

---

# PROYECTO BASE DE DATOS

---

Supermercado



JANUARY 12, 2023  
UNIVERSIDAD DE LA LAGUNA  
Administración y diseño de base de datos  
Airam Rafael Luque León & Andrés Pérez Castellano

## Índice

<b>Objetivos</b>	<b>3</b>
<b>Supuesto teórico</b>	<b>4</b>
Gestión de empleados	5
Gestión de stock	6
Gestión de clientes y transacciones	7
Restricciones semánticas	8
<b>Modelo relacional</b>	<b>10</b>
<b>Grafo relacional</b>	<b>12</b>
<b>Implementación en Postgresql</b>	<b>13</b>
Triggers	13
<b>Carga inicial de datos</b>	<b>13</b>
<b>Consultas de prueba</b>	<b>14</b>
Consultas básicas	14
Tabla Cliente	14
Tabla Producto	14
Tabla Empleado	14
Tabla Cajero	15
Tabla Charcuteria	15
Tabla Logistica	15
Tabla Tienda	15
Tabla Almacén	15
Tabla Trabaja	16
Tabla DisponibilidadTienda	16
Tabla DisponibilidadAlmacen	17
Tabla Compra	17
Tabla Carrito	17

Tabla Transaccion	18
Otras consultas	18
<b>REST API</b>	<b>21</b>
Validación de datos: POST	23
Validación de datos: GET	25

## Objetivos

- Almacenar información de las tiendas que tenemos en la empresa (Dirección, identificador, superficie, etc.)
- Almacenar la información de los productos disponibles en las diferentes tiendas.
- Almacenar la información del personal contratado, designado a distintos puestos, así como información correspondiente al trabajo designado.
- Contemplar las diferentes categorías de productos de la empresa.
- Contemplar los diferentes roles de los empleados, pudiendo almacenar la información correspondiente de estos, como el cargo, la información personal, horario, etc.
- La base de datos deberá almacenar la información correspondiente a los clientes.
- Almacenar información de interés extraída de las transacciones realizadas.
- Actualización automática del stock de las tiendas, tras realizar una compra.
- Aplicación automática de descuentos a ciertos clientes.
- Tener un sistema preventivo de inserción de consultas erróneas (precios negativos, saldos por debajo del salario mínimo, etc.).
- Desarrollar una interfaz de comunicación tipo REST API, que facilite tanto a los usuarios finales, como a otras aplicaciones y servicios internos de la empresa el acceso a la información disponible en la base de datos.

## Supuesto teórico

Se desea desarrollar una base de datos para una empresa que tiene diferentes supermercados. Esta base de datos ha de almacenar la información necesaria para llevar un control de los productos que hay en cada una de las tiendas, de las cuales conocemos su identificador, dirección y superficie.

Debemos almacenar de cada producto información de interés como el identificador del producto, el nombre, el distribuidor, la marca, el precio, la descripción, la categoría ('alimentación', 'Limpieza', 'Higiene', 'Textil', 'Herramientas' y 'Otros') y la fecha de caducidad para algunos de ellos.

De igual manera se quiere llevar un control de la contratación de los empleados, guardando su información personal (DNI, nombre, apellidos, dirección, salario, horario de entrada y salida, número de cuenta y rol dentro de la tienda (Cajero, charcutería, logística, Gerente, pescadería, carnicería), destino y duración temporal de cada periodo laboral.

Cabe resaltar que el rol nos servirá para mantener almacenada la información de algunos puestos específicos, como las herramientas que usen, para poder llevar un control de las herramientas asignadas a cada uno, como puede ser a los encargados del almacén los montacargas de cada uno.

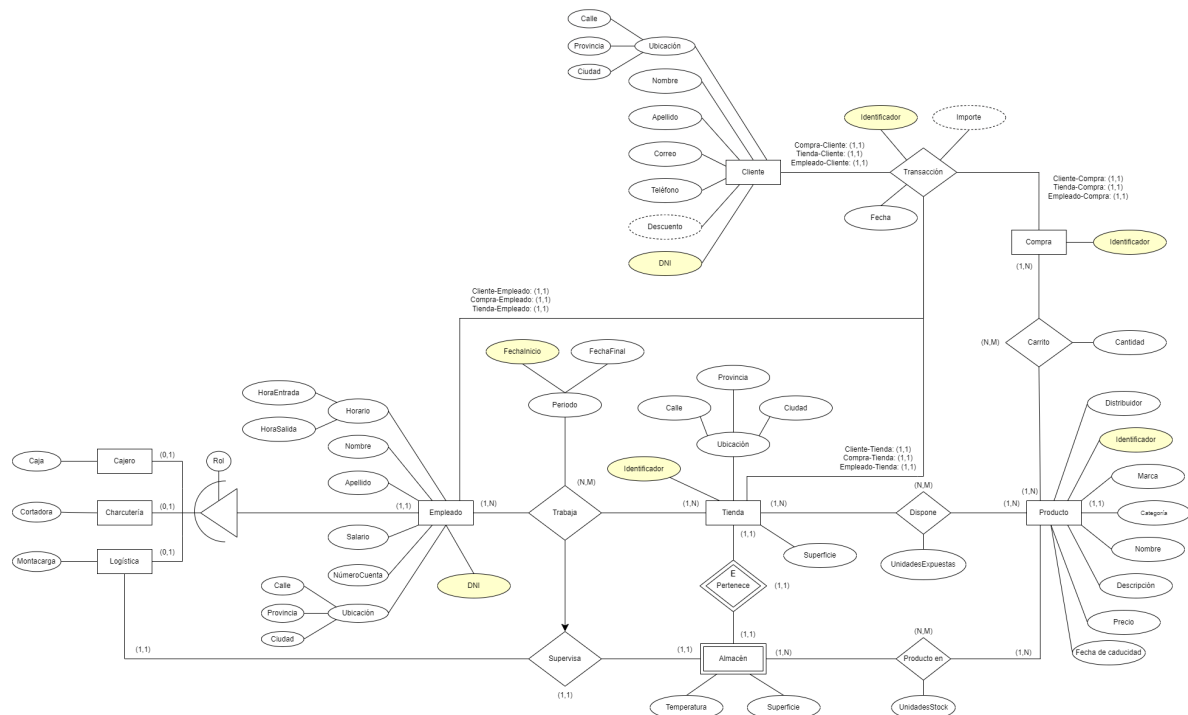
Otro aspecto a recalcar, es el hecho de que cada tienda tiene un único almacén, del cual nos interesa guardar la información de la superficie, la temperatura y el empleado (rol de logística) que se encarga de este. También tendremos que guardar el stock de cada producto en cada almacén, así como también queremos saber la cantidad de cada producto en exposición.

La empresa desea implementar un sistema de fidelización de clientes, por lo que queremos almacenar la información de los clientes, siendo esta DNI, nombre, apellido, correo, teléfono, dirección y un descuento que se le aplica aquellos que realicen compras totales por un valor superior a 100€, a nivel mensual.

Por último, la empresa quiere llevar un control de las transacciones que se realizan en cada una de las tiendas, almacenando el identificador del cliente que la ha realizado, el cajero que ha gestionado la compra, los productos comprados, el importe total, un identificador de la compra y la fecha de la misma. Todo esto servirá a su vez para aplicar el sistema de descuento a los clientes mencionados anteriormente, se les aplicará un descuento de 10€ en su próxima compra superior a 50€.

## Modelo entidad-relación (E-R)

De cara a las decisiones fundamentales de nuestro diseño las vamos a dividir en diferentes secciones: la gestión de los empleados, la gestión de los productos, y la gestión de los clientes y las transacciones.



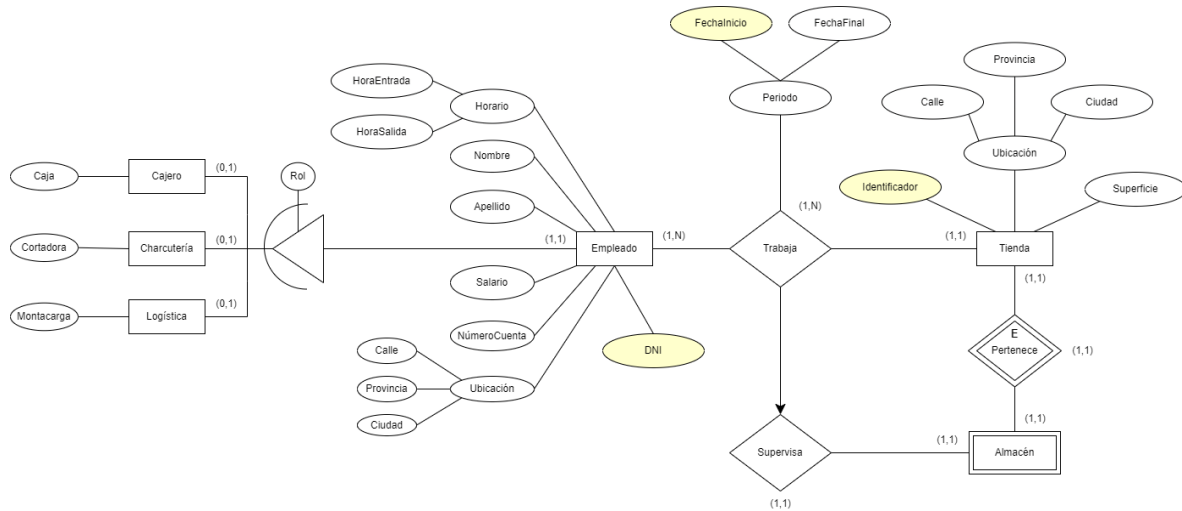
## Gestión de empleados

Como se nos estipula en el supuesto teórico, vamos a tener que almacenar cierta información de los empleados, de ahí que creemos dicha entidad, no obstante como se nos menciona que dependiendo del rol de estos, se almacenará la herramienta que tengan asignada, que no siempre existirá, se optó por hacer una implementación jerárquica de herencia, guardando en diferentes tablas simplemente dicha información. La entidad base Empleado existe porque pueden haber roles (por ejemplo gerente), que no disponen de herramientas específicas que deban tenerse en cuenta, de esta forma nos aseguramos de almacenar la información general de cada empleado y de forma independiente la específica, como en este caso es, la herramienta asignada.

Otro aspecto importante es la utilización de una relación de inclusión entre supervisor trabaja, puesto que un supervisor de un almacén debe de haber trabajado al menos en una ocasión en dicha tienda. Esta relación entre los almacenes, con los empleados de logística tiene cardinalidad 1 a 1, debido a que necesitamos que los almacenes tengan un supervisor y este sea destinado a ese rol.

Por otro lado, también vamos a tener que guardar un registro de las destinaciones de cada empleado y como debemos tener una entidad tienda para almacenar la información de interés de cada una de las tiendas, simplemente los relacionamos

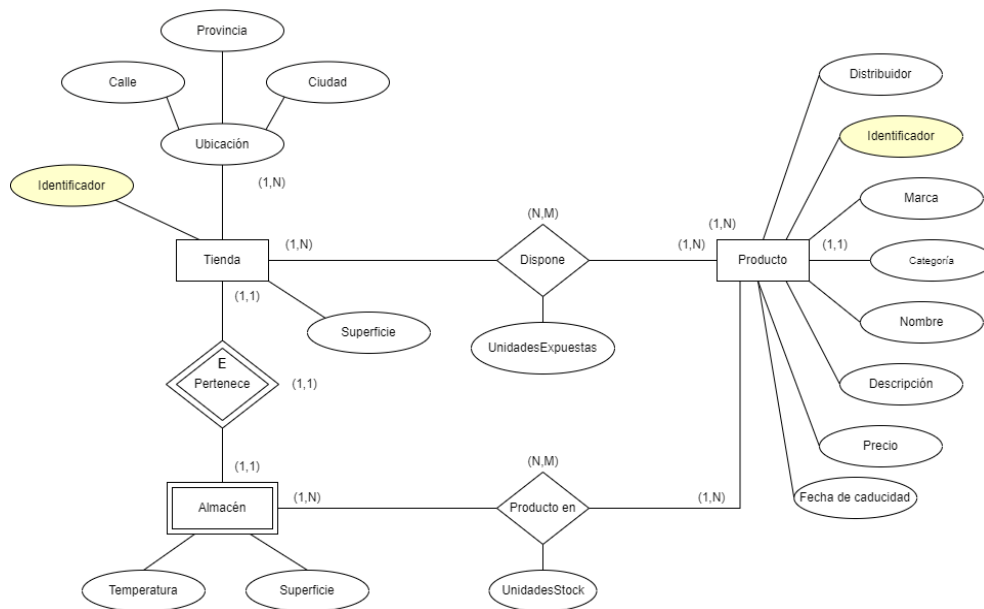
mediante una relación de “trabaja”. Nótese que un empleado puede cambiar su destino laboral de forma libre a lo largo del tiempo, por eso necesitamos que la fecha de inicio sea un atributo discriminante, la fecha de final puede llegar a ser nula, cuando el empleado continúa trabajando allí de forma indefinida. Ambas fechas son atributos propios de la relación, que se combinan para determinar el periodo laboral.



## Gestión de stock

Para la gestión de los productos en cada tienda, partimos de 3 entidades básicas: la tienda, el producto, y el almacén. Estas entidades optamos por unirlos con relaciones de “dispone” y “producto en”, con un atributo asociado a cada relación que determina la cantidad de cada producto. de esta forma registramos el stock expuesto en tienda y el stock en el almacén para cada uno de los supermercados.

Cabe resaltar que “almacén” es una entidad débil, debido a que cada almacén está ligado a una tienda en una relación 1 a 1 de pertenencia, por lo que, si desaparece la tienda, debería desaparecer su almacén.



## Gestión de clientes y transacciones

Con respecto a los clientes, éstos se representan como una entidad propia y se recogen diversos atributos. Los clientes pueden recibir descuentos en función de su volumen de gasto mensual, por ello dicho atributo debe calcularse posteriormente.

Como se ha comentado al inicio, las transacciones realizadas en las tiendas son un aspecto central del negocio y de nuestro diseño. Éstas se representan como una relación múltiple, que asocia 4 entidades fundamentales:

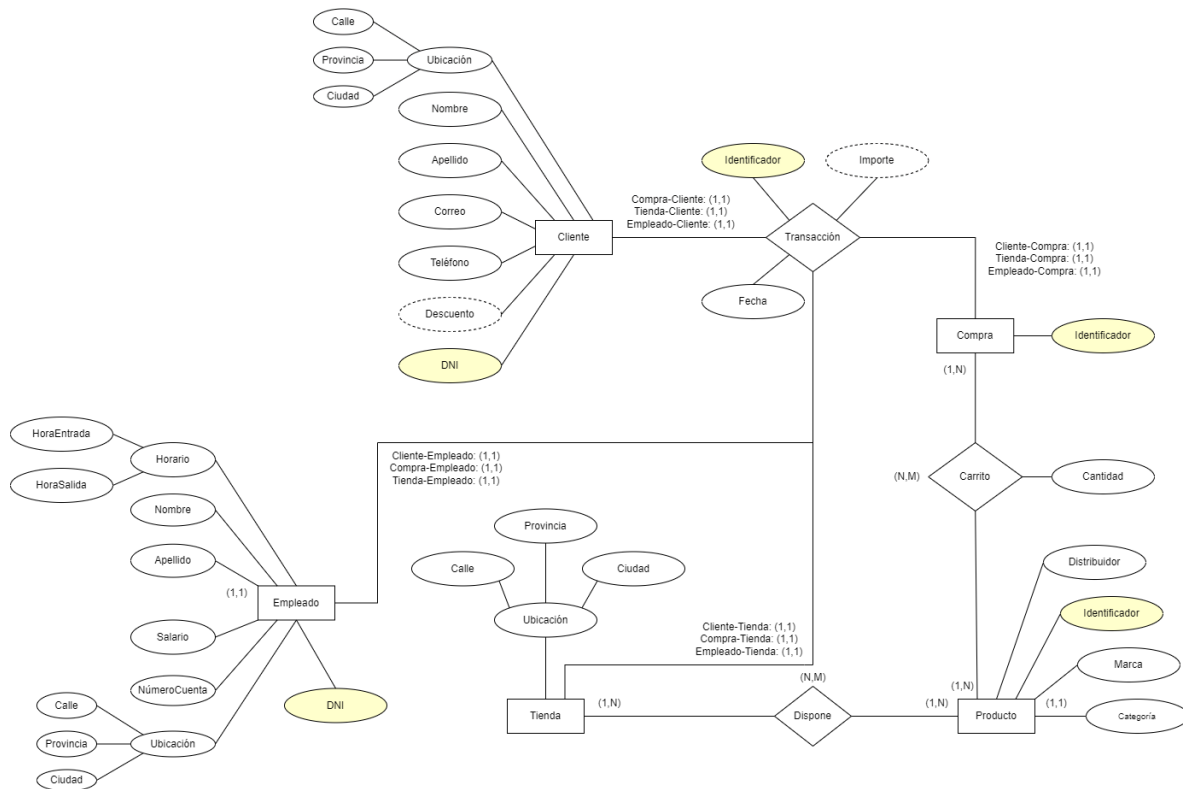
- Cliente. Es el cliente quien realiza la compra.
- Empleado. Atiende al cliente.
- Tienda. Lugar dónde se efectúa la compra.
- Compra. Representa los productos comprados.

Respecto a las cardinalidades de la relación, una transacción asocia de manera unívoca (según el identificador único de la transacción): un cliente, un empleado, una tienda y una compra.

Cabe destacar que el importe monetario total, se calcula automáticamente en base al contenido del carrito y los precios de los productos. Por tanto, cada transacción debe hacer referencia a un conjunto de productos (las compras individuales son poco frecuentes), esto es posible gracias a una entidad auxiliar llamada *Compra*, de lo contrario, no podríamos hacer referencia a la entidad *Carrito* a través de una clave ajena en *Transacción*, pues el atributo referenciado debe ser una clave primaria (por tanto, única por sí misma) en la otra entidad. Además, queremos garantizar el siguiente orden semántico: primero se llena la cesta, luego se ejecuta el pago. Es decir, el carrito de la compra debe existir antes de crear la transacción y no al revés.



Finalmente, la entidad Carrito contiene los (posiblemente múltiples) productos de una compra, así como las cantidades adquiridas para cada uno. Es necesario relacionarse con la entidad Producto, para poder obtener su precio.



## Restricciones semánticas

El modelo E-R y el modelo relacional no son capaces de representar todos los aspectos importantes que el supuesto debe cubrir, por lo que necesitamos especificar las restricciones semánticas oportunas que nos harán falta para garantizar la integridad de los datos. Estas restricciones se expresan más adelante en SQL, y se pueden implementar en el SGBD mediante triggers.

- Tanto el precio de un producto, la cantidad de producto disponible, la superficie de un edificio, como el importe de una transacción deben ser positivos.
- El descuento de un cliente se ubica en el intervalo 0 y 25€, se calcula y aplica en base al volumen de compras mensual y al dinero gastado.
- El identificador de una máquina/herramienta no puede ser negativo.
- La fecha de finalización debe ser posterior a la fecha de inicio, en un mismo periodo laboral.
- El salario mínimo en la compañía es de 900€.

- Un empleado de logística puede supervisar un almacén, siempre y cuando haya trabajado en esa tienda alguna vez.
- El descuento de 10€ se aplica para clientes cuyo gasto total supere 100€, en este último mes.
- El importe de una transacción se calcula en base al precio y la cantidad de los productos del carrito.
- Todos los productos de una compra deben estar disponibles en la tienda con suficiente cantidad.
- En una transacción, no se puede comprar un producto que no esté en esa tienda.
- En una transacción, debe atendernos un empleado de caja, que trabaje en dicha tienda.

## Modelo relacional

El modelo relacional que se obtiene a partir del modelo E-R es el siguiente:

### **Clave\_primaria**

*Clave ajena*

Atributo\_Calculado

Cliente (**DNI\_CLI**, Nombre, Apellidos, Correo, Teléfono, Calle, Ciudad, Provincia, Descuento)

Producto (**ID\_PROD**, Nombre, Distribuidor, Marca, Precio, Descripción, Categoría, FechaCaducidad)

Empleado(**DNI\_EMP**, Nombre, Apellidos, Calle, Ciudad, Provincia, Salario, HoraEntrada, HoraSalida, NumCuenta, rol)

Cajero(**DNI\_EMP**, Caja)

Foreign Key (DNI\_EMP) references Empleado(DNI\_EMP)

Charcutería(**DNI\_EMP**, Cortadora)

Foreign Key (DNI\_EMP) references Empleado(DNI\_EMP)

Logística(**DNI\_EMP**, Montacargas)

Foreign Key (DNI\_EMP) references Empleado(DNI\_EMP)

Tienda(**ID\_TIE**, Calle, Ciudad, Provincia, Superficie)

Almacén(**ID\_ALM**, Temperatura, Superficie, *DNI\_SUPER*)

Foreign Key (ID\_ALM) references Tienda(ID\_TIE)

Foreign Key (DNI\_SUPER) references Empleado(DNI\_EMP)

Trabaja(**DNI\_EMP**, **ID\_TIE**, **FechaInicio**, FechaFin)

Foreign Key (DNI\_EMP) references Empleado(DNI\_EMP)

Foreign Key (ID\_TIE) references Tienda(ID\_TIE)

DisponibilidadTienda(**ID\_TIE**, **ID\_PROD**, Cantidad)

Foreign Key (ID\_TIE) references Tienda(ID\_TIE)

Foreign Key (ID\_PROD) references Producto(ID\_PROD)

DisponibilidadAlmacen(**ID\_ALM**, **ID\_PROD**, Cantidad)

Foreign Key (ID\_ALM) references Almacén(ID\_ALM)

Foreign Key (ID\_PROD) references Producto(ID\_PROD)

Compra(**ID\_COMP**)

Transaccion(**ID\_TRANS**, DNI\_CLI, DNI\_EMP, ID\_TIE, ID\_COMP, Importe, Fecha)

Foreing Key (DNI\_CLI) references Cliente(DNI\_CLI)

Foreing Key (DNI\_EMP) references Empleado(DNI\_EMP)

Foreing Key (ID\_TIE) references Tienda(ID\_TIE)

Foreign Key (ID\_COMP) references Compra(ID\_COMP)

Carrito(**ID\_COMP**, **ID\_PROD**, Cantidad)

Foreign Key (ID\_COMP) references Compra(ID\_COMP)

Foreing Key (ID\_PROD) references Producto(ID\_PROD)

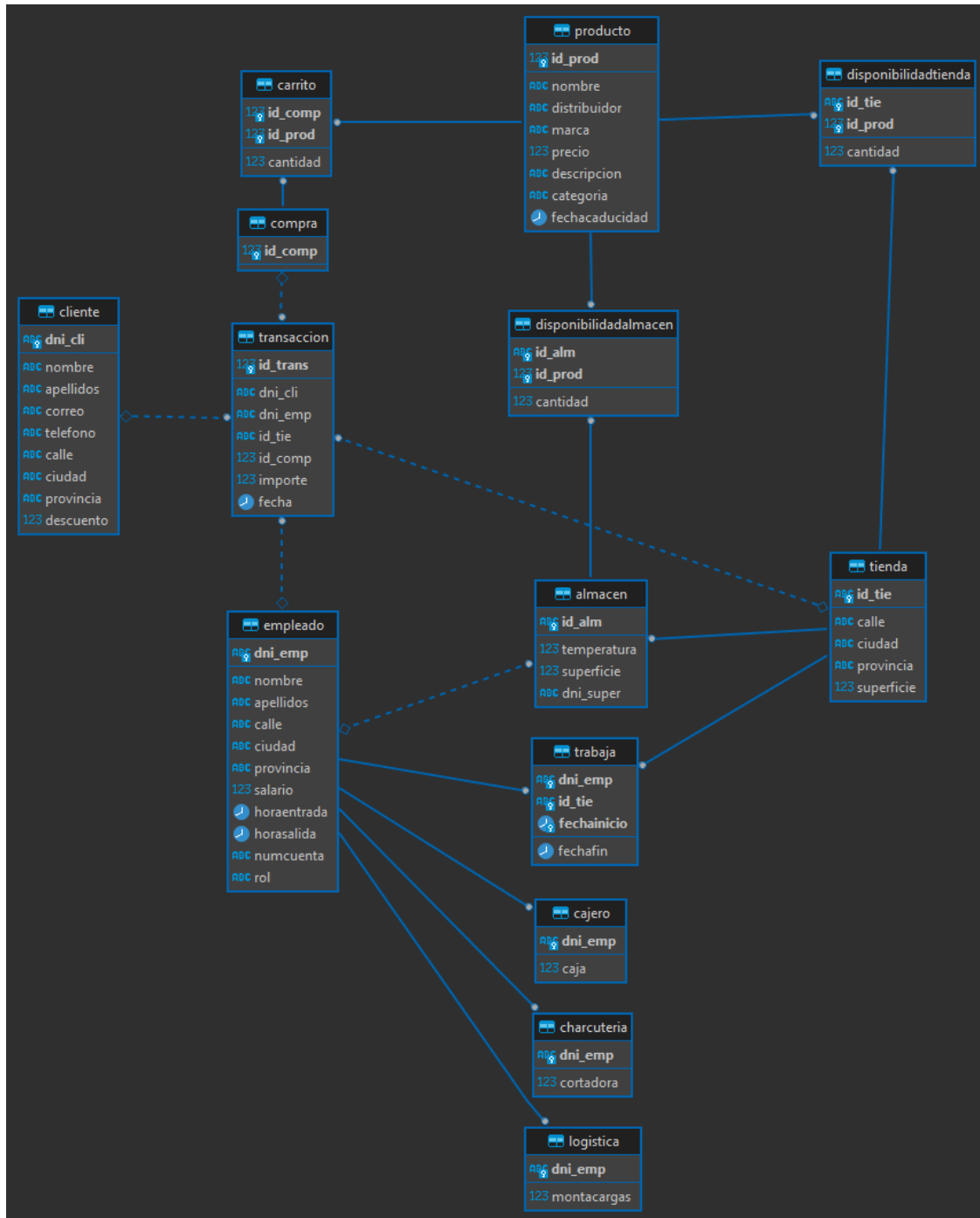
En él se ve como hemos tomado diversas decisiones de diseño, aunque cada entidad se convierte en una tabla, también hemos creado tablas adicionales desde relaciones, como puede ser el caso de “Trabaja”, “DisponibilidadTienda” y “DisponibilidadAlmacen”, cuyas cardinalidades son N:M. De esta forma estas tablas son las encargadas de representar dicha relación, además de almacenar los atributos de sus relaciones, como puede ser la cantidad, en el caso de DisponibilidadAlmacen.

Por otro lado la relación de herencia entre empleado y los diferentes tipos de empleados en base al rol, la resolvimos al agregar el atributo rol a la tabla empleado y tener tablas para cada uno de los roles que empleen herramientas, relacionados por el DNI con la tabla “empleado”.

De cara a la elección de claves primarias, cabe resaltar el hecho de mencionar que la tabla trabaja cuenta con una clave primaria compuesta, formada por las columnas DNI\_EMP, ID\_TIE y FECHA\_INICIO, ya que en la misma tabla se pueden encontrar dos tuplas con el mismo DNI\_EMP y ID\_TIE, pero con fechas de inicio diferentes, por lo que no se puede usar como clave primaria, de esta forma podemos tener un empleado que ha trabajado en la misma tienda en instantes diferentes de tiempo. De igual forma en la tabla trabaja también tenemos una doble clave primaria, formada por ID\_COMP y ID\_PROD, permitiendo que para una misma compra existan diversos productos.

## Grafo relacional

Aquí se puede ver el grafo relacional que se obtiene a partir de la base de datos desarrollada en la herramienta DBeaver.



## Implementación en Postgresql

Primero asegúrese de crear la base de datos en el sistema gestor:

```
CREATE DATABASE supermercado;
```

Luego ejecute el script `init_db.sql`, ubicado en el directorio “src/”. Allí se encuentra la definición de todas y cada una de las tablas implicadas, así como la definición de los disparadores y las restricciones oportunas.

```
sudo su postgres
psql
\c supermercado
\i init_db.sql
```

### Triggers

- Revisar que el supervisor del almacén sea un empleado de logística y ya haya trabajado en la tienda con anterioridad
- Calcular automáticamente el importe de una transacción, con respecto al carrito
- Revisar que, al intentar comprar un producto, esté en la tienda y haya suficiente
- Revisar que el empleado a cargo de la transacción sea un cajero y trabaje en esa tienda
- Actualizar el stock disponible de una tienda después de una compra

### Carga inicial de datos

Mediante el script correspondiente, insertamos múltiples registros en la base de datos, intentando cubrir tantos casos como sea posible.

```
sudo su postgres
psql
\c supermercado postgres
\i data.sql
```

Este script contiene información de ejemplo, que puede ser utilizada para revisar un funcionamiento básico de la base de datos, insertando diversas filas en cada una de las tablas e incluso disparando algunos triggers como en los que revisan que los empleados que llevan una compra sean cajeros de dicha tienda, el cálculo automático del importe dado un carrito, etc.

NOTA: La visualización de los valores de cada una de las tablas una vez cargada los datos mediante el script se mostrarán en las consultas de ejemplo.

## Consultas de prueba

Una vez hemos realizado una carga de información inicial en la base de datos, procedemos a comprobar su correcto funcionamiento, mediante algunas consultas de prueba.

### Consultas básicas

#### Tabla Cliente

```
SELECT DNI_CLI, NOMBRE, CORREO, TELEFONO, CALLE, CIUDAD, DESCUENTO FROM cliente;
```

dni_cli	nombre	correo	telefono	calle	ciudad	descuento
12345678A	Juan	JuanPerez@gmail.com	346404646	Calle 1	Ciudad 1	0
12345678B	Pepe	PepeGarcia@gmail.com	346656766	Calle 2	Ciudad 2	0
12345678C	Maria	MariaGonzalez@gmail.com	646886861	Calle 3	Ciudad 3	0
12345678D	Luis		122678646	Calle 4	Ciudad 4	0
12345678E	Ana		562163336	Calle 5	Ciudad 5	0
12345678G	Laura		969699626	Calle 7	Ciudad 7	0
12345678H	Antonio		346404646	Calle 8	Ciudad 8	0
12345678F	Jose		232362366	Calle 6	Ciudad 6	10

(8 rows)

#### Tabla Producto

```
SELECT ID_PROD, NOMBRE, MARCA, PRECIO, CATEGORIA, FECHACADUCIDAD  
FROM PRODUCTO;
```

id_prod	nombre	marca	precio	categoria	fechacaducidad
1	Crema hidratante	Soft Skin S.A.	12.99	Higiene	2023-01-31
2	Sábanas de algodón	Comfy Bedding S.L.	24.99	Textil	
3	Bolsa de deporte	Fit Gear S.A.	29.99	Otros	
4	Juego de cuchillos de cocina	Sharp Blades S.L.	89.99	Herramientas	2023-12-01
5	Aceite de oliva	Olive Oil S.A.	3.99	Alimentacion	2023-05-01
6	Limpiador multiusos	Fresh Home S.L.	4.99	Limpieza	
7	Cepillo de dientes eléctrico	Clean Teeth Inc.	49.99	Higiene	
8	Toallas de baño	Absorbent Towels S.A.	9.99	Textil	
9	Llave inglesa	Power Tools S.L.	9.99	Herramientas	
10	Regla metálica	Precise Measures S.A.	3.99	Otros	
11	Arroz integral	Healthy Grains S.L.	2.99	Alimentacion	2023-09-01
12	Leche desnatada	Low Fat Dairy S.A.	1.99	Alimentacion	2023-01-15
13	Jugo de naranja	Fresh Squeezed S.L.	2.49	Alimentacion	2023-04-01
14	Galletas integrales	Whole Wheat S.A.	3.99	Alimentacion	2023-02-01
15	Cereales integrales	Whole Grains S.L.	4.99	Alimentacion	2023-07-01

(15 rows)

#### Tabla Empleado

```
SELECT DNI_EMP, NOMBRE, CALLE, SALARIO, HORAENTRADA, NUMCUENTA, ROL  
FROM empleado;
```

dni_emp	nombre	calle	salario	horaentrada	numcuenta	rol
87654321A	Juan	Calle del Sol	1200	09:00:00	123456781234	Cajero
87654321B	Ana	Calle de la Luna	1400	08:00:00	123456791234	Cajero
87654321C	Pablo	Calle del Mar	1600	09:00:00	12345680234	Cajero
87654321D	Sandra	Calle de las Estrellas	1800	09:00:00	12345681234	Cajero
87654321E	Alberto	Calle del Rio	2000	09:00:00	12345682234	Charcuteria
87654321F	Laura	Calle de la Montaña	2200	09:00:00	12345683234	Charcuteria
87654321G	Javier	Calle del Cielo	2400	09:00:00	12345684234	Logistica

87654321H	Cristina	Calle de la Tierra	2600	09:00:00	12345685234	Logistica
87654321I	Raquel	Calle del Sol	2800	09:00:00	12345686234	Logistica
87654321J	Irene	Calle de la Luna	3000	09:00:00	12345687234	Gerente
87654321K	Manuel	Calle Castro	3000	09:00:00	12345687234	Pescaderia
87654321L	Maria	Calle Palomar	3000	09:00:00	43253234534	Carniceria

(12 rows)

### Tabla Cajero

**SELECT \* FROM CAJERO;**

dni_emp	caja
87654321A	1
87654321B	2
87654321C	3
87654321D	1

(4 rows)

### Tabla Charcuteria

**SELECT \* FROM CHARCUTERIA;**

dni_emp	cortadora
87654321E	1
87654321F	2

(2 rows)

### Tabla Logistica

**SELECT \* FROM LOGISTICA;**

dni_emp	montacargas
87654321G	1
87654321H	1
87654321I	4

(3 rows)

### Tabla Tienda

**SELECT \* FROM TIENDA;**

id_tie	calle	ciudad	provincia	superficie
T001	Calle de la Paz	Madrid	Madrid	200
T002	Calle del Sol	Barcelona	Cataluña	300
T003	Calle de la Luna	Valencia	Comunidad Valenciana	250
T004	Calle de las Estrellas	Sevilla	Andalucía	350
T005	Calle del Mar	Málaga	Andalucía	400
T006	Calle de la Montaña	Bilbao	País Vasco	300
T007	Calle del Río	Zaragoza	Aragón	250
T008	Calle del Bosque	Mallorca	Islas Baleares	350
T009	Calle de la Pradera	Granada	Andalucía	400
T010	Calle del Desierto	Tenerife	Islas Canarias	350

(10 rows)

### Tabla Almacén

**SELECT \* FROM ALMACEN;**

id_alm	temperatura	superficie	dni_super
T001	18	500	87654321G



T002	-2	700	87654321H
T003	0	400	87654321I
T004	24	1000	
T005	18	800	
T006	20	900	
T007	22	700	
T008	24	1200	
T009	18	1000	
T010	20	1200	

(10 rows)

### Tabla Trabaja

**SELECT \* FROM TRABAJA;**

dni_emp	id_tie	fechainicio	fechafin
87654321A	T001	2019-01-01	2019-02-01
87654321A	T001	2020-01-01	
87654321B	T007	2019-01-04	2019-02-27
87654321B	T002	2020-02-01	2020-07-31
87654321C	T003	2020-03-01	
87654321D	T004	2020-04-01	2020-09-30
87654321E	T001	2020-05-01	2020-10-31
87654321F	T002	2020-06-01	2020-11-30
87654321G	T001	2020-07-01	
87654321H	T002	2020-08-01	2021-01-31
87654321I	T003	2020-09-01	2020-10-01
87654321J	T001	2020-10-01	
87654321K	T001	2020-10-01	2021-03-31
87654321L	T001	2021-04-29	

(14 rows)

### Tabla DisponibilidadTienda

**SELECT \* FROM DISPONIBILIDADTIENDA;**

id_tie	id_prod	cantidad
T001	3	12
T001	6	31
T001	7	4
T001	9	1
T001	12	15
T001	13	20
T001	15	11
T002	4	10
T002	5	12
T002	9	5
T002	10	14
T002	11	19
T002	12	20
T004	3	2
T005	4	1
T006	2	9
T007	8	5
T008	2	6
T008	4	3
T009	1	2
T009	5	5
T010	3	8
T001	1	4

T001	2	5
T003	3	2
T003	1	6
T004	5	5
T001	4	37
T001	5	38
T001	10	32
T001	14	29
T001	11	13
T002	8	22
T001	8	5
T002	3	6
T003	14	7
T003	12	6
T003	7	5
T004	1	0

(39 rows)

#### Tabla DisponibilidadAlmacen

**SELECT \* FROM** DisponibilidadAlmacen;

id_alm	id_prod	cantidad
T001	1	10
T001	2	20
T002	3	30
T002	4	40
T003	5	50
T003	6	60
T004	7	70
T004	8	80
T005	9	90
T005	10	100

(10 rows)

#### Tabla Compra

**SELECT \* FROM** COMPRA;

id_comp
1
2
3
4
5
6
7
8
9
10

(10 rows)

#### Tabla Carrito

**SELECT \* FROM** CARRITO;

id_comp	id_prod	cantidad
1	1	3
1	2	2
2	2	1

3	3	2
3	1	1
4	5	3
5	4	6
5	5	4
5	10	10
5	14	2
5	11	3
6	8	10
7	8	4
8	3	3
9	14	2
9	12	4
9	7	1
10	1	1

(18 rows)

### Tabla Transaccion

**SELECT** id\_trans, dni\_cli, dni\_emp, id\_tie, id\_comp, importe **FROM** transaccion;

id_trans	dni_cli	dni_emp	id_tie	id_comp	importe
1	12345678A	87654321A	T001	1	88.95
2	12345678B	87654321A	T001	2	24.99
3	12345678F	87654321C	T003	3	72.97
4	12345678D	87654321D	T004	4	11.97
5	12345678F	87654321A	T001	5	612.75
6	12345678C	87654321B	T002	6	99.90
7	12345678G	87654321A	T001	7	39.96
8	12345678H	87654321B	T002	8	89.97
9	12345678F	87654321C	T003	9	65.93
10	12345678D	87654321D	T004	10	12.99

(10 rows)

### Otras consultas

Valor total de los productos almacenados por cada tienda.

**SELECT** ID\_TIE, **SUM**(Cantidad \* Precio) **AS** valor  
**FROM** DisponibilidadTienda **JOIN** Producto **USING** (ID\_PROD)  
**GROUP BY** ID\_TIE  
**ORDER BY** valor **DESC**;

id_tie	valor
T001	4849.429999999999
T002	1549.9199999999998
T003	427.74
T008	419.90999999999997
T010	239.92
T006	224.91
T005	89.99
T004	79.93
T007	49.95
T009	45.93000000000001

(10 rows)

Añadimos 2 unidades del producto 4 y 1 unidad del producto 2 a la tienda "T001".

```
INSERT INTO DisponibilidadTienda VALUES
('T001', 4, 2),
('T001', 2, 1);
```

Hacemos una compra (productos 2 y 4) superior a 100€.

id_prod	precio
2	24.99
4	89.99

(2 rows)

```
INSERT INTO Compra VALUES (2000);
```

```
INSERT INTO Carrito VALUES
(2000, 4, 1),
(2000, 2, 1);
```

id_comp	id_prod	cantidad
2000	4	1
2000	2	1

(2 rows)

```
INSERT INTO Transaccion
VALUES (DEFAULT, '12345678A', '87654321A', 'T001', 2000);
```

id_trans	dni_cli	dni_emp	id_tie	id_comp	importe	fecha
29	12345678A	87654321A	T001	2000	114.97999999999999	2023-01-10 19:10:22.245297

(1 row)

El cliente ya dispone de un descuento de 10€.

```
SELECT Descuento FROM Cliente WHERE DNI_CLI = '12345678A';
```

descuento
10

(1 row)

Veamos si en la siguiente compra se aplica el descuento.

```
INSERT INTO Compra VALUES (3030);
```

```
INSERT INTO Carrito VALUES (3030, 4, 1);
```

```
INSERT INTO Transaccion
VALUES (5050, '12345678A', '87654321A', 'T001', 3030);
```

```
SELECT Importe FROM Transaccion WHERE ID_TRANS = 5050;
```

importe
79.99

(1 row)

Por otra parte, disponemos de un script para comprobar el correcto funcionamiento de algunos disparadores. Probamos con algunos casos erróneos.

*-- Consultas que revisan el correcto funcionamiento de Los triggers, Las restricciones*

*-- Cambiar el supervisor de la tienda T001.*

*-- Se espera un error, el empleado nunca ha trabajado allí.*

```
UPDATE Almacen
SET DNI_SUPER = '87654321J'
WHERE ID_ALM = 'T001';
```

*-- check\_cajero:*

```
INSERT INTO Compra VALUES
(5000),
(5001),
(5002);
```

**INSERT INTO Carrito VALUES**

```
(5000, 1, 3),
(5000, 2, 2),
(5001, 3, 1),
(5002, 4, 2),
(5002, 1, 1000);
```

*-- trabaja en T001, pero no es cajero -> error*

```
INSERT INTO Transaccion VALUES
(DEFAULT, '12345678A', '87654321E', 'T001', 5000);
```

*-- cajero de otra tienda -> error*

```
INSERT INTO Transaccion VALUES
(DEFAULT, '12345678A', '87654321A', 'T002', 5001);
```

*-- check producto:*

*-- producto inexistente en una tienda o con stock insuficiente -> error*

```
INSERT INTO Transaccion VALUES
(DEFAULT, '12345678A', '87654321A', 'T001', 5002);
```

supermercado=# \i check.sql

psql:check.sql:8: ERROR: El empleado 87654321J no ha trabajado nunca en la tienda T001

CONTEXT: PL/pgSQL function check\_supervisa() line 8 at RAISE

psql:check.sql:25: ERROR: El empleado 87654321E no es cajero en la tienda T001

CONTEXT: PL/pgSQL function check\_cajero() line 7 at RAISE

psql:check.sql:29: ERROR: El empleado 87654321A no es cajero en la tienda T002

CONTEXT: PL/pgSQL function check\_cajero() line 7 at RAISE

psql:check.sql:35: ERROR: No hay stock suficiente de alguno de los productos de la compra 5002

CONTEXT: PL/pgSQL function check\_producto() line 12 at RAISE

## REST API

Para diseñar el REST API, hemos optado por el framework [FastAPI](#) para *Python*. Ofrece una serie de ventajas, entre ellas:

- Anotaciones de tipos para: clases, variables, parámetros de funciones, etc.
- Gracias a lo anterior: validación automática del cuerpo de la petición.
- Conversión de las respuestas a formato *json*.
- Generación de documentación interactiva.

Para instalar las dependencias en un entorno virtual, podemos ejecutar los siguientes comandos:

```
cd ProyectoADBD/  
  
python -m venv venv  
  
source ./venv/bin/activate  
  
pip install -r requirements.txt
```

Pruebe a usar `python3` si obtiene algún error. Este proyecto ha sido desarrollado con Python 3.8, en un entorno *Linux*.

Coloque las credenciales de su base de datos en un fichero `.env`, ubicado en la raíz del proyecto. Por ejemplo:

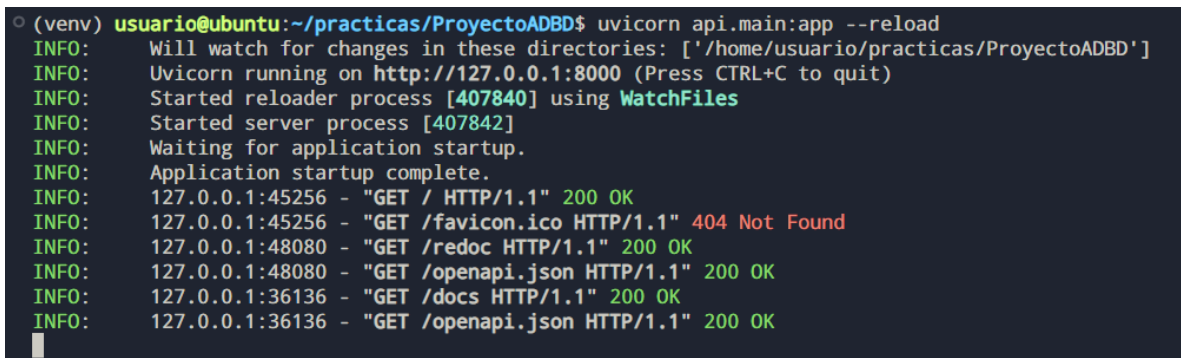
```
DB_NAME="supermercado"  
DB_USER="usuario"  
DB_PASSWORD="clave"
```

Para ejecutar el servidor, utilice la siguiente orden. Si ha seguido los pasos, debería estar ubicado en la raíz del proyecto. Puede cambiar el *host* y el puerto por defecto, si así lo desea.

```
uvicorn api.main:app
```

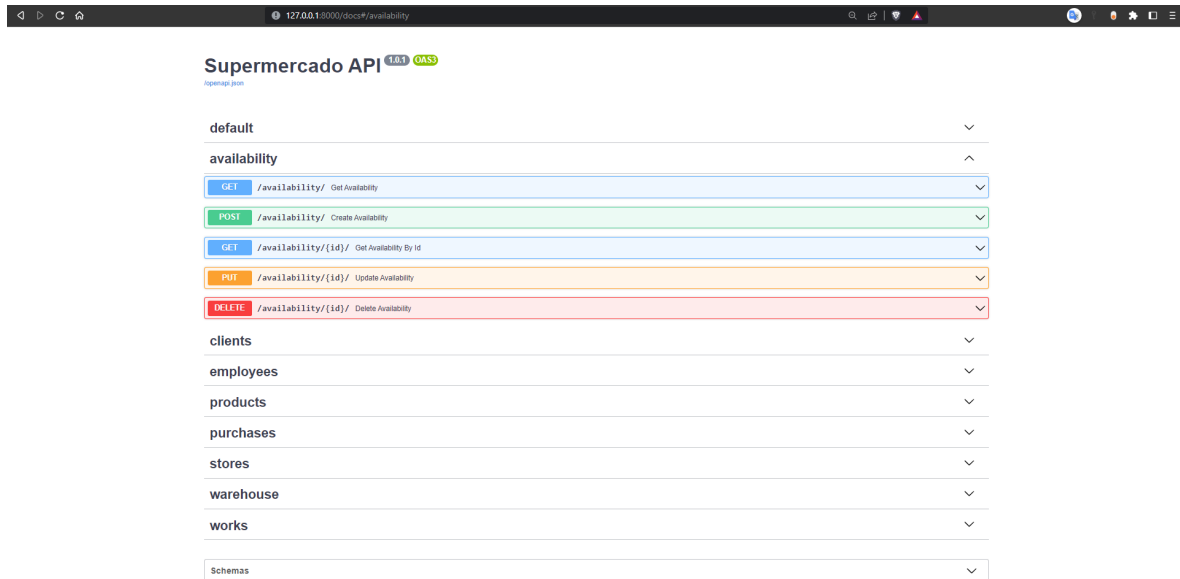
*# If you want to specify host address and port number.*

```
uvicorn api.main:app --host 127.0.0.1 --port 8000
```



```
(venv) usuario@ubuntu:~/practicass/ProyectoADBD$ uvicorn api.main:app --reload  
INFO: Will watch for changes in these directories: ['/home/usuario/practicass/ProyectoADBD']  
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)  
INFO: Started reloader process [407840] using WatchFiles  
INFO: Started server process [407842]  
INFO: Waiting for application startup.  
INFO: Application startup complete.  
INFO: 127.0.0.1:45256 - "GET / HTTP/1.1" 200 OK  
INFO: 127.0.0.1:45256 - "GET /favicon.ico HTTP/1.1" 404 Not Found  
INFO: 127.0.0.1:48080 - "GET /redoc HTTP/1.1" 200 OK  
INFO: 127.0.0.1:48080 - "GET /openapi.json HTTP/1.1" 200 OK  
INFO: 127.0.0.1:36136 - "GET /docs HTTP/1.1" 200 OK  
INFO: 127.0.0.1:36136 - "GET /openapi.json HTTP/1.1" 200 OK
```

Visite la documentación interactiva en <http://127.0.0.1:8000/docs> para conocer más sobre las funcionalidades ofrecidas. Puede probar las distintas operaciones **CRUD**, categorizadas por el método *http* (*GET*, *POST*, *PUT*, *DELETE*), haciendo click en cada *endpoint*.



Pruebe con una operación *GET*, haga click sobre *Try it out*, luego *execute*, eche un vistazo a la respuesta recibida. También puede copiar el comando *curl* sugerido en su terminal, si así lo prefiere.

Para las operaciones *GET*, se documenta el *json schema* de la respuesta, al igual que para el cuerpo de las peticiones *POST*. Puede completar los campos del objeto que aparece por defecto.

Respecto a la implementación, se han agrupado las rutas de acuerdo con un criterio semántico (productos, clientes, tiendas, etc), las operaciones **CRUD** se implementan en un fichero independiente. Finalmente, el fichero principal incorpora todas las rutas bajo la misma aplicación.

Para realizar consultas a la base de datos, usamos un conector de *Postgresql* para *Python*, llamado *psycopg2*. Se ha incorporado código para manejar excepciones, que se devuelven al cliente como un código de estado estándar acorde y un mensaje de error en formato *json*.

## Validación de datos: POST

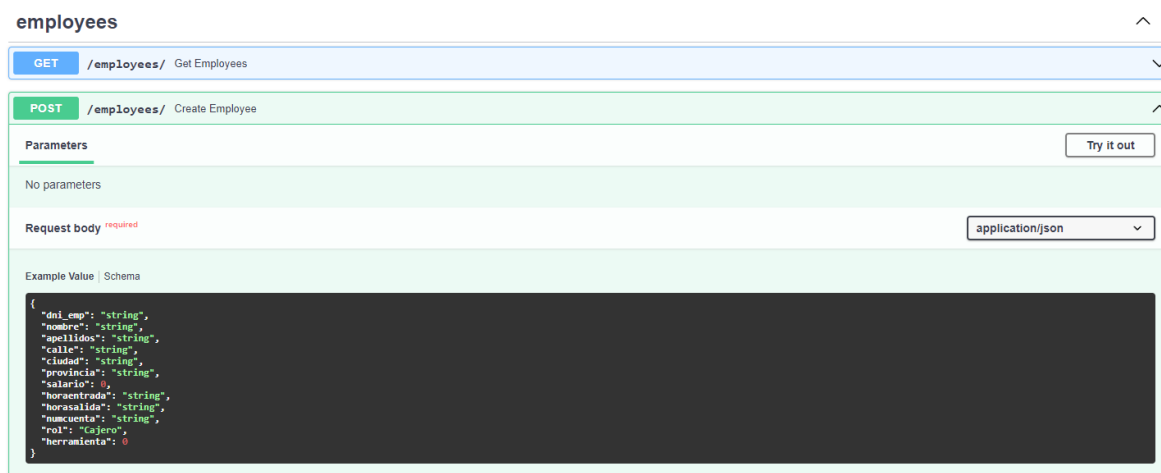
FastAPI permite definir de antemano el formato del cuerpo de la petición, con sus tipos de datos predefinidos. Estas anotaciones se integran perfectamente con las declaraciones de clases y funciones (parámetros y retorno).

```
class Puesto(Enum):
    CAJERO = "Cajero"
    CHARCUTERIA = "Charcuteria"
    LOGISTICA = "Logistica"
    GERENTE = "Gerente"
    PESCADERIA = "Pescaderia"
    CARNICERIA = "Carniceria"

class Empleado(BaseModel):
    dni_emp: str
    nombre: str
    apellidos: str
    calle: str
    ciudad: str
    provincia: str
    salario: float
    horaentrada: time
    horasalida: time
    numcuenta: str
    rol: Puesto
    herramienta: Union[int, None] = None # Atributo opcional.

# Endpoint handler function.
@router.post("/employees/", status_code=201)
def create_employee(employee: Empleado):
    ...
```

El framework aprovecha estas definiciones para documentar el *API* y puede proporcionar objetos *json* de ejemplo:



The screenshot displays the Swagger UI for a FastAPI application. The top section shows the 'employees' endpoint with a GET method for 'Get Employees'. Below this, the POST method for 'Create Employee' is highlighted. The 'Parameters' section indicates 'No parameters'. The 'Request body' section is marked as 'required' and shows a dropdown menu set to 'application/json'. The 'Example Value' section displays a JSON object representing an employee:

```
{
  "dni_emp": "string",
  "nombre": "string",
  "apellidos": "string",
  "calle": "string",
  "ciudad": "string",
  "provincia": "string",
  "salario": 0,
  "horaentrada": "string",
  "horasalida": "string",
  "numcuenta": "string",
  "rol": "Cajero",
  "herramienta": 0
}
```



Cuando se incumple con las restricciones anteriores, por ejemplo, se incluye un salario que no es numérico, FastAPI revisa la petición y devuelve un error claro y conciso al respecto

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/employees/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "dni_emp": "string",
    "nombre": "string",
    "apellidos": "string",
    "calle": "string",
    "ciudad": "string",
    "provincia": "string",
    "salario": "dfh",
    "horasentrada": "12:00:23",
    "horasalida": "15:00:23",
    "nuncuenta": "string",
    "rol": "Cajero",
    "herramienta": 0
  }'
```

Request URL

http://127.0.0.1:8000/employees/

Server response

Code	Details
422	Error: Unprocessable Entity

Response body

```
{
  "detail": [
    {
      "loc": [
        "body",
        "salario"
      ],
      "msg": "value is not a valid float",
      "type": "type_error.float"
    }
  ]
}
```

Download

De la misma forma, si un atributo no es opcional, entonces debe especificarse en cada petición, de lo contrario se considera un error.

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/employees/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "dni_emp": "string",
    "provincia": "string",
    "rol": "Cajero",
    "herramienta": 0
  }'
```

Request URL

http://127.0.0.1:8000/employees/

Server response

Code	Details
422	Error: Unprocessable Entity

Response body

```
{
  "detail": [
    {
      "loc": [
        "body",
        "nombre"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "apellidos"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "calle"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "ciudad"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "provincia"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "salario"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "horasentrada"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "horasalida"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "nuncuenta"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "herramienta"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    }
  ]
}
```

Download

## Validación de datos: GET

Ocurre algo similar para las respuestas de las peticiones *GET*, FastAPI se encarga de convertir los tipos y diccionarios de *Python* al formato *json*.

```
class CantidadProducto(BaseModel):
    id_prod: int
    cantidad: int

class TransaccionCompra(BaseModel):
    id_trans: int
    dni_emp: str
    dni_cli: str
    id_tie: str
    importe: float
    fecha: datetime
    carrito: List[CantidadProducto]

@router.get("/purchases/", status_code=200)
def get_purchases() -> List[TransaccionCompra]:
    ...
```

También se documenta en la interfaz interactiva, lo cual supone una información muy valiosa para los desarrolladores que quieran usar el *API*.

