utilizadas

Para la base de datos, se usa netamente PostgreSQL, mediante el gestor de bases de datos DBeaver (además de PGAdmin4 para ver **notices** y **logs**). Cual nos ayuda a generar el modelo físico de la base de datos (ERC), además de usar el editor Visual Studio Code, para los archivos de código en SQL. Los archivos SQL, se dividen principalmente en `ddl.sql`; donde se crean las tablas con sus atributos y llaves primarias o foráneas, `dml.sql`; donde se insertan los datos de prueba iniciales, ya que más adelante se hará uso de la interfaz gráfica para la inserción de datos, y `dql.sql`; donde se hacen las consultas a la base de datos. Aparte de esos archivos, existe el apartado **procedimientos** donde estarán los procedimientos almacenados de la base de datos.

Para la interfaz gráfica o aplicación, se hace uso de *django*, un framework de Python, que crea una aplicación web con autenticación de usuarios, más la interfaz de administración de la base de datos, configurada con postgresQL.

Además de eso, como se menciona en la parte de la arquitectura, se hace uso de 2 programas en python3 que se encargan de la cobranza y reporte de periodos.

## 4. Implementación

El repositorio se encuentra en <https://gitlab.com/jassiel-uni/si-314>, en el directorio `proyectos/final/doc/src/`, el usado en este documento, y el otro en `proyectos/final/veterinaria.sql`, para el sql sin corregir y comentarios.

### 4.1. Base de Datos

#### 4.1.1. Código en SQL

Hecho en postgresSQL, si bien hay una generalización con la mayoría de codigos en `text`, se usa para poder hacer un generador automático con ayuda de django, que usa un autoguardado, por cada objeto creado, siguiendo una sintaxis con los códigos de clientes, y mascotas, principalmente.

Fuera de los códigos de cliente y mascotas, se tienen por ejemplo los códigos de habitación (números) o de servicio, que están aceptando una cantidad de texto más grande, pero se lo corrige con la interfaz de administración de django, donde solo acepta un aproximado de 5 caracteres, y se hace uso de la interfaz gráfica para la inserción de datos.

```
1  -- DDL: Data Definition Language
2  -- Creación de la base de datos en PostgreSQL bajo 'template1'
3  CREATE DATABASE veterinaria
4  WITH OWNER = jassiel
5  ENCODING = 'UTF8'
6  LC_COLLATE = 'es_BO.UTF-8'
7  LC_CTYPE = 'es_BO.UTF-8'
8  TEMPLATE = template1
9  TABLESPACE = pg_default connection
10 LIMIT = - 1;
11
12 -- Tablas primarias:
13 CREATE TABLE cliente(
14     codigo_cliente text,
15     apellido_paterno text NOT NULL,
16     cuenta_bancaria text NOT NULL,
17     banco text NOT NULL,
18     direccion text NOT NULL,
19     telefono text NOT NULL,
20     correo_electronico text,
21     CONSTRAINT PK_cliente PRIMARY KEY (codigo_cliente)
22 );
23
24 CREATE TABLE persona(
25     codigo_persona text,
26     nombre text NOT NULL,
27     relacion_cliente text NOT NULL,
```

```

28 telefono text NOT NULL,
29 direccion text,
30 correo_electronico text,
31 CONSTRAINT PK_persona PRIMARY KEY (codigo_persona)
32 );
33
34 CREATE TABLE encargado(
35     -- Por fines de compatibilidad con django, se hace uso de la columna id,
36     -- ya que django no tiene un buen soporte de modelos-tablas sin llaves primarias,
37     -- o tablas con llaves primarias compuestas,
38     -- para las compuestas se hacen restricciones únicas.
39     id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
40     codigo_cliente text,
41     codigo_persona text,
42     CONSTRAINT PK_encargado PRIMARY KEY (id),
43     CONSTRAINT UQ_encargado UNIQUE (codigo_cliente, codigo_persona),
44     CONSTRAINT FK_encargado_cliente FOREIGN KEY (codigo_cliente) REFERENCES cliente(
45         codigo_cliente) ON DELETE CASCADE ON UPDATE CASCADE,
46     CONSTRAINT FK_encargado_persona FOREIGN KEY (codigo_persona) REFERENCES persona(
47         codigo_persona) ON DELETE CASCADE ON UPDATE CASCADE
48 );
49
50 CREATE TABLE mascota(
51     codigo_mascota text,
52     codigo_cliente text,
53     alias text DEFAULT 'sin nombre',
54     especie text NOT NULL,
55     sexo char(1) NOT NULL CHECK (sexo IN ('M', 'F', 'O')),
56     raza text DEFAULT 'desconocida',
57     peso_actual_kg real NOT NULL CHECK (peso_actual_kg > 0::real),
58     fecha_nacimiento date CHECK (fecha_nacimiento <= CURRENT_DATE),
59     CONSTRAINT PK_mascota PRIMARY KEY (codigo_mascota),
60     CONSTRAINT FK_mascota_cliente FOREIGN KEY (codigo_cliente) REFERENCES cliente(
61         codigo_cliente) ON DELETE CASCADE ON UPDATE CASCADE
62 );
63
64 CREATE TABLE historial_peso(
65     id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
66     fecha_registro date,
67     codigo_mascota text,
68     peso_kg real NOT NULL CHECK (peso_kg > 0::real),
69     CONSTRAINT PK_historial_peso PRIMARY KEY (id),
70     CONSTRAINT UQ_historial_peso UNIQUE (fecha_registro, codigo_mascota),
71     CONSTRAINT FK_historial_peso_mascota FOREIGN KEY (codigo_mascota) REFERENCES
72         mascota(codigo_mascota) ON DELETE CASCADE ON UPDATE CASCADE
73 );

```

```

70
71 CREATE TABLE historial_medico(
72     id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
73     codigo_mascota text,
74     fecha_consulta date,
75     enfermedad text NOT NULL DEFAULT 'sin enfermedad',
76     detalle_diagnostico text NOT NULL DEFAULT 'sin diagnostico',
77     fecha_enfermedad date CHECK (fecha_enfermedad <= fecha_consulta),
78     detalle_tratamiento text DEFAULT 'sin tratamiento',
79     CONSTRAINT PK_historial_medico PRIMARY KEY (id),
80     CONSTRAINT UQ_historial_medico UNIQUE (codigo_mascota, fecha_consulta),
81     CONSTRAINT FK_historial_medico_mascota FOREIGN KEY (codigo_mascota) REFERENCES
        mascota(codigo_mascota) ON DELETE CASCADE ON UPDATE CASCADE
82 );
83
84 CREATE TABLE vacuna(
85     id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
86     tipo text NOT NULL,
87     fabricante text NOT NULL,
88     -- Dado que se usa en models.py la clase MoneyField, se necesitan dos columnas para
        el precio.
89     precio_currency varchar(3) NOT NULL,
90     precio numeric(24, 4) NOT NULL CHECK (precio >= 0),
91     CONSTRAINT PK_vacuna PRIMARY KEY (id),
92     CONSTRAINT UQ_vacuna UNIQUE (tipo, fabricante)
93 );
94
95 CREATE TABLE calendario_vacuna(
96     id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
97     codigo_mascota text NOT NULL,
98     fecha_vacunacion date NOT NULL DEFAULT CURRENT_DATE,
99     cantidad_aplicada int NOT NULL DEFAULT 1 CHECK (cantidad_aplicada > 0),
100     vacuna bigint NOT NULL,
101     CONSTRAINT PK_calendario_vacunas PRIMARY KEY (id),
102     CONSTRAINT UQ_calendario_vacuna UNIQUE (codigo_mascota, fecha_vacunacion, vacuna),
103     CONSTRAINT FK_calendario_vacunas_mascota FOREIGN KEY (codigo_mascota) REFERENCES
        mascota(codigo_mascota) ON DELETE CASCADE ON UPDATE CASCADE,
104     CONSTRAINT FK_calendario_vacuna_vacuna FOREIGN KEY (vacuna) REFERENCES vacuna(id)
        ON DELETE CASCADE ON UPDATE CASCADE
105 );
106
107 CREATE TABLE habitacion(
108     codigo_habitacion text,
109     precio_currency varchar(3) NOT NULL,
110     precio numeric(24, 4) NOT NULL,
111     CONSTRAINT PK_habitacion PRIMARY KEY (codigo_habitacion)

```

```

112 );
113
114 CREATE TABLE estadia(
115     id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
116     codigo_mascota text,
117     codigo_habitacion text,
118     fecha_registro date DEFAULT CURRENT_DATE,
119     fecha_fin date DEFAULT NULL,
120     dias_estadia int,
121     CONSTRAINT PK_estadia PRIMARY KEY (id),
122     CONSTRAINT UQ_estadia UNIQUE (codigo_mascota, codigo_habitacion, fecha_registro),
123     CONSTRAINT FK_estadia_mascota FOREIGN KEY (codigo_mascota) REFERENCES mascota(
        codigo_mascota) ON DELETE CASCADE ON UPDATE CASCADE,
124     CONSTRAINT FK_estadia_habitacion FOREIGN KEY (codigo_habitacion) REFERENCES
        habitacion(codigo_habitacion) ON DELETE CASCADE ON UPDATE CASCADE
125 );
126
127 CREATE TABLE servicio(
128     codigo_servicio text,
129     tipo text NOT NULL CHECK (tipo IN ('alimentación', 'aseo', 'médico', 'otros', '
        extras')),
130     precio_currency varchar(3) NOT NULL,
131     precio numeric(24, 4) NOT NULL CHECK (precio >= 0),
132     detalle text DEFAULT 'sin detalle',
133     CONSTRAINT PK_servicio PRIMARY KEY (codigo_servicio)
134 );
135
136 CREATE TABLE requerimiento(
137     id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
138     codigo_servicio text NOT NULL,
139     estadia bigint NOT NULL,
140     detalle text DEFAULT 'sin detalle',
141     cantidad int NOT NULL DEFAULT 1 CHECK (cantidad > 0),
142     CONSTRAINT PK_requerimiento PRIMARY KEY (id),
143     CONSTRAINT FK_requerimiento_servicio FOREIGN KEY (codigo_servicio) REFERENCES
        servicio(codigo_servicio) ON DELETE CASCADE ON UPDATE CASCADE,
144     CONSTRAINT FK_requerimiento_estadia FOREIGN KEY (estadia) REFERENCES estadia(id) ON
        DELETE CASCADE ON UPDATE CASCADE
145 );

```

## 4.2. Aplicación

### 4.2.1. Django — CRUD

### 4.2.2. Cobranza

Listing 1: Cobranza de huéspedes hecho en Python3.

```
1 import psycopg2
2 from tabulate import tabulate
3 from datetime import datetime
4
5 # Parámetros de la conexión a la base de datos.
6 db_params = {
7     'user': 'jassiel',
8     'password': '08112002',
9     'database': 'final_veterinaria',
10    'host': 'localhost'
11 }
12
13 def connect_to_database():
14     try:
15         connection = psycopg2.connect(**db_params)
16         return connection
17     except Exception as e:
18         print(f"Error conectándose a la base de datos: {e}")
19         return None
20
21 def fetch_pending_estadias(connection):
22     try:
23         with connection.cursor() as cursor:
24             query = """
25                 SELECT e.id, e.fecha_registro, m.codigo_mascota
26                 FROM estadia e
27                 JOIN mascota m ON e.codigo_mascota = m.codigo_mascota
28                 WHERE e.fecha_fin IS NULL
29             """
30             cursor.execute(query)
31             result = cursor.fetchall()
32             return result
33     except Exception as e:
34         print(f"Error buscando estadias pendientes: {e}")
35         return None
36
37 def display_estadias(estadias):
38     headers = ["Estadía ID", "Fecha Registro", "Mascota Código"]
39     print(tabulate(estadias, headers=headers, tablefmt="fancy_grid"))
40
```

```

41 def fetch_services_for_estadia(connection, estadia_id):
42     try:
43         with connection.cursor() as cursor:
44             query = """
45                 SELECT r.id, s.codigo_servicio, s.tipo, s.precio, r.cantidad
46                 FROM requerimiento r
47                 JOIN servicio s ON r.codigo_servicio = s.codigo_servicio
48                 WHERE r.estadia = %s
49             """
50             cursor.execute(query, (estadia_id,))
51             result = cursor.fetchall()
52             return result
53     except Exception as e:
54         print(f"Error al buscar servicio para la estadia '{estadia_id}': {e}")
55         return None
56
57 def calculate_total_price(services, habitacion_price, estadia_days):
58     total_price = 0
59
60     # Revisar si el precio de la habitación no es nulo
61     if habitacion_price is not None:
62         total_price += habitacion_price * estadia_days if estadia_days is not None
63         else 0
64
65     for service in services:
66         # Revisar si el precio y la cantidad no son nulos.
67         price = service[3]
68         quantity = service[4]
69         if price is not None and quantity is not None:
70             total_price += price * quantity # Price * Quantity
71
72     return total_price
73
74 def update_estadia_fin_date(connection, estadia_id):
75     try:
76         with connection.cursor() as cursor:
77             query = "UPDATE estadia SET fecha_fin = CURRENT_DATE WHERE id = %s"
78             print("Consulta:", cursor.mogrify(query, (estadia_id,)))
79
80             cursor.execute(query, (estadia_id,))
81             connection.commit()
82     except Exception as e:
83         print(f"Error al actualizar la fecha de salida de estadia '{estadia_id}': {e}")
84         ")

```

```

85 def main():
86     connection = connect_to_database()
87
88     if connection:
89         estadias = fetch_pending_estadias(connection)
90
91         if estadias:
92             display_estadias(estadias)
93
94             estadia_id_input = input("Ingrese la ID de la estadia a procesar: ")
95
96             try:
97                 estadia_id = int(estadia_id_input)
98             except ValueError:
99                 print("Opción inválida, ingrese una ID válida.")
100                 return
101
102             services = fetch_services_for_estadia(connection, estadia_id)
103
104             if services:
105                 habitacion_price_query = "SELECT precio FROM habitacion WHERE
                                     codigo_habitacion = (SELECT codigo_habitacion FROM estadia WHERE
                                     id = %s)"
106
107                 with connection.cursor() as cursor:
108                     cursor.execute(habitacion_price_query, (estadia_id,))
109                     habitacion_price = cursor.fetchone()[0]
110
111                 estadia_days_query = "SELECT dias_estadia FROM estadia WHERE id = %s"
112                 with connection.cursor() as cursor:
113                     cursor.execute(estadia_days_query, (estadia_id,))
114                     estadia_days = cursor.fetchone()[0]
115
116                 total_price = calculate_total_price(services, habitacion_price,
117                                                         estadia_days)
118
119                 update_estadia_fin_date(connection, estadia_id)
120
121                 print("\nRecibo:")
122                 print("Información de la mascota:")
123
124                 print("Servicio (basada en los requerimientos):")
125                 headers = ["Servicio", "Tipo", "Cobranza"]
126                 services_data = [(s[1], s[2], s[3] * s[4]) for s in services]
127                 print(tabulate(services_data, headers=headers, tablefmt="fancy_grid")
128                             )
129                 print("Habitación:")

```



```

127         habitacion_data = [("Dias Estadía", "Cobranza")]
128         habitacion_data.append((estadia_days, habitacion_price))
129         print(tabulate(habitacion_data, tablefmt="fancy_grid"))
130         print(f"Total: {total_price:.2f}")
131
132     else:
133         print("Sin servicio en la estadía.")
134         print("Actualizando la fecha de salida...")
135         update_estadia_fin_date(connection, estadia_id)
136
137     else:
138         print("Sin estadías pendientes.")
139
140     connection.close()
141
142 if __name__ == "__main__":
143     main()

```

#### 4.2.3. Reporte de Huéspedes entre 2 Fechas

Listing 2: Reporte de huéspedes entre 2 fechas hecho en Python3. breaklines

```

1 import psycpg2
2 from tabulate import tabulate
3
4 fecha_a = input("Fecha A de la forma 2023-12-30: ")
5 fecha_b = input("Fecha B de la forma 2023-12-30: ")
6
7 conn = psycpg2.connect(
8     database="final_veterinaria",
9     user="jassiel",
10    password="08112002",
11    host="localhost",
12    port="5432"
13 )
14
15 cursor = conn.cursor()
16
17 query = f"""
18 WITH EstadiaStatus AS (
19     SELECT
20         id,
21         codigo_mascota,
22         codigo_habitacion,
23         fecha_registro,
24         fecha_fin--,

```

```

25     FROM estadia
26     WHERE
27         (fecha_fin IS NULL AND fecha_registro <= '{fecha_b}')
28         OR (fecha_fin IS NOT NULL AND fecha_registro <= '{fecha_b}' AND fecha_fin >= '{
           fecha_a}')
29 )
30 SELECT
31     id,
32     codigo_mascota,
33     codigo_habitacion,
34     fecha_registro,
35     fecha_fin,
36     CASE
37         WHEN fecha_fin IS NULL THEN 'Huésped aun en el hotel'
38         ELSE 'Huésped salió el ' || fecha_fin::text
39     END AS estado,
40     COUNT(*) FILTER (WHERE fecha_fin IS NULL) OVER () AS cantidad_en_hotel,
41     COUNT(*) FILTER (WHERE fecha_fin IS NOT NULL) OVER () AS cantidad_atendida
42 FROM EstadiaStatus;
43 """
44
45 cursor.execute(query)
46
47 result = cursor.fetchall()
48 headers = [desc[0] for desc in cursor.description]
49 print(tabulate(result, headers=headers, tablefmt='psql'))
50
51 cursor.close()
52 conn.close()

```

## A. Django

Debido a que son 12 tablas, poner, todas las interfaces que existen, resulta en un documento muy largo, por lo tanto, se pondrán interfaces de cada tabla en diferentes etapas.

### Site administration

ADMINISTRACION_VETERINARIA		
Calendario vacunas	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Cientes	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Encargados	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Estadias	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Habitacions	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Historial medicos	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Historial pesos	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Mascotas	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Personas	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Requerimientos	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Servicios	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Vacunas	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
AUTHENTICATION AND AUTHORIZATION		
Groups	<a href="#">+ Add</a>	<a href="#">✎ Change</a>
Users	<a href="#">+ Add</a>	<a href="#">✎ Change</a>

Figura 12: Menú de administración de django para agregar, eliminar, ver, actualizar datos.

## Add cliente


<b>Apellido paterno:</b>	<input type="text"/>
<b>Número de cuenta bancaria:</b>	<input type="text"/>
<b>Banco:</b>	<input type="text" value="-----"/> 
<b>Direccion:</b>	<input type="text"/>
<b>Telefono:</b>	<input type="text"/>
<b>Correo electronico:</b>	<input type="text"/>

Figura 13: Agregar cliente en django.

## Add calendario vacuna













<b>Codigo mascota:</b>	<input type="text" value="-----"/>    
<b>Fecha vacunacion:</b>	<input type="text" value="2023-11-29"/> Today    <small>Note: You are 4 hours behind server time.</small>
<b>Cantidad aplicada:</b>	<input type="text" value="1"/>  
<b>Vacuna:</b>	<input type="text" value="-----"/>    

Figura 14: Agregar una estancia en el calendario de vacunas en django.

## Add encargado

**Codigo cliente:**

**Codigo persona:**

SAVE

Save and add another

Save and continue editing

Figura 15: Agregar un encargado en django.

## Add estadia











<b>Codigo mascota:</b>	<input type="text" value="-----"/>    
<b>Codigo habitacion:</b>	<input type="text" value="-----"/>    
<b>Fecha registro:</b>	<input type="text" value="2023-11-29"/> Today    <small>Note: You are 4 hours behind server time.</small>
<b>Fecha fin:</b>	<input type="text"/> Today    <small>Note: You are 4 hours behind server time.</small>

Figura 16: Agregar una estancia en el hotel en django.

Action:	-----	Go	0 of 20 selected
<input type="checkbox"/>	PERSONA		
<input type="checkbox"/>	Persona: Alejandro Rojas - BP020, Relación: Proveedor, +5911111111, Avenida 20, Villamontes, alejandro.rojas@email.com		
<input type="checkbox"/>	Persona: Isabel Castro - BP019, Relación: Cliente, +5912222222, Calle 19, Riberalta, isabel.castro@email.com		
<input type="checkbox"/>	Persona: Ricardo Sandoval - BP018, Relación: Cliente, +5916666666, Calle 18, Puerto Suarez, ricardo.sandoval@email.com		
<input type="checkbox"/>	Persona: Natalia Arce - BP017, Relación: Proveedor, +5918888888, Avenida 17, Camiri, natalia.arce@email.com		
<input type="checkbox"/>	Persona: Oscar Velasco - BP016, Relación: Cliente, +5915555555, Calle 16, Villa Montes, oscar.velasco@email.com		
<input type="checkbox"/>	Persona: Eva Morales - BP015, Relación: Proveedor, +5913333333, Avenida 15, Yacuiba, eva.morales@email.com		
<input type="checkbox"/>	Persona: Diego Fernandez - BP014, Relación: Cliente, +5914444444, Calle 14, Warnes, diego.fernandez@email.com		
<input type="checkbox"/>	Persona: Silvia Rodríguez - BP013, Relación: Proveedor, +5919999999, Avenida 13, Riberalta, silvia.rodriguez@email.com		
<input type="checkbox"/>	Persona: Hugo Medina - BP012, Relación: Cliente, +5917777777, Calle 12, El Alto, hugo.medina@email.com		
<input type="checkbox"/>	Persona: Luisa Soto - BP011, Relación: Cliente, +5911111111, Calle 11, Quillacollo, luisa.soto@email.com		
<input type="checkbox"/>	Persona: Gabriela Torres - BP010, Relación: Proveedor, +5912222222, Avenida 10, Montero, gabriela.torres@email.com		
<input type="checkbox"/>	Persona: Pedro Romero - BP009, Relación: Cliente, +5916666666, Calle 9, Cobija, pedro.romero@email.com		
<input type="checkbox"/>	Persona: Marta Vargas - BP008, Relación: Cliente, +5914444444, Calle 8, Trinidad, marta.vargas@email.com		
<input type="checkbox"/>	Persona: Raul Chavez - BP007, Relación: Proveedor, +5918888888, Avenida 7, Oruro, raul.chavez@email.com		
<input type="checkbox"/>	Persona: Laura Fernandez - BP006, Relación: Cliente, +5917777777, Calle 6, Tarija, laura.fernandez@email.com		
<input type="checkbox"/>	Persona: Jorge Mendoza - BP005, Relación: Proveedor, +5919999999, Avenida 5, Potosi, jorge.mendoza@email.com		
<input type="checkbox"/>	Persona: Ana Gutierrez - BP004, Relación: Cliente, +5913333333, Calle 4, Sucre, ana.gutierrez@email.com		
<input type="checkbox"/>	Persona: Carlos Rodríguez - BP003, Relación: Cliente, +5915555555, Calle 3, Santa Cruz, carlos.rodriguez@email.com		
<input type="checkbox"/>	Persona: María Lopez - BP002, Relación: Proveedor, +59198765432, Avenida 2, Cochabamba, maria.lopez@email.com		
<input type="checkbox"/>	Persona: Juan Perez - BP001, Relación: Cliente, +59112345678, Calle 1, La Paz, juan.perez@email.com		
20 personas			

Figura 17: Ver las personas en django.

Action:	-----	Go	0 of 1 selected
<input type="checkbox"/>	CLIENTE		
<input type="checkbox"/>	Cliente morales - CLI-1, Teléfono: +59175098424, Correo-E: None		
1 cliente			

Figura 18: Generación de código de cliente en django (al momento de crear uno).



Figura 19: Asignación de encargado en base a cliente y persona.

Add encargado

**Codigo cliente:** Cliente morales - CLI-1, Teléfono: +59175098424, Correo-E: None    

**Codigo persona:** Persona: Raul Chavez - BP007, Relación: Proveedor, +59188888888, Avenida 7, Oruro, raul.chavez@email.com    

-----

Persona: Juan Perez - BP001, Relación: Cliente, +59112345678, Calle 1, La Paz, juan.perez@email.com

Persona: Maria Lopez - BP002, Relación: Proveedor, +59198765432, Avenida 2, Cochabamba, maria.lopez@email.com

Persona: Carlos Rodriguez - BP003, Relación: Cliente, +59155555555, Calle 3, Santa Cruz, carlos.rodriguez@email.com

Persona: Ana Gutierrez - BP004, Relación: Cliente, +59133333333, Calle 4, Sucre, ana.gutierrez@email.com

Persona: Jorge Mendoza - BP005, Relación: Proveedor, +59199999999, Avenida 5, Potosi, jorge.mendoza@email.com

Persona: Laura Fernandez - BP006, Relación: Cliente, +59177777777, Calle 6, Tarija, laura.fernandez@email.com

Persona: Raul Chavez - BP007, Relación: Proveedor, +59188888888, Avenida 7, Oruro, raul.chavez@email.com

Persona: Marta Vargas - BP008, Relación: Cliente, +59144444444, Calle 8, Trinidad, marta.vargas@email.com

Persona: Pedro Romero - BP009, Relación: Cliente, +59166666666, Calle 9, Cobija, pedro.romero@email.com

Persona: Gabriela Torres - BP010, Relación: Proveedor, +59122222222, Avenida 10, Montero, gabriela.torres@email.com

Persona: Luisa Soto - BP011, Relación: Cliente, +59111111111, Calle 11, Quillacollo, luisa.soto@email.com

Persona: Hugo Medina - BP012, Relación: Cliente, +59177777777, Calle 12, El Alto, hugo.medina@email.com

Persona: Silvia Rodriguez - BP013, Relación: Proveedor, +59199999999, Avenida 13, Riberalta, silvia.rodriguez@email.com

Persona: Diego Fernandez - BP014, Relación: Cliente, +59144444444, Calle 14, Warnes, diego.fernandez@email.com

Persona: Eva Morales - BP015, Relación: Proveedor, +59133333333, Avenida 15, Yacuiba, eva.morales@email.com

Persona: Oscar Velasco - BP016, Relación: Cliente, +59155555555, Calle 16, Villa Montes, oscar.velasco@email.com

Persona: Natalia Arce - BP017, Relación: Proveedor, +59188888888, Avenida 17, Camiri, natalia.arce@email.com

Persona: Ricardo Sandoval - BP018, Relación: Cliente, +59166666666, Calle 18, Puerto Suarez, ricardo.sandoval@email.com





**SAVE** **Save and add**

Change mascota

**Mascota: CLI-1-M1 - 'None', de morales**

Codigo cliente:

Cliente morales - CLI-1, Teléfono: +59175098424, Correo-E: None

Alias:

Especie:

canino

Sexo:

Macho

Raza:


Peso en kilogramos:

12.0

Fecha nacimiento:

2023-02-07

Today



Note: You are 4 hours behind server time.

SAVE

Save and add another

Save and continue editing

Figura 20: Generación de código de mascota en django (al momento de crear una).

Action:   0 of 10 selected

<input type="checkbox"/>	HISTORIAL PESO
<input type="checkbox"/>	Fecha: 2023-10-09 - Código Mascota: (CLI-1-M1) - Peso: 500.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-14 - Código Mascota: (CLI-1-M1) - Peso: 11.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-15 - Código Mascota: (CLI-1-M1) - Peso: 8.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-02 - Código Mascota: (CLI-1-M1) - Peso: 7.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-29 - Código Mascota: (CLI-1-M1) - Peso: 6.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-01 - Código Mascota: (CLI-1-M1) - Peso: 5.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-25 - Código Mascota: (CLI-1-M1) - Peso: 4.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-26 - Código Mascota: (CLI-1-M1) - Peso: 3.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-27 - Código Mascota: (CLI-1-M1) - Peso: 2.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-28 - Código Mascota: (CLI-1-M1) - Peso: 1.0 Kg

10 historial pesos

---

Figura 21: 10 registros de peso de mascota en django.  
Comprobando que se mantiene un historial de 10 registros de peso de mascota. Si se agrega un nuevo registro, el más viejo es eliminado.

## Select historial peso to change

Action:   0 of 10 selected











<input type="checkbox"/>	HISTORIAL PESO
<input type="checkbox"/>	Fecha: 2023-11-30 - Código Mascota: (CLI-1-M1) - Peso: 1000.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-14 - Código Mascota: (CLI-1-M1) - Peso: 11.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-15 - Código Mascota: (CLI-1-M1) - Peso: 8.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-02 - Código Mascota: (CLI-1-M1) - Peso: 7.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-29 - Código Mascota: (CLI-1-M1) - Peso: 6.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-01 - Código Mascota: (CLI-1-M1) - Peso: 5.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-25 - Código Mascota: (CLI-1-M1) - Peso: 4.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-26 - Código Mascota: (CLI-1-M1) - Peso: 3.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-27 - Código Mascota: (CLI-1-M1) - Peso: 2.0 Kg
<input type="checkbox"/>	Fecha: 2023-11-28 - Código Mascota: (CLI-1-M1) - Peso: 1.0 Kg

10 historial pesos

Figura 22: 10 registros de peso de mascota en django.

Comprobando que se mantiene un historial de 10 registros de peso de mascota. Ahora el más viejo de 500 Kg. se elimino. Esta funcionalidad se limita a la interfaz de django, si se hace por SQL, no se elimina el registro. El código será mostrada en la sección de `models.py`.

## Add estadia

<b>Codigo mascota:</b>	Mascota: CLI-1-M1 - 'None', de morales ▾	   
<b>Codigo habitacion:</b>	HAB001: BOB150.00 ▾	   
<b>Fecha registro:</b>	2023-11-29	Today   
Note: You are 4 hours behind server time.		
<b>Fecha fin:</b>		Today   
Note: You are 4 hours behind server time.		

SAVE

Save and add another

Save and continue editing

Figura 23: Agregar una estancia en el hotel en django.

## Select requerimiento to change

Action:  ▾  0 of 2 selected

<input type="checkbox"/>	REQUERIMIENTO
<input type="checkbox"/>	(Mascota: CLI-1-M1 - 'None', de morales, HAB004: BOB180.00, SAS002 - aseo): 3
<input type="checkbox"/>	(Mascota: CLI-1-M1 - 'None', de morales, HAB004: BOB180.00, SAL001 - alimentación): 3

2 requerimientos

Figura 24: Agregar dos requerimientos a una estadía

## B. cobranza.py

```
(app) jassiel@IdeaPad-flex-5 ~/G/g/j/s/p/final (main)> python cobranza.py
```

Estadia ID	Fecha Registro	MascotaCodigo
8	2023-11-29	CLI-1-M1

```
Ingrese la ID de la estadia a procesar: _
```

Figura 25: Ejecución de cobranza.py mostrando lista principal.

```
(app) jassiel@IdeaPad-flex-5 ~/G/g/j/s/p/final (main)> python cobranza.py
```

Estadia ID	Fecha Registro	MascotaCodigo
8	2023-11-29	CLI-1-M1

```
Ingrese la ID de la estadia a procesar: 8
Consulta: b'UPDATE estadia SET fecha_fin = CURRENT_DATE WHERE id = 8'

Recibo:
Información de la mascota:
Servicio (basada en los requerimientos):
```

Servicio	Tipo	Cobranza
SAL001	alimentación	61.5
SAS002	aseo	107.25

```
Habitación:
```

Dias Estadia	Cobranza
	180.0000

```
Total: 168.75
(app) jassiel@IdeaPad-flex-5 ~/G/g/j/s/p/final (main)> python cobranza.py
Sin estadias pendientes.
(app) jassiel@IdeaPad-flex-5 ~/G/g/j/s/p/final (main)> _
```

Figura 26: Ejecución de cobranza.py mostrando el recibo.

Action: 

-----

Go

 0 of 1 selected

<input type="checkbox"/>	CODIGO MASCOTA	CODIGO HABITACION	FECHA REGISTRO	FECHA FIN
<input type="checkbox"/>	Mascota: CLI-1-M1 - 'None', de morales	HAB004: BOB180.00	Nov. 29, 2023	Nov. 29, 2023

1 estadia

Figura 27: Revisión en django de que se agrego la fecha.

## C. reporte-por-periodo.py

```
(app) jassiel@IdeaPad-flex-5 ~/G/g/j/s/p/final (main)> python reporte-por-periodo.py
Fecha A de la forma 2023-12-30: 2023-11-01
Fecha B de la forma 2023-12-30: 2023-12-01
```

id	codigo_mascota	codigo_habitacion	fecha_registro	fecha_fin	estado	cantidad_en_hotel	cantidad_atendida
8	CLI-1-M1	HAB004	2023-11-29	2023-11-29	Huésped salió el 2023-11-29	0	1

```
(app) jassiel@IdeaPad-flex-5 ~/G/g/j/s/p/final (main)> _
```

Figura 28: Ejecución de reporte-por-periodo.py mostrando la lista de huéspedes atendidos.



## D. models.py

Listing 3: modelos para CRUD en django, en base a la base de datos. breaklines

```
1 # This is an auto-generated Django model module.
2 # You'll have to do the following manually to clean this up:
3 # * Rearrange models' order
4 # * Make sure each model has one field with primary_key=True
5 # * Make sure each ForeignKey and OneToOneField has 'on_delete' set to the desired
  behavior
6 # * Remove 'managed = False' lines if you wish to allow Django to create, modify,
  and delete the table
7 # Feel free to rename the models, but don't rename db_table values or field names.
8 # import uuid
9 from datetime import date
10 from django.db import models
11 from django.db.models import signals
12 from django.dispatch import receiver
13 from phonenumber_field.modelfields import PhoneNumberField
14 from djmoney.models.fields import MoneyField
15 from django.core.exceptions import ValidationError
16
17
18 def validatePastDate(value):
19     if value > date.today():
20         raise ValidationError("La fecha no puede ser futura.")
21
22 class CalendarioVacuna(models.Model):
23     codigo_mascota = models.ForeignKey('Mascota', models.DO_NOTHING, db_column='
  codigo_mascota')
24     fecha_vacunacion = models.DateField(default=date.today, validators=[
  validatePastDate])
25     cantidad_aplicada = models.IntegerField(default=1)
26     vacuna = models.ForeignKey('Vacuna', models.DO_NOTHING, db_column='vacuna')
27
28     def __str__(self):
29         return f"({self.fecha_vacunacion}; {self.codigo_mascota}): {self.vacuna}, {
  self.cantidad_aplicada}"
30
31     class Meta:
32         managed = False
33         db_table = 'calendario_vacuna'
34         unique_together = (('codigo_mascota', 'fecha_vacunacion', 'vacuna'),)
35
36 class Cliente(models.Model):
37     LISTA_BANCOS = [
38         ('ganadero', 'Banco Ganadero'),
```

```

39         ('sol', 'Banco Sol'),
40         ('nacional', 'Banco Nacional de Bolivia'),
41         ('mercantil_santa_cruz', 'Banco Mercantil Santa Cruz'),
42         ('bisa', 'Banco BISA'),
43         ('credito_bcp', 'Banco de Crédito BCP'),
44         ('fie', 'Banco FIE'),
45         ('union', 'Banco Unión'),
46         ['otro', 'Otro']
47     ]
48
49     codigo_cliente = models.CharField(primary_key=True, editable=False)
50     apellido_paterno = models.CharField(max_length=50)
51     cuenta_bancaria = models.CharField("Número de cuenta bancaria", max_length=25)
52     banco = models.CharField(choices=LISTA_BANCOS)
53     direccion = models.CharField(max_length=100)
54     telefono = PhoneNumberField(region='BO')
55     correo_electronico = models.EmailField(blank=True, null=True)
56
57     def save(self, *args, **kwargs):
58         if not self.codigo_cliente:
59             count = Cliente.objects.count() + 1
60             self.codigo_cliente = f"CLI-{count}"
61         super().save(*args, **kwargs)
62
63     def __str__(self):
64         return f"Cliente {self.apellido_paterno} - {self.codigo_cliente}, Teléfono: {self.telefono}, Correo-E: {self.correo_electronico}"
65
66     class Meta:
67         managed = False
68         db_table = 'cliente'
69
70
71 class Persona(models.Model):
72     LISTA_RELACIONES = [
73         ('cliente', 'Cliente mismo'),
74         ('padre', 'Padre'),
75         ('madre', 'Madre'),
76         ('hermano', 'Hermano'),
77         ('hermana', 'Hermana'),
78         ('tio', 'Tío'),
79         ('tia', 'Tía'),
80         ('abuelo', 'Abuelo'),
81         ('abuela', 'Abuela'),
82         ('primo', 'Primo'),
83         ('prima', 'Prima'),

```

```

84         ('amigo', 'Amigo'),
85         ('amiga', 'Amiga'),
86         ('vecino', 'Vecino'),
87         ('vecina', 'Vecina'),
88         ('conocido', 'Conocido'),
89         ('conocida', 'Conocida'),
90         ('otro', 'Otro')
91     ]
92
93     codigo_persona = models.CharField("Carnet de Identidad", primary_key=True,
94                                     max_length=15)
95     nombre = models.CharField()
96     relacion_cliente = models.CharField("Relación con el cliente", choices=
97                                       LISTA_RELACIONES, default='cliente')
98     telefono = PhoneNumberField()
99     direccion = models.CharField(blank=True, null=True)
100     correo_electronico = models.EmailField(blank=True, null=True)
101
102     def __str__(self):
103         return f"Persona: {self.nombre} - {self.codigo_persona}, Relación: {self.
104               relacion_cliente}, {self.telefono}, {self.direccion}, {self.
105               correo_electronico}"
106
107     class Meta:
108         managed = False
109         db_table = 'persona'
110
111 class Encargado(models.Model):
112     codigo_cliente = models.ForeignKey(Cliente, models.DO_NOTHING, db_column='
113                                     codigo_cliente')
114     codigo_persona = models.ForeignKey('Persona', models.DO_NOTHING, db_column='
115                                     codigo_persona')
116
117     def __str__(self):
118         return f"Cliente: {self.codigo_cliente.codigo_cliente} - {self.codigo_cliente
119               .apellido_paterno}, con persona: {self.codigo_persona.codigo_persona} - {
120               self.codigo_persona.nombre}"
121
122     class Meta:
123         managed = False
124         db_table = 'encargado'
125         unique_together = (('codigo_cliente', 'codigo_persona'),)
126
127 class Habitacion(models.Model):

```

```

122     codigo_habitacion = models.CharField(primary_key=True, max_length=5)
123     precio = MoneyField(max_digits=24, decimal_places=4, default_currency='BOB')
124
125     def clean(self):
126         if self.precio.amount <= 0:
127             raise ValidationError("El precio no debe ser negativo.")
128
129     def __str__(self):
130         return f"{self.codigo_habitacion}: {self.precio}"
131
132     class Meta:
133         managed = False
134         db_table = 'habitacion'
135
136 class EstadiaManager(models.Manager):
137     def filter_by_date_range(self, start_date, end_date):
138         return self.filter(fecha_registro__range=[start_date, end_date],
139                             fecha_fin__range=[start_date, end_date])
140
141 class Estadia(models.Model):
142     codigo_mascota = models.ForeignKey('Mascota', models.DO_NOTHING, db_column='
143         codigo_mascota')
144     codigo_habitacion = models.ForeignKey('Habitacion', models.DO_NOTHING, db_column='
145         codigo_habitacion')
146     fecha_registro = models.DateField(default=date.today)
147     fecha_fin = models.DateField(null=True, blank=True)
148
149     DisplayFields = ['codigo_mascota', 'codigo_habitacion', 'fecha_registro', '
150         fecha_fin']
151     SearchableFields = ['codigo_mascota', 'codigo_habitacion', 'fecha_registro', '
152         fecha_fin']
153     FilterFields = ['fecha_registro', 'fecha_fin']
154
155     @property
156     def dias_estadia(self):
157         if self.fecha_registro and self.fecha_fin:
158             return self.fecha_fin - self.fecha_registro
159         return None
160
161     def save(self, *args, **kwargs):
162         super(Estadia, self).save(*args, **kwargs)
163
164     # def clean(self):
165     #     if self.fecha_fin < self.fecha_registro and self.fecha_fin:
166     #         raise ValidationError("La fecha de salida (fín) debe ser mayor o igual
167     #             a la fecha de entrada (registro).")

```

```

162
163     def __str__(self):
164         return f"{self.codigo_habitacion}, {self.codigo_mascota}: ({self.
            fecha_registro}, {self.fecha_fin}), {self.dias_estadia}"
165
166     class Meta:
167         managed = False
168         db_table = 'estadia'
169         unique_together = (('codigo_mascota', 'codigo_habitacion', 'fecha_registro')
            ,)
170
171
172 class HistorialMedico(models.Model):
173     fecha_consulta = models.DateField(default=date.today, validators=[
        validatePastDate])
174     codigo_mascota = models.ForeignKey('Mascota', models.DO_NOTHING, db_column='
        codigo_mascota')
175     enfermedad = models.CharField()
176     detalle_diagnostico = models.CharField()
177     fecha_enfermedad = models.DateField(blank=True, null=True, default=date.today)
178     detalle_tratamiento = models.TextField(blank=True, null=True)
179
180     def clean(self):
181         if self.fecha_enfermedad > self.fecha_consulta:
182             raise ValidationError("La fecha de enfermedad no puede ser mayor a la
                fecha de consulta.")
183
184     def __str__(self):
185         return f"Fecha: {self.fecha_consulta}, Mascota {self.codigo_mascota.
            codigo_mascota} de {self.codigo_mascota.codigo_cliente.apellido_paterno}.
            "
186
187     class Meta:
188         managed = False
189         db_table = 'historial_medico'
190         unique_together = (('codigo_mascota', 'fecha_consulta'),)
191
192
193 def validateNonNegative(value):
194     if value <= 0:
195         raise ValidationError("El peso no puede ser negativo.")
196
197 class HistorialPeso(models.Model):
198     fecha_registro = models.DateField(default=date.today)
199     codigo_mascota = models.ForeignKey('Mascota', models.DO_NOTHING, db_column='
        codigo_mascota')

```

```

200     peso_kg = models.FloatField(verbose_name="Peso en kilogramos", validators=[
201         validateNonNegative])
202
203     def __str__(self):
204         return f"Fecha: {self.fecha_registro} - Código Mascota: ({self.codigo_mascota
205             .codigo_mascota}) - Peso: {self.peso_kg} Kg"
206
207     class Meta:
208         managed = False
209         db_table = 'historial_peso'
210         unique_together = (('fecha_registro', 'codigo_mascota'),)
211
212 def discard_old_records(sender, instance, **kwargs):
213     max_records = 10
214     records_to_keep = HistorialPeso.objects.filter(
215         codigo_mascota=instance.codigo_mascota
216     ).order_by('-fecha_registro')[:max_records]
217
218     # Delete records beyond the maximum allowed
219     HistorialPeso.objects.filter(
220         codigo_mascota=instance.codigo_mascota
221     ).exclude(
222         fecha_registro__in=[record.fecha_registro for record in records_to_keep]
223     ).delete()
224
225 @receiver(signals.post_save, sender=HistorialPeso)
226 def keep_latest_records(sender, instance, **kwargs):
227     # Call the discard_old_records function after a new record is saved
228     discard_old_records(sender, instance, **kwargs)
229
230 class Mascota(models.Model):
231     LISTA_SEXO = [
232         ('M', 'Macho'),
233         ('F', 'Hembra'),
234         ('O', 'Indefinido')
235     ]
236
237     codigo_mascota = models.CharField(primary_key=True, editable=False)
238     codigo_cliente = models.ForeignKey(Cliente, models.DO_NOTHING, db_column='
239         codigo_cliente', blank=True, null=True)
240     alias = models.CharField(blank=True, null=True)
241     especie = models.CharField()
242     sexo = models.CharField(max_length=1, choices=LISTA_SEXO)
243     raza = models.CharField(blank=True, null=True)
244     peso_actual_kg = models.FloatField(verbose_name="Peso en kilogramos", validators=[
245         validateNonNegative])

```

```

242     fecha_nacimiento = models.DateField(blank=True, null=True, default=date.today,
243                                           validators=[validatePastDate])
244
245     def save(self, *args, **kwargs):
246         if not self.codigo_mascota:
247             count = Mascota.objects.filter(codigo_cliente=self.codigo_cliente).count
248             count = count + 1
249             self.codigo_mascota = f"{self.codigo_cliente.codigo_cliente}-M{count}"
250             super().save(*args, **kwargs)
251
252     def __str__(self):
253         return f"Mascota: {self.codigo_mascota} - '{self.alias}', de {self.
254             codigo_cliente.apellido_paterno}"
255
256     class Meta:
257         managed = False
258         db_table = 'mascota'
259
260 class Requerimiento(models.Model):
261     codigo_servicio = models.ForeignKey('Servicio', models.DO_NOTHING, db_column='
262         codigo_servicio')
263     cantidad = models.IntegerField()
264     estadia = models.ForeignKey(Estadia, models.DO_NOTHING, db_column='estadia')
265     detalle = models.TextField(blank=True, null=True)
266
267     def clean(self):
268         if self.cantidad <= 0:
269             raise ValidationError("La cantidad debe ser mayor a cero.")
270
271     def __str__(self):
272         return f"({self.estadia.codigo_mascota}, {self.estadia.codigo_habitacion}, {
273             self.codigo_servicio}): {self.cantidad}"
274
275     class Meta:
276         managed = False
277         db_table = 'requerimiento'
278         # unique_together = (('codigo_mascota', 'codigo_habitacion', 'codigo_servicio
279             ', 'fecha_registro'),)
280
281 class Servicio(models.Model):
282     LISTA_SERVICIO = [
283         ('alimentación', 'Alimentación'),
284         ('aseo', 'Aseo'),

```

```

282         ('médico', 'Médico'),
283         ('otros', 'Otros'),
284         ('extras', 'Extras')
285     ]
286
287     codigo_servicio = models.CharField(primary_key=True, max_length=10)
288     tipo = models.CharField(max_length=100, choices=LISTA_SERVICIO)
289     precio = MoneyField(max_digits=24, decimal_places=2, default_currency='BOB')
290     detalle = models.TextField(blank=True, null=True)
291
292     def clean(self):
293         if self.precio.amount <= 0:
294             raise ValidationError("El precio no puede ser negativo.")
295
296     def __str__(self):
297         return f"{self.codigo_servicio} - {self.tipo}"
298
299     class Meta:
300         managed = False
301         db_table = 'servicio'
302
303
304
305 class Vacuna(models.Model):
306     tipo = models.CharField(max_length=100)
307     fabricante = models.CharField()
308     precio = MoneyField(max_digits=24, decimal_places=2, default_currency='BOB')
309
310     def __str__(self):
311         return f"{self.tipo}: {self.fabricante}, {self.precio}"
312
313     class Meta:
314         managed = False
315         db_table = 'vacuna'
316         unique_together = (('tipo', 'fabricante'),)

```