台灣人工智慧學校 工百業用AI

Web Framework FastAPI

Daniel
chris

Agenda

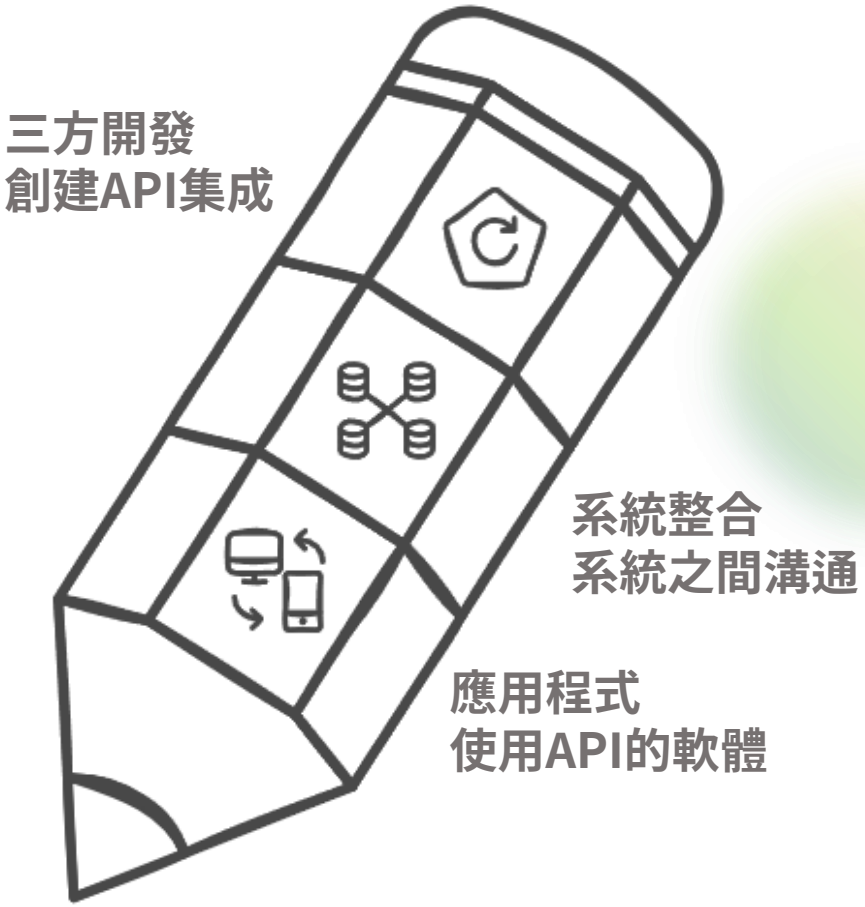
- FastAPI 入門與基礎概念
- 非同步程式設計與依賴注入
- Server-Sent Events (SSE) 的應用
- 實現 WebSocket 通訊

台灣人工智慧學校 工百業用AI

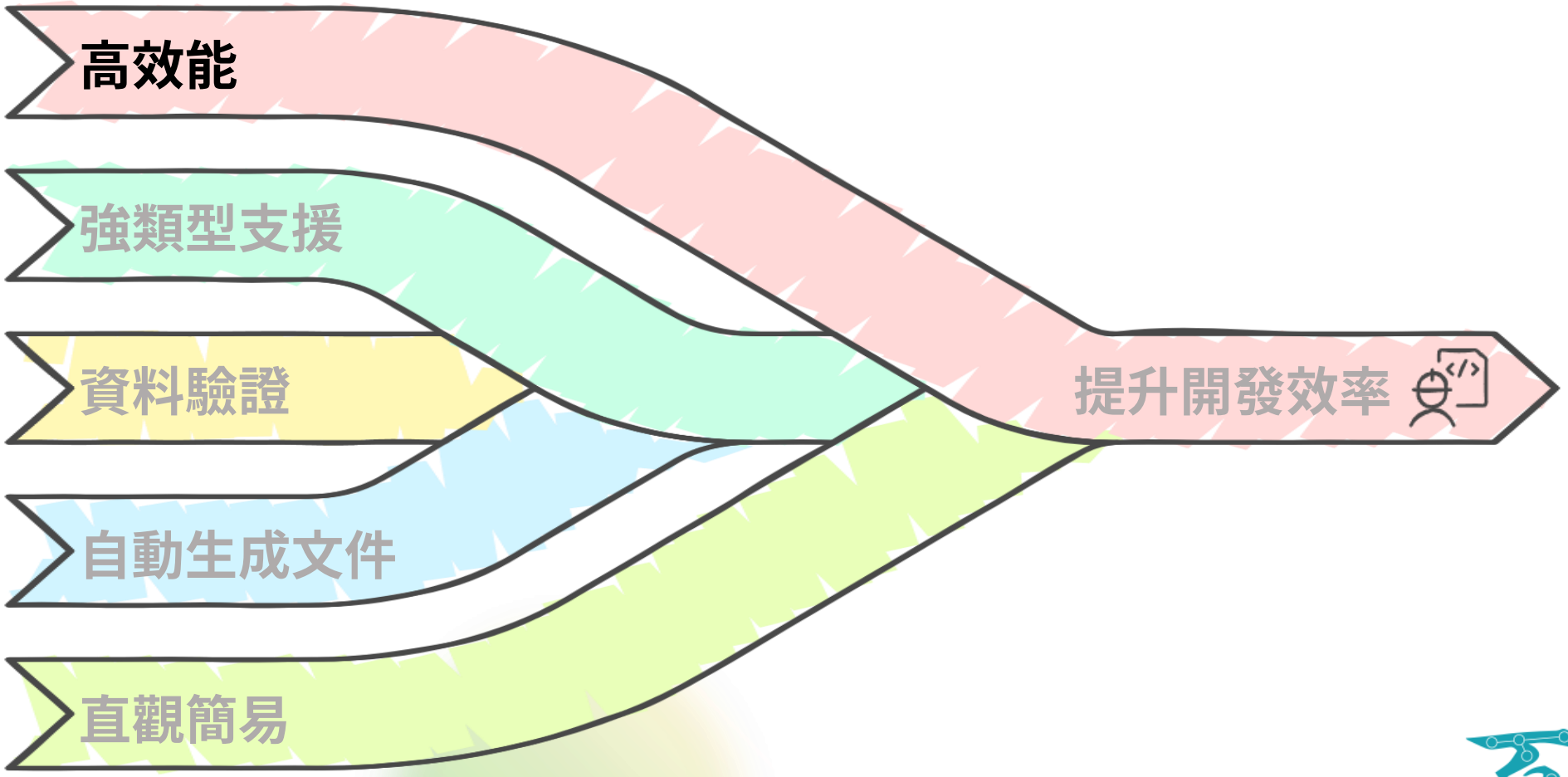
FastAPI入門與基礎概念

API 是什麼?

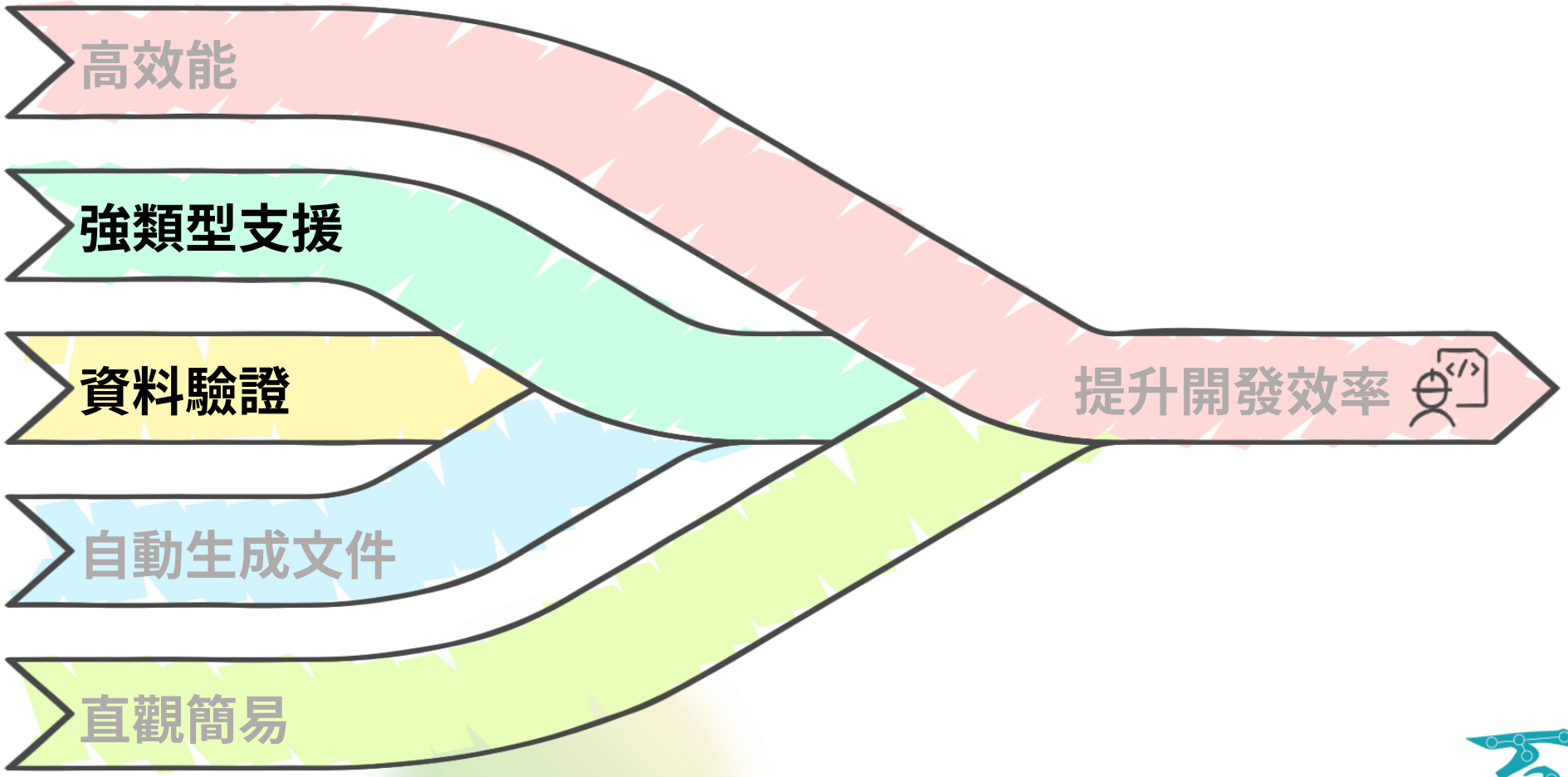
- 全名 Application Programming Interface (應用程式介面)
- 開發出的一種接口，讓第三方可以額外開發、應用在自身的產品上的系統溝通介面。



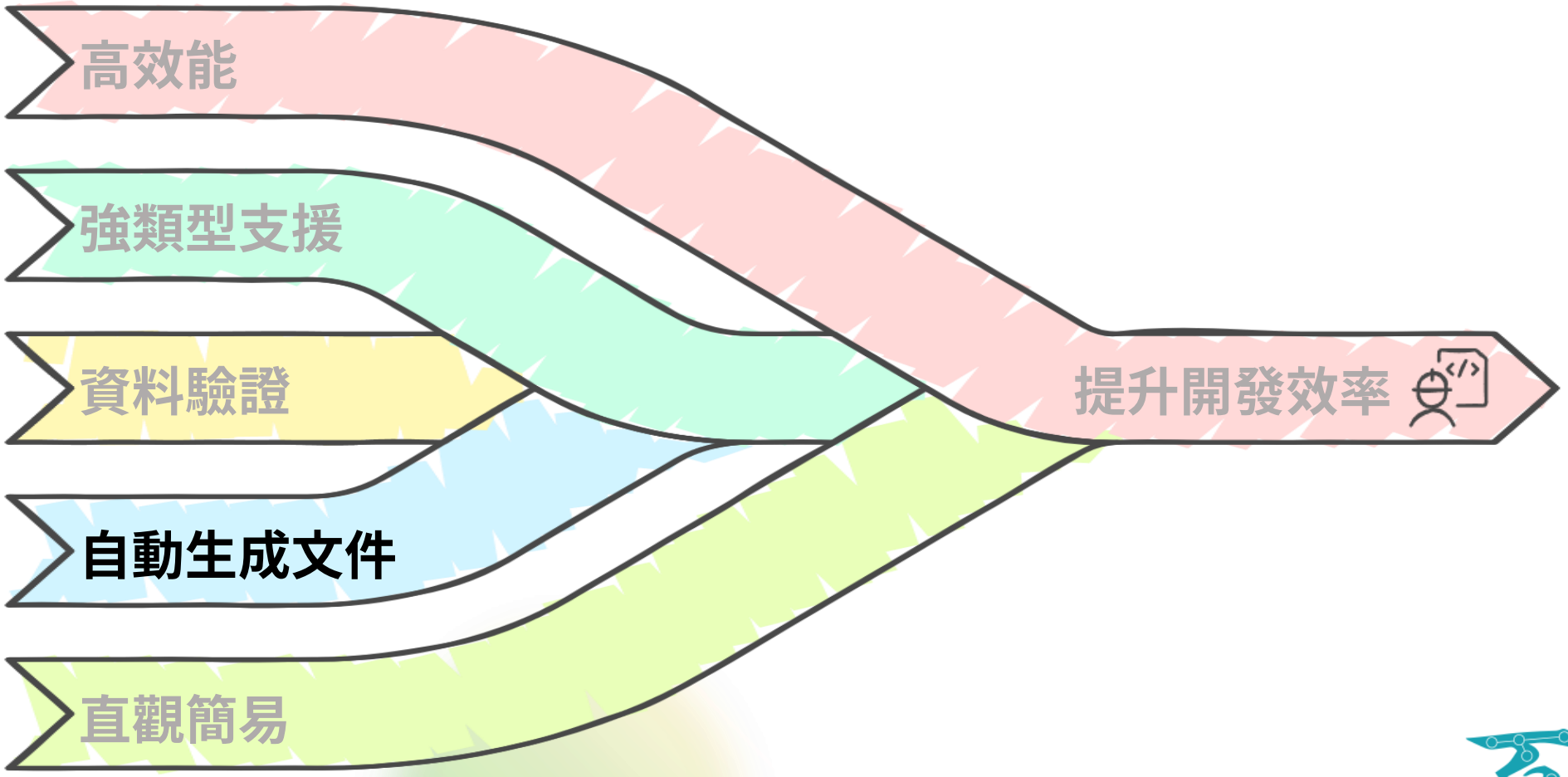
FastAPI 框架介紹：框架特性與優勢



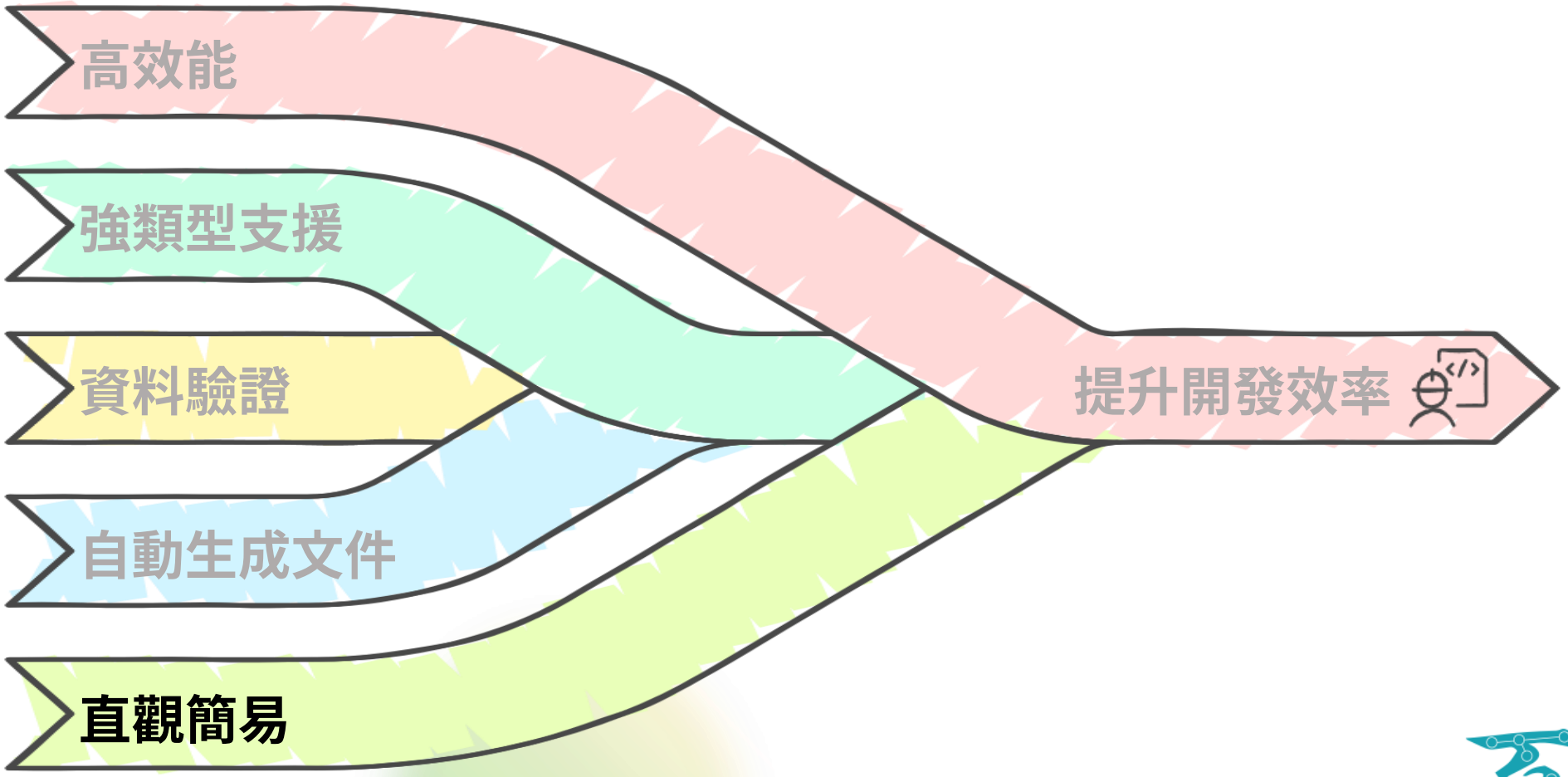
FastAPI 框架介紹：框架特性與優勢



FastAPI 框架介紹：框架特性與優勢

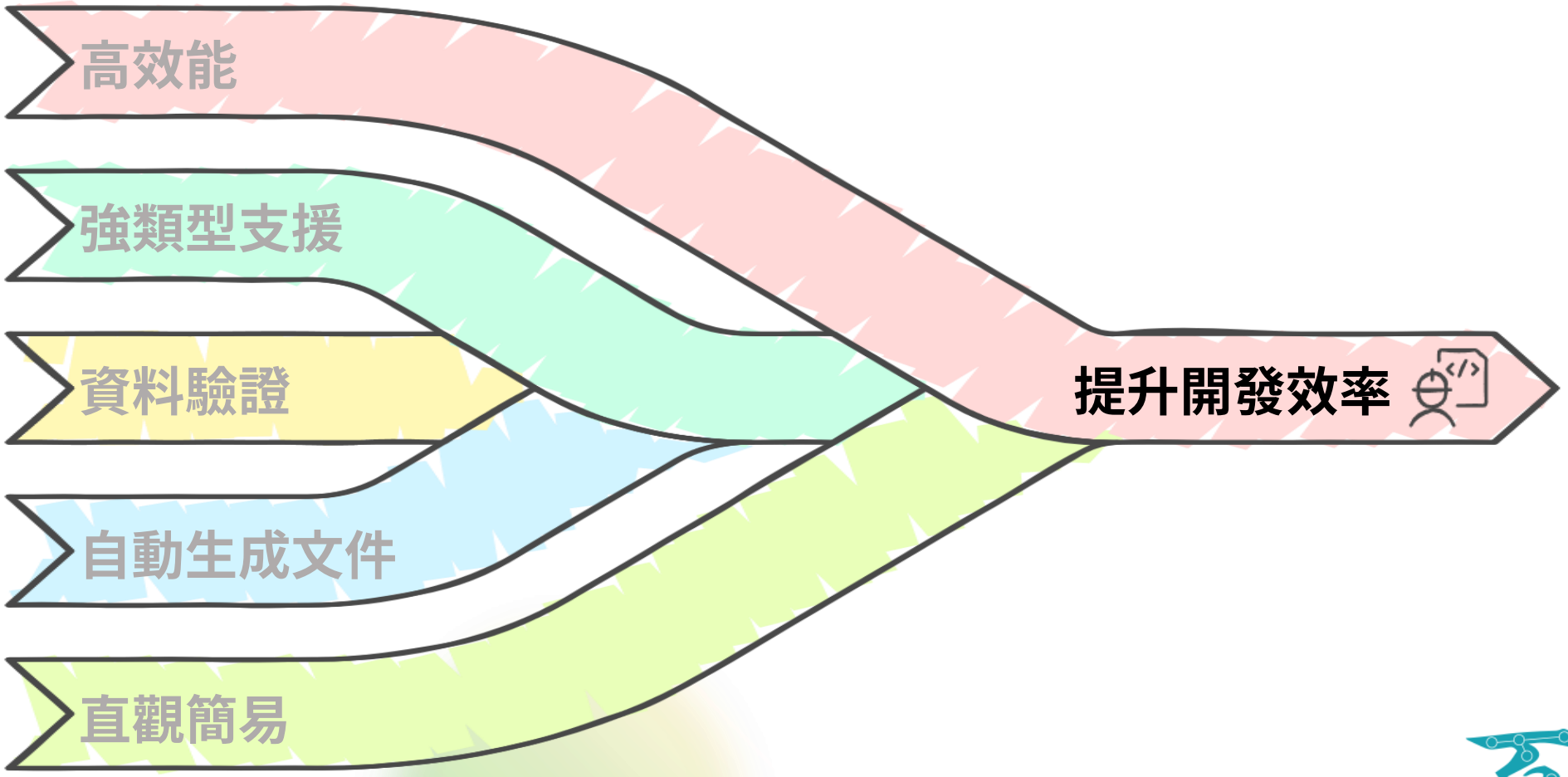


FastAPI 框架介紹：框架特性與優勢



FastAPI學習

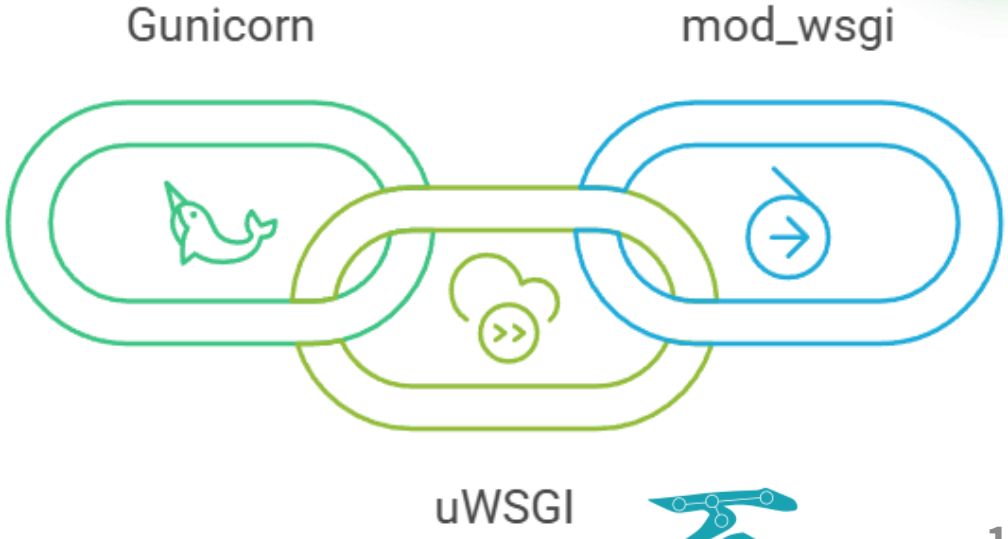
FastAPI 框架介紹：框架特性與優勢



WSGI (Web Server Gateway Interface)

- 什麼是 WSGI？
  - WSGI 是 Python Web 應用程序與 Web 伺服器之間的 同步通信標準。

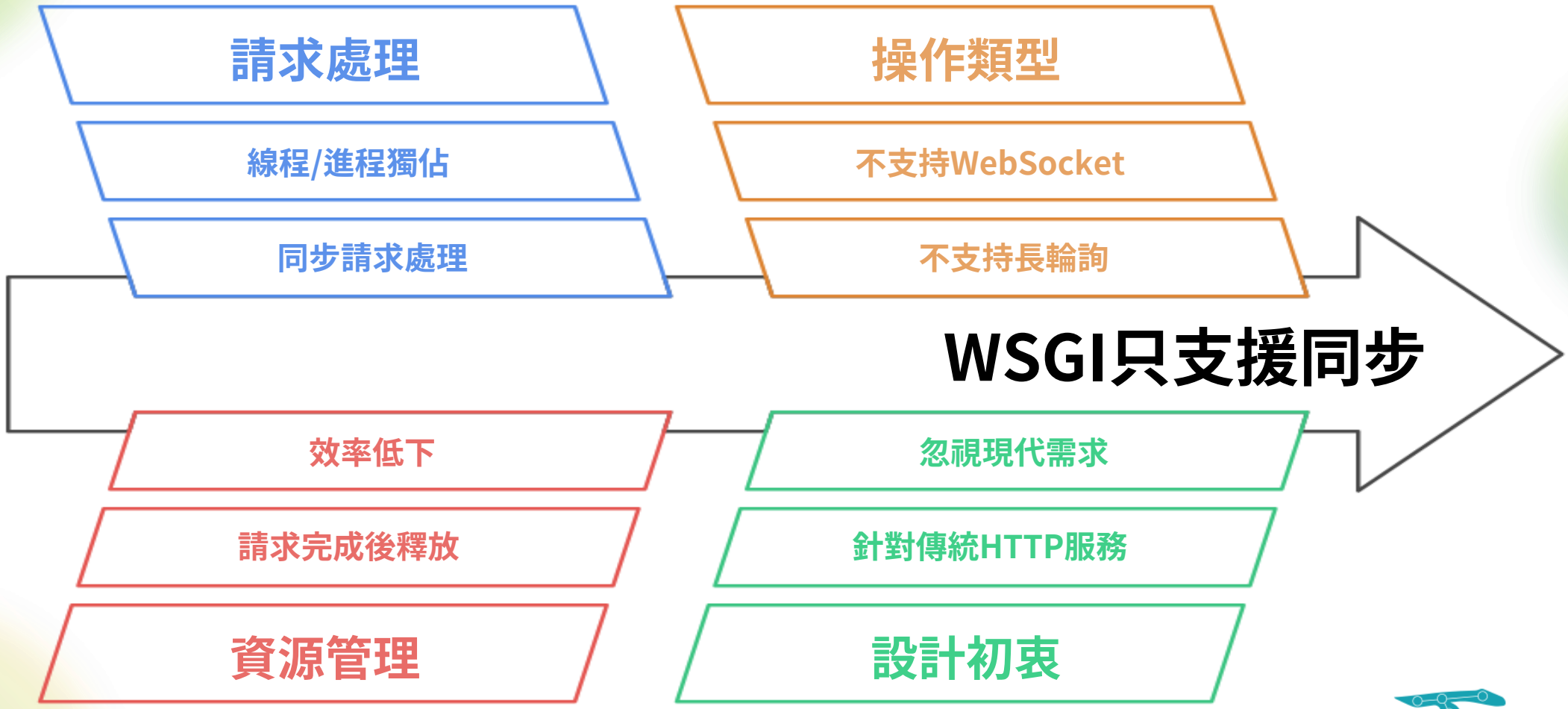
    它讓應用程式能夠與不同的 Web 伺服器兼容，例如：
    - Gunicorn（常用於 Flask 和 Django）
    - uWSGI
    - mod_wsgi（Apache 模組）



WSGI (Web Server Gateway Interface)

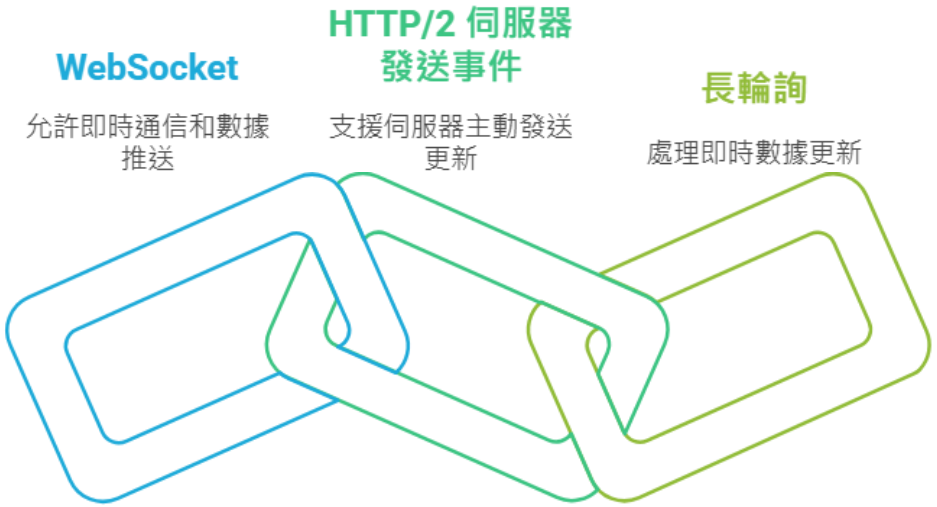
- WSGI 的工作方式

WSGI (Web Server Gateway Interface)



ASGI (Asynchronous Server Gateway Interface)

- 什麼是 ASGI？
  - ASGI 是 WSGI 的進化版，支援 同步 和 異步 的通信方式。
  - 解決 WSGI 只能處理同步請求的問題，適用於現代的 Web 技術。

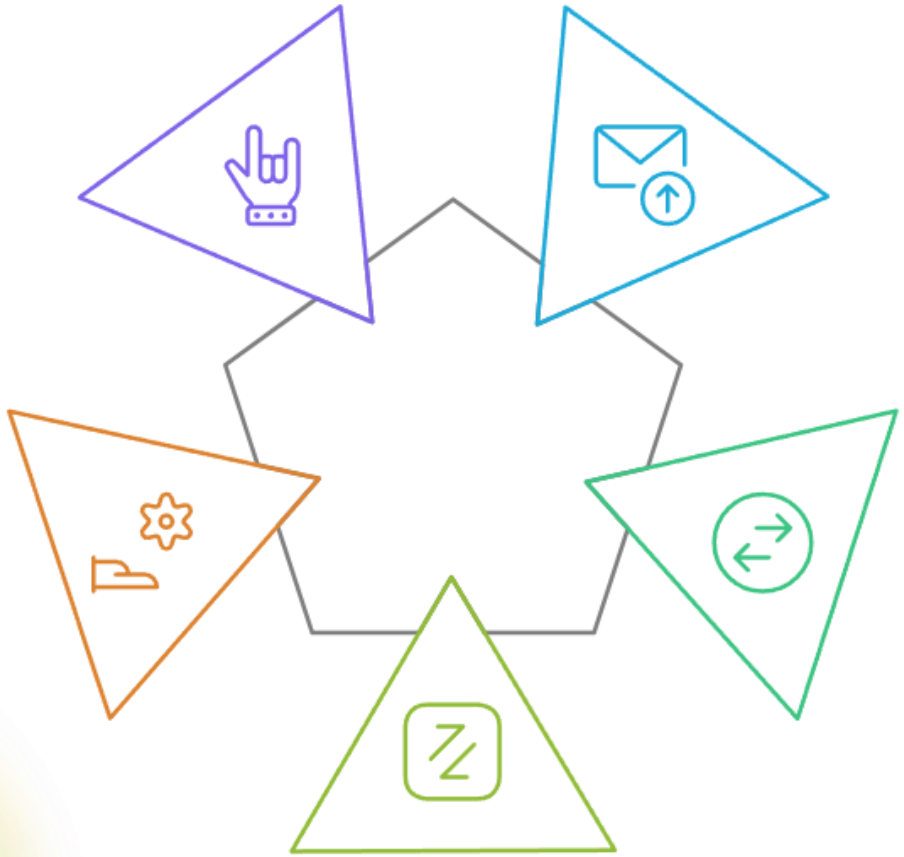


ASGI
(Asynchronous Server Gateway Interface)

- ASGI 的工作方式

ASGI
(Asynchronous Server Gateway Interface)

- ASGI 支援同步 & 異步



傳統 HTTP

處理標準網絡請求

WebSocket

實現雙向即時通信

SSE

用於單向實時更新

背景任務

管理後台處理

高並發

支持大量同時用戶

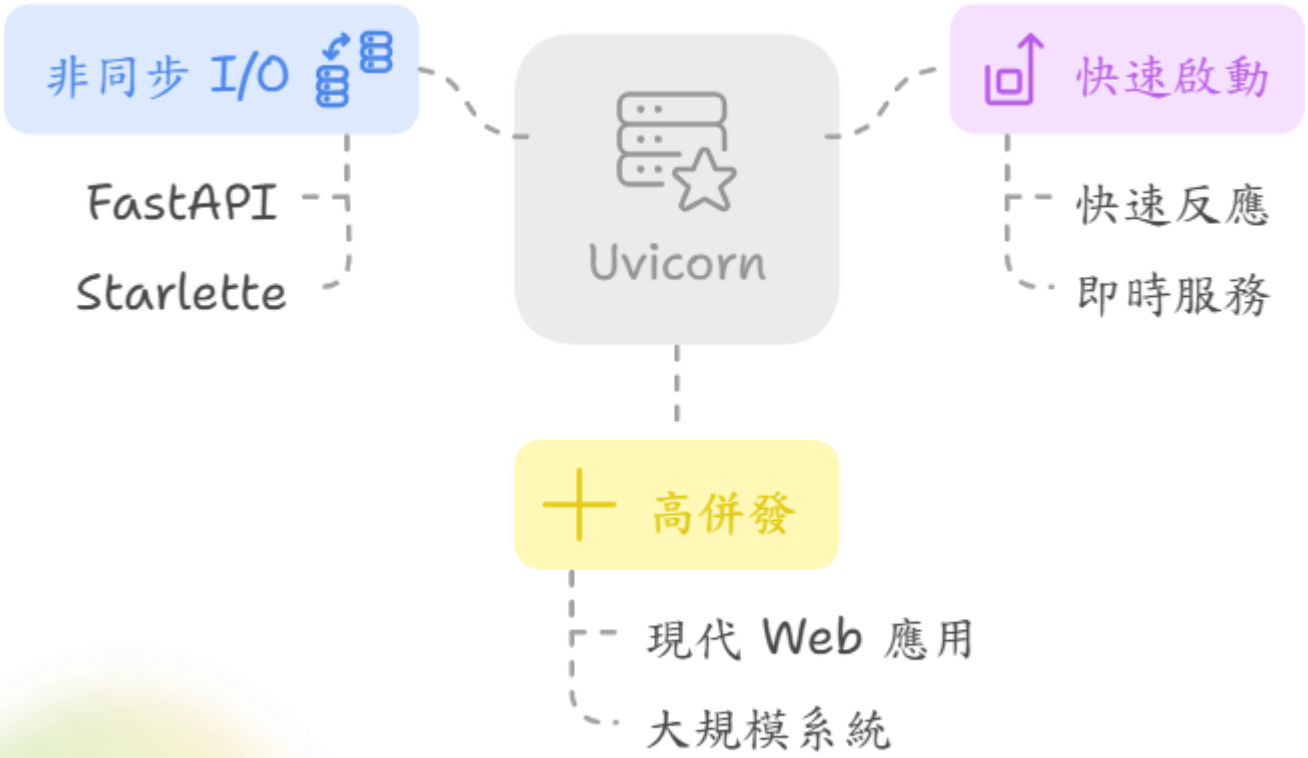
WSGI vs ASGI 的對比

特性	同步	異步	WebSocket	場景	伺服器	框架
ASGI	✓	✓	✓	高併發 即時應用	Uvicorn Daphne	FastAPI Django Channels
WSGI	✓	✗	✗	一般 HTTP 服務	Gunicorn uWSGI	Flask Django

FastAPI vs Flask vs Django

	FASTAPI	Flask	Django
框架類型	高效能API框架	微服務框架	全功能框架
框架特點	注重效能 支援非同步I/O	簡單設計 靈活自由	嚴格設計 Django ORM
內建功能	自動生成 API文件	無內建 需額外擴充	ORM、驗證、 模板引擎
學習曲線	簡單	平緩	陡峭
框架效能	非常高 適合高效能應用	較低 適合靜態應用	良好 適合全功能應用

Uvicorn

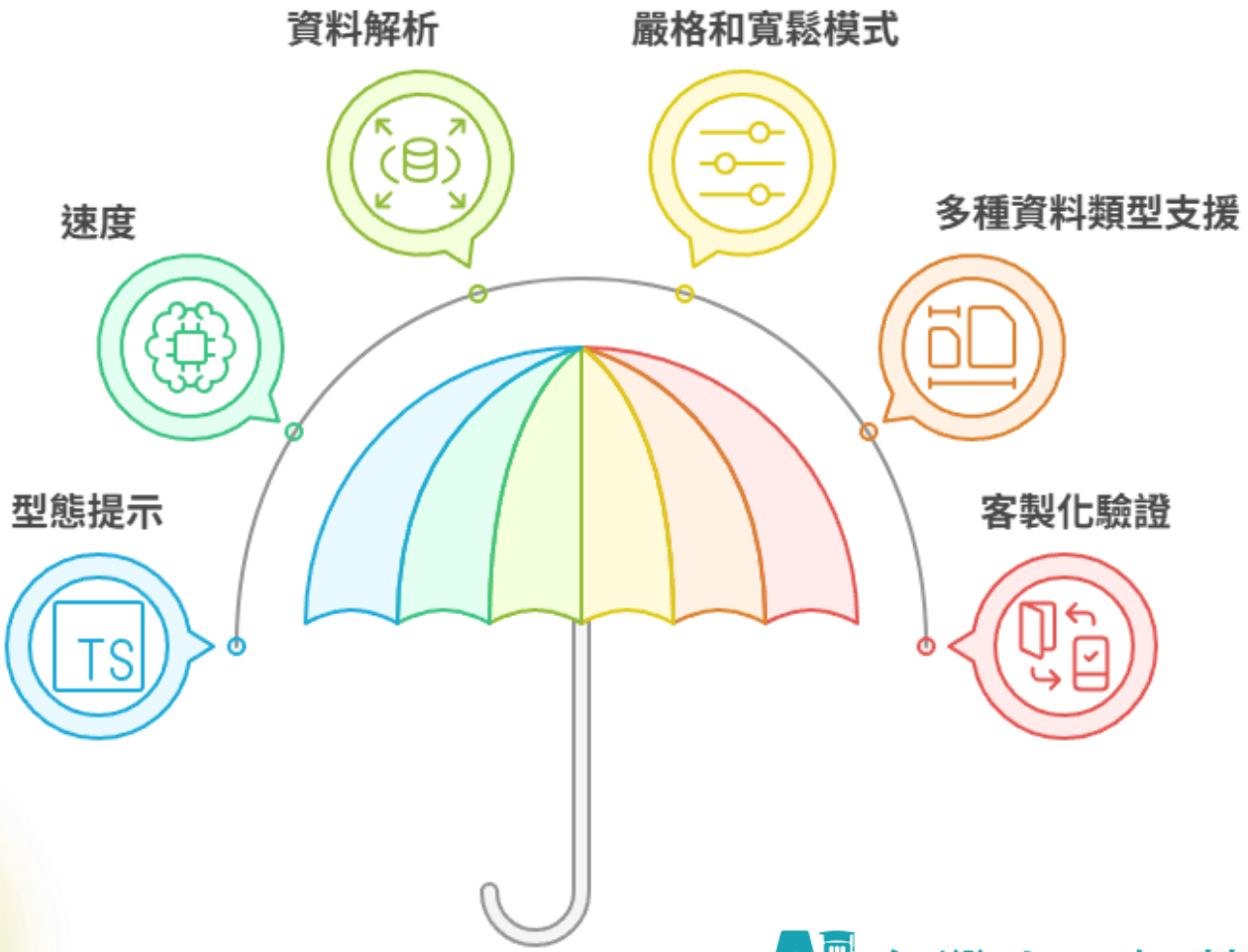


FastAPI實作基礎框架(GET)

python

```
pip install fastapi uvicorn

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)
```

pydantic

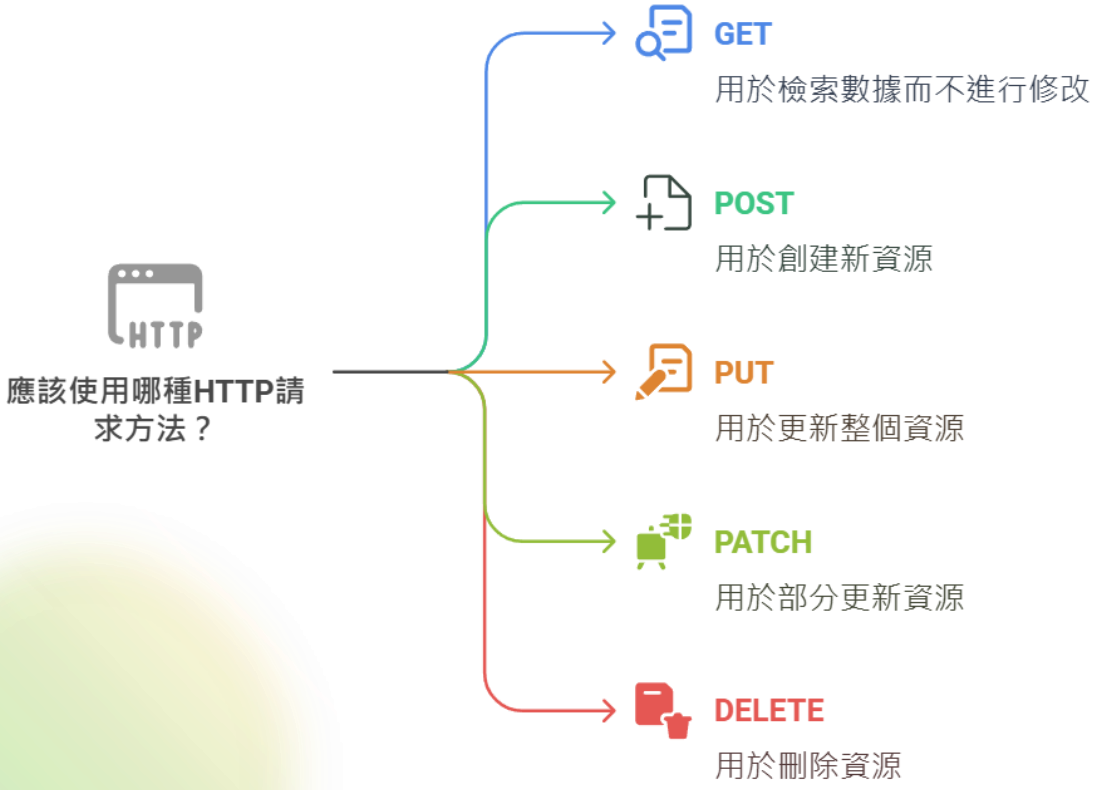


HTTP協定(HyperText Transfer Protocol)

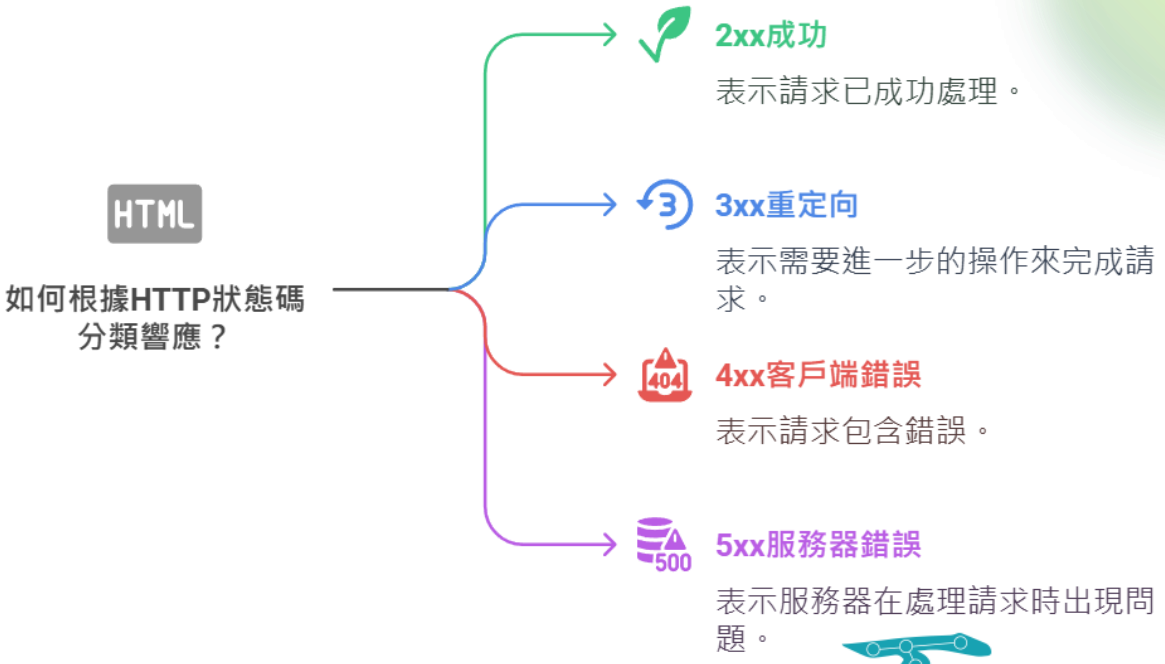
- **用於傳輸超文本資料的應用層協定。**

HTTP請求方法

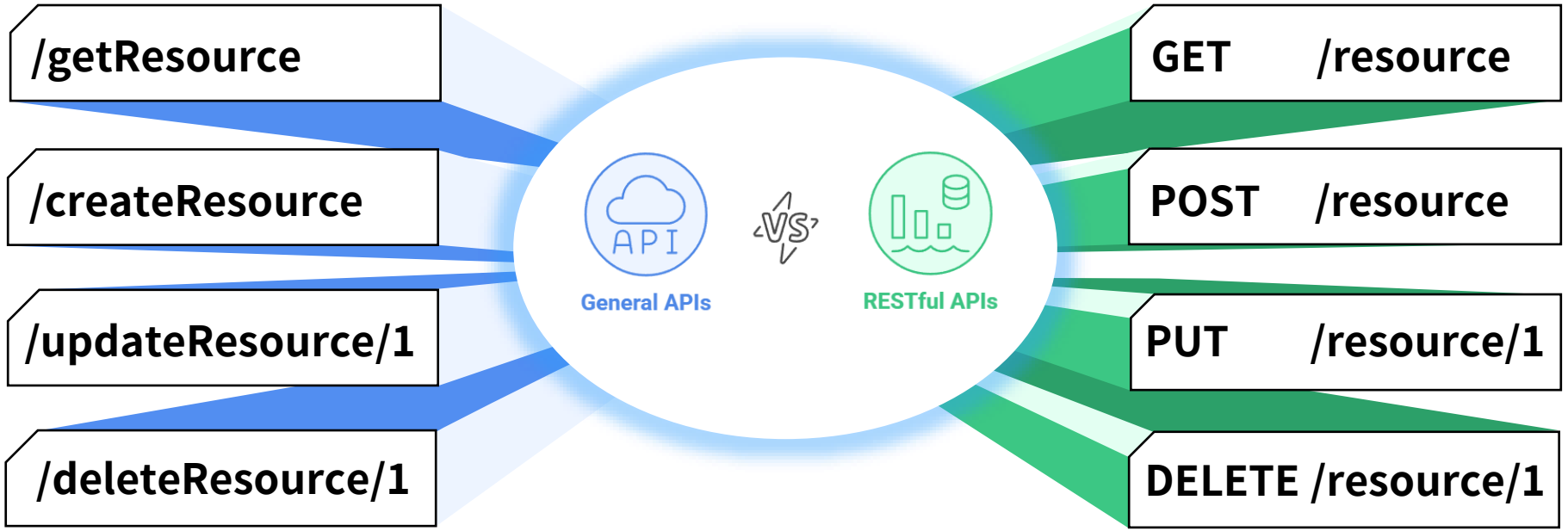
- 常見的請求方法CRUD。



- 狀態碼分為以下幾種：



RESTful API設計風格



Swagger/OpenAPI

- SmartBear Software
- OpenAPI Initiative
- 公共可用性

FastAPI實作RESTful API(GET)

python

```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
```

FastAPI實作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)
```

FastAPI實作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]
```

FastAPI實作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
```

FastAPI實作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
```

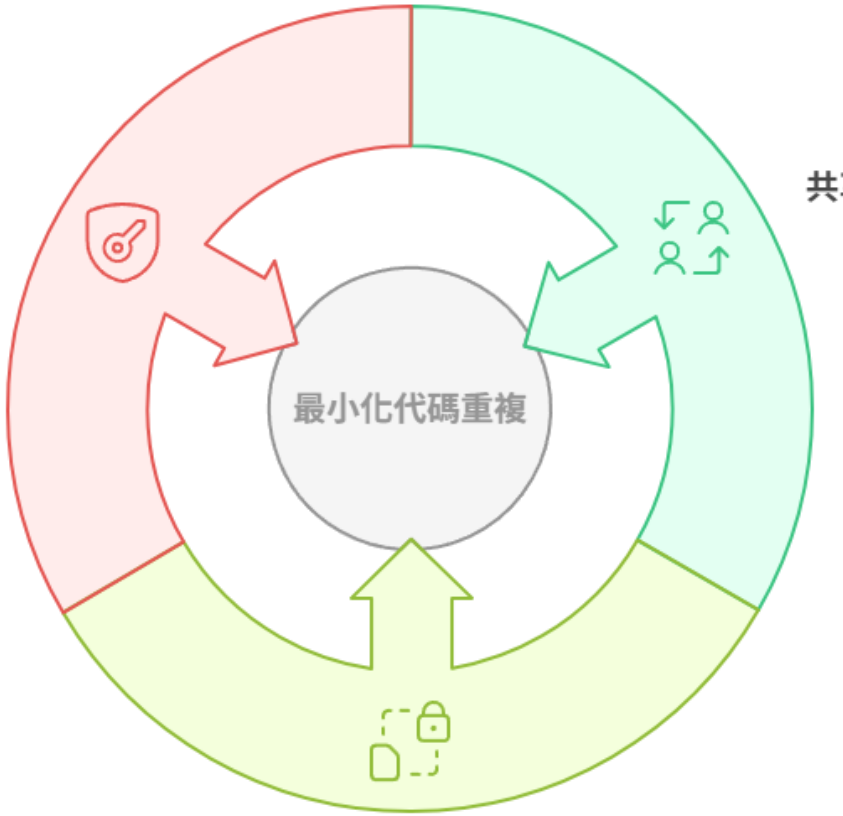
FastAPI實作RESTful API(DELETE)

python

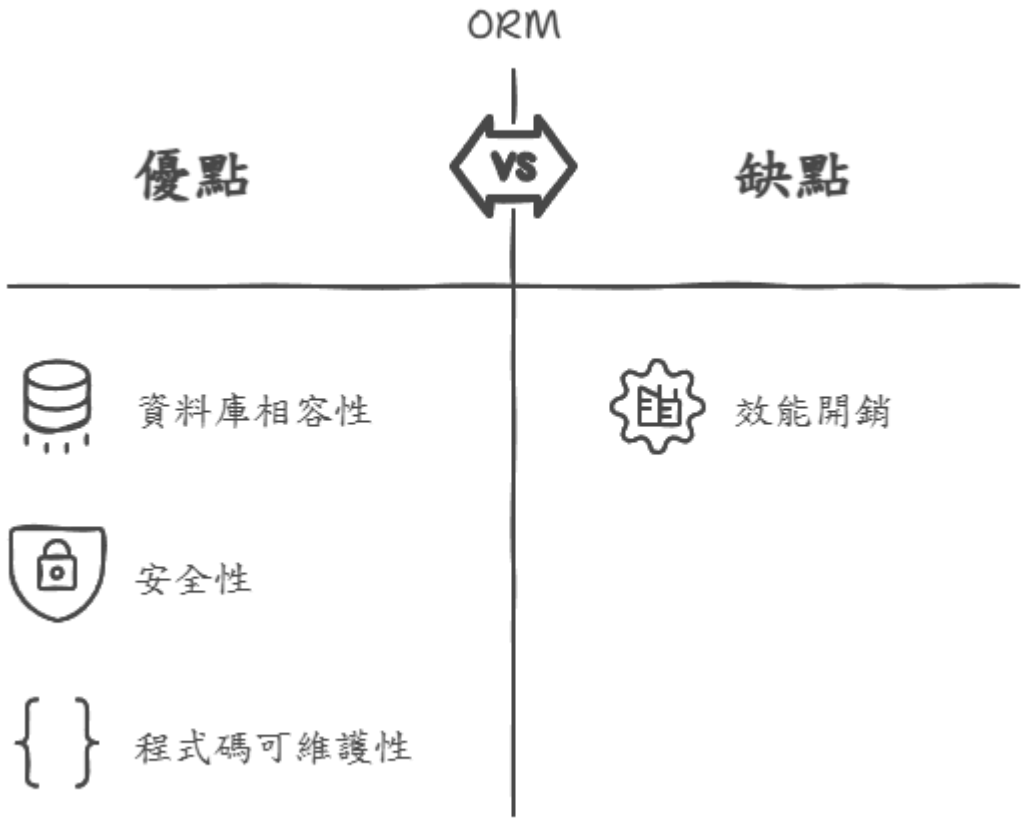
```python
@app.delete("/fruit/{fruit_id}", tags=["Fruit"])
def delete_Fruit(fruit_id: int):
    if fruit_id not in fake_db:
        raise HTTPException(status_code=404, detail="Fruit not found")
    del fake_db[fruit_id]
    return {"message": "Fruit deleted successfully"}
```

非同步程式設計與依賴注入

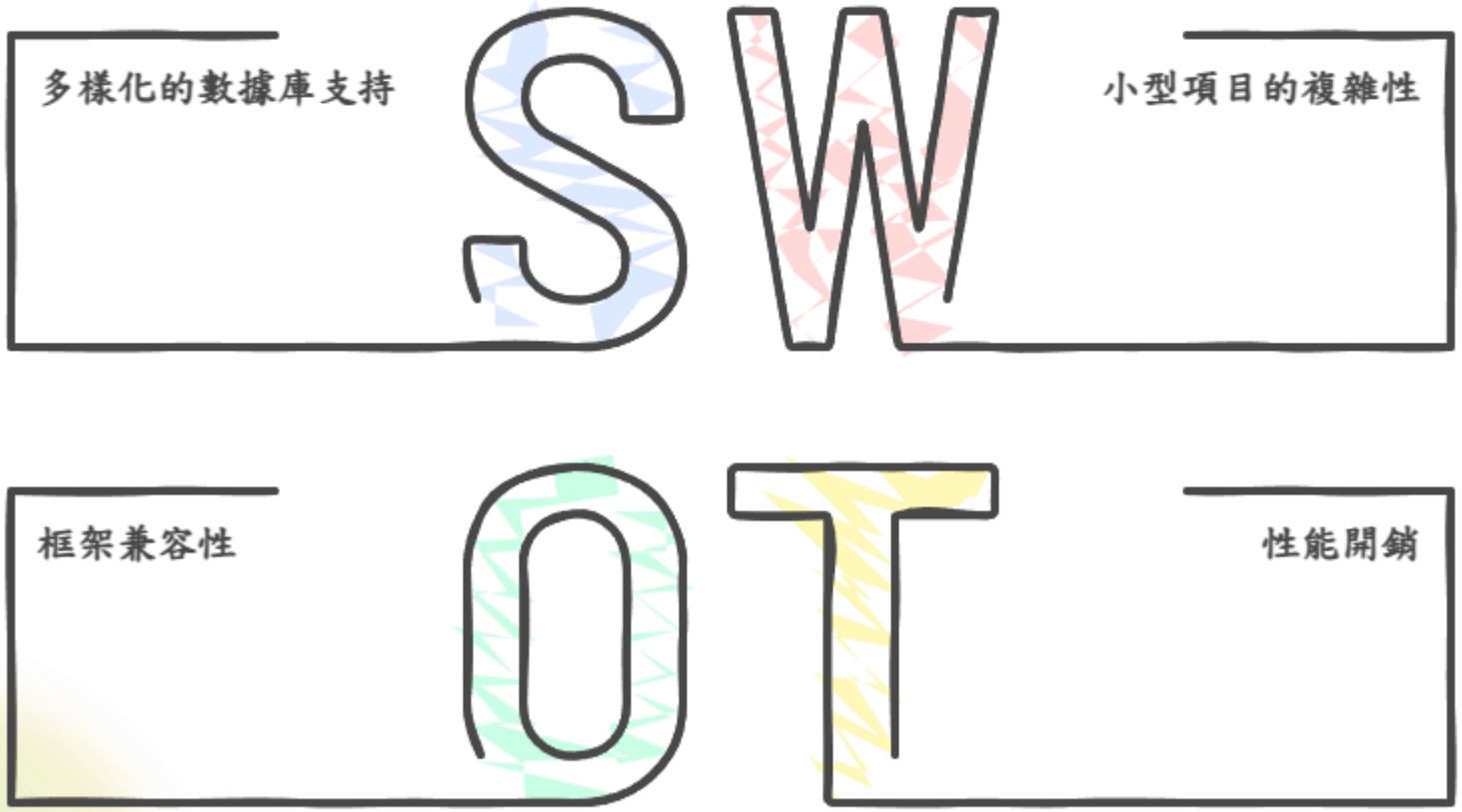
依賴注入



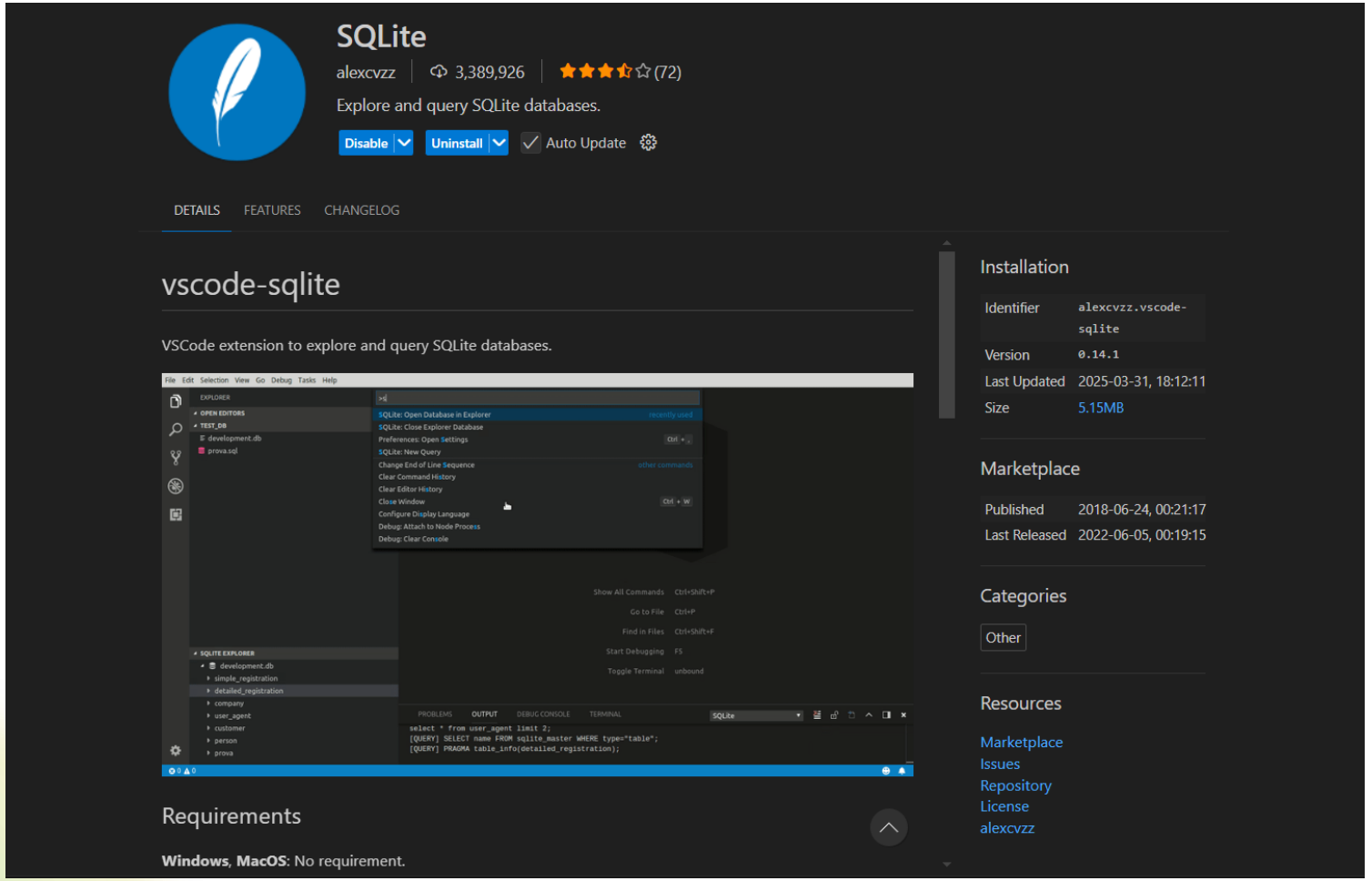
ORM(Object-Relational Mapping)



SQLAlchemy



安裝SQLite



FastAPI實作連接資料庫(建立DB)

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
)
''')

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()
```

FastAPI實作連接資料庫(套件安裝)

python

```
pip install sqlalchemy aiosqlite
```

FastAPI實作連接資料庫(建立框架)

```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)
```

FastAPI實作連接資料庫(設定資料庫)

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)
```

FastAPI實作連接資料庫(建立資料庫模型)

python

```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)
```

FastAPI實作連接資料庫(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
```

FastAPI實作連接資料庫(設定資料庫)

python

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

FastAPI實作連接資料庫(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
```

FastAPI實作連接資料庫(POST)

python

```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
```

FastAPI實作連接資料庫(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
```

FastAPI實作連接資料庫(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)
```

補充(Postman)