台灣人工智慧學校 工百業用AI

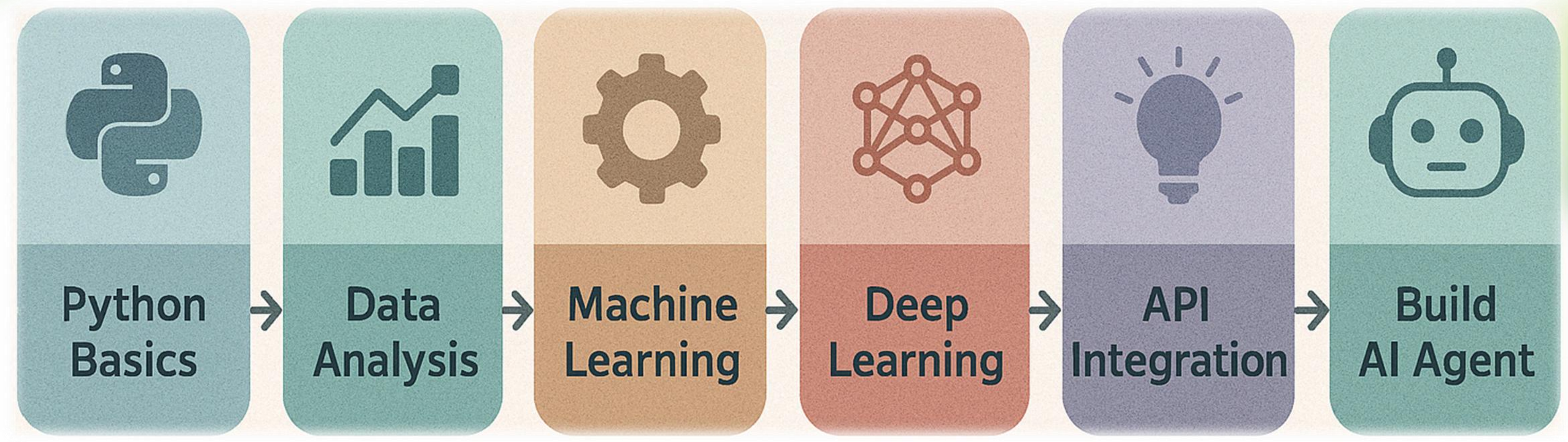
Web Framework FastAPI

Daniel ，AIA台灣人工智慧學校　AI 工程師

@20250417

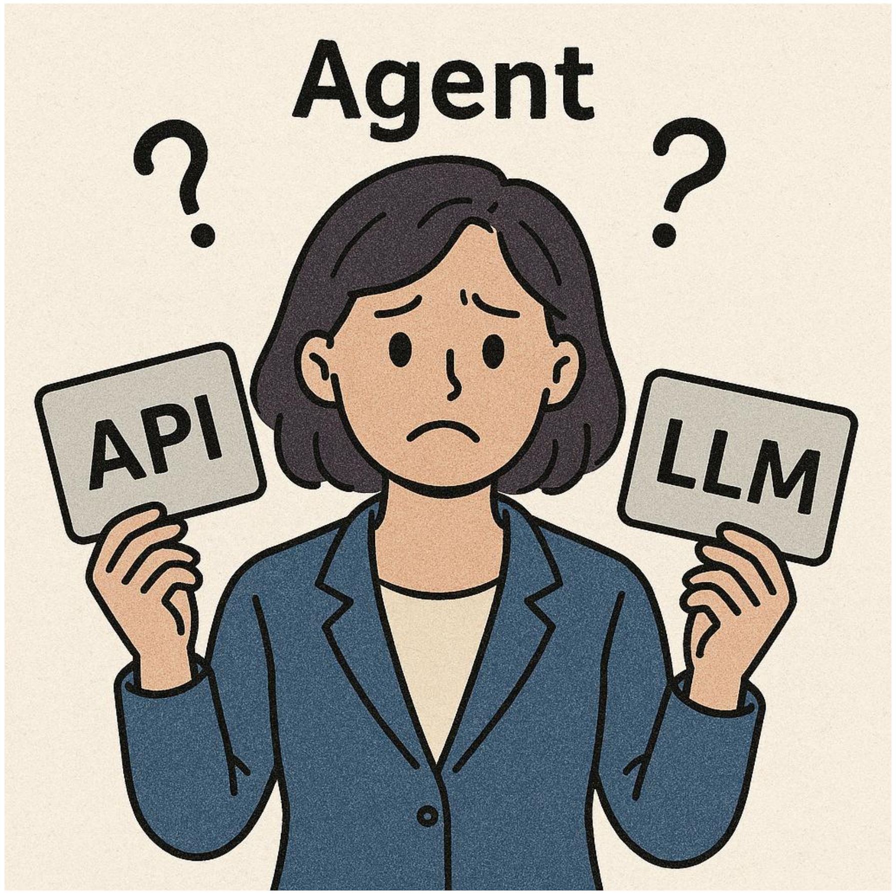
- AIA台灣人工智慧學校
    - 技術發展處　AI 工程師
- 學經歷：
    - 國立高雄應用科技大學　電子工程系所
- 工作經歷：
    - 英業達股份有限公司 系統工程師
- 專長：
    - 系統整合
    - AI 應用
- Email: lee.daniel@aiacademy.tw

我們學到什麼?

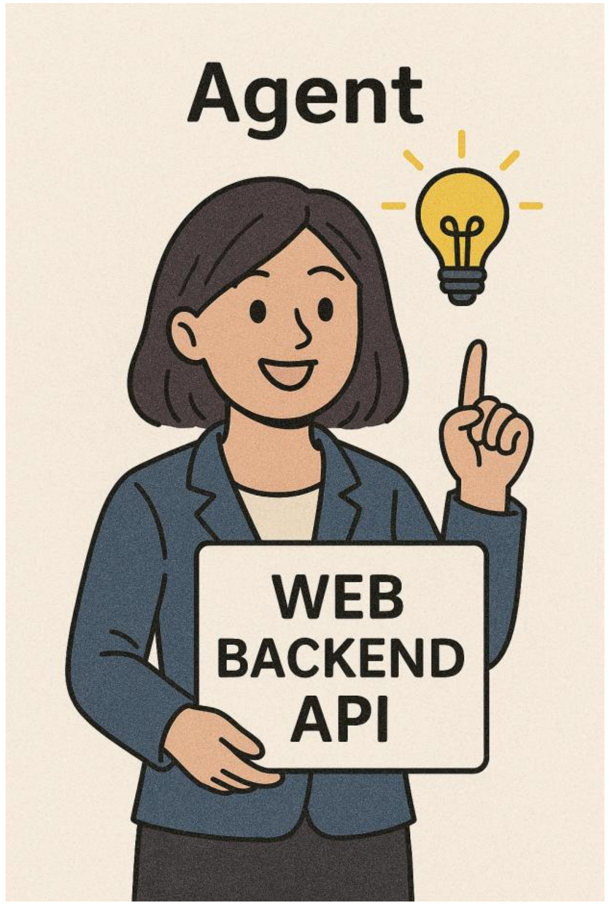


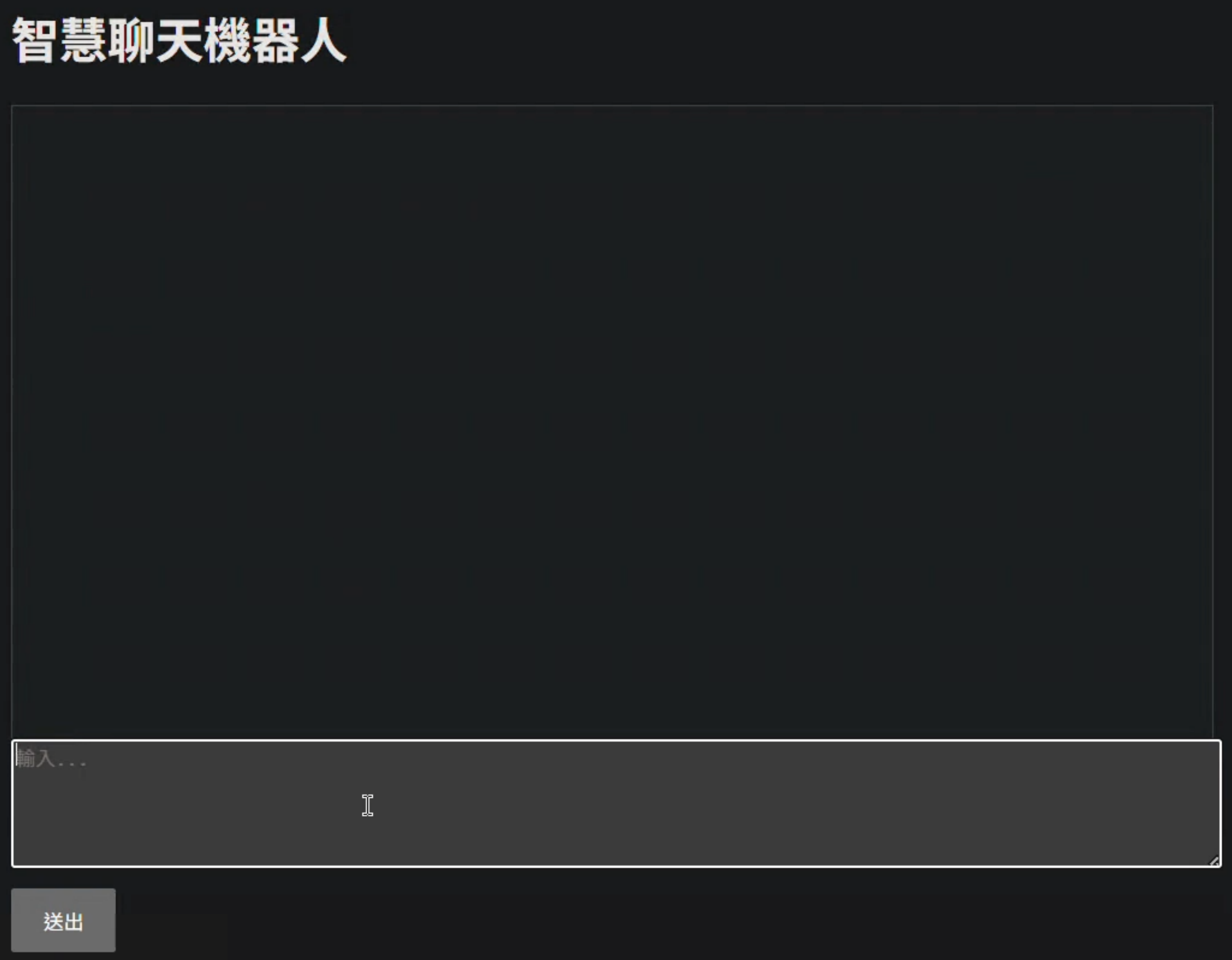
Python Basics
Data Analysis
Machine Learning
Deep Learning
API Integration
Build AI Agent

Agent 拿到 API 與 LLM，接下來該怎麼做？



部屬： Agent 伺服器!

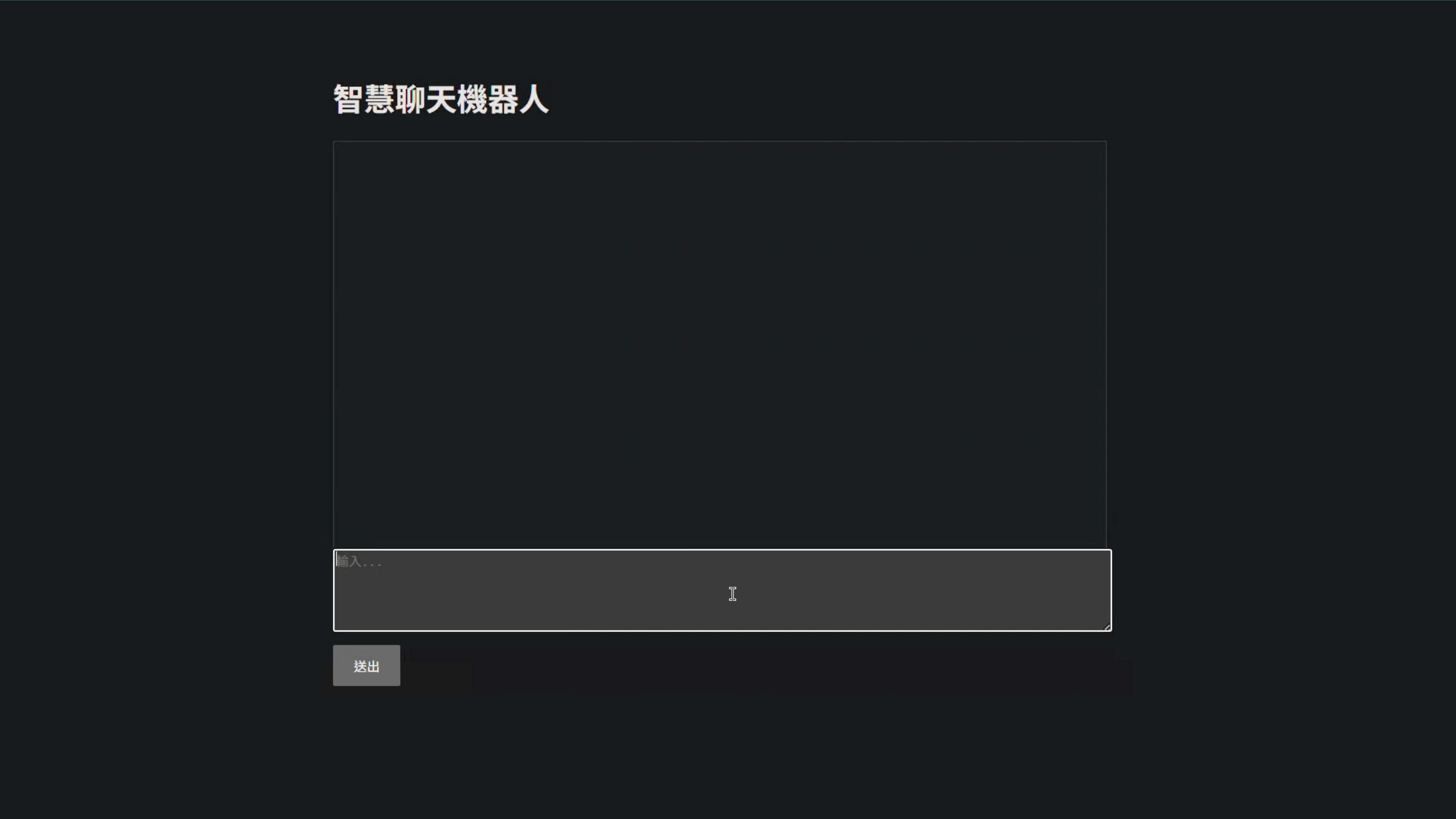




智慧聊天機器人

輸入...

送出


智慧聊天機器人

Agenda

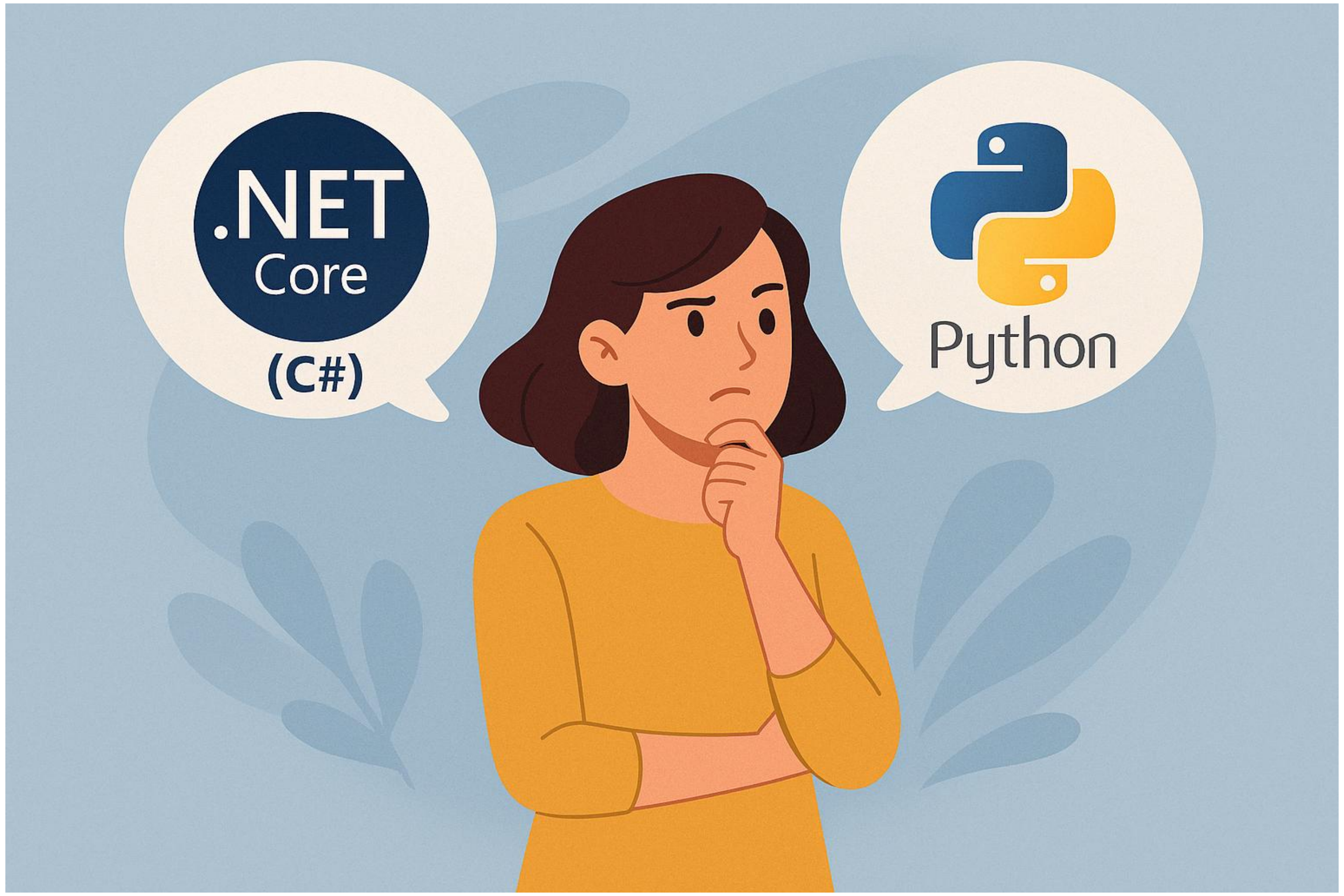
- Web Server 比較
- FastAPI 基礎框架
- Restful API 資料驗證
- 依賴注入 非同步 連線資料庫

台灣人工智慧學校 工百業用AI

Web Server 比較

台灣人工智慧學校 工百業用AI

你的 AI 後端該怎麼選？
.NET Core 或 Python？



不同語言的網頁後端

	FastAPI（Python）	ASP.NET Core（C#）
開發效率	語法簡潔，開發快速	強型別，開發較嚴謹
學習門檻	易學	較高
效能	效能高	效能更高
部署	輕量，支援多平台	較大，支援多平台
社群資源	AI/資料科學生態強，成長快速	微軟支持，穩定成熟
適合應用	原型、微服務、AI 應用	企業系統、大型 Web 應用

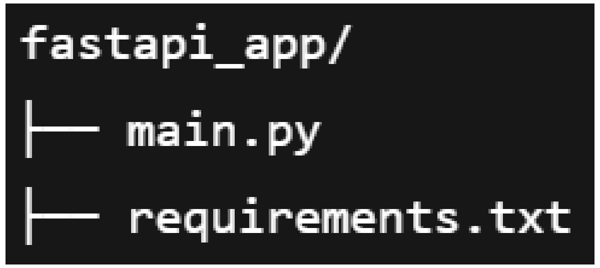
結論

	FastAPI	ASP.NET Core
開發方案	快速打造 API / 原型 / AI 服務	跨部門企業級 Web 系統
團隊技術	開發團隊熟 Python、AI	已有 C# 團隊
部署方案	雲端部署輕量微服務	Azure 雲原生整合

台灣人工智慧學校 工百業用AI

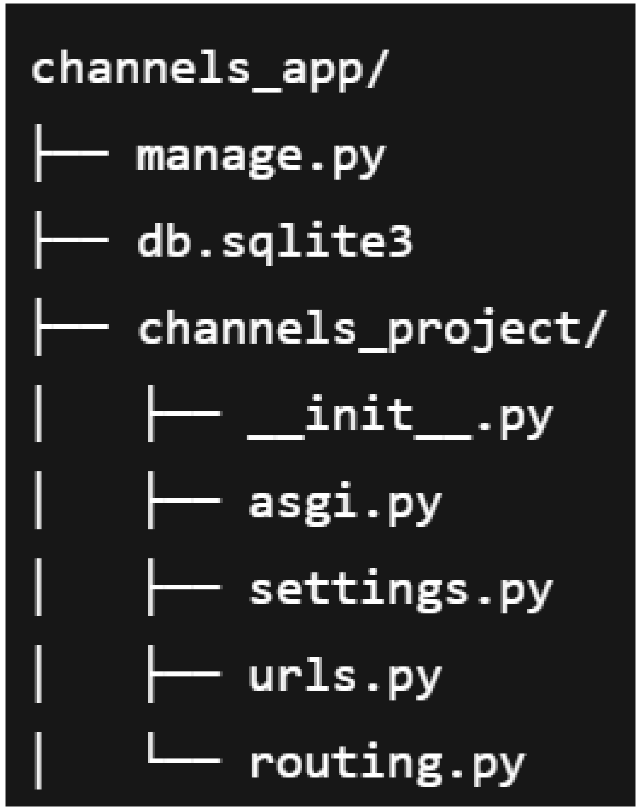
Python 後端框架，我該選誰？

FastAPI vs Django Channels

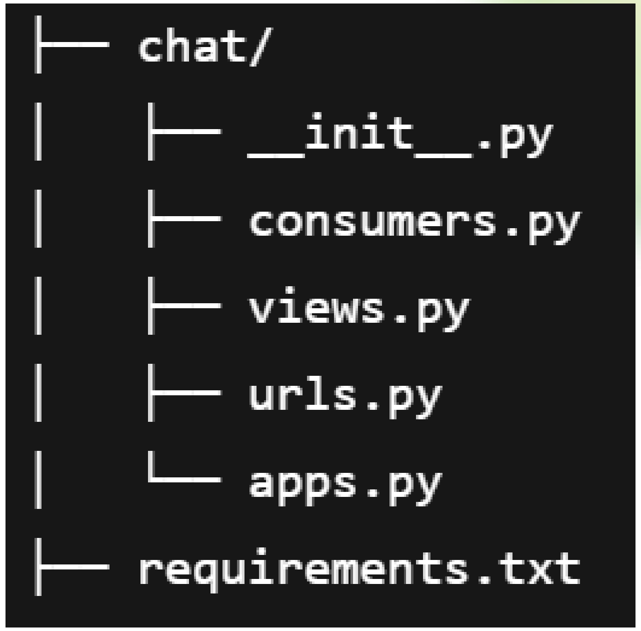


```
fastapi_app/
├── main.py
├── requirements.txt
```

FastAPI



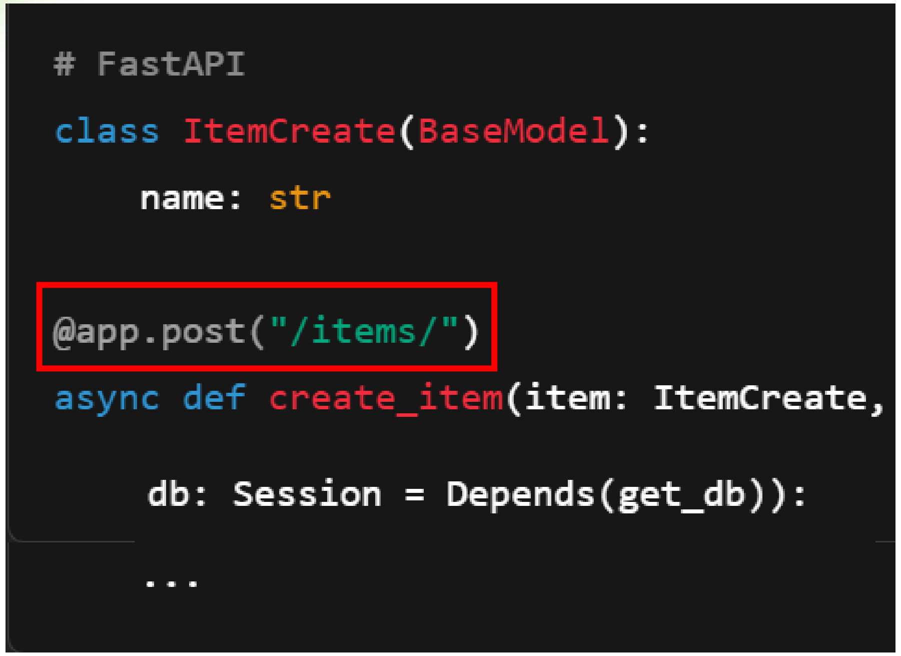
```
channels_app/
├── manage.py
├── db.sqlite3
├── channels_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── routing.py
```



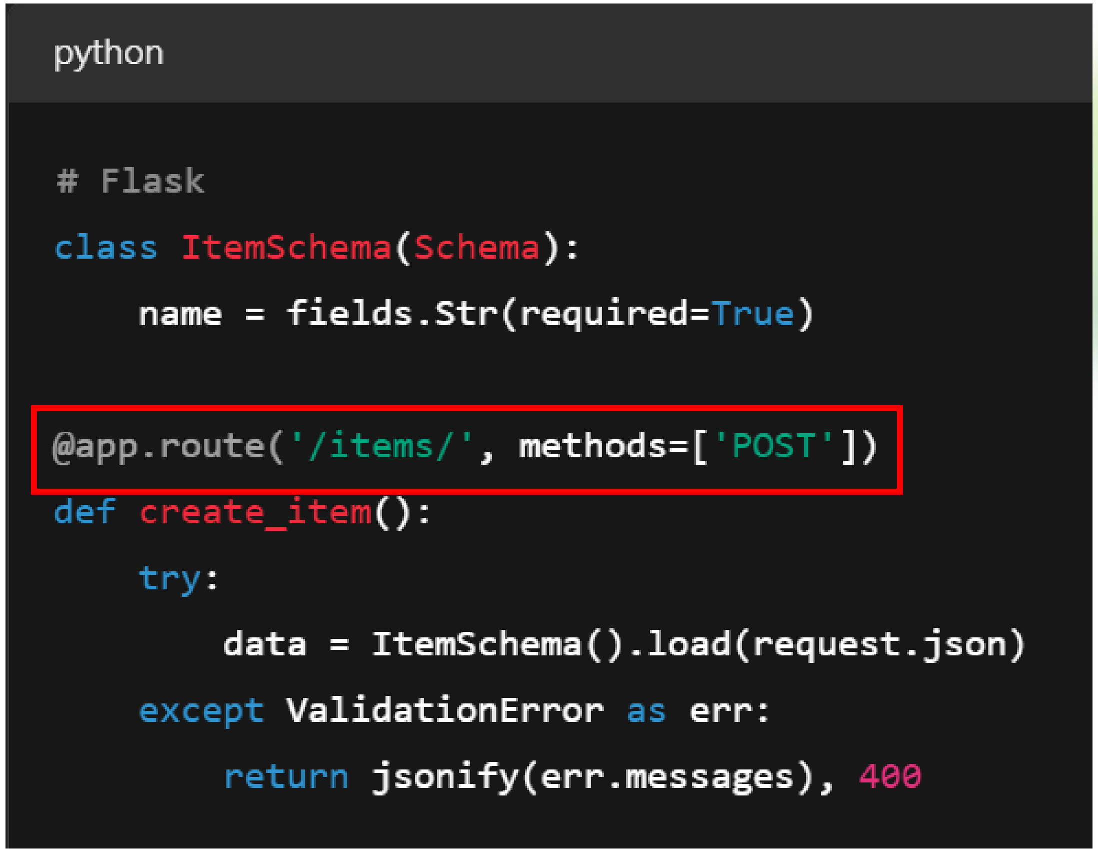
```
├── chat/
│   ├── __init__.py
│   ├── consumers.py
│   ├── views.py
│   ├── urls.py
│   └── apps.py
├── requirements.txt
```

Django Channels

FastAPI vs Flask (Route)



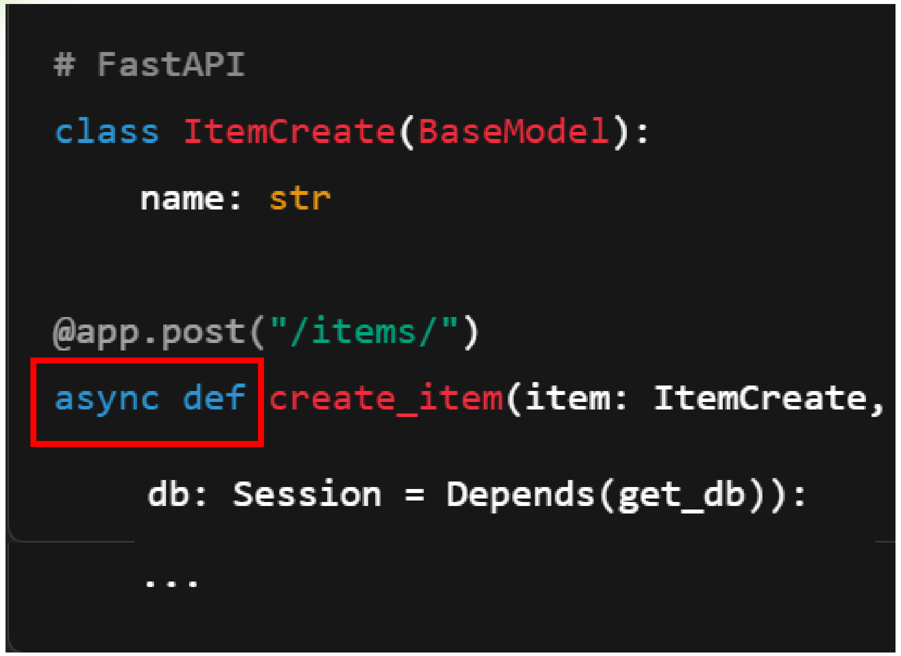
```python
# FastAPI
class ItemCreate(BaseModel):
    name: str

@app.post("/items/")
async def create_item(item: ItemCreate,

      db: Session = Depends(get_db)):

    ...
```



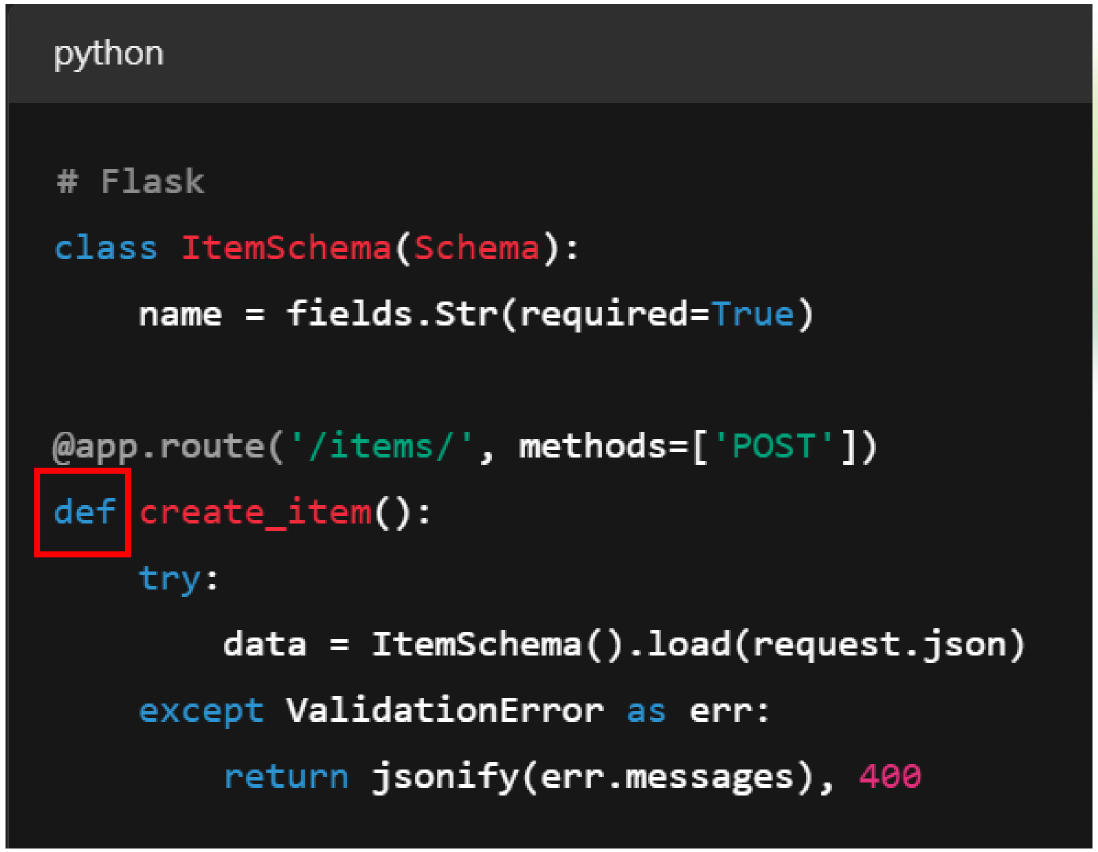
python

```python
# Flask
class ItemSchema(Schema):
    name = fields.Str(required=True)

@app.route('/items/', methods=['POST'])
def create_item():
    try:
        data = ItemSchema().load(request.json)
    except ValidationError as err:
        return jsonify(err.messages), 400
```

FastAPI vs Flask (Async)



```python
# FastAPI
class ItemCreate(BaseModel):
    name: str

@app.post("/items/")
async def create_item(item: ItemCreate,
     db: Session = Depends(get_db)):
    ...
```

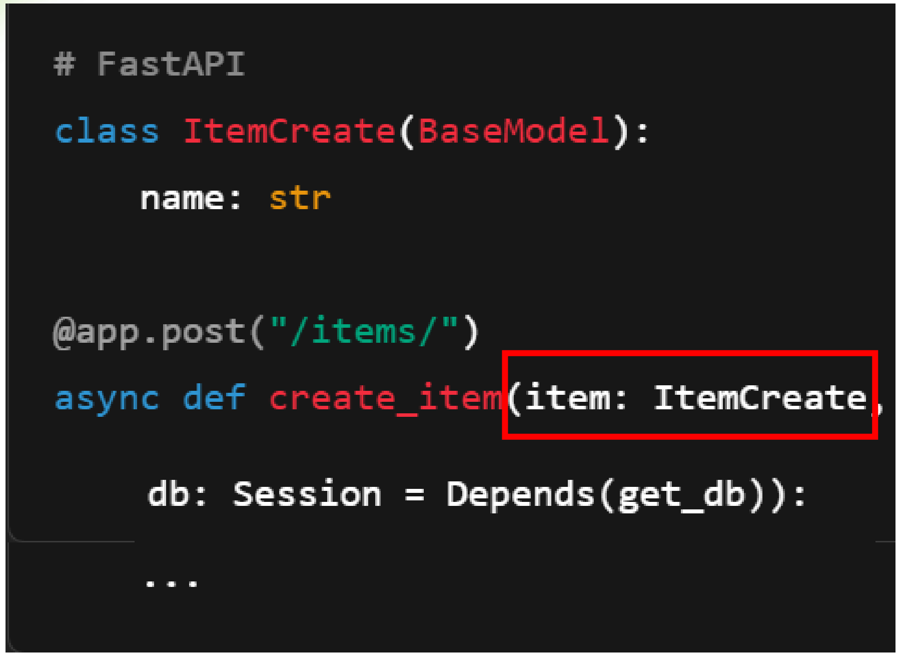


python

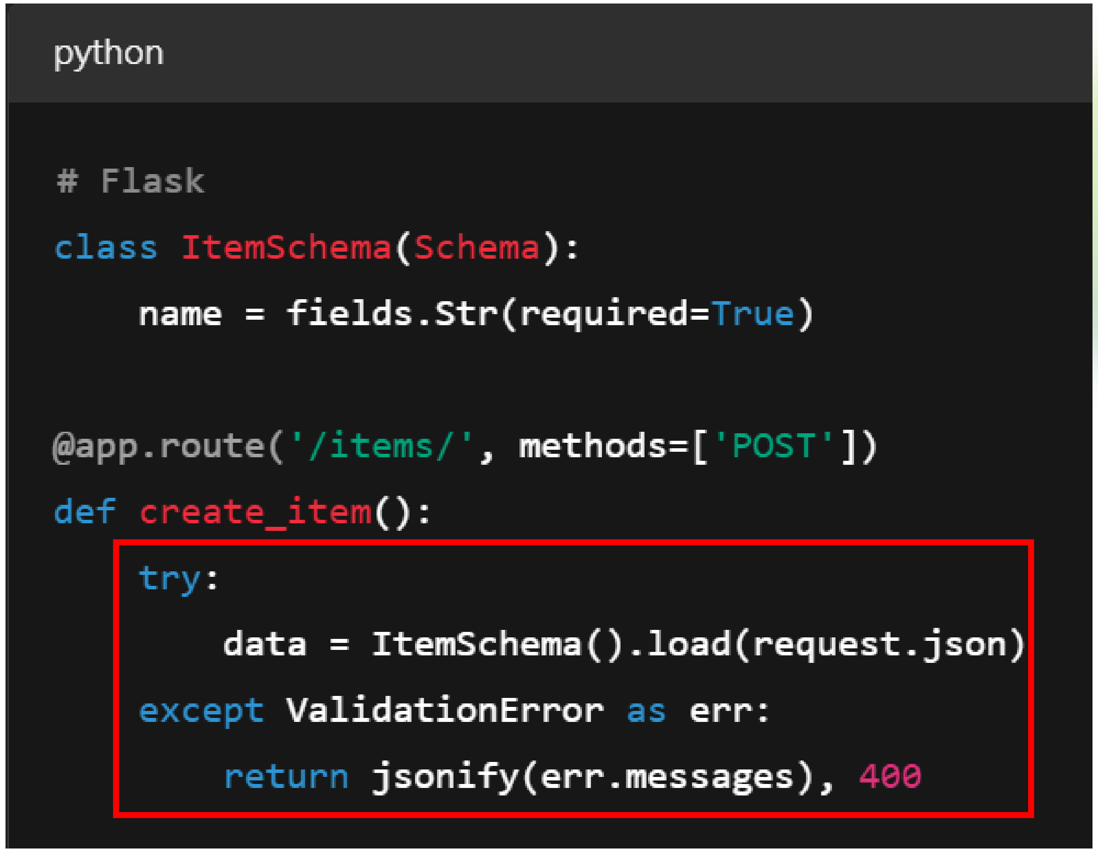
```python
# Flask
class ItemSchema(Schema):
    name = fields.Str(required=True)

@app.route('/items/', methods=['POST'])
def create_item():
    try:
        data = ItemSchema().load(request.json)
    except ValidationError as err:
        return jsonify(err.messages), 400
```

台灣人工智慧學校 工百業用AI

FastAPI vs Flask (驗證)



```python
# FastAPI
class ItemCreate(BaseModel):
    name: str

@app.post("/items/")
async def create_item(item: ItemCreate,
     db: Session = Depends(get_db)):
    ...
```

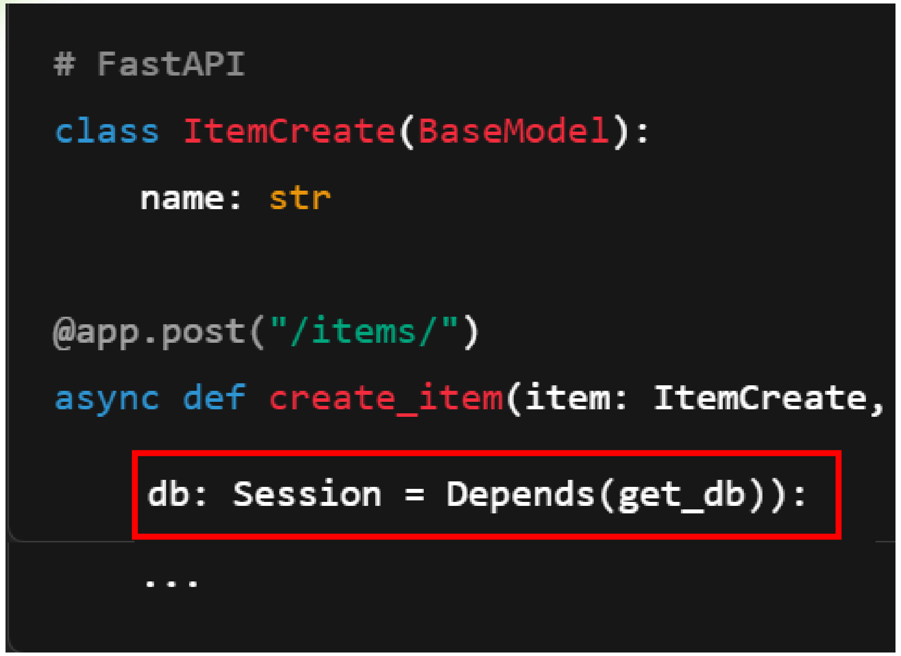


python

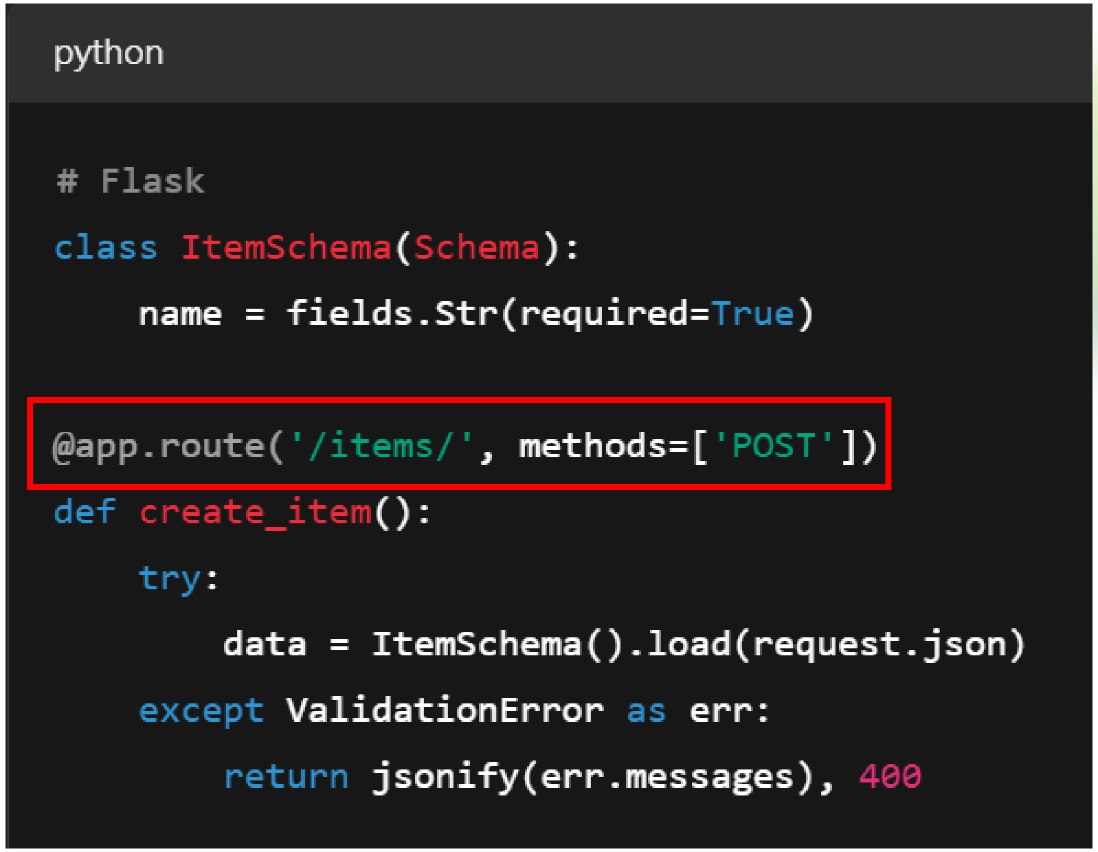
```python
# Flask
class ItemSchema(Schema):
    name = fields.Str(required=True)

@app.route('/items/', methods=['POST'])
def create_item():
    try:
        data = ItemSchema().load(request.json)
    except ValidationError as err:
        return jsonify(err.messages), 400
```

台灣人工智慧學校 工百業用AI

FastAPI vs Flask (依賴注入)



```python
# FastAPI
class ItemCreate(BaseModel):
    name: str

@app.post("/items/")
async def create_item(item: ItemCreate,
    db: Session = Depends(get_db)):
    ...
```



python

```python
# Flask
class ItemSchema(Schema):
    name = fields.Str(required=True)

@app.route('/items/', methods=['POST'])
def create_item():
    try:
        data = ItemSchema().load(request.json)
    except ValidationError as err:
        return jsonify(err.messages), 400
```

台灣人工智慧學校 工百業用AI

FastAPI vs Flask (Swagger)

```python
# FastAPI
class ItemCreate(BaseModel):
    name: str

@app.post("/items/")
async def create_item(item: ItemCreate,
     db: Session = Depends(get_db)):
    ...
```

python

```python
# Flask
class ItemSchema(Schema):
    name = fields.Str(required=True)

@app.route('/items/', methods=['POST'])
def create_item():
    try:
        data = ItemSchema().load(request.json)
    except ValidationError as err:
        return jsonify(err.messages), 400
```

FastAPI vs Flask vs Django

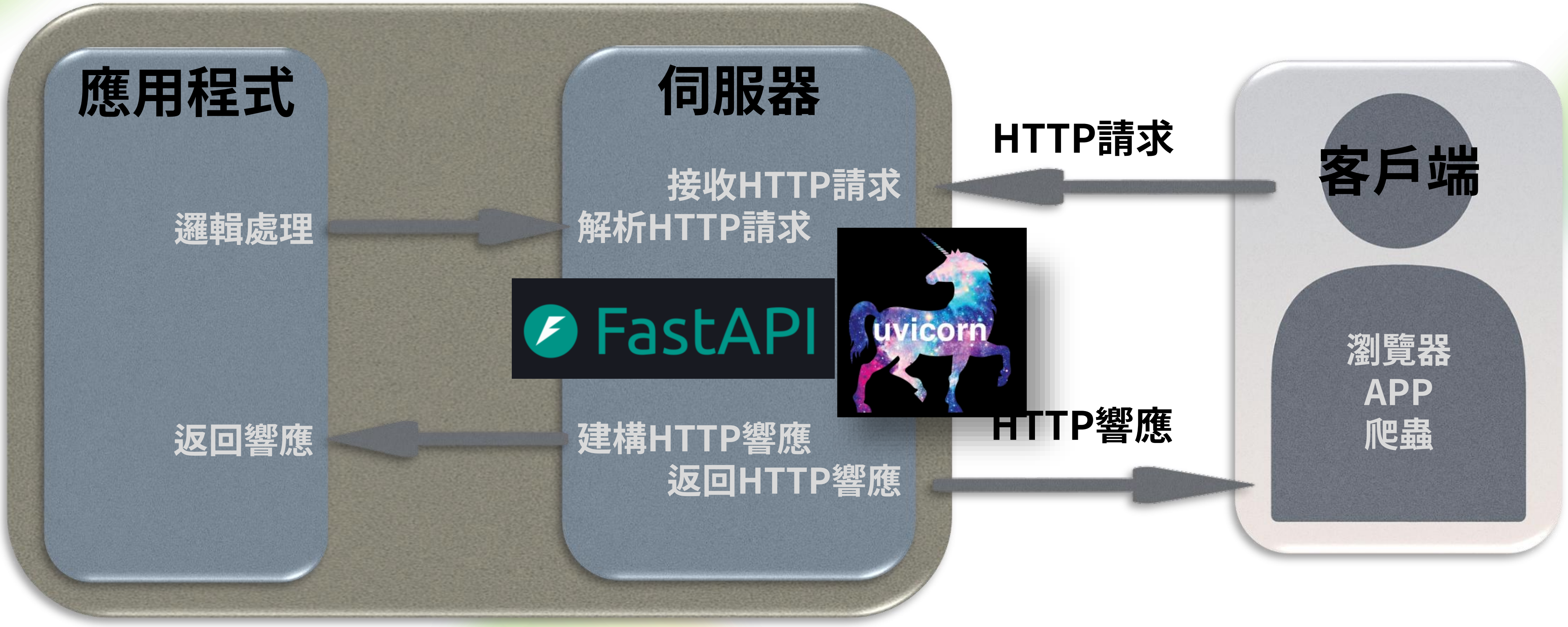
Python後端開發框架比較

	Flask	FastAPI	Django
框架類型	微服務框架	高效能API框架	全功能框架
框架特點	簡單靈活	注重效能	嚴格設計
內建功能	無內建	自動生成 API文件	很多
學習曲線	平緩	簡單	陡峭
框架效能	較低 適合靜態應用	非常高 適合大量請求應用	良好 適合全功能應用

FastAPI基礎框架

台灣人工智慧學校 工百業用AI

Web 應用程式架構


應用程式
邏輯處理
返回響應
伺服器
接收HTTP請求
解析HTTP請求
FastAPI
uvicorn
建構HTTP響應
返回HTTP響應
HTTP請求
HTTP響應
客戶端
瀏覽器
APP
爬蟲

FastAPI 基礎框架

套件安裝

```python
pip install fastapi uvicorn
```

實作 FsatAPI 基礎框架

python

```python
from fastapi import FastAPI
import uvicorn

app = FastAPI()

@app.get("/resource")
def read_resource ():
   return {"message": "Ok!"}

if __name__ == "__main__":
   uvicorn.run("FastAPI:app", host="127.0.0.1", port=8000)
```

啟動 FastAPI 的不同方法

```python
Terminal:

fastapi dev main.py
fastapi run main.py
uvicorn main:app --host 0.0.0.0 --port 8000

Python:

if __name__ == "__main__":
   uvicorn.run("FastAPI:app", host="127.0.0.1", port=8000)
```

https://fastapi.tiangolo.com/zh-hant/deployment/manually/?h=manual

撰寫 Swagger

Swagger
撰寫 Swagger 文件
1.基本資訊
2.分組資訊
3.路由資訊

加上 Swagger 資訊

```python
app = FastAPI(
  title="我的 API",
  description="提供資源",
  version="1.0.0",
  contact={
    "name": "Daniel Lee",
    "email": "daniel@example.com",
  },
  license_info={
    "name": "MIT",
    "url": "https://opensource.org/licenses/MIT",
  }
)
```

加上 Swagger Tags

```python
tags_metadata = [
    {
        "name": "Resourse",
        "description": "資源",
    }
]
```

台灣人工智慧學校 工百業用AI

Route 加上 Swagger 資訊

python

```python
@app.get(
  "/resource",
  tags=["Resource"],
  summary="取得資源狀態",
  description="確認伺服器資源的狀態是否正常。",
  response_description="成功時回傳確認訊息"
)
```

Restful API 資料驗證

台灣人工智慧學校 工百業用AI

使用 Rest 風格撰寫 API，Pydantic 驗證

RESTful API (GET)

python

```
import uvicorn
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List

app = FastAPI()

class Fruit(BaseModel):
   id: int
   name: str
   description: str = None
   price: float
   on_offer: bool = False
```

RESTful API (GET)

python

```python
fake_db = {
    1: Fruit(id=1, name="香蕉", description="這是香蕉", price=41.9, on_offer=True),
    2: Fruit(id=2, name="蘋果", description="這是蘋果", price=36.0, on_offer=False),
    3: Fruit(id=3, name="芭樂", description="這是芭樂", price=39.7, on_offer=True),
}

@app.get("/fruit", response_model=List[Fruit], tags=["Fruit"])
def query_Fruits():
    return list(fake_db.values())

if __name__ == "__main__":
    uvicorn.run("FastAPI_Restful:app", host="127.0.0.1", port=8000, reload=True)
```

RESTful API (GET ByID)

python

```python
@app.get("/fruit/{fruit_id}", response_model=Fruit, tags=["Fruit"])
def query_Fruit(fruit_id: int):
    if fruit_id not in fake_db:
        raise HTTPException(status_code=404, detail="Fruit not found")
    return fake_db[fruit_id]
```

RESTful API (POST)

python

```python
@app.post("/fruit", response_model=Fruit, tags=["Fruit"])
def create_Fruit(fruit: Fruit):
    if any(existing_fruit.name == fruit.name for existing_fruit in fake_db.values()):
        raise HTTPException(status_code=400, detail="fruit already exists")
    fake_db[fruit.id] = fruit
    return fruit
```

RESTful API (PUT)

python

```python
@app.put("/fruit/{fruit_id}", response_model=Fruit, tags=["Fruit"])
def update_Fruit(fruit_id: int, fruit: Fruit):
    if fruit_id not in fake_db:
        raise HTTPException(status_code=404, detail="Fruit not found")
    fake_db[fruit_id] = fruit
    return fruit
```

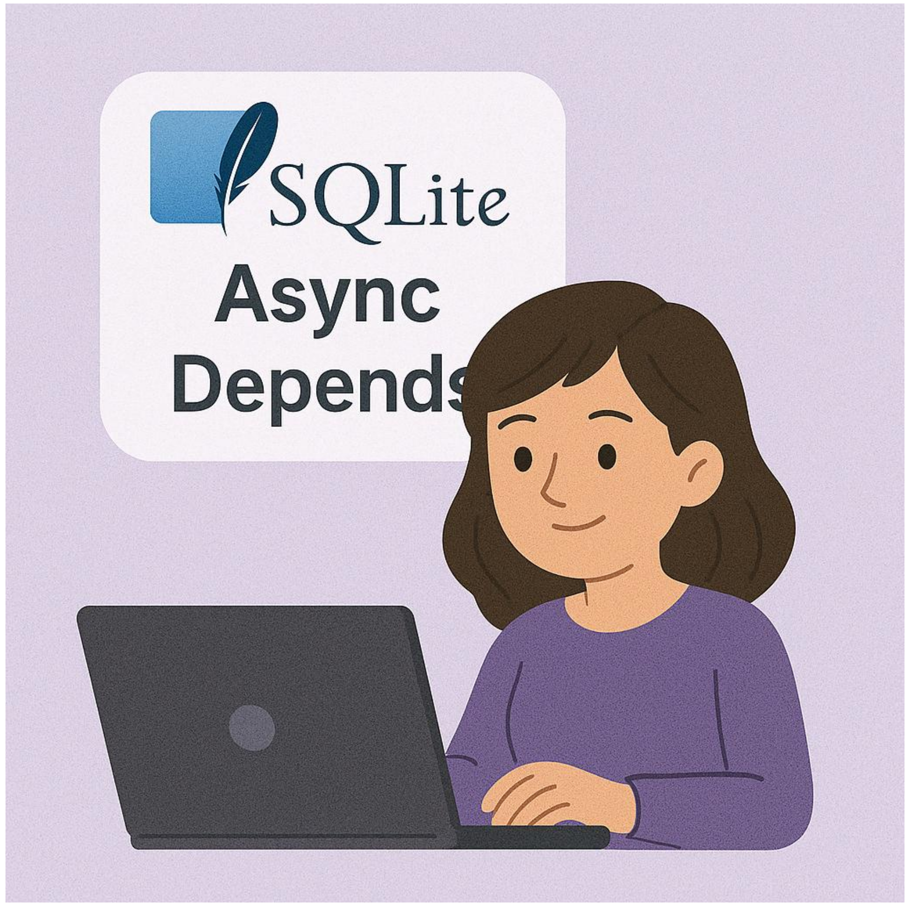
RESTful API (DELETE)

python

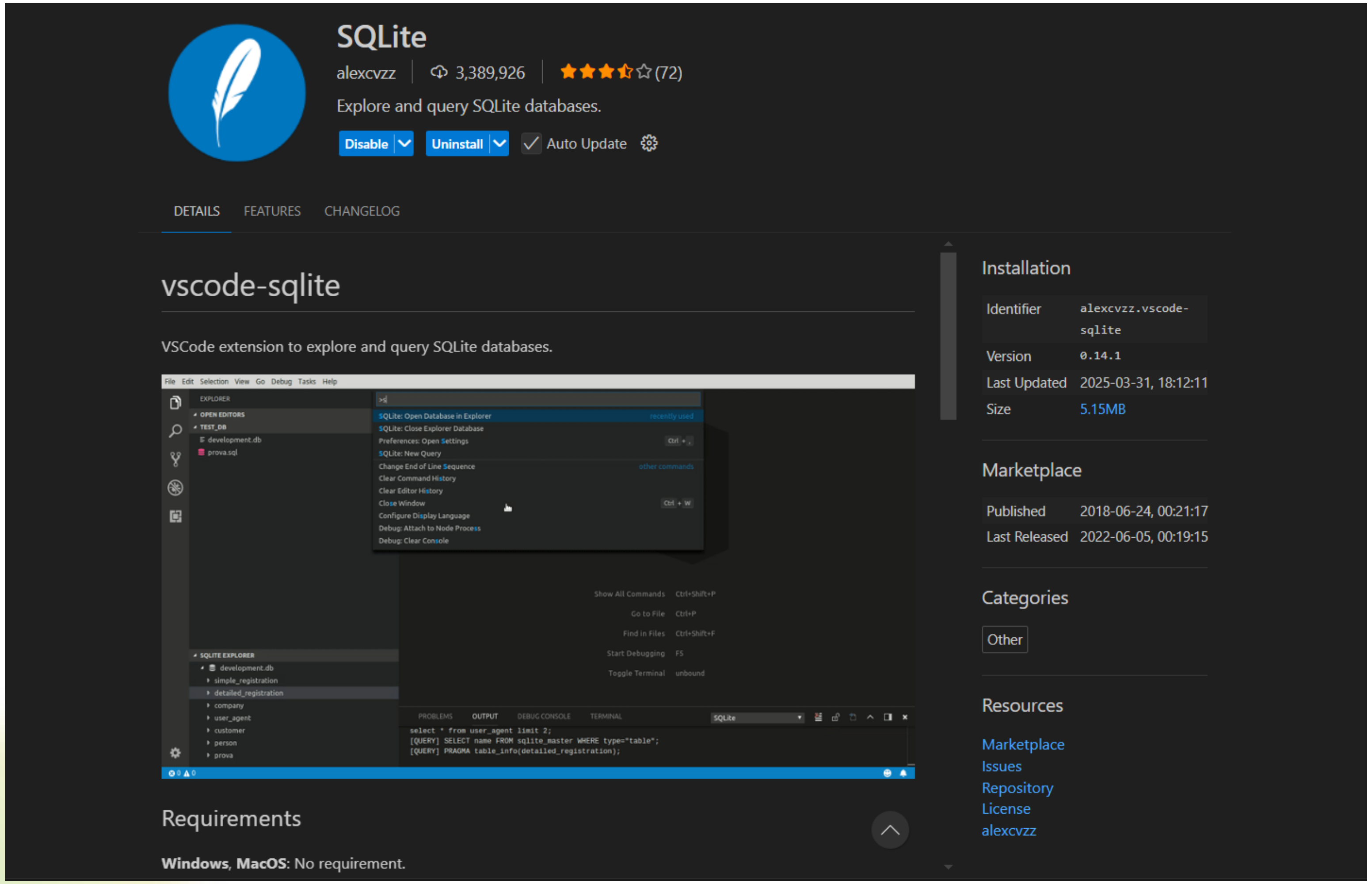
```python
@app.delete("/fruit/{fruit_id}", tags=["Fruit"])
def delete_Fruit(fruit_id: int):
    if fruit_id not in fake_db:
        raise HTTPException(status_code=404, detail="Fruit not found")
    del fake_db[fruit_id]
    return {"message": "Fruit deleted successfully"}
```

依賴注入非同步連線資料庫

依賴注入非同步資料庫



安裝 SQLite



SQLite

alexcvzz | 3,389,926 | (72)

Explore and query SQLite databases.

vscode-sqlite

VSCode extension to explore and query SQLite databases.

```
select * from user_agent limit 2;
[QUERY] SELECT name FROM sqlite_master WHERE type="table";
[QUERY] PRAGMA table_info(detailed_registration);
```

Requirements

Windows, **MacOS**: No requirement.

Installation

Identifier	alexcvzz.vscode-sqlite
Version	0.14.1
Last Updated	2025-03-31, 18:12:11
Size	5.15MB

Marketplace

Published	2018-06-24, 00:21:17
Last Released	2022-06-05, 00:19:15

Categories

Other

Resources

Marketplace
Issues
Repository
License
alexcvzz

建立資料庫

python

```python
import sqlite3

conn = sqlite3.connect("test.db")
cursor = conn.cursor()

cursor.execute('''
CREATE TABLE IF NOT EXISTS fruit (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT,
    price REAL NOT NULL,
    on_offer BOOLEAN DEFAULT 0
)
''')
```

```python
cursor.executemany('''
INSERT INTO fruit (name, description, price, on_offer)
VALUES (?, ?, ?, ?)
''', [
    ('香蕉', '這是香蕉', 41.9, True),
    ('蘋果', '這是蘋果', 36.0, False),
    ('芭樂', '這是芭樂', 39.7, True)
])

cursor.execute("SELECT * FROM fruit")
for row in cursor.fetchall():
    print(row)
conn.commit()
conn.close()
```

安裝必須套件

```python
pip install sqlalchemy aiosqlite
```

建立FastAPI框架

python

```python
import uvicorn
from fastapi import FastAPI, HTTPException, Depends, Response
from pydantic import BaseModel
from typing import List , Optional
from sqlalchemy import Column, Integer, String, Float, Boolean
from sqlalchemy.ext.asyncio import create_async_engine, AsyncSession
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from sqlalchemy.future import select

app = FastAPI()

if __name__ == "__main__":
  uvicorn.run("FastAPI_DB:app", host="127.0.0.1", port=8000, reload=True)
```

設定資料庫連線

python

```python
DATABASE_URL = "sqlite+aiosqlite:///./test.db"
engine = create_async_engine(DATABASE_URL, echo=True)
SessionLocal = sessionmaker(bind=engine, class_=AsyncSession, expire_on_commit=False)
```

建立資料庫模型(Sqlalchemy)

python

```
Base = declarative_base()

class Fruit(Base):
    __tablename__ = "fruit"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    description = Column(String, default=None)
    price = Column(Float)
    on_offer = Column(Boolean, default=False)
```

定義Pydantic模型

python

```python
class FruitCreate(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
    on_offer: bool = False

    class Config:
        orm_mode = True
```

```python
class FruitRead(BaseModel):
    id: int
    name: str
    description: Optional[str] = None
    price: float
    on_offer: bool

    class Config:
        orm_mode = True
```

定義資料庫注入

python

```python
async def get_db():
    async with SessionLocal() as session:
        yield session
```

實作 GET GETByID

python

```python
@app.get("/fruit", response_model=List[FruitRead], tags=["Fruit"])
async def query_Fruits(db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(Fruit))
    fruits = result.scalars().all()
    return fruits

@app.get("/fruit/{fruit_id}", response_model=FruitRead, tags=["Fruit"])
async def query_Fruit(fruit_id: int, db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(Fruit).filter(Fruit.id == fruit_id))
    fruit = result.scalars().first()
    if not fruit:
        raise HTTPException(status_code=404, detail="Fruit not found")
    return fruit
```

實作 POST

```python
@app.post("/fruit", response_model=FruitRead, tags=["Fruit"])
async def create_Fruit(fruit: FruitCreate, db: AsyncSession = Depends(get_db)):
  result = await db.execute(select(Fruit).filter(Fruit.name == fruit.name))
  existing_fruit = result.scalars().first()
  if existing_fruit:
    raise HTTPException(status_code=400, detail="Fruit already exists")
  db_fruit = Fruit(name=fruit.name, description=fruit.description, price=fruit.price,
on_offer=fruit.on_offer)
  db.add(db_fruit)
  await db.commit()
  await db.refresh(db_fruit)
  return db_fruit
```

實作 PUT

python

```python
@app.put("/fruit/{fruit_id}", response_model=FruitRead, tags=["Fruit"])
async def update_Fruit(fruit_id: int, fruit: FruitCreate, db: AsyncSession = Depends(get_db)):
    db_fruit = await db.execute(select(Fruit).filter(Fruit.id == fruit_id))
    db_fruit = db_fruit.scalars().first()
    if not db_fruit:
        raise HTTPException(status_code=404, detail="Fruit not found")
    db_fruit.name = fruit.name
    db_fruit.description = fruit.description
    db_fruit.price = fruit.price
    db_fruit.on_offer = fruit.on_offer
    await db.commit()
    await db.refresh(db_fruit)
    return db_fruit
```

實作 DELETE

python

```python
@app.delete("/fruit/{fruit_id}", status_code=204, tags=["Fruit"])
async def delete_Fruit(fruit_id: int, db: AsyncSession = Depends(get_db)):
    db_fruit = await db.execute(select(Fruit).filter(Fruit.id == fruit_id))
    db_fruit = db_fruit.scalars().first()
    if not db_fruit:
        raise HTTPException(status_code=404, detail="Fruit not found")

    await db.delete(db_fruit)
    await db.commit()
    return Response(status_code=204)
```