**FastApi Core Models**

*#/Core/models*

***\_\_all\_\_ = (***

***"Base",***

***)***

***from .base import Base***

***import datetime***

***from typing import Annotated***

***from sqlalchemy import (***

***String, text,***

***)***

***from sqlalchemy.orm import DeclarativeBase, Mapped, declared\_attr, mapped\_column***

***created\_at = Annotated[datetime.datetime, mapped\_column(server\_default=text("TIMEZONE('utc', now())"))]***

***updated\_at = Annotated[datetime.datetime, mapped\_column(server\_default=text("TIMEZONE('utc', now())"))]***

***str\_30 = Annotated[str, mapped\_column(String(30))]***

***str\_300 = Annotated[str, mapped\_column(String(300))]***

***datetime\_column = Annotated[datetime.datetime, mapped\_column(DateTime)]***

***class Base(DeclarativeBase):***

***\_\_abstract\_\_ = True***

***@declared\_attr.directive***

***def \_\_tablename\_\_(cls) -> str:***

***return f"{cls.\_\_name\_\_.lower()}"***

***id: Mapped[int] = mapped\_column(primary\_key=True)***

**FastApi Models**

***\_\_all\_\_ = (  
 "User",  
)  
  
from .user\_model import (  
 User  
)***

***from uuid import UUID  
from uuid import uuid4***

***class User(Base):***

***first\_name: Mapped[str\_30] = mapped\_column(nullable=True)  
last\_name: Mapped[str\_30] = mapped\_column(nullable=True)  
role: Mapped[UserRole] = mapped\_column(nullable=True)  
created\_at: Mapped[created\_at]***

***updated\_at: Mapped[updated\_at]  
booking: Mapped[list["Booking"]] = relationship("Booking", back\_populates="user")***

***class Booking(Base):***

***confirmation\_id: Mapped[UUID] = mapped\_column(primary\_key=True, default=uuid4)***

***description: Mapped[str\_300] = mapped\_column(nullable=True)  
created\_at: Mapped[created\_at]  
payment\_status: Mapped[PaymentStatus] = mapped\_column(nullable=True)  
payment\_time: Mapped[time\_column] = mapped\_column(nullable=True)  
  
user\_id = mapped\_column(ForeignKey("user.id", ondelete="CASCADE"))  
user: Mapped["User"] = relationship("User", back\_populates="booking")***

**FastApi alembic and .env**

***pip install python-dotenv***

***pip install psycopg2-binary***

***DB\_HOST = localhost  
DB\_PORT = 5432  
DB\_USER = denis  
DB\_NAME = booking\_service\_db  
DB\_PASS = 1234***

*# setting.py*

***import os  
  
from dotenv import load\_dotenv  
load\_dotenv(".env")  
  
DB\_HOST = os.environ.get("DB\_HOST")  
DB\_PORT = os.environ.get("DB\_PORT")  
DB\_USER = os.environ.get("DB\_USER")  
DB\_NAME = os.environ.get("DB\_NAME")  
DB\_PASS = os.environ.get("DB\_PASS")***

***SYNC\_URL= f'postgresql+psycopg2://{DB\_USER}:{DB\_PASS}@{DB\_HOST}:{DB\_PORT }/{ DB\_NAME}'***

***ECHO = True***

***AUTOCOMMIT = False***

***AUTOFLUSH =False***

*#alembic.ini*

***sqlalchemy.url = postgresql://%(DB\_USER)s:%(DB\_PASS)s@%(DB\_HOST)s:%(DB\_PORT)s/%(DB\_NAME)s***

*#env.py*

***config = context.config***  
  
***section = config.config\_ini\_section  
config.set\_section\_option(section, "DB\_HOST", settings.DB\_HOST)  
config.set\_section\_option(section, "DB\_PORT", settings.DB\_PORT)  
config.set\_section\_option(section, "DB\_USER", settings.DB\_USER)  
config.set\_section\_option(section, "DB\_NAME", settings.DB\_NAME)  
config.set\_section\_option(section, "DB\_PASS", settings.DB\_PASS)***

***from src.core.model.base import Base  
target\_metadata = Base.metadata***

*#import other models*

*command*

*# alembic init alembic*

*# alembic revision –autogenerate –m “your message”*

*# alembic upgrade head*

*# alembic downgrade -1*

**Database engine**

*#database.py*

***from sqlalchemy import MetaData  
from sqlalchemy import create\_engine  
  
from sqlalchemy.orm import sessionmaker  
  
from src.config import settings  
from src.core.model import Base  
  
metadata = MetaData()  
  
sync\_engine = create\_engine(settings.SYNC\_URL, echo=settings. ECHO)  
  
SessionLocal = sessionmaker(autocommit=settings.AUTOCOMMIT, autoflush=settings.AUTOFLUSH, bind=sync\_engine)***

***def get\_session():  
 session = SessionLocal()  
 try:  
 yield session  
 finally:  
 session.close()***

**Schemas**

***from datetime import datetime   
from typing import Optional  
from uuid import UUID  
  
from pydantic import (  
 BaseModel,  
 ConfigDict,  
 Field,  
 field\_validator,  
)***  
  
***class UserBaseSchema(BaseModel):  
 first\_name: str  
 last\_name: str  
 role: UserRole***

***created\_at: Optional[datetime] = None***

***updated\_at: Optional[datetime] = None***  
 ***class UserCreateSchema(UserBaseSchema):***

***pass***

***class UserUpdatePartialSchema(BaseModel):***  
 ***first\_name: str | None = None  
 last\_name: str | None = None*  
 *role: UserRole | None = None***  
 ***class UserUpdateSchema(BaseModel):  
 first\_name: str   
 last\_name: str   
 role: UserRole***

***class UserCreateRetrieveSchema(UserBaseSchema):  
 model\_config = ConfigDict(from\_attributes=True)  
 id: int***

**Routers**

***from fastapi import APIRouter, status, Depends  
  
from ..models import User  
from ..schemas.user\_schema import (  
 UserSchema,  
 UserCreateSchema,  
 UserUpdateSchema,  
 UserUpdatePartialSchema,   
)  
  
from ..services.user\_service import (  
 get\_users,  
 create\_user,  
 update\_user,  
 delete\_user,   
)  
from ..utils.user\_utils import get\_user\_info\_by\_id  
  
  
@router.post("", response\_model=UserCreateSchema, status\_code=status.HTTP\_201\_CREATED)  
def create\_user\_handler(  
 user\_schema: UserCreateSchema,  
) -> User:  
 user = create\_user(  
 user\_schema =user\_schema,  
 )  
 return user  
  
  
@router.get("", response\_model= list[UserSchema])  
def get\_users\_handler():  
 data = get\_users()  
 return data  
  
  
@router.get("/{user\_id}", response\_model=UserSchema)  
def get\_user\_handler(  
 user=Depends(get\_user\_info\_by\_id),  
) -> User:  
 return user  
  
  
@router.put("/{user\_id}", response\_model=UserUpdateSchema)  
def update\_user\_handler(  
 user\_object: UserUpdateSchema,  
) -> User:  
 return update\_user(  
 user\_schema=user\_schema,  
 )  
  
@router.delete("/{user\_id}", status\_code=status.HTTP\_204\_NO\_CONTENT)  
async def delete\_user\_handler(  
 user=Depends(get\_user\_info\_by\_id),  
) -> None:  
 delete\_user(  
  
 user=user  
 )***

**Servicies and Utils**

***from uuid import UUID  
  
from fastapi import HTTPException  
from sqlalchemy import select, Result  
from ..models import User  
from ..schemas.user\_schema import (  
 UserUpdateSchema,  
 UserCreateSchema,  
 UserUpdatePartialSchema,  
)  
  
  
def create\_user(  
 session: AsyncSession,  
 user\_schema: UserCreateSchema,  
) -> User:  
   
 user = User(\*\*user\_schema.model\_dump())  
 session.add(user)  
 session.commit()  
  
 return user  
  
  
def get\_users(  
) -> list[User]:  
  
 query = select(User)  
 result: session.execute(query)  
 data = result.scalars().all()  
  
 return list(data)  
  
def get\_user\_by\_id(user\_id: int) -> User | None:  
 return await session.get(User, user\_id)***

***def update\_user(  
 session: AsyncSession,  
 user: User,  
 user\_schema: UserUpdateSchema   
) -> User:  
 for name, value in user\_schema.model\_dump().items():  
 setattr(user, name, value)  
 session.commit()  
 return user  
async def delete\_user(  
 session: AsyncSession,  
 user: User,  
) -> None:  
 await session.delete(user)  
 await session.commit()***