DATA MODELLING Y PYTHON

1ºMIINT

PROYECTO FINAL

Nombres y correos electrónicos:

Daniel Aguilera — *202101788@alu.comillas.edu*

Álvaro Gracia — *202524068@alu.comillas.edu*

José Manuel Guinea aka Pepo — *202111777@alu.icai.comillas.edu*

Claudia María Agromayor — *202102295@alu.comillas.edu*

Curso y grupo: 1ºMIINT

Tabla de contenido

Índice de ilustraciones

Índice de tablas

CONTEXTUALIZACIÓN DEL PROYECTO

En los tiempos que corren, la digitalización ha pasado de ser una herramienta muy útil, a ser una necesidad para el prosperar de un negocio. De entre todos los sectores de producción que se han visto afectados por este cambio en el paradigma, el sector de la hostelería es, probablemente, el que más tiene que ganar (o que perder) en función de cómo lleven a cabo su paso del mundo físico al mundo virtual. En un mercado de competencia perfecta como puede ser el hostelero, toda herramienta que permita que los clientes se inclinen a consumir en uno u otro restaurante es bienvenida.

En este contexto nace TheKnife, como un proyecto que pretende conectar a clientes y restaurantes en una única plataforma, ofreciendo a cada uno distintas funcionalidades para facilitar el proceso de enlace entre ambos. Desde el punto de vista del cliente, TheKnife funciona como un gestor de reservas. A través de la plataforma, el cliente puede meterse dentro de su cuenta, realizar búsquedas de restaurantes de acuerdo con sus criterios de selección (localización, precio, estilo de comida, platos concretos, menú sin ciertos alérgenos…) y formalizar una reserva en dicho restaurante para un día y una hora determinados. El cliente podrá entonces gestionar esa reserva, por ejemplo, cambiándola de fecha, cambiando el número de personas, o cancelándola, sin necesidad de las tediosas llamadas al restaurante en caso de imprevistos. Asimismo, podrá llevar una gestión de las reservas que ya ha realizado, valorando los restaurantes y disponiendo de una copia del ticket, de forma que la gestión de los gastos será mucho más fácil y amigable para el usuario.

Por otro lado, permite al restaurante la gestión de las reservas, de las facturas, y de todo el flujo de datos que estas generan. Un restaurante, por ejemplo, podrá saber cuales son sus platos más vendidos, qué días tienen mayor clientela, o cuanto de media se gasta la gente en su restaurante, en comparación con otros restaurantes de la zona. Dispondrán además de los datos de sus clientes, como edad, frecuencia a la que visitan el restaurante, o las valoraciones que hacen del mismo, de forma que podrán usar toda esta información para el desarrollo y la evolución de su negocio.

El objetivo con TheKnife es claro: por un lado, facilitar a los clientes la gestión de sus reservas y salidas a restaurantes. Por otro, mejorar la gestión de información de las empresas para que puedan rentabilizar mejor su negocio.

1. DISTRIBUCIÓN DE RESPONSABILIDADES Y CONTRIBUCIÓN DE CADA MIEMBRO AL PROYECTO

La distribución de responsabilidades para el proyecto se estructuró en dos grandes áreas principales: Estructura de la Base de Datos (BBDD) y Ejecución e Implementación del Código en Python.

1. Estructura y creación de la base de datos
    - Responsales: Álvaro y José Manuel
    - Tareas clave
        - Diseño de la BBDD: Crear un esquema de base de datos en MySQL con las 8 entidades principales requeridas, asegurando que fueran representativas de la temática escogida.
        - Implementación de Consultas Avanzadas: Ejecutar consultas SQL complejas desde la consola que incluyeron diferentes tipologías de JOINs, uso de agrupaciones, funciones de agregación y manipulación, y el uso de subconsultas para demostrar la complejidad.
        - Control de la Manipulación de Datos: Implementar funciones de Crear, Actualizar y Eliminar desde la consola, incluyendo operaciones para gestionar múltiples filas y múltiples tablas. También se encargaron de establecer el control de borrado de claves primarias en la definición del esquema.

2. Ejecucuión del código e implementación con Python
    - Responsales: Daniel y Claudia María
    - Tareas clave:
    - Desarrollo de la Interfaz: Crear la interfaz de usuario con un menú sencillo que ofrece opciones mínimas para interactuar con la base de datos y archivos externos.
    - Conexión y Gestión de Errores: Implementar el login de usuario para conectar con la BBDD y añadieron la gestión de errores
    - Implementación del CRUD en Python: Desarrollar las funciones en Python para permitir al usuario realizar las operaciones básicas Crear, Leer, Actualizar y Eliminar registros sobre las tablas.
    - Operaciones de Ficheros : Implementar la carga/descarga de datos,desarrollando el código para componer dinámicamente las sentencias INSERT. Esto incluyó un control de errores para cargar solo los registros válidos y generar un informe de los fallidos.

- Visualización de Datos: Utilizar la librería Matplotlib para mostrar los datos de la BBDD, proporcionando una vista consolidada mediante la agrupación según criterios relevantes.

2. ESTRUCTURA DE LA BASE DE DATOS

2.1 EXPLICACIÓN EXPLÍCITA DEL MODELO

El primer paso en este proyecto fue la creación e implementación de una Base de Datos destinada a gestionar información de restaurantes. Se optó por un diseño relacional que permite modelar las entidades principales del negocio y las relaciones que existen entre ellas, asegurando la integridad y coherencia de los datos.

El diseño de la base de datos de restaurantes se estructura en ocho tablas principales e independientes. Estas entidades han sido definidas para cubrir los procesos clave de la gestión de un restaurante, desde la información de los usuarios hasta las transacciones y la retroalimentación.

A continuación, se detallan las tablas que componen el esquema relacional:

- Restaurantes: Contiene los datos de identificación, ubicación y características de cada establecimiento.
- Clientes: Almacena la información personal y de contacto de los usuarios.
- Platos: Registra los elementos del menú ofrecidos por los restaurantes.
- Facturas: Documenta las transacciones económicas realizadas.
- Reservas: Gestiona la planificación y el estado de las mesas reservadas por los clientes.
- Comandas: Contiene la información sobre los platos que se incluyeron en cada factura y en qué cantidad.
- Alergenos: Lista los diferentes tipos de alérgenos alimentarios.
- Alergia: Donde se indican los alérgenos que contienen cada plato.

El siguiente esquema ilustra el diseño relacional de la base de datos. En él se visualizan las entidades previamente mencionadas, sus atributos clave y las relaciones que las vinculan, estableciendo las claves primarias y claves foráneas necesarias. Este modelo garantiza que la BD pueda manejar de manera eficiente las operaciones de consulta.

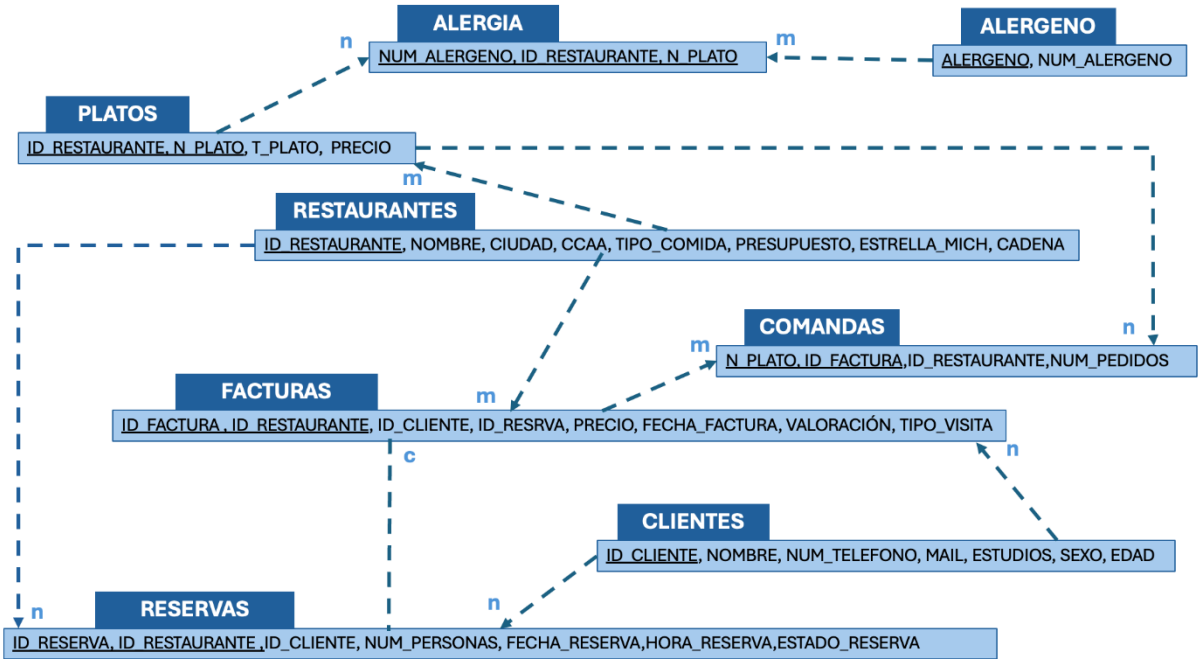


Ilustración 1. Diseño relacional de la base de datos

El diseño de la base de datos se basa en ocho tablas interconectadas para gestionar la información de restaurantes, clientes, platos, reservas, facturas, reseñas, comandas, alergenos y alergias.

2.2 MODELO LÓGICO DE LA BASE DE DATOS

1. RESTAURANTES

Esta es la tabla central del sistema, que almacena toda la información principal de los establecimientos.

ATRIBUTO	DESCRIPCIÓN	TIPO DATO	NOTAS Y EJEMPLOS
ID_RESTAURANTE	Identificador único del restaurante.	CHAR Primary key	Generado mediante una regla específica: 2 primeras letras del nombre + número de tres dígitos + siglas de la provincia (ejemplo: MB001PV).
NOMBRE	Nombre comercial del restaurante.	VARCHAR	(ejemplo: Martín Berasategui).
CIUDAD	Ciudad de ubicación del restaurante.	VARCHAR	(ejemplo: Lasarte-Oria).

CCAA	Comunidad Autónoma del restaurante.	VARCHAR	(ejemplo: País Vasco).
TIPO_COMIDA	Estilo culinario principal	VARCHAR	(ejemplo: Cocina de Autor, Vanguardia, Fusión).
PRESUPUESTO	Nivel de precio en una escala predefinida.	INT (1-5)	Permite la categorización por rango de precios. (ejemplo: 5 para Martín Berasategui).
ESTRELLA_MICH	Número de estrellas Michelin otorgadas.	INT (0-3)	(ejemplo: 3 para Martín Berasategui).
CADENA	Indica si el restaurante pertenece a una cadena o grupo.	VARCHAR NULL	(ejemplo: No para Martín Berasategui o Grupo Berasategui para el restaurant Lasarte).

Tabla 1. Estructura de la tabla RESTAURANTES

2. CLIENTES

En esta tabla se recoge toda la información relativa respecto a los clientes usuarios que utilizan el servicio de reservas y visitan los restaurantes.

ATRIBUTO	DESCRIPCIÓN	TIPO DATO	NOTAS Y EJEMPLOS
ID_CLIENTE	Identificador único del cliente.	CHAR Primary key	En formato de DNI: 8caracter numéricos + 1 letra (ejemplo: 05333811P)
NOMBRE	Nombre complete del cliente.	VARCHAR	Apellido, Nombre del cliente (ejemplo: Estrada, Angelina).
NUM_TELEFONO	Número de contacto del cliente.	INT	9 dígitos sin prefijo (ejemplo: 667345091).
MAIL	Correo electrónico del cliente.	VARCHAR	Correo electrónico o NULL si no lo ah introducido (ejemplo: angelina.estrada@hotmail.com).
ESTUDIOS	Nivel de estudios alcanzado por el cliente.	VARCHAR	(ejemplo: Estudiante, Trabajador, Jubilado o no definido con las siglas (NC)).
SEXO	Sexo del cliente.	CHAR	F para femenino, M para masculino, O para otros.
EDAD	Edad actual del cliente.	INT	(ejemplo: 54).

Tabla 2. Estructura de la tabla CLIENTES

3. PLATOS

Esta tabla almacena el menú de cada restaurante, incluyendo el nombre y el precio de cada oferta gastronómica.

ATRIBUTO	DESCRIPCIÓN	TIPO DATO	NOTAS Y EJEMPLOS
ID_RESTAUARNTE	Identificador único del restaurante.	CHAR Primary key Foreign key respecto a la tabla RESTAURANTES	Generado mediante una regla específica: 2 primeras letras del nombre + número de tres dígitos + siglas de la provincia (ejemplo: MB001PV).
N_PLATO	Nombre del plato en dicha carta del restaurant.	VARCHAR Primary key	(ejemplo: Jamón Ibérico).
T_PLATO	Tipo de plato	VARCHAR	(ejemplo: Entrante, Postre, Principal, Bebida).
PRECIO	Precio de venta del plato.	FLOAT	(ejemplo: 24.50).

Tabla 3. Estructura de la tabla PLATOS

4. FACTURAS

Esta tabla registra los detalles finales de la transacción económica, incluyendo el precio total, la fecha y la valoración del cliente.

ATRIBUTO	DESCRIPCIÓN	TIPO DATO	NOTAS Y EJEMPLOS
ID_FACTURA	Identificador único de la factura.	CHAR Primary key	Generado mediante una regla específica: 1letra+1nuemro aleatorio+6 números correspondientes al día en formato DDMMAA (ejemplo: A0240917)
ID_RESTAURANTE	Identificador único del restaurante.	CHAR Primary key Foreign key respecto a la tabla RESTAURANTES	Generado mediante una regla específica: 2 primeras letras del nombre + número de tres dígitos + siglas de la provincia (ejemplo: MB001PV).
ID_CLIENTE	Identificador único del cliente.	CHAR Foreign key respect a la tabla CLIENTES.	En formato de DNI: 8caracter numéricos + 1 letra (ejemplo: 05333811P)
ID_RESERVA	Identificador único de la reserva.	CHAR	Generado con el siguiente formato: 6 dígitos correspondientes a la fecha con

		Foreign key respecto a la tabla RESRVAS.	estructura AAMMDD + letra + número aleatorio (ejemplo: 170924U1)
PRECIO	Precio total de la factura.	FLOAT	(ejemplo: 252.46).
FECHA_FACTURA	Fecha en que se emitió dicha factura.	DATE	Fecha en formato AAAA-MM-DD (ejemplo: 2017-09-24)
VALORACIÓN	Puntuación de la experiencia otorgada por el cliente.	INT (1-5)	Numero entero del 1 al 5 o NULL si el cliente no ha realizado la valoración (ejemplo: 5).
TIPO_VISITA	Motivo o contexto de la visita.	VARCHAR	(ejemplo: Pareja, Solo, Empresa, Grupo).

Tabla 4. Estructura de la tabla FACTURAS

5. RESERVAS

Esta tabla registra cada solicitud de mesa realizada por un cliente en un restaurante específico.

ATRIBUTO	DESCRIPCIÓN	TIPO DATO	NOTAS Y EJEMPLOS
ID_RESERVA	Identificador único de la reserva.	CHAR Primary key	Generado con el siguiente formato: 6 digitos correspondientes a la fecha con estructura AAMMDD + letra + número aleatorio (ejemplo: 170924U1)
ID_RESTAURANTE	Identificador único del restaurante.	CHAR Primary key Foreign key respecto a la tabla RESTAURANTES	Generado mediante una regla específica: 2 primeras letras del nombre + número de tres dígitos + siglas de la provincia (ejemplo: MB001PV).
ID_CLIENTE	Identificador único del cliente.	CHAR Foreign key, respecto a la tabla CLIENTES.	En formato de DNI: 8caracter numéricos + 1 letra (ejemplo: 05333811P)
NUM_PERSONAS	Número de comensales.	INT	Ejemplo: 2
FECHA_RESERVA	Día de la reserva.	DATE	Fecha en formato AAAA-MM-DD (ejemplo: 2017-09-24)
HORA_RESERVA	Hora acordad de la reserva.	CHAR	Hora en formato HH:MM (ejemplo: 20:15)

ESTADO_RESRVA	Estado actual de la reserva.	VARCHAR	(ejemplo: Confirmada, Cancelada).

Tabla 5. Estructura de la tabla RESRVAS

6. COMANDAS

Esta tabla enlaza los platos que se han pedido con la factura final, registrando las cantidades solicitadas.

ATRIBUTO	DESCRIPCIÓN	TIPO DATO	NOTAS Y EJEMPLOS
N_PLATO	Nombre del plato en dicha carta del restaurant.	VARCHAR Primary key Foreign key respecto a la tabla PLATOS	(ejemplo: Jamón Ibérico).
ID_FACTURA	Identificador único de la factura.	CHAR Primary key Foreign key respecto a la tabla FACTURAS	Generado mediante una regla específica: 1letra+1nuemro aleatorio+6 números correspondientes al día en formato DDMMAA (ejemplo: A0240917)
ID_RESTAURANTE	Identificador único del restaurante.	CHAR Primary key Foreign key respecto a la tabla RESTAURANTES	Generado mediante una regla específica: 2 primeras letras del nombre + número de tres dígitos + siglas de la provincia (ejemplo: MB001PV).
NUM_PEDIDOS	Cantidad de veces que se pidió ese plato en esta factura.	INT	Ejemplo: 2.

Tabla 6. Estructura de la tabla COMANDAS

7. ALERGENOS

Esta tabla es el catálogo que lista y codifica los diferentes tipos de alérgenos que deben ser controlados.

ATRIBUTO	DESCRIPCIÓN	TIPO DATO	NOTAS Y EJEMPLOS
NUM_ALERGENO	Identificador único del tipo de alérgeno.	INT Primary key	(ejemplo: número 10, correspondiente únicamente al alergeno “gluten”).
ALERGENO	Nombre del alérgeno	VARCHAR	(ejemplo: Gluten, Molusco…).

Tabla 7. Estructura de la tabla ALERGENOS

8.ALERGIAS

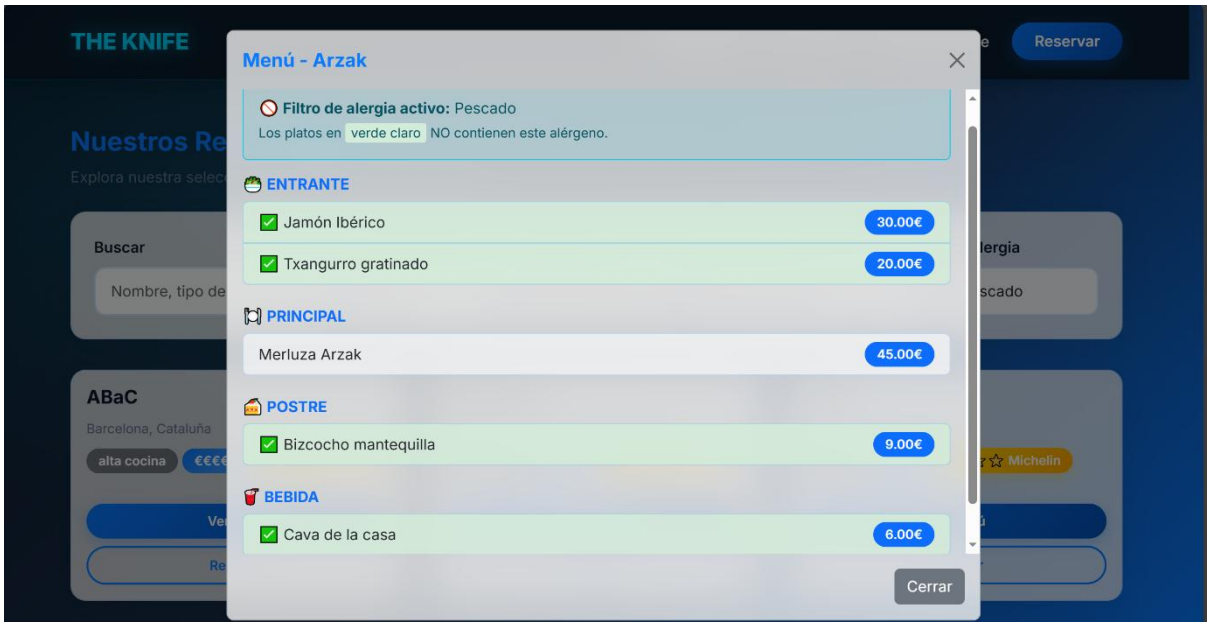
Esta tabla de relación establece la conexión entre un plato específico y los alérgenos que contiene.

ATRIBUTO	DESCRIPCIÓN	TIPO DATO	NOTAS Y EJEMPLOS
NUM_ALERGENO	Identificador único del tipo de alérgeno.	INT Primary key Foreign key respecto a la tabla ALERGENOS	(ejemplo: número 10 correspondiente únicamente al alérgeno “gluten”).
ID_RESTAURANTE	Identificador único del restaurante.	CHAR Primary key Foreign key respecto a la tabla RESTAURANTES	Generado mediante una regla específica: 2 primeras letras del nombre + número de tres dígitos + siglas de la provincia (ejemplo: MB001PV).
N_PLATO	Nombre del plato en dicha carta del restaurant.	VARCHAR Primary key Foreign key respecto a la tabla PLATOS	(ejemplo: Jamón Ibérico).

Tabla 8. Estructura de la table ALERGIAS

3. CONSULTAS SQL

1. Restaurantes con opciones sin un alérgeno (tipo: GET)



Queremos obtener los restaurantes que ofrecen al menos un plato que **no contiene** el alérgeno indicado por el usuario. Es decir, identificar locales que tienen alguna opción apta para la alergia recibida como parámetro.

La consulta parte de la tabla *restaurantes* y, para cada restaurante, comprueba si existe algún plato suyo que no aparezca en la lista de platos asociados al alérgeno. Esto se consigue usando un WHERE EXISTS sobre *platos*, combinado con una subconsulta NOT IN que devuelve todos los platos que incluyen ese alérgeno. Si el restaurante tiene aunque sea un plato que no está en esa lista, se devuelve como resultado.

Ejemplo con 'Huevo' (sería %s para la string) del frontend:

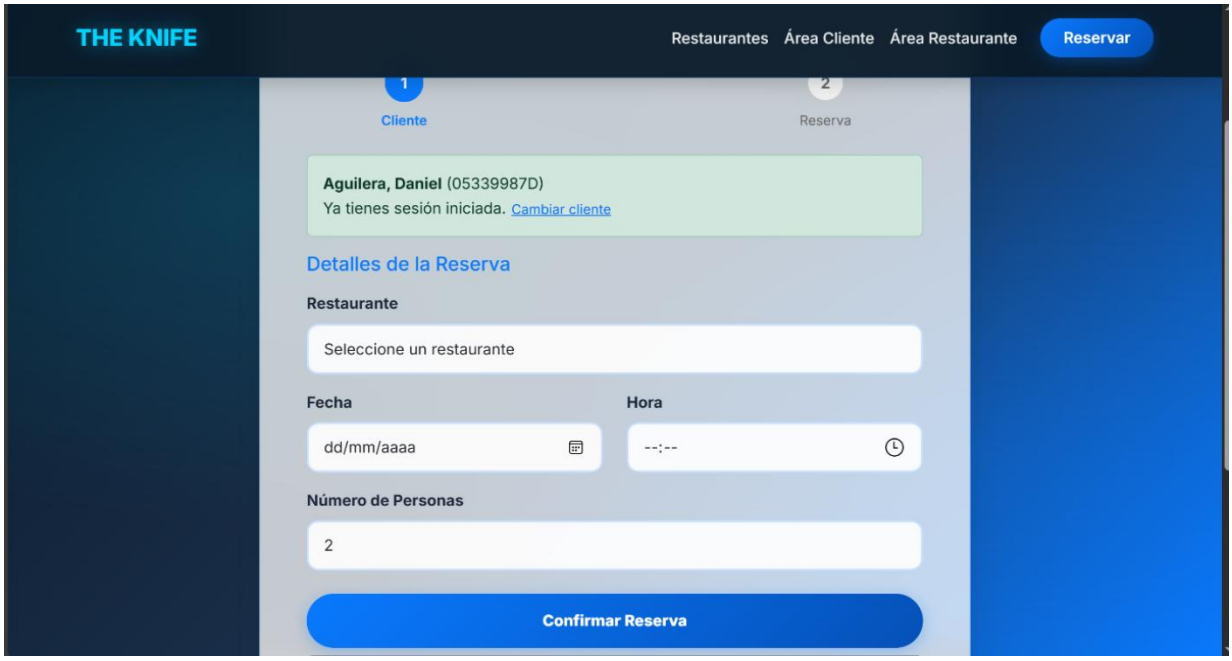
```
SELECT DISTINCT r.ID_RESTAURANTE, r.NOMBRE, r.CIUDAD, r.CCAA, r.T_COMIDA,
                r.PRESUPUESTO, r.ESTRELLA_MICH, r.CADENA
FROM restaurantes r
WHERE EXISTS (
  SELECT 1
  FROM platos p
  WHERE p.ID_RESTAURANTE = r.ID_RESTAURANTE
    AND (p.ID_RESTAURANTE, p.N_PLATO) NOT IN (
      SELECT al.ID_RESTAURANTE, al.N_PLATO
      FROM alergias al
      JOIN alergenos a ON al.NUM_ALERGENO = a.NUM_ALERGENO
      WHERE a.ALERGENO = 'Huevo'
```

```
    )
)
ORDER BY r.ESTRELLA_MICH DESC, r.NOMBRE;
```

El resultado: todos los restaurantes que tienes al menos un plato que pueda tomar el cliente con alergia a huevo.

```
mysql> SELECT DISTINCT r.ID_RESTAURANTE, r.NOMBRE, r.CIUDAD, r.CCAA, r.T_COMIDA,
    ->                 r.PRESUPUESTO, r.ESTRELLA_MICH, r.CADENA
    -> FROM restaurantes r
    -> WHERE EXISTS (
    ->   SELECT 1
    ->   FROM platos p
    ->   WHERE p.ID_RESTAURANTE = r.ID_RESTAURANTE
    ->     AND (p.ID_RESTAURANTE, p.N_PLATO) NOT IN (
    ->       SELECT al.ID_RESTAURANTE, al.N_PLATO
    ->       FROM alergias al
    ->       JOIN alergenos a ON al.NUM_ALERGENO = a.NUM_ALERGENO
    ->       WHERE a.ALERGENO = 'Huevo'
    ->     )
    -> )
    -> ORDER BY r.ESTRELLA_MICH DESC, r.NOMBRE;
+---------------+--------------------------+---------------------------+---------------------+--------------+-------------+--------------+---------+
| ID_RESTAURANTE | NOMBRE                   | CIUDAD                    | CCAA                | T_COMIDA     | PRESUPUESTO | ESTRELLA_MICH | CADENA  |
+---------------+--------------------------+---------------------------+---------------------+--------------+-------------+--------------+---------+
| AB001CT       | ABaC                     | Barcelona                 | Cataluña            | alta cocina  |           5 |            3 | NULL    |
| AP001AN       | Aponiente                | El Puerto de Santa María  | Andalucía           | marinera     |           5 |            3 | NULL    |
| AA003PV       | Arzak                    | San Sebastián             | País Vasco          | vasca        |           5 |            3 | NULL    |
| AZ002PV       | Azurmendi                | Larrabetzu                | País Vasco          | vanguardia   |           5 |            3 | NULL    |
| MN003CT       | Disfrutar                | Barcelona                 | Cataluña            | vanguardia   |           5 |            3 | NULL    |
| DR001MD       | DiverXO                  | Madrid                    | Comunidad de Madrid | vanguardia   |           5 |            3 | NULL    |
| MB001PV       | Martín Berasategui       | Lasarte-Oria              | País Vasco          | alta cocina  |           5 |            3 | NULL    |
| RQ002VC       | Quique Dacosta           | Dénia                     | Comunidad Valenciana | alta cocina |           5 |            3 | NULL    |
| PB001EX       | Restaurante Atrio        | Cáceres                   | Extremadura         | alta cocina  |           5 |            3 | NULL    |
| AS002CB       | Annua                    | San Vicente de la Barquera | Cantabria          | alta cocina  |           5 |            2 | NULL    |
| MS001AS       | Casa Marcial             | Parres                    | Asturias            | asturiana    |           5 |            2 | NULL    |
| CM002GA       | Culler de Pau            | O Grove                   | Galicia             | alta cocina  |           5 |            2 | NULL    |
| AS001NA       | El Molino de Urdániz     | Urdániz                   | Navarra             | navarra      |           5 |            2 | NULL    |
| CA001VC       | El Poblet                | Valencia                  | Comunidad Valenciana | mediterránea |           5 |            2 | NULL    |
| TC001CN       | El Rincón de Juan Carlos | Adeje                     | Canarias            | alta cocina  |           5 |            2 | NULL    |
| MB003CN       | M.B.                     | Guía de Isora             | Canarias            | internacional |           5 |            2 | NULL    |
| PA003GA       | Pepe Vieira              | Poio                      | Galicia             | creativa     |           5 |            2 | NULL    |
| RE003MD       | Ramón Freixa Madrid      | Madrid                    | Comunidad de Madrid | alta cocina  |           5 |            2 | NULL    |
| VA001IB       | Zaranda                  | Palma de Mallorca         | Islas Baleares      | alta cocina  |           5 |            2 | NULL    |
| AA002AN       | Abantal                  | Sevilla                   | Andalucía           | andaluza     |           4 |            1 | NULL    |
| JO003IB       | Adrián Quetglas          | Palma de Mallorca         | Islas Baleares      | contemporánea |           4 |            1 | NULL    |
| AE003CM       | Alma Mater               | Cuenca                    | Castilla-La Mancha  | moderna      |           4 |            1 | NULL    |
```

2. Reservar restaurante (tipo: Insert)



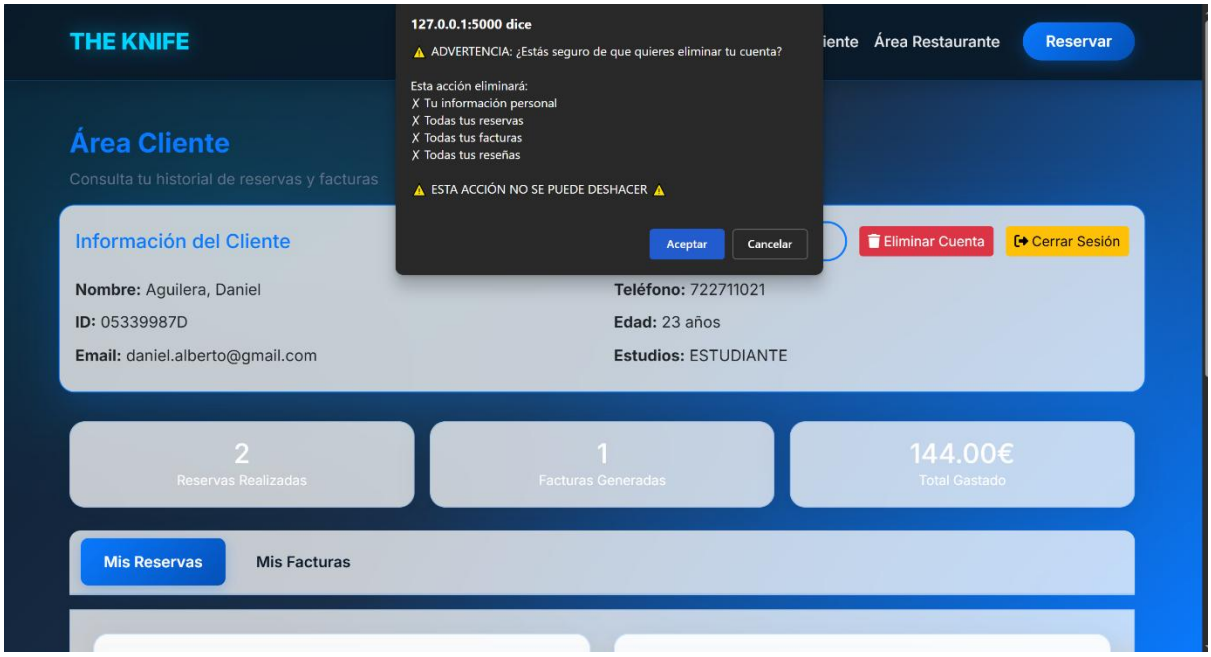
Buscamos insertar una nueva reserva en la base de datos, asignándole un ID generado y marcándola directamente como “Confirmada”.

Para ello el sistema genera el código de la reserva y ejecuta un INSERT con los datos enviados (cliente, número de personas, fecha, hora y restaurante). Lo hacemos con un INSERT INTO y VALUES que asigna por posición. Tras ejecutarse, la operación se guarda en la base de datos mediante COMMIT.

```
INSERT INTO reservas (
    ID_RESERVA, ID_CLIENTE, NUM_PERSONAS,
    FECHA_RESERVA, HORA_RESERVA, ID_RESTAURANTE, ESTADO_RESERVA
)  VALUES  ('AB12CD34',  '40477975W',  4,  '2025-11-30',  '20:00',  'MB001PV',
'Confirmada');
```

```
mysql> INSERT INTO reservas (
    ->     ID_RESERVA, ID_CLIENTE, NUM_PERSONAS,
    ->     FECHA_RESERVA, HORA_RESERVA, ID_RESTAURANTE, ESTADO_RESERVA
    -> ) VALUES ('AB12CD34', '40477975W', 4, '2025-11-30', '20:00', 'MB001PV', 'Confirmada');
Query OK, 1 row affected (0.19 sec)
```

3. Borrar cliente (tipo: Delete)

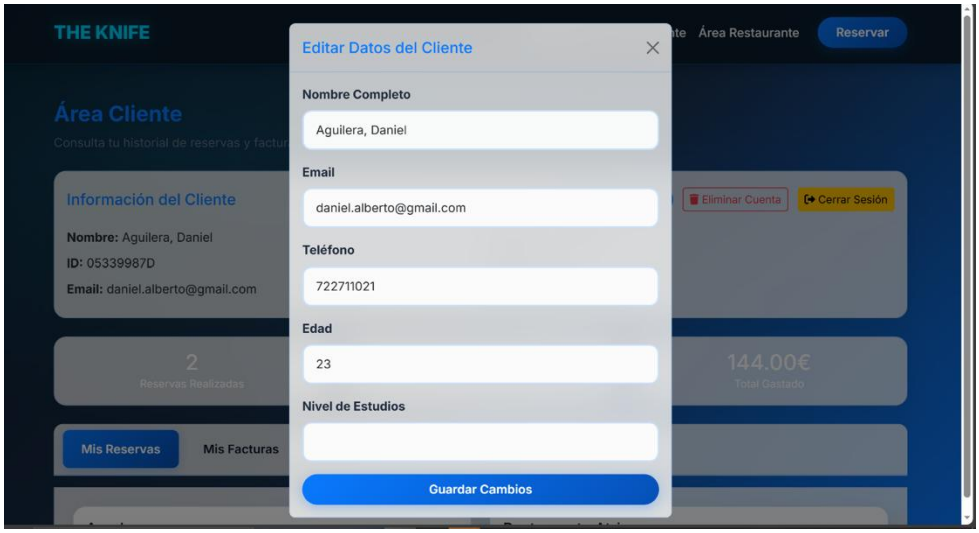


Un ejemplo de uso de delete en la DB

```
DELETE FROM clientes WHERE ID_CLIENTE = 'CL001';
```

```
mysql> DELETE FROM clientes WHERE ID_CLIENTE = 'CL001';
Query OK, 0 rows affected (0.00 sec)
```

4. Actualizar información cliente (tipo: Update)



Queremos modificar la información de un cliente ya existente, como su nombre, teléfono, email, estudios, sexo o edad. De forma genérica usamos la string ID_CLIENTE %s con la que también validamos si el ID que se quiere actualizar ya existe.

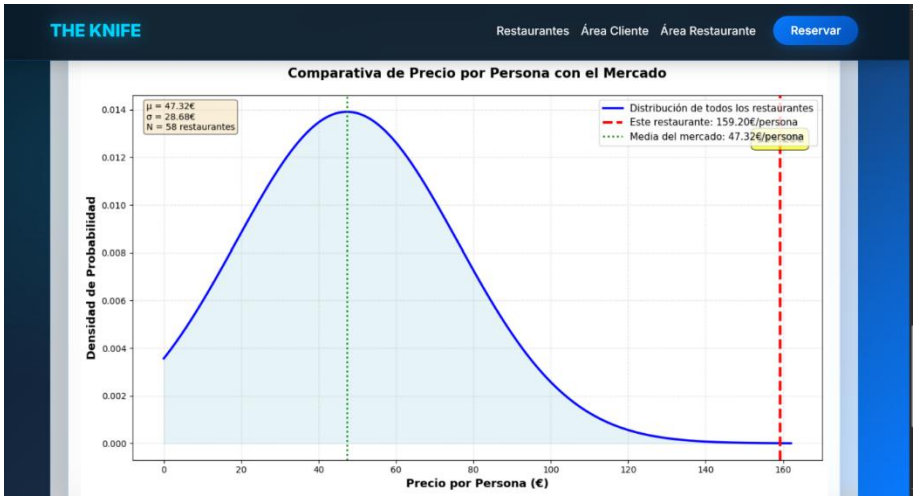
Ejemplo:

```
UPDATE clientes
SET
    N_CLIENTE = 'María López',
    NUM_TELEFONO = 600123456,
    EMAIL = 'maria.lopez@gmail.com',
    ESTUDIOS = 'Universitarios',
    SEXO = 'F',
    EDAD = 34
WHERE ID_CLIENTE = 'CL001';
```

```
mysql> UPDATE clientes
    -> SET
    ->     N_CLIENTE = 'María López',
    ->     NUM_TELEFONO = 600123456,
    ->     EMAIL = 'maria.lopez@gmail.com',
    ->     ESTUDIOS = 'Universitarios',
    ->     SEXO = 'F',
    ->     EDAD = 34
    -> WHERE ID_CLIENTE = '40477975W';
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

5. Cálculo de gasto de medio de persona por restaurante para tabla Python (tipo: GET)



Queremos calcular cuánto gasto medio por persona genera cada restaurante, usando los datos de las facturas y del número de personas de cada reserva.

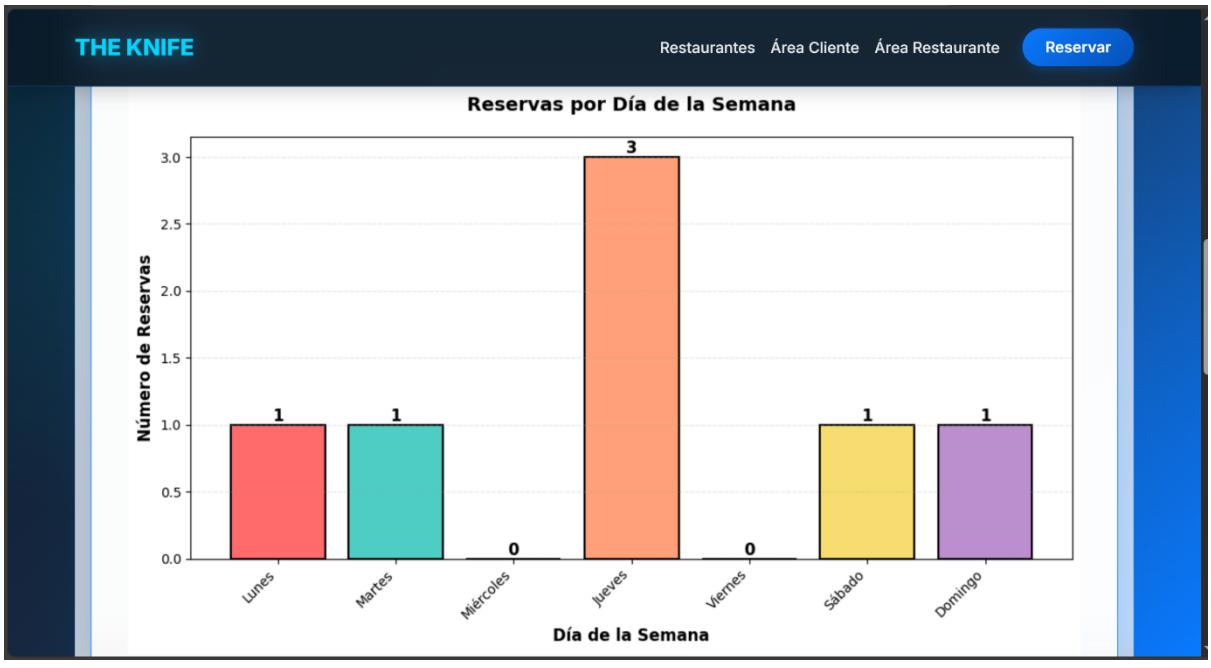
Por tanto, la consulta une las tablas *facturas* y *reservas* para relacionar el total cobrado con el tamaño del grupo. Después calcula el promedio del importe por persona (AVG(precio / num_personas)) agrupando por restaurante. Solo se muestran resultados cuando el cálculo es válido (no nulos).

```sql
SELECT
    r.ID_RESTAURANTE,
    AVG(f.PRECIO / res.NUM_PERSONAS) AS gasto_medio
FROM facturas f
JOIN reservas res ON f.ID_RESERVA = res.ID_RESERVA
JOIN restaurantes r ON f.ID_RESTAURANTE = r.ID_RESTAURANTE
GROUP BY r.ID_RESTAURANTE
HAVING gasto_medio IS NOT NULL;
```

```
mysql> SELECT
    ->     r.ID_RESTAURANTE,
    ->     AVG(f.PRECIO / res.NUM_PERSONAS) AS gasto_medio
    -> FROM facturas f
    -> JOIN reservas res ON f.ID_RESERVA = res.ID_RESERVA
    -> JOIN restaurantes r ON f.ID_RESTAURANTE = r.ID_RESTAURANTE
    -> GROUP BY r.ID_RESTAURANTE
    -> HAVING gasto_medio IS NOT NULL;
+----------------+----------------+
| ID_RESTAURANTE | gasto_medio    |
+----------------+----------------+
| AA002AN        |   6.4698611110 |
| AA003PV        |  30.1932857138 |
| AB001CT        |  45.3665561223 |
| AE003CM        |  27.6344285714 |
| AM002CM        |  32.4796666664 |
| AP001AN        | 159.1967380951 |
| AS001NA        |  43.6292857140 |
| AS002CB        |  45.6102500000 |
| AZ002PV        |  53.7401870747 |
| BK001CT        |  89.8160000000 |
| BL003CB        |  19.2911269838 |
| BR002NA        |  46.3429067458 |
| CA001VC        | 112.1056250000 |
| CD002CT        |  81.0878124999 |
| CL001CL        |  49.4200000000 |
| CM001MD        |  48.2309047618 |
| CM002GA        |  37.0204166665 |
```

6.

Reservas por día de la semana restaurante



Se quiere contar cuántas reservas tiene un restaurante según el día de la semana, que usaremos más tarde en nuestras gráficas con Python usando Matplotlib.

A partir de *reservas*, se obtiene el día de la semana de cada fecha mediante DAYOFWEEK y DAYNAME. Luego se agrupa por estos valores y se hace un conteo. Los resultados se ordenan por el número del día para mantener una secuencia coherente al mostrar los datos.

```sql
SELECT
    DAYOFWEEK(FECHA_RESERVA) AS dia_num,
    DAYNAME(FECHA_RESERVA) AS dia_nombre,
    COUNT(*) AS total
FROM reservas
WHERE ID_RESTAURANTE = 'MB001PV'
GROUP BY
    DAYOFWEEK(FECHA_RESERVA),
    DAYNAME(FECHA_RESERVA)
ORDER BY DAYOFWEEK(FECHA_RESERVA);
```

```
mysql> SELECT
    ->     DAYOFWEEK(FECHA_RESERVA) AS dia_num,
    ->     DAYNAME(FECHA_RESERVA) AS dia_nombre,
    ->     COUNT(*) AS total
    -> FROM reservas
    -> WHERE ID_RESTAURANTE = 'MB001PV'
    -> GROUP BY
    ->     DAYOFWEEK(FECHA_RESERVA),
    ->     DAYNAME(FECHA_RESERVA)
    -> ORDER BY DAYOFWEEK(FECHA_RESERVA);
+---------+------------+-------+
| dia_num | dia_nombre | total |
+---------+------------+-------+
|       1 | Sunday     |     2 |
|       5 | Thursday   |     2 |
|       6 | Friday     |     1 |
+---------+------------+-------+
3 rows in set (0.00 sec)
```

4. ARQUITECTURA PYTHON Y ESTRUCTURA DEL CÓDIGO

Para *theknife* hemos optado por una base de datos donde toda su lógica reside en los endpoints Flask. En Python no definimos modelos de dominio complejos ni clases, únicamente empleamos clases utilitarias como DevelopmentConfig para la configuración y el Blueprint del frontend que agrupa las vistas.

El backend se centraliza en app.py, donde definimos nuestra API REST para gestionar restaurantes, platos, reservas, facturas, reseñas, alérgenos y analítica. La aplicación se inicializa con la configuración DevelopmentConfig (en config.py), registra el Blueprint del frontend y establece la conexión MySQL. Además, durante el desarrollo nos encontramos con muchos problemas de acentos, eñes y otros caracteres especiales que no se mostraban bien. Para ello, forzamos la codificación UTF-8 (utf8mb4) tanto en la configuración de la base de datos como en cada conexión que hace la aplicación. Además, añadimos un ajuste extra en Flask usando un @before_request para asegurarnos de que absolutamente todas las consultas y respuestas usaran siempre UTF-8.

El frontend se organiza mediante un Blueprint definido en frontend.py, que sirve las vistas principales: index, restaurants, reservations, clients, book y restaurant_area. Las plantillas se encuentran en templates/ y los scripts en static/js/. Para los módulos de analítica, generamos gráficos con Matplotlib y enviamos las imágenes como cadenas base64 directamente al navegador.

4.1. Endpoints y Funcionalidad

A continuación describimos los endpoints principales definidos en **app.py**, explicando para qué los utilizamos y cómo encajan en el flujo global de la plataforma.

Restaurantes y Platos

GET /api/restaurantes[?alergia=...]
Lista todos los restaurantes ordenados aplicando, si procede, un filtro por alergia. Solo mostramos restaurantes con al menos un plato seguro. Además, ocultamos el campo *CADENA* si su valor es NULL.

GET /api/restaurantes/<id>
Devuelve los detalles completos de un restaurante.

GET /api/restaurantes/<id>/platos[?alergia=...]
Lista los platos del restaurante y marca aquellos que son seguros para la alergia seleccionada.

Reservas

POST /api/reservas
Crea una nueva reserva asignándole el estado inicial “Confirmada”.

GET /api/reservas/<id_cliente>
Recupera todas las reservas de un cliente (pasadas y futuras).

GET /api/restaurantes/<id>/reservas
Muestra las reservas que tiene un restaurante desde su área privada.

PUT /api/reservas/update/<id>
Permite actualizar una reserva.

DELETE /api/reservas/cancel/<id>
Cancela una reserva existente.

Facturas y Reseñas

GET /api/facturas/<id_cliente>
Lista las facturas del cliente. La interfaz muestra el botón “Valorar” si la factura no tiene valoración.

POST /api/reservas/<id_reserva>/factura
Genera una factura asociada a una reserva.

POST /api/resenas
Guarda la valoración y el tipo de visita en la última factura no valorada de ese cliente y restaurante.

POST /api/restaurantes/factura/crear
Desde el área del restaurante generamos una factura completa junto con sus comandas.

Alérgenos

GET /api/alergenos
Devuelve el catálogo completo de alérgenos, usado por la UI para poblar filtros.

Analítica (Área Restaurante)

GET /analytics/sin-valorar
Lista clientes con facturas pendientes de valoración.

GET /analytics/gasto-medio
Calcula el gasto medio por persona del restaurante.

GET /analytics/dia-mas-concurrido
Devuelve en texto el día con mayor volumen de reservas.

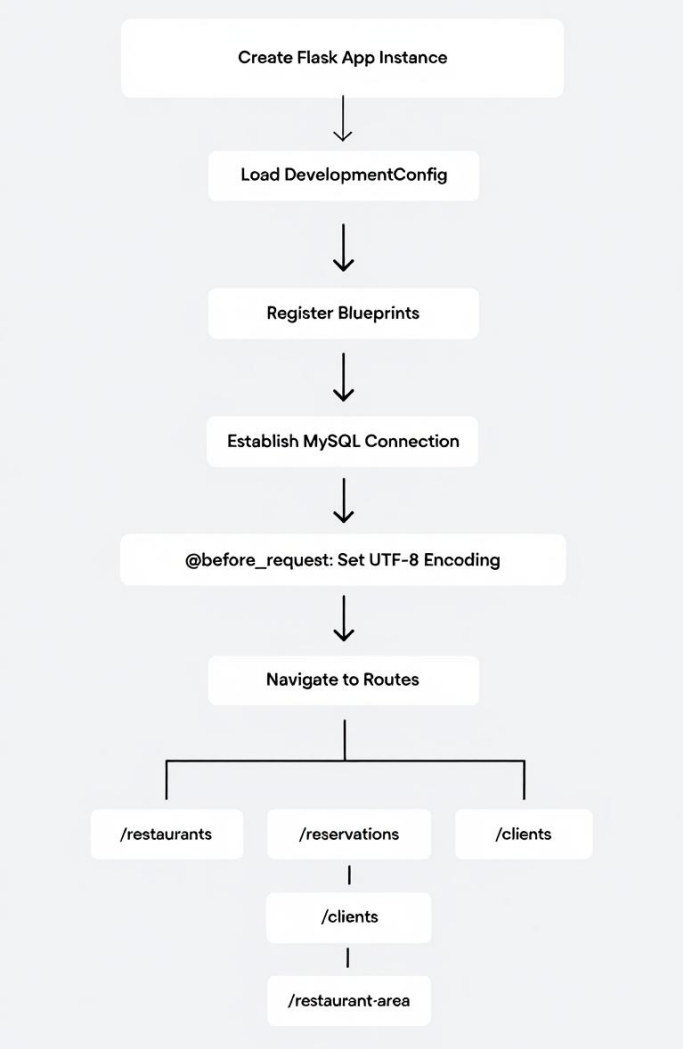
GET /analytics/grafico-dias
Envía un gráfico base64 con la distribución de reservas por día de la semana.

GET /analytics/top-platos
Obtiene los tres platos más vendidos según las comandas.

GET /analytics/grafico-precio-comparativo
Genera una distribución gaussiana del mercado y marca la posición del restaurante (media, percentil, línea de referencia).

4.2. Flujo Típico de Ejecución

Cuando iniciamos la aplicación:



Ejecuciones uso típicas:

1. Filtro por Alergias

La interfaz obtiene todos los alérgenos mediante **GET** /api/alergenos.

El usuario selecciona un alérgeno, que lanza **GET** /api/restaurantes?alergia=....

Al elegir un restaurante, consultamos **GET** /api/restaurantes/<id>/platos?alergia=....

En las vistas, los platos seguros se muestran en verde con el icono correspondiente.

2. Cliente: Reservas, Facturas y Valoración

GET /api/reservas/<id_cliente> muestra reservas pasadas y futuras.

Las pasadas no son editables y permiten solicitar factura o valorar (si existe).

Las futuras se pueden editar y validamos la fecha para evitar reservas hacia atrás.

Las facturas se obtienen con **GET** /api/facturas/<id_cliente>. Si una factura no está valorada, aparece el botón *Valorar*, que ejecuta **POST** /api/resenas.

3. Área Restaurante

El restaurante visualiza sus reservas mediante **GET** /api/restaurantes/<id>/reservas, con badges normalizados.

Puede generar facturas completas a través de **POST** /api/restaurantes/factura/crear.

Accede a todas las métricas: gasto medio, top platos, día con más reservas, clientes sin valorar y gráficos comparativos.

5. CONCLUSIONES

Esta práctica ha sido un reto interesante y muy enriquecedor. No solo me ha permitido mejorar mis habilidades técnicas en SQL, Python y JavaScript, sino también comprender cómo correlacionar datos para construir un caso realista. El objetivo era desarrollar una solución capaz de gestionar y relacionar información desde dos perspectivas distintas: la de los clientes y la del restaurante. Para ello, fue necesario trabajar con varias tablas que debían estar correctamente conectadas —incluyendo algunas utilizadas por ambos perfiles— y, además, ser lo suficientemente dinámicas para adaptarse a futuros cambios o ampliaciones.

Uno de los principales desafíos fue precisamente la correlación entre las tablas. La misma información se interpreta de manera diferente según se analice desde el punto de vista del cliente o del restaurante, lo que exigió diseñar una estructura de base de datos flexible y lógica, con relaciones claras entre las entidades.

Otro reto importante fue la gestión de datos con caracteres especiales, como tildes, al leerlos desde SQL y mostrarlos correctamente en la web. Aunque pueda parecer un detalle menor, resultó esencial para garantizar la calidad y la usabilidad de la aplicación.

La parte más sencilla de la práctica fue la creación de las tablas y la inserción inicial de los datos, ya que son tareas más directas y mecánicas. Por el contrario, lo más complejo fue diseñar la lógica de correlación, asegurar la dinámica de las relaciones y lograr que la integración de los datos en la web funcionara de manera correcta y eficiente.

En resumen, esta práctica nos ha permitido profundizar en el diseño de bases de datos relacionales, en el manejo de datos con diferentes codificaciones y en la importancia de establecer una lógica clara para unir distintos componentes de un sistema. Además, nos ha ayudado a enfrentar problemas reales que surgen en el desarrollo de aplicaciones, reforzando tanto mis habilidades técnicas como la capacidad de análisis y resolución de problemas.

6. ANEXOS

Para los códigos ver la carpeta proyecto_final