Dependency Injection by example with FastAPI

Karlsruhe Python Meetup 2023-07-12

Patrick Mühlbauer

@tmuxbee

treebee

solute

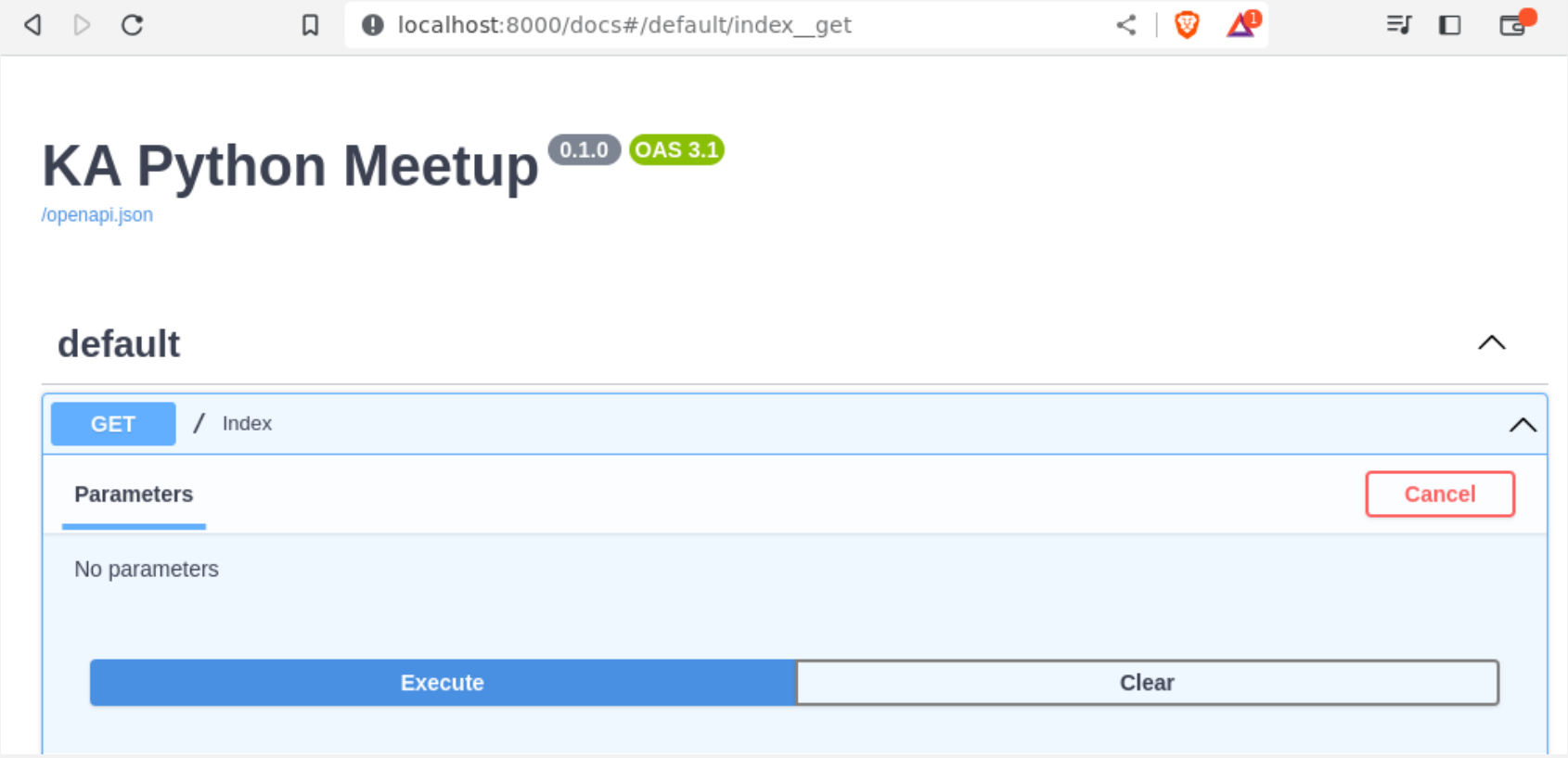
Agenda

- Dependency Injection in FastAPI
- Dependency Injection in general
- Extended Dependency Injection Example

FastAPI

```python
from fastapi import FastAPI

app = FastAPI(title="KA Python Meetup")

@app.get("/")
def index():
    return {"title": "Dependency Injection by Example"}


```

FastAPI


localhost:8000/docs#/default/index__get
KA Python Meetup
0.1.0
OAS 3.1
/openapi.json
default
GET
/ Index
Parameters
Cancel
No parameters
Execute
Clear

FastAPI with SQLAlchemy

```python
from fastapi import FastAPI, HTTPException
import sqlalchemy as sa
from sqlalchemy.orm import sessionmaker

from . import models, schemas

app = FastAPI(title="KA Python Meetup")
engine = sa.create_engine("postgresql://user:password@postgresserver/mydb")
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)


@app.get("/offers/{offer_id}", response_model=schemas.Offer)
def get_offer(offer_id: int):
    db_session = SessionLocal()
    try:
        if (offer := db_session.query(models.Offer).get(offer_id)) is None:
            raise HTTPException(status_code=404, detail="Offer not found")
        return offer
    finally:
        db_session.close()
```

- global state
- session/connection cleanup
- multiple layers (presentation, business, storage) in view functions

FastAPI – Dependencies

```python
import functools
from fastapi import Depends, FastAPI, HTTPException

# ...

@functools.lru_cache
def get_session_factory():
    engine = sa.create_engine("postgresql://user:password@postgresserver/mydb")
    return sessionmaker(autocommit=False, autoflush=False, bind=engine)

def get_db_session(session_factory: sessionmaker = Depends(get_session_factory)):
    db_session = session_factory()
    try:
        yield db_session
    finally:
        db_session.close()

@app.get("/offers/{offer_id}", response_model=schemas.Offer)
def get_offer(offer_id: int, db_session: Session = Depends(get_db_session)):
    if (offer := db_session.query(models.Offer).get(offer_id)) is None:
        raise HTTPException(status_code=404, detail="Offer not found")
    return offer
```

- Database session **injected** into view function as **dependency**
- Dependency handles cleanup
- Dependencies can depend on other dependencies
- Dependencies are simple functions

Centralized Application Config

```python
from pydantic import AmqpDsn, Field, PostgresDsn
from pydantic_settings import BaseSettings, SettingsConfigDict

class Settings(BaseSettings):
    postgres_url: PostgresDsn
    api_key: str = Field( ... )
    amqp_dsn: AmqpDsn

    model_config = SettingsConfigDict(frozen=True)

@functools.lru_cache
def get_settings():
    return Settings()

@functools.lru_cache
def get_session_factory(settings: Settings = Depends(get_settings)):
    engine = sa.create_engine(str(settings.postgres_url))
    return sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

- BaseSettings loads config from environment variables (e.g. POSTGRES_URL, API_KEY...)
- one single object/dependency for application config
  -> easy to override for testing

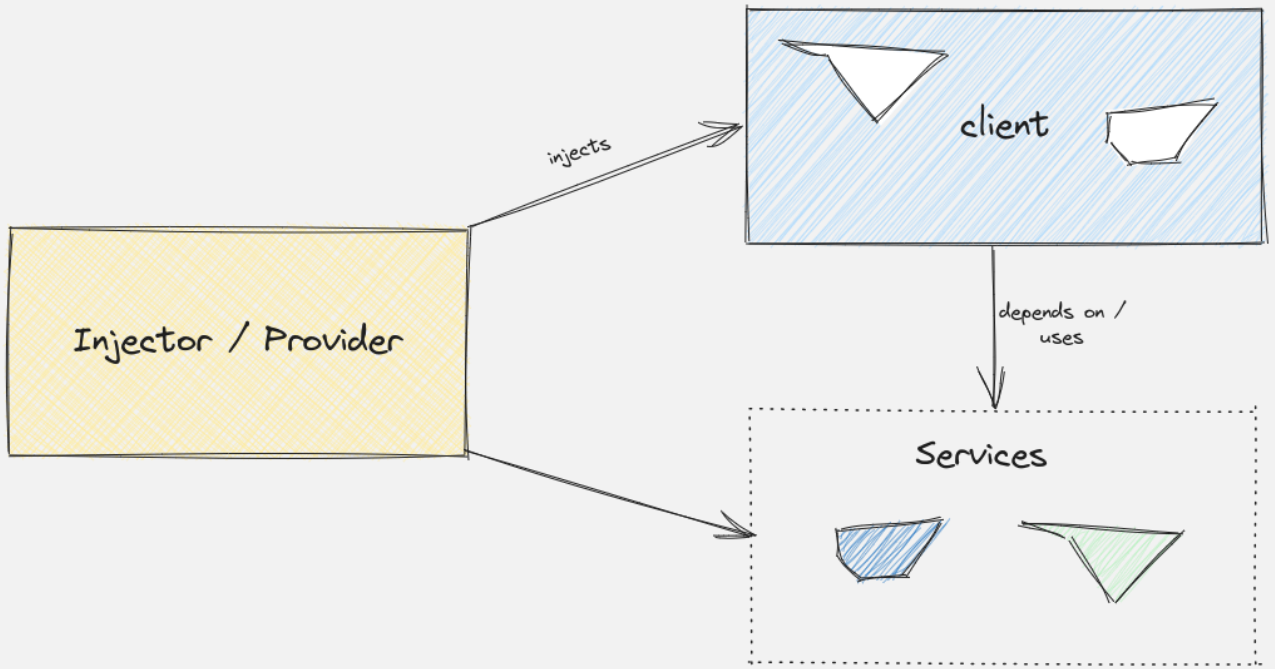
Overriding Dependencies

```python
import pytest
from fastapi.testclient import TestClient

from meetup_app import app, get_settings
from meetup.settings import Settings

def override_get_settings():
    return Settings(
        postgres_url="postgres://user:pass@localhost:5432/foobar",
        amqp_dsn="amqp://user:pass@localhost:5672/",
        api_key="my-test-key"
    )

@pytest.fixture
def client() -> TestClient:
    app.dependency_override[get_settings] = override_get_settings
    return TestClient(app)
```

- no need to set/patch environment variables
- every dependency can be overridden

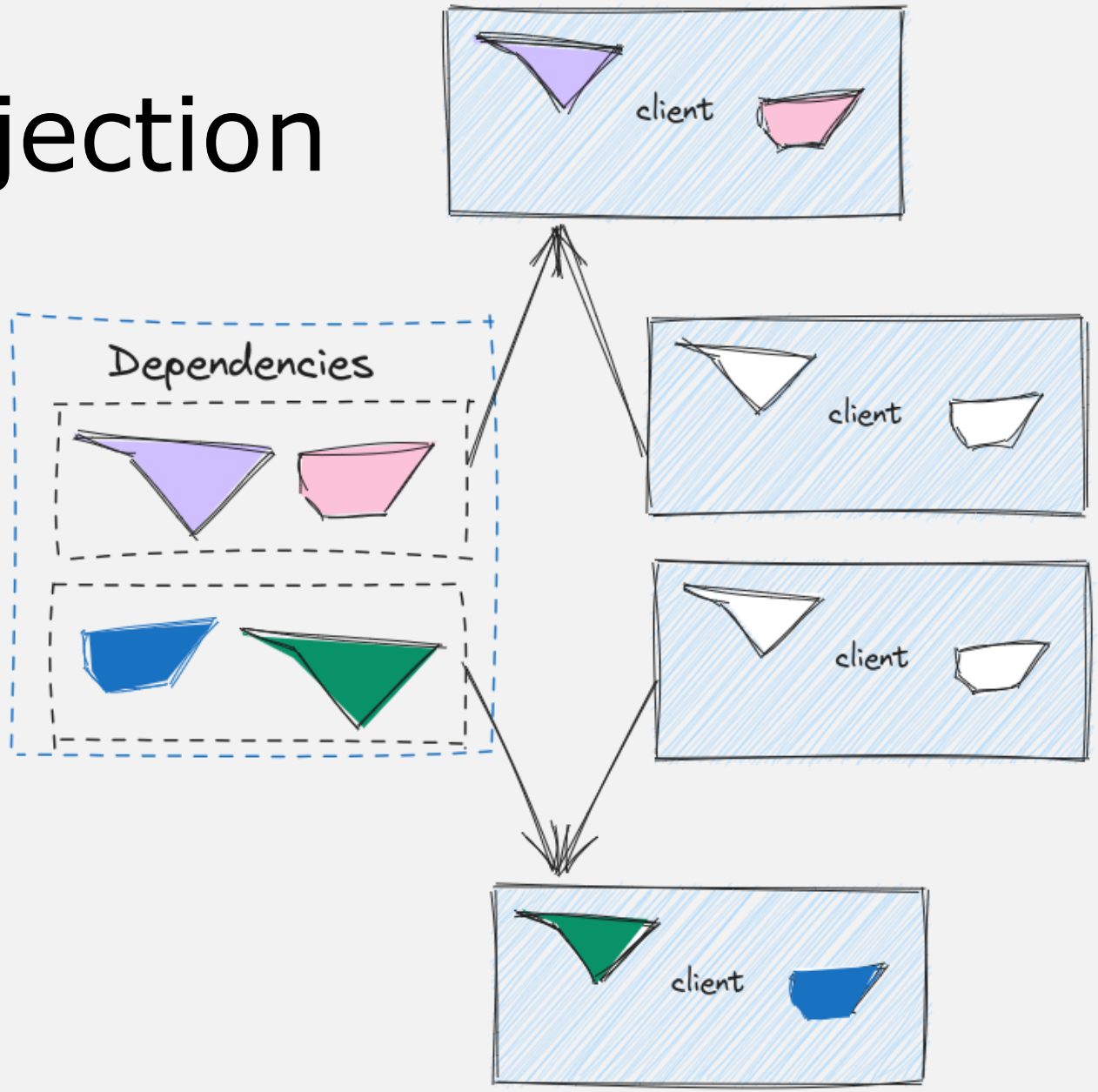
Dependency Injection

- **Client** – uses services (the dependencies)
- **Service** – implements functionality needed by clients
- **Interface** – API of services known to clients
- **Injector/Provider** – creates objects and injects dependencies



Dependency Injection

Clients don't care
about the implementation
(color) of the dependencies,
only the interface (shape).



Extended DI Example

```python
@functools.lru_cache
def get_session_factory():
    engine = sa.create_engine("postgresql://user:password@postgresserver/mydb")
    return sessionmaker(autocommit=False, autoflush=False, bind=engine)


def get_db_session(session_factory: sessionmaker = Depends(get_session_factory)):
    db_session = session_factory()
    try:
        yield db_session
    finally:
        db_session.close()


@app.get("/offers/{offer_id}", response_model=schemas.Offer)
def get_offer(offer_id: int, db_session: Session = Depends(get_db_session)) -> models.Offer | None:
    if (offer := db_session.query(models.Offer).get(offer_id)) is None:
        raise HTTPException(status_code=404, detail="Offer not found")
    return offer
```

Extended DI Example

View functions simple wrapper around business logic implemented in service

```python
@router.get("/")
def list_offers(offer_service: OfferService = Depends(get_offer_service)):
    return offer_service.list()

@router.get("/{id}")
def get_offer(
    id: int, offer_service: OfferService = Depends(get_offer_service)
):
    if (offer := offer_service.get(id)) is None:
        raise HTTPException(status_code=404, detail="Offer not found")
    return offer

@router.post("/")
def create_offer(
    offer: OfferCreate, offer_service: OfferService = Depends(get_offer_service)
):
    return offer_service.create(offer)
```

Extended DI Example

```python
class OfferService(abc.ABC):
    @abc.abstractmethod
    def get(self, id: int) -> Optional[Offer]:
        ...

    @abc.abstractmethod
    def list(self) -> List[Offer]:
        ...

    @abc.abstractmethod
    def create(self, offer: OfferCreate) -> Offer:
        ...
```

Extended DI Example

```python
class PostgresOfferService(OfferService):
    def __init__(self, db_session: Session):
        self._db_session = db_session

    def get(self, id: int) -> Optional[Offer]:
        offer = self._db_session.query(models.Offer).get(id)
        if offer:
            return Offer.model_validate(offer)
        return None

    def list(self) -> List[Offer]:
        return [Offer.model_validate(o) for o in self._db_session.query(models.Offer).all()]

    def create(self, offer: OfferCreate) -> Offer:
        offer = models.Offer(**offer.model_dump())
        self._db_session.add(offer)
        self._db_session.commit()
        return Offer.model_validate(offer)
```

Extended DI Example

```python
class RedisOfferService(OfferService):
    def __init__(self, client: redis.Redis):
        self._client = client

    def get(self, id: int) -> Optional[Offer]:
        offer = self._client.get(f"offer:{id}")
        if offer:
            return Offer.model_validate_json(offer)
        return None

    def list(self) -> List[Offer]:
        keys = self._client.keys("offer:*")
        return [Offer.model_validate_json(offer)
                for offer in [self._client.get(key) for key in keys]
                if offer is not None]

    def create(self, offer: OfferCreate) -> Offer:
        offer_id = self._client.incr("offer_id")
        offer = Offer(id=offer_id, **offer.model_dump())
        self._client.set(f"offer:{offer_id}", offer.model_dump_json())
        return offer

```

Extended DI Example

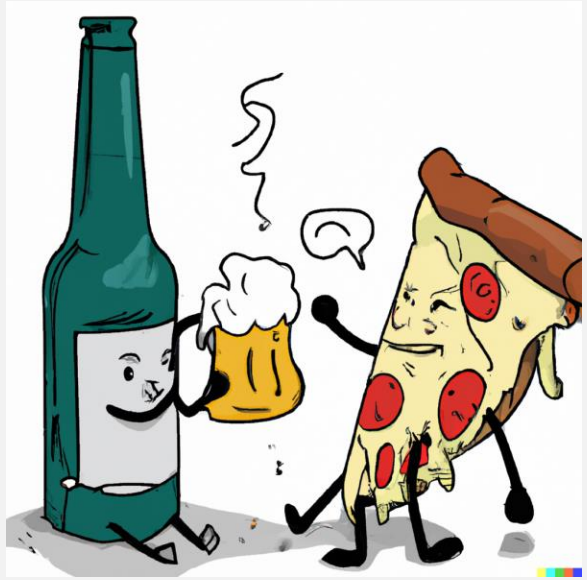
```python
def create_redis_offer_service(settings: Settings) -> RedisOfferService:
    if settings.redis_url is None:
        raise ValueError("Redis URL not configured")
    with redis.from_url(str(settings.redis_url)) as client:
        yield RedisOfferService(client)

def create_postgres_offer_service(settings: Settings) -> PostgresOfferService:
    session_factory = get_session_factory(settings)
    session = session_factory()
    try:
        yield PostgresOfferService(session)
    finally:
        session.close()

def get_offer_service(settings: Settings = Depends(get_settings)) -> OfferService:
    if settings.storage_backend == StorageBackendEnum.redis:
        yield from create_redis_offer_service(settings)
    elif settings.storage_backend == StorageBackendEnum.postgres:
        yield from create_postgres_offer_service(settings)
    else:
        raise NotImplementedError(f"Storage backend {settings.storage_backend} not implemented")
```

Extended DI Example

```python
class DummyOfferService:
    def __init__(self):
        self._offers = {
            1: Offer(id=1, title="Offer 1", description="Description 1", price="10.0"),
            2: Offer(id=2, title="Offer 2", description="Description 2", price="20.0"),
            3: Offer(id=3, title="Offer 3", description="Description 3", price="30.0"),
        }

    def get(self, id: int) -> Offer | None:
        return self._offers.get(id)

    ...

def client():
    from meetup_up.app import app

    app.dependency_overrides[get_offer_service] = lambda: DummyOfferService()
    with TestClient(app) as c:
        yield c

# test view functions
```

Recap

- DI helps to avoid global state
- DI makes testing easier
- Dependencies can be exchanged easily



Thanks

Example Code: https://github.com/treebee/di-example-fastapi

solute