Week 4: FastAPI + Database

CSI403 Full Stack Development | Lab 3 (8%)

Semester 1/2569

Agenda

1. Database Introduction
2. MSSQL with Docker
3. SQLAlchemy Setup
4. SQLAlchemy Models
5. CRUD with Database
6. Initialize Database
7. Lab 3 Assignment

Why Database?

Problem with In-Memory Storage:

- Data is lost when server restarts
- Cannot scale to multiple servers
- No data integrity or constraints
- Cannot handle complex queries

Solution: Use a Real Database

Microsoft SQL Server

Why MSSQL for this course?

- Enterprise-grade relational database
- Widely used in Thai companies
- Good integration with Python
- Free Developer edition available
- Easy to run with Docker

SQL vs ORM

Raw SQL:

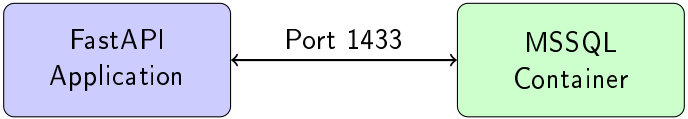
- Write SQL queries directly
- More control
- Database-specific
- Harder to maintain

ORM (SQLAlchemy):

- Python objects = Database tables
- Database-agnostic
- Easier to maintain
- Automatic migrations

We will use SQLAlchemy ORM

Docker for Database



Benefits:

- No installation required
- Same environment for everyone
- Easy to reset/recreate
- Isolated from host system

Run MSSQL Container

```
# Pull and run MSSQL Server
docker run -d \
  --name mssql-taskflow \
  -e "ACCEPT_EULA=Y" \
  -e "SA_PASSWORD=YourStrong@Pass123" \
  -p 1433:1433 \
  mcr.microsoft.com/mssql/server:2022-latest

# Check if container is running
docker ps

# View container logs
docker logs mssql-taskflow
```

Create Database

```
# Connect to MSSQL and create database
docker exec -it mssql-taskflow /opt/mssql-tools18/bin/sqlcmd \
  -S localhost -U sa -P "YourStrong@Pass123" \
  -C \
  -Q "CREATE␣DATABASE␣taskflow"

# Verify database was created
docker exec -it mssql-taskflow /opt/mssql-tools18/bin/sqlcmd \
  -S localhost -U sa -P "YourStrong@Pass123" \
  -C \
  -Q "SELECT␣name␣FROM␣sys.databases"
```

Database Connection String

Format:

```
mssql+pyodbc://user:password@host:port/database?driver=...
```

Example:

```
mssql+pyodbc://sa:YourStrong@Pass123@localhost:1433/taskflow?driver=ODBC+Driver+17+fo
```

What is SQLAlchemy?

Python SQL Toolkit and ORM

- Map Python classes to database tables
- Write queries in Python, not SQL
- Works with many databases
- Handles connections and transactions

Database Configuration

```python
# app/config.py
from pydantic_settings import BaseSettings

class Settings(BaseSettings):
    APP_NAME: str = "TaskFlow"

    DB_HOST: str = "localhost"
    DB_PORT: int = 1433
    DB_USER: str = "sa"
    DB_PASSWORD: str = "YourStrong@Pass123"
    DB_NAME: str = "taskflow"

    @property
    def DATABASE_URL(self) -> str:
        return (f"mssql+pyodbc://{self.DB_USER}:{self.DB_PASSWORD}"
                f"@{self.DB_HOST}:{self.DB_PORT}/{self.DB_NAME}"
                f"?driver=ODBC+Driver+17+for+SQL+Server")

settings = Settings()
```

Database Connection

```python
# app/database.py
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
from app.config import settings

# Create database engine
engine = create_engine(settings.DATABASE_URL)

# Create session factory
SessionLocal = sessionmaker(
    autocommit=False,
    autoflush=False,
    bind=engine
)

# Base class for models
Base = declarative_base()
```

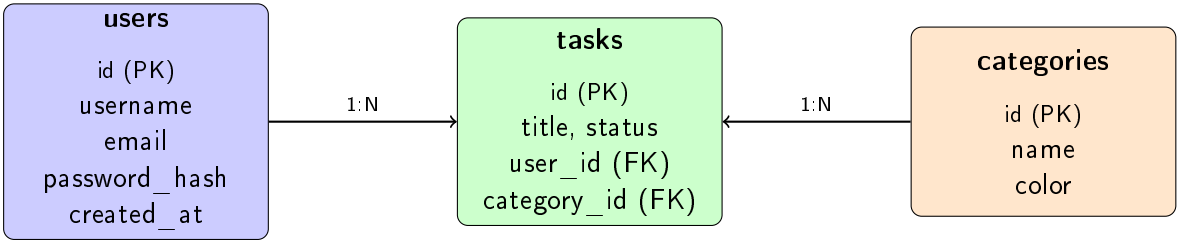
Dependency Injection

```python
# app/database.py (continued)

def get_db():
    """
    Dependency that provides database session.
    Automatically closes session after request.
    """
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

def create_tables():
    """Create all tables in database"""
    Base.metadata.create_all(bind=engine)
```

Database Schema Design



User Model

```python
# app/models/user.py
from sqlalchemy import Column, Integer, String, Boolean, DateTime
from sqlalchemy.orm import relationship
from datetime import datetime
from app.database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    username = Column(String(50), unique=True, nullable=False)
    email = Column(String(100), unique=True, nullable=False)
    password_hash = Column(String(255), nullable=False)
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime, default=datetime.utcnow)

    # Relationship
    tasks = relationship("Task", back_populates="owner")
```

Category Model

```python
# app/models/category.py
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import relationship
from app.database import Base

class Category(Base):
    __tablename__ = "categories"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(50), unique=True, nullable=False)
    color = Column(String(20), default="#6c757d")

    # Relationship
    tasks = relationship("Task", back_populates="category")
```

Task Model (Part 1)

```python
# app/models/task.py
from sqlalchemy import Column, Integer, String, Text, DateTime
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship
from datetime import datetime
from app.database import Base

class Task(Base):
    __tablename__ = "tasks"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String(200), nullable=False)
    description = Column(Text, nullable=True)
    status = Column(String(20), default="pending")
    priority = Column(String(20), default="medium")
```

Task Model (Part 2)

```
    # Foreign Keys
    user_id = Column(Integer, ForeignKey("users.id"))
    category_id = Column(Integer, ForeignKey("categories.id"),
                         nullable=True)

    # Timestamps
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, onupdate=datetime.utcnow)

    # Relationships
    owner = relationship("User", back_populates="tasks")
    category = relationship("Category", back_populates="tasks")
```

Models Init File

```python
# app/models/__init__.py
from app.models.user import User
from app.models.category import Category
from app.models.task import Task

# Export all models
__all__ = ["User", "Category", "Task"]
```

This allows importing models easily:

```python
from app.models import User, Task, Category
```

Create Task with Database

```python
from fastapi import Depends
from sqlalchemy.orm import Session
from app.database import get_db
from app.models import Task

@router.post("/", response_model=TaskResponse,
             status_code=status.HTTP_201_CREATED)
def create_task(
    task: TaskCreate,
    db: Session = Depends(get_db)
):
    db_task = Task(**task.model_dump(), user_id=1)
    db.add(db_task)
    db.commit()
    db.refresh(db_task)
    return db_task
```

List Tasks with Filters

```python
@router.get("/", response_model=list[TaskResponse])
def list_tasks(
    status: str | None = None,
    priority: str | None = None,
    db: Session = Depends(get_db)
):
    query = db.query(Task)

    if status:
        query = query.filter(Task.status == status)
    if priority:
        query = query.filter(Task.priority == priority)

    return query.order_by(Task.created_at.desc()).all()
```

Get Single Task

```python
@router.get("/{task_id}", response_model=TaskResponse)
def get_task(
    task_id: int,
    db: Session = Depends(get_db)
):
    task = db.query(Task).filter(Task.id == task_id).first()

    if not task:
        raise HTTPException(
            status_code=404,
            detail=f"Task {task_id} not found"
        )

    return task
```

Update Task

```python
@router.put("/{task_id}", response_model=TaskResponse)
def update_task(
    task_id: int,
    task_update: TaskUpdate,
    db: Session = Depends(get_db)
):
    task = db.query(Task).filter(Task.id == task_id).first()
    if not task:
        raise HTTPException(status_code=404, detail="Not found")

    for key, value in task_update.model_dump(exclude_unset=True).items():
        setattr(task, key, value)

    db.commit()
    db.refresh(task)
    return task
```

Delete Task

```python
@router.delete("/{task_id}",
               status_code=status.HTTP_204_NO_CONTENT)
def delete_task(
    task_id: int,
    db: Session = Depends(get_db)
):
    task = db.query(Task).filter(Task.id == task_id).first()

    if not task:
        raise HTTPException(status_code=404, detail="Not found")

    db.delete(task)
    db.commit()
```

Create Tables on Startup

```python
# app/main.py
from fastapi import FastAPI
from app.database import create_tables
from app.routes import tasks, categories, users

app = FastAPI(title="TaskFlow")

@app.on_event("startup")
def startup():
    create_tables()
    print("Database tables created!")

app.include_router(tasks.router)
app.include_router(categories.router)
app.include_router(users.router)
```

Lab 3: FastAPI + Database (8%)

Requirements:

1. Run MSSQL with Docker
2. Create database.py with connection
3. Create 3 SQLAlchemy models (User, Category, Task)
4. Setup relationships (1:N)
5. Update Task CRUD to use database
6. Create Category CRUD API
7. Create basic User registration

Lab 3 Grading Rubric

Criteria	Points
MSSQL Docker + database.py	1.5%
SQLAlchemy Models (3 models)	2%
Relationships (1:N)	1.5%
Task + Category CRUD with DB	2%
Code Quality	1%
Total	**8%**

Questions?

Connect your API to the database!

Next Week: Frontend Basics