

AI2SQL

4.1 Elección de proyecto y nombre

La idea del proyecto se basa en solucionar un problema vigente en la mayoría de empresas grandes. Y es que, a día de hoy, la gestión y consulta de una base de datos necesita muchos intermediarios.

Pongamos un ejemplo:

Tenemos una empresa de envíos de productos que almacena información sobre los pedidos en una base de datos estructurada, y un alto ejecutivo de la empresa, sin conocimientos técnicos, requiere consultar un dato específico de su base de datos, digamos, necesita una lista de los productos que más compran las personas de jóvenes (entre 20 y 40 años) en madrid, y la cantidad de dinero que se gastan en estos productos.

Los pasos para solucionar el problema serán:

- Este ejecutivo tendrá que comunicarle esta petición al jefe o al intermediario de tecnología de la empresa.
- Este intermediario tendrá que buscar al gestor de la base de datos y solicitarle esta información.
- El gestor deberá analizar la petición, crear la consulta SQL, obtener la tabla y pasársela al intermediario.
- El intermediario deberá pasarle la información al alto ejecutivo.

En definitiva, es un camino largo en el que puede perderse información sobre los requisitos, y en el que se requiere la presencia y el tiempo de distintas personas cuyos sueldos no suelen ser nada despreciables.

Sin embargo, nuestro proyecto consigue solucionar este problema permitiendo al ejecutivo solicitar esta información directamente a una página web que, utilizando inteligencia artificial, consigue mostrarle la información requerida.

Nuestra primera misión será decidir si centrarnos en bases de datos SQL o noSQL. Para ello recopilaremos información con la ayuda de una herramienta de inteligencia artificial y haremos un análisis simple de ellos.

4.1.1 Copilot de VS Code

Copilot es una herramienta de inteligencia artificial creada por OpenAI que puede ayudar en tareas repetitivas de programación y mejorar la originalidad de ciertos parámetros.



Como podemos ver, en pocos meses se ha convertido en una de las extensiones más descargadas de VS code.



También está la versión Nightly, la cual incorpora las últimas novedades que OpenAI le va añadiendo, y puede llegar a ser más eficiente en según qué ámbitos.

Puede ser una herramienta muy valiosa para los programadores que buscan ahorrar tiempo y mejorar la eficiencia en tareas repetitivas, mientras también pueden aprovechar la creatividad y originalidad de la inteligencia artificial para mejorar la calidad de su trabajo.

Podemos encontrarnos ante el problema de necesitar algo de originalidad en la búsqueda; en este caso quería una lista de las bases de datos más usadas.

Si no tuviera Copilot habría necesitado buscar en google “Tipos de Bases de Datos más usadas” e investigar entre las distintas páginas hasta encontrar las más interesantes.

Copilot me lo ha generado automáticamente:

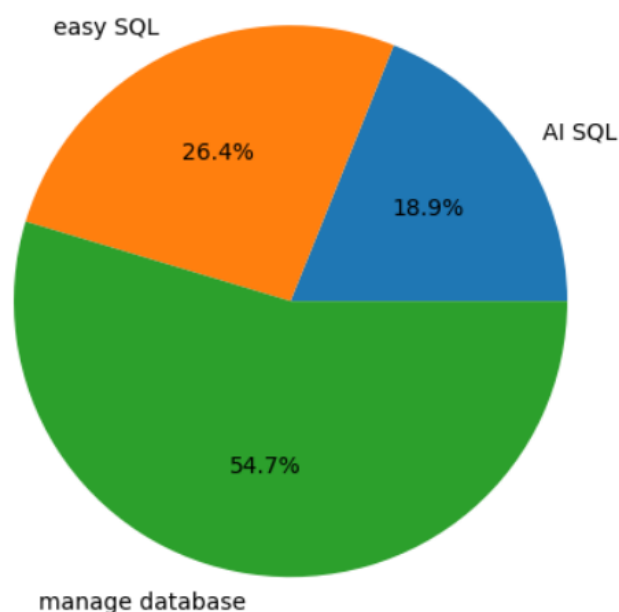
```
# Ask google for data about diverse databases
trend = pytrend.build_payload(kw_list=["SQL", "noSQL", "MongoDB", "Casandra", "CouchDB"])
```

Cualquier persona que haya usado librerías de representación como matplotlib o seaborn se ha encontrado con el problema de querer representar ciertos datos, pero necesitar usar una barbaridad de parámetros, y muchos hemos tenido que ponernos a buscar en internet (frecuentemente cosas que ya habíamos “aprendido”). Copilot nos soluciona estos problemas. Antiguamente tendría que haber buscado en google “pie in matplotlib”, y , dentro de la documentación de matplotlib, o en stackoverflow, busca qué parámetros hay que pasar a la función y después adecuar mis datos a estos parámetros.

Copilot me lo ha generado automáticamente, incluido adaptando mi dataframe:

```
# Pie chart
plt.pie(df.mean()[0:-1], labels = df.axes[1][0:-1], autopct='%1.1f%%')
```

Resultado:



También

O, incluso si
qué función
escribimos

estamos buscando y Copilot nos encuentra la función automáticamente:

```
# Interest over time
df_overtime = pytrend.interest_over_time()
```

no sabemos
usar,
lo que

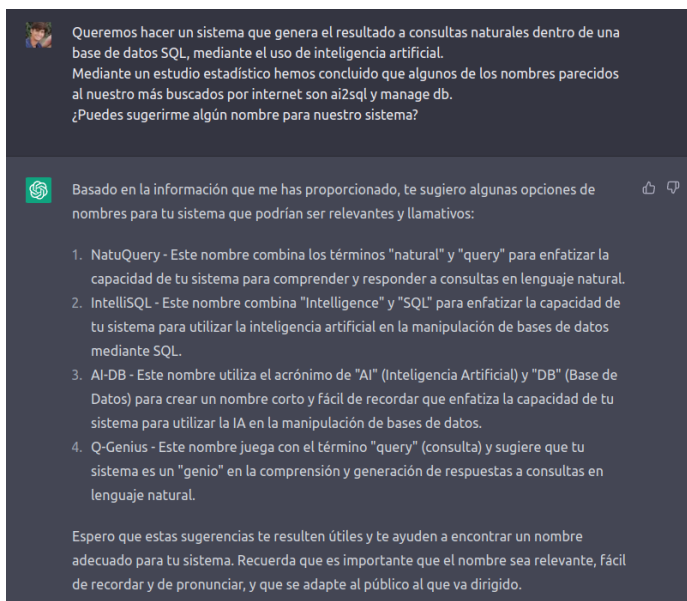
4.1.1 Elección de tipo de base de datos y nombre

Una vez obtenidos estos datos y convertidos en un formato csv, pasamos a analizarlos mediante una herramienta de Business Intelligence; en este caso, Tableau.

Estos datos generados con Copilot tienen gran utilidad en nuestro caso particular de AI2DB, pues el mismo nombre procede de un [análisis de ellos](#).

En este ámbito hay conclusiones a las que un humano puede llegar más fácilmente que una IA, como lo es ver que las bases de datos noSQL se centran en países con una infraestructura empresarial algo más débil, y esto puede implicar que sea más utilizado por Start-ups o por empresas que no tienen una base de datos demasiado compleja ni tienen el capital suficiente para invertir en una gestión algo más ordenada de ellos.

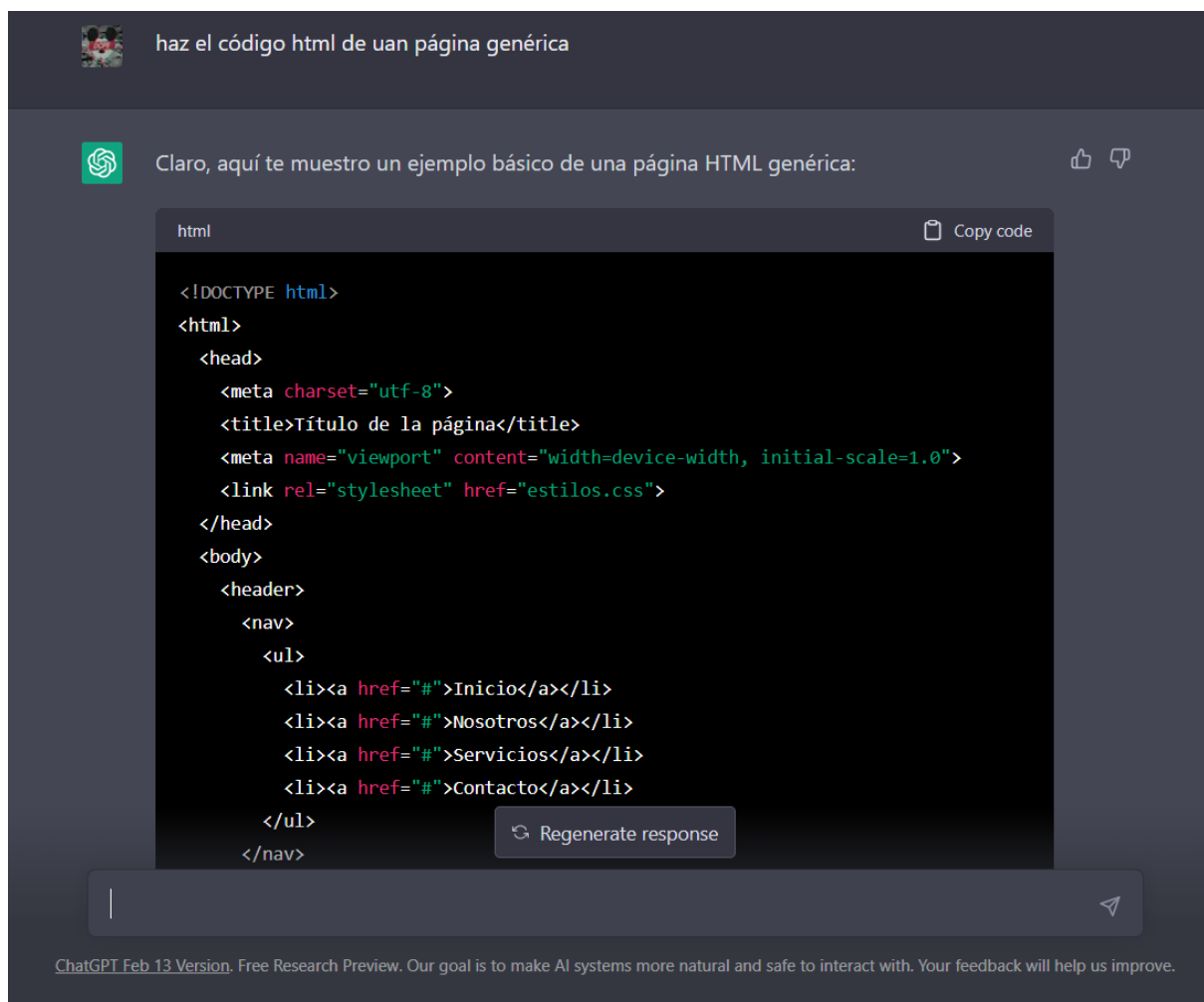
Esta es una de las razones por las que hemos preferido utilizar bases de datos SQL. Sin embargo, hemos hecho una posterior petición de sugerencias para nombre por chat gpt, y no estamos del todo contentos con la respuesta, por lo que a nivel grupal decidimos llamar “AI2DB” al proyecto.



4.2 Creación de Front-end

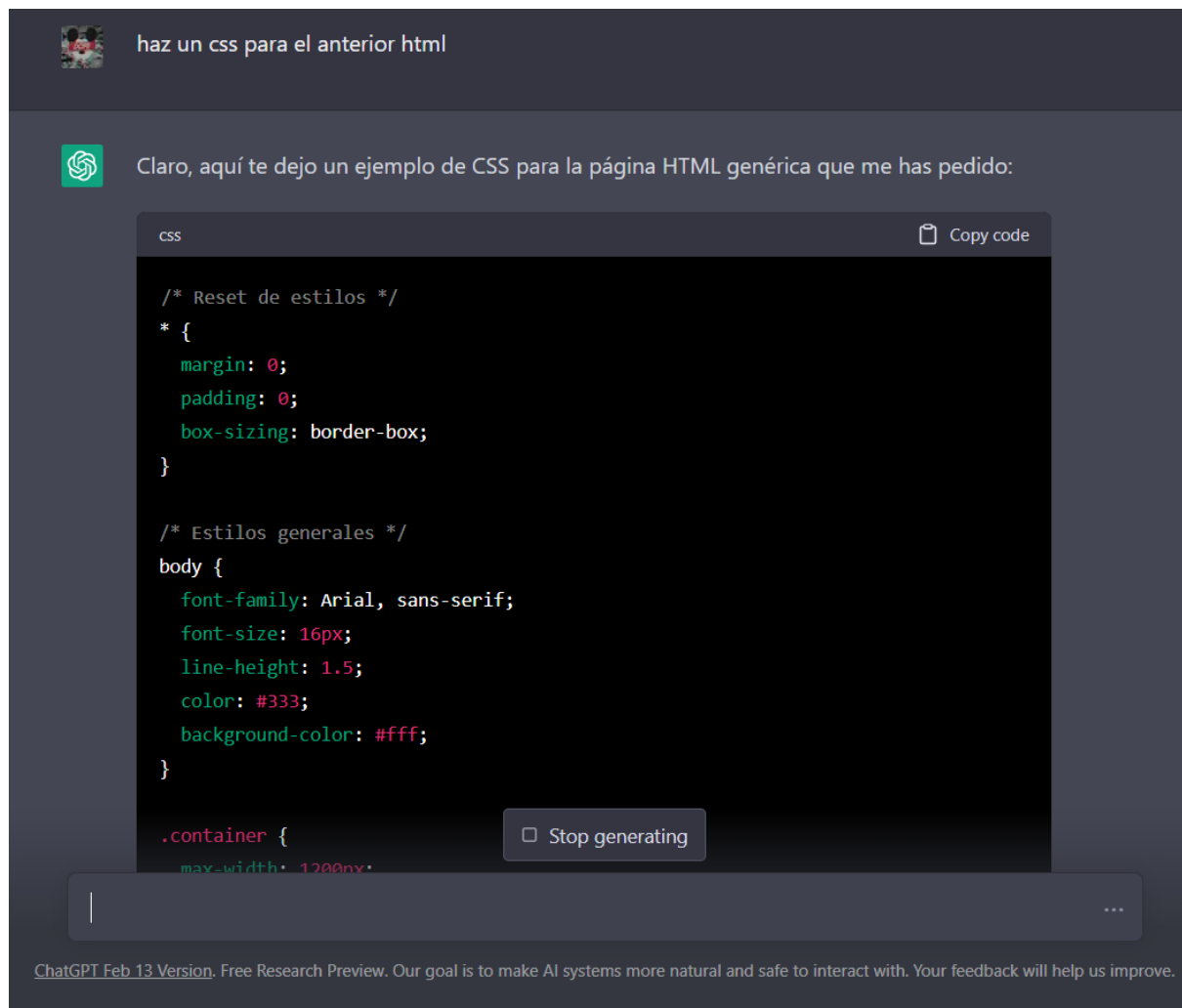
De cara la creación del Front-end la IA puede utilizarse de diversas maneras pero suele estar muy limitada.

En primer lugar a la hora de hacer el “.html” la IA es bastante útil debido a que tu puedes pedirle que genere un “.html” genérico y te ahorras escribir bastantes líneas de código que son iguales en todos los “.html”.



Esto es muy útil sobre todo en el momento en el que se empieza a crear la web ya que a lo mejor no tienen mucho dinero y no pueden contratar a alguien con estos conocimientos pero a la que consigan más dinero pueden ahorrarse algo del mismo ya que tienen algo como base.

En segundo lugar cuando hablamos del css también se le puede pedir a la iA que haga un “.css” básico com este:

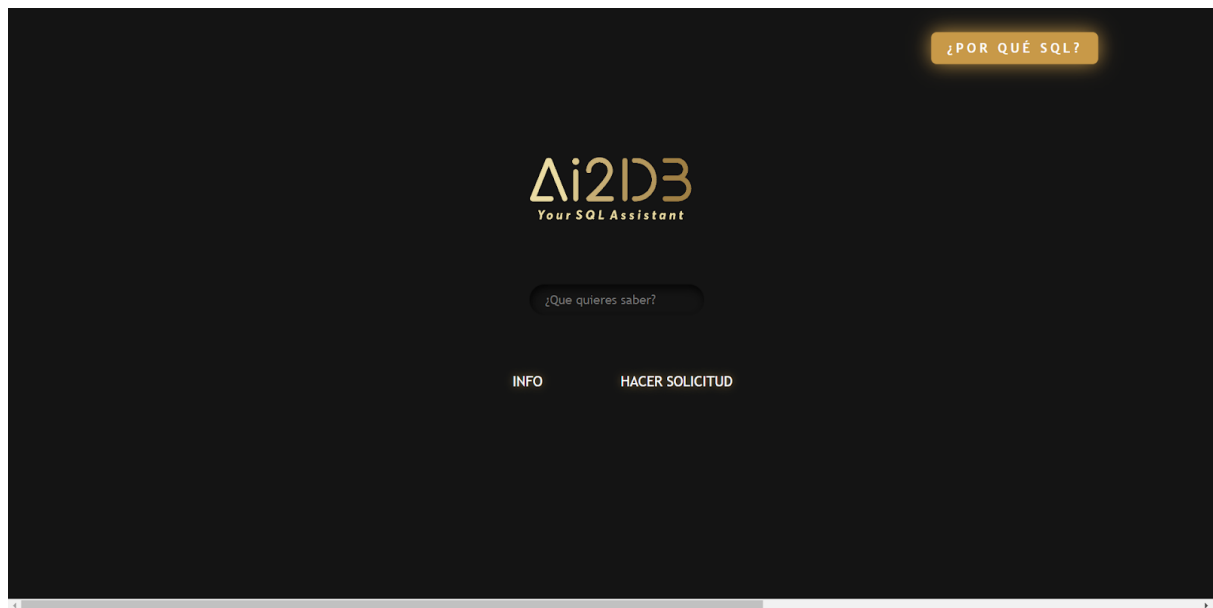


Que a la hora de empezar está bastante bien pero es muy simple para que no se cambie con el tiempo.

Se ha hablado de las ventajas que tiene la IA de cara al Front-end pero no todo es tan bonito ya que también tiene grandes desventajas sobre todo en el apartado estético. Esto es debido a que normalmente el css de una página es muy subjetivo y tiene que ver mucho con la opinión de cada persona y al ser una IA no tiene opiniones.

Además cuando quieres implementar cambios en el css utilizando una IA hay varios problemas. Por ejemplo, encuentras el estilo de un botón por internet y quieres implementarlo a tu css si le pides a la IA que lo implemente pueden ocurrir dos cosas, que lo implemente pero con fallos por lo que no tiene un correcto funcionamiento o

que te explique cómo tienes que hacerlo pero no te da el código y aquí lo que estamos buscando es que nos facilite el trabajo de de mejor forma posible.



Esta es la página de este trabajo. Esta página es “responsive” es decir se puede abrir en diferentes tipos de dispositivos y este apartado del css ha sido llevado a cabo por Chat GPT

```
Luego, se pueden usar las media queries para ajustar el estilo en diferentes tamaños de
pantalla. Por ejemplo:

css
Copy code

@media only screen and (max-width: 768px) {
  /* Estilos para pantallas menores o iguales a 768px */
  #navdiv{
    width: 100%;
    margin-left: 0%;
    justify-content: center;
  }
  nav ul li {
    float: none;
  }
  /* Estilos para el input de búsqueda */
  #maindiv{
    width: 80%;
  }
  span:first-child{
    width: 15%;
  }
  span:last-child{
    width: 15%;
  }
  /* Estilos para la sección de botones */
  section{
    width: 100%;
  }
}
```

Se le proporciona a Chat GPT el css al principio de la creación de la página y lo convierte en “responsive” pero ocurren varios problemas ya que faltaban trozos de código o no estaba bien cuadrado con la página por lo que una persona tuvo que revisarlo. Si se quiere que se haga un trabajo rápido es una buena opción pero si se busca que sea un trabajo bien logrado no es el mejor camino para llegar a conseguirlo.

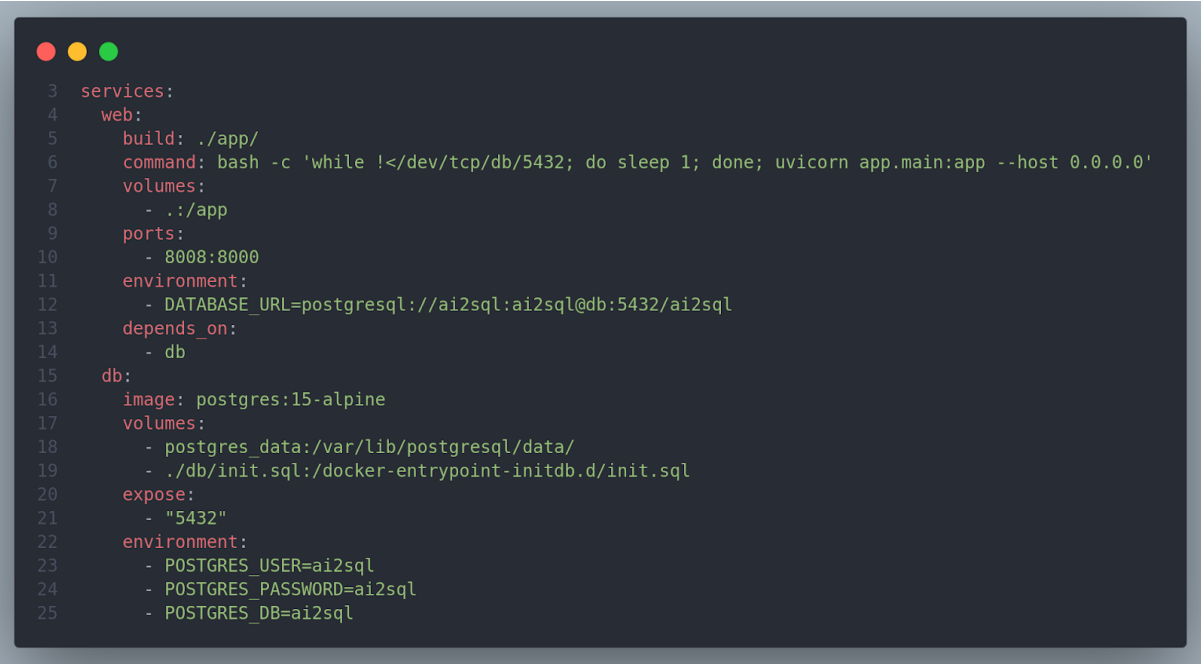
El css y html de la página que podemos ver en en una de las anteriores imágenes ha sido elaborado por este grupo con ayuda de chat GPT pero donde se a usado mayormente ha sido en la creación de un html y css básico además de3 para el responsive todo los demás no se ha usado IA o se ha usado mínimamente . La IA se puede utilizar para facilitar el trabajo pero en un apartado tan subjetivo no es tan útil.

(El Front-end se ha centrado en la utilización de la IA “Chat GPT sobre todo lo cual no quiere decir que no existan otras IA’s que sean mejores para la creación de html y css ”)

4.3 Creación de Back-end


4.3.1 Infraestructura

La creación del contenedor se hizo con un archivo dockercompose.yml que automatiza la creación del sistema, el cual está compuesto de dos servicios principales; un entorno python en el que se ejecuta la aplicación de FastAPI y el otro servicio es una base de datos PostgreSQL.



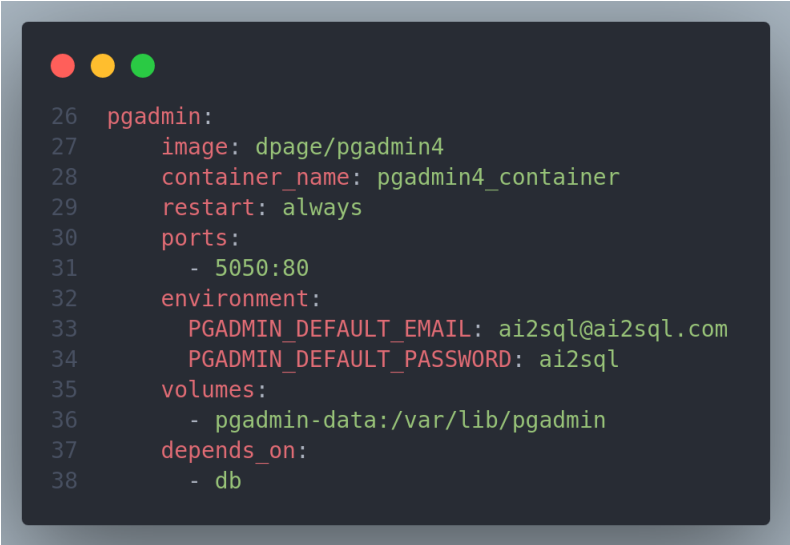
```
3 services:
4   web:
5     build: ./app/
6     command: bash -c 'while !</dev/tcp/db/5432; do sleep 1; done; uvicorn app.main:app --host 0.0.0.0'
7     volumes:
8       - ./app
9     ports:
10      - 8008:8000
11     environment:
12       - DATABASE_URL=postgresql://ai2sql:ai2sql@db:5432/ai2sql
13     depends_on:
14       - db
15   db:
16     image: postgres:15-alpine
17     volumes:
18       - postgres_data:/var/lib/postgresql/data/
19       - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
20     expose:
21       - "5432"
22     environment:
23       - POSTGRES_USER=ai2sql
24       - POSTGRES_PASSWORD=ai2sql
25       - POSTGRES_DB=ai2sql
```

Al iniciarse el servidor se instalan las librerías necesarias para la base de datos, y mediante un fichero requirements.txt se importan las librerías de python necesarias:



```
1  asyncpg==0.27.0
2  fastapi==0.89.1
3  ormar==0.12.1
4  psycopg2-binary==2.9.5
5  uvicorn==0.20.0
6  openai
7  jinja2
8  python-multipart
9  python-dotenv
```

Además incluimos, en el `docker-compose.yml`, el servicio “pgadmin”, el cual nos ha permitido realizar depuración de estos otros servicios para comprobar que el funcionamiento era el correcto y esperado:



```
26  pgadmin:
27    image: dpage/pgadmin4
28    container_name: pgadmin4_container
29    restart: always
30    ports:
31      - 5050:80
32    environment:
33      PGADMIN_DEFAULT_EMAIL: ai2sql@ai2sql.com
34      PGADMIN_DEFAULT_PASSWORD: ai2sql
35    volumes:
36      - pgadmin-data:/var/lib/pgadmin
37    depends_on:
38      - db
```

4.3.2 Creación de base de datos ficticia

Para poder hacer consultas en nuestra prueba quisimos tener algunos datos aleatorios sobre una empresa que tenga ventas online, y necesite tablas para los clientes, ciudades, pedidos y productos.

Por ello aplicamos las técnicas vistas en el análisis de prompts para crear nuestros datos ficticios de forma aleatoria mediante los prompts correctos, y los incorporamos a los ficheros de inicialización.

```

1 CREATE DATABASE nombre_de_tu_base_de_datos;
2
3 \c nombre_de_tu_base_de_datos;
4
5 CREATE TABLE ciudad (
6     id SERIAL PRIMARY KEY,
7     nombre VARCHAR(100) NOT NULL,
8     paradas_de_metro INTEGER NOT NULL,
9     país VARCHAR(100) NOT NULL
10 );
11
12 CREATE TABLE persona (
13     dni VARCHAR(10) PRIMARY KEY,
14     nombre VARCHAR(100) NOT NULL,
15     fecha_de_nacimiento DATE NOT NULL,
16     ciudad_nacimiento INTEGER NOT NULL,
17     FOREIGN KEY (ciudad_nacimiento) REFERENCES ciudad(id)
18 );
19
20 CREATE TABLE productos (
21     id SERIAL PRIMARY KEY,
22     nombre VARCHAR(100) NOT NULL,
23     pvp NUMERIC(10,2) NOT NULL,
24     categoria VARCHAR(100) NOT NULL
25 );
26
27 CREATE TABLE pedidos (
28     id_persona VARCHAR(10) NOT NULL,
29     id_producto INTEGER NOT NULL,
30     precio NUMERIC(10,2) NOT NULL,
31     FOREIGN KEY (id_persona) REFERENCES persona(dni),
32     FOREIGN KEY (id_producto) REFERENCES productos(id),
33     PRIMARY KEY (id_persona, id_producto)
34 );

```

Ejemplos varios de la construcción de las tablas creadas:

```

36 INSERT INTO ciudad (nombre, paradas_de_metro, país)
37 VALUES ('Madrid', 12, 'España'),
38         ('Barcelona', 8, 'España'),
39         ('París', 14, 'Francia'),
40         ('Londres', 10, 'Reino Unido'),
41         ('Nueva York', 24, 'EE. UU.'),
42         ('Tokio', 9, 'Japón'),
43         ('Roma', 11, 'Italia'),
44         ('Sídney', 6, 'Australia'),
45         ('Buenos Aires', 8, 'Argentina'),
46         ('Moscó', 9, 'Rusia'),
47         ('Estocolmo', 6, 'Suecia'),
48         ('México D.F.', 13, 'México'),
49         ('Lisboa', 4, 'Portugal'),
50         ('Berlín', 12, 'Alemania'),
51         ('Shanghái', 14, 'China'),
52         ('Toronto', 5, 'Canadá'),
53         ('Ámsterdam', 9, 'Países Bajos'),
54         ('Dubái', 2, 'Emiratos Árabes Unidos'),
55         ('Sao Paulo', 10, 'Brasil'),
56         ('Seúl', 10, 'Corea del Sur');

```



```
58 INSERT INTO productos (nombre, pvp, categoria)
59 VALUES ('Camiseta', 12.99, 'Ropa'),
60         ('Pantalón', 29.99, 'Ropa'),
61         ('Calcetines', 5.99, 'Ropa'),
62         ('Zapatillas', 59.99, 'Calzado'),
63         ('Botas', 79.99, 'Calzado'),
64         ('Gafas de sol', 19.99, 'Complementos'),
65         ('Bolso', 49.99, 'Complementos'),
66         ('Reloj', 99.99, 'Complementos'),
67         ('Móvil', 499.99, 'Electrónica'),
68         ('Ordenador portátil', 899.99, 'Electrónica'),
69         ('Televisor', 799.99, 'Electrónica'),
70         ('Cámara de fotos', 299.99, 'Electrónica'),
71         ('Videojuego', 59.99, 'Ocio'),
72         ('Libro', 14.99, 'Ocio'),
73         ('Película en DVD', 9.99, 'Ocio'),
74         ('Disco de música', 12.99, 'Ocio'),
75         ('Perfume', 49.99, 'Cosmética'),
76         ('Crema hidratante', 19.99, 'Cosmética'),
77         ('Champú', 7.99, 'Cosmética'),
78         ('Maquillaje', 29.99, 'Cosmética');
```

4.3.3 FastAPI

En FastAPI hemos seguido los pasos predeterminados para montar una app que pueda recibir peticiones.

En particular, nos hemos enfocado en la función get:

```
22  ## CONTROLLERS
23  @app.get("/", response_class=HTMLResponse)
24  async def main( request: Request, prompt: str = ""):
25      response: HTMLResponse
26      if prompt == "":
27          response = templates.TemplateResponse("index.html", {"request": request})
28      else:
29          headings, data = get_table_from_prompt(prompt=prompt, traslator=traslator)
30          response = templates.TemplateResponse("index.html",{
31              "request": request,
32              "prompt": prompt,
33              "headings": headings,
34              "data": data
35          })
36      return response
```

Como vemos, lo único que se sale de lo normal dentro de esta función es la función “get_table_from_prompt”.

Veamos cómo está construida:

```
3  def get_table_from_prompt(prompt: str, traslator: SQL_Traslator) -> list:
4      ## 1. Use the traslator for get the sql query
5      sql: str = traslator.convert_to_sql(text_for_query=prompt)
6
7      ## 2. Use the query function to return the table
8      return query(sql)
```

De nuevo, lo más fuera de lo común es este método del objeto Traslador, construido por nuestro equipo, y es precisamente éste el que integra funcionalidades de Inteligencia Artificial.

```
19 def convert_to_sql(self, text_for_query: str):
20     completion = openai.Completion.create(engine="text-davinci-003",
21                                           prompt="Conviérteme a un SELECT en sql la siguiente petición:" +
22                                                 text_for_query +
23                                                 "\n, Dentro de una base de datos creada con el siguiente script: " +
24                                                 self._creation_db_script,
25                                           max_tokens=100)
26     return completion.choices[0].text
```

Es una simple llamada a la API con unos parámetros bastante estándar. Lo más importante es el prompt que le hemos proporcionado. Y es que este prompt es precisamente un prompt estudiado en la parte teórica, y que consigue proporcionar a ChatGPT con la base necesaria para respondernos únicamente con la sentencia SQL que posteriormente se utilizará.