Module 4: Database Integration with FastAPI

Overview

- Intro to databases and SQLAlchemy ORM
- Creating models and schemas
- Performing CRUD operations
- Alembic for database migrations

Intro to SQLAlchemy ORM

- SQLAlchemy is a popular ORM for Python
- `ORM` = Object-Relational Mapping
    - Maps Python classes to database tables
- Install:

```
pip install sqlalchemy
```

Creating Database Models

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Item(Base):
    __tablename__ = 'items'
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
```

Database Configuration

- Use SQLAlchemy's `create_engine` to connect to the database

```
from sqlalchemy import create_engine

SQLALCHEMY_DATABASE_URL = 'sqlite:///./test.db'
engine = create_engine(SQLALCHEMY_DATABASE_URL)
```

- Postgres example:

```
SQLALCHEMY_DATABASE_URL = f'postgresql://{DB_USER}:{DB_PASS}@localhost/fastapi_week4'
```

Session Local and Base

- Setup DB session:

```
from sqlalchemy.orm import sessionmaker

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

- Create the tables in the database (typically in `models.py` or `main.py`):

```
Base.metadata.create_all(bind=engine)
```

SQLAlchemy

- Data Types
- Relationship patterns
- ORM cascade
- Sessions

SQLAlchemy - Data Types

```
class Product(Base):
    __tablename__ = 'products'
    id=Column(Integer, primary_key=True)
    title=Column('title', String(32))
    in_stock=Column('in_stock', Boolean)
    quantity=Column('quantity', Integer)
    price=Column('price', Numeric)
```

SQLAlchemy - One-to-many relationship (ORM level)

```
class Article(Base):
    __tablename__ = 'articles'
    id = Column(Integer, primary_key=True)
    comments = relationship("Comment")


class Comment(Base):
    __tablename__ = 'comments'
    id = Column(Integer, primary_key=True)
    article_id = Column(Integer, ForeignKey('articles.id'))
```

- Unidirectional; Article --> Comment

SQLAlchemy - One-to-many relationship (DB level)

```sql
-- articles table
CREATE TABLE articles (
    id INTEGER PRIMARY KEY
);

-- comments table with foreign key to articles
CREATE TABLE comments (
    id INTEGER PRIMARY KEY,
    article_id INTEGER REFERENCES articles(id)
);
```

SQLAlchemy - Many-to-one relationship (ORM level)

```
class Article(Base):
    __tablename__ = 'articles'
    id = Column(Integer, primary_key=True)

class Comment(Base):
    __tablename__ = 'comments'
    id = Column(Integer, primary_key=True)
    article_id = Column(Integer, ForeignKey('articles.id'))
    article = relationship(Article)
```

- Unidirectional; Comment --> Article

SQLAlchemy - Many-to-one relationship (DB level)

- there is no difference in the DB schema!!!

```sql
-- articles table
CREATE TABLE articles (
    id INTEGER PRIMARY KEY
);

-- comments table with foreign key to articles
CREATE TABLE comments (
    id INTEGER PRIMARY KEY,
    article_id INTEGER REFERENCES articles(id)
);
```

SQLAlchemy - One-to-one relationship (ORM level)

```
class Person(Base):
    __tablename__ = 'people'
    id = Column(Integer, primary_key=True)
    mobile_phone = relationship("MobilePhone",
                        uselist=False,
                        back_populates="person")

class MobilePhone(Base):
    __tablename__ = 'mobile_phones'
    id = Column(Integer, primary_key=True)
    person_id = Column(Integer, ForeignKey('people.id'))
    person = relationship("Person",
                          back_populates="mobile_phone")
```

SQLAlchemy - One-to-one relationship (DB level)

```sql
-- people table
CREATE TABLE people (
    id INTEGER PRIMARY KEY
);

-- mobile_phones table with a one-to-one relationship to people
CREATE TABLE mobile_phones (
    id INTEGER PRIMARY KEY,
    person_id INTEGER UNIQUE REFERENCES people(id)
);
```

SQLAlchemy - Many-to-many relationship (ORM level)

SQLAlchemy - Many-to-many relationship (DB level)

CRUD Operations with SQLAlchemy

- Typical functions:
  - Create: add new record
  - Read: fetch by ID or all
  - Update: modify record
  - Delete: remove record

CRUD Example: Create Item

```
@app.post("/items/")
def create_item(item: ItemCreate,
  db: Session = Depends(get_db)):
    db_item = models.Item(**item.dict())
    db.add(db_item)
    db.commit()
    db.refresh(db_item)
    return db_item
```

Alembic for Migrations

- Alembic is a lightweight database migration tool for SQLAlchemy
- Install:

```
pip install alembic
```

- Initialize Alembic:

```
alembic init alembic
```

Live Coding Example

- Connecting to Postgres
- Creating a model and schema
- Writing create/read routes
- Applying migrations with Alembic

Homework

- Create models for User and Post
- Implement full CRUD for both
- Apply Alembic migrations
- Push code to GitHub

Remember

- SQLAlchemy ORM basics
- Creating models and schemas
- Performing CRUD operations
- Using Alembic for migrations
- Dependency injection with FastAPI