

# FastAPI 回傳資料格式



我在寫fastapi，寫好的api要給react call，把值存到 mariadb，為什麼我看mariadb存的值是{message: "hello"}，在swagger的/doc看到的reponse是 `"/"message/":"/"hello/"`



你遇到的問題可能是 **FastAPI 回傳的資料格式**或是資料在送進 **MariaDB** 時沒有正確解析造成的。以下是針對你描述的情況做的分析與解法：

## 問題描述拆解：

### 1. **FastAPI 的 Swagger** `/docs` 顯示回傳是：

```
"/"message/":"/"hello/"
```

這看起來像是字串被多次序列化（例如 JSON -> string -> 再變成字串形式的 JSON）。

### 2. **MariaDB** 裡存的是：

```
{message: "hello"}
```

如果它是字串，代表資料被當作一整塊 JSON 字串塞進資料庫了，而不是個別欄位。

## 🧠 常見原因：

### ✅ 原因一：你把整個 JSON 當作一個字串存在資料庫

範例程式碼：

```
@app.post("/save")
async def save_data(data: dict):
    cursor.execute("INSERT INTO my_table (payload) VALUES (%s)",
        (json.dumps(data),))
```

這會把整個 `{"message": "hello"}` 存成字串放到一個欄位裡（例如 `payload`），導致資料庫看到的就是整串 JSON。

### ✅ 原因二：你在回傳 response 時，把 JSON 當作字串回傳

範例錯誤寫法：

```
return JSONResponse(content=json.dumps({"message": "hello"}))
```

這樣會把 JSON 物件先變成字串 `{"message": "hello"}`，然後又被 FastAPI 包一次，就變成：

```
"\"{\\\\"message\\\\"": \\\\"hello\\"}\\\""
```

在 Swagger `/docs` 看起來就像你說的那樣亂碼。

### ✅ 正確寫法示範

### ✅ FastAPI 正確的 Response 回傳：

```

from fastapi import FastAPI
from pydantic import BaseModel
import mariadb

app = FastAPI()

class Message(BaseModel):
    message: str

# 假設你有一個資料表 `messages` 欄位為 `message` (VARCHAR)

@app.post("/save")
async def save_message(msg: Message):
    # 儲存到資料庫
    conn = mariadb.connect(user="root", password="pass", database="test")
    cursor = conn.cursor()
    cursor.execute("INSERT INTO messages (message) VALUES (?)", (msg.message,))
    conn.commit()
    conn.close()

    # 正確回傳 JSON
    return {"status": "success", "message": msg.message}

```

🔍 `/docs` 頁面看到的會是：

```

{
  "status": "success",
  "message": "hello"
}

```

✅ **React** 呼叫這個 **API** 的方式建議：

```

fetch("http://localhost:8000/save", {
  method: "POST",
  headers: {

```

```
"Content-Type": "application/json"
},
body: JSON.stringify({ message: "hello" })
})
```

✅ 資料庫內容會是：

id	message
1	hello

## 💡 小結論

問題原因	解法
回傳內容被重複 JSON 序列化	不要用 <code>json.dumps()</code> 再傳給 <code>JSONResponse</code>
MariaDB 儲存 JSON 字串而非欄位值	將字典中的值取出來存欄位

如果你能貼出你目前的 FastAPI 處理與寫入 MariaDB 的程式碼，我可以幫你更具體對症下藥。需要我幫你檢查嗎？



我想要理解json資料存進mariadb的用法(資料格式 LONGTEXT)，為了做之後跟react串接的實驗，我對fastapi的

寫法與mariadb的用法不熟，你能教我嗎，或是你能給更簡單的範例：



已更新儲存的記憶

當然可以！以下是一個**簡單又完整的範例**，教你如何使用 FastAPI 接收 JSON 並存進 MariaDB 的 `LONGTEXT` 欄位中，適合做與 React 串接的實驗：

## 資料庫準備 (MariaDB)

假設你要儲存的資料像這樣：

```
{
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"]
}
```

那你可以在 MariaDB 建一張表：

```
CREATE TABLE json_data (
  id INT AUTO_INCREMENT PRIMARY KEY,
  content LONGTEXT
);
```

## FastAPI 範例

## ✅ 安裝必要套件

```
pip install fastapi uvicorn mariadb pydantic
```

## ✅ `main.py` 完整程式

```
from fastapi import FastAPI, Request
import mariadb
import json

app = FastAPI()

# 資料庫連線設定
def get_connection():
    return mariadb.connect(
        user="your_user",
        password="your_password",
        host="127.0.0.1",
        port=3306,
        database="your_database"
    )

# 接收 JSON 並存進資料庫
@app.post("/store")
async def store_json(request: Request):
    data = await request.json() # 取得 JSON 資料

    conn = get_connection()
    cursor = conn.cursor()

    # 存成 LONGTEXT, 使用 json.dumps() 轉成字串
    cursor.execute("INSERT INTO json_data (content) VALUES (?)",
        (json.dumps(data),))

    conn.commit()
    cursor.close()
    conn.close()
```

```
return {"status": "ok", "stored_data": data}
```

## 用 Swagger `/docs` 測試

啟動服務：

```
uvicorn main:app --reload
```

開啟瀏覽器到 <http://localhost:8000/docs>，你會看到 `/store` API。

你可以直接貼上 JSON：

```
{  
  "username": "佳恩",  
  "age": 28,  
  "hobbies": ["coding", "gaming"]  
}
```

送出後，這份資料會被轉為字串存進資料庫 `content` 欄位中。

## 資料庫裡看到的會是：

id	content
----	---------

1	{"username": "佳恩", "age": 28, "hobbies": ["coding", "gaming"]}
---	--

## □ React 未來串接方式

未來你可以用 React 傳送資料這樣：

```
fetch("http://localhost:8000/store", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    username: "佳恩",
    age: 28,
    hobbies: ["coding", "gaming"]
  })
})
```

如果你想要我幫你改成特定的欄位 schema（例如不是存整包 JSON，而是 username、age 分開存），我也可以調整給你！

需要我幫你做「讀取資料」的 API 嗎？還是你現在想先把資料成功塞進 MariaDB 就好？



我要存的格式是

```
{
  "username": "佳恩",
  " age ": 28,
  " hobbies ": ["coding", "gaming"],
  " content":
  {
    message: "hello",
    student: "allen"
  }
}, username、username、hobbies、content 都是mariadb
schema的欄位
```





了解！你想要的資料結構如下：

```
{
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],
  "content": {
    "message": "hello",
    "student": "allen"
  }
}
```

而你希望 MariaDB 中的欄位如下：

欄位名	資料型別	說明
`username`	`VARCHAR`	使用者名稱
`age`	`INT`	年齡
`hobbies`	`LONGTEXT`	陣列轉成 JSON 存文字
`content`	`LONGTEXT`	巢狀 JSON 物件，轉文字存入

## ✅ 第一步：MariaDB 建表語法

```
CREATE TABLE user_data (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(100),
```



```
age INT,  
hobbies LONGTEXT,  
content LONGTEXT  
);
```

## ✅ 第二步：FastAPI 實作

### 🔧 安裝套件（如果還沒裝）

```
pip install fastapi uvicorn mariadb pydantic
```

### 🐍 `main.py` 完整範例

```
from fastapi import FastAPI  
from pydantic import BaseModel  
from typing import List, Dict, Any  
import mariadb  
import json  
  
app = FastAPI()  
  
# 定義資料格式  
class Content(BaseModel):  
    message: str  
    student: str  
  
class UserData(BaseModel):  
    username: str  
    age: int  
    hobbies: List[str]  
    content: Content
```

# 連線設定

```
def get_connection():  
    return mariadb.connect(  
        user="your_user",  
        password="your_password",  
        host="127.0.0.1",  
        port=3306,  
        database="your_database"  
    )
```

# API 接收 JSON 並存入資料庫

```
@app.post("/store")
```

```
async def store_user(data: UserData):
```

```
    conn = get_connection()
```

```
    cursor = conn.cursor()
```

# hobbies 和 content 是複雜資料，轉成 JSON 字串儲存

```
    cursor.execute(  
        "INSERT INTO user_data (username, age, hobbies, content) VALUES (?, ?,  
        ?, ?)",  
        (  
            data.username,  
            data.age,  
            json.dumps(data.hobbies),  
            json.dumps(data.content.dict()) # content 是 Pydantic Model, 要先轉  
成 dict  
        )  
    )
```

```
    conn.commit()
```

```
    cursor.close()
```

```
    conn.close()
```

```
    return {"status": "ok", "stored": data}
```

### ✅ 第三步：用 ` /docs ` 測試

啟動 FastAPI：

```
uvicorn main:app --reload
```

然後開瀏覽器打開 <http://localhost:8000/docs>，測試 `/store` API。

貼入這段資料測試：

```
{
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],
  "content": {
    "message": "hello",
    "student": "allen"
  }
}
```

## ✅ 結果：MariaDB 會儲存成這樣

id	username	age	hobbies	content
1	佳恩	28	<code>["coding", "gaming"]</code>	<code>{"message": "hello", "student": "allen"}</code>

是否需要我幫你做一個 `GET /user/{id}` 的 API，把資料從 MariaDB 讀出、還原成原本的 JSON 結構？還是你目前想先專注在寫入就好？



我要用sessionmaker不要async



了解！你想使用 **同步版（非 async）FastAPI** 且用 ``sessionmaker`` + **SQLAlchemy ORM** 存資料進 MariaDB，欄位分別為 ``username``、``age``、``hobbies``、``content``，其中 ``hobbies`` 和 ``content`` 是要用 ``LONGTEXT`` 存 JSON。

我這裡提供一個**簡單完整範例**，不使用 ``async``，使用同步的 ``FastAPI`` + ``SQLAlchemy``（``sessionmaker``），幫你打好實驗的基礎。

## ✓ 1. 安裝必要套件

```
pip install fastapi uvicorn sqlalchemy mariadb pydantic
```

## ✓ 2. 建立資料庫表（MariaDB）

```
CREATE TABLE user_data (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    username VARCHAR(100),  
    age INT,  
    hobbies LONGTEXT,  
    content LONGTEXT  
);
```

### ✓ 3. `main.py` 同步 + sessionmaker 實作

```
from fastapi import FastAPI, Depends
from pydantic import BaseModel
from typing import List
import json

from sqlalchemy import create_engine, Column, Integer, String, Text
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session

# MariaDB 連線設定 (請改成你自己的帳密/資料庫)
DB_URL =
"mariadb+mariadbconnector://user:password@localhost:3306/your_database"

engine = create_engine(DB_URL, echo=True)
SessionLocal = sessionmaker(bind=engine)

Base = declarative_base()

app = FastAPI()

# 資料表對應的 ORM Model
class UserData(Base):
    __tablename__ = "user_data"

    id = Column(Integer, primary_key=True, index=True)
    username = Column(String(100))
    age = Column(Integer)
    hobbies = Column(Text) # LONGTEXT
    content = Column(Text) # LONGTEXT

# 如果資料表還沒建立, 這行會建立
Base.metadata.create_all(bind=engine)

# Pydantic Schema
class Content(BaseModel):
    message: str
    student: str

class UserInput(BaseModel):
```

```

username: str
age: int
hobbies: List[str]
content: Content

# Dependency: 取得 DB Session
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# 寫入資料的 API
@app.post("/store")
def store_user(user: UserInput, db: Session = Depends(get_db)):
    db_user = UserData(
        username=user.username,
        age=user.age,
        hobbies=json.dumps(user.hobbies),
        content=json.dumps(user.content.dict())
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)

    return {"status": "ok", "id": db_user.id}

```

## 4. 測試用 JSON (Swagger `/docs`)

打開 <http://localhost:8000/docs>, 傳入：

```

{
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],

```

```
"content": {  
  "message": "hello",  
  "student": "allen"  
}
```

## ✅ 資料庫內容長這樣：

id	username	age	hobbies	content
1	佳恩	28	`["coding", "gaming"]`	`{"message": "hello", "student": "allen"}`

需要我再補充：

- 查詢資料 (`GET /user/{id}`) ?
- 修改資料？
- 或搭配 React 使用方式？

都可以幫你擴充！



我要DeclarativeBase改寫(使用python typing 與 SQLAlchemy 2.0寫法)





太好了！你想使用 **SQLAlchemy 2.0 的 Declarative Base + typing 寫法**，這是最新推薦方式，語法更現代、型別更清楚，特別適合搭配 FastAPI 使用。

## ✅ 最終目標表結構（你要存的欄位）

```
{
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],
  "content": {
    "message": "hello",
    "student": "allen"
  }
}
```

對應 MariaDB schema：

欄位名	類型	說明
id	INT	主鍵，自動增加
username	VARCHAR	使用者名稱
age	INT	年齡
hobbies	LONGTEXT	JSON 字串化後存入
content	LONGTEXT	巢狀 JSON 字串儲存

## ✅ 完整範例（SQLAlchemy 2.0 typing + FastAPI + sessionmaker）

## 安裝套件 (SQLAlchemy 2.0 搭配 `mariadb`)

```
pip install fastapi uvicorn sqlalchemy mariadb pydantic
```

## `main.py` 同步版程式碼 (建議存檔使用)

```
from fastapi import FastAPI, Depends
from pydantic import BaseModel
from typing import List
import json

from sqlalchemy import create_engine, String, Integer, Text
from sqlalchemy.orm import sessionmaker, Mapped, mapped_column,
DeclarativeBase, Session

# ✳ 資料庫連線字串 (請依照你的帳密與資料庫改)
DATABASE_URL =
"mariadb+mariadbconnector://user:password@localhost:3306/your_database"

engine = create_engine(DATABASE_URL, echo=True)
SessionLocal = sessionmaker(bind=engine)

# ✅ SQLAlchemy 2.0 Base 類別
class Base(DeclarativeBase):
    pass

# ✅ ORM Model 使用 typing 與 mapped_column
class UserData(Base):
    __tablename__ = "user_data"

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    username: Mapped[str] = mapped_column(String(100))
    age: Mapped[int] = mapped_column(Integer)
    hobbies: Mapped[str] = mapped_column(Text) # 儲存 JSON string
    content: Mapped[str] = mapped_column(Text) # 儲存 JSON string
```

```

# 自動建表 (第一次跑會用到)
Base.metadata.create_all(engine)

# ✅ FastAPI App
app = FastAPI()

# ✅ Pydantic 輸入格式 (前端傳入 JSON)
class Content(BaseModel):
    message: str
    student: str

class UserInput(BaseModel):
    username: str
    age: int
    hobbies: List[str]
    content: Content

# ✅ Dependency : 資料庫 session
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# ✅ 儲存資料 API
@app.post("/store")
def store_user(user: UserInput, db: Session = Depends(get_db)):
    db_user = UserData(
        username=user.username,
        age=user.age,
        hobbies=json.dumps(user.hobbies),
        content=json.dumps(user.content.dict())
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)

    return {"status": "ok", "id": db_user.id}

```

## 啟動 FastAPI

```
uvicorn main:app --reload
```

打開瀏覽器進入 <http://localhost:8000/docs>，使用 `/store` 測試 API。

## 資料庫會儲存內容如下：

id	username	age	hobbies	content
1	佳恩	28	<code>["coding", "gaming"]</code>	<code>{"message": "hello", "student": "allen"}</code>

如果你之後想實作：

- `/list` 讀出所有資料
- `/get/{id}` 取得一筆資料並還原成原本 JSON 格式
- `/update/{id}` 更新某筆資料

我也可以幫你一步步加上。

需要我幫你加「讀出資料」的 API 嗎？還是你現在要先測試寫入？



只確定content是一筆json資料，不確定會有哪一些key



了解！你想把 `content` 儲存為一筆 **結構不固定的 JSON 物件**，例如：

```
"content": {  
  "message": "hello",  
  "student": "allen"  
}
```

但有時候可能是：

```
"content": {  
  "ip": "1.2.3.4",  
  "user_agent": "Mozilla/5.0",  
  "details": {"region": "Asia"}  
}
```

重點是：

“ `content` 是一筆不定欄位的 JSON

 你只要把它整包存進資料庫即可 (例如 MariaDB 的 `LONGTEXT` 欄位)”

 **該怎麼改？**

 **Pydantic 寫法**

你可以把 `content` 設成 `Dict[str, Any]`：

```
from typing import Dict, Any

class UserInput(BaseModel):
    username: str
    age: int
    hobbies: List[str]
    content: Dict[str, Any] # ✅ 可接收任意結構的 JSON
```

## ✅ 實際測試用 JSON

這些都是合法的 `content`：

```
{
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],
  "content": {
    "message": "hello",
    "student": "allen"
  }
}
```

或：

```
{
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],
  "content": {
    "ip": "1.2.3.4",
    "region": "TW",
    "browser": {
      "name": "Chrome",
      "version": "114.0"
    }
  }
}
```

```
}  
}  
}
```

## ✅ ORM Model 不需要改變

因為你已經用：

```
content: Mapped[str] = mapped_column(Text)
```

只要在存資料時繼續用 `json.dumps()` 序列化即可：

```
db_user = UserData(  
    username=user.username,  
    age=user.age,  
    hobbies=json.dumps(user.hobbies),  
    content=json.dumps(user.content) # ✅ content 是 dict, 可以直接轉 JSON  
)
```

## ✅ 小結論

類別	內容
Pydantic	<code>content: Dict[str, Any]</code>
DB 欄位	<code>content</code> 用 <code>Text</code> or <code>LONGTEXT</code>
存入 DB	<code>json.dumps(content)</code> 序列化後存入
讀出 DB	可用 <code>json.loads(content)</code> 還原成 dict

需要我補一個 `/get/{id}` API 幫你從資料庫撈出並還原 `content` 為 JSON 嗎？我可以幫你加讀出和還原的部分。



我想要理解json資料存進mariadb的用法(資料格式 LONGTEXT)，為了做之後跟react串接的實驗，我對fastapi的寫法與json存入mariadb的用法不熟，這樣寫會有甚麼問題嗎？，你能教我嗎，或是你能給更簡單的範例：

```
class UserData(Base):
    __tablename__ = "user_data"

    id: Mapped[int] = mapped_column(primary_key=True,
    autoincrement=True)
    username: Mapped[str] = mapped_column(String(100))
    age: Mapped[int] = mapped_column(Integer)
    hobbies: Mapped[str] = mapped_column(Text) # 儲存 JSON string
    content: Mapped[str] = mapped_column(LONGTEXT)
    # 儲存 JSON string

class UserInput(BaseModel):
```



```
username: str
age: int
hobbies: List[str]
content: Any
```



你目前的寫法其實**基本上是正確的**，能成功把 JSON 資料（用 `json.dumps()` 轉成字串）存入 MariaDB 的 `LONGTEXT` 欄位中。

我來幫你一一拆解分析，並補上一個**簡化後最小可執行範例**幫你快速上手，讓你可以順利串接 React 🚀

## ✅ 你的寫法優點

```
class UserData(Base):
    __tablename__ = "user_data"

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    username: Mapped[str] = mapped_column(String(100))
    age: Mapped[int] = mapped_column(Integer)
    hobbies: Mapped[str] = mapped_column(Text) # 儲存 JSON string
    content: Mapped[str] = mapped_column(LONGTEXT) # 儲存 JSON string

class UserInput(BaseModel):
    username: str
    age: int
    hobbies: List[str]
    content: Any
```

👍 優點：

- ✅ 使用 `LONGTEXT` 儲存大型 JSON 非常合適

- ✓ Pydantic 設為 `Any` 可以接受任意格式的 JSON（彈性最大）
- ✓ ORM 寫法用 SQLAlchemy 2.0 typing 標準語法，沒問題

## ！可能的問題與注意事項

問題類型	描述	解法
1 資料未序列化	你不能直接把 `dict` 存進 `Mapped[str]` 欄位，要用 `json.dumps()`	儲存前用 `json.dumps(user.content)`
2 資料讀出後無法直接用	從資料庫讀出來的是 JSON 字串，要再用 `json.loads()` 才能還原成 dict	顯示前加 `json.loads(row.content)`
3 MariaDB 限制	雖然 `LONGTEXT` 容量夠大，但效率可能不如結構化欄位（僅在資料量極大時）	初期做實驗沒問題

## ✓ 最簡範例：JSON 資料存進 MariaDB `LONGTEXT`

### 資料庫表

```
CREATE TABLE user_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100),
    age INT,
    hobbies TEXT,
    content LONGTEXT
);
```

## FastAPI + SQLAlchemy 最小執行範例

```
from fastapi import FastAPI, Depends
from pydantic import BaseModel
from typing import List, Any
import json

from sqlalchemy import create_engine, String, Integer, Text
from sqlalchemy.dialects.mysql import LONGTEXT
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column,
sessionmaker, Session

DATABASE_URL =
"mariadb+mariadbconnector://user:password@localhost:3306/your_database"

engine = create_engine(DATABASE_URL, echo=True)
SessionLocal = sessionmaker(bind=engine)

# SQLAlchemy Declarative Base
class Base(DeclarativeBase):
    pass

class UserData(Base):
    __tablename__ = "user_data"

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    username: Mapped[str] = mapped_column(String(100))
    age: Mapped[int] = mapped_column(Integer)
    hobbies: Mapped[str] = mapped_column(Text)
    content: Mapped[str] = mapped_column(LONGTEXT) # JSON string

Base.metadata.create_all(engine)

# FastAPI schema
class UserInput(BaseModel):
    username: str
    age: int
    hobbies: List[str]
    content: Any # 可接受任何 JSON 結構
```

```

app = FastAPI()

# Dependency
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post("/store")
def store_data(user: UserInput, db: Session = Depends(get_db)):
    db_user = UserData(
        username=user.username,
        age=user.age,
        hobbies=json.dumps(user.hobbies),
        content=json.dumps(user.content)
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)

    return {"status": "ok", "id": db_user.id}

```

## 總結：你學會了什麼？

技術概念	說明
`LONGTEXT`	可儲存不固定結構的 JSON 字串
`json.dumps()`	dict → JSON 字串，存入 MariaDB
`json.loads()`	JSON 字串 → dict，用於前端還原
`content: Any`	接收任意格式 JSON，未定義 key 也可

---

✅ 這樣的設計非常適合你現在做實驗用 React 串接 FastAPI，未來也能擴充變動欄位。

要不要我幫你補上 `/get/{id}` API，把資料從資料庫讀出並還原 `content` 和 `hobbies` 成 JSON 格式再回傳？這樣你可以前後端串起來測試。



store\_data能加上response\_model嗎



當然可以！在 FastAPI 中使用 `response_model` 的好處是：

- 自動驗證與格式化回傳資料
- 在 `/docs` 頁面清楚顯示 API 回傳的 JSON 結構
- 保證回傳的資料結構一致（對 React 前端更友善）

---

✅ 改寫你的 `/store` 加上 `response_model`



步驟概覽：

1. 定義 `UserOutput` schema（對應要回傳的格式）
2. 在路由函式加上 `response_model=UserOutput`



## ✅ 修改後完整程式碼片段如下：

```
from fastapi import FastAPI, Depends
from pydantic import BaseModel
from typing import List, Any
import json

from sqlalchemy import create_engine, String, Integer, Text
from sqlalchemy.dialects.mysql import LONGTEXT
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column,
sessionmaker, Session

DATABASE_URL =
"mariadb+mariadbconnector://user:password@localhost:3306/your_database"

engine = create_engine(DATABASE_URL, echo=True)
SessionLocal = sessionmaker(bind=engine)

# --- SQLAlchemy base ---
class Base(DeclarativeBase):
    pass

class UserData(Base):
    __tablename__ = "user_data"

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    username: Mapped[str] = mapped_column(String(100))
    age: Mapped[int] = mapped_column(Integer)
    hobbies: Mapped[str] = mapped_column(Text)
    content: Mapped[str] = mapped_column(LONGTEXT)

Base.metadata.create_all(engine)

# --- Pydantic schemas ---
class UserInput(BaseModel):
    username: str
    age: int
    hobbies: List[str]
    content: Any
```

```

class UserOutput(BaseModel):
    id: int
    username: str
    age: int
    hobbies: List[str]
    content: Any

class Config:
    orm_mode = True # 允許從 ORM 物件轉換為 Pydantic 模型

# --- FastAPI app ---
app = FastAPI()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# --- 加上 response_model ---
@app.post("/store", response_model=UserOutput)
def store_data(user: UserInput, db: Session = Depends(get_db)):
    db_user = UserData(
        username=user.username,
        age=user.age,
        hobbies=json.dumps(user.hobbies),
        content=json.dumps(user.content)
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)

# 回傳時轉換成 UserOutput schema 所需格式
return UserOutput(
    id=db_user.id,
    username=db_user.username,
    age=db_user.age,
    hobbies=json.loads(db_user.hobbies),

```

```
        content=json.loads(db_user.content)
    )
```

## ✅ 實測結果 ` /store ` 回傳：

```
{
  "id": 1,
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],
  "content": {
    "message": "hello",
    "student": "allen"
  }
}
```

## 🧠 補充：為什麼要用 `orm\_mode = True` ？

這是讓 Pydantic 知道你要處理的資料是 SQLAlchemy ORM 物件，可以自動從欄位拉資料。

雖然這裡你手動建了 `UserOutput(...)`，但你未來如果直接用 `return db\_user`，有 `orm\_mode` 就不會出錯。

需要我幫你再補 ` /get/{id} ` 或 ` /list ` API，讓 React 前端能抓資料回來用嗎？你現在已經可以完整寫入 JSON 了 🍌





詳細說明:

```
@app.post("/store", response_model=UserOutput)
def store_data(user: UserInput, db: Session =
Depends(get_db)):
    db_user = UserData(
        username=user.username,
        age=user.age,
        hobbies=json.dumps(user.hobbies),
        content=json.dumps(user.content)
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)

    # 回傳時轉換成 UserOutput schema 所需格式
    return UserOutput(
        id=db_user.id,
        username=db_user.username,
        age=db_user.age,
        hobbies=json.loads(db_user.hobbies),
        content=json.loads(db_user.content)
    )
```



你提供的 `store\_data` 函式設計得很好，這是 FastAPI 搭配 SQLAlchemy 存取資料庫時非常常見的寫法。

我來**逐行**幫你詳細說明這段程式碼的意義和作用，讓你徹底理解這段怎麼運作，為日後開發打下基礎。



函式總覽



```
@app.post("/store", response_model=UserOutput)
def store_data(user: UserInput, db: Session = Depends(get_db)):
```

## 解釋：

- `@app.post("/store")`：定義一個 POST 路由 `/store`。
- `response_model=UserOutput`：定義回傳結果格式是 `UserOutput`，這會影響 `/docs` 自動產生的 API 文件與回傳驗證。
- `user: UserInput`：這個參數會自動接收來自前端的 JSON，並轉換為 Pydantic 的 `UserInput` 模型。
- `db: Session = Depends(get_db)`：透過 FastAPI 的 `Depends` 取得 SQLAlchemy session，從 `get_db()` 產生，做資料庫操作。

## 資料寫入流程

```
db_user = UserData(
    username=user.username,
    age=user.age,
    hobbies=json.dumps(user.hobbies),
    content=json.dumps(user.content)
)
```

## 解釋：

你建立一個 ORM 物件 `db_user`，對應到 `UserData` 資料表：

- `user.username`：從 Pydantic 模型抓出來，是 `str`
- `user.age`： `int`

- `user.hobbies`：是 `List[str]`，要先用 `json.dumps()` 序列化成字串後才能存進資料庫的 `Text` 欄位
- `user.content`：是 `Any`，即 Python 的 `dict`、巢狀 JSON，用 `json.dumps()` 儲存為 `LONGTEXT`

⚠ **重點：SQLAlchemy 的欄位是 `Mapped[str]`，不接受 `dict`，必須先用 `json.dumps()` 變成字串。**

## ✓ 寫入資料庫

```
db.add(db_user)
db.commit()
db.refresh(db_user)
```

解釋：

1. `db.add(...)`：將物件加入 session，告訴 SQLAlchemy 要新增
2. `db.commit()`：提交 transaction，實際寫入資料庫
3. `db.refresh(db_user)`：更新 ORM 物件，會把 `id`（auto increment）等欄位自動帶入

## ↺ 回傳資料（轉換回 JSON）

```
return UserOutput(
    id=db_user.id,
    username=db_user.username,
    age=db_user.age,
    hobbies=json.loads(db_user.hobbies),
```

```
        content=json.loads(db_user.content)
    )
```

## 解釋：

因為你在資料庫中是用 ``json.dumps()`` 存的字串，回傳前要還原回 **Python 結構 (list/dict)**：

- ``json.loads(db_user.hobbies)``：把 ``["coding", "gaming"]`` 這種字串還原成 List
- ``json.loads(db_user.content)``：把 ``{"message": "hello"}`` 字串還原成 Dict

然後塞進 ``UserOutput`` 模型，讓 FastAPI 根據這個 Pydantic schema 回傳正確格式的 JSON。

## ✅ 最終前端收到的 JSON：

```
{
  "id": 1,
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],
  "content": {
    "message": "hello",
    "student": "allen"
  }
}
```

這個格式適合用在 React，前端可以直接解構、顯示、甚至再送出修改。

## ← 總結重點

END

步驟	說明
接收資料	使用 Pydantic <code>`UserInput`</code> 自動驗證 JSON
資料序列化	<code>`json.dumps()`</code> 將 list、dict 存為文字
寫入資料庫	使用 SQLAlchemy ORM + <code>`db.add()`</code>
讀出資料	使用 <code>`json.loads()`</code> 還原為 JSON 結構
回傳結構	使用 <code>`response_model`</code> 確保統一、乾淨

如果你想讓讀取 API ``/get/{id}`` 自動也還原 ``hobbies``、``content`` 成 JSON，我也可以幫你寫好。

需要我幫你補這個嗎？或者你現在想串接 React 前端測試？



想讓讀取 API `/get/{id}` 自動也還原 hobbies、content 成 JSON



太好了，這是你目前架構下非常自然的一步！

你要建立 ``/get/{id}`` API，用來從資料庫撈一筆資料，並自動把 ``hobbies`` 和 ``content`` 這兩個欄位從儲存在資料庫裡的 JSON 字串還原成真正的 Python 結構 (list / dict)，再包成 Pydantic 的 ``response_model`` 回傳給前端 (如 React)。

## ✅ 加入 `/get/{id}` API 的做法

你已經有的：

- SQLAlchemy ORM `UserData`
- Pydantic schema `UserOutput`（可以重複用）

## ✅ 加在你的 `main.py` 中

```
from fastapi import HTTPException

@app.get("/get/{id}", response_model=UserOutput)
def get_user(id: int, db: Session = Depends(get_db)):
    db_user = db.query(UserData).filter(UserData.id == id).first()
    if not db_user:
        raise HTTPException(status_code=404, detail="User not found")

    return UserOutput(
        id=db_user.id,
        username=db_user.username,
        age=db_user.age,
        hobbies=json.loads(db_user.hobbies),
        content=json.loads(db_user.content)
    )
```

## 🔍 功能說明：

區塊

功能

`@app.get("/get/{id}")`

建立 GET API，支援網址像 `/get/1`

區塊	功能
<code>`response_model=UserOutput`</code>	用同樣格式回傳跟 <code>`/store`</code> 一致的 JSON 結構
<code>`db.query(...).filter(...).first()`</code>	查詢一筆資料
<code>`json.loads(...)`</code>	將資料庫中的字串型 JSON 還原成 list/dict
<code>`UserOutput(...)`</code>	組成乾淨的 Pydantic schema 讓 FastAPI 幫你產出漂亮的 JSON
<code>`raise HTTPException(...)`</code>	若找不到資料就回傳 404

✅ 呼叫 ``/get/1`` 的回傳結果（範例）：

```
{
  "id": 1,
  "username": "佳恩",
  "age": 28,
  "hobbies": ["coding", "gaming"],
  "content": {
    "message": "hello",
    "student": "allen"
  }
}
```

## ✅ 額外小建議（進階）

若你未來大量使用 ``json.dumps/json.loads``，你可以：

- 改用 SQLAlchemy JSON type，讓資料庫自動處理（但 MariaDB 支援較差）
- 或自己包裝儲存與還原邏輯，讓程式更乾淨（我可以教你如何抽成函式）

---

如果你要再做：

- ``/list``：回傳所有資料
- ``/update/{id}``：修改某筆資料
- ``/delete/{id}``：刪除資料

我也可以幫你補上，讓你完整 CRUD 一條龍。要繼續嗎？



包裝儲存與還原邏輯，讓程式更乾淨、CRUD 一條龍



太棒了！我會幫你完成以下兩個目標：

---

## 目標一：包裝 ``json.dumps``/``json.loads`` 的儲存與還原邏輯

避免每次都手動呼叫 ``json.dumps()`` 和 ``json.loads()``，讓程式更乾淨、好維護。

---

## 目標二：提供完整 **CRUD API 一條龍**

- ``POST /store``：新增資料
- ``GET /get/{id}``：查詢一筆





- ``GET /list``：查詢全部
- ``PUT /update/{id}``：更新
- ``DELETE /delete/{id}``：刪除

## ✅ 完整程式碼（最乾淨的設計）

```
from fastapi import FastAPI, Depends, HTTPException
from pydantic import BaseModel
from typing import List, Any
import json

from sqlalchemy import create_engine, String, Integer, Text, select
from sqlalchemy.dialects.mysql import LONGTEXT
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column,
sessionmaker, Session

# -----
# 資料庫設定
# -----
DATABASE_URL =
"mariadb+mariadbconnector://user:password@localhost:3306/your_database"

engine = create_engine(DATABASE_URL, echo=True)
SessionLocal = sessionmaker(bind=engine)

class Base(DeclarativeBase):
    pass

# -----
# ORM 模型
# -----
class UserData(Base):
    __tablename__ = "user_data"
```

```

id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
username: Mapped[str] = mapped_column(String(100))
age: Mapped[int] = mapped_column(Integer)
hobbies: Mapped[str] = mapped_column(Text)      # JSON 字串
content: Mapped[str] = mapped_column(LONGTEXT)  # JSON 字串

```

# 儲存前自動處理轉 JSON 字串

```
@classmethod
```

```

def from_input(cls, user: "UserInput") -> "UserData":
    return cls(
        username=user.username,
        age=user.age,
        hobbies=json.dumps(user.hobbies),
        content=json.dumps(user.content),
    )

```

# 還原為輸出型別 (回傳前)

```

def to_output(self) -> "UserOutput":
    return UserOutput(
        id=self.id,
        username=self.username,
        age=self.age,
        hobbies=json.loads(self.hobbies),
        content=json.loads(self.content),
    )

```

# -----

# Pydantic Schema

# -----

```

class UserInput(BaseModel):
    username: str
    age: int
    hobbies: List[str]
    content: Any

```

```

class UserOutput(BaseModel):
    id: int
    username: str
    age: int
    hobbies: List[str]
    content: Any

```

```

class Config:
    orm_mode = True

# -----
# FastAPI 主體
# -----
app = FastAPI()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# -----
# CRUD API 一條龍
# -----

# ✅ 建立
@app.post("/store", response_model=UserOutput)
def store_user(user: UserInput, db: Session = Depends(get_db)):
    db_user = UserData.from_input(user)
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user.to_output()

# ✅ 查詢單筆
@app.get("/get/{id}", response_model=UserOutput)
def get_user(id: int, db: Session = Depends(get_db)):
    db_user = db.get(UserData, id)
    if not db_user:
        raise HTTPException(status_code=404, detail="User not found")
    return db_user.to_output()

# ✅ 查詢多筆
@app.get("/list", response_model=List[UserOutput])
def list_users(db: Session = Depends(get_db)):
    results = db.execute(select(UserData)).scalars().all()

```

```
return [user.to_output() for user in results]
```

#  更新

```
@app.put("/update/{id}", response_model=UserOutput)
```

```
def update_user(id: int, user: UserInput, db: Session = Depends(get_db)):
```

```
    db_user = db.get(UserData, id)
```

```
    if not db_user:
```

```
        raise HTTPException(status_code=404, detail="User not found")
```

```
    db_user.username = user.username
```

```
    db_user.age = user.age
```

```
    db_user.hobbies = json.dumps(user.hobbies)
```

```
    db_user.content = json.dumps(user.content)
```

```
    db.commit()
```

```
    db.refresh(db_user)
```

```
    return db_user.to_output()
```

#  刪除

```
@app.delete("/delete/{id}")
```

```
def delete_user(id: int, db: Session = Depends(get_db)):
```

```
    db_user = db.get(UserData, id)
```

```
    if not db_user:
```

```
        raise HTTPException(status_code=404, detail="User not found")
```

```
    db.delete(db_user)
```

```
    db.commit()
```

```
    return {"status": "deleted", "id": id}
```

## 測試方式建議

你可以用 Swagger UI 進行操作：

- 啟動：`uvicorn main:app --reload`
- 開啟：`http://localhost:8000/docs`

- 測試順序：
  1. ``POST /store`` → 新增
  2. ``GET /list`` → 查看全部
  3. ``GET /get/{id}`` → 查看一筆
  4. ``PUT /update/{id}`` → 修改
  5. ``DELETE /delete/{id}`` → 刪除

---

## Bonus：開發測試小技巧

操作	建議做法
測試 JSON 輸入	用 Swagger UI 貼上 JSON
觀察儲存內容	使用 DBeaver 或 Navicat 檢查資料庫欄位內容
整合 React	把 <code>`/store`</code> 、 <code>`/list`</code> 等當成 API 直接用 <code>`fetch`</code> 發請求

---

需要我幫你補 React 前端的串接範例？還是教你怎麼用 React 呼叫 ``/store`` + ``/list`` 呢？我可以幫你寫簡單範例！



補 React 前端的串接範例



太好了！這邊幫你補一個完整且簡潔的 **React 前端範例**，用來呼叫你剛剛建立的 FastAPI API，實現：

- ✓ 輸入資料（送到 `/store`）
- ✓ 顯示所有資料（從 `/list` 取得）

## ✓ 1. 安裝 React 專案（若你還沒建立）

```
npx create-react-app fastapi-json-demo
cd fastapi-json-demo
npm start
```

## ✓ 2. 編輯 `App.js`

把原本的內容全部改成以下範例：

```
import React, { useState, useEffect } from "react";

function App() {
  const [form, setForm] = useState({
    username: "",
    age: "",
    hobbies: "",
    content: "",
  });

  const [users, setUsers] = useState([]);

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };
}
```

```

const handleSubmit = async (e) => {
  e.preventDefault();

  try {
    const payload = {
      username: form.username,
      age: parseInt(form.age),
      hobbies: form.hobbies.split(",").map((h) => h.trim()),
      content: JSON.parse(form.content),
    };

    const res = await fetch("http://localhost:8000/store", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(payload),
    });

    const data = await res.json();
    alert("新增成功, ID: " + data.id);
    fetchUsers(); // 重新載入列表
  } catch (err) {
    console.error("提交失敗:", err);
    alert("錯誤: 請確認 content 欄位為合法 JSON 格式");
  }
};

const fetchUsers = async () => {
  const res = await fetch("http://localhost:8000/list");
  const data = await res.json();
  setUsers(data);
};

useEffect(() => {
  fetchUsers();
}, []);

return (
  <div style={{ padding: "2rem" }}>

```

```

<h1>新增使用者資料</h1>
<form onSubmit={handleSubmit}>
  <div>
    <label>使用者名稱：</label>
    <input name="username" value={form.username} onChange={handleChange}
required />
  </div>
  <div>
    <label>年齡：</label>
    <input name="age" value={form.age} onChange={handleChange} required
/>
  </div>
  <div>
    <label>興趣（用逗號分隔）：</label>
    <input name="hobbies" value={form.hobbies} onChange={handleChange}
required />
  </div>
  <div>
    <label>內容 content（請填入合法 JSON）：</label>
    <textarea
      name="content"
      value={form.content}
      onChange={handleChange}
      placeholder='例如：{"message": "hello", "student": "allen"}'
      rows={4}
      cols={40}
      required
    />
  </div>
  <button type="submit">送出</button>
</form>

<hr />

<h2>使用者清單</h2>
{users.length === 0 ? (
  <p>尚無資料</p>
) : (
  <ul>
    {users.map((u) => (
      <li key={u.id}>

```



```

        <strong>{u.username}</strong> ({u.age} 歲)
        <br />
        興趣：{u.hobbies.join(", ")} <br />
        content: <pre>{JSON.stringify(u.content, null, 2)}</pre>
    </li>
  )})
</ul>
})
</div>
);
}

export default App;

```

### ✓ 3. 測試資料格式

你可以這樣輸入：

- 使用者名稱：佳恩
- 年齡：28
- 興趣：coding, gaming
- content：

```
{"message": "hello", "student": "allen"}
```

### ✓ 快速回顧後端 API

API 路徑	方法	功能
<code>`/store`</code>	POST	新增資料
<code>`/list`</code>	GET	查詢全部

## ✅ CORS 問題處理（如果有跨域錯誤）

在 ``FastAPI`` 中加上 CORS 設定（建議放在最上面）：

```
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

# 允許所有來源（開發用）
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # 或改成你的 React 來源 http://localhost:3000
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

## 🎉 你完成了！

你現在有一個完整的 **React + FastAPI + MariaDB** 資料流程，並用 JSON 結構成功存取 ``LONGTEXT`` 欄位，非常適合做進一步實驗與開發。

需要我幫你補：

- ``PUT /update/{id}`` 在前端怎麼呼叫？

- 或整合成編輯功能？

都可以幫你實作！你接下來想做什麼呢？