🎤 FastAPI 프레임워크 이해하기

👋 현대적인 API 개발을 위한 강력한 프레임워크

⚡ 발표자 : 채다현

🚀 FastAPI의 핵심 개념

✅ FastAPI를 제대로 이해하려면?

- 동기(Sync) vs 비동기(Async) 프로그래밍 이해
- ASGI와 WSGI 차이점 파악
- FastAPI의 핵심 기능 및 장점 분석

🔄 동기(Synchronous) 프로그래밍

- 코드가 작성된 순서대로 실행되며, 하나의 작업이 끝나야 다음 작업이 시작됩니다.
- I/O 작업(예: DB 조회, API 호출)에서 시간이 오래 걸릴 경우 전체 프로그램이 대기 상태가 됩니다.

```
import time

def task(name):
    print(f"{name} 작업 시작")
    time.sleep(3)
    print(f"{name} 작업 완료")

task("A")
task("B")
```

- `task("A")`가 끝날 때까지 `task("B")`는 실행되지 않습니다.

⚡ 비동기(Asynchronous) 프로그래밍

- 하나의 작업이 끝날 때까지 기다리지 않고, 다른 작업을 동시에 실행할 수 있습니다.
- `async`와 `await` 키워드를 사용하여 비동기 처리를 수행합니다.

```
import asyncio

async def task(name):
    print(f"{name} 작업 시작")
    await asyncio.sleep(3)  # 3초 동안 대기하지만, 다른 작업을 실행할 수 있음
    print(f"{name} 작업 완료")

async def main():
    await asyncio.gather(task("A"), task("B"))  # 동시에 실행됨
    print("모든 작업 완료")

asyncio.run(main())
```

- `task("A")`와 `task("B")`가 동시에 실행되므로 전체 실행 시간이 단축됩니다.

🔥 ASGI와 WSGI

- **ASGI**
  - 비동기 요청을 지원하며, 실시간 데이터 처리에 유리
- **WSGI**
  - 동기 기반으로 요청을 순차적으로 처리

특징	ASGI	WSGI
비동기 지원	✅	❌
실시간 데이터 처리	✅	❌
WebSocket 지원	✅	❌
멀티스레드 성능	높음	낮음

🚀 Uvicorn(ASGI 서버)

ASGI 서버인 **Uvicorn**은 FastAPI와 같은 비동기 웹 프레임워크에서 매우 중요한 역할을 합니다.

설치 방법

```
pip install uvicorn
```

FastAPI 애플리케이션 실행

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

📌 Starlette과 Pydantic

- **Starlette**
  - FastAPI의 비동기 웹 서버 프레임워크로, ASGI를 기반으로 동작하며 WebSocket, 백그라운드 태스크 등의 기능을 제공합니다.
- **Pydantic**
  - 데이터 검증과 직렬화를 담당하는 라이브러리로, Python의 타입 힌트를 활용하여 데이터 구조를 정의할 수 있습니다.

🎯 Dependency Injection (의존성 주입)

Dependency Injection은 일반적으로 **DI로** 명칭 하며, **의존성 주입**을 의미합니다.

- **의존성**
  - 클래스나 객체가 다른 클래스나 객체를 사용하는 관계를 의미합니다.
    (A 클래스가 B 클래스를 사용하는 경우, A는 B에 의존한다고 말할 수 있습니다.)
- **의존성 주입**
  - A가 B를 사용해야 할 때, A가 직접 B를 생성하지 않고, 외부에서 B를 주입하는 방식입니다.
    이렇게 하면 A와 B의 결합도가 낮아지고, 더 유연하고 테스트하기 쉬운 코드를 만들 수 있습니다.
    - 생성자 주입 (Constructor Injection)
    - 세터 주입 (Setter Injection)
    - 인터페이스 주입 (Interface Injection)

🚀 1. FastAPI 소개

- 빠르고 간결한 개발
- 비동기 프로그래밍 지원
- 자동화된 API 문서화
- 타입 검증
- 고성능

🔥 FastAPI vs Flask vs Django

항목	FastAPI	Flask	Django
비동기 지원	✅	❌	❌
성능	🚀 매우 빠름	⚡ 보통	⚡ 보통
타입 힌트 지원	✅	❌	❌
자동 문서화	✅	❌	❌
API 지원	✅ 기본 제공	❌ Flask-RESTful 필요	❌ Django REST Framework 필요

🚀 2. FastAPI의 내부 동작 원리

FastAPI의 Request-Response Cycle

1. 클라이언트가 API 요청을 보냄
2. ASGI 서버(Uvicorn)가 요청을 FastAPI 애플리케이션으로 전달
3. FastAPI는 요청을 적절한 엔드포인트로 라우팅
4. 요청 데이터는 **Pydantic**을 사용하여 검증됨
5. **의존성 주입 시스템**이 필요한 인스턴스를 생성 및 주입
6. 비즈니스 로직 실행 후 응답 데이터를 반환
7. FastAPI는 **Pydantic**을 사용하여 데이터를 직렬화하고 응답을 반환

FastAPI의 Dependency Injection 동작 원리

FastAPI의 `Depends()`는 필요한 의존성을 자동으로 주입해주는 방식으로 동작합니다.

```python
from fastapi import Depends, FastAPI

def get_db():
    return "데이터베이스 연결"

app = FastAPI()

@app.get("/items/")
def read_items(db: str = Depends(get_db)): # FastAPI가 자동으로 `get_db()`를 실행
    return {"db": db}                       # 결과를 `db` 매개변수에 주입
```

1. 요청이 들어오면 FastAPI가 `Depends()`를 확인
2. 필요한 의존성을 생성하고 엔드포인트 함수에 주입
3. 요청이 처리된 후 의존성을 정리

FastAPI의 고급 기능 활용하기

(1) 미들웨어(Middleware) 활용

미들웨어는 요청을 처리하기 전에 또는 응답을 반환하기 전에 특정 로직을 실행할 수 있는 기능입니다.

```
from fastapi import FastAPI
from starlette.middleware.base import BaseHTTPMiddleware
import time

class TimingMiddleware(BaseHTTPMiddleware): # 응답 시간 측정
    async def dispatch(self, request, call_next):
        start_time = time.time()
        response = await call_next(request)
        process_time = time.time() - start_time
        response.headers["X-Process-Time"] = str(process_time)
        return response

app = FastAPI()
app.add_middleware(TimingMiddleware)
```

FastAPI의 고급 기능 활용하기

(2) FastAPI 이벤트 시스템 활용

FastAPI는 서버 시작 및 종료 시 실행할 이벤트를 등록할 수 있습니다.

```python
from fastapi import FastAPI

app = FastAPI()

@app.on_event("startup")
async def startup_event():
    print("서버가 시작되었습니다.")

@app.on_event("shutdown")
async def shutdown_event():
    print("서버가 종료됩니다.")
```

- `startup_event` : 서버가 실행될 때 한 번 실행.
- `shutdown_event` : 서버가 종료될 때 한 번 실행.

FastAPI가 사용된 주요 프로젝트 및 기업 사례

- **Netflix**
  - 데이터 API 및 머신러닝 API 구축
- **Uber**
  - 내부 API 및 서비스 관리
- **Microsoft**
  - AI 및 데이터 분석 API
- **Zulip**
  - 실시간 채팅 API
- **GitHub Trends**
  - FastAPI의 사용률이 꾸준히 증가하고 있음

FastAPI로 간단한 REST API 구축

프로젝트 구조

```
my_fastapi_project/
├── main.py             # FastAPI 엔트리포인트
├── models.py           # SQLAlchemy 데이터 모델
├── schemas.py          # Pydantic 데이터 검증
├── database.py         # 데이터베이스 연결 설정
├── routers/
│   ├── users.py        # 사용자 관련 API 엔드포인트
├── requirements.txt    # 프로젝트 의존성 목록
```

FastAPI로 간단한 REST API 구축

1) FastAPI 앱 초기화 (main.py)

```
from fastapi import FastAPI
from routers import users

# FastAPI 앱 인스턴스 생성
app = FastAPI()

# 라우터 등록
app.include_router(users.router)
```

FastAPI로 간단한 REST API 구축

2) 데이터 모델 정의 (models.py)

```
from sqlalchemy import Column, Integer, String
from database import Base

# User 테이블 모델 정의
class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
```

FastAPI로 간단한 REST API 구축

3) 데이터베이스 설정 (database.py)

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# SQLite 데이터베이스 URL 설정
SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

import models # 테이블 생성 위해 models import
Base.metadata.create_all(engine)  # 테이블 생성
```

FastAPI로 간단한 REST API 구축

4) Pydantic을 활용한 데이터 검증 (schemas.py)

```
from pydantic import BaseModel

# 사용자 생성 요청 데이터 검증 모델
class UserCreate(BaseModel):
    name: str
```

FastAPI로 간단한 REST API 구축

5) API 엔드포인트 정의

```
# import 문 생략

router = APIRouter()

def get_db(): # 데이터베이스 세션을 제공하는 의존성 함수
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

@router.post("/users/")  # 사용자 생성 API 엔드포인트
def create_user(user: UserCreate, db: Session = Depends(get_db)):
    db_user = User(name=user.name)
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user
```

FastAPI로 간단한 REST API 구축

6) FastAPI 실행 방법

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

7) 프로젝트 의존성 파일 (requirements.txt)

```
fastapi      # 빠르고 효율적인 웹 API 서버 구축을 위한 프레임워크
uvicorn      # ASGI 서버, FastAPI 앱을 실행하는 데 사용되는 웹 서버
pydantic     # 데이터 검증 및 변환을 위한 라이브러리, FastAPI에서 데이터 모델 정의 시 사용
SQLAlchemy   # ORM(Object Relational Mapping) 라이브러리, 데이터베이스와의 상호작용을 위한 라이브러리
starlette    # FastAPI가 의존하는 ASGI 프레임워크, 라우팅 및 요청 처리 기본 기능 제공
typing_extensions  # Python 3.8 이상의 타입 힌트 기능을 확장하는 라이브러리
```

🎉 7. 결론

✅ FastAPI는 **현대적인 웹 개발을 위한 강력한 프레임워크**
✅ **비동기 처리**와 **자동 문서화** 기능 제공
✅ **미들웨어, 이벤트 시스템 활용 지원**
✅ **Netflix, Uber, Microsoft 등에서 사용 중**
✅ **고성능 API 서버를 구축하기 위한 다양한 최적화 가능**